

Computational model validation using a novel multiscale multidimensional spatio-temporal meta model checking approach

A thesis submitted
for the degree of Doctor of Philosophy

by
Ovidiu Pârvu

Department of Computer Science
Brunel University London
August 2015

BLANK

Computational model validation using a novel multiscale multidimensional spatio-temporal meta model checking approach

Ovidiu Pârvu

Department of Computer Science
Brunel University London

Abstract

Computational models of complex biological systems can provide a better understanding of how living systems function but need to be validated before they are employed for real-life (e.g. clinical) applications.

One of the most frequently employed *in silico* approaches for validating such models is model checking. Traditional model checking approaches are limited to uniscale non-spatial computational models because they do not explicitly distinguish between different scales, and do not take properties of (emergent) spatial structures (e.g. density of multicellular population) into account.

This thesis defines a novel multiscale multidimensional spatio-temporal meta model checking methodology which enables validating multiscale (spatial) computational models of biological systems relative to how both numeric (e.g. concentrations) and spatial system properties are expected to change over time and across multiple scales. The methodology has two important advantages. First it supports computational models encoded using various high-level modelling formalisms because it is defined relative to time series data and not the models used to produce them. Secondly the methodology is generic because it can be automatically reconfigured according to case study specific types of spatial structures and properties using the meta model checking approach. In addition the methodology could be employed for multiple domains of science, but we illustrate its applicability here only against biological case studies. To automate the computational model validation process, the approach was implemented in software tools, which are made freely available online. Their efficacy is illustrated against two uniscale and four multiscale quantitative computational models encoding phase variation in bacterial colonies and the chemotactic aggregation of cells, respectively the rat cardiovascular system dynamics, the uterine contractions of labour, the *Xenopus laevis* cell cycle and the acute inflammation of the gut and lung.

This novel model checking approach will enable the efficient construction of reliable multiscale computational models of complex systems.

The predictive power of a model is dependent on the rigour of its validation.
(Walpole et al., 2013)

Contents

Contents	v
List of Figures	x
List of Tables	xii
List of Abbreviations	xiv
Acknowledgements	xvi
Author's declaration	xviii
1 Introduction	1
1.1 Systems biology	1
1.2 Computational models in systems biology	2
1.2.1 Development	3
1.2.2 Types of models	4
1.2.3 Standards	4
1.2.4 Validation	5
1.3 Motivation	5
1.4 Contributions	7
1.4.1 Description	8
1.4.2 Publications	9
1.5 Structure	10
2 Model checking	13
2.1 Preliminaries	13
2.1.1 Formal verification methods	13
2.1.2 Model checking background	14
2.2 Model construction	15
2.2.1 Labelled state transition systems	16
2.2.2 Probabilistic labelled state transition systems	19
2.3 Formal specification	21
2.3.1 Linear time temporal logics	21
2.3.1.1 Linear Temporal Logic	22
2.3.1.2 Bounded Linear Temporal Logic	24
2.3.1.3 Probabilistic Linear Temporal Logic	25
2.3.2 Branching time temporal logics	26

2.3.2.1	Computation Tree Logic	26
2.3.2.2	Extended Computation Tree Logic	29
2.3.2.3	Bounded and probabilistic branching time logics	32
2.4	Model verification	32
2.4.1	Model checking labelled state transition systems	33
2.4.1.1	LTL model checking	33
2.4.1.2	CTL model checking	33
2.4.1.3	CTL* model checking	34
2.4.1.4	State space explosion problem	34
2.4.2	Model checking probabilistic labelled state transition systems	37
2.4.2.1	Computing probabilities over computation paths	37
2.4.2.2	Exhaustive probabilistic model checking	40
2.4.2.3	Approximate probabilistic model checking	40
2.4.2.4	Comparing probabilistic model checking approaches	42
2.5	Model checking computational models of biological systems	43
2.5.1	Computational modelling formalisms	44
2.5.2	Formal specification	45
2.5.3	Computational model checking approaches	45
2.5.4	Limitations	46
3	Multidimensional spatio-temporal model checking	48
3.1	Spatial computational models of biological systems	48
3.2	Multidimensional spatio-temporal model checking workflow	50
3.3	Model construction	51
3.3.1	Explicitly encoding space	52
3.3.2	Stochastic spatial discrete-event systems	54
3.4	Spatio-temporal detection and analysis	58
3.4.1	Spatial entity types	58
3.4.1.1	Regions	58
3.4.1.2	Clusters	60
3.4.2	Spatial measures	61
3.4.2.1	Computing spatial measures values for regions	62
3.4.2.2	Computing spatial measures values for clusters	64
3.4.3	Spatial Temporal Markup Language	65
3.5	Formal specification	67
3.5.1	Bounded Linear Spatial Temporal Logic	67
3.5.1.1	Syntax	69
3.5.1.2	Semantics	73
3.5.1.3	Illustrative examples of BLSTL statements	77
3.5.2	Probabilistic Bounded Linear Spatial Temporal Logic	79
3.6	Model checking	79
3.6.1	Proof that the multidimensional model checking problem is well-defined	81
3.6.1.1	Finite number of required simulations	81
3.6.1.2	Finite number of state transitions	83

3.6.1.3	Well-defined model checking problem	85
3.7	Implementation	86
3.7.1	Spatio-temporal detection and analysis modules	88
3.7.2	Model checker Mudi	88
3.7.3	Availability	92
3.8	Related work	92
3.8.1	Epidemiology	92
3.8.2	Spatial information theory	93
4	Validation of multidimensional computational models of biological systems	97
4.1	Description	97
4.2	Phase variation patterning in bacterial colony growth	98
4.2.1	Model construction	99
4.2.2	Spatio-temporal analysis	99
4.2.3	Formal specification	99
4.2.4	Model checking	102
4.3	Chemotactic aggregation of cells	104
4.3.1	Model construction	105
4.3.2	Spatio-temporal analysis	105
4.3.3	Formal specification	106
4.3.4	Model checking	109
4.4	Discussion	110
4.4.1	Supported modelling formalisms	110
4.4.2	Spatio-temporal analysis based on image processing	111
4.4.3	STML files generated on demand	111
4.4.4	Supported model checking algorithms	111
4.4.5	Scalability	112
4.4.6	Limitations	112
5	Multiscale multidimensional spatio-temporal meta model checking	114
5.1	Multiscale computational models of biological systems	114
5.2	Multiscale multidimensional spatio-temporal model checking workflow	120
5.3	Model construction	122
5.3.1	Encoding the hierarchical system structure	123
5.3.2	Multiscale stochastic spatial discrete-event systems	127
5.4	Multiscale spatio-temporal analysis	132
5.4.1	Detection and analysis of spatial entities from multiple scales	132
5.4.2	Multiscale Spatial Temporal Markup Language	133
5.5	Formal specification	136
5.5.1	Bounded Linear Multiscale Spatial Temporal Logic	136
5.5.1.1	Syntax	138
5.5.1.2	Semantics	143
5.5.1.3	Illustrative examples of BLMSTL statements	150

5.5.2	Probabilistic Bounded Linear Multiscale Spatial Temporal Logic	151
5.6	Model checking	152
5.7	Meta model checking	152
5.8	Implementation	155
5.8.1	Multiscale spatio-temporal detection and analysis module	156
5.8.2	Model checker Mule	156
5.8.3	Availability	158
5.9	Related work	159
5.9.1	Pattern recognition	159
5.9.2	Spatial information theory	160
6	Validation of multiscale computational models of biological systems	162
6.1	Description	162
6.2	Model construction	165
6.2.1	Rat cardiovascular system dynamics	165
6.2.2	Uterine contractions of labour	166
6.2.3	<i>Xenopus laevis</i> cell cycle	167
6.2.4	Acute inflammation of the gut and lung	167
6.3	Multiscale spatio-temporal analysis	168
6.4	Formal specification	169
6.4.1	Rat cardiovascular system dynamics	169
6.4.2	Uterine contractions of labour	171
6.4.3	<i>Xenopus laevis</i> cell cycle	173
6.4.4	Acute inflammation of the gut and lung	176
6.5	Model checking	178
6.6	Discussion	181
6.6.1	Model validation and experimental data analysis	182
6.6.2	Automatic reconfiguration according to case study specific spatial entity types and measures	182
6.6.3	Scalability	183
7	Conclusions, open problems and future work	185
7.1	Summary and conclusions	185
7.1.1	Multidimensional spatio-temporal model checking	186
7.1.2	Multiscale multidimensional spatio-temporal meta model checking	188
7.2	Open problems and future work	191
7.2.1	Analysis of time series data recorded in the <i>in vitro</i> environment	191
7.2.2	Validation of computational models from other domains of science	191
7.2.3	Parameter estimation, model construction and robustness computation	192
7.2.4	Distributed multiscale model checking web service	192

7.2.5	Alternative model representations and spatio-temporal analysis modules	193
7.2.6	Usability improvement	193
Appendix A	Approximate probabilistic model checking approaches	194
A.1	Chernoff-Hoeffding bounds based model checking	194
A.2	Frequentist statistical model checking	195
	A.2.1 Single acceptance sampling plan	196
	A.2.2 Sequential acceptance sampling plan	197
A.3	Statistical black-box model checking	198
A.4	Bayesian mean and variance estimate based model checking . .	198
A.5	Bayesian statistical model checking	199
Appendix B	Existing model checking approaches for computational models of biological systems	201
Appendix C	Multidimensional spatio-temporal model checking supplementary materials	209
C.1	Mapping between subalgorithms of region detection mechanism and OpenCV functions	209
C.2	Numeric measures for encoding formal specifications	210
C.3	Subset measures for encoding formal specifications	211
C.4	Improved frequentist statistical model checking	213
	C.4.1 Notations	214
	C.4.2 Description of initialisation error	214
	C.4.3 Solution	215
C.5	Proof that the semantics of a BLSTL statement can be defined based on a finite prefix of an infinite execution	217
Appendix D	Multiscale multidimensional spatio-temporal meta model checking supplementary materials	224
D.1	Proof that the multiscale multidimensional spatio-temporal model checking problem is well-defined	224
	D.1.1 Finite number of required simulations	224
	D.1.2 Finite number of state transitions	225
	D.1.3 Well-defined model checking problem	233
References		234

List of Figures

1.1	Thesis structure	11
2.1	Description of the model checking process	15
2.2	<i>Dictyostelium discoideum</i> labelled state transition system	18
2.3	<i>Dictyostelium discoideum</i> probabilistic labelled state transition system	20
2.4	Linear time structure example	21
2.5	Branching time structure example	27
2.6	Well-known approaches for tackling the state space explosion problem	35
3.1	Workflow for constructing and validating multidimensional spatio-temporal computational models of biological systems	52
3.2	Initial state of SSpDES encoding the growth of a population of cells in a fixed size environment	55
3.3	State space of SSpDES encoding the growth of a population of cells in a fixed size environment	56
3.4	Detection of sector-like patterns in bacterial colonies	59
3.5	Detection of clusters in multicellular populations	62
3.6	Visual description of spatial measures computed for regions and clusters	63
3.7	Computational model validation relative to how both numeric and spatial properties are expected to change over time using the model checker Mudi and the spatio-temporal detection and analysis modules	87
3.8	Multidimensional spatio-temporal model checking use case diagram	89
3.9	Architecture of the multidimensional spatio-temporal model checker Mudi	90
3.10	Class diagram corresponding to the model checking approaches considered, and the PBLSTL logic property parser and evaluator	91
3.11	A graphical description of the topological spatial relations in \mathcal{RCC} -8	94
4.1	Spatio-temporal detection and analysis for a phase variation model simulation	100
4.2	Spatio-temporal detection and analysis for a chemotaxis model simulation	106
5.1	Multiscale multidimensional spatio-temporal model checking workflow	122
5.2	Illustrative example on how to construct a multiscale architecture graph	125

5.3	Initial state of the MSSpDES encoding the movement of a unicellular organism	128
5.4	The state space of the MSSpDES encoding the movement of a unicellular organism	129
5.5	The multiscale architecture graph corresponding to the MSSpDES encoding the movement of a unicellular organism	131
5.6	The multiscale spatio-temporal analysis workflow	134
5.7	Workflow for creating multiscale multidimensional spatio-temporal model checking methodology instances	155
5.8	Implementation of workflow for generating multiscale multidimensional model checker instances	157
6.1	<i>MA</i> graph representing the multiscale organization of the rat cardiovascular system dynamics computational model	166
6.2	<i>MA</i> graph representing the multiscale organization of the uterine contractions of labour computational model	166
6.3	<i>MA</i> graph representing the multiscale organization of the <i>Xenopus laevis</i> cell cycle computational model	167
6.4	<i>MA</i> graph representing the multiscale organization of the acute inflammation of the gut and lung computational model	168
6.5	Average execution times (measured in seconds) corresponding to the validation of the rat cardiovascular system dynamics, the uterine contractions of labour, the <i>Xenopus laevis</i> cell cycle and the acute inflammation of the gut and lung computational models	181

List of Tables

2.1	Considered approximate probabilistic model checking approaches	42
3.1	Translation between full and abbreviated BLSTL symbol names	73
3.2	Interval Algebra relations	95
4.1	Model checking statistical analysis results for the phase variation case study	103
4.2	Model checking statistical analysis results for the chemotaxis case study	109
5.1	Problem independent multiscale modelling approaches for computational models of biological systems	117
5.2	Translation of full BLMSTL symbol names to abbreviated forms	144
6.1	Considered multiscale systems biology computational models against which the multiscale multidimensional spatio-temporal meta model checking methodology and implementation were validated	163
6.2	Model simulation and analysis execution times for the rat cardiovascular system dynamics, the uterine contractions of labour, the <i>Xenopus laevis</i> cell cycle and the acute inflammation of the gut and lung case studies	169
6.3	Model checking statistical analysis results for the rat cardiovascular system dynamics, the uterine contractions of labour, the <i>Xenopus laevis</i> cell cycle and the acute inflammation of the gut and lung case studies	179
6.4	Comparison of average model checker execution times when PBLM-STL statements corresponding to a computational model are stored in a single, respectively multiple separate files	180
B.1	Existing model checking approaches employed for validating computational models of biological systems	201
C.1	Mapping between subalgorithms employed by Algorithm 1 and functions from the Open source Computer Vision library OpenCV	209
C.2	Description of unary numeric measures which can be employed for writing formal specifications	210
C.3	Description of binary numeric measures which can be employed for writing formal specifications	211

C.4	Description of unary subset measures which can be employed for writing formal specifications	211
C.5	Description of binary subset measures which can be employed for writing formal specifications	211
C.6	Description of ternary subset measures which can be employed for writing formal specifications	213
C.7	Description of quaternary subset measures which can be employed for writing formal specifications	213
C.8	Valid intervals of δ values for improved frequentist statistical model checking	216

List of Abbreviations

ABM	Agent Based Model
AP	Approximate Probabilistic
APC	Anaphase-Promoting Complex
apLTL	abstraction-preserved Linear Temporal Logic
ARCTL	Action Restricted Computation Tree Logic
ATL	Alternating-time Temporal Logic
BDD	Binary Decision Diagram
BLMSTL	Bounded Linear Multiscale Spatial Temporal Logic
BLSTL	Bounded Linear Spatial Temporal Logic
BLTL	Bounded Linear Temporal Logic
BNF	Backus-Naur Form
BOSL	Biological Oscillators Synchronisation Logic
BSTL	Bounded Spatio-Temporal Logic
CA	Cellular Automata
cAMP	cyclic Adenosine Monophosphate
CDK1	Cyclin-Dependent Kinase 1
CellML	Cell Markup Language
CPM	Cellular Potts Model
CSL	Continuous-time Stochastic Logic
CSMMR	Coloured Stochastic Multilevel Multiset Rewriting
CTL	Computation Tree Logic
CTL*	Extended Computation Tree Logic
DBSCAN	Density Based Spatial Clustering of Applications with Noise
DDE	Delay Differential Equations
DNA	Deoxyribonucleic Acid
DSD	Deoxyribonucleic Acid Strand Displacement
EGF	Epidermal Growth Factor
EGFR	Epidermal Growth Factor Receptor
EN	Exhaustive Non-probabilistic
EP	Exhaustive Probabilistic
ERK	Extracellular Signal-Regulated Kinase
FGF	Fibroblast Growth Factor
FieldML	Field Markup Language
gp130/JAK/STAT	glycoprotein 130/Janus Kinase/Signal Transducer and Activator of Transcription
GUI	Graphical User Interface
HASL	Hybrid Automata Stochastic Logic
HMGB1	High-Mobility Group Box 1
IA	Interval Algebra
LSTS	Labelled State Transition System
LTL	Linear Temporal Logic
LTLc	Linear Temporal Logic with constraints

<i>MA</i>	Multiscale Architecture
MAPK	Mitogen-Activated Protein Kinase
MC	Model Construction
MITL	Metric Interval Temporal Logic
MLC	Myosin Light Chain
MSSpDES	Multiscale Stochastic Spatial Discrete-Event System
MSTML	Multiscale Spatial Temporal Markup Language
MV	Model Validation
NGF	Nerve Growth Factor
NME	Not Mentioned Explicitly
ODE	Ordinary Differential Equation
PBL	Probabilistic Bounded Linear Temporal Logic
PBLMSTL	Probabilistic Bounded Linear Multiscale Spatial Temporal Logic
PBLSTL	Probabilistic Bounded Linear Spatial Temporal Logic
PBLTL	Probabilistic Bounded Linear Temporal Logic
PBMTL	Probabilistic Bounded Metric Temporal Logic
PCTL	Probabilistic Computation Tree Logic
PDE	Partial Differential Equation
PEPA	Performance Evaluation Process Algebra
PHML	Physiological Hierarchy Markup Language
PI	Parameter Identification
PLSTS	Probabilistic Labelled State Transition System
PLTL	Probabilistic Linear Temporal Logic
PLTLC	Probabilistic Linear Temporal Logic with constraints
QFCTL	Quantifier Free Computation Tree Logic
QFLTL	Quantifier Free Linear Temporal Logic
RA	Rectangle Algebra
RC	Robustness Computation
<i>RCC</i>	Region Connection Calculus
RKIP	Raf Kinase Inhibitor Protein
SDES	Stochastic Discrete-Event System
SSpDES	Stochastic Spatial Discrete-Event System
SSTL	Signal Spatio-Temporal Logic
STL	Signal Temporal Logic
STML	Spatial Temporal Markup Language
TCTL	Timed Computation Tree Logic
TJ	Tight Junction
UML	Unified Modelling Language

Acknowledgements

I am grateful to everyone who has supported me throughout my doctoral studies.

First of all I would like to thank my principal supervisor, Professor David Gilbert, for introducing me to the interdisciplinary area of systems biology, for all the provided guidance and support, and for allowing me to freely explore my own ideas. I would also like to thank you for giving me the opportunity to interact and collaborate with many people outside the department by financially supporting my attendance to multiple scientific seminars, conferences and summer schools throughout the years. In addition I am grateful that you always made time to discuss with me about my project, in spite of your very busy schedule.

Secondly I would like to thank my second supervisor, Professor Nigel Saunders, for all the provided help on the biologically relevant aspects of my work, and for the insightful contributions and comments on the manuscripts related to phase variation in bacterial colony growth. Moreover I would like to thank you and the members of your team, namely Piyali Basu, Dr. Antima Gupta, Arshad Khan and Dr. Carlos Pires, for making me feel as part of the group when I moved from St John's to the Heinz Wolff building.

I would also like to thank all the academic members of staff in the Department of Computer Science who supported me, patiently read early drafts of my work, and provided insightful comments which helped me improve: Dr. Julie Eatock, Dr. Crina Groşan, Professor Xiaohui Liu, Dr. Alessandro Pandini, Professor Martin Shepperd, Dr. Larisa Soldatova and Dr. Allan Tucker. I would like to especially thank Dr. Crina Groşan for giving me the opportunity to get involved in research projects ever since I was an undergraduate student, for all the provided support since, and for encouraging me to apply for the current PhD position straight after my undergraduate studies. Without your help and advice I would not have been here today.

In addition I would like to thank the collaborators with whom I have been fortunate to work, namely Professor Monika Heiner and current/former members of her team (Christian Rohr, Dr. Mostafa Herajy and Dr. Fei Liu), Professor Wolfgang Marwan, Mary Ann Blätke and Dr. Simon Shaw, for introducing me to the concept of modelling using Petri nets, for involving me in their ongoing projects on the phase variation in bacterial colony growth and the *Dictyostelium discoideum* case studies, for all the intellectually provoking discussions, and for taking the time to read early drafts of my manuscripts and providing feedback on my work. Furthermore I would like to thank Dr. Matthias Maischak for giving me the opportunity to run computational model simulations on the computer cluster of the Mathematics department.

Moreover I would like to thank Professor Joseph O'Rourke who has always provided timely feedback and support on the implementation of the solution to the minimal area enclosing triangle problem.

In addition I would like to thank the thesis examiners, Dr. Larisa Soldatova and Professor Adelinde Uhmacher, for their insightful comments which helped

improve the quality of this thesis.

I would also like to thank all the PhD students in the Department of Computer Science who I have been fortunate to meet, and which have made the PhD journey a more enjoyable experience. I would like to especially thank Dr. Qian Gao, a former PhD student supervised by Professor David Gilbert, who provided many useful insights during my first year as a PhD student.

In addition I would like to thank Teresa Czachowska and Ela Heaney for always providing support regarding administrative and teaching issues.

I would also like to thank Brunel University London for financially supporting my PhD studies.

Finally, and most importantly, I would like to thank my girlfriend, my brother, and my parents, for their unconditional love, support and encouragement. This thesis is dedicated to you.

Author's declaration

This thesis is the result of my own work carried out in the Computer Science department of Brunel University London. The presented results are my own unless otherwise stated in the text. Partial results described in this thesis have been previously published as conference and journal papers. The work presented in this thesis has not been, in full or in part, submitted for any other academic qualification.

London,
August 2015

Ovidiu Pârvu


Introduction

1.1 Systems biology

According to Noble (Noble, 2008) the concept of systems biology dates back to 1865 when Claude Bernard emphasized the need to study biological organisms as collections of interacting rather than independent subsystems (Bernard, 1865). Bernard claimed that new insights could be potentially gained by studying interacting biological subsystems in the context of the entire organism and/or across multiple levels of organization (Dada and Mendes, 2011). Moreover he predicted that the complexity of biological organisms could be overcome by employing mathematical models which abstract away from all biological details irrelevant to the problem one tries to address.

However the term systems biology as we know it today was coined only in 1960 by Dennis Noble who published the first systems level mathematical model of a biological system (Noble, 1960). The model encoded the action and pacemaker potentials of the cardiac muscle and served as an initial proof that new insights can be gained by studying biological organisms at the entire system level using mathematical (and computational) modelling.

In spite of its early development systems biology, as a scientific field, has started receiving significant attention only after the beginning of the 21st century due to the advancements in high-throughput sequencing technologies (Ghosh et al., 2011), which led to an explosion in the amounts of available biological data (e.g. the first human genome sequences (Lander et al., 2001; Venter et al., 2001)), and the increasing availability of computational power, which enabled building more complex computational models.

Although it is now more than ten years since systems biology has gained momentum there is still no single definition for it in the literature (Ideker et al.,

2001; Kitano, 2002a,b). However for the purposes of this thesis we will interpret systems biology as the interdisciplinary field of science whose main aim is to gain a systems level understanding of how natural biological organisms function. For achieving this aim, due to the same reasons as in 1865, one of the main employed approaches is computational modelling.

1.2 Computational models in systems biology

The main benefits of employing computational models in systems biology is that they can provide a better understanding of the mechanisms underlying biological systems, and they can predict how the behaviour of a biological system changes when the system is perturbed (Kell and Knowles, 2006).

One of the main advantages of computational models is that they can be simulated in the *in silico* environment usually faster and cheaper than the corresponding *in vitro* experiments. Moreover biological systems state changes can be observed and/or recorded easier during model simulations than *in vitro* experiments. Conversely the main disadvantage of computational models is that they cannot fully replace *in vitro* experiments when proving biological hypotheses (Kaazempur-Mofrad et al., 2003). Therefore a mixed *in vitro-in silico* approach is usually employed where computational models are used to predict which *in vitro* experiments will provide the most biologically relevant information, and the results obtained from those *in vitro* experiments are used to refine the model such that it can make better predictions (Di Ventura et al., 2006).

Another advantage of employing computational models is that by running *in silico* simulations the number of required *in vitro* human/animal tests could be potentially reduced (Mone, 2014). For instance in scientific fields such as toxicology computational models could replace animals for predicting the potential adverse response of an organism to different chemicals (Andersen and Krewski, 2009; Kleinstreuer et al., 2013).

Similarly computational models could be employed in synthetic biology (Andrianantoandro et al., 2006; Cheng and Lu, 2012; Endy, 2005) studies for predicting how to (genetically) modify a biological system in order to obtain a desired behaviour. Inspired by the success of predictive computational models in engineering (e.g. where a Boeing 777 jet airliner was entirely designed and tested *in silico* before manufacturing (Selick et al., 2002)) some of the envisaged synthetic biology applications are biofuel production using synthetically engineered microorganisms (Lee et al., 2008), microbiome engineering (Ruder et al., 2011) (e.g. using natural commensal microorganisms as a vector for deploying synthetic gene circuits in an

attempt to cure diseases), treatment and prevention of infections (Ruder et al., 2011) (e.g. using bacteriophage viruses to infect specific bacteria and disrupt their antibiotic defence mechanisms), and the development of novel cancer therapies which can distinguish between healthy and cancerous cells (Cheng and Lu, 2012) (e.g. synthetically engineered bacteria could selectively invade tumour tissues and partially inhibit the division of cancerous cells (Weber and Fussenegger, 2012)).

1.2.1 Development

To minimize their complexity, computational models are usually built such that they contain sufficient details about the real system to answer the biological question considered while disregarding any additional information (as per Occam's razor principle). The main advantage of this approach is that model simulations can be executed in reasonable time and the simulation output is easier to interpret. Conversely one of the main disadvantages of this approach is that as many computational models need to be developed as there are biological questions.

The behaviour of computational models is defined by a set of rules which are derived from existing literature and/or new experimental data. Therefore the ability to construct a computational model depends on the availability of prior knowledge and/or the possibility to run new *in vitro* experiments. For lower organisms (e.g. the *Escherichia coli* bacterium) running *in vitro* experiments and obtaining the required information is usually constrained by the availability of physical equipment and/or financial resources. Conversely for higher organisms (e.g. humans) there are additional complexity, ethical and/or legal constraints preventing certain types of *in vitro/in vivo* experiments to be run (e.g. testing the effect of potentially toxic chemicals on humans). Therefore one of the main limitations when modelling higher organisms is the inability to obtain the required information through *in vitro/in vivo* experimentation.

To address this limitation representative lower organisms, called model organisms, are employed instead. Usually the criteria for choosing the model organism are that it is sufficiently genetically/mechanistically similar to the higher organism and relevant *in vitro* experiments can be run. Examples of model organisms which have been successfully used in the past for studying human diseases and/or identifying potential drug targets include the *Caenorhabditis elegans* round worm (Kaletta and Hengartner, 2006), the *Drosophila melanogaster* fruit fly (Pandey and Nichols, 2011) and the *Danio rerio* zebrafish (Lieschke and Currie, 2007).

1.2.2 Types of models

Once the biological question and organism considered are fixed, different types of relevant computational models can be constructed ranging from deterministic to stochastic, discrete- to continuous-time, non-spatial to spatial, respectively spanning one or multiple levels of organization. Moreover after choosing the appropriate computational model type a corresponding modelling formalism is employed for encoding the model (Heath and Kavraki, 2009; Machado et al., 2011). Some of the most employed modelling formalisms in systems biology are *agent based modelling* (An et al., 2009; Thorne et al., 2007), *Boolean networks* (Kauffman, 1969), *Bayesian networks* (Wilkinson, 2007), *cellular automata* (Deutsch and Dormann, 2007; Ermentrout and Edelstein-Keshet, 1993), *constraint-based modelling* (Becker et al., 2007; Orth et al., 2010), *Glazier-Graner-Hogeweg models (also known as Cellular Potts)* (Balter et al., 2007; Graner and Glazier, 1992), *interacting state machines* (Kugler et al., 2010), *P (or membrane) systems* (Barbuti et al., 2011; Besozzi et al., 2008), *ordinary/partial differential equations* (Hoops et al., 2006; Schaff et al., 1997), *Petri nets* (Hardy and Robillard, 2004; Heiner et al., 2008), *process algebras* (Feng and Hillston, 2014; John et al., 2010), and *rule based modelling* (Blinov et al., 2004; Danos et al., 2007; John et al., 2011; Maus et al., 2011; Nikolić et al., 2012). The large number of available modelling formalisms led to the construction of many application-specific computational models which cannot be reused in other applications and/or integrated with other existing models.

1.2.3 Standards

To partially address this problem standard model representation formats and ontologies have been developed which describe the model components and their semantics in a generic manner. Systems biology relevant examples of some of the most used model representation formats include the Systems Biology Markup Language (Hucka et al., 2003) and the Systems Biology Graphical Notation (Novère et al., 2009), respectively some of the most employed ontologies are the Gene Ontology (Ashburner et al., 2000) and the Systems Biology Ontology (Courtot et al., 2011). One of the main limitations of these standard notations is that they were mainly designed for small scale systems (e.g. intracellular pathways) and cannot explicitly encode properties specific to more complex multiscale biological systems.

Large international projects which aim to scale up the existing modelling methodologies and notations to the multiscale context include the International

Union of Physiological Sciences Physiome (Hunter and Borg, 2003), Virtual Physiological Human (Kohl and Noble, 2009), High-Definition Physiology (Kurachi, 2014), Virtual Physiological Rat (Beard et al., 2012), Human Brain (Markram, 2012) and Virtual Liver (Holzhütter et al., 2012) projects. However a generic multiscale modelling framework is yet to emerge (Hoekstra et al., 2014).

1.2.4 Validation

In spite of all their advantages computational models have one important drawback — they are only abstractions of real systems. Therefore any prediction generated by a model can be employed for real-life applications only if the model has been validated first.

Traditionally this has been done by comparing the model simulation output with biological observations recorded in the wet-lab. If significant inconsistencies are detected the model needs to be updated and/or the experiments have to be repeated. The main disadvantage of this approach is that it is both expensive and time consuming.

In an attempt to reduce the costs and detect modelling errors as soon as possible *in silico* model validation methods could be additionally employed (Cheng and Lu, 2012). Although useful, *in silico* model validation approaches complement but cannot replace the corresponding *in vitro* validation experiments. Similarly to computational modelling which reduces the number of required *in vitro* experiments, *in silico* model validation approaches could potentially reduce the number of required *in vitro* validation experiments.

In systems biology one of the most employed *in silico* model validation approaches is model checking, a formal method which automatically decides if a computational model is valid relative to a specification describing the expected system behaviour; see Chapter 2 for a more detailed description of model checking.

1.3 Motivation

Traditional model checking approaches employed in systems biology (see Chapter 2) usually consider only how numeric properties (e.g. concentrations) change over time and are appropriate for small scale biological systems (e.g. intracellular pathways).

However when scaling up to more complex, large scale systems (e.g. multicellular populations) there is an additional need to explicitly consider how properties of (emergent) spatial structures (e.g. area of multicellular population) change over

time, which are not taken into account by traditional non-spatial model checking approaches.

Moreover when modelling biological systems that span multiple levels of organization (e.g. cellular and organ) the relation between changes occurring at different temporal and/or spatial scales needs to be additionally considered.

Therefore the identified limitations of traditional model checking approaches employed for validating computational models of complex large scale biological systems are:

1. The inability to validate models with respect to how spatial structures and their properties change over time;
2. The inability to validate models with respect to how both numeric and spatial properties change over time considering multiple levels of organization.

In order to address these limitations the following challenges are considered:

1. Developing **theoretical models** which explicitly:
 - 1.1 Distinguish between variables encoding numeric values (called numeric state variables) and the spatial domain (called spatial state variables), including functions which compute the values of these variables (called numeric, respectively spatial value assignment functions). The main advantage of distinguishing between numeric and spatial state variables is that state variable type specific analysis functions can be developed (as illustrated by Challenge 2).
 - 1.2 Define and map both numeric and spatial state variables to different scales for reasoning about how changes at one scale reflect at another scale and vice versa.
2. Defining **spatio-temporal analysis approaches** for automatically detecting spatial structures and computing how their properties change over time. These analysis approaches are required for reasoning about spatial structures representing subsets of the spatial domain which were not explicitly encoded in the model but emerge during the model simulation.
3. Creating a **standard representation format** for time series data describing how both numeric and spatial properties change over time with or without explicitly considering different scales. Such a standard representation format is required to enable the exchange of model simulation results across the research community.

4. Defining a **quantitative formal logic** for encoding the specification against which the computational models are validated that enables reasoning about how spatial and numeric properties change over time and across one or multiple scales. There is a need for developing a new formal logic because, to the best of my knowledge, most of the existing spatio-temporal logics are either qualitative or semi-quantitative and are unable to capture how biologically relevant quantitative properties change over time.
5. Adapting existing **model checking algorithms** to the new theoretical model and formal logic, and proving that the algorithms are decidable.

1.4 Contributions

In order to tackle these challenges a novel model checking methodology is defined in this thesis that enables validating multiscale computational models of complex biological systems relative to formal specifications describing how both numeric and spatial properties are expected to change over time and across multiple levels of organization. Throughout this thesis it is assumed that biological systems which span multiple levels of organization (i.e. are multilevel) inherently span multiple spatio-temporal scales (i.e. are multiscale), where each level of organization has a distinct corresponding spatio-temporal scale. Therefore the terms multilevel and multiscale, respectively level and scale are used interchangeably.

The novel model checking methodology has two important advantages. First of all it supports computational models encoded using various high-level modelling formalisms because it is defined relative to time series data and not the models used to produce them. Secondly the methodology is generic because it can be automatically reconfigured according to case study specific types of spatial structures and properties. In addition the methodology can be applied to multiple domains of science (e.g. astrophysics, engineering, environmental science etc.), but for the purpose of this thesis, its efficacy is illustrated only against computational models of biological systems.

In order to automate the computational model validation procedure the model checking method was implemented in model checking software, which is freely available online. The efficacy of the approach is illustrated by employing the model checking software to validate two uniscale spatial and four multiscale (spatial) computational models of biological systems.

1.4.1 Description

The main contributions of this thesis are:

1. ***Multidimensional* spatio-temporal model checking:**
 - a) A ***multidimensional* spatio-temporal model checking methodology** which enables validating uniscale spatial computational models relative to formal specifications. The methodology comprises a theoretical uniscale model for abstractly representing biological systems, spatio-temporal analysis approaches for automatically detecting and analysing specific types of spatial structures in time series data, a standard representation format for the model simulation output, a formal language for writing specifications describing the expected system behaviour, and corresponding Bayesian and frequentist model checking algorithms. Moreover a proof is provided indicating that the multidimensional spatio-temporal model checking problem can be solved considering a finite number of model simulations and simulation time points i.e. it is well-defined (Chapter 3).
 - b) **Cross-platform implementation of the *multidimensional* spatio-temporal model checking approach** in the software tool Mudi made freely available online at <http://mudi.modelchecking.org> (Section 3.7).
 - c) **Validation of the *multidimensional* model checking methodology** against two computational models of biological systems encoding phase variation in bacterial colony growth and the chemotactic aggregation of cells (Chapter 4).
2. ***Multiscale* multidimensional spatio-temporal *meta* model checking:**
 - a) A ***multiscale* multidimensional spatio-temporal model checking methodology** which extends the multidimensional model checking methodology with mechanisms for explicitly representing the hierarchical structure of multiscale biological systems, automatically detecting and analysing certain types of spatial structures from multiple scales, and reasoning about how changes of biological subsystems from different scales relate to each other. Similarly to the multidimensional case a proof is provided to show that the corresponding multiscale model checking problem is well-defined (Chapter 5).

- b) A ***multiscale multidimensional spatio-temporal meta model checking methodology*** that generalizes the multiscale model checking methodology and enables automatically reconfiguring it according to case study specific types of spatial structures and/or properties; the term *meta* is used in meta model checking similarly to how it is used in meta-programming (Sheard, 2001) (where instances of generic meta-programs can be created to solve particular problems), which is different from how the term *meta* is employed in other contexts (e.g. meta data — data about data). From a theoretical point of view meta model checking enables employing arbitrary spatial structure types and properties for the validation of computational models. Conversely from an implementation point of view a meta model checking program, similarly to a meta-program, takes program templates and case study specific types of spatial structures and properties as input and produces a corresponding case study specific model checking program as output. Multiscale multidimensional model checking problems corresponding to different types of spatial structures and/or properties are well-defined (Section 5.7).
- c) **Cross-platform implementation of the *multiscale multidimensional spatio-temporal meta model checking approach*** in the software tool Mule made freely available online at <http://mule.modelchecking.org> (Section 5.8).
- d) **Validation of the *multiscale multidimensional meta model checking methodology*** against four systems biology computational models encoding the rat cardiovascular system dynamics, the uterine contractions of labour, the *Xenopus laevis* oocytes cell cycle and the acute inflammation of the gut and lung (Chapter 6).

1.4.2 Publications

Partial results presented in this thesis have been previously described in the following publications:

1. Pârvu, O. and Gilbert, D. (submitted). A novel method to validate multilevel computational models of biological systems using multiscale spatio-temporal meta model checking. *PLoS ONE*
(Contributed to Chapters 5 and 6);

2. Pârvu, O., Gilbert, D., Heiner, M., Liu, F., Saunders, N., and Shaw, S. (2015). Spatial-Temporal Modelling and Analysis of Bacterial Colonies with Phase Variable Genes. *ACM Trans. Model. Comput. Simul.*, 25(2):13:1–13:25
(Contributed to Chapters 3 and 4);
3. Pârvu, O. and Gilbert, D. (2014a). Automatic validation of computational models using pseudo-3D spatio-temporal model checking. *BMC Systems Biology*, 8(1):124
(Contributed to Chapters 3 and 4);
4. Pârvu, O. and Gilbert, D. (2014b). Implementation of linear minimum area enclosing triangle algorithm. *Computational and Applied Mathematics*, pages 1–16
(Contributed to Chapter 3);
5. Pârvu, O., Gilbert, D., Heiner, M., Liu, F., and Saunders, N. (2013). Modelling and Analysis of Phase Variation in Bacterial Colony Growth. In Gupta, A. and Henzinger, T. A., editors, *Computational Methods in Systems Biology*, number 8130 in LNCS, pages 78–91. Springer Berlin Heidelberg
(Contributed to Chapter 4).

and at the following scientific events:

1. Pârvu, O., Formal validation of multidimensional computational models, *International Study Group for Systems Biology (ISGSB) 2014*, Durham, United Kingdom, 1st–5th September, 2014 (best presentation award);
2. Pârvu, O., Multidimensional model verification, *Doctoral colloquium of the Conference On Spatial Information Theory (COSIT) 2013*, Scarborough, United Kingdom, 2nd–6th September 2013 (best presentation award);
3. Pârvu, O., Gilbert, D., Heiner, M., Liu, F., Saunders, N., A systems biology approach to modelling growth and phase variation in bacterial colonies, *Annual Computational Life and Medical Sciences (CLMS) Symposium 2013*, UCL, London, United Kingdom, 28th June 2013 (3rd best poster award).

1.5 Structure

The thesis is structured as depicted in Figure 1.1, and a description of the next chapters is provided below.

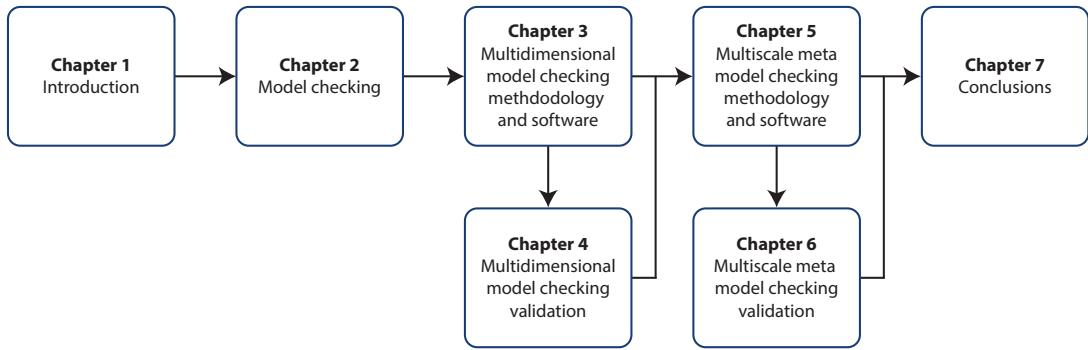


Figure 1.1: Thesis structure where each chapter has an associated number and representative title.

Chapter 2 provides an introduction to model checking and its corresponding model validation workflow comprising model construction, formal specification, and model checker execution. In the beginning theoretical models for abstractly representing real-life systems are presented. Afterwards the main types of formal languages (i.e. linear and branching temporal logics) are described which are employed to write the specifications against which the models are validated. The two major classes of model checking approaches, non-probabilistic and probabilistic, are described next with a focus on the latter due to its wide adoption for the validation of computational models of biological systems. Traditional model checking approaches specific to systems biology applications are reviewed in the end emphasizing that they only consider how numeric properties such as concentrations change over time.

The multidimensional spatio-temporal model checking methodology is introduced in Chapter 3. First of all the theoretical model employed to represent biological systems which evolve in time and space is defined. Secondly the spatio-temporal analysis approaches are described which automatically detect spatial structures in the model simulation output and describe how their properties change over time. Next a standard model simulation output representation format is introduced. Then a formal probabilistic multidimensional spatio-temporal logic is defined for encoding the specifications against which the uniscale computational models are validated. Afterwards the model checking procedure and implementation are described. In the end related approaches are briefly presented.

The validation of the multidimensional spatio-temporal model checking methodology against the phase variation in bacterial colony growth and the chemotactic aggregation of cells biological case studies is described in Chapter 4. The obtained results and identified limitations of the methodology are discussed at the end of the chapter.

Chapter 5 describes how the multidimensional model checking methodology is extended to account for multiscale computational models, and how it can be

parameterized such that the considered spatial structure types and/or properties can be tailored to specific biological questions. The extended theoretical model and the employed data structure for encoding the hierarchical organization of multiscale biological systems are first described. Next the multiscale spatio-temporal analysis approach and the standard simulation output representation format are defined. Then a formal probabilistic multiscale spatio-temporal logic is introduced which enables formally specifying how systems potentially spanning multiple levels of organization are expected to change over time. Afterwards the meta model checking methodology is introduced which enables validating computational models with respect to case study specific types of spatial structures and/or properties. Finally a brief description of the multiscale multidimensional meta model checker implementation, and a comparison with related approaches is given.

The multiscale multidimensional meta model checking methodology is validated in Chapter 6 against four illustrative systems biology case studies encoding the rat cardiovascular system dynamics, the uterine contractions of labour, the *Xenopus laevis* oocytes cell cycle and the acute inflammation of the gut and lung.

Final conclusions, open problems and potential directions for future work are presented in Chapter 7.

CHAPTER 2

Model checking

Introduction

This chapter provides a brief introduction to the formal method called model checking employed to verify if computational models are valid relative to given specifications. The first step for employing model checking approaches is to construct a formal model of the system considered, which can be either probabilistic or not. Afterwards the properties which are expected to hold for the system are encoded as a formal specification using linear or branching time temporal logics. Finally given a model and a specification, a model checking software tool automatically verifies if the specification holds for the model or not. Details regarding each one of these model checking steps are provided in this chapter. In the end model checking approaches specifically employed for validating computational models of biological systems are described.

2.1 Preliminaries

2.1.1 Formal verification methods

Formal verification methods are mathematical approaches employed to (dis)prove the correctness of a system relative to a specification (Baier and Katoen, 2008, Chapter 1). One of their main advantages compared to other verification approaches (e.g. testing) is that they usually consider all possible system behaviours and therefore guarantee the (in)validity of the system relative to the specification. However since all possible system behaviours are considered, formal verification methods cannot usually be employed to validate complex systems in reasonable time.

In the 1970s and early 1980s most formal verification methods involved writing proofs by hand (Clarke, 2008). One of their main disadvantages was that they were not scalable (Emerson, 2008). The main reason for this was that the difficulty of manually writing a proof usually increased with the complexity of the system.

One type of complex systems which could not be verified using formal verification methods was concurrent systems. Moreover such systems could not be verified using testing based approaches either because concurrency errors were usually hard to reproduce.

2.1.2 Model checking background

In order to address the limitations of existing approaches a new concurrent systems formal verification method was developed called model checking (Clarke and Emerson, 1982; Queille and Sifakis, 1982). When it was initially proposed model checking was a formal verification method for finite-state concurrent systems i.e. concurrent systems whose possible behaviours could be described by a finite number of states (e.g. the Alternating Bit network communication protocol (Clarke et al., 1986)). Its novelty relied on the fact that it was an algorithmic approach which could be employed to prove the correctness of a system relative to a specification in an automatic manner without human intervention.

A graphical description of the model checking process is depicted in Figure 2.1. The model checker is usually a software tool which takes as input a formal specification and a mathematical representation of the system (i.e. a model), and automatically decides if the model is valid relative to the specification. Depending on the system considered different formal modelling (see Section 2.2) and specification (see Section 2.3) languages can be employed. The output of the model checker is either yes (i.e. the model is valid) or no (i.e. the model is invalid). In the latter case a counterexample is additionally provided for model debugging purposes.

The general steps for verifying a system using model checking are (Clarke et al., 1999):

1. **Model construction:** Creating an abstract formal representation of the system;
2. **Formal specification:** Encoding the specification using a formal language. The specification should cover all properties that are expected to hold for the system;

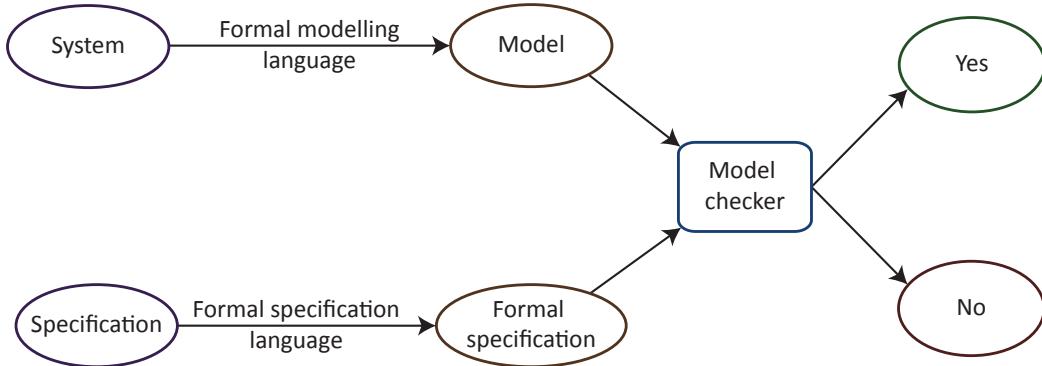


Figure 2.1: Description of the model checking process. The system is encoded as a model using a formal modelling language. Similarly the specification is translated to its formal representation using a formal specification language. Both model and formal specification are taken as input by the model checker which verifies if the model is valid relative to the specification. The output of the model checker is either yes (i.e. the model is valid) or no (i.e. the model is invalid).

3. **Verification:** Ideally totally autonomous, but in reality whenever a counterexample is provided there is typically the need for human intervention.

2.2 Model construction

Real world systems verified using model checking approaches are assumed to be reactive (Clarke et al., 1999, Chapter 1) which means they have the possibility to react to changes in their environment. Moreover the behaviour of such systems is usually described as a sequence of discrete states where changes between states occur instantaneously and are triggered by events.

Such reactive discrete systems are usually represented as state transition systems. Formally these are directed graphs where the vertices represent system states and the edges encode possible transitions between states.

Assuming a directed graph representation, at any given moment the system is described by one vertex/state and can transition to the successor vertices/states indicated by outgoing edges.

An execution/run of the system is described by a computation path through the graph $\sigma = \{s_0, s_1, s_2, \dots\}$, where s_0, s_1, s_2, \dots represent states and for all $s_i \in \sigma, i \geq 1$ there exists a directed edge in the graph starting from $s_{i-1} \in \sigma$ and ending in s_i .

The length of the execution can be finite or infinite. In case of the latter the system either cycles infinitely often through a finite set of states, or the number of states is infinite.

In order to reason about the behaviour of the system, semantic information is associated with states and edges using labelling functions. Moreover the subset of states from which the system execution could start is explicitly defined. This

particular type of system is called a labelled state transition system (LSTS).

2.2.1 Labelled state transition systems

Definition 1 Labelled state transition system (LSTS)

A labelled state transition system (Baier and Katoen, 2008, Chapter 2) is a 6-tuple $\langle S, Act, \rightarrow, I, AP, L \rangle$ where:

- S is the set of states;
- Act is the set of actions;
- $\rightarrow \subseteq S \times Act \times S$ is the relation encoding state transitions;
- $I \subseteq S$ is the set of initial states;
- AP is the set of atomic propositions;
- $L : S \rightarrow 2^{AP}$ is the function employed to label states with atomic propositions.

The set S represents the system states, and the vertices in the corresponding directed graph. Actions executed when the system transitions between states are described by the set Act . Transitions between states, and edges in the corresponding directed graph, labelled by actions are encoded by the relation \rightarrow . The set I represents the initial states from which the execution of the system could start. AP is a set of atomic propositions i.e. Boolean expressions which cannot be divided in simpler statements. Atomic propositions usually encode the semantics of the system and are defined over variables describing the system state (i.e. state variables), constants and predicate symbols. The subset of atomic propositions which evaluate true for a given state is encoded by the labelling function L .

An LSTS is denoted as finite if the sets S , Act and AP are finite (Baier and Katoen, 2008, Chapter 2). Moreover if the number of initial states $|I| = 1$ and the number of possible state transitions $\sum_{\alpha \in Act} |\rightarrow(s_i, \alpha)| \leq 1$ for all states $s_i \in S$ then the LSTS is deterministic. Otherwise it is non-deterministic.

An execution of a LSTS starts in one of the initial states $s_0 \in I$ and proceeds according to the state transition function \rightarrow . Whenever the system transitions from one state s_i to another state s_j the corresponding action α is executed ($(s_i, \alpha, s_j) \in \rightarrow$). The set of atomic properties $L(s_i)$ which hold in each state s_i are computed using the labelling function L .

Example 1 LSTS encoding the life cycle of a *Dictyostelium discoideum* population

Let us assume we would like to model the life cycle of a *Dictyostelium discoideum* (Williams et al., 2006) cellular population using an LSTS. The initial state of the population is “unicellular” when the cells are assumed to be sparsely distributed in space. In the initial state cells can either divide if sufficient nutrients are available in their environment, or start to starve otherwise. If cells divide the population doubles in size and transitions back into the “unicellular” state. Otherwise if cells starve the population transitions into the “aggregating” state. Once cells start to aggregate they will move chemotactically (i.e. chemotact) towards a common point and start forming a slug. When the slug formation is completed the population is assumed to have transitioned from the “aggregating” to the “slug” state. Afterwards cells in the population will start to differentiate and the population transitions to the next “fruiting body” state. Finally the fruiting body will release a new generation of cells and the population will transition back to the initial “unicellular” state.

The corresponding LSTS is defined as follows:

- $S = \{\text{unicellular, aggregating, slug, fruiting body}\};$
- $Act = \{\text{divide, starve, chemotact, differentiate, release}\};$
- $\longrightarrow = \{(\text{unicellular, divide, unicellular}), (\text{unicellular, starve, aggregating}), (\text{aggregating, chemotact, slug}), (\text{slug, differentiate, fruiting body}), (\text{fruiting body, release, unicellular})\};$
- $I = \{\text{unicellular}\};$
- $AP = \{\text{distance} = \text{short}, \text{distance} = \text{medium}, \text{distance} = \text{long}, \text{population} = \text{homogeneous}, \text{population} = \text{heterogeneous}\},$ where the atomic propositions $a \in AP$ were defined over the state variables:
 - $\text{distance} \in \{\text{short, medium, long}\}$ representing the average distance between cells;
 - $\text{population} \in \{\text{homogeneous, heterogeneous}\}$ representing the cellular population type.
- $L = \{(\text{unicellular, (distance} = \text{long, population} = \text{homogeneous)}), (\text{aggregating, (distance} = \text{medium, population} = \text{homogeneous)}), (\text{slug, (distance} = \text{short, population} = \text{homogeneous)}), (\text{fruiting body, (distance} = \text{short, population} = \text{heterogeneous)})\}.$

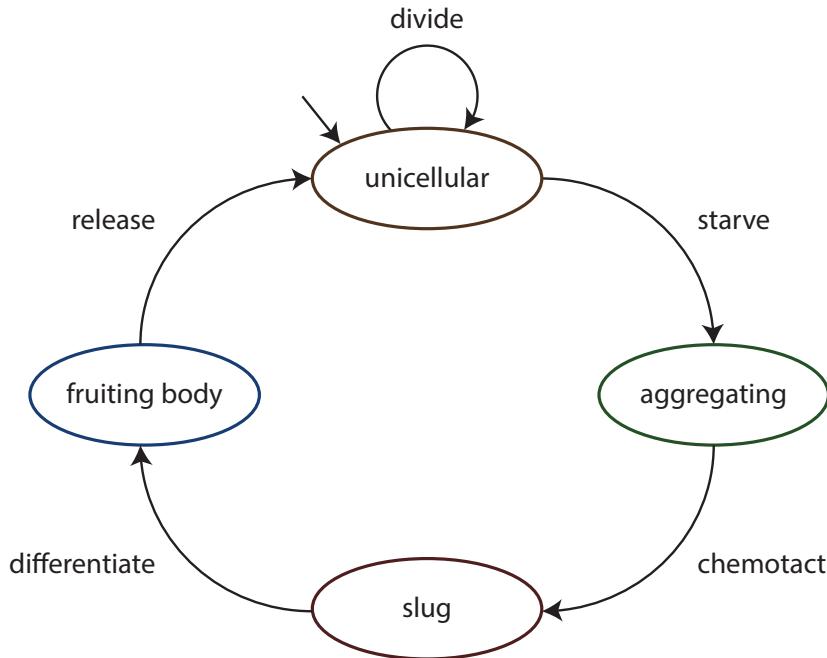


Figure 2.2: Labelled state transition system representing the life cycle of the *Dictyostelium discoideum* cellular population. States are represented as ellipses, and transitions between states as directed arcs. Arc labels describe the actions associated with the corresponding transitions. The initial state is marked by an incoming directed arc which does not have a source state.

A graphical representation of the LSTS is provided in Figure 2.2. ■

The main advantage of the LSTS as a formal representation is that it is generic and can account for many types of real world systems. However due to its general structure an LSTS cannot encode specific types of constraints (e.g. explicitly accounting for time).

In order to address this limitation several modelling formalisms have been developed which build on the concepts of LSTSs. These modelling formalisms include Büchi automata (Büchi, 1962; Vardi and Wolper, 1986) (that additionally specify a set of states called final states in which the system execution ends), timed automata (Alur and Dill, 1994) (which extend Büchi automata with explicit time constraints for modelling real-time systems), hybrid automata (Alur et al., 1995; Henzinger, 1996) (employed to represent systems which have both discrete and continuous semantics), process algebras (Baeten et al., 2010; Ceccarelli et al., 2014) and Petri Nets (Heiner et al., 2008; Peterson, 1981; Petri, 1962) (employed to represent concurrent interacting systems).

Finally one class of real world systems which cannot be represented neither by LSTSs nor extensions thereof are probabilistic systems. The main distinguishing characteristic of probabilistic systems is that transitions between states have an associated probability which cannot be encoded by LSTS based formalisms. In

order to address this limitation probabilistic LSTSs (PLSTS) were defined.

2.2.2 Probabilistic labelled state transition systems

Definition 2 Probabilistic labelled state transition system (PLSTS)

A probabilistic labelled state transition system (Baier and Katoen, 2008, Chapter 10) is a 6-tuple $\langle S, Act, \rightarrow_{prob}, I_{prob}, AP, L \rangle$ where:

- S, Act, AP and L have the same semantics as for an LSTS (see Definition 1);
- $\rightarrow_{prob} : S \times Act \times S \rightarrow [0, 1]$ is the relation encoding state transitions;
- $I_{prob} : S \rightarrow [0, 1]$ is the function employed for encoding initial states.

There are two differences between the definitions of LSTSs and PLSTSs. First of all the state transition function in a PLSTS (i.e. \rightarrow_{prob}) compared to an LSTS (i.e. \rightarrow) additionally associates to each state transition and action a probability value $p \in [0, 1]$. Secondly initial states are chosen probabilistically in a PLSTS (see Definition 2, I_{prob}) rather than deterministically as in an LSTS (see Definition 1, I).

Example 2 PLSTS encoding the life cycle of a *Dictyostelium discoideum* population

In contrast to Example 1 let us assume we would like to model the life cycle of a *Dictyostelium discoideum* (Williams et al., 2006) cellular population in a probabilistic manner using a PLSTS. Two differences will be considered in the description of the case study. Firstly the probability of the system to start in the “unicellular” state is 0.8, in the “fruiting body” state 0.2, and 0 for all other states. Secondly the additional probability associated to the state transition cycling through the “unicellular” state is 0.75, between the “unicellular” and “aggregating” state is 0.25, and 1 for all other state transitions.

The corresponding PLSTS is defined as follows:

- S, Act, AP and L are defined identically to how they were defined for the LSTS in Example 1;
- $\rightarrow_{prob} = \{(\text{unicellular}, \text{divide}, \text{unicellular}, 0.75), (\text{unicellular}, \text{starve}, \text{aggregating}, 0.25), (\text{aggregating}, \text{chemotact}, \text{slug}, 1), (\text{slug}, \text{differentiate}, \text{fruiting body}, 1), (\text{fruiting body}, \text{release}, \text{unicellular}, 1)\};$
- $I_{prob} = \{ (\text{unicellular}, 0.8), (\text{aggregating}, 0), (\text{slug}, 0), (\text{fruiting body}, 0.2) \}.$

A graphical representation of the PLSTS is provided in Figure 2.3.

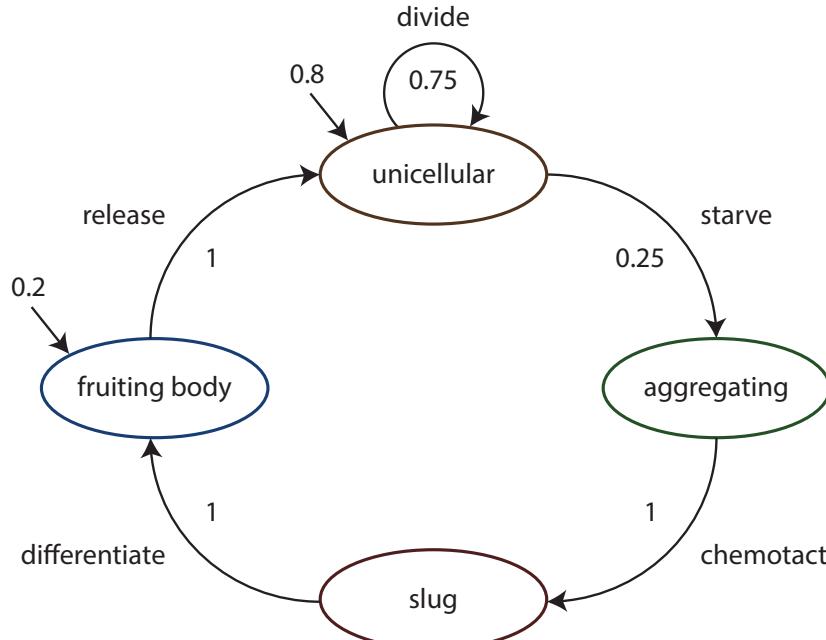


Figure 2.3: Probabilistic labelled state transition system representing the life cycle of the *Dictyostelium discoideum* cellular population. States are represented as ellipses, and transitions between states as directed arcs. Arc labels describe the actions (i.e. alphabetic strings) and probabilities (i.e. numeric values) associated with the corresponding transitions. Potential initial states s_i (i.e. $I(s_i) > 0$) are marked by incoming directed arcs which do not have a source state. Numeric values closest to the incoming directed arcs represent the initial state probabilities $I(s_i)$ of the corresponding states s_i .

Remark 1. The probabilities employed in Example 2 were chosen for explanatory purposes and were not derived from experimental data or the literature. ■

Similarly to LSTSs, PLSTSs have a general structure which does not allow encoding specific types of constraints (e.g. explicitly accounting for time). Modelling formalisms developed to address this limitation include stochastic discrete-event systems (SDES) (Cassandras and Lafortune, 2008, Chapter 6), (Younes, 2005b, Section 2.3) (which can employ either a discrete or continuous time representation) usually represented as discrete or continuous time Markov chains (Norris, 1998), and probabilistic extensions of modelling formalisms employed for LSTSs, such as probabilistic Büchi automata (Baier and Grosser, 2005) (that additionally specify a set of states called final states in which the system execution ends), stochastic timed automata (D'Argenio and Katoen, 2005) (used to represent real-time systems and encode time constraints explicitly), stochastic hybrid automata (Bartocci et al., 2013; Hu et al., 2000) (employed to represent stochastic systems that have both a discrete and continuous semantics), stochastic process algebras (Harrison

and Strulo, 1995; Hermanns et al., 2000) and stochastic Petri nets (Florin et al., 1991) (used to encode stochastic concurrent interacting systems).

2.3 Formal specification

A computational model can be validated automatically by a model checker relative to a specification. Since the model checking process is a formal verification method the specification needs to be written in a formal language. Moreover the chosen formal language needs to contain operators which enable reasoning about how a system changes over time.

A class of formal languages which satisfy this requirement are called temporal logics. Although their foundations were laid more than two millennia ago by philosophers such as Aristotle and Cronus (Øhrstrøm and Hasle, 1995), modern temporal logics, as known today, were formalized only in the second half of the twentieth century by logicians such as Prior (Prior, 1967).

Following on from the development of modern temporal logics Pnueli was the first to illustrate how they could be employed for the verification of concurrent systems (Pnueli, 1977). Usually concurrent systems are formally represented as (P)LSTSs (see Section 2.2) which assume a discrete representation of time. Therefore formal specifications describing the expected behaviour of such systems are usually encoded using temporal logics which similarly assume a point-wise (not interval) and discrete (rather than continuous) representation of time (Emerson, 1995).

Depending on the underlying structure of time the employed point-wise discrete temporal logics can be either linear or branching.

2.3.1 Linear time temporal logics

Linear time logics assume the time structure is linear which means that at each moment a system state has at most one possible successor state (Emerson, 1995; Konur, 2010) as shown in Figure 2.4. The sequence of states describing the changes of the system over time is denoted as a computation path.



Figure 2.4: Linear structure of time. At each moment a system state has at most one successor state.

One of the most employed temporal logics considering a linear time structure used for model checking (concurrent systems) is Linear Temporal Logic

(LTL) (Finkbeiner and Sipma, 2001; Pnueli, 1977).

2.3.1.1 Linear Temporal Logic

Logic statements written in LTL are composed of atomic, Boolean and temporal logic propositions.

Similarly to the description provided in the definition of (P)LSTSSs (see Definitions 1 and 2) atomic propositions are Boolean expressions defined over variables, constants and predicate symbols, which cannot be divided into simpler logic statements.

Conversely a Boolean proposition is a compound statement comprising a Boolean operator and one/two logic propositions (denoted here by ϕ):

- $\sim \phi$ (not): The **negation** of logic proposition ϕ is true i.e. ϕ is false;
- $\phi_1 \wedge \phi_2$ (and): Logic proposition ϕ_1 is true **and** logic proposition ϕ_2 is true;
- $\phi_1 \vee \phi_2$ (or): Logic proposition ϕ_1 is true **or** logic proposition ϕ_2 is true;
- $\phi_1 \Rightarrow \phi_2$ (implication): Logic proposition ϕ_1 is true **implies** logic proposition ϕ_2 is true;
- $\phi_1 \Leftrightarrow \phi_2$ (equivalence): Logic proposition ϕ_1 is true **equivalent to** logic proposition ϕ_2 is true,

where \sim is a unary Boolean operator, and $\wedge, \vee, \Rightarrow, \Leftrightarrow$ are binary Boolean operators.

Finally temporal propositions are used to reason about how the system changes over time. They comprise a temporal operator and logic proposition(s):

- $F\phi$ (**Future**): **Eventually** logic proposition ϕ holds;
- $G\phi$ (**Globally**): Logic proposition ϕ holds **always**;
- $\phi_1 U \phi_2$ (**Until**): Logic proposition ϕ_1 holds **until** logic proposition ϕ_2 holds;
- $X\phi$ (**neXt**): Logic proposition ϕ holds in the **next** time point,

where F, G, U, X are temporal operators.

Syntax

Therefore the syntax of LTL formulae over a set of atomic propositions AP is defined by the following grammar:

$$\phi ::= \text{true} \mid a \mid \phi_1 \wedge \phi_2 \mid \sim \phi \mid X\phi \mid \phi_1 U \phi_2$$

where $a \in AP$, \wedge and \sim are the usual Boolean operators, and X and U are temporal operators.

Only two (X , U) out of four possible temporal operators were specified in the grammar because the other two temporal operators (F , G) can be defined based on the U temporal operator as follows:

$$F\phi \equiv \text{true } U \phi;$$

$$G\phi \equiv \sim F \sim \phi.$$

Similarly the Boolean operators \vee , \Rightarrow and \Leftrightarrow can be defined based on the Boolean operators contained by the grammar (\sim , \wedge):

$$\phi_1 \vee \phi_2 \equiv \sim (\sim \phi_1 \wedge \sim \phi_2);$$

$$\phi_1 \Rightarrow \phi_2 \equiv \sim \phi_1 \vee \phi_2;$$

$$\phi_1 \Leftrightarrow \phi_2 \equiv (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1).$$

Semantics

An LTL formula encodes a property of the system with respect to a linear computation path. Let us denote the computation path as $\sigma = \{s_0, s_1, \dots\}$, where s_0, s_1, \dots is the sequence of states describing how the system changes over time, and σ^i as the suffix of σ starting after the first i states (e.g. $\sigma^2 = s_2, s_3, \dots$). According to this notation σ and σ^0 are identical.

The semantics of an LTL formula with respect to a computation path σ corresponding to a model of the system \mathcal{M} is defined as follows:

- $\sigma \models \text{true};$
- $\sigma \models a \text{ if and only if } a \text{ is true in } s_0;$
- $\sigma \models \phi_1 \wedge \phi_2 \text{ if and only if } \sigma \models \phi_1 \text{ and } \sigma \models \phi_2;$
- $\sigma \models \sim \phi \text{ if and only if } \sigma \not\models \phi;$
- $\sigma \models X\phi \text{ if and only if } \sigma^1 \models \phi;$
- $\sigma \models \phi_1 U \phi_2 \text{ if and only if } \text{there exists } i \geq 0 \text{ such that } \sigma^i \models \phi_2, \text{ and for all } j, 0 \leq j < i, \text{ it holds that } \sigma^j \models \phi_1.$

The extended semantics of an LTL formula with respect to a computation path σ corresponding to a model of the system \mathcal{M} is defined as follows:

- $\sigma \models F\phi \text{ if and only if } \text{there exists } i \geq 0 \text{ such that } \sigma^i \models \phi;$

- $\sigma \models G\phi$ if and only if for all $i \geq 0$ it holds that $\sigma^i \models \phi$;
- $\sigma \models \phi_1 \vee \phi_2$ if and only if $\sigma \models \phi_1$ or $\sigma \models \phi_2$;
- $\sigma \models \phi_1 \Rightarrow \phi_2$ if and only if $\sigma \models \sim \phi_1$ or $\sigma \models \phi_2$;
- $\sigma \models \phi_1 \Leftrightarrow \phi_2$ if and only if $\sigma \models \phi_1 \Rightarrow \phi_2$ and $\sigma \models \phi_2 \Rightarrow \phi_1$.

Example 3 LTL property corresponding to the *Dictyostelium discoideum* life cycle case study

Let us assume that we would like to encode in LTL a logic property corresponding to the *Dictyostelium discoideum* case study described in Example 1. The logic property is defined over the set of atomic propositions AP provided in the LSTS corresponding to the case study (see Example 1), and is described both in natural language and LTL below.

Natural language: Always, if the population is in the “fruiting body” state (identified by atomic propositions $distance = short$, $population = heterogeneous$), then it will next transition into the “unicellular” state (identified by atomic propositions $distance = long$, $population = homogeneous$).

LTL: $G (((distance = short) \wedge (population = heterogeneous))) \Rightarrow (X ((distance = long) \wedge (population = homogeneous))))$.

■

2.3.1.2 Bounded Linear Temporal Logic

One of the limitations of LTL is that it cannot specify logic properties relative to finite sequences of states (e.g. the first 10 states) in a given computation path. Such logic properties are called bounded and are usually employed for complex systems whose behaviour is described as a potentially infinite sequence of states. The evaluation of unbounded logic properties against infinite sequences of states can prove intractable and therefore corresponding bounded logic properties are usually employed instead.

To enable writing such bounded logic properties various extensions of LTL were developed. One of these extensions is a sublogic of Koymans’s Metric Temporal Logic (Koymans, 1990; Zuliani et al., 2010) and is called Bounded Linear Temporal Logic (BLTL). As indicated by Jha et al. (Jha et al., 2009a) BLTL augments classic LTL temporal operators F , G and U with an upper bound $t \in \mathbb{Q}_{\geq 0}$:

- $F^t \phi$: Eventually logic proposition ϕ holds within the time interval $[0, t]$;

- $G^t \phi$: Logic proposition ϕ holds always within the time interval $[0, t]$;
- $\phi_1 U^t \phi_2$: Logic proposition ϕ_1 holds until logic proposition ϕ_2 holds within the time interval $[0, t]$.

Moreover as suggested later by Jha and Ramanathan (Jha and Ramanathan, 2012) it is possible to additionally augment the temporal operators F , G and U with intervals $[t_1, t_2]$, $t_1, t_2 \in \mathbb{Q}_{\geq 0}$, such that logic propositions are evaluated against bounded time intervals which start at time point $t_1 \neq 0$.

Example 4 Bounded LTL property corresponding to the *Dicystostelium discoideum* life cycle case study

Let us assume that we would like to transform the LTL property in Example 3 to a bounded form as described both in natural language and BLTL below.

Natural language: Always within the first ten states (i.e. simulation time interval $[0, 9]$), if the population is in the “fruiting body” state (identified by atomic propositions $distance = short$, $population = heterogeneous$), then it will next transition into the “unicellular” state (identified by atomic propositions $distance = long$, $population = homogeneous$).

BLTL: $G^9 (((distance = short) \wedge (population = heterogeneous)) \Rightarrow (X ((distance = long) \wedge (population = homogeneous))))$.

■

One limitation of both LTL and BLTL is that they cannot express probabilistic logic properties which are usually required for the formal specifications of probabilistic systems.

2.3.1.3 Probabilistic Linear Temporal Logic

To address this limitation probabilistic extensions of LTL and BLTL were developed called Probabilistic Linear Temporal Logic (PLTL) (Baier, 1998), respectively Probabilistic Bounded Linear Temporal Logic (PBLTL) (Langmead, 2009). The difference between (B)LTL and P(B)LTL is that the latter has an additional probabilistic specification associated with the (B)LTL property.

Syntactically a P(B)LTL property ϕ is defined as $P_{\bowtie\theta}[\psi]$ where $\bowtie \in \{<, \leq, \geq, >\}$, $\theta \in (0, 1)$ and ψ is a (B)LTL property. Considering a model of a system \mathcal{M} , the formal specification $\phi \equiv P_{\bowtie\theta}[\psi]$ evaluates to true (i.e. $\mathcal{M} \models P_{\bowtie\theta}[\psi]$) if and only if the probability of ψ to hold for an execution of \mathcal{M} is $\bowtie\theta$.

Example 5 Probabilistic BLTL property corresponding to the *Dicyostelium discoideum* life cycle case study

Let us assume that we would like to translate the BLTL property in Example 4, corresponding to the LSTS in Example 1, to a probabilistic form, applicable to the PLSTS in Example 2. The resulting logic property is described both in natural language and PBLTL below.

Natural language: The probability is greater than 90% that always within the first ten states (i.e. simulation time interval $[0, 9]$), if the population is in the “fruiting body” state (identified by atomic propositions $\text{distance} = \text{short}$, $\text{population} = \text{heterogeneous}$), then it will next transition into the “unicellular” state (identified by atomic propositions $\text{distance} = \text{long}$, $\text{population} = \text{homogeneous}$).

PBLTL: $P > 0.9 [G^9 (((\text{distance} = \text{short}) \wedge (\text{population} = \text{heterogeneous})) \Rightarrow (X ((\text{distance} = \text{long}) \wedge (\text{population} = \text{homogeneous}))))].$

Remark 2. The probabilities employed in Example 5 were chosen for illustrative purposes and were not derived from experimental data or the literature. ■

2.3.2 Branching time temporal logics

Branching time logics assume the structure of time to be tree-like which means that at each moment a system state has zero or more possible successor states (Emerson, 1995; Konur, 2010) as shown in Figure 2.5. Similarly to linear time temporal logics a sequence of states starting in the initial state and describing the changes of the system over time is denoted as a computation path. However in contrast to linear time logics, branching time logics allow reasoning about multiple (or even infinitely many) computation paths branching out from each state.

One of the most well-known branching time logics used for model checking concurrent systems was introduced by Clarke and Emerson (Clarke and Emerson, 1982) and is called Computation Tree Logic (CTL). At approximately the same time a slightly different form of CTL was defined by Queille and Sifakis (Queille and Sifakis, 1982).

2.3.2.1 Computation Tree Logic

In addition to the LTL temporal operators which enable reasoning about the changes of system states over a single computation path, CTL contains new

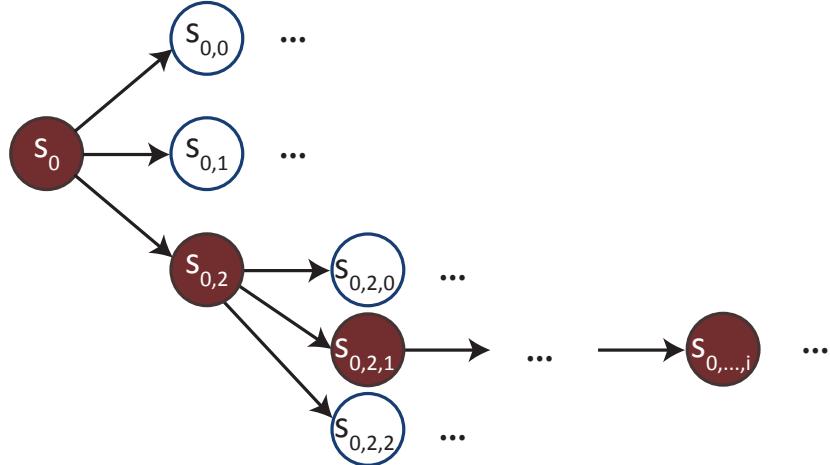


Figure 2.5: Branching structure of time. At each moment a system state has zero or more successor states. An example of a computation path was highlighted in dark red.

temporal operators which enable reasoning over multiple (branching) computation paths:

- $E\phi$ (**Exists**): There **exists** some computation path such that ϕ holds;
- $A\phi$ (**All**): For **all** computation paths ϕ holds.

Syntax

CTL formulae can be of one of the following two types: state and path. State formulae capture properties of atomic propositions in a state and its branching structure. Conversely path formulae describe how the system states change over time with respect to a particular computation path.

The syntax of CTL state formulae over a set of atomic propositions AP is defined by the following grammar:

$$\psi ::= \text{true} \mid a \mid \psi_1 \wedge \psi_2 \mid \sim \psi \mid E\phi$$

where ϕ is a CTL path formula, $a \in AP$, \wedge and \sim are the usual Boolean operators, and E is a CTL specific temporal operator.

The syntax of CTL path formulae over a set of atomic propositions AP is defined by the following grammar:

$$\phi ::= X\psi \mid \psi_1 U \psi_2$$

where ψ , ψ_1 and ψ_2 are CTL state formulae, and X and U are temporal operators.

The extended syntax of CTL is defined similarly to the extended syntax of LTL. For completeness purposes the syntax is given below for three additional

temporal operators (A , F , G):

$$A\phi \equiv \sim E \sim \phi;$$

$$F\psi \equiv \text{true } U \psi;$$

$$G\psi \equiv \sim F \sim \psi,$$

and three additional Boolean connectives ($\vee, \Rightarrow, \Leftrightarrow$):

$$\psi_1 \vee \psi_2 \equiv \sim (\sim \psi_1 \wedge \sim \psi_2);$$

$$\psi_1 \Rightarrow \psi_2 \equiv \sim \psi_1 \vee \psi_2;$$

$$\psi_1 \Leftrightarrow \psi_2 \equiv (\psi_1 \Rightarrow \psi_2) \wedge (\psi_2 \Rightarrow \psi_1).$$

Semantics

A CTL formula encodes a property of the system with respect to a branching computation structure. Let us denote a single computation path in this structure as $\sigma = \{s_0, s_1, \dots\}$, where s_0, s_1, \dots is the sequence of states describing how the system changes over time, σ^i as the suffix of σ starting after the first i states (e.g. $\sigma^2 = s_2, s_3, \dots$), $\sigma[i]$ as the $(i+1)$ -th state in σ (e.g. $\sigma[2] = s_2$), and $Paths(s_i)$ as the computation paths branching out of state s_i .

The semantics of a CTL state formula with respect to a state s corresponding to a model of the system \mathcal{M} is defined as follows:

- $s \models \text{true};$
- $s \models a \text{ if and only if } a \text{ is true in } s;$
- $s \models \psi_1 \wedge \psi_2 \text{ if and only if } s \models \psi_1 \text{ and } s \models \psi_2;$
- $s \models \sim \psi \text{ if and only if } s \not\models \psi;$
- $s \models E\phi \text{ if and only if } \sigma \models \phi \text{ for some } \sigma \in Paths(s).$

The semantics of a CTL path formula with respect to a computation path σ corresponding to a model of the system \mathcal{M} is defined as follows:

- $\sigma \models X\psi \text{ if and only if } \sigma[1] \models \psi;$
- $\sigma \models \psi_1 U \psi_2 \text{ if and only if } \text{there exists } i \geq 0 \text{ such that } \sigma[i] \models \psi_2, \text{ and for all } j, 0 \leq j < i, \text{ it holds that } \sigma[j] \models \psi_1.$

The extended semantics of CTL is defined similarly to the extended semantics of LTL. For completeness purposes it will be stated below:

- $s \models A\phi$ if and only if $\sigma \models \phi$ for all $\sigma \in Paths(s)$;
- $\sigma \models F\psi$ if and only if there exists $i \geq 0$ such that $\sigma[i] \models \psi$;
- $\sigma \models G\psi$ if and only if for all $i \geq 0$ it holds that $\sigma[i] \models \psi$;
- $s \models \psi_1 \vee \psi_2$ if and only if $s \models \psi_1$ or $s \models \psi_2$;
- $s \models \psi_1 \Rightarrow \psi_2$ if and only if $s \models \sim \psi_1$ or $s \models \psi_2$;
- $s \models \psi_1 \Leftrightarrow \psi_2$ if and only if $s \models \psi_1 \Rightarrow \psi_2$ and $s \models \psi_2 \Rightarrow \psi_1$.

Example 6 CTL property corresponding to the *Dictyostelium discoideum* life cycle case study

Let us assume that we would like to encode in CTL a logic property corresponding to the *Dictyostelium discoideum* case study described in Example 1. The logic property is defined over the same set of atomic propositions AP as the LSTS, and is described both in natural language and CTL below.

Natural language: For all computation paths at some point in the future the population will transition into the “unicellular” state (identified by atomic propositions $distance = long$, $population = homogeneous$).

CTL: $A(F((distance = long) \wedge (population = homogeneous)))$. ■

2.3.2.2 Extended Computation Tree Logic

One of the main limitations of CTL with respect to LTL is that its syntax (see Subsubsection 2.3.2.1) does not allow combining path formulae using Boolean operators. For instance the following statement cannot be written in CTL:

$$A(F\psi_1 \wedge F\psi_2).$$

Conversely one of the main limitations of LTL with respect to CTL is that its syntax (see Subsubsection 2.3.1.1) does not enable explicitly addressing the branching structure of time. For example the following statement cannot be expressed in LTL:

$$AG(\psi_1 \Rightarrow EF\psi_2).$$

In order to overcome the limitations of CTL and LTL a more generic and expressive temporal logic was developed called CTL* (Emerson and Halpern,

1986), also known as the extended CTL logic. Similarly to CTL it distinguishes between state and path formulae. However in contrast to CTL it allows writing Boolean propositions comprising path formulae (as in LTL). Therefore CTL* is a variant of CTL in which path formulae are replaced by LTL formulae.

Syntax

The syntax of CTL* state formulae over a set of atomic propositions AP is defined by the following grammar:

$$\psi ::= \text{true} \mid a \mid \psi_1 \wedge \psi_2 \mid \sim \psi \mid E\phi$$

where ϕ is a CTL* path formula, $a \in AP$, \wedge and \sim are the usual Boolean operators, and E is a temporal operator.

The syntax of CTL* path formulae over a set of atomic propositions AP is defined by the following grammar:

$$\phi ::= \psi \mid \phi_1 \wedge \phi_2 \mid \sim \phi \mid X\phi \mid \phi_1 U \phi_2$$

where ψ is a CTL* state formula, ϕ , ϕ_1 and ϕ_2 are CTL* path formulae, \wedge and \sim are the usual Boolean operators, and X and U are temporal operators.

The extended syntax of CTL* is defined similarly to LTL and CTL. For completeness purposes the syntax is given below for both state and path formulae considering three additional temporal operators (A , F , G):

$$A\phi \equiv \sim E \sim \phi;$$

$$F\phi \equiv \text{true} \ U \phi;$$

$$G\phi \equiv \sim F \sim \phi,$$

and three additional Boolean connectives ($\vee, \Rightarrow, \Leftrightarrow$):

$$\psi_1 \vee \psi_2 \equiv \sim (\sim \psi_1 \wedge \sim \psi_2);$$

$$\psi_1 \Rightarrow \psi_2 \equiv \sim \psi_1 \vee \psi_2;$$

$$\psi_1 \Leftrightarrow \psi_2 \equiv (\psi_1 \Rightarrow \psi_2) \wedge (\psi_2 \Rightarrow \psi_1);$$

$$\phi_1 \vee \phi_2 \equiv \sim (\sim \phi_1 \wedge \sim \phi_2);$$

$$\phi_1 \Rightarrow \phi_2 \equiv \sim \phi_1 \vee \phi_2;$$

$$\phi_1 \Leftrightarrow \phi_2 \equiv (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1).$$

Semantics

The semantics of a CTL* state formula with respect to a state s corresponding to a model of the system \mathcal{M} is defined as follows:

- $s \models \text{true};$
- $s \models a \text{ if and only if } a \text{ is true in } s;$
- $s \models \psi_1 \wedge \psi_2 \text{ if and only if } s \models \psi_1 \text{ and } s \models \psi_2;$
- $s \models \sim \psi \text{ if and only if } s \not\models \psi;$
- $s \models E\phi \text{ if and only if } \sigma \models \phi \text{ for some } \sigma \in \text{Paths}(s).$

The semantics of a CTL* path formula with respect to a computation path σ corresponding to a model of the system \mathcal{M} is defined as follows:

- $\sigma \models \psi \text{ if and only if } \sigma[0] \models \psi;$
- $\sigma \models \phi_1 \wedge \phi_2 \text{ if and only if } \sigma \models \phi_1 \text{ and } \sigma \models \phi_2;$
- $\sigma \models \sim \phi \text{ if and only if } \sigma \not\models \phi;$
- $\sigma \models X\phi \text{ if and only if } \sigma^1 \models \phi;$
- $\sigma \models \phi_1 U \phi_2 \text{ if and only if } \text{there exists } i \geq 0 \text{ such that } \sigma^i \models \phi_2, \text{ and for all } j, 0 \leq j < i, \text{ it holds that } \sigma^j \models \phi_1,$

where σ^i is the suffix of σ starting after the first i states (e.g. $\sigma^2 = s_2, s_3, \dots$).

The extended semantics of CTL* is defined similarly to the extended semantics of CTL and LTL:

- $s \models A\phi \text{ if and only if } \sigma \models \phi \text{ for all } \sigma \in \text{Paths}(s);$
- $\sigma \models F\phi \text{ if and only if } \text{there exists } i \geq 0 \text{ such that } \sigma^i \models \phi;$
- $\sigma \models G\phi \text{ if and only if } \text{for all } i \geq 0 \text{ it holds that } \sigma^i \models \phi;$
- $s \models \psi_1 \vee \psi_2 \text{ if and only if } s \models \psi_1 \text{ or } s \models \psi_2;$
- $s \models \psi_1 \Rightarrow \psi_2 \text{ if and only if } s \models \sim \psi_1 \text{ or } s \models \psi_2;$
- $s \models \psi_1 \Leftrightarrow \psi_2 \text{ if and only if } s \models \psi_1 \Rightarrow \psi_2 \text{ and } s \models \psi_2 \Rightarrow \psi_1;$
- $\sigma \models \phi_1 \vee \phi_2 \text{ if and only if } \sigma \models \phi_1 \text{ or } \sigma \models \phi_2;$
- $\sigma \models \phi_1 \Rightarrow \phi_2 \text{ if and only if } \sigma \models \sim \phi_1 \text{ or } \sigma \models \phi_2;$

- $\sigma \models \phi_1 \Leftrightarrow \phi_2$ if and only if $\sigma \models \phi_1 \Rightarrow \phi_2$ and $\sigma \models \phi_2 \Rightarrow \phi_1$,

where σ^i is the suffix of σ starting after the first i states (e.g. $\sigma^2 = s_2, s_3, \dots$).

Example 7 CTL* property corresponding to the *Dictyostelium discoideum* life cycle case study

Let us assume that we would like to encode in CTL* a logic property corresponding to the *Dictyostelium discoideum* case study described in Example 1. The logic property is defined over the same set of atomic propositions AP as the LSTS, and is described both in natural language and CTL* below.

Natural language: There exists a computation path in which the population will eventually transition into the “unicellular” state (identified by atomic propositions $distance = long$, $population = homogeneous$), and will eventually transition into the “fruiting body” state (identified by atomic propositions $distance = short$, $population = heterogeneous$).

CTL*: $E(F((distance = long) \wedge (population = homogeneous))) \wedge F((distance = short) \wedge (population = heterogeneous)))$.

■

2.3.2.3 Bounded and probabilistic branching time logics

Following on from the same reasons as in the case of LTL (see Subsubsections 2.3.1.2 and 2.3.1.3) bounded (Emerson et al., 1992; Lewis, 1990; Ruf and Kropf, 1997) and probabilistic (Aziz et al., 1996; Hansson and Jonsson, 1994) extensions of CTL were developed.

The syntax and semantics of the bounded and probabilistic CTL extensions will not be introduced here because they will not be explicitly used in the remainder of this thesis; see corresponding references for details.

2.4 Model verification

Considering a model of a system \mathcal{M} and a formal specification ϕ the model checking problem is to algorithmically verify if \mathcal{M} is valid relative to ϕ (i.e. $\mathcal{M} \models \phi$) (Baier and Katoen, 2008).

The solution to the model checking problem varies with the considered model type which can be represented either as an LSTS or PLSTS.

2.4.1 Model checking labelled state transition systems

If the system is represented as an LSTS \mathcal{M} , the model checking problem is to determine if the formal specification ϕ evaluates to true for all computation paths starting from the initial states $s_0 \in I \subseteq S$, where S and I are the sets of states, respectively initial states corresponding to \mathcal{M} (Clarke et al., 1999, Chapter 4).

In order to ensure that the provided answer is correct, algorithms solving the model checking problem for an LSTS \mathcal{M} explore the entire state space of \mathcal{M} in a brute-force manner. Since the size of the state space can become very large (e.g. 10^{120} (Miller et al., 2010)) efficient data structures and algorithms for state space exploration were developed; they differ depending on the temporal logic used to encode the formal specification ϕ .

2.4.1.1 LTL model checking

In case the specification is written in LTL (see Subsubsection 2.3.1.1), a structure is usually constructed which records how atomic propositions and logic subformulae of ϕ evaluate for each state of the model \mathcal{M} . Typical examples of such structures are tableaus (Lichtenstein and Pnueli, 1985) (i.e. graphs encoding which logic formulae hold in each state and how the system can transition between states) or Büchi automata (Vardi and Wolper, 1986) (i.e. an automaton which encodes how a system transition between states when one or multiple logic subformulae hold). Counterexample computation paths are searched in these structures and if found, the model is declared invalid, otherwise it is declared valid. The computational complexity of these LTL model checking algorithms ($O(|\mathcal{M}|2^{|\phi|})$) is linear in the size of the model \mathcal{M} and exponential in the size of the specification ϕ .

One of the most prominent LTL model checkers based on modelling the LTL formula as a Büchi automaton (Vardi and Wolper, 1986) was developed by Holzmann and is called SPIN (Holzmann, 1997). The model of the system is described using the language PROMELA and LTL formulae are verified using “on-the-fly” verification techniques (Gerth et al., 1996). Conversely, one of the most well-known tableau-based (Clarke et al., 1997) model checkers is NuSMV (Cimatti et al., 1999, 2002).

2.4.1.2 CTL model checking

Conversely in case of CTL specifications (see Subsubsection 2.3.2.1) labelling functions are usually employed which evaluate atomic propositions and logic subformulae of ϕ for each state of the model \mathcal{M} (Clarke et al., 1986). The corresponding model checking algorithms work in iterations by first labelling

states with logic formulae of length one, then of length two, and so on and so forth until the length of the specification ϕ is reached. Based on the results of the labelling functions the model is declared valid if and only if all initial states of \mathcal{M} are labelled with the specification ϕ (i.e. ϕ holds in these states). The computational complexity of these CTL model checking algorithms ($O(|\mathcal{M}||\phi|)$) is linear in the size of both the model \mathcal{M} and the specification ϕ .

The significant complexity difference between LTL ($O(|\mathcal{M}|2^{|\phi|})$) and CTL ($O(|\mathcal{M}||\phi|)$) model checking algorithms is due to the type of properties which can be expressed in each logic.

The first CTL model checker was developed and described by Clarke and Emerson and was called EMC (Clarke and Emerson, 1982). Approximately at the same time Queille and Sifakis presented the model checker CESAR (Queille and Sifakis, 1982) which was taking as input properties formalised in a temporal branching logic very similar to CTL. EMC was later optimised and extended to support fairness constraints (Clarke et al., 1986).

2.4.1.3 CTL* model checking

Finally in case of CTL* specifications (see Subsubsection 2.3.2.2) the model checking algorithms usually employ the labelling function based approach for CTL state and path logic subformulae, and the structure based approach for LTL path subformulae (Emerson and Lei, 1987). Similarly to LTL the computational complexity of CTL* model checking algorithms ($O(|\mathcal{M}|2^{|\phi|})$) is linear in the size of the model \mathcal{M} and exponential in the size of the specification ϕ .

One of the first CTL* model checkers was described by Visser et al. (Visser et al., 1997) and is called AltMC. The SPIN model checker could be also extended to support CTL* specifications as suggested by Visser and Barringer (Visser and Barringer, 2000).

2.4.1.4 State space explosion problem

In spite of the efficient data structures and independently of the temporal logic employed to encode the formal specification, the complexity of the model checking algorithms depends on the size of the state space (see complexity of LTL, CTL and CTL* algorithms above). For complex systems the number of states usually increases exponentially with the number of concurrent processes considered and the domains of possible state variable values. This is known as the state space explosion problem and is the cause for the main limitation of model checking approaches i.e. poor scalability.

In an attempt to address this limitation multiple approaches for reducing the size of the state space have been developed; several of the most employed approaches are depicted in Figure 2.6.

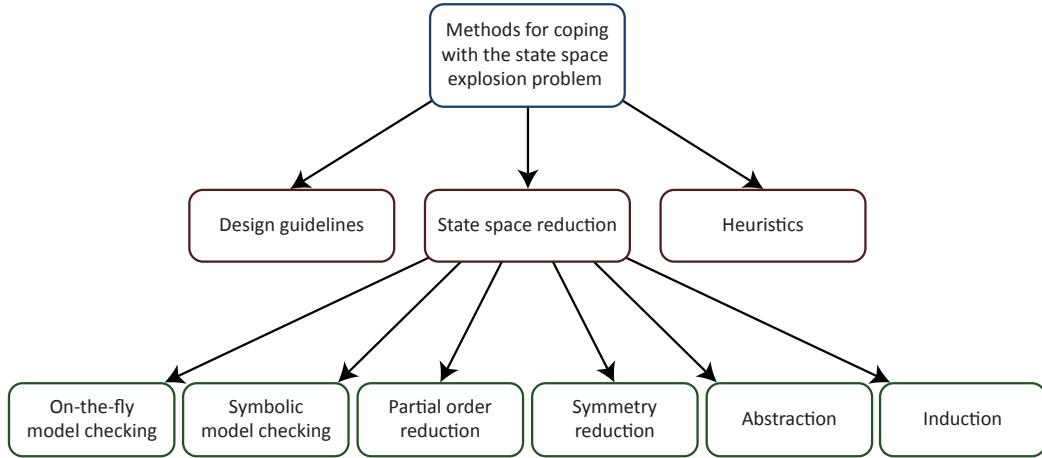


Figure 2.6: Well-known approaches for tackling the state space explosion problem. State space reduction methods are traditional while heuristics and design guidelines are more recent.

State space reduction

Most approaches attempt to combat the state space explosion problem by reducing the size of the state space and/or the memory footprint. The most well-known methods of this type are:

- **On-the-fly model checking:** Compared to explicit state static (Rafe et al., 2013) model checking, on-the-fly model checking dynamically constructs only the required part of the state space (Gerth et al., 1996).
- **Symbolic model checking:** The state space is represented in a compact form (symbolically) using binary decision diagrams (BDD) (Bryant, 1986). Each symbolic state in the BDD represents a subset of states in the original state space. Therefore a BDD representation of the state space can be exponentially more compact than the original representation (Clarke et al., 2001). Advances in BDD-based data structures enabled representing state spaces of sizes up to 10^{120} states.
- **Partial order reduction:** When modelling concurrent systems there usually are sequences of events which occur in parallel. For completeness purposes all possible permutations of the order in which the events occur need to be considered. This leads to an exponential increase in the size of the state space. However if the events are independent from each other (with respect to the property to be checked) the order in which they are executed

does not influence the final outcome. Thus only one of the sequences needs to be considered. Partial order methods reduce the size of the state space by applying this principle and thus eliminating all redundant states (Godefroid, 1991; Peled, 1994; Valmari, 1991).

- **Symmetry reduction:** Systems usually comprise multiple subcomponents, some of which might be identical. Symmetry reduction takes advantage of this fact and simplifies the model by removing subcomponents from the system which are identical with respect to an equivalence relation (Clarke et al., 1996; Emerson and Sistla, 1996; Ip and Dill, 1996).
- **Abstraction:** From the point of view of the specification the model should contain enough details to enable checking all the properties of the system but not more. Abstraction is a state space reduction method which abstracts away all unnecessary details of the model with respect to a formal specification. Some of the most well-known methods involve eliminating variables which do not have an effect on the variables described in the specification, and mapping the set of data values to a smaller set of abstract data values (Baier and Katoen, 2008, Chapter 7),(Clarke et al., 1999, Chapter 13).
- **Induction:** Some systems can be described as the composition of multiple copies of a single subsystem. If it is possible to show that one subsystem is always valid (or invariant) with respect to the given formal specification, and that the composition of this subsystem with the $(i + 1)$ -th copy is also always valid, then by induction the composition of multiple copies of the subsystem will be always valid (Clarke et al., 1986; Clarke, 2008; Kurshan and McMillan, 1989).

Heuristics

An alternative category of methods for tackling the state space explosion problem is based on heuristics. They are mainly used to explore the state space in a directed manner to find states which violate the formal specification. An example of a model checking approach using random-walks guided by heuristics is given by Bui and Nymeyer (Bui and Nymeyer, 2009), and an example employing genetic algorithms to explore large state spaces is given by Godefroid and Khurshid (Godefroid and Khurshid, 2002).

Design guidelines

Finally the third distinct strand (in some respects) is to construct verifiable models by following a set of five guidelines starting from the model design stage (Groote et al., 2012). Some of the given rules (e.g. using data categories instead of explicit data values) are similar to the more traditional methods (e.g. abstraction) while others (e.g. information polling) are challenging them (e.g. information pushing).

Although multiple specific approaches for combating the state space explosion problem were developed over the years a unified solution is yet to emerge.

2.4.2 Model checking probabilistic labelled state transition systems

In case the system is represented as a PLSTS model \mathcal{M} , the probabilistic model checking problem is to determine if the formal specification ψ evaluates to true with probability $\bowtie \theta$ for all computation paths starting from the initial states $s_0 \in S$ ($I_{prob}(s_0) > 0$), where $\bowtie \in \{<, \leq, \geq, >\}$, $\theta \in (0, 1)$, and S is the set of states corresponding to \mathcal{M} (Baier and Katoen, 2008, Chapter 10) (Vardi and Wolper, 1986).

The main difference between model checking an LSTS and a PLSTS is that in case of the latter probabilities over computation paths need to be additionally computed.

2.4.2.1 Computing probabilities over computation paths

For calculating probabilities over computation paths a corresponding probability measure needs to be defined which computes the likelihood of a subset of computation paths to be generated (representing events) considering the collection of all possible computation paths (representing the sample space). The mathematical construct encoding a sample space, a corresponding subset of events and a probability measure is denoted as a probability space.

Definition 3 Probability space

A probability space is a triple $\langle \Omega, \mathcal{F}, \mathcal{P} \rangle$ such that:

- Ω represents the entire sample space;
- \mathcal{F} is a σ -algebra $\subseteq 2^\Omega$ representing subsets of events;
- \mathcal{P} is a probability measure defined on \mathcal{F} with values in $[0, 1]$.

For defining such a probability space the structure of the probabilistic models used to generate all possible computation paths needs to be fixed. Since biological systems are usually modelled as stochastic processes which transition from the current state to the successor state when an event occurs (e.g. a biochemical reaction) we will assume throughout this thesis that all probabilistic systems are represented as SDESs.

Definition 4 Stochastic discrete-event system (SDES)

The factored representation of an SDES \mathcal{M} (Younes, 2005b, Chapter 2) is a 5-tuple $\langle S, T, \mu, SV, V \rangle$ where:

- S is the set of all possible states of the system;
- T is the transition rates matrix which records the probability of the system to transition from the current state s_i to the next state s_j , $\forall s_i, s_j \in S$;
- μ is a probability measure defined over sets of computation paths;
- SV is the set of state variables describing the state of the system;
- V is the value assignment function which computes the value $\in \mathbb{R}$ of each state variable for a given computation path and state of the system.

From a structural point of view one of the main differences between a PLSTS and an SDES is that in case of the latter states are not labelled with atomic propositions but are described by a set of state variables (SV) which can be evaluated (V) in each system state. Moreover actions (Act) and initial states (I_{prob}) are not explicitly encoded in a SDES, and there is an additional function (μ) for computing probabilities over computation paths.

The behaviour of an SDES \mathcal{M} changing over time can be captured by a sequence of pairs (s, t) where s is the current state of the system and t the amount of time spent in state s . Therefore a computation path or trajectory

$$\sigma = (s_0, t_0), (s_1, t_1), \dots$$

is a sequence of pairs (s, t) describing the evolution of \mathcal{M} along the sequence of states s_0, s_1, \dots with $t_0, t_1, \dots \in \mathbb{R}$ time durations spent in each state. For all s_i the probability $P(s_i, s_{i+1})$ to make a transition to state s_{i+1} is greater than zero. Moreover it is assumed throughout that the behaviour of the system is time

divergent (non-Zeno) which means that it is not possible for an infinite number of transitions to occur in a finite amount of time.

Therefore a constraint imposed on \mathcal{M} is that it cannot make infinite numbers of transitions between states in a finite amount of time. This means that for any given sequence of times t_0, t_1, \dots associated with a simulation of \mathcal{M} ,

$$\sum_{i=0}^{\infty} t_i = t_0 + t_1 + \dots$$

must be a divergent series.

Computation paths can be either finite or infinite. Similarly to the definition provided for CTL let us denote the set of all paths starting from state s by $Paths(s)$. Moreover denote the set of all finite paths in \mathcal{M} by $Paths_{finite}(\mathcal{M})$ and the set of all infinite paths by $Paths_{infinite}(\mathcal{M})$. A prefix of a computation path $\sigma = (s_0, t_0), (s_1, t_1), \dots$ is a sequence of pairs $\sigma' = (s'_0, t'_0), (s'_1, t'_1), \dots, (s'_k, t'_k)$ such that $(s'_i, t'_i) = (s_i, t_i)$ for all $0 \leq i \leq k$.

Based on the collection of paths $Paths(\mathcal{M})$ generated by a SDES model \mathcal{M} a probability space $\langle \Omega_{\mathcal{M}}, \mathfrak{E}_{\mathcal{M}}, \mu \rangle$ can be defined where:

- $\Omega_{\mathcal{M}} = Paths(\mathcal{M})$;
- The σ -algebra $\mathfrak{E}_{\mathcal{M}} \subseteq 2^{\Omega_{\mathcal{M}}}$ associated with \mathcal{M} contains all the cylinder sets $C(\sigma_{finite})$ ranging over the set of finite paths $Paths_{finite}(\mathcal{M})$, where

$$C(\sigma_{finite}) = \{\sigma \in Paths_{infinite}(\mathcal{M}) \mid \sigma_{finite} \text{ is a prefix of } \sigma\}$$

for all $\sigma_{finite} \in Paths_{finite}(\mathcal{M})$;

- The unique SDES probability measure μ is defined on the σ -algebra $\mathfrak{E}_{\mathcal{M}}$ (Baier and Katoen, 2008, Chapter 10) where the probabilities of cylinder sets $C(\sigma_{finite})$ defined over finite prefixes $\sigma_{finite} = (s_0, t_0), (s_1, t_1), \dots, (s_n, t_n)$ are computed as follows:

$$\mu(C(\sigma_{finite})) = I_{prob}(s_0) \cdot Pr(s_0, s_1, \dots, s_n),$$

where

$$Pr(s_0, s_1, \dots, s_n) = \prod_{i=0}^{n-1} T(s_i, s_{i+1}),$$

and $I_{prob}(s_i)$ denotes the probability of the system to start in state s_i . In case of zero length computation paths $Pr(s_0) = 1$.

Depending on the required level of accuracy for the model checking results, the probabilistic model checking problem can be solved using either exhaustive or approximate approaches.

2.4.2.2 Exhaustive probabilistic model checking

Exhaustive probabilistic (numerical) model checking algorithms explore the entire state space in a brute-force manner to determine if the model is valid relative to the specification.

Considering an SDES \mathcal{M} with a well-defined probability space and a formal specification $\phi \equiv P_{\bowtie\theta}[\psi]$, it holds that $\mathcal{M} \models P_{\bowtie\theta}[\psi]$ if and only if

$$\mu\{\sigma \in \text{Paths}(s) \mid \sigma \models \psi\} \bowtie \theta,$$

where s is an initial state in \mathcal{M} .

In order to explicitly account for probabilities model checking algorithms employed for LSTSs and LTL, CTL, or CTL* formal specifications have been adapted to PLSTSs, and formal specifications encoded in PLTL (Courcoubetis and Yannakakis, 1995), PCTL (Hansson and Jonsson, 1994), CSL (Aziz et al., 2000) or PCTL* (Aziz et al., 1995).

Similarly to the algorithms employed for LSTSs exhaustive probabilistic model checking approaches depend on the size of the state space, and therefore suffer from the state space explosion problem. Potential solutions (e.g. state space reduction techniques) applied to LSTSs have been adapted to PLSTSs (Baier et al., 1999).

Two well-known model checkers which support exhaustive probabilistic model checking approaches are MRMC (Katoen et al., 2011) and PRISM (Kwiatkowska et al., 2011).

2.4.2.3 Approximate probabilistic model checking

Approximate probabilistic (statistical) model checking approaches explore the state space in a partial manner and therefore determine if a model is valid relative to a specification based only on a finite subset of simulations (Legay et al., 2010).

Approximate probabilistic model checking approaches are usually employed whenever exhaustive alternatives are either too slow or cannot be used due to the large (potentially infinite) size of the state space.

Similarly to exhaustive approaches approximate probabilistic model checking algorithms decide if the probability p of a logic property ψ to hold for a model \mathcal{M} is $\bowtie \theta$. However in contrast to exhaustive approaches they do not compute

the exact value of p by considering all possible model simulations, but instead approximate it based on a finite subset of simulations. Although the correctness of the result is not guaranteed an upper bound can be placed on the tolerated approximation error.

To ensure that approximation errors are below the thresholds specified by the user methods from statistics are usually employed. Using such methods requires rephrasing the traditional model checking problem as a statistical problem.

Let us assume that an SDES \mathcal{M} is validated relative to a temporal logic property $\phi \equiv P_{\bowtie\theta}[\psi]$. The model is declared valid if the probability p of a randomly generated computation path σ to hold relative to ψ is $\bowtie\theta$, and invalid otherwise. For approximating the value of p the model \mathcal{M} is simulated multiple times.

Each model simulation can be represented as an experiment which evaluates to true relative to specification ψ with probability p , and false with probability $1 - p$. Therefore the evaluation of each model simulation can be represented as a Bernoulli random variable X which takes the value 1 (i.e. success) with probability p and 0 (i.e. failure) with probability $1 - p$. In general n simulations can be represented as a sequence of independent, identically distributed (iid) Bernoulli variables X_1, X_2, \dots, X_n , where each X_i is a Bernoulli variable with the success probability p . The sum of a sequence of iid Bernoulli variables $Y = \sum_{i=1}^n X_i$ is a random variable that follows a binomial distribution with parameters n and p .

Based on the Bernoulli or binomial distribution representation several frequentist and Bayesian approximate probabilistic model checking approaches have been developed (Grosu and Smolka, 2005; Héault et al., 2004; Jha et al., 2009a,b; Langmead, 2009; Sen et al., 2004; Younes et al., 2006; Younes and Simmons, 2002; Younes, 2005a,b). A comparison of the main approaches was previously given by Reijnsbergen et al. (Reijnsbergen et al., 2014, 2015) and therefore will not be restated here.

Depending on the employed method and temporal logic the expected complexity of the model checking algorithms differs. In contrast to exhaustive approaches, the complexity of approximate probabilistic model checking algorithms is proportional to the length of the computation paths instead of the size of the state space.

A classification of approximate probabilistic model checking approaches considered throughout this thesis is given in Table 2.1 and a brief description of each approach in Appendix A.

To determine if a formal specification $\phi \equiv P_{\bowtie\theta}[\psi]$ holds for a system model \mathcal{M} the approximate probabilistic model checking approaches considered estimate

Table 2.1: Classification of considered approximate probabilistic model checking approaches. Bayesian methods consider prior knowledge about the parameters and variables in the model when deciding if a logic property holds. Conversely frequentist approaches assume no prior knowledge is available. All methods except probabilistic black-box take as input a user-defined upper bound on the approximation error. They request additional model executions until the result is sufficiently accurate. Probabilistic black-box model checking takes a fixed number of model simulations as input and computes a p-value as the confidence measure of the result.

	Frequentist	Bayesian
Estimate	Chernoff-Hoeffding bounds (Hérault et al., 2004)	Mean and variance (Langmead, 2009)
Hypothesis testing	Statistical (Younes et al., 2006; Younes, 2005b) Probabilistic black-box (Sen et al., 2004; Younes, 2005a)	Statistical (Jha et al., 2009a,b)

the probability p of ϕ evaluating true as the number of model simulations for which ϕ holds divided by the total number of model simulations. Depending on the model checking approach considered the total number of model simulations, and the way in which p is afterwards compared to θ considering a user-defined tolerated approximation error differ.

Illustrative examples of approximate probabilistic model checkers include APMC (Hérault et al., 2004), MC2 (Donaldson and Gilbert, 2008b) and PRISM (Kwiatkowska et al., 2011).

2.4.2.4 Comparing probabilistic model checking approaches

The main advantage of exhaustive probabilistic model checking approaches is that they compute highly accurate values of the probabilities when deciding if a logic property ϕ holds for a model \mathcal{M} . The state space is explored exhaustively or until enough evidence is provided that ϕ does not hold. Therefore the main disadvantage of exhaustive approaches is that their complexity is proportional to the size of the state space. Although state space reduction techniques have been developed (see Subsubsection 2.4.1.4) models with large state spaces cannot be verified in reasonable time (e.g. a tandem queuing network computational model with state space of size approximately 10^7 was validated in more than $5 \cdot 10^5$ seconds i.e. approximately 6 days (Younes et al., 2006, Figure 5a)).

Conversely the main advantage of approximate probabilistic model checking approaches is that their complexity does not scale up with the size of the state space. Therefore they can be employed for both small and large scale models. Moreover approximate model checking approaches are model independent. Their main disadvantage is that the provided answer is an approximation and is not guaranteed to be correct. However all approximate methods considered provide an upper bound on the approximation error or compute the confidence level of

the answer. In theory setting the approximation error to minimum, and the measure of confidence to maximum, would yield the same results as the exhaustive approaches.

Thus exhaustive approaches are appropriate for models with a manageable state space for which exact probabilities need to be computed. Approximate approaches are suitable for both small and large scale models for which approximate probability values suffice.

2.5 Model checking computational models of biological systems

In computational (systems) biology, model checking approaches have been employed to solve four different classes of problems:

- **Model validation problems:** The main aim is to check if a computational model is valid relative to a given formal specification (e.g. (Calder et al., 2006; Chabrier-Rivier et al., 2004; Heath et al., 2008)).
- **Robustness computation problems:** The main aim is to estimate the robustness (Kitano, 2007) of the system to perturbations (e.g. (Bartocci et al., 2015; Česka et al., 2014; Fages and Rizk, 2009; Rizk et al., 2009)). Perturbations are usually induced by changing the model parameter values and model checking is employed to verify if the updated model conforms to the formal specification.
- **Parameter identification problems:** One of the inverse problems in systems biology (Engl et al., 2009) whose main aim is to find suitable parameter values for the model such that the behaviour of the model (approximately) matches experimental data, or the model is valid relative to a given formal specification (e.g. (Barnat et al., 2010b; Batt et al., 2007a; Brim et al., 2013a; Donaldson and Gilbert, 2008a; Islam et al., 2015; Liu et al., 2015; Mancini et al., 2015; Rizk et al., 2008)). Related concepts employed in the literature for parameter identification are parameter estimation or parameter synthesis. The difference between these concepts is defined differently by various authors. For instance Brim et al. define parameter estimation as the problem of finding parameter values such that the model behaviour (approximately) matches experimental data, and parameter synthesis as a parameter estimation approach based on model checking (Brim et al., 2013b). Conversely Zuliani states that parameter estimation problems attempt to

find a single combination of parameter values such that the model is valid, and parameter synthesis problems aim to find sets of parameter values combinations such that the model is valid (Zuliani, 2014).

- **Model construction problems:** The main aim is to modify both the parameter values and the structure of the model until the model becomes valid according to a given formal specification (e.g. (Calzone et al., 2006)).

In case of model validation problems the structure and parameter values of the model are fixed and the output of the model checker execution can be either true (i.e. the model is valid) or false (i.e. the model is invalid). Similarly for robustness computation problems the model checker output can be true/false but the parameter values of the model are modified. In case of parameter identification problems the parameter values of the model are modified but the model checking output is (eventually) expected to be true. Similarly in case of model construction problems the (eventually) expected model checking output is true but both parameter values and model structure can be changed.

For readability purposes only several of the references considered were explicitly included in this section; see Appendix B for a complete list, and the recent review papers (Brim et al., 2013b; Fisher and Piterman, 2014; Zuliani, 2014) for a more detailed description.

2.5.1 Computational modelling formalisms

Computational models of biological systems validated using model checking approaches are usually encoded using (non-)probabilistic high level modelling formalisms which can be translated to a corresponding (P)LSTS representation.

In case of non-probabilistic systems several of the most employed high level modelling formalisms are (*extended*) *Boolean networks* (Gong and Feng, 2014; Miskov-Zivanov et al., 2013), *hybrid* (Liu et al., 2014a) or (*oscillator*) *timed automata* (Bartocci et al., 2010; Siebert and Bockmayr, 2006; Van Goethem et al., 2013), *Petri nets* (Gilbert et al., 2007; Heiner et al., 2008), *piecewise multi-affine/linear (ordinary) differential equations* (Barnat et al., 2009a; Batt et al., 2007a, 2005; Monteiro et al., 2008; Yordanov and Belta, 2011), (*weighted/logic*) *regulatory graphs* (Bérenguier et al., 2013; Bernot et al., 2004; Giacobbe et al., 2015), and software-specific (e.g. BIOCHAM (Chabrier and Fages, 2003)) *rule-based modelling languages*.

Conversely for probabilistic systems some of the most employed high level modelling formalisms are *continuous/discrete time Markov chains* encoded using software-specific modelling languages (e.g. iBioSim (Madsen et al., 2012),

PRISM (Ballarini and Guerriero, 2010; Češka et al., 2014; Kwiatkowska et al., 2008)), *dynamic Bayesian networks* (Liu et al., 2012), *generalized stochastic* (Ballarini et al., 2014) or *hybrid functional* (Li et al., 2011) *Petri nets*, *stochastic hybrid automata* (David et al., 2012), software-specific (e.g. BIOCHAM (Calzone et al., 2006), BioNetGen (Clarke et al., 2008; Gong et al., 2012)) *rule-based modelling languages* and *stochastic process algebras* (e.g. Bio-PEPA (Ciocchetta et al., 2009; Guerriero, 2009)).

Therefore the high level modelling formalisms considered are either specific to non-probabilistic (e.g. ordinary differential equations) or probabilistic (e.g. continuous time Markov chains) systems, or can support both systems types (e.g. rule-based languages, process algebras, Petri nets). The main advantage of the latter is that computational models can be easily translated from the non-probabilistic to the probabilistic setting and vice versa.

2.5.2 Formal specification

The computational models are validated against formal specifications encoded using (non-)probabilistic temporal logics.

Non-probabilistic temporal logics usually employed for encoding biological systems' formal specifications are (quantifier-free (Fages and Rizk, 2009; Rizk et al., 2009)) LTL (Batt et al., 2008; Giacobbe et al., 2015) or CTL (Chabrier and Fages, 2003; Gong and Feng, 2014) and extensions thereof including numerical (Gilbert et al., 2007) and/or temporal constraints (David et al., 2012; Gong et al., 2012; Schivo et al., 2012).

Conversely some of the most employed probabilistic temporal logics for encoding biological systems' formal specifications are the probabilistic extensions of LTL and CTL, namely PLTL (Calzone et al., 2006), CSL (Heath et al., 2008; Kwiatkowska et al., 2007) and PCTL (Barbuti et al., 2012) similarly extended with numerical (Donaldson and Gilbert, 2008a) and/or temporal (Hussain et al., 2014a; Palaniappan et al., 2013) constraints.

2.5.3 Computational model checking approaches

Depending on the computational model and formal specification considered the employed model checking algorithm is exhaustive non-probabilistic (Antoniotti et al., 2003; Barnat et al., 2009b; Fages and Rizk, 2009; Monteiro et al., 2014; Siebert and Bockmayr, 2006), exhaustive probabilistic (Ballarini and Guerriero, 2010; Braz et al., 2013; Calder et al., 2006; Heath et al., 2008; Madsen et al.,

2012) or approximate probabilistic (Ballarini et al., 2014; Cavaliere et al., 2014; Clarke et al., 2008; David et al., 2012; Jha and Langmead, 2011).

Exhaustive (non-)probabilistic model checking approaches are employed whenever the state space corresponding to the computational model can be explored in reasonable time. Conversely for computational models with intractable, potentially infinite state spaces, approximate probabilistic model checking approaches are usually used.

Several of the most employed model checkers supporting exhaustive non-probabilistic, and exhaustive and approximate probabilistic model checking approaches are MARCIE (Heiner et al., 2013) and PRISM (Calder et al., 2006; Češka et al., 2014; Heath et al., 2008; Kwiatkowska et al., 2011; Lakin et al., 2012). Conversely two of the most employed model checkers for both exhaustive non-probabilistic and approximate probabilistic model checking are BIOCHAM (Calzone et al., 2006; Chabrier and Fages, 2003; Fages and Soliman, 2008; Maria et al., 2009; Rizk et al., 2009) and UPPAAL (Behrman et al., 2011; Bulychev et al., 2012; David et al., 2012; Siebert and Bockmayr, 2006; Van Goethem et al., 2013). Prominent model checkers implementing only exhaustive non-probabilistic model checking algorithms are NuSMV (Batt et al., 2008; Bérenguier et al., 2013; Cimatti et al., 2002; Fages and Soliman, 2008; Monteiro et al., 2014) and (Bio-)DiVinE (Barnat et al., 2009a, 2010a, 2013). Conversely one of the most employed model checking tools supporting exhaustive probabilistic model checking is MRMC (Katoen et al., 2011). Approximate probabilistic model checkers usually employed for validating computational models of biological systems are APMC (Calzone et al., 2006; Hérault et al., 2004), COSMOS (Ballarini et al., 2011, 2012, 2014), MC2 (Donaldson and Gilbert, 2008a,b), MIRACH (Koh et al., 2011) and PLASMA (Boyer et al., 2013; Cavaliere et al., 2014).

2.5.4 Limitations

Computational (systems) biology models validated using model checking approaches usually encode biological processes/subsystems from the (intra-)cellular level (e.g. cell cycle (Brim et al., 2013a; Chabrier and Fages, 2003; Fages and Rizk, 2009; Gong and Feng, 2014; Maria et al., 2009; Rizk et al., 2008; Van Goethem et al., 2013), gene (expression/regulatory) networks (Batt et al., 2008, 2005; Ciocchetta et al., 2009; Giacobbe et al., 2015; Yordanov and Belta, 2011), signalling pathways (Ballarini et al., 2014; Calder et al., 2006; Clarke et al., 2008; Donaldson and Gilbert, 2008a; Gilbert et al., 2007; Gong et al., 2012; Guerriero, 2009; Heath et al., 2008; Heiner et al., 2008; Kwiatkowska et al., 2007; Rizk et al., 2008))

where most experimental data is available. One common characteristic of these computational models is that the system behaviour is described as numeric values (e.g. concentrations) changing over time, without explicitly considering the system representation in space and/or across multiple levels of organization.

Consequently one of the main limitations of the corresponding model checking approaches is that they have been similarly defined only relative to how numeric values change over time. However in order to gain a systems level understanding of how biological organisms function it is essential to consider computational models of larger scale systems (e.g. multicellular populations). Such computational models additionally capture how properties of (emergent) spatial structures (e.g. area of multicellular population) change over time and/or across multiple levels of organization, which are not considered by existing non-spatial uniscale model checking approaches.

To address this limitation existing model checking methods need to be extended with two types of functions, namely functions that enable describing how properties of spatial structures change over time, and functions that enable associating both numeric and spatial state variables with specific levels of organization.

Summary

This chapter has provided a brief description of the formal method called model checking employed to validate models of reactive systems relative to formal specifications. Non-probabilistic systems were represented as labelled state transition systems (LSTS), and probabilistic systems were represented as probabilistic LSTSs (PLSTS). Two classes of temporal logics were described for encoding the formal specifications, linear time logics (e.g. LTL, BLTL and P(B)LTL) which assume a linear representation of time, and branching time logics (e.g. CTL, CTL*, PCTL, CSL) which assume a branching structure of time. Depending on the model and formal specification considered different types of model checking algorithms have been presented for both non-probabilistic and probabilistic systems, which were either exhaustive (i.e. considering the entire state space) or approximate (i.e. exploring the state space only partially). Model checking approaches specifically employed for validating computational models of biological systems were described in the end, including one of their main limitations i.e. that they only capture how numeric values (e.g. concentrations) change over time, without explicitly considering the evolution of the system in space and/or across multiple levels of organization.

CHAPTER 3

Multidimensional spatio-temporal model checking

Introduction

In this chapter a novel multidimensional spatio-temporal model checking methodology is introduced which enables validating computational models of biological systems with respect to how both numeric and spatial properties change over time. The methodology comprises a theoretical model for abstractly representing biological systems, a spatio-temporal analysis method for automatically detecting and analysing spatial structures in the model simulation output, a standard representation format for time series data comprising numeric and spatial properties, a formal language for encoding the specification against which the model is validated, and corresponding model checking algorithms. A brief description of the model checking method implementation, and a comparison with related approaches from other domains of science are provided in the end.

3.1 Spatial computational models of biological systems

Different types of computational models are employed to represent biological systems depending on the level of organization considered.

At intracellular or more fine-grained levels it is often assumed that species (e.g. proteins/molecules) are uniformly distributed in space. Therefore computational models only capture how their average concentration changes over time without explicitly taking space into account.

Conversely at cellular and more coarse-grained levels it is assumed that the heterogeneity of species (e.g. cells) is important because it can lead to the development of different structures in space. Therefore corresponding computational models usually explicitly record how the number/density of species evolves both over time and space and are called (multidimensional) spatial(-temporal) computational models.

In order to support the development of such spatial computational models appropriate modelling formalisms have been developed; they represent the spatial domain in either a continuous or discrete fashion.

Continuous spatial models are usually encoded as partial differential equations (Schaff et al., 1997) and have been used to represent variations of reaction-diffusion (Kondo and Miura, 2010) or predator-prey (Arditi et al., 2001) systems, and the chemotactic movement of cells (Hillen and Painter, 2009). The main reason for modelling processes such as diffusion (reaction-diffusion) or population variation (predator-prey, chemotaxis) using continuous approaches is that only the average density of the species is of interest for each time point and position in space.

Conversely, if the interactions between individual species are of interest discrete spatial models could be employed instead. Representative discrete spatial modelling formalisms which employ a lattice-based representation of space and local rules to specify how the system changes from one state to the next are Cellular Automata (Deutsch and Dormann, 2007, Chapters 5-11) and Glazier-Graner-Hogeweg (Balter et al., 2007; Graner and Glazier, 1992) models (also known as Cellular Potts). In contrast individual-based models (An et al., 2009; Thorne et al., 2007) can employ either an on-lattice or off-lattice spatial representation, and their evolution over time is determined by rules specific to individuals (or agents) instead of lattice positions. Modelling formalisms which are not inherently spatial but have been extended with spatial attributes recording the species' position in space (e.g. coordinates in Euclidean space) include Petri nets (Gao et al., 2013; Gilbert et al., 2013; Liu et al., 2014b; Pârvu et al., 2013), process algebras (Feng and Hillston, 2014; John et al., 2010), rule-based modelling languages (Blinov et al., 2004; Danos et al., 2007; John et al., 2011; Maus et al., 2011; Nikolić et al., 2012) and P (or membrane) systems (Barbuti et al., 2011; Besozzi et al., 2008).

Examples of biologically relevant case studies encoded using these spatial modelling formalisms include the cardiac and gastrointestinal tissue electrophysiology (Corrias et al., 2012), chemo-/photo-taxis (John et al., 2008, 2010), the growth of microbial populations (Ferrer et al., 2008; Pârvu et al., 2015), host-pathogen interactions (Bauer et al., 2009), organisms development or morphogenesis (Marée

et al., 2007; Merks and Glazier, 2005) and tumour growth (Mallet and De Pillis, 2006; Moreira and Deutsch, 2002; Norton and Popel, 2014).

Similarly to computational models of intracellular networks, spatial computational models of biological systems need to be validated before they are employed for real-life applications. However there is a lack of corresponding model checking approaches.

The main reason why existing model checking approaches cannot be employed to validate spatial computational models is that they do not consider how properties of (emergent) spatial structures change over time.

These spatial structures are not hardcoded into the models but are emergent behaviours i.e. they are dynamic behaviours that occur at simulation time as a result of the interaction between their constituent entities (e.g. cells). Therefore one of the main challenges of validating spatial computational models is to automatically detect spatial structures in the model simulation output and analyse how their properties change over time. Moreover a suitable spatio-temporal formal language needs to be defined for encoding the specifications against which the models are validated. Finally the employed model checking algorithms need to be updated accordingly.

3.2 Multidimensional spatio-temporal model checking workflow

To address these challenges a multidimensional spatio-temporal model checking methodology was developed which enables validating spatial computational models with respect to both how numeric values (e.g. concentrations) and (quantitative) properties of spatial structures change over time.

One of the main assumptions made here is that biological systems are inherently stochastic. Therefore only probabilistic models, formal logics and model checking algorithms are employed throughout. Moreover by considering both numeric and spatial properties it is expected that the size of the state space will be usually larger for spatial compared to non-spatial computational models. Due to the higher complexity inherent to spatial computational models only approximate probabilistic model checking approaches will be considered here; employing exhaustive probabilistic approaches requires replacing the approximate with exhaustive probabilistic model checking algorithms.

The main contributions of this chapter are:

- Definition of stochastic spatial discrete-event systems (SSpDES) as an

abstract representation for describing how stochastic biological systems evolve in time and space (Section 3.3).

- A formal Probabilistic Bounded Linear Spatial Temporal Logic (PBLSTL) for specifying spatio-temporal logic statements (Section 3.5).
- Implementation of the approach in the multidimensional model checking platform Mudi which enables validating spatio-temporal models against PBLSTL properties. Mudi comprises both Bayesian and frequentist, estimate and hypothesis testing based validation approaches (Section 3.7).
- Definition of a spatio-temporal analysis module for automatically detecting and analysing spatial structures and clusters of such structures in time series data. The output of this module is formatted according to the Spatial Temporal Markup Language (STML) introduced here (Section 3.4).

The general workflow for constructing and validating spatial computational models using the multidimensional spatio-temporal model checking methodology is depicted in Figure 3.1 and comprises the following steps:

1. **Model construction:** Building the computational model from biological observations and/or relevant information from the literature.
2. **Spatio-temporal detection and analysis:** The model is simulated to generate time series data in which spatial structures and clusters of such structures are automatically detected and analysed. The output of the spatio-temporal analysis is formatted according to the STML standard representation format.
3. **Formal specification:** Natural language properties representing the specification of the system are translated into formal PBLSTL statements.
4. **Model checking:** The model checker Mudi takes the spatio-temporal analysis output and the PBLSTL statements as input and decides if the model is valid or not using the validation method chosen by the user (e.g. frequentist statistical model checking). In case the model is invalid it is updated and then checked again.

3.3 Model construction

Due to their inherent probabilistic nature biological systems are usually encoded as stochastic processes which transition from one state to the next only when

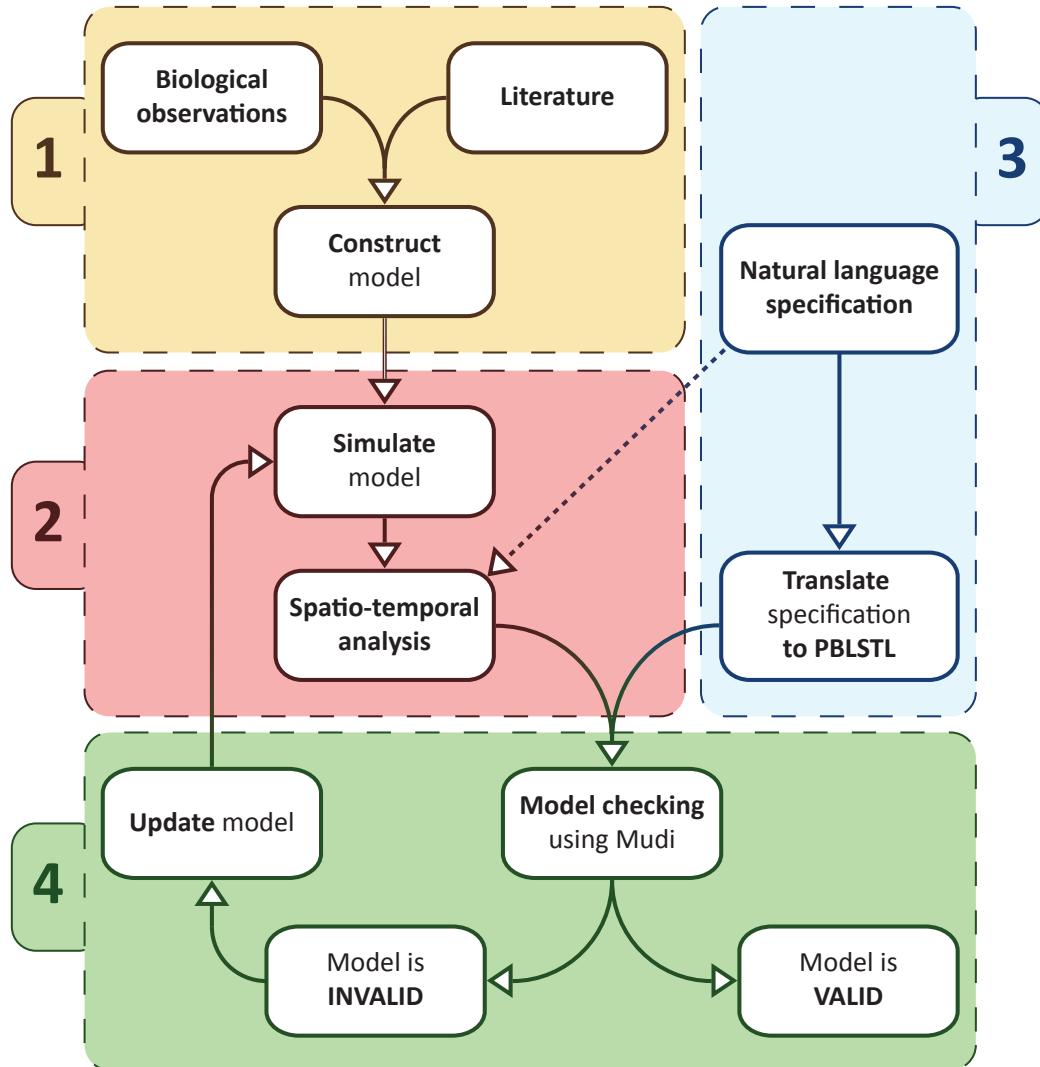


Figure 3.1: Workflow comprising all steps from construction to validation of multidimensional spatio-temporal computational models of biological systems. The first step (1) describes how the model is constructed from biological observations and/or information from the literature. In the second step (2) the model is simulated to generate time series data which is passed to the spatio-temporal analysis module. The main purpose of this module is to automatically detect and analyse how (clusters of) spatial structures and their properties change over time. The types of spatial structures and properties considered in the spatio-temporal analysis correspond to the spatial structures and properties described in the natural language specification; the dependency between the natural language specification and the spatio-temporal analysis is represented by a dashed arrow. The third step (3) comprises the manual translation of the natural language system specification to a formal PBLSTL specification. Finally the fourth step (4) describes the validation of the model with respect to the PBLSTL specification using the model checker Mudi. In case the model is invalid it is updated and steps (2) and (4) are repeated.

an event occurs (e.g. a biochemical reaction). This specific class of stochastic processes are called SDESSs (see Subsection 2.4.2.1, Definition 4).

3.3.1 Explicitly encoding space

Our aim is to reason about properties of spatial structures produced by such systems, and to quantify how these properties change over time. The following assumptions are made regarding the representation of space:

1. In the following only the discretised version of the 2D and pseudo-3D Euclidean space is considered. A pseudo-3D space extends a 2D Euclidean space with a density measure for each position. The density measure indicates the proportion of occupied positions on the Oz axis for a fixed (x, y) position. Compared to a full 3D representation it does not specify explicitly which positions of the Oz axis are occupied but only their proportion. Changing the spatial representation from pseudo-3D to 1/2/3D requires only updating the number of spatial dimensions considered. This can be done automatically using the meta model checking concept, which is introduced later in Chapter 5.
2. The 2D Euclidean space is discretised by splitting it into m rows and n columns obtaining an $m \times n$ regular grid where m and n are finite, natural, positive numbers. The resolution of the results depends on the values of m and n . Higher values guarantee a fine-grained resolution while lower values account for a coarse-grained resolution.

The evolution of an SDES in space could be represented using one/multiple collections of $m \cdot n$ state variables such that each state variable represents one discretised position in space. The main advantage of this is that the structure of SDESs does not change when adding spatial information to a model. However the main disadvantage is that semantically different state variables (e.g. concentrations, value of discretised position in space) belong to the same set without the possibility to explicitly distinguish between them at the entire set level.

In the following we would like to reason about how emergent spatial structures occupying subsets of positions in the discretised space (e.g. representing subpopulations of cells) and their properties change over time. Therefore there is a need to define detection and analysis methods which are specific to the collection of state variables encoding space, and do not apply to state variables encoding numeric values such as concentrations. For this reason the state variables encoding spatial information will be extracted in a separate set denoted as spatial state variables (*SpSV*). Moreover instead of representing space using $m \cdot n$ spatial state variables such that the value of each state variable $\in \mathbb{R}_+$, a single spatial state variable whose value $\in \mathbb{R}_+^{m \times n}$ is employed. The evaluation of such state variables to $m \times n$ real-valued non-negative matrices cannot be performed by the existing value assignment function V whose codomain is \mathbb{R} . Thus a corresponding spatial value assignment function (*SpV*) is defined.

The main advantage of explicitly distinguishing between numeric and spatial state variables is that state variable type specific functions can be defined.

Conversely the main disadvantage is that SDESs need to be extended with an additional set of spatial state variables $SpSV$ and a spatial value assignment function SpV . An alternative approach that could overcome this disadvantage is to represent all state variables as spatial state variables. The reason for not considering this alternative approach is that it introduces redundancy because state variables which were previously of numeric type and inherently encoded a single real value would be evaluated to matrices containing $m \cdot n$ copies of the real value.

3.3.2 Stochastic spatial discrete-event systems

Considering the above notations we define stochastic spatial discrete-event systems (SSpDES) as an extension of SDES with a set of spatial state variables $SpSV$ and a spatial value assignment function SpV .

Definition 5 Stochastic spatial discrete-event system (SSpDES)

An SSpDES \mathcal{M} is a 7-tuple $\langle S, T, \mu, NSV, SpSV, NV, SpV \rangle$ where:

- $\langle S, T, \mu, NSV, NV \rangle$ is a SDES (see Subsection 2.4.2.1, Definition 4);
- $SpSV$ is the set of *spatial state variables*;
- SpV is the *spatial value assignment function*.

The set $SpSV$ contains all spatial state variables i.e. the variables recording the configuration of the discretised space in the current system state. The value of these variables is computed using the spatial value assignment function SpV :

$$SpV : E \times S \times SpSV \rightarrow \mathbb{R}_+^{m \times n},$$

where E denotes the set of all possible model executions/simulations, S the set of states, $SpSV$ the set of spatial state variables, and m and n the dimensions of the discretised space. Given a model simulation σ at state s and a spatial state variable $spsv$, $SpV(\sigma, s, spsv) = sv$ such that $sv \in \mathbb{R}_+^{m \times n}$ returns a $m \times n$ matrix of real non-negative values, where each element of the matrix corresponds to a position in the discretised space. For explanatory purposes an illustrative example of a simple SSpDES is provided below.

Example 8 Illustrative example of an SSpDES encoding the growth of a population of cells

Let us assume that we would like to model the growth of a population of cells in a fixed size environment. For simplicity purposes let us consider that the environment comprises 2×2 spatial compartments where each compartment can hold at most one cell. Cells can be of two types, wild type (A) or mutant (B). The probability of obtaining a type A/B offspring cell when a parent cell of type A/B divides is 70%, respectively 30% if the parent cell is of type B/A. Since each compartment can be occupied by at most one cell, whenever a parent cell divides the offspring cell is displaced to a neighbouring compartment. Two compartments are neighbouring if the Manhattan distance between them is at most 1. Finally the cell population survival condition is that the concentration of O_2 in the environment is greater or equal to 50%. Each new type A cell reduces the O_2 concentration with 20%, and type B cell by 15%.

Although the above described scenario is not realistic for practical applications, it is sufficient to illustrate how an SSpDES model can be constructed for a biological system which evolves in time and space. The reason for strongly constraining the size of the environment, the neighbourhood relation between different compartments and the behaviour of the cells was to limit the number of possible system states such that they can all be explicitly enumerated.

The behaviour of this simple system is characterised at each moment in time by a set of state variables. Spatial state variables of interest are $Cells_A$ and $Cells_B$ representing the number of type A, respectively type B cells in the environment. Conversely the numeric state variable O_2 is used to record the concentration of O_2 in the environment. Considering these spatial and numeric state variables the initial state/configuration S_0 of the system is depicted in Figure 3.2.

State S_0					
$Cells_A$		$Cells_B$		O_2	
0	0	0	0	80%	

Figure 3.2: Initial state of SSpDES encoding the growth of a population of cells in a fixed size environment. $Cells_A$ and $Cells_B$ are the spatial state variables representing the number of type A, respectively type B cells in the environment. O_2 represents the current concentration of O_2 in the environment.

Starting from S_0 the system probabilistically transitions from one state to the next until it reaches its final configuration; see Figure 3.3 for all possible states which can be reached starting from the initial state.

Considering the initial state S_0 the system can transition to four possible states described by the following behaviours: the type A cell from the lower right corner either divides and the offspring is of the same type (S_1, S_3) or of type B

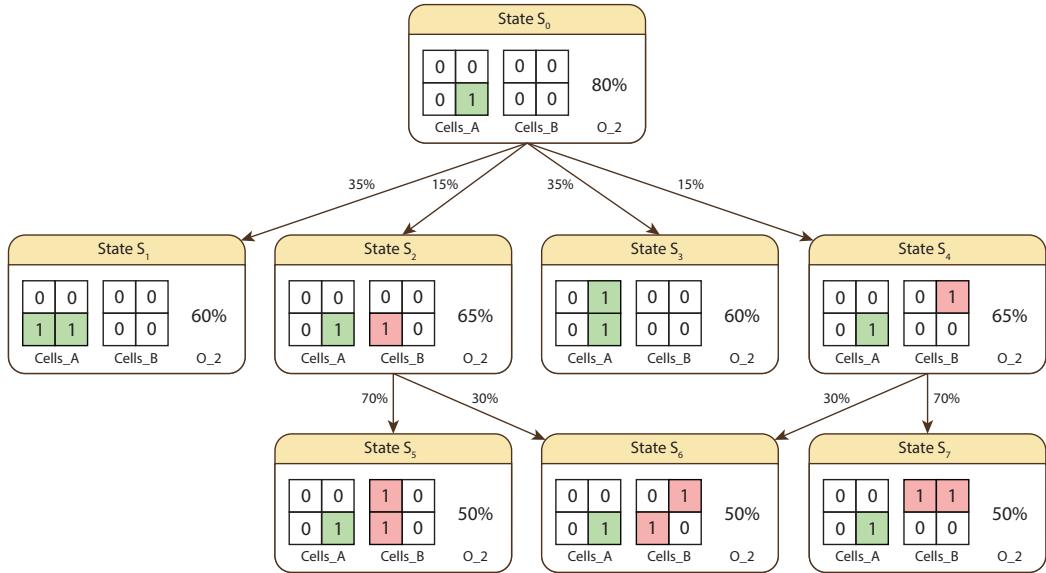


Figure 3.3: The state space of SSpDES encoding the growth of a population of cells in a fixed size environment i.e. all possible states which can be reached from the initial state S_0 . $Cells_A$ and $Cells_B$ are the spatial state variables representing the number of type A, respectively type B cells in the environment. O_2 represents the current concentration of O_2 in the environment. The percentages associated with the arrows connecting each pair of states represent the probability of transitioning between states.

(S_2 , S_4). In both cases the offspring can be either displaced above the parent (S_3 , S_4) or to its left (S_1 , S_2). Given that the overall probability of a cell to produce offspring of the same type is 70% and in our case there are 2 relevant state transitions ($S_0 \rightarrow S_1$, $S_0 \rightarrow S_3$), the probability associated with each of these state transitions is $70\% / 2 = 35\%$. Analogously the probability associated with each state transition where the offspring cell is of different type ($S_0 \rightarrow S_2$, $S_0 \rightarrow S_4$) is equal to $30\% / 2 = 15\%$. The concentration of O_2 has been decreased by 20% in states S_1 and S_3 due to a new type A cell, respectively by 15% in states S_2 and S_4 due to a new type B cell. Therefore the O_2 level is $80\% - 20\% = 60\%$ in states S_1 and S_3 , and $80\% - 15\% = 65\%$ in states S_2 and S_4 . Since the birth of a new cell reduces the O_2 concentration by at least 15%, and the minimal O_2 concentration required by the cell population to survive is 50%, no further cellular division can occur starting from states S_1 and S_3 ($60\% - 15\% < 50\%$). Conversely starting from states S_2 and S_4 at most one new type B cell can be created ($65\% - 15\% \geq 50\%$). Given state S_2 a type B cell can be produced either from the existing type A ($S_2 \rightarrow S_6$, probability 30%) or type B ($S_2 \rightarrow S_5$, probability 70%) cell. Similarly given state S_4 a type B cell can be produced either from the existing type A ($S_4 \rightarrow S_6$, probability 30%) or type B ($S_4 \rightarrow S_7$, probability 70%) cell.

Using the above descriptions the formal SSpDES $\mathcal{M} = \langle S, T, \mu, NSV, SpSV, NV, SpV \rangle$ corresponding to the system is defined as follows:

- $S = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$.

$$\bullet T = \begin{array}{c} \begin{matrix} & S_0 & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 \\ S_0 & \left(\begin{matrix} 0 & 35\% & 15\% & 35\% & 15\% & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 70\% & 30\% & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30\% & 70\% \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \right) \\ \end{matrix} \end{array}.$$

- μ is the function used to compute probabilities associated with cylinder sets $C(\sigma_{finite})$ defined over finite computation path prefixes σ_{finite} . The probability value associated with $C(\sigma_{finite})$ is computed by multiplying the probabilities of the state transitions encoded by σ_{finite} . For instance, if $\sigma_{finite} = \{S_0, S_2, S_6\}$ then $\mu(C(\sigma_{finite})) = P(S_0, S_2) \cdot P(S_2, S_6) = T[S_0, S_2] \cdot T[S_2, S_6] = 15\% \cdot 30\% = 4.5\%$.
- $NSV = \{O_2\}$, and NV is the function used to compute the value of O_2 in the current system state.
- $SpSV = \{Cells_A, Cells_B\}$, and SpV is the function used to evaluate $Cells_A$ and $Cells_B$ in the current system state.

Although only a simple example was considered here the same modelling principles are employed to construct SSpDES models of more complex (realistic) systems. One of the main differences is that due to the high complexity associated with some real systems the number of possible system states is very large, even potentially infinite. Therefore in such cases explicitly enumerating all possible paths starting from the initial state is not feasible in reasonable time.

Remark 3. The probabilities employed in Example 8 were chosen for explanatory purposes and were not derived from experimental data or the literature. ■

The size of the discretised space and the semantics of the values stored for each spatial compartment depend on the addressed biological problem. In general the granularity of the discretised spatial domain should be sufficiently fine to enable answering the biological problem of interest but not finer than that. The main reason for this is that increasing the resolution of the discretised space through

fine-graining leads to a potential increase in the size of the state space and/or model simulation time. Conversely reducing the resolution of the discretised space too much may lead to large approximation errors and/or biologically irrelevant conclusions.

Finally one of the main advantages of defining SSpDESSs as an extension of SDESs is backwards compatibility i.e. existing SDES models can be interpreted as SSpDESSs having an empty set of spatial state variables *SpSV*. Moreover SSpDESSs enable scaling up the development of computational models by extending existing non-spatial models, typical for subcellular scales (e.g. intracellular networks), with spatial information relevant to potentially higher scales (e.g. cellular/tissue level).

3.4 Spatio-temporal detection and analysis

Simulations of an SSpDES (see Definition 5) generate time series data describing how both numeric values encoded by numeric state variables, and values of the positions in the discretised space encoded by spatial state variables change over time. To reason about emergent spatial structures an automatic mechanism for detecting and analysing the corresponding subsets of positions in the discretised space is required. A spatio-temporal analysis module is developed for this purpose comprising two parameterised mechanisms; one for spatial structures denoted in the rest of the thesis as *regions*, and the other for *clusters*. Depending on the values of the detection parameters a more fine- or coarse-grained subset of the discretised space is considered.

For clarity purposes we will refer throughout to spatial structures as *spatial entities*, and to the different types of spatial structures, namely regions and clusters, as *spatial entity types*.

3.4.1 Spatial entity types

3.4.1.1 Regions

One of the main assumptions of the region detection mechanism is that subsets and not individual positions of the discretised space are considered. Secondly the value of each position in the discretised space records the number/density of entities of interest. Each position can hold 0 or more entities without pileup and the identity of the elements forming the region is not explicitly taken into account. It is assumed that the shape and size of the entities is constant throughout the entire space. Therefore the region detection mechanism operates in a homogeneous context with respect to the entities considered. If the system comprises multiple

types of entities each type should be represented by a different spatial state variable. Therefore different types of regions can be detected by repeatedly applying the region detection mechanism for each corresponding spatial state variable.

Given a model execution/simulation σ and a spatial state variable $spsv$, let us denote the i -th state of the model execution σ by $\sigma[i]$, $0 \leq i \leq |\sigma|$, where $|\sigma|$ represents the length of σ .

Definition 6 Region

A region reg with respect to $\sigma[i]$ and $spsv$ is a subset of neighbouring positions in $SpV(\sigma, \sigma[i], spsv)$ such that $\forall x \in reg, value(x) \geq \epsilon_{value}$ and $|reg| > \epsilon_{size}$, where $\epsilon_{value}, \epsilon_{size} \in \mathbb{R}$ are user-defined parameters.

Two positions in the discretised space are neighbouring if they share at least one corner/border.

The problem of finding regions is similar to the segmentation problem in the Computer Vision literature (Szeliski, 2010). 2D images can be represented as vectors or matrices where each position records the colour (multi-channel) or intensity (single channel) of the image. In order to apply Computer Vision methods for finding regions the matrix encoding the current state of the discretised spatial domain is translated to a greyscale image. The value of each position in the matrix is normalised and converted to the intensity value of the corresponding pixel in the resulting image. Examples of greyscale images in which sector-like patterns have been detected in bacterial colonies are depicted in Figure 3.4.

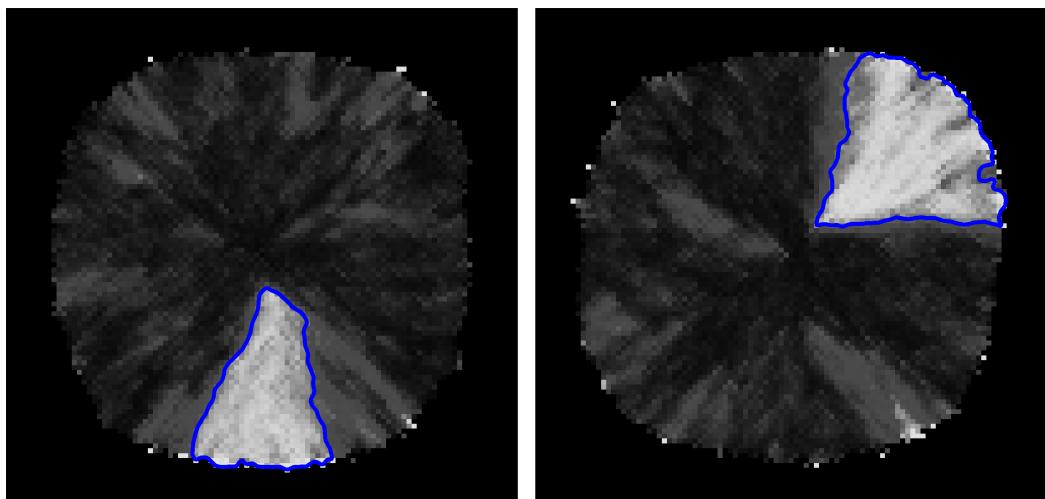


Figure 3.4: *In silico* generated greyscale images depicting bacterial colonies containing “wild type” (dark grey) and “mutant” (light grey) cells. Sector-like patterns corresponding to high-proportions of “mutant” cells are automatically detected, analysed and outlined in blue.

The parameterised mechanism for detecting regions in greyscale images is

described in Algorithm 1. All mentioned subalgorithms are implemented in the open source Computer Vision library OpenCV (Bradski and Kaehler, 2008); see Appendix C.1 for a mapping between the subalgorithms described in Algorithm 1 and the OpenCV functions. Detailed descriptions of the OpenCV function parameters are provided in the official OpenCV documentation (Itseez, 2013) and will not be restated here.

Algorithm 1 Algorithm for region detection

Require: $image$ is a greyscale image

Ensure: $regions$ defines the set of regions detected in the image

```

1: ChangeBrightnessAndContrast( $image$ ,  $alpha$ ,  $beta$ );      ▷ Adjust brightness
2:                                         and contrast
3: MorphologicalCloseOperation( $image$ ,  $morphCloseNrOfIter$ );    ▷ Connect
4:                                         discontinued but
5:                                         close regions and
6:                                         remove noise
7: GaussianBlur( $image$ ,  $kernelSize$ ,  $standardDev$ ); ▷ Remove remaining noise
8: Threshold( $image$ ,  $\epsilon_{value}$ );           ▷ Apply binary threshold method to
9:                                          $image$  considering threshold value  $\epsilon_{value}$ 
10:
11:  $contours = \text{DetectAndApproximateContours}(image, approximationLevel)$ ;
    ▷ Detect regions contours
12:
13: for all  $contour \in contours$  do
14:   if  $\text{size}(contour) < \epsilon_{size}$  then
15:     Mark the region defined by  $contour$  as noise;
16:   end if
17: end for
18:
19:  $regions = \{reg \mid reg \in contours, reg \text{ not marked as noise}\}$ ;    ▷ The set of
20:                                         regions is defined by
21:                                         the subset of contours
22:                                         not marked as noise
23:                                         with size greater or
24:                                         equal to  $\epsilon_{size}$ 
25:
26: return  $regions$ ;

```

3.4.1.2 Clusters

Given a collection of regions, the cluster detection mechanism constructs groups of sufficiently similar regions. During this procedure no assumption is made regarding the size and type of the regions. In contrast to the region detection

mechanism, the mechanism for detecting clusters operates in a heterogeneous context where both fixed and variable size subsets of the discretised space are considered.

Our assumption is that two regions are similar and should belong to the same cluster if the distance between them is below a certain threshold. A distance pseudometric $dist$ is defined for this purpose:

$$dist : REG \times REG \rightarrow \mathbb{R}, dist(A, B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2},$$

where REG is the set of all regions, and $dist(A, B)$ computes the Euclidean distance between the centroids of two regions $A, B \in REG$. The Euclidean distance measure is considered because we are interested in detecting and analysing how groups of entities that are sufficiently close to each other in space (i.e. the Euclidean distance between them is bounded above) change over time; see Section 4.3 for an illustrative case study describing how single cells chemotactically aggregate into groups, and how these groups change over time.

Definition 7 Cluster

A cluster $clust$ with respect to a set of regions REG , and a pseudometric $dist$, is a subset of regions in REG such that $\forall x, y \in clust, dist(x, y) \leq \zeta_{distance}$ and $|clust| > \zeta_{size}$, where $\zeta_{distance} \in \mathbb{R}$ and $\zeta_{size} \in \mathbb{N}$ are user-defined parameters.

The problem of grouping entities into clusters is addressed by the cluster analysis literature (Jain, 2010). A popular algorithm which considers distance (not necessarily Euclidean) as a criterion for grouping objects is DBSCAN (Ester et al., 1996). The original algorithm has a known issue because the assignment of border objects (i.e. objects between multiple clusters) to clusters depends on the order in which the set of objects is iterated. An improved version of the DBSCAN algorithm was introduced by Tran et al. (Tran et al., 2013) for addressing this issue and is employed by our cluster detection mechanism considering the pseudometric $dist$ as the distance function. Illustrative examples of greyscale images in which clusters of cells are automatically detected and analysed are depicted in Figure 3.5.

3.4.2 Spatial measures

Each detected region/cluster is characterised by the set of pseudo-3D spatial measures {clusteredness, density, area, perimeter, distance from origin, angle (degrees), triangle measure, rectangle measure, circle measure, centroid (x-coord), centroid (y-coord)}. A detailed description of the semantics specific to regions

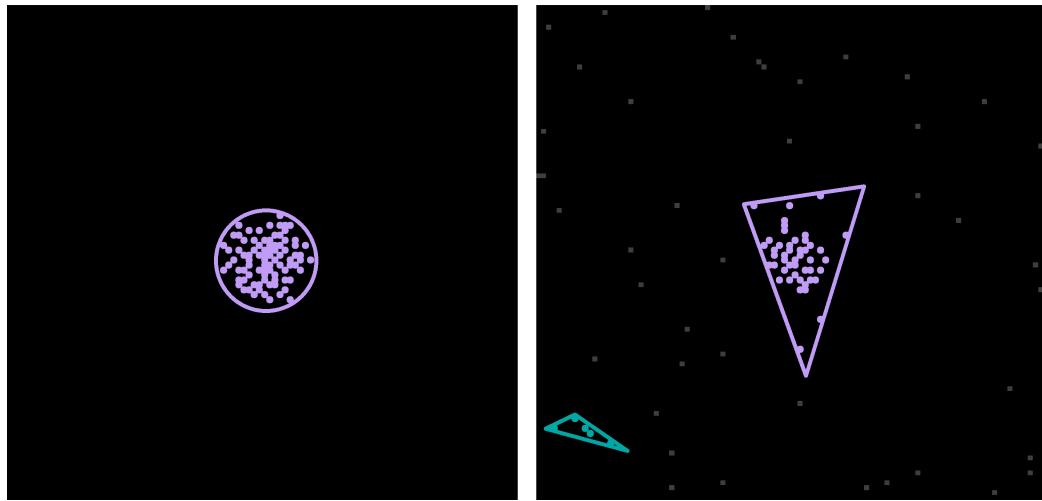


Figure 3.5: *In silico* generated greyscale images representing the distribution of cells in space. Clusters comprising at least 5 sufficiently close cells are automatically detected and outlined using different colours. Cells are represented as grey points if they do not belong to a cluster. Otherwise they are represented as coloured points such that the colour of the cell matches the colour of the cluster it is a member of. Each cluster is enclosed by a polygon whose shape (triangular, rectangular or circular) best matches the shape of the cluster.

and clusters is provided below; see Figure 3.6 for a graphical illustration.

The collection of spatial measures was defined such that it is sufficiently generic to be applied to a wide range of case studies, and sufficiently comprehensive to enable reasoning about how relevant properties of spatial structures change over time.

3.4.2.1 Computing spatial measures values for regions

The *clusteredness* of a set of regions represents the inverse of the average Euclidean distance between the centroids of the regions. Conversely the clusteredness of a single region is computed as follows:

$$\text{clusteredness}(\text{reg}) = \frac{\text{area}(\text{reg})}{\text{area}(\text{reg}) + \sum_{h \in \text{holes}} \text{area}(h)},$$

where reg is a region and holes is the set of holes contained by reg . As the area of the holes contained by regions increases the value of the clusteredness degree decreases and vice versa.

The *density* of a set of regions is equal to the average density of the regions divided by the average Euclidean distance between the centroids of the regions. Conversely the density of a single region represents the average density value of the positions defining the region in the discretised space.

The *area* of the region is equal to the area of the polygon defined by the neighbouring positions in the Euclidean plane (subtracting the area of holes).

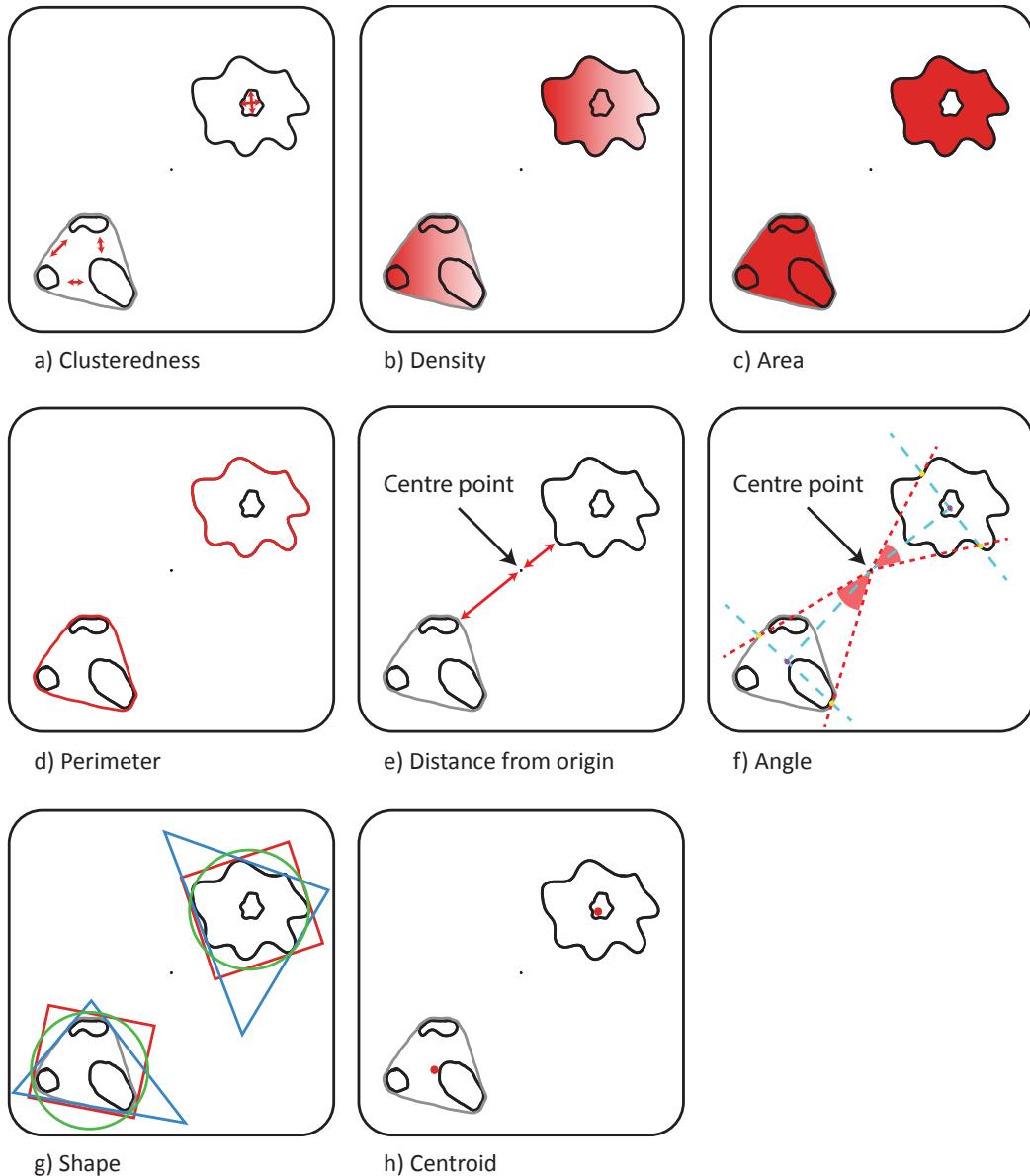


Figure 3.6: The clusteredness (a) computes how close regions/neighbouring positions are to each other in a cluster/region. Density (b) measures the average value (e.g. concentration) of the positions considered in the discretised space. Area (c) and perimeter (d) have the usual meaning from discrete 2D geometry. Distance from the origin (e) represents the minimum distance between the point from the centre of the discretised space and the considered region/cluster. The angle (f) associated to a region/cluster is determined by three points: the origin, and the points found at the intersections of the region/cluster convex hull with the line perpendicular on the line determined by the origin and the centroid of the region/cluster. The shape (g) is determined by computing the degree of similarity between the shape of the region/cluster and a triangle, rectangle and circle. The centroid (h) is the geometric centre of the considered region/cluster.

The *perimeter* of the region is equal to the perimeter of the polygon defined by the neighbouring positions in the Euclidean plane. Holes contained by the region are ignored in this case.

The *distance from the origin* is equal to the minimum distance between the polygon defined by the region and the centre point of the discretised space (origin).

The *angle (degrees)* is equal to the angle determined by the centre point of

the discretised space and the points obtained from the intersection of the line perpendicular on the line determined by the centre point of the discretised space and the centroid of the region, and the convex hull of the polygon defined by the region.

The shape of the region is determined in a fuzzy manner by the *triangular*, *rectangular* and *circular* measures. Each one of these measures computes the likelihood of the region to have a triangular, rectangular or circular shape using the following formula:

$$measure_{sh}(reg) = \frac{area(reg)}{area(\text{minimum area } sh\text{-shaped polygon enclosing } reg)},$$

where reg is a region, and the value of $measure_{sh}(reg) \in [0, 1], \forall sh \in \{\text{triangular, rectangular, circular}\}$. Algorithms for computing minimum area enclosing triangles (O'Rourke et al., 1986; Pârvu and Gilbert, 2014b), rectangles (Freeman and Shapira, 1975; Toussaint, 1983) and circles (Gärtner, 1999) which were previously published in the literature are considered here.

The *x/y-coordinates* of the centroid are computed using moments of the polygon defined by the neighbouring positions in the Euclidean plane (Steger, 1996).

3.4.2.2 Computing spatial measures values for clusters

The *clusteredness* of a set of clusters represents the inverse of the average Euclidean distance between the centroids of the clusters.

Remark 4. The *clusteredness* of a set of clusters is additionally computed using the Silhouette (Rousseeuw, 1987) cluster validity index. Although there is no index which performs best for all scenarios Silhouette obtains good/best results in the majority of cases (Arbelaitz et al., 2013). The Silhouette value is computed with respect to the regions in all clusters. Thus in our case it could be determined only at cluster detection and analysis time when the information about individual regions is available. At a particular time point we associate to a set of clusters a unique Silhouette value which means we could encode it as a numeric state variable in our model.

Conversely the clusteredness of a single cluster represents the inverse of the average Euclidean distance between the centroids of the regions in the cluster.

The *density* of a set of clusters is equal to the average density of the clusters divided by the average Euclidean distance between the centroids of the clusters. Conversely the density of a single cluster represents the average density value of

the spatial entities defining the cluster in the discretised space.

The *area* of a cluster is equal to the area of the polygon defined by the convex hull of all regions in the cluster (ignoring the holes between regions).

The *perimeter*, *distance from the origin*, *angle (degrees)*, *shape* and *x/y-coordinates* of the centroid of the cluster are determined using the same methods employed for regions. The main difference is that the polygon used to determine the outer boundary of the cluster is the convex hull computed for a group of regions instead of a single one.

The output of the spatio-temporal analysis module is time series data describing how the spatial measures values change over time for the detected spatial entities.

3.4.3 Spatial Temporal Markup Language

The output of the spatio-temporal analysis merged with time series data describing how numeric state variable values change over time represents the model simulation output. To represent this model simulation output in a uniform and consistent manner which facilitates exchange of data sets and integration of software tools a corresponding standard data representation format is required.

The main requirement for the data representation format is that it supports recording different numbers of values at different time points because the collection of (emergent) spatial structures considered could potentially change over time. Traditional tabular (e.g. csv) representation formats are not suitable because they assume that the number of recorded values is constant throughout the entire time series. Moreover defining a representation format similar to csv that does not annotate numeric values with their meaning could be potentially difficult to interpret.

For portability, structuring and readability purposes an eXtensible Markup Language (xml) based standard data representation format is defined called Spatial Temporal Markup Language (STML). The rules and constraints for the structure of the xml files are formalised in XML Schema Definition (xsd) files with the filename format STML_LxVy.xsd (i.e. Spatial Temporal Markup Language Level x, Version y); see <http://mudi.modelchecking.org/stml> for the latest version of the format. An example of an xml file recording experimental spatio-temporal data is depicted in Listing 3.1.

The results of an (*in silico/in vitro/in vivo*) experiment are recorded as a list of time points. The constraint imposed on experiment elements is that they must contain at least one time point.

Each `timepoint` element contains an optional `value` attribute which indicates

Listing 3.1: An example STML file recording spatio-temporal data

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <experiment>
3   <timepoint>
4     <spatialEntity>
5       <pseudo3D type="cluster">
6         <clusteredness>0.01</clusteredness>
7         <density>5</density>
8         <area>15</area>
9         <perimeter>28</perimeter>
10        <distanceFromOrigin>81</distanceFromOrigin>
11        <angle>10.5</angle>
12        <triangleMeasure>0.5</triangleMeasure>
13        <rectangleMeasure>1.0</rectangleMeasure>
14        <circleMeasure>0.1</circleMeasure>
15        <centroid>
16          <x>703.4999</x>
17          <y>118.087</y>
18        </centroid>
19      </pseudo3D>
20    </spatialEntity>
21    <numericStateVariable>
22      <name>avgClusterednessClusters</name>
23      <value>0.4</value>
24    </numericStateVariable>
25  </timepoint>
26  ...
27 </experiment>
```

the moment in time when the measurement was taken. If the `timepoint value` can be inferred, then it should not be defined explicitly in order to reduce the size of the STML file. For instance if the computational model considered assumes a discrete representation of time, the time difference between consecutive time points is typically constant and equal to 1. Consequently the value of each time point can be inferred to be equal to the value of the preceding time point + 1, with the exception of the first time point whose value is 0. Conversely if the computational model considered assumes a continuous representation of time, the time difference between consecutive time points is typically variable. Consequently the value of each time point needs to be defined explicitly. Therefore in general the value $value_{t_i}$ corresponding to time point t_i is computed using the following formula:

$$value_{t_i} = \begin{cases} val, & \text{if the value } val \text{ was defined explicitly for } t_i \\ 0, & \text{if no value was defined explicitly for } t_i \text{ and } i = 0 \\ value_{t_{i-1}} + 1, & \text{otherwise.} \end{cases}$$

The information stored in `timepoint` elements is a list of zero or more unique

spatial entities (i.e. `spatialEntity` elements) and/or numeric state variables (i.e. `numericStateVariable` elements).

A `spatialEntity` element currently comprises only one element called `pseudo3D` which stores a pseudo-3D spatial description of the entity. In the future if 2D or full 3D representations are of interest the `pseudo3D` element could be renamed accordingly.

Every `pseudo3D` element has an associated type which can be either *cluster* or *region*. Similarly to the detected regions/clusters every `pseudo3D` element is characterised by a set of spatial measures constrained as described below:

- *clusteredness, density, triangleMeasure, rectangleMeasure* and *circleMeasure*
- real non-negative values between 0 and 1;
- *angle* - a real non-negative value between 0 and 360;
- *area, perimeter, distanceFromOrigin, centroidX* and *centroidY* - real non-negative values.

The basic shapes considered by the current version of STML are appropriate to describe simple spatial patterns such as patches which spread outwards as they develop (triangular), ordered structures/streams (rectangular), and (uniform) groups/clusters (circular). In contrast complex patterns comprising multiple basic shapes cannot be described appropriately by the current shape similarity measures. In order to address this issue a potential future version of STML could include a more complex suite of shape descriptors.

Finally `numericStateVariable` elements contain a `name` and a `value` child element where the name is a string and the value a real number.

STML files encoding the simulation output of SSpDES models are evaluated by the model checker against formal specifications describing how both numeric and spatial properties are expected to change over time.

3.5 Formal specification

3.5.1 Bounded Linear Spatial Temporal Logic

To enable writing such specifications a corresponding formal language called Bounded Linear Spatial Temporal Logic (BLSTL) is defined. BLSTL is an extension of BLTL with spatial, arithmetic and statistical functions. The Boolean propositions specific to BLTL remain unchanged, and temporal operators F , G and U are augmented by bounded time intervals (e.g. $[0, 10]$). Moreover the

temporal operator X (i.e. next) and its variant $X[k]$ are considered which enable reasoning about the immediately next, respectively the next k -th state (Clarke et al., 2010). In addition new functions are introduced enabling to reason about how (distributions of) regions/clusters and their spatial properties change over time.

The same non-dimensional properties, spatial entities (regions and clusters) and measures (clusteredness, density, area, perimeter, distance from origin, angle (degrees), triangle measure, rectangle measure, circle measure, centroid (x-coord) and centroid (y-coord)) are considered both by the STML specification and the BLSTL formal language. Therefore BLSTL enables encoding logic statements with respect to both non-dimensional (e.g. species/proteins concentrations) and spatial properties, and correlations between the two.

To enable the construction of more complex logic statements BLSTL additionally enables specifying how arithmetic expressions comprising numeric or spatial properties change over time. The considered functions which enable the construction of complex logic statements are either unary (e.g. absolute value, round, square root etc.) or binary (e.g. addition, division, power etc.).

These arithmetic functions take a single real-valued variable as input and are directly applicable to non-dimensional properties. However in order to apply the same functions to collections of regions/clusters, the distribution of spatial measures characterising the regions/clusters has to be reduced to a single real value. A set of statistical functions is made available in the specification of BLSTL in order to address this problem. The statistical functions considered are either unary (e.g. count), binary (e.g. median with respect to a user specified spatial measure), ternary (e.g. percentile with respect to a user specified spatial measure) or quaternary (e.g. covariance between two potentially different types of spatial entities and measures). One of the main differences between BLSTL and traditional BLTL-based formal languages is that the former enables reasoning about dynamic sets of spatial entities whose cardinality changes over time, whereas the latter usually only consider static sets of numeric state variables.

Although the arithmetic and statistical functions described above enable the construction of more complex logic statements, there is a need for a mechanism which enables reasoning about particular subsets of the detected regions/clusters. For instance it may be the case that only regions with the area greater than a certain value, or clusters close to a particular point in space are of interest. In order to address this challenge BLSTL comprises a constraint-based mechanism which filters out all regions/clusters whose spatial measures do not meet a set of user-defined conditions.

3.5.1.1 Syntax

The syntax of BLSTL is defined by a context-free grammar using the Backus-Naur Form (BNF) notation. A definition of a non-terminal symbol (element) in such grammars has the following form:

$$\begin{aligned} \langle \text{defined-element} \rangle &::= \langle \text{element1} \rangle \\ &\mid \langle \text{element2} \rangle \\ &\mid \dots \end{aligned}$$

where ::= introduces a new definition and | represents an alternative. In natural language this reads $\langle \text{defined-element} \rangle$ is either an $\langle \text{element1} \rangle$ or $\langle \text{element2} \rangle$ or (...).

In contrast to the BLTL definition the symbol ϕ was replaced by the non-terminal symbol $\langle \text{logic-property} \rangle$ throughout.

Definition 8 BLSTL syntax

The syntax of BLSTL is given by the following grammar formally expressed in BNF:

$$\begin{aligned} \langle \text{logic-property} \rangle &::= \langle \text{numeric-spatial-measure} \rangle \langle \text{comparator} \rangle \langle \text{numeric-measure} \rangle \\ &\mid \langle \text{numeric-state-variable} \rangle \langle \text{comparator} \rangle \langle \text{numeric-measure} \rangle \\ &\mid d(\langle \text{numeric-measure} \rangle) \langle \text{comparator} \rangle \langle \text{numeric-measure} \rangle \\ &\mid \sim \langle \text{logic-property} \rangle \\ &\mid \langle \text{logic-property} \rangle \wedge \langle \text{logic-property} \rangle \\ &\mid \langle \text{logic-property} \rangle \vee \langle \text{logic-property} \rangle \\ &\mid \langle \text{logic-property} \rangle \Rightarrow \langle \text{logic-property} \rangle \\ &\mid \langle \text{logic-property} \rangle \Leftrightarrow \langle \text{logic-property} \rangle \\ &\mid \langle \text{logic-property} \rangle \text{ U}[\langle \text{unsigned-real-number} \rangle, \langle \text{unsigned-real-number} \rangle] \\ &\quad \langle \text{logic-property} \rangle \\ &\mid \text{F}[\langle \text{unsigned-real-number} \rangle, \langle \text{unsigned-real-number} \rangle] \langle \text{logic-property} \rangle \\ &\mid \text{G}[\langle \text{unsigned-real-number} \rangle, \langle \text{unsigned-real-number} \rangle] \langle \text{logic-property} \rangle \\ &\mid X \langle \text{logic-property} \rangle \\ &\mid X [\langle \text{natural-number} \rangle] \langle \text{logic-property} \rangle \\ &\mid (\langle \text{logic-property} \rangle) \\ \\ \langle \text{numeric-measure} \rangle &::= \langle \text{numeric-spatial-measure} \rangle \\ &\mid \langle \text{real-number} \rangle \\ &\mid \langle \text{numeric-state-variable} \rangle \\ &\mid \langle \text{unary-numeric-measure} \rangle (\langle \text{numeric-measure} \rangle) \\ &\mid \langle \text{binary-numeric-measure} \rangle (\langle \text{numeric-measure} \rangle, \langle \text{numeric-measure} \rangle) \end{aligned}$$

```

⟨numeric-spatial-measure⟩ ::= ⟨unary-subset-measure⟩(⟨subset⟩)
|   ⟨binary-subset-measure⟩(⟨subset⟩, ⟨spatial-measure⟩)
|   ⟨ternary-subset-measure⟩(⟨subset⟩, ⟨spatial-measure⟩, ⟨real-number⟩)
|   ⟨quaternary-subset-measure⟩(⟨subset⟩, ⟨spatial-measure⟩, ⟨subset⟩,
|                                ⟨spatial-measure⟩)

⟨unary-subset-measure⟩ ::= count
|   clusteredness
|   density

⟨binary-subset-measure⟩ ::= avg
|   geomean
|   harmean
|   kurt
|   max
|   median
|   min
|   mode
|   product
|   skew
|   stdev
|   sum
|   var

⟨ternary-subset-measure⟩ ::= percentile
|   quartile

⟨quaternary-subset-measure⟩ ::= covar

⟨unary-numeric-measure⟩ ::= abs
|   ceil
|   floor
|   round
|   sign
|   sqrt
|   trunc

⟨binary-numeric-measure⟩ ::= add
|   div
|   log
|   mod

```

```

|   multiply
|   power
|   subtract

⟨subset⟩ ::= ⟨subset-specific⟩
|   filter(⟨subset-specific⟩, ⟨constraint⟩)

⟨subset-specific⟩ ::= regions
|   clusters

⟨constraint⟩ ::= ⟨spatial-measure⟩ ⟨comparator⟩ ⟨filter-numeric-measure⟩
|   ~⟨constraint⟩
|   ⟨constraint⟩ ∧ ⟨constraint⟩
|   ⟨constraint⟩ ∨ ⟨constraint⟩
|   ⟨constraint⟩ ⇒ ⟨constraint⟩
|   ⟨constraint⟩ ⇔ ⟨constraint⟩
|   (⟨constraint⟩)

⟨filter-numeric-measure⟩ ::= ⟨numeric-measure⟩
|   ⟨spatial-measure⟩
|   ⟨unary-numeric-measure⟩(⟨filter-numeric-measure⟩)
|   ⟨binary-numeric-measure⟩(⟨filter-numeric-measure⟩,
    ⟨filter-numeric-measure⟩)

⟨spatial-measure⟩ ::= clusteredness
|   density
|   area
|   perimeter
|   distanceFromOrigin
|   angle
|   triangleMeasure
|   rectangleMeasure
|   circleMeasure
|   centroidX
|   centroidY

⟨real-number⟩ ::= ⟨unsigned-real-number⟩
|   ⟨sign⟩ ⟨unsigned-real-number⟩

⟨unsigned-real-number⟩ ::= ⟨fractional-part⟩
|   ⟨fractional-part⟩ ⟨exponent-part⟩

```

$\langle \text{fractional-part} \rangle ::= \langle \text{digit-sequence} \rangle . \langle \text{digit-sequence} \rangle$

- | . $\langle \text{digit-sequence} \rangle$
- | $\langle \text{digit-sequence} \rangle .$
- | $\langle \text{digit-sequence} \rangle$

$\langle \text{digit-sequence} \rangle ::= \langle \text{digit} \rangle$

- | $\langle \text{digit} \rangle \langle \text{digit-sequence} \rangle$

$\langle \text{digit} \rangle ::= 0$

- | 1
- | 2
- | 3
- | 4
- | 5
- | 6
- | 7
- | 8
- | 9

$\langle \text{natural-number} \rangle ::= \langle \text{digit-sequence} \rangle$

- | + $\langle \text{digit-sequence} \rangle$

$\langle \text{exponent-part} \rangle ::= e \langle \text{digit-sequence} \rangle$

- | E $\langle \text{digit-sequence} \rangle$
- | e $\langle \text{sign} \rangle \langle \text{digit-sequence} \rangle$
- | E $\langle \text{sign} \rangle \langle \text{digit-sequence} \rangle$

$\langle \text{sign} \rangle ::= +$

- | -

$\langle \text{comparator} \rangle ::= <$

- | <=
- | =
- | >=
- | >

$\langle \text{numeric-state-variable} \rangle ::= \langle \text{state-variable} \rangle$

$\langle \text{state-variable} \rangle ::= \{\langle \text{string} \rangle\}$

$\langle \text{string} \rangle ::= \langle \text{character} \rangle \mid \langle \text{character} \rangle \langle \text{string} \rangle$

$\langle \text{character} \rangle ::= \text{based on the Unicode character set except “{” and “}”}$

The operators' order of precedence is given by the definition of the BLSTL syntax. In the absence of parentheses the logic expressions are evaluated from left to right.

3.5.1.2 Semantics

The semantics of BLSTL is defined with respect to executions/simulations of an SSpDES \mathcal{M} . Let us assume that

$$\sigma = (s_0, t_0), (s_1, t_1), \dots$$

is an execution of \mathcal{M} along the sequence of states s_0, s_1, \dots with $t_0, t_1, \dots \in \mathbb{R}$ time durations spent in each state. Given an execution trace $\sigma = \{(s_0, t_0), (s_1, t_1), \dots\}$, a time value $t \in \mathbb{R}_+$ and a natural number i , the length of (or the number of states in) the execution trace is denoted by $|\sigma|$, the i -th state of the execution trace by $\sigma[i]$, the execution trace suffix starting at the i -th state by σ^i , the execution trace suffix starting after t time by $\sigma(t) = \sigma^i$, where $i \in \mathbb{N}$ is the minimum index such that $t \leq \sum_{j=0}^i t_j$, and the fact that the execution σ satisfies a property ϕ by $\sigma \models \phi$. For an execution σ at state s the value of a numeric state variable nsv is given by $NV(\sigma, s, nsv)$ and the value of a spatial state variable $spsv$ is given by $SpV(\sigma, s, spsv)$.

In order to have a compact and easy to follow semantics description the full symbol names provided in the BLSTL syntax definition were replaced with shorter abbreviations as described in Table 3.1.

Table 3.1: Translation of full BLSTL symbol names to abbreviated forms. The left column contains the full BLSTL symbol name. The right column contains the corresponding abbreviated form.

Full BLSTL symbol name	Abbreviated BLSTL symbol name
<logic-property>	ψ
<numeric-measure>	nm
<numeric-spatial-measure>	$nspm$
<unary-subset-measure>	usm
<binary-subset-measure>	bsm
<ternary-subset-measure>	tsm
<quaternary-subset-measure>	qsm
<unary-numeric-measure>	unm
<binary-numeric-measure>	bnm
<spatial-measure>	sm
<subset>	ss
<filter-numeric-measure>	fnm
<comparator>	\asymp
<real-number>	re
<numeric-state-variable>	nsv
<spatial-state-variable>	$spsv$

Definition 9 BLSTL semantics

Let $\mathcal{M} = \langle S, T, \mu, NSV, SpSV, NV, SpV \rangle$ be an SSpDES and σ an execution of \mathcal{M} . The semantics of BLSTL for σ is defined as follows:

- $\sigma \models nspm \asymp nm$ if and only if $nspm \asymp nm$, where $nspm$ and $nm \in \mathbb{R}$, and $\asymp \in \{<, \leq, =, \geq, >\}$;
- $\sigma \models nsv \asymp nm$ if and only if $NV(\sigma, \sigma[0], nsv) \asymp nm$, where $nsv \in NSV$, $nm \in \mathbb{R}$, and $\asymp \in \{<, \leq, =, \geq, >\}$;
- $\sigma \models d(nm_1) \asymp nm_2$ if and only if $|\sigma| > 1$ and $d(nm_1) \asymp nm_2$, where $d(nm_1)$ and $nm_2 \in \mathbb{R}$, and $\asymp \in \{<, \leq, =, \geq, >\}$;
- $\sigma \models \sim \psi$ if and only if $\sigma \not\models \psi$;
- $\sigma \models \psi_1 \wedge \psi_2$ if and only if $\sigma \models \psi_1$ and $\sigma \models \psi_2$;
- $\sigma \models \psi_1 \vee \psi_2$ if and only if $\sigma \models \psi_1$ or $\sigma \models \psi_2$;
- $\sigma \models \psi_1 \Rightarrow \psi_2$ if and only if $\sigma \models \sim \psi_1$ or $\sigma \models \psi_2$;
- $\sigma \models \psi_1 \Leftrightarrow \psi_2$ if and only if $\sigma \models \psi_1 \Rightarrow \psi_2$ and $\sigma \models \psi_2 \Rightarrow \psi_1$;
- $\sigma \models \psi_1 U[a, b] \psi_2$ if and only if there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi_2$, and for all $j, j \in [a, i)$, it holds that $\sigma(j) \models \psi_1$ with $a, b \in \mathbb{R}_+$;
- $\sigma \models F[a, b] \psi$ if and only if there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi$ with $a, b \in \mathbb{R}_+$;
- $\sigma \models G[a, b] \psi$ if and only if for all $i, i \in [a, b]$, it holds that $\sigma(i) \models \psi$ with $a, b \in \mathbb{R}_+$;
- $\sigma \models X\psi$ if and only if $|\sigma| > 1$ and $\sigma^1 \models \psi$;
- $\sigma \models X[k] \psi$ if and only if $|\sigma| > k$ and $\sigma^k \models \psi$;
- $\sigma \models (\psi)$ if and only if $\sigma \models \psi$.

The nm symbol represents the category of real-valued numeric measures. Considering a given model execution σ , nm is evaluated according to one of the definitions described below:

- **Numeric spatial measure:** $nm = nspm$;
- **Real number:** $nm = re \in \mathbb{R}$;

- **Numeric state variable:** $nm = NV(\sigma, \sigma[0], nsv)$, where nsv is a numeric state variable;
- **Unary numeric measure:** $nm = unm(nm')$, where nm' is a numeric measure;
- **Binary numeric measure:** $nm = bnm(nm', nm'')$, where nm' and nm'' are numeric measures.

The values of the unary (unm) and binary (bnm) numeric measures are computed as described in Appendix C.2 (Tables C.2 and C.3).

The $nspm$ symbol represents the category of numeric (real-valued) spatial measures. Considering a given execution σ , $nspm$ is evaluated according to one of the definitions described below:

- **Unary subset measure:** $nspm = usm(ss)$, where ss is a subset of the considered spatial entities (clusters or regions);
- **Binary subset measure:** $nspm = bsm(ss, sm)$, where ss is a subset of the considered spatial entities (clusters or regions) and sm is a spatial measure;
- **Ternary subset measure:** $nspm = tsm(ss, sm, re)$, where ss is a subset of the considered spatial entities (clusters or regions), sm is a spatial measure and re is a real value;
- **Quaternary subset measure:** $nspm = qsm(ss, sm, ss', sm')$, where ss and ss' are subsets of the considered spatial entities (clusters or regions), and sm and sm' are spatial measures.

If the considered subset of spatial entities is empty the numeric spatial measures are evaluated to zero.

Spatial measures sm are defined over the set {clusteredness, density, area, perimeter, distanceFromOrigin, angle, triangleMeasure, rectangleMeasure, circleMeasure, centroidX, centroidY} which is identical to the set of spatial measures recorded in an STML file for each detected region/cluster.

The value of unary (usm), binary (bsm), ternary (tsm) and quaternary (qsm) subset measures are computed as described in Appendix C.3 (Tables C.4, C.5, C.6 and C.7). Some of the binary and all ternary and quaternary subset measures are statistical functions which can be employed for reasoning about the distribution of the regions/clusters measures at a particular time point. In contrast to

traditional logic formalisms BLSTL allows specifying properties of both single spatial properties and/or distributions of spatial properties.

Subsets of the collections of regions/clusters are represented by the ss symbol. Considering a given execution σ , ss is evaluated according to one of the definitions described below:

- **Specific subset:** $ss = specificSubset$, where $specificSubset$ represents either the collection of all clusters (see Definition 7) or the collection of all regions (see Definition 6) corresponding to $\sigma[0]$;
- **Filtered specific subset:** $ss = filter(specificSubset, constraints)$, where $specificSubset$ has the semantics defined above, and $constraints$ is a set of logic properties restricting the spatial entities considered to a subset of $specificSubset$.

Given an execution σ the value of the $specificSubset$ symbol is computed using one of the definitions described below:

- **Regions:** $specificSubset = \bigcup_{spsv \in SpSV} \{reg \mid reg \in \text{regionDetectionMechanism}(spsv)\}$ considering the state $\sigma[0]$;
- **Clusters:** $specificSubset = \text{clustersDetectionMechanism}(se)$, where $se = \bigcup_{spsv \in SpSV} \{reg \mid reg \in \text{regionDetectionMechanism}(spsv)\}$ considering the state $\sigma[0]$.

Subsets of the collection returned by $specificSubset$ can be computed using the $filter$ predicate. Considering an execution σ $filter$ is evaluated using the definition described below:

$$filter = \{e \in specificSubset \mid e \models c, \forall c \in constraints\}.$$

The semantics of the constraint satisfaction problem considering a region/cluster e and a constraint c is defined below:

- $e \models sm \asymp fnm$ **if and only if** $sm(e) \asymp fnm$, where $sm(e)$ evaluates the spatial measure sm for the given spatial entity e , $sm \in \{\text{clusteredness}, \text{density}, \text{area}, \text{perimeter}, \text{distanceFromOrigin}, \text{angle}, \text{triangleMeasure}, \text{rectangleMeasure}, \text{circleMeasure}, \text{centroidX}, \text{centroidY}\}$, fnm is a filter numeric measure $\in \mathbb{R}$, and $\asymp \in \{<, \leq, =, \geq, >\}$;
- $e \models \sim c$ **if and only if** $e \not\models c$;

- $e \models c_1 \wedge c_2$ if and only if $e \models c_1$ and $e \models c_2$;
- $e \models c_1 \vee c_2$ if and only if $e \models c_1$ or $e \models c_2$;
- $e \models c_1 \Rightarrow c_2$ if and only if $e \models \sim c_1$ or $e \models c_2$;
- $e \models c_1 \Leftrightarrow c_2$ if and only if $e \models c_1 \Rightarrow c_2$ and $e \models c_2 \Rightarrow c_1$.

The f_{nm} symbol represents the (real-valued) numeric measure computed for the *filter* predicate. Given an execution σ and a region/cluster e , the value of f_{nm} is computed using one of the definitions given below:

- **Numeric measure:** $f_{nm} = nm$, where nm is a numeric measure;
- **Spatial measure:** $f_{nm} = sm(e)$, where $sm \in \{\text{clusteredness}, \text{density}, \text{area}, \text{perimeter}, \text{distanceFromOrigin}, \text{angle}, \text{triangleMeasure}, \text{rectangleMeasure}, \text{circleMeasure}, \text{centroidX}, \text{centroidY}\}$;
- **Unary filter numeric measure:** $f_{nm} = unm(f_{nm}')$, where unm is a unary numeric measure, and f_{nm}' is a filter numeric measure;
- **Binary filter numeric measure:** $f_{nm} = bnm(f_{nm}', f_{nm}'')$, where bnm is a binary numeric measure, respectively f_{nm}' and f_{nm}'' are filter numeric measures.

The d symbol stands for derivative. Considering a given execution σ , such that $|\sigma| > 1$, and a numeric measure nm , the value of $d(nm)$ is computed as follows:

$$d(nm) = \frac{nm^1 - nm^0}{time^1 - time^0} ,$$

where nm^i represents the result of evaluating nm against σ^i , and $time^i$ represents the value of the first time point in σ^i .

3.5.1.3 Illustrative examples of BLSTL statements

Illustrative examples of natural language statements which can be encoded in BLSTL are defined below:

- **Natural language:** At some point in the future, considering the time interval $[0, 100]$, the concentration of *cAMP* is less than 20, and the number of cell clusters emerging in the environment is greater than zero.

BLSTL: $F[0, 100] ((\{cAMP\} < 20) \wedge (count(clusters) > 0))$.

A detailed description of the mapping between natural and formal BLSTL constructs considering the previous statement is given below.

Natural language	BLSTL
“At some point in the future”	F
“considering the time interval [0, 100]”	$[0, 100]$
“the concentration of $cAMP$ is less than 20”	$(\{cAMP\} < 20)$
“and”	\wedge
“the number of cell clusters emerging in the environment is greater than zero”	$(count(clusters) > 0)$

The mapping is done in a similar manner for the next statements and therefore will not be stated explicitly.

- **Natural language:** The mean area of all cancerous regions grows throughout the entire simulation interval [5, 25].

BLSTL: $G[5, 25] (d(mean(regions, area)) > 0)$.

Remark 5. If the computational model considered is stochastic the evolution of state variables values (e.g. area of cancerous regions) over time is typically characterized by fluctuations or noise. Due to such fluctuations state variables values do not constantly increase or decrease between consecutive time points but alternate between the two, although the general trend considering multiple time points is decreasing/increasing. Consequently logic statements which specify that a state variable value always increases over a particular time interval (e.g. [5, 25]) evaluate to false. To specify that state variables values are expected to increase over multiple time points one potential solution is to state that the values are within certain bounds in particular time subintervals (e.g. area of cancerous regions is between 22.3 and 22.9 in time subinterval [5, 10], between 22.8 and 30.1 in time subinterval [10, 20], and between 30.0 and 40.0 in time subinterval [20, 25]).

- **Natural language:** Within the time interval [0, 300] the number of mutant cell populations emerging at a distance smaller than 10 from the area of inflammation (origin) is greater than 0 until the concentration of X drops below 5.

BLSTL: $(count(filter(clusters, distanceFromOrigin < 10)) > 0) U[0, 300] (\{X\} < 5)$.

To enable specifying the probability with which a formal BLSTL statement is expected to hold a probabilistic extension of BLSTL called Probabilistic BLSTL (PBLSTL) is defined.

3.5.2 Probabilistic Bounded Linear Spatial Temporal Logic

Definition 10 Probabilistic Bounded Linear Spatial Temporal Logic (PBLSTL)

A Probabilistic Bounded Linear Spatial Temporal Logic property ϕ is a logic property of the form $P_{\bowtie\theta}[\psi]$ where $\bowtie \in \{<, \leq, \geq, >\}$, $\theta \in (0, 1)$ and ψ is a BLSTL property.

An illustrative example of a natural language probabilistic statement mapped into PBLSTL is defined below:

Natural language: The probability is greater than 0.7 that at some point in the future, considering the time interval [0, 100], the concentration of *cAMP* is less than 20, and the number of cell clusters emerging in the environment is greater than zero.

PBLSTL: $P > 0.7 [F[0, 100] ((\{cAMP\} < 20) \wedge (count(clusters) > 0))]$.

Remark 6. The probability employed in the immediately above example was chosen for illustrative purposes and was not derived from experimental data or the literature.

A PBLSTL property $\phi \equiv P_{\bowtie\theta}[\psi]$ holds for an SSpDES \mathcal{M} (i.e. $\mathcal{M} \models P_{\bowtie\theta}[\psi]$) if and only if the probability of ψ to hold for an execution of \mathcal{M} is $\bowtie \theta$. Therefore in order to determine the truth value of a PBLSTL property ϕ the likelihood of ψ being true is computed.

As in the case of Jha et al. (Jha et al., 2009a) evaluating the truth value of a PBLSTL property ϕ is harder than determining the truth value of a BLSTL property ψ . One counterexample for a BLSTL property is sufficient to decide that the property does not hold. Conversely one counterexample for a PBLSTL property ϕ does not necessarily imply that ϕ is not satisfied. A PBLSTL property ϕ does not hold if the likelihood of all counterexamples provides sufficient evidence to invalidate ϕ .

3.6 Model checking

Definition 11 Multidimensional spatio-temporal model checking problem

The multidimensional spatio-temporal model checking problem is to automatically verify if an SSpDES \mathcal{M} satisfies a PBLSTL property $\phi \equiv P_{\bowtie\theta}[\psi]$.

Different approximate probabilistic model checking algorithms can be employed depending on the method of constraining the approximation error and the approach for deciding if a logic property holds. For flexibility and completeness purposes both Bayesian and frequentist, estimate and hypothesis testing based model checking methods are considered; see Subsection 2.4.2, Table 2.1 for references to the corresponding algorithms.

All methods except probabilistic black-box take a user-defined (set of) parameter(s) as input representing the maximum tolerated approximation error. Such methods are employed to evaluate a variable number of model simulations until a result can be provided considering the user-defined approximation error constraints. Conversely, the probabilistic black-box model checking approach decides based on a fixed number of model simulations if the logic property is satisfied. However in this case the confidence measure of the provided result is not specified by the user and varies depending on the number of available model simulations.

Bayesian approaches should be used when information about the prior probability distribution of parameters in the model is available. This could lead to a reduced number of required samples in order to decide if a logic property holds. Conversely if no prior knowledge is available frequentist methods could be employed instead.

Statistical hypothesis test based approaches should be employed whenever deciding between two hypotheses where usually the null hypothesis represents the PBLSTL logic property ϕ , and the alternative hypothesis $\sim \phi$. Conversely if the true probability of ϕ being true is computed and then compared to θ estimate based methods should be considered.

The algorithms provided in the original papers describing the model checking methods (see Subsection 2.4.2, Table 2.1) were employed for all approaches except frequentist statistical. An improved version of this model checking method requiring less input parameters was introduced by Koh et al. (Koh et al., 2012). However the initialisation step of the improved algorithm could potentially lead to invalid arithmetic expressions if extra conditions are not added to the algorithm implementation (C.H. Koh, personal communication, 2nd June, 2014). Therefore a variant of the improved algorithm is proposed which has a modified initialisation step that no longer requires adding extra conditions to the implementation. A more detailed description of the proposed solution is given in Appendix C.4.

3.6.1 Proof that the multidimensional model checking problem is well-defined

To show that the model checking problem is well-defined we will first prove that the number of required simulations and state transitions within each simulation are finite.

3.6.1.1 Finite number of required simulations

Probabilistic black-box model checking is the only approach considered which can provide an answer regardless of the number of available model simulations. Conversely all other methods considered require a minimum number of model simulations, which can be computed at the beginning (Chernoff-Hoeffding bounds) or not (frequentist and Bayesian statistical, Bayesian mean and variance estimate), to provide an answer considering a user-defined confidence level. Although all model checking methods require a finite number of model simulations the expected time required for an answer to be provided varies with the value of the true probability p and the user-defined probability θ . However for practical applications users might want to set an upper bound on the time to wait until an answer is provided. Thus we employ the wrapper Algorithm 2 to execute each specific model checking algorithm in Subsection 2.4.2, Table 2.1. If an answer can be provided using the requested approach within the specified extra evaluation time interval then it is reported to the user. Otherwise probabilistic black-box model checking is employed to report the answer based on the model simulations generated and evaluated so far.

In the initialisation step of Algorithm 2 `nrOfTimeoutSeconds`, the number of seconds to wait between re-executing the extra evaluation program is fixed. The reason for introducing such a variable is to temporarily wait and allow the model simulator to finish its execution before verifying if new simulations were provided. Afterwards the collection of valid model simulations is initialised based on the given `simulationsInputSet`. The model checker of type `modelCheckingType` is then executed to verify if the logic property `logicProperty` holds considering the available simulations and set of `modelCheckingParameters`. While the number of elapsed minutes is less than `extraEvaluationTime` and the number of available model simulations is insufficient to evaluate `logicProperty` the loop comprising the following steps is executed:

1. Run `extraEvaluationProgram` to generate new simulations;

Algorithm 2 The wrapper algorithm employed to call specific model checking algorithms (see Sub-section 2.4.2, Table 2.1 for the considered approaches). If sufficient model simulations are available, or generated and evaluated within *extraEvaluationTime* minutes, then the chosen specific model checking algorithm is used to provide an answer. Otherwise the user is informed that the maximum extra evaluation time threshold was reached and the answer is provided using the probabilistic black-box model checking approach. Model simulations are generated and stored in an input set *simulationsInputSet* using the external model simulation program *extraEvaluationProgram*. The logic property to be verified is stored in the variable *logicProperty*.

Require: *modelCheckingType* is the specific model checking approach, *modelCheckingParameters* is the collection of parameters required by the chosen *modelCheckingType*, *extraEvaluationTime* is the maximum number of minutes allowed for generating and evaluating additional model simulations, *extraEvaluationProgram* is the model simulation program which is called whenever new simulations are required, *simulationsInputSet* is the set containing the simulations and *logicProperty* is the PBLSTL logic property to be verified

Ensure: A true/false answer together with a measure of confidence is provided

```

1: nrOfTimeoutSeconds ← 30;           ▷ The default number of seconds to wait
2:                                         between re-executing the extra
3:                                         evaluation program and evaluating
4:                                         the generated traces
5:
6: simulations ← GetSimulations(simulationsInputSet);
7:
8: RunModelChecker(modelCheckingType, modelCheckingParameters,
9:                  simulations, logicProperty, result, confidence);
10:
11: while (elapsed number of minutes < extraEvaluationTime) AND
12:      (more model simulations are required) do
13:      GenerateModelSimulations(extraEvaluationProgram);
14:      Wait(nrOfTimeoutSeconds);
15:      UpdateCollectionOfSimulations(simulations, simulationsInputSet);
16:      RunModelChecker(modelCheckingType, modelCheckingParameters,
17:                      simulations, logicProperty, result, confidence);
18: end while
19:
20: if more model simulations are required then
21:     RunProbBlackBoxModelChecker(simulations, logicProperty,
22:                               result, confidence);
23: end if
24:
25: Output result and confidence;

```

2. Wait for *nrOfTimeoutSeconds* to give the extra evaluation program enough time to output results;
3. The collection of *simulations* is updated considering valid and previously

- unevaluated simulation input files;
4. The `modelCheckingType` model checker execution is resumed considering the additional simulations.

The loop is exited when either `extraEvaluationTime` minutes elapsed or enough model simulations have been provided. In the former case the probabilistic black-box model checker is executed to provide a `result`. Otherwise the `result` is computed using the `modelCheckingType` model checker. In the end both `result` and `confidence` measure are reported to the user.

The main advantages of Algorithm 2 are:

- The model checking execution time and number of generated and evaluated simulations is finite. Depending on the parameters of the model checker, the distribution of the data and the number of required simulations the answer will be provided using the desired model checker type or the default probabilistic black-box model checker.
- In contrast to traditional model checking methods in our approach the model checking task is decoupled from a specific model and model simulation environment (e.g. Matlab (Palaniappan et al., 2013)). An external program which can generate simulations is provided as input to the model checker. Whenever additional model simulations are required this external program is executed. For the algorithm implementation our recommendation is that the external program employed should be a script (e.g. Bash [UNIX], Batch [Windows]) which calls the model simulator and stores the output into the specified location.

3.6.1.2 Finite number of state transitions

Logic properties are evaluated with respect to simulations of computational models. For deciding if the logic property is satisfied, the model simulation must cover a sufficiently long time frame. Stopping the simulation early could potentially render the evaluation of temporal logic properties undecidable. Therefore there is a need for a mechanism to decide when a simulation execution can be stopped.

When verifying BLSTL logic properties an upper bound can be placed on the required simulation time because all temporal logic operators are bounded. Let us denote the upper bound corresponding to a BLSTL logic property ψ by $\lceil \psi \rceil$.

Definition 12 Model simulation time upper bound for BLSTL logic statement

The upper bound $\lceil \psi \rceil \in \mathbb{R}_+$ corresponding to a BLSTL logic property ψ considering an execution σ is defined recursively on the structure of the logic property as follows:

- $\lceil nspm \asymp nm \rceil = 0$ because the value of $nspm$ and nm is computed considering only $\sigma[0]$;
- $\lceil nsv \asymp nm \rceil = 0$ because the value of nsv and nm is computed considering only $\sigma[0]$;
- $\lceil d(nm_1) \asymp nm_2 \rceil = 1$ because the value of nm_1 is computed considering both $\sigma[0]$ and $\sigma[1]$;
- $\lceil \sim \psi \rceil = \lceil \psi \rceil$;
- $\lceil \psi_1 \wedge \psi_2 \rceil = \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil \psi_1 \vee \psi_2 \rceil = \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil \psi_1 \Rightarrow \psi_2 \rceil = \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil \psi_1 \Leftrightarrow \psi_2 \rceil = \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil \psi_1 U[a, b] \psi_2 \rceil = \max(b - 1 + \lceil \psi_1 \rceil, b + \lceil \psi_2 \rceil) \leq b + \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil F[a, b] \psi \rceil = b + \lceil \psi \rceil$;
- $\lceil G[a, b] \psi \rceil = b + \lceil \psi \rceil$;
- $\lceil X\psi \rceil = 1 + \lceil \psi \rceil$;
- $\lceil X[k] \psi \rceil = k + \lceil \psi \rceil$;
- $\lceil (\psi) \rceil = \lceil \psi \rceil$.

Thus the minimum upper bound for the simulation time interval to be covered by model executions when verifying a BLSTL logic property ψ is $\lceil \psi \rceil$.

Lemma 1 **BLSTL semantics based on finite prefix of infinite execution**
 Let us assume that a BLSTL logic property ψ is verified against an infinite execution $\sigma = \{(s_0, t_0), (s_1, t_1), (s_2, t_2), \dots\}$. Moreover let us denote a finite prefix of σ by $\hat{\sigma} = \{(\hat{s}_0, \hat{t}_0), (\hat{s}_1, \hat{t}_1), \dots, (\hat{s}_m, \hat{t}_m)\}$, where

$$\hat{s}_i = s_i \text{ and } \hat{t}_i = t_i, \forall i = \overline{0, m} \text{ with } \sum_{i=0}^m t_i \geq \lceil \psi \rceil \text{ and } \sum_{i=0}^{m-1} t_i < \lceil \psi \rceil.$$

Then $\sigma \models \psi$ if and only if $\hat{\sigma} \models \psi$.

Proof 1 BLSTL semantics based on finite prefix of infinite execution (proof sketch)

As per Definition 12 a BLSTL statement ψ is evaluated against a model execution σ considering only the time interval $[0, \lceil \psi \rceil]$. Following on from the assumptions of Lemma 1 the states and time points associated with σ and $\hat{\sigma}$ are equivalent over the time interval $[0, \lceil \psi \rceil]$. Hence $\sigma \models \psi$ if and only if $\hat{\sigma} \models \psi$; see Appendix C.5 for a complete version of this proof defined recursively on the structure of the logic statement ψ . \square

Lemma 2 Finite number of state transitions to evaluate BLSTL logic statement

The number of state transitions required to verify a BLSTL logic property is finite.

Proof 2 Finite number of state transitions to evaluate BLSTL logic statement

From Lemma 1 it follows that a BLSTL logic property ψ can be verified against a model simulation σ based on a finite prefix $\hat{\sigma}$. The minimum time interval captured by $\hat{\sigma}$ is bounded and can be computed using Definition 12. Since we assume the time divergence property (see Subsubsection 2.4.2.1) holds for all the systems considered, only a finite number of state transitions can occur in a bounded interval of time. \square

3.6.1.3 Well-defined model checking problem

Theorem 1 Well-defined multidimensional spatio-temporal model checking problem

The multidimensional spatio-temporal model checking problem is well-defined.

Proof 3 Well-defined multidimensional spatio-temporal model checking problem

It was shown that the number of required model executions in order to verify if a PBLSTL logic property ϕ holds is finite. Moreover considering Lemmas 1 and 2 only a finite prefix and a finite number of state transitions has to be considered for each model execution. Thus the evaluation of ϕ is reduced to the problem of evaluating non-temporal properties over a finite number of states for each model execution. This implies evaluating arithmetic expressions and/or detecting spatial

entities which are both decidable problems. Hence the model checking problem is well-defined. \square

3.7 Implementation

To automate the computational model validation process the spatio-temporal detection and analysis approaches and the multidimensional spatio-temporal model checking method were implemented in software tools. The name of the model checking software tool is Mudi, and it is composed from the uppercase letters in the word “MULTIDIMENSIONAL”. Using the model checker Mudi and the spatio-temporal detection and analysis modules a computational model can be validated relative to a PBLSTL specification as described in Figure 3.7. In contrast to Figure 3.1, Figure 3.7 focusses on the implementation specific rather than conceptual details of the multidimensional spatio-temporal model checking approach.

Given a computational model and a corresponding model simulator, the computational model is simulated to generate time series data. If the resulting time series data is encoded in a format different from csv then it is converted to csv. The csv formatted time series is then split into two time subseries, one recording the values of numeric state variables, and the other recording the values of spatial state variables. The former is converted to STML. Conversely the latter is provided as input to the spatio-temporal detection and analysis modules which detect and analyse how regions and/or clusters change over time. The output of the spatio-temporal detection and analysis modules is merged with the STML formatted time subseries recording how numeric state variables values change over time and is stored in an STML file.

The workflow steps employed to generate STML files (i.e. model simulation, conversion of time series data to the csv format, spatio-temporal detection and analysis) are integrated and executed using a script (in our case a Bash script). For reproducibility purposes the specification of the model simulation and analysis could be encoded using a standardized representation format in the future (e.g. SESSL (Ewald and Uhrmacher, 2014)).

The STML files and a PBLSTL specification are provided as input to the model checker Mudi which evaluates the specification against the STML files to determine the correctness of the corresponding computational model. The number of STML files required to determine the model correctness depends on the model checking algorithm considered; Mudi supports all approximate probabilistic model checking algorithms given in Table 2.1. If the number of STML files available is

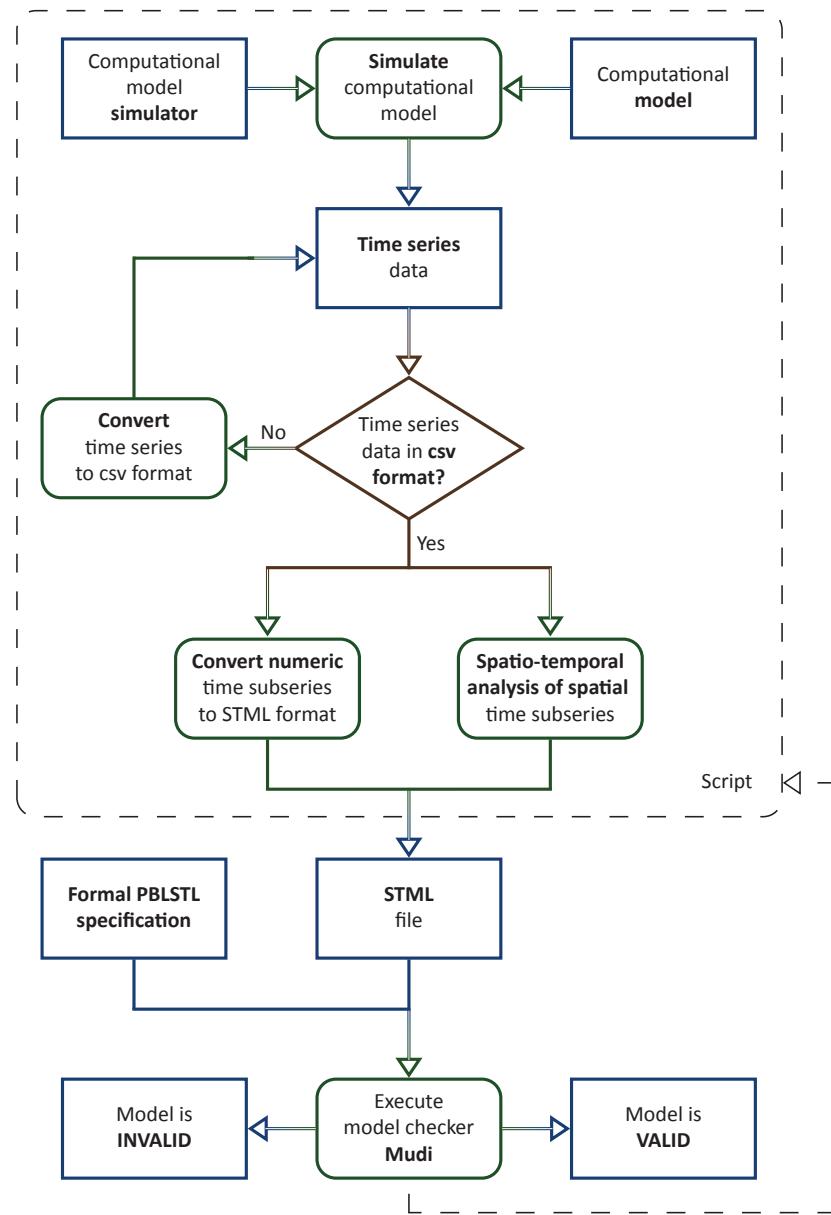


Figure 3.7: Workflow for the validation of computational models relative to PBLSTL specifications encoding how both numeric and spatial properties are expected to change over time using the model checker Mudi and the spatio-temporal detection and analysis modules. Given a computational model and a corresponding computational model simulator the model is simulated to generate time series data. If the resulting time series data is not csv formatted then it is converted to csv. Next the time subseries encoding the evolution of spatial state variables over time are provided as input to the spatio-temporal detection and analysis modules, whereas the time subseries encoding the evolution of numeric state variables over time are converted to STML. The STML formatted numeric time subseries are then merged with the output of the spatio-temporal detection and analysis modules, which describes how the detected spatial entities change over time. The resulting time series is stored into an STML file. The workflow steps employed to generate STML files (i.e. model simulation, conversion of time series data to csv format and spatio-temporal detection and analysis) are executed using a script. The model checker Mudi takes the formal PBLSTL specification and STML files as input, and evaluates the specification against sufficiently many STML files to decide if the model is valid relative to the specification. Depending on the specific model checking algorithm considered the number of STML files required during the model checker execution differs. However Mudi can take the path to the STML file generation script as input and execute the script on demand.

not sufficient Mudi can generate new STML files on demand if the path to the STML generation script is additionally provided as input.

3.7.1 Spatio-temporal detection and analysis modules

Two spatio-temporal detection and analysis modules were implemented corresponding to the two spatial entity types considered i.e. regions and clusters.

The region spatio-temporal detection and analysis module is called `RectangularDetectRegions` and it implements Algorithm 1 using the OpenCV functions given in Table C.1. Conversely the cluster spatio-temporal detection and analysis module is called `SimulationDetectClusters` and it implements the improved DBSCAN algorithm described by Tran et al. (Tran et al., 2013).

Both spatio-temporal detection and analysis modules were implemented in C++ for efficiency purposes. In order to avoid recompilation the parameter values for the spatio-temporal detection and analysis modules are loaded at runtime from xml configuration files. The contents of these xml files can either be changed by hand or via the Graphical User Interface (GUI) of the corresponding module which displays in real time how the detected regions/clusters change when altering the values of the parameters. A description of the command line arguments and execution syntax is given when running the spatio-temporal detection and analysis modules with the “`--help`” command line argument.

3.7.2 Model checker Mudi

The main use case considered for the model checker Mudi is to validate a given computational model relative to a PBLSTL specification describing how both numeric and spatial properties are expected to change over time. Other use cases considered include choosing the model checking algorithm (see Table 2.1) and the maximum tolerated model checking approximation error, and providing as input the maximum number of minutes to wait for new STML files to be generated and the paths to the file containing the PBLSTL specification, the folder containing STML files and the script used to generate STML files on demand (see Algorithm 2). A graphical description of the use cases is given in Figure 3.8 as a Unified Modelling Language (UML) use case diagram.

The architecture of Mudi was designed to be modular and is conceptually separated into the model checking and the inference engine layers as depicted in Figure 3.9. The main advantage of this design choice is that changes of the model checking layer do not require updates of the inference engine layer and vice versa.

The model checking layer comprises all model checking approaches supported by Mudi. Independently of the chosen model checking approach the same inference engine is employed to evaluate formal PBLSTL statements against executions

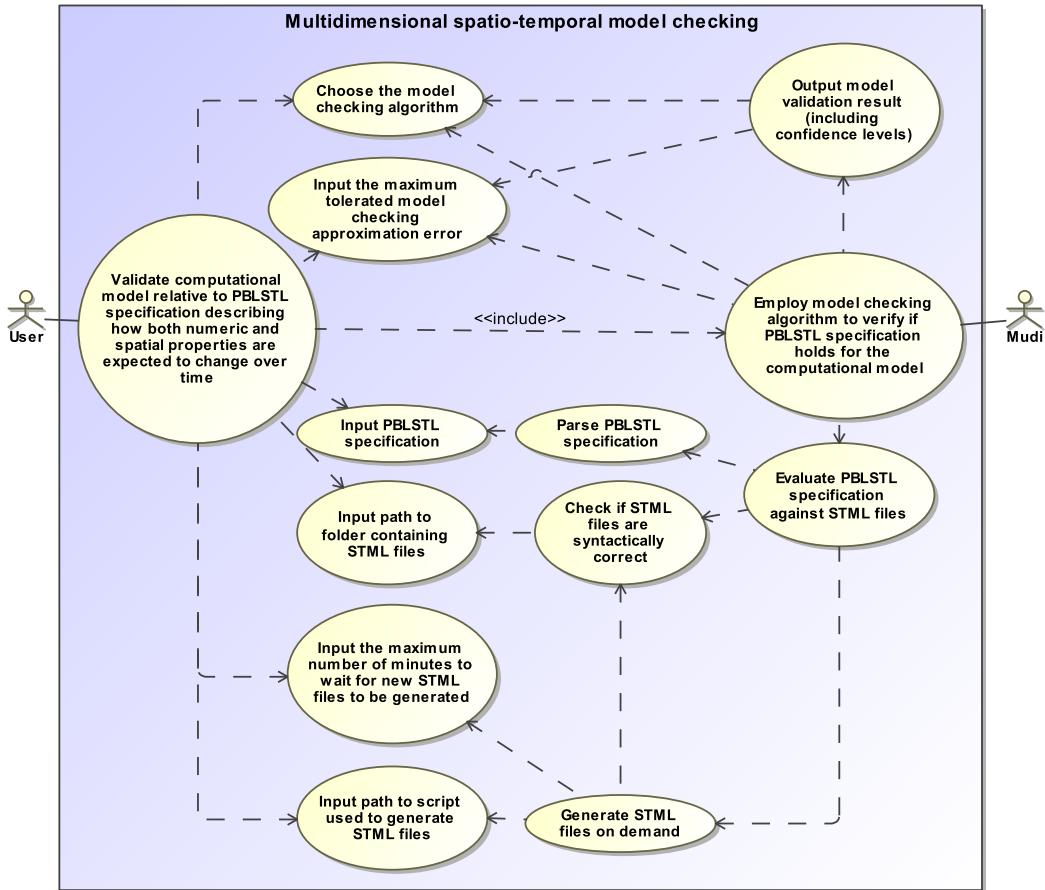


Figure 3.8: Use case diagram for multidimensional spatio-temporal model checking using the model checker Mudi. The user can validate a computational model relative to a PBLSTL specification describing how both numeric and spatial properties are expected to change over time. The validation of the computational model relative to a PBLSTL specification includes choosing the model checking algorithm and the maximum tolerated model checking approximation error, and providing as input the maximum number of minutes to wait for new STML files to be generated and the paths to the file containing the PBLSTL specification, the folder containing STML files, and the script used to generate STML files on demand. Conversely the model checker Mudi employs the model checking algorithm chosen by the user to verify if a PBLSTL specification holds for the computational model considered. The validation of the computational model includes evaluating the PBLSTL specification against STML files and outputting the model validation results. The evaluation of the PBLSTL specification against STML files includes parsing the PBLSTL specification, checking if the STML files are syntactically correct and generating STML files on demand.

of the model considered. For explanatory purposes the integration of the model checking approaches, and the PBLSTL logic property parser and evaluator is described by the UML class diagram in Figure 3.10.

For both efficiency and cross platform compatibility purposes Mudi was implemented in C++. The current version of the model checker was designed to be executed only from the command line. The user chooses the desired model checking algorithm and enters the required parameters via command line flags; run Mudi with the “--help” command line argument for more details. The model checking approaches supported were implemented without any external library dependencies. Conversely the PBLSTL logic property parsing and evaluation

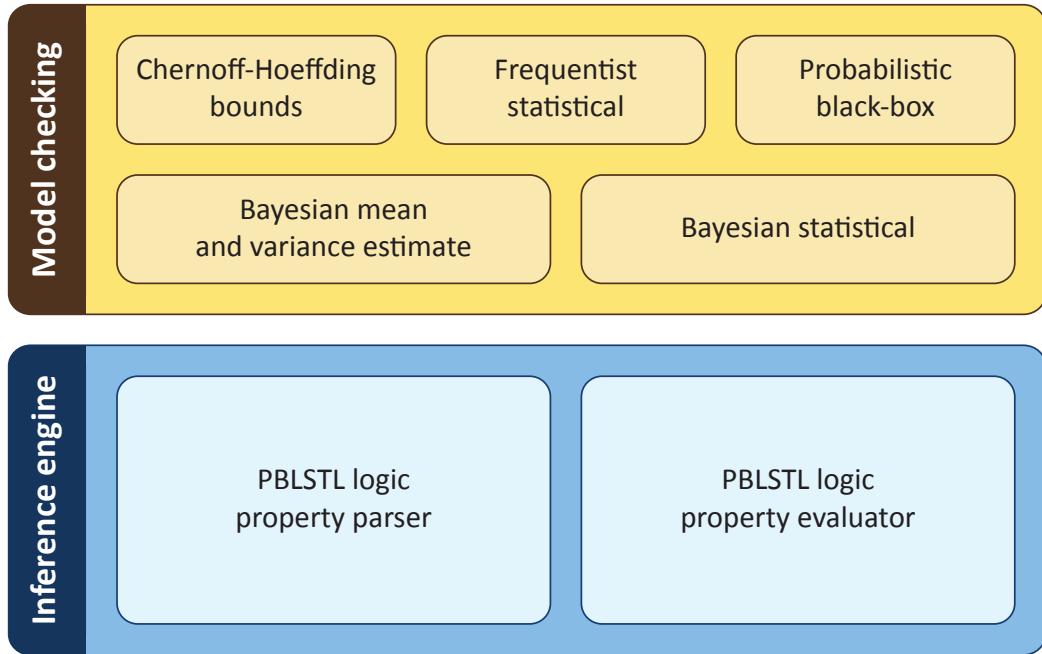


Figure 3.9: The modular architecture of Mudi comprises the model checking and inference engine layers. The model checking layer contains all model checking approaches supported by Mudi. Conversely the inference engine layer consists of the PBLSTL logic property parser (considering the BLSTL syntax) and evaluator (considering the BLSTL semantics). Every model checking approach supported by Mudi employs the PBLSTL logic property evaluator to determine if PBLSTL logic properties hold for executions of the model considered.

modules depend on the Boost Spirit C++ parser generator library (Guzman and Kaiser, 2015). The main reason for choosing this specific parser generator as opposed to more established ones (e.g. Bison and yacc) is its ability to generate the parser and construct the abstract syntax tree corresponding to the logic property using inline C++ code. Parsers generated with this library are top-down recursive descent.

Mudi was implemented as an offline model checker and takes as input model simulation traces (i.e. time series data) rather than computational models. The offline model checking approach has two main advantages. First of all the model checker implementation is decoupled from the specific modelling formalisms employed to encode the computational models. Consequently Mudi can be employed to verify computational models encoded using various modelling formalisms provided that the corresponding computational models satisfy the constraints of an SSpDES model. Secondly Mudi can be employed to evaluate PBLSTL specifications against time series data recorded both during *in silico* model simulations and *in vitro* experiments. Therefore the model checker can be employed for systems biology applications to check if executions of computational models match observations of the real-life systems they encode, and for synthetic biology applications to check if the behaviour of synthetically engineered biological systems matches the

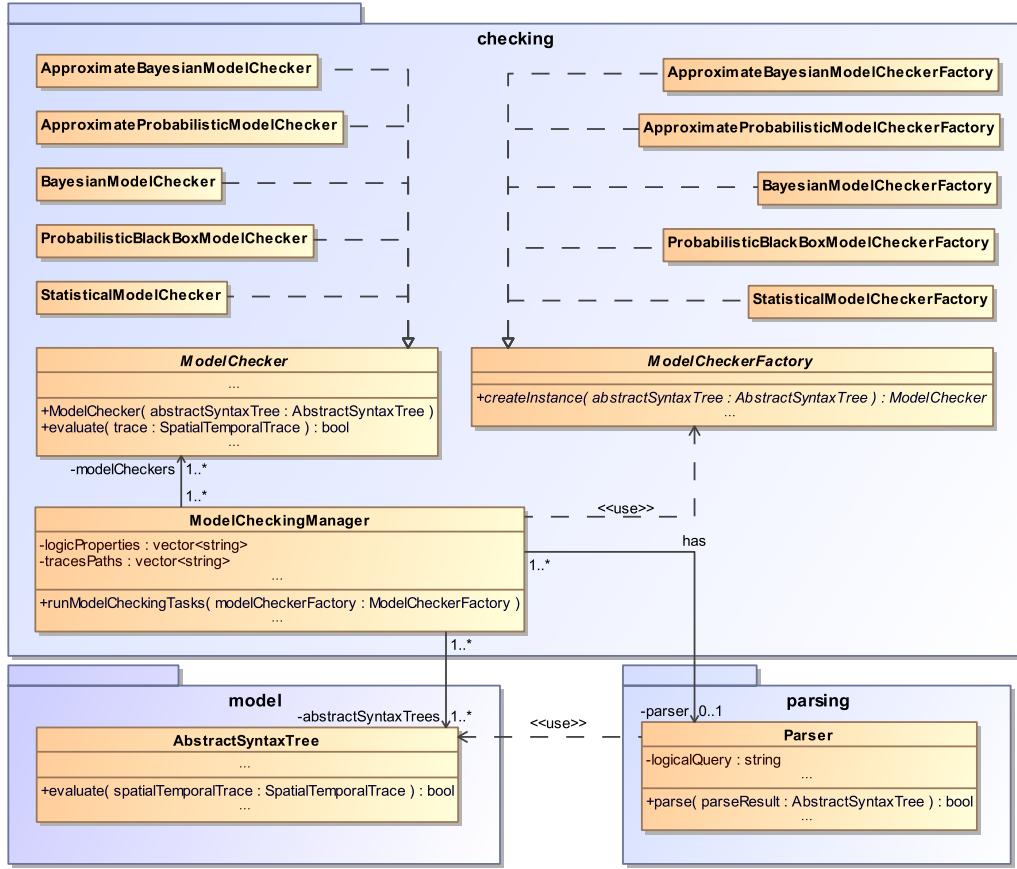


Figure 3.10: Class diagram corresponding to the model checking approaches considered (see Table 2.1), and the PBLSTL logic property parser and evaluator. The ModelCheckingManager class is employed to run the model checking tasks. It stores a collection of (PBLSTL) logic properties represented as strings of characters, and a collection of paths to STML files. For each logic property considered a separate model checker instance is created (e.g. StatisticalModelChecker) using a model checker factory class (e.g. StatisticalModelCheckerFactory) corresponding to the model checking approach chosen by the user. The supported model checking approaches and corresponding factories are implemented in classes which realize the ModelChecker, respectively the ModelCheckerFactory interface. An instance of a model checking class can be used to evaluate a logic property, represented as an instance of the AbstractSyntaxTree class, relative to a given execution of the model, represented as an instance of the SpatialTemporalTrace class. The classes used to encode the model checking approaches correspond to the model checking layer in Figure 3.9. To evaluate logic properties, the logic properties need to be first parsed to check if they are syntactically correct using the Parser class, which corresponds to the PBLSTL logic property parser in Figure 3.9. If the PBLSTL logic property is syntactically correct then it is stored as an instance of the AbstractSyntaxTree class and can be evaluated against executions of the model considered (i.e. spatio-temporal traces). Therefore the AbstractSyntaxTree class corresponds to the PBLSTL logic property evaluator in Figure 3.9.

in silico predictions of the computational models employed for their design. Conversely the main disadvantage of implementing Mudi as an offline model checker is efficiency, because model simulation traces cannot be generated on-demand, in-memory and potentially stopped early. Moreover the model simulation traces need to be stored and loaded from disk which leads to increased model checker execution times.

To check for the presence of bugs in the implementation of Mudi both black- and white-box testing was employed. Unit tests were implemented using the Google

Test unit testing framework (Google, 2015) and covered all the main functionalities of the model checker, namely parsing, evaluation and model checking. The case studies considered in the unit tests were illustrative examples chosen for validation purposes and were not derived from experimental studies or the literature. As per the principles of test-driven development unit tests were typically written first, and then the corresponding features were implemented such that the unit tests executed successfully. Moreover when releasing a new version of Mudi the minimum requirement was that all unit tests needed to execute successfully.

3.7.3 Availability

The source code for the spatio-temporal detection and analysis modules is made freely available online via <https://github.com/IceRage/Mule>, in the GitHub repository of the model checker Mule (see Section 5.8). Conversely the model checker Mudi is made freely available online in binary format via the official Mudi website <http://mudi.modelchecking.org>. In addition the official Mudi website contains the xsd schema for STML, the datasets of STML files and PBLSTL specifications for the case studies against which the efficacy of Mudi is illustrated (see Chapter 4), a tutorial on how to download, install and use Mudi, and a link to the Mudi issue tracking webpage.

3.8 Related work

Although there is currently a lack of multidimensional spatio-temporal model checking approaches for computational models of biological systems, the need for spatio-temporal models and corresponding analysis and validation approaches was mentioned previously in other fields of science. Illustrative examples of such approaches employed in epidemiology and spatial information theory are provided below.

3.8.1 Epidemiology

A quantitative spatio-temporal model checking approach was described by Jha and Ramanathan (Jha and Ramanathan, 2012) for reasoning about uncertainty in epidemiological models. The authors define a Bounded Spatio-Temporal Logic (BSTL) which extends BLTL with two spatial functions, namely $P(A, C)$ for computing the number of type A entities present in the compartment C , and $N(A, B, rad)$ for computing the number of type A entities lying within a radius rad of one or more type B entities. More recently this work was extended by

Hussain et al. (Hussain et al., 2014b) who define the probabilistic spatio-temporal specification language EpiSpec. Compared to the previous approach, EpiSpec is based on first-order logic, defines functions with a similar semantics to P and N and additionally enables the use of potentially complex arithmetic (e.g. $\frac{dE}{dt}$, $\int_{t_1}^{t_2} Edt$) expressions.

Another quantitative spatio-temporal model checking approach was introduced by Nenzi and Bortolussi (Nenzi and Bortolussi, 2014; Nenzi, 2014) and its applicability was illustrated based on an epidemiology case study describing the spreading of cholera. The formal logic underlying the approach is called Signal Spatio-Temporal Logic (SSTL) and it extends Signal Temporal Logic (STL) with two spatial operators $\diamondsuit_{[w_1, w_2]} \phi$ and $\square_{[w_1, w_2]} \phi$. Translated into natural language the semantics of $\diamondsuit_{[w_1, w_2]} \phi$ and $\square_{[w_1, w_2]} \phi$ is that there exists (\diamondsuit) a location, respectively for all (\square) locations at a distance $d \in [w_1, w_2]$ from the current location, logic statement ϕ holds.

From a spatial point of view these approaches enable reasoning only about properties (e.g. number of entities) in specific locations (BSTL/EpiSpec) and/or some/all locations at a bounded distance from the current location (BSTL/EpiSpec, SSTL). Therefore they do not support reasoning about how emergent spatial entities and their properties (e.g. area) change over time.

3.8.2 Spatial information theory

The spatial information theory literature describes several formal languages called spatial logics which enable reasoning about the representation of systems in space and potentially how this representation changes over time (Aiello et al., 2007, Chapter 9).

Depending on the considered application the employed spatial logic can be qualitative (e.g. (McKinsey and Tarski, 1944; Montanari et al., 2009; Randell et al., 1992; Tarski, 1938)), (semi-)quantitative (e.g. (Condotta, 2000; Xu, 2007)) or a combination thereof (i.e. hybrid) (e.g. (Kor and Bennett, 2013; Liu et al., 2009b)). Due to the uncertainty or lack of precision usually associated with spatial information, qualitative spatial logics are usually employed (Bresolin et al., 2010). Most qualitative spatial logics are defined using constraint based techniques, which were initially developed for temporal reasoning (Aiello et al., 2007, Chapter 4).

The constraints often considered are topology, orientation and distance (Aiello et al., 2007, Chapter 4) considering a 2D representation of space.

Topological qualitative spatial logics enable describing topological relations between spatial entities. One of the most employed topological qualitative spatial

logics is \mathcal{RCC} -8 which is an instance of the Region Connection Calculus (\mathcal{RCC}) proposed by Randell et al. (Randell et al., 1992). The primitive relation of \mathcal{RCC} is $C(x, y)$ read as “region x connects to region y ” and which holds when x and y share at least one common point. \mathcal{RCC} -8 is an instance of \mathcal{RCC} that contains 8 topological relations, namely $DC(x, y)$ (i.e. x is disconnected from y), $PO(x, y)$ (i.e. x and y are partially overlapping), $EC(x, y)$ (i.e. x is externally connected with y), $EQ(x, y)$ (i.e. x is equal to y), $TPP(x, y)$ (i.e. x is a tangential proper part of y), $NTPP(x, y)$ (i.e. x is a non-tangential proper part of y), $TPP^{-1}(x, y)$ (i.e. x is an inverse tangential proper part of y), $NTPP^{-1}(x, y)$ (i.e. x is an inverse non-tangential proper part of y); see Figure 3.11 for a graphical description of these relations.

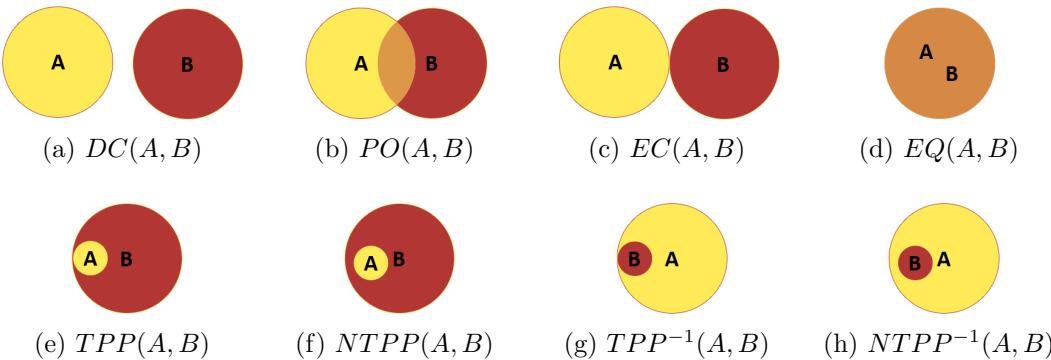
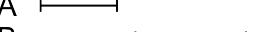
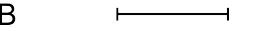
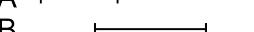
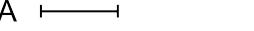
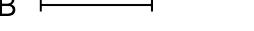
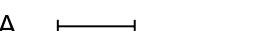
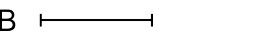


Figure 3.11: A graphical description of the topological spatial relations in \mathcal{RCC} -8. $DC(x, y)$ states that regions x and y are disconnected. $PO(x, y)$ states that x and y are partially overlapping. $EC(x, y)$ states that x and y are externally connected. $EQ(x, y)$ states that x and y are equal. $TPP(x, y)$ states that x is a tangential proper part of y . $NTPP(x, y)$ states that x is a non-tangential proper part of y . $TPP^{-1}(x, y)$ states that x is an inverse tangential proper part of y . $NTPP^{-1}(x, y)$ states that x is an inverse non-tangential proper part of y .

Directional qualitative spatial logics enable reasoning about the relative positioning/orientation of entities in space. An illustrative example of such a logic is Rectangle Algebra (RA). In RA the relative positioning of regions in 2D space is defined with respect to the relative positioning of the regions’ projections on the Ox and Oy axes. The projection of a 2D region on a one-dimensional axis is an interval. To determine the relative positioning of two intervals the set of 13 relations defined in Allen’s Interval Algebra (IA) (Allen and Hayes, 1989) are employed {before, after, meets, met by, overlaps, overlapped by, starts, started by, during, contains, finishes, finished by, equals}; see Table 3.2 for a description of these relations. Since the number of IA relations is 13, the number of relations employed to describe the relative positioning of two regions in 2D space is 13 (considering the Ox axis) · 13 (considering the Oy axis) = 169.

Finally qualitative spatial logics considering distance constraints describe the relative distance between two entities in space using relations such as “very far”,

Table 3.2: Interval Algebra relations defined over two intervals A and B , where A^- and A^+ are the endpoints of A , respectively B^- and B^+ are the endpoints of B .

Relation	Example	Meaning
<i>Before</i> (A, B)	 A --- B ---	$A^- < A^+ < B^- < B^+$
<i>After</i> (B, A)	 B --- A ---	$A^- < A^+ < B^- < B^+$
<i>Meets</i> (A, B)	 A --- B ---	$A^- < A^+ = B^- < B^+$
<i>MetBy</i> (B, A)	 B --- A ---	$A^- < A^+ = B^- < B^+$
<i>Overlaps</i> (A, B)	 A --- B ---	$A^- < B^- < A^+ < B^+$
<i>OverlappedBy</i> (B, A)	 B --- A ---	$A^- < B^- < A^+ < B^+$
<i>Starts</i> (A, B)	 A --- B ---	$A^- = B^- < A^+ < B^+$
<i>StartedBy</i> (B, A)	 B --- A ---	$A^- = B^- < A^+ < B^+$
<i>During</i> (A, B)	 A --- B ---	$B^- < A^- < A^+ < B^+$
<i>Contains</i> (B, A)	 B --- A ---	$B^- < A^- < A^+ < B^+$
<i>Finishes</i> (A, B)	 A --- B ---	$B^- < A^- < A^+ = B^+$
<i>FinishedBy</i> (B, A)	 B --- A ---	$B^- < A^- < A^+ = B^+$
<i>Equals</i> (A, B)	 A --- B ---	$A^- = B^- < A^+ = B^+$

“far”, “commensurate”, “close” and “very close”. Depending on the problem considered the number of different distance levels varies. For instance assuming a coarse grained representation of space two distance levels such as “far” and “close” could potentially suffice. An example of a qualitative spatial logic considering distance constraints is described by Clementini et al. (Clementini et al., 1997).

One of the main advantages of qualitative spatial logics is that they enable reasoning about the spatial representation of a system in the presence of uncertainty. Conversely one of the main disadvantages is that qualitative spatial logic descriptions are often imprecise. Therefore it could be potentially difficult using such logics to accurately describe how emergent spatial entities and their properties change over time.

PBLSTL inherently supports only quantitative spatial properties. However qualitative spatial properties can be additionally expressed in PBLSTL if they are rewritten in a quantitative manner. For instance if we would like to specify in PBLSTL that a point-wise region A at position (x, y) in 2D space is a non-tangential proper part of a rectangular region B defined by the points $M(0, 10)$, $N(10, 10)$, $O(10, 20)$ and $P(0, 20)$ (e.g. corresponding to $NTPP(A, B)$ in \mathcal{RCC} -8), then we could write that the coordinates of the region A lie within and do not touch the border of region B (i.e. $0 < x < 10$ and $10 < y < 20$). Moreover high-level spatial functions could be added to PBLSTL to enable encoding quantitative

spatial properties in a compact form (e.g. $\text{inside}(A, B)$ if and only if $0 < x < 10$ and $10 < y < 20$); PBLSTL could be adapted automatically to the new type of spatial functions using the meta model checking concept, which is introduced later in Chapter 5.

Summary

This chapter introduced a novel multidimensional spatio-temporal model checking methodology which enables validating computational models of biological systems with respect to how both their numeric and spatial properties change over time considering a single level of organization. In the beginning SSpDESs are defined as theoretical models for abstractly representing biological systems. Next spatio-temporal analysis modules are introduced for automatically detecting regions or clusters in time series data and computing how their spatial properties (e.g. area, perimeter etc.) change over time. Time series data describing how both numeric and spatial properties change over time are formatted according to the standard representation format STML introduced here. Then the temporal logic (P)BLSTL is defined for encoding the formal specification against which SSpDES models are validated. Afterwards corresponding Bayesian and frequentist, estimate and hypothesis testing based model checking algorithms are described. Moreover a proof is provided illustrating that the multidimensional spatio-temporal model checking problem is well-defined. Implementation details and a concise comparison of the approach with other spatio-temporal model checking methods from the epidemiology and spatial information theory literature are presented in the end.

CHAPTER 4

Validation of multidimensional computational models of biological systems

Introduction

This chapter illustrates how the multidimensional spatio-temporal model checking methodology described in Chapter 3 can be employed to validate computational models of biological systems. The biological case studies considered are phase variation in bacterial colony growth and the chemotactic aggregation of cells. Corresponding computational models have been validated against relevant PBLSTL specifications using the model checker Mudi. Conclusions and limitations of the multidimensional model checking methodology are described at the end.

4.1 Description

The efficiency and expressivity of the multidimensional spatio-temporal model checking methodology was assessed based on two biological case studies encoding phase variation patterning in bacterial colony growth, and the chemotactic aggregation of cells.

The corresponding computational models are stochastic and have been encoded using high-level modelling formalisms which can be translated to an equivalent SSpDES representation.

For generalizability purposes the stochastic computational models have been encoded using different high-level modelling formalisms, namely Coloured Stochas-

tic Petri Nets for phase variation in bacterial colony growth, and Cellular Potts and partial differential equations for the chemotactic aggregation of cells.

Results generated via model simulation were processed by the spatio-temporal analysis modules and were translated to STML. The spatial entity types considered for the phase variation and the chemotactic aggregation of cells case studies were regions, respectively clusters.

STML files representing the model behaviour were evaluated against formal PBLSTL specifications. The main purpose of these specifications was to illustrate the expressivity of the methodology and not to test novel biological hypotheses. Probability values considered in the PBLSTL statements are only approximations of corresponding qualitative natural language descriptions (e.g. high probability $\Rightarrow 0.9$) and were chosen for illustrative purposes.

For model checking purposes no prior information was employed other than the computational model and the PBLSTL specification. Therefore the frequentist rather than the Bayesian statistical model checking approach was employed throughout. Relevant comparisons between different approximate probabilistic model checking approaches are provided in the original papers introducing the approaches (see Subsubsection 2.4.2.3). Since the comparison results are independent of particular model representations and logic formalisms they will not be restated here.

For reproducibility purposes the generated STML files, and the formal PBLSTL specification corresponding to both computational models have been made available at <http://mudi.modelchecking.org/case-studies>.

4.2 Phase variation patterning in bacterial colony growth

Phase variation is a stochastic gene expression switching mechanism employed by microbial populations to potentially develop variants (i.e. mutants) which adapt to foreseeable frequent environmental or selective conditions (e.g. host immune responses) (Salaiin et al., 2003, 2005; Saunders et al., 2003).

One of the most readily observable compositional effects of phase variation in cultures grown *in vitro* is the development of sector-like patterns (Pârvu et al., 2013). The geometric properties (e.g. angle, area, shape) of these sector-like patterns are potentially correlated with the mutation and/or fitness rates of the bacteria.

4.2.1 Model construction

To investigate the potential relationship between the mutation and/or fitness rates of bacteria with phase variables genes and the geometric properties of the emerging sector-like patterns a Coloured Stochastic Petri Net model was constructed using the modelling software Snoopy (Heiner et al., 2012); the model is made freely available online (Pârvu et al., 2015, Supplementary materials).

In this model the spatial domain is represented explicitly as a 101×101 regular square lattice where the values recorded for each lattice position are the number of type *A* (i.e. wild type) and type *B* (i.e. mutant) bacteria. Starting from a single type *A* bacteria model simulations describe the growth of the colony according to predefined bacterial mutation and fitness rates. The resulting sector-like patterns are defined by lattice positions in which the proportion of type *B* relative to type *A* bacteria is higher than a user-defined threshold.

4.2.2 Spatio-temporal analysis

The computational model was unfolded using Snoopy and simulated on a Unix cluster using the Gillespie Stochastic Simulation Algorithm (Gillespie, 1977) implemented in MARCIE (Heiner et al., 2013). The average model simulation time was approximately 50 minutes.

For model checking purposes the simulation output was processed using the spatio-temporal analysis and translated to STML. For this case study the region detection and analysis module was employed because sector-like patterns (and not clusters of such patterns) are of interest. An illustrative example of the translation steps applied to each spatio-temporal time series is depicted in Figure 4.1.

4.2.3 Formal specification

The generated STML dataset was evaluated against the formal specification comprising PBLSTL logic properties. Depending on the modelled microorganism and the associated mutation/fitness rates the values and/or parameters of the logic properties varied. We describe here a generic set of logic statements to illustrate the expressivity of the formal language PBLSTL. Therefore the structure of PBLSTL statements is emphasized and not particular parameter values. For comprehension purposes the specification will be described both in natural language and PBLSTL below.

1. **Natural language:** One of the first requirements is that the probability of the number of sector-like patterns to increase or stay constant (but

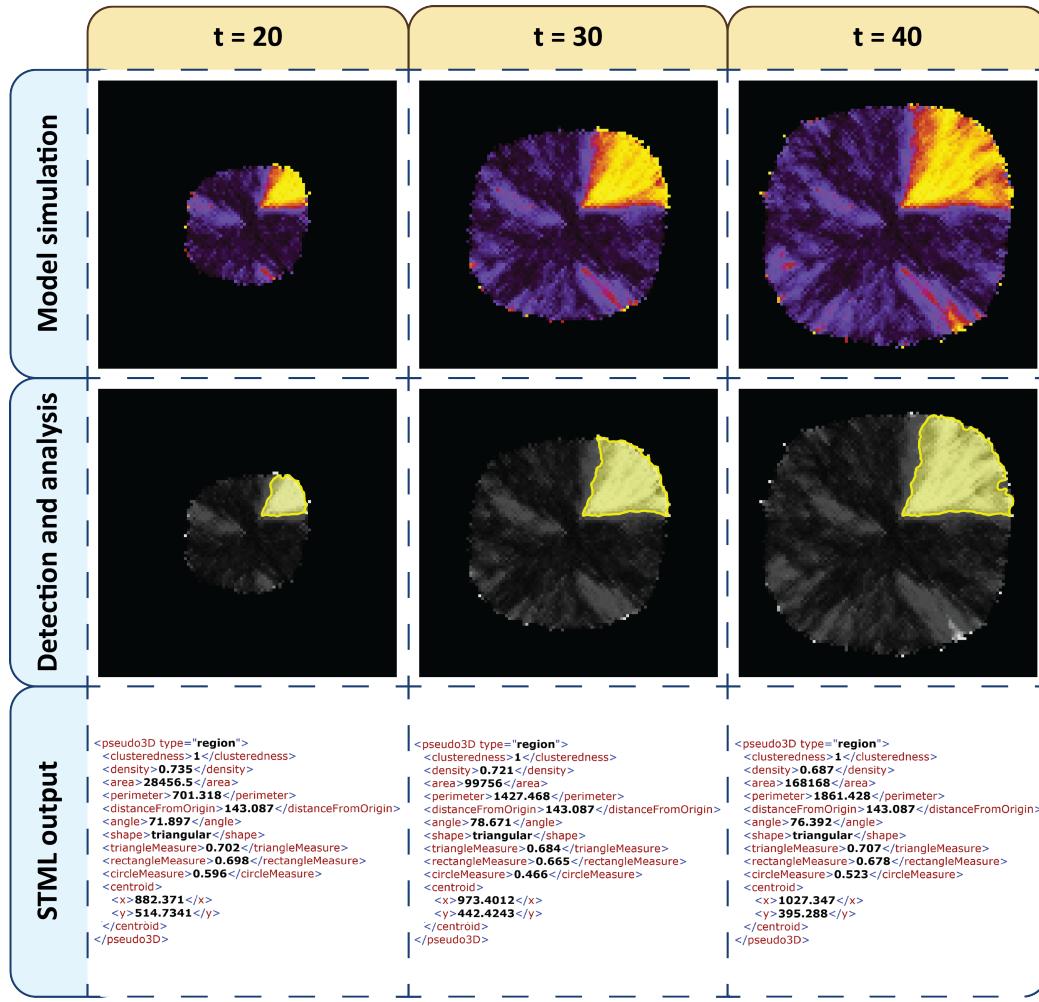


Figure 4.1: Spatio-temporal detection and analysis for a phase variation model simulation. Each column corresponds to a different time point from the simulation ($t = 20, 30$ and 40). The rows considered from top to bottom represent the stages of translating time series data to STML output files (model simulation, automatic detection and analysis of regions/sector-like patterns, and output in STML format).

never decrease) during the bacterial colony growth is greater or equal to a threshold value. In our case we set this threshold to 0.95. The reason for this requirement is that we do not expect developed sectors to disappear.

PBLSTL: $P \geq 0.95 [G [0, 100] (d(count(regions)) \geq 0)]$.

2. Natural language: In case sector-like patterns emerge the probability that one of them will contain holes is less than 0.05. This statement can be rewritten using the clusteredness measure of the regions i.e. the probability that the minimum clusteredness degree of all sectors is less than a certain threshold value (in our case 0.9) is less than 0.05.

PBLSTL: $P < 0.05 [F [0, 100] ((count(regions) > 0) \wedge (min(regions, clusteredness) < 0.9))]$.

3. **Natural language:** The average density of the detected sectors, representing the concentration of “mutant” (type B) cells relative to “normal” (type A) cells, should be greater than 0.5 with probability greater than 0.95.

PBLSTL: $P > 0.95 [(F [0, 100] (G [0, 100] ((count(regions) > 0) \wedge (avg(regions, density) > 0.5)))) \vee (G [0, 100] (count(regions) = 0))].$

4. **Natural language:** Moreover the average area of the sectors oscillates at least one time during the growth of the bacterial colony with probability greater than 0.5. By oscillations we mean an increase of the average area followed eventually by a decrease or vice versa. In PBLSTL oscillations can be represented using the difference operator d . For this particular statement we will specify that at some point in the future the rate of change (difference) of the average area will be positive and then eventually negative or vice versa. Such oscillations are expected because the relative density of “mutant” cells with respect to “normal” cells is considered when detecting sectors. Therefore as the colony grows it may be the case that at the most outward edge of a sector initially the “mutant” cells dominate a position in the discretised space but then they are overrun by the “normal” cells. In other words it may be the case that a position which is contained by a sector will no longer do so in the future.

PBLSTL: $P > 0.5 [(F [0, 100] ((count(regions) > 0) \wedge (d(avg(regions, area)) > 0) \wedge F [0, 100] ((d(avg(regions, area)) < 0)))) \vee (F [0, 100] ((d(avg(regions, area)) < 0) \wedge F [0, 100] ((count(regions) > 0) \wedge (d(avg(regions, area)) > 0)))) \vee (G [0, 100] (count(regions) = 0))].$

5. **Natural language:** Following the same reasoning we also specify that the average perimeter value of the sectors oscillates at least five times during the growth of the bacterial colony with probability greater than 0.6. The number of oscillations was chosen to illustrate the rate at which the model checker execution time increases when nesting multiple temporal logic propositions.

PBLSTL: $P > 0.6 [(F [0, 100] ((count(regions) > 0) \wedge (d(avg(regions, perimeter)) > 0) \wedge F [0, 100] ((d(avg(regions, perimeter)) < 0)) \wedge F [0, 100] ((count(regions) > 0) \wedge (d(avg(regions, perimeter)) > 0)) \wedge F [0, 100] ((d(avg(regions, perimeter)) < 0)) \wedge F [0, 100] ((d(avg(regions, perimeter)) > 0)) \wedge F [0, 100] ((d(avg(regions, perimeter)) < 0))) \vee (G [0, 100] (count(regions) = 0))].$

$$\begin{aligned}
& F [0, 100] ((count(regions) > 0) \wedge \\
& (d(avg(regions, perimeter)) > 0) \wedge \\
& F [0, 100] ((d(avg(regions, perimeter)) < 0))))))) \vee \\
& (F [0, 100] ((d(avg(regions, perimeter)) < 0) \wedge \\
& F [0, 100] ((count(regions) > 0) \wedge \\
& (d(avg(regions, perimeter)) > 0) \wedge \\
& F [0, 100] ((d(avg(regions, perimeter)) < 0) \wedge \\
& F [0, 100] ((count(regions) > 0) \wedge \\
& (d(avg(regions, perimeter)) > 0) \wedge \\
& F [0, 100] ((d(avg(regions, perimeter)) < 0) \wedge \\
& F [0, 100] ((count(regions) > 0) \wedge \\
& (d(avg(regions, perimeter)) > 0))))))) \vee \\
& (G [0, 100] (count(regions) = 0))].
\end{aligned}$$

6. **Natural language:** The maximum angle described by any sector with respect to the origin is expected to be greater than 120° with probability less than 0.1.

PBLSTL: $P < 0.1 [F [0, 100] ((count(regions) > 0) \wedge (max(regions, angle) > 120))].$

7. **Natural language:** Moreover sectors are expected to develop from the origin outwards. Therefore the minimum distance from the origin would be expected to be greater than 100 (relative to scale of analysed images) with probability greater or equal to 0.95.

PBLSTL: $P \geq 0.95 [(F [0, 100] (G [0, 100] ((count(regions) > 0) \wedge (min(regions, distanceFromOrigin) > 100)))) \vee (G [0, 100] (count(regions) = 0))].$

8. **Natural language:** Finally on average most of the sectors should develop and maintain a triangular-like shape throughout the entire bacterial colony growth with probability greater than 0.8.

PBLSTL: $P > 0.8 [(F [0, 100] (G [0, 100] ((count(regions) > 0) \wedge (min(regions, triangleMeasure) > max(regions, rectangleMeasure)) \wedge (min(regions, triangleMeasure) > max(regions, circleMeasure)))))) \vee (G [0, 100] (count(regions) = 0))].$

4.2.4 Model checking

Each PBLSTL statement, stored in a separate input file, was individually evaluated against the STML files representing the model behaviour 500 times using the

frequentist statistical model checking approach with the probability of type I (i.e. false positives) and type II (i.e. false negatives) errors set to 5%; see Appendix A.2 for a more detailed description of type I and type II errors in the context of statistical model checking. Conclusions drawn from the statistical analysis of the model checking results corresponding to each PBLSTL statement are summarized in Table 4.1.

Table 4.1: Model checking statistical analysis results for the phase variation case study. Entries in the “id” column represent the numeric identifiers associated with each PBLSTL statement. The “% true PBLSTL” column describes what percentage of the 500 executions concluded that the PBLSTL statement is true. “#total STML” represents the total number of STML files evaluated for the PBLSTL statement; columns “#true STML” and “#false STML” represent the number of STML files for which the PBLSTL statement was evaluated true, respectively false. “Execution times” presents the average model checking execution time for each PBLSTL evaluation using the “minutes:seconds” format. “ μ ” and “ σ ” represent the mean and standard deviation.

id	% true PBLSTL	#total STML		#true STML		#false STML		Execution times	
		μ	σ	μ	σ	μ	σ	μ	σ
1	100	67.95	12.51	67.12	11.52	0.83	0.98	0:2.87	0:0.57
2	100	103.21	57.70	0.81	1.18	102.40	56.56	0:4.40	0:2.60
3	100	63.05	8.43	62.62	7.74	0.43	0.69	0:2.75	0:0.43
4	99.6	3.29	12.79	1.84	7.51	1.45	5.36	0:0.22	0:0.83
5	74.6	15.35	14.17	6.98	5.94	8.36	8.72	0:0.99	0:0.92
6	99.8	982.63	111.18	10.74	1.62	971.88	109.59	0:43.71	0:9.70
7	100	106.21	42.53	102.53	39.42	3.68	3.11	0:4.67	0:2.02
8	99.8	30.04	63.24	24.87	51.40	5.17	11.99	0:1.33	0:2.59

For half of the PBLSTL statements ($id = 1, 2, 3, 7$) 100% of the 500 model checker executions concluded with the answer true. However in case of PBLSTL statements 6 and 8 the percentage was 99.8%, respectively 99.6% for PBLSTL statement 4 and 75.6% for PBLSTL statement 5. It is important to note that this does not mean that the model checking results are incorrect. Moreover in the approximate probabilistic setting if the model checking result is false for a logic property ϕ this does not imply that $\sim\phi$ is true. The variation in the results obtained for PBLSTL statements 4, 5, 6 and 8 is due to the fact that the model checker was executed with the maximum probability of type I and type II errors equal to 5%. Under these assumptions the evaluation result for a PBLSTL statement depends on the order and number of obtained true/false evaluations for individual STML files.

To reduce the variation of the PBLSTL evaluations the value of the probability of type I/II errors needs to be decreased. The required number of evaluated simulations is indirectly proportional to the type I/II error probability. Thus more simulation evaluations are required as the error probabilities are decreased. In the extreme case if the probability of both type I and type II errors is set to zero the expected number of evaluated simulations is potentially infinite because the entire state space of the model would be potentially investigated.

Similarly there is a significant difference in the average total number of STML files against which the PBLSTL statements were evaluated. Depending on the comparison operator ($<$, \leq , \geq , $>$) and the specificity of the probability θ corresponding to each PBLSTL statement more/less evidence is required to prove that the statement is true/false. In our case the logic statement 6 required on average more than 950 STML evaluations and most of the time more than the maximum number of available simulations 1000. Since no path to an external model simulator was specified the model checker did not have enough evidence to decide using the frequentist statistical model checking approach if the PBLSTL statement holds. Therefore the provided answer was computed using the probabilistic black-box model checking approach.

The considerable difference in the number of required STML files is additionally reflected in the average execution times of the model checker. Thus the highest average execution time was recorded for the evaluation of PBLSTL statement 6. Since the formal specification for this case study comprises all PBLSTL statements the average execution time for the entire specification is computed as the sum of all average execution times (see Table 4.1, column 9):

$$\begin{aligned} \text{Execution time}_{\text{specification}} &= 0 : 2.87 + 0 : 4.40 + 0 : 2.75 + 0 : 0.22 \\ &\quad + 0 : 0.99 + 0 : 43.71 + 0 : 4.67 + 0 : 1.33 \\ &= 01 : 0.94 \text{ (minutes:seconds)}. \end{aligned}$$

In order to decrease the overall execution time the model checker was extended such that it can evaluate the specification comprising all PBLSTL statements in a single run. In this case each STML file is read into memory only once and thus reduces the number of required input/output (I/O) operations. Under these conditions the average execution time for the entire specification considering 500 runs was 0:44.41 (minutes:seconds), compared to 01:0.94 when the PBLSTL statements were evaluated individually.

4.3 Chemotactic aggregation of cells

Chemotaxis is the process through which cells detect concentration changes in chemical gradients and move towards chemical attractants, respectively away from chemical repellents. It is employed both by prokaryotic and eukaryotic cells and underpins many biological processes (e.g. human leukocytes migrate to sites of inflammation, cancer cells metastasize to other organs) (Cai and Devreotes, 2011; Jin, 2013).

In an attempt to better understand the intracellular mechanisms underlying chemotaxis computational models for various types of cells have been constructed (Jilkine and Edelstein-Keshet, 2011). Although such models differ at the intracellular level they exhibit relatively similar behaviours at the population level i.e. cells aggregate in the area with the highest concentration of chemical attractants.

4.3.1 Model construction

To gain a better understanding of the chemotactic aggregation of cells at the entire population level a corresponding computational model was constructed using the modelling and simulation software Morpheus (Starruß et al., 2014). The cells and their movement in the environment was represented using a Cellular Potts model and the distribution of the chemical gradient was encoded using partial differential equations; the model is made freely available online (Pârvu et al., 2015, Supplementary materials).

The discretised 2D space was represented using a 100×100 square lattice on which 100 cells were randomly distributed; cells' positions are recomputed for each model simulation. In order to activate the chemotactic behaviour of the cells a chemical attractant gradient was added in the environment according to a Gaussian distribution with parameters $\mu_x = \mu_y = 50$ and $\sigma_x = \sigma_y = 10$.

4.3.2 Spatio-temporal analysis

Each model simulation was generated in approximately 5 seconds and was translated to STML using the cluster detection mechanism because groups of (and not individual) cells were of interest. Cells occupied only one position of the discretised space and therefore their detection in images was straightforward. Instead of employing the region detection mechanism we implemented a custom lightweight cell detector which verifies the presence/absence of cells in each position (including pileup) considering the average pixel intensity; see Figure 4.2 for an example of the translation steps performed by the cluster detection mechanism for each model simulation.

To illustrate the integration of Mudi with a model simulator the model checker was executed initially without making available any STML files. Instead an external script responsible for simulating the model and converting the output to STML was provided as a command-line parameter. Thus Mudi executed the script on demand whenever extra model simulations were required. In general if

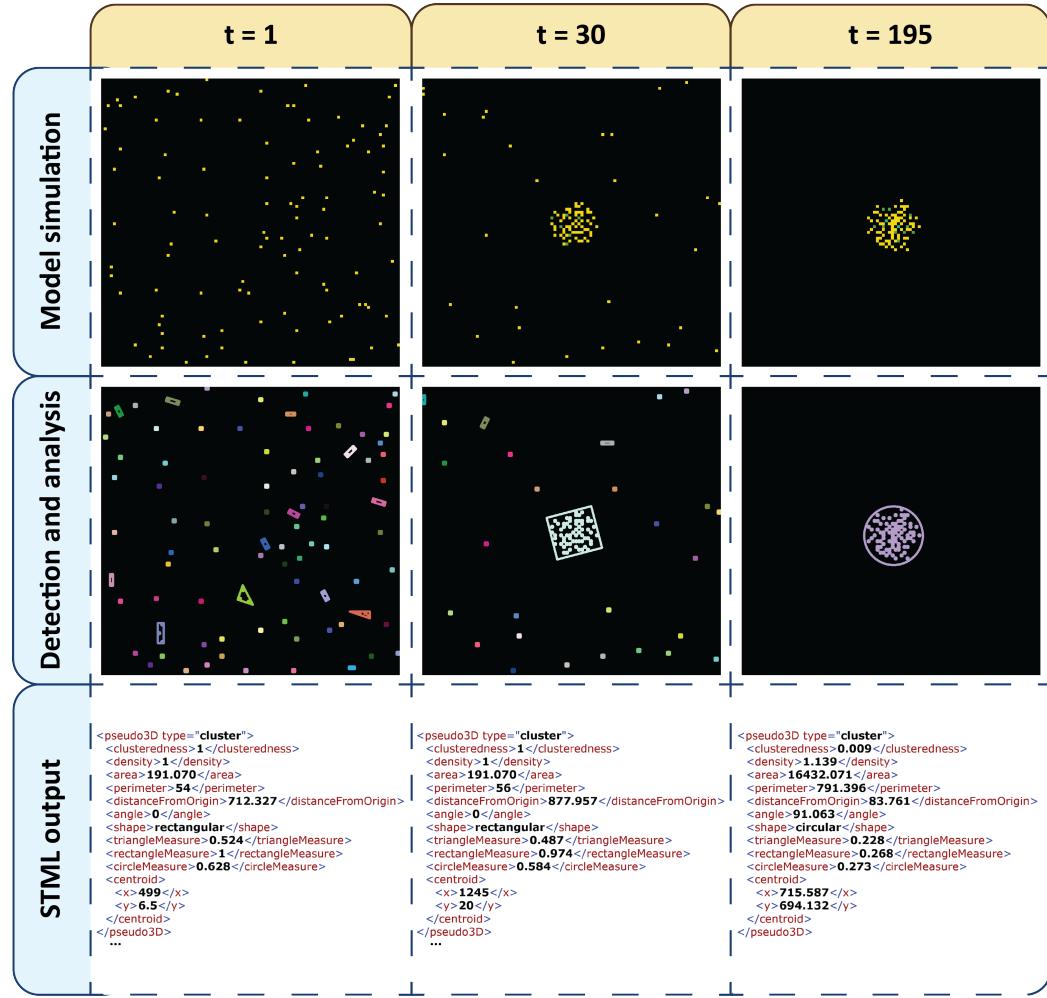


Figure 4.2: Spatio-temporal detection and analysis for a chemotaxis model simulation. Each column corresponds to a different time point from the simulation ($t = 1, 30$ and 195). The rows considered from top to bottom represent the stages of translating time series data to STML output files (model simulation, automatic detection and analysis of clusters, and output in STML format). The colour employed in the first row plots represents degree of pileup. “Yellow” positions in the discretised 2D space are occupied by 1 cell, “green” positions by 2 cells, and “teal” positions by 3 cells. The colours employed in the second row plots are used only to distinguish between different clusters. Each cluster is surrounded by a polygon whose shape (triangular/rectangular/circular) best matches the shape of the cluster.

a large number of simulations is required the maximum model checking time can be bounded via a user-defined parameter.

4.3.3 Formal specification

The generated STML files are evaluated against a formal PBLSTL specification describing the expected system behaviour. Similarly to the phase variation case study the logic statements considered are generic and were chosen to illustrate the expressivity of PBLSTL and not the phenotypic characteristics specific to a particular type of cells. Both a natural language and PBLSTL description of the specification is provided below.

9. **Natural language:** One of the most important properties is that cells aggregate in the area with highest concentration of chemical attractant. This means that at least one cluster is formed at a distance smaller than $\epsilon_{distance} > 0$ from the chemical gradient centre. Let us assume that the cluster centroid is the point (x, y) , and the centroid of the chemical gradient is $(703.5, 678.5)$. Then a cluster is at a distance smaller or equal to $\epsilon_{distance}$ from $(703.5, 678.5)$ if and only if:

$$dist((x, y), (703.5, 678.5)) = \sqrt{(x - 703.5)^2 + (y - 678.5)^2} < \epsilon_{distance}.$$

Considering that $y \in (678.5 - \epsilon_{distance}, 678.5 + \epsilon_{distance})$ this means that:

$$\begin{cases} x < 703.5 + \sqrt{\epsilon_{distance}^2 - (678.5 - y)^2} \\ x > 703.5 - \sqrt{\epsilon_{distance}^2 - (678.5 - y)^2} \\ y < 678.5 + \epsilon_{distance} \\ y > 678.5 - \epsilon_{distance}. \end{cases}$$

For this particular case study we set the value of $\epsilon_{distance}$ to 50.

PBLSTL: $P \geq 0.9 [F [0, 200] (count(filter(clusters, (centroidX < add(703.5, sqrt(subtract(2500, power(subtract(678.5, centroidY), 2)))) \wedge (centroidX > subtract(703.5, sqrt(subtract(2500, power(subtract(678.5, centroidY), 2)))) \wedge (centroidY < add(678.5, 50)) \wedge (centroidY > subtract(678.5, 50)))) \geq 1)]$.

10. **Natural language:** In addition the average clusteredness degree of individual clusters increases at least 5 times during the simulation time interval $[0, 200]$ with probability greater than 0.8. This means that the average distance between cells in the clusters is reduced at least five times during the specified time interval.

PBLSTL: $P \geq 0.8 [F [0, 200] (d(avg(clusters, clusteredness)) > 0) \wedge (X (F [0, 200] (d(avg(clusters, clusteredness)) > 0) \wedge (X (F [0, 200] (d(avg(clusters, clusteredness)) > 0) \wedge (X (F [0, 200] (d(avg(clusters, clusteredness)) > 0) \wedge (X (F [0, 200] (d(avg(clusters, clusteredness)) > 0))))))))]]$.

11. **Natural language:** In order to quantify the degree of clusteredness within and between different clusters a cluster validity index such as the Silhouette

can be employed. The value of the Silhouette is recorded for each time point by the *avgClusterednessClusters* numeric state variable. The probability of the *avgClusterednessClusters* to decrease under a threshold value (in our case 0.5) during the time interval [0, 50] is less than 0.05. Note that *avgClusterednessClusters* could be replaced by any other numeric state variable representing the concentration of a species/protein. Therefore our approach can be employed to reason about both spatial and non-spatial properties, and how changes of non-spatial properties reflect on the spatial properties and vice versa.

PBLSTL: $P \leq 0.05 [F [0, 50] (\{avgClusterednessClusters\} < 0.5)].$

12. **Natural language:** Similarly the number of clusters is expected to decrease and remain throughout the entire simulation less than 5 with probability greater than 0.75. The reason for this is that simulations start with multiple small clusters which are then expected to merge and form larger clusters close to the area where the chemical attractant concentration is highest.

PBLSTL: $P > 0.75 [F [0, 200] (count(clusters) \leq 5 \wedge G [0, 200] (count(clusters) \leq 5))].$

13. **Natural language:** The chemical gradient is distributed such that the areas of approximately equal chemical concentration have a circular/ring shape. Therefore the shape of at least one aggregated cells cluster should be eventually circular with probability greater or equal to 0.6.

PBLSTL: $P \geq 0.6 [F [0, 200] (count(filter(clusters, (circleMeasure > triangleMeasure) \wedge (circleMeasure > rectangleMeasure))) > 0)].$

14. **Natural language:** Finally the probability of the average clusters' density to never oscillate is less than 0.1. Oscillations of the density are expected because sometimes cells pile up.

PBLSTL: $P < 0.1 [\sim ((F [0, 200] ((count(clusters) > 0) \wedge (d(avg(clusters, density)) > 0)) \wedge F [0, 200] ((d(avg(clusters, density)) < 0)))) \vee (F [0, 200] ((count(clusters) > 0) \wedge (d(avg(clusters, density)) < 0)) \wedge F [0, 200] ((d(avg(clusters, density)) > 0))))].$

4.3.4 Model checking

Each PBLSTL statement, stored in a separate input file, was individually evaluated against the STML dataset 500 times using the frequentist statistical model checking approach. The output of the statistical analysis of the model checking results corresponding to each PBLSTL statement is reported in Table 4.2.

Table 4.2: Model checking statistical analysis results for the chemotaxis case study. Entries in the “id” column represent the numeric identifiers associated with each PBLSTL statement. The “% true PBLSTL” column describes what percentage of the 500 executions concluded that the PBLSTL statement is true. “#total STML” represents the total number of STML files evaluated for the PBLSTL statement; columns “#true STML” and “#false STML” represent the number of STML files for which the PBLSTL statement was evaluated true, respectively false. “Execution time” presents the average model checking execution time for each PBLSTL evaluation using the “minutes:seconds” format. “ μ ” and “ σ ” represent the mean and standard deviation.

id	% true PBLSTL	#total STML		#true STML		#false STML		Execution time	
		μ	σ	μ	σ	μ	σ	μ	σ
9	100	28	0	28	0	0	0	0:22.04	0:0.13
10	100	14	0	14	0	0	0	0:11.50	0:0.08
11	100	58	0	0	0	58	0	0:44.87	0:0.44
12	100	10.96	0.18	10.92	0.37	0.03	0.18	0:9.27	0:0.16
13	95.6	17.04	73.33	9.57	40.67	7.46	32.73	0:13.73	0:55.45
14	100	28	0	0	0	28	0	0:22.10	0:0.20

Similarly to the phase variation case study there are fluctuations in the evaluation results of some PBLSTL statements. Moreover the number of required STML files to reach a conclusion differs depending on the specificity of the logic statement and the distribution of PBLSTL truth evaluations. In contrast to the phase variation case study for many PBLSTL statements the variation in the number of required STML files, and the number of true and false STML evaluations is equal to zero. Furthermore although the average number of required STML files for the evaluation of a PBLSTL statement (≈ 26) is less than for the phase variation case study (≈ 171.47), the average execution time is higher (chemotaxis: 20.585s, phase variation: 7.6175s). The reason for this is that most of the execution time of the model checker is spent on I/O operations. Thus the execution time depends on both the number and size of STML files which are read into memory. The average STML file size for the phase variation case study is 64759.2 bytes, respectively 1397460 bytes for the chemotaxis case study. Thus the ratio between the file size for the phase variation and chemotaxis case study is 0.04, but the ratio between their average execution times is only 0.37.

Finally the average execution time for the entire specification is computed as

the sum of all average execution times (see Table 4.2, column 9):

$$\begin{aligned}\text{Execution time}_{\text{specification}} &= 0 : 22.04 + 0 : 11.50 + 0 : 44.87 + 0 : 9.27 \\ &\quad + 0 : 13.73 + 0 : 22.10 \\ &= 02 : 3.51 \text{ (minutes:seconds).}\end{aligned}$$

Similarly to the phase variation case study evaluating the specification comprising all PBLSTL statements in the same model checker run leads to a decrease in the execution time. The average execution time recorded for the entire chemotaxis specification considering 500 runs of the model checker was 0:56.18 (minutes:seconds) i.e. less than 50% of the average execution time when each PBLSTL statement was evaluated separately (02:3.51 minutes:seconds).

4.4 Discussion

The multidimensional spatio-temporal model checking methodology is an extension of the existing model checking approaches because it enables the validation of models with respect to (clusters of) spatial entities and how their properties change over time. The ability to reason about spatial entities and the interactions between them proves useful for the automatic *in silico* validation of complex spatio-temporal models. Stochastic biological systems are represented as SSpDESSs and the formal specification is encoded in PBLSTL. The model checking approach has been implemented in the model checker Mudi which is made freely available via the official web page <http://mudi.modelchecking.org>.

4.4.1 Supported modelling formalisms

The presented methodology and the model checker Mudi can be employed to validate computational models encoded using various high-level modelling formalisms because only the model simulation output and not the computational model used to produce it is considered.

To illustrate the generalizability of our approach the computational model for the phase variation case study was formalised as a Coloured Stochastic Petri Net, and the computational model for the chemotaxis case study as a Cellular Potts model integrated with a system of partial differential equations.

Although Mudi is not dependent on the model type it does place a restriction on the simulation output format. All time series data need to be translated to the standard data representation format STML. In our approach this conversion is

carried out automatically by scripts calling the parameterised region (e.g. phase variation) and cluster (e.g. chemotaxis) detection mechanisms.

4.4.2 Spatio-temporal analysis based on image processing

The main reason for choosing image processing functions for detecting and analysing spatial entities in time series data is that images could be generated from *in silico* simulations but also recorded during wet-lab experiments. Therefore our methodology could be potentially used in the future to automatically determine if certain spatio-temporal properties hold for both *in silico* and *in vitro* generated datasets.

Quantifying how many logic statements hold for computational models vs wet-lab datasets could prove to be useful as a measure of similarity/fitness and therefore be employed in automatic model construction and/or parameter estimation/synthesis algorithms.

Although image processing functions were employed here to detect and analyse spatial entities in time series data the system was designed in a modular fashion such that the model checker Mudi (and the associated binary) is decoupled from the region/cluster detection mechanism (and its implementation). Thus potential users of the model checker could extend our implementation with their own customized spatial entity detection and analysis modules.

4.4.3 STML files generated on demand

In addition Mudi supports validating models based on pre-generated STML files (e.g. phase variation) or it can generate STML files on demand (e.g. chemotaxis). In case STML files are generated on demand a user-defined script calling the model simulator needs to be made available. For the chemotaxis case study a Bash script was created to execute the Linux version of the Morpheus model simulator and translate the simulation output to STML. Although writing scripts for the integration of Mudi with various model simulators requires expert knowledge, the scripts, if designed properly, need to be potentially written only once.

4.4.4 Supported model checking algorithms

The efficiency and complexity of the methodology was illustrated for the phase variation and chemotaxis case studies by employing only the frequentist statistical model checking algorithm. However Mudi comprises both Bayesian and frequentist,

estimate and statistical hypothesis testing based model validation approaches. Depending on the availability of prior knowledge and the preferred method to formulate the model validity problem different algorithms could be used.

4.4.5 Scalability

The scalability of the methodology depends on the size and representation of the modelled system. An increase in the size of the system will negatively impact the model simulation time directly, and the spatio-temporal analysis and the evaluation of logic properties indirectly. The rate at which the model simulation time changes, with respect to the system size, can vary considerably depending on the employed model representation and simulation algorithm. For instance the systems considered by the phase variation and chemotaxis case studies were of similar size (discretised space of size 101×101 for phase variation, and 100×100 for chemotaxis) and complexity, but their simulation time was significantly different (average model simulation time was 50 minutes for phase variation, and 5 seconds for chemotaxis). In contrast both the spatio-temporal analysis and evaluation of logic properties depend on the size of the simulation traces, and not the models used to produce them. Therefore they are expected to scale well (polynomially) with respect to the size of the system. To conclude, one potential bottleneck, if any, for the scalability of the methodology is the model representation and/or simulation algorithm, and not the model validation.

4.4.6 Limitations

In spite of the above described features our approach has the following limitations, which will be addressed in Chapter 5.

First of all the collections of spatial entity types (e.g. regions) and properties (e.g. area) considered are hardcoded into the methodology and the model checker Mudi. Therefore the current version of the model checker cannot be employed for validating models which correspond to other spatial entity types (e.g. 3D structure) and properties (e.g. volume, 3D shape). For instance adapting the methodology to the full 3D scenario would require changing the definition of SSpDESSs such that spatial state variables are evaluated to three- instead of two-dimensional arrays of real values, defining a set of 3D specific spatial properties, including them in the logic PBLSTL and developing algorithms for automatically extracting such spatial properties from 3D images.

Secondly the presented methodology is limited to spatio-temporal uniscale models i.e. it assumes that all spatial properties correspond to the same spatial

scale. However for real-life applications there is a need to build and integrate models across multiple temporal and/or spatial scales which are not covered here. Multiple spatial scale models are not currently supported because the methodology does not include a mechanism to explicitly distinguish between spatial patterns from different scales.

Finally our approach has been validated only on simulated data but it should be applicable to real-life datasets as well. Moreover the usefulness of our methodology was illustrated only on biological case studies. However there is nothing inherent to the methodology which limits it to the biological and/or medical scenarios. Therefore one potential direction for future work is to apply this approach to non-biological case studies as well in an attempt to test its applicability limits and/or discover new features which should be included.

Summary

In this chapter the efficiency and applicability of the multidimensional spatio-temporal model checking methodology was assessed against two biological case studies encoding phase variation in bacterial colony growth and the chemotactic aggregation of cells. The conclusions drawn were that the methodology is general because it can be employed for computational models encoded using various high-level modelling formalisms, it employs spatio-temporal analysis methods which can be applied to time series data potentially originating from outside the *in silico* environment, and it supports both Bayesian and frequentist model checking approaches. Conversely one of the main limitations of the methodology is that the collections of spatial entity types and properties considered are fixed. Therefore the methodology cannot be employed in its current form for case studies in which other spatial entity types (e.g. 3D structure) and properties (e.g. volume) are relevant. Moreover the methodology is currently limited to uniscale computational models and therefore does not enable reasoning about how properties corresponding to biological subsystems from different scales relate to each other. Both of these limitations are addressed in Chapter 5.

CHAPTER 5

Multiscale multidimensional spatio-temporal meta model checking

Introduction

In this chapter the multidimensional spatio-temporal model checking methodology and implementation are extended such that they can be employed to validate both uniscale and multiscale computational models of biological systems with respect to case study specific spatial entity types and measures. The resulting approach is called multiscale multidimensional spatio-temporal meta model checking and is described and compared with the multidimensional model checking approach throughout the chapter. Related approaches for reasoning about how systems evolve over space, time and across multiple scales are described in the end.

5.1 Multiscale computational models of biological systems

Most of the existing computational models of biological systems are uniscale and therefore abstract away all biologically relevant details from more fine- and/or coarse-grained levels of organization (Sloot and Hoekstra, 2010). The main reason for this is that by minimizing the amount of details included in the model its complexity, simulation and analysis times are reduced. However the main disadvantage of uniscale computational models is that they do not enable gaining a truly systems level understanding of how biological systems function (Dada

and Mendes, 2011) i.e. how changes at fine-grained scales are responsible for the behaviours observed at coarse-grained scales and vice versa, which is one of the main aims of systems biology.

To overcome this limitation multiscale computational models of biological systems need to be developed instead (Schnell et al., 2007).

The importance of multiscale computational models has been recognized at an international level as shown by the large number of active multiscale modelling projects in the European Union and the United States alone, which has reached at least a few hundred in 2014 and is currently following an increasing trend (Groen et al., 2014), and by the 2013 Nobel prize in chemistry awarded to Martin Karplus, Michael Levitt and Arieh Warshel for their contributions to the development of multiscale computational models of complex chemical systems (Thiel and Hummer, 2013).

The minimum requirements for a model to be considered multiscale are (Bernard, 2013):

- The model covers two or more spatial and/or temporal scales;
- There is interaction between scales.

When studying biological systems the spatial and/or temporal scales considered usually correspond to a subset of the following ten levels of biological organization (Southern et al., 2008) (ordered from fine- to coarse-grained):

1. **Quantum:** Modelling electron-electron interactions.
2. **Molecular:** Representing the interactions between atoms (and ions) of interest.
3. **Macro-molecular:** Considering the interactions between several molecules.
4. **Subcellular:** When the number of molecules/particles considered is large it is too computationally expensive to simulate the interactions between them explicitly. The entire process can be modelled as a single continuum capturing how the average number of molecules/particles changes over time. The natural upper bound of this continuum is the cell membrane.
5. **Cellular:** Cells are the basic structural and functional component of an organism and lie at the interface between most micro- and macro-scale biological processes. Therefore this is the level from which most middle-out multiscale modelling integration procedures start.

6. **Tissue:** Modelling how large groups of connected cells of the same type perform a specific function (e.g. myocardium).
7. **Organ:** Integrating multiple tissue models of potentially different types into a discrete entity performing a function or group of functions (e.g. heart). Such models usually account for the explicit geometry of the organ.
8. **Organ system:** Representing a group of organs which perform a common function together (e.g. cardiovascular system).
9. **Organism:** Modelling an individual life form (e.g. human).
10. **Environment:** Considering the external factors (e.g. temperature, humidity) and their influence on the development of the organism.

Depending on the organism considered and its inherent complexity (e.g. lower vs. higher organisms) some of these levels might not be present. For instance lower organisms such as bacteria do not have organs, whereas higher organisms such as primates do. Moreover the spatial and temporal scales corresponding to each level of biological organization can vary significantly depending on the biological system considered. For instance the spatial and temporal scales associated with the organism level of organization are much smaller for a mayfly (i.e. $\approx 10^5$ s, $\approx 10^{-3}$ m) than for a human (i.e. $\approx 10^9$ s, $\approx 10^0$ m).

The construction of a multiscale computational model usually starts from the level of biological organization where the most data and knowledge are available. Once a single scale model is built and validated, the construction continues with the integration of models from the subsequent levels of organization, which are either above or below depending on the chosen model construction strategy (i.e. top-down, bottom-up or middle-out). Integrating computational models across scales represents one of the biggest challenges of multiscale modelling (Dada and Mendes, 2011; Groen et al., 2014). Several reasons for this are:

- The uniscale computational models considered have been potentially encoded using different formalisms and their integration is not straightforward;
- The complexity of the multiscale model could increase nonlinearly when integrating multiple uniscale models to the point where model simulations and/or parameterizations cannot be executed in reasonable time;
- Quantifying the magnitude of errors in the multiscale model can prove challenging especially when the submodels have been developed considering different levels of approximation (Yang, 2013).

In order to tackle these challenges and enable the systematic construction of multiscale models there is a need to develop a generic multiscale modelling methodology which will be adopted by most of the scientific community (Hoekstra et al., 2014).

Although such a generic community-wide adopted approach does not yet exist various multiscale modelling approaches have been developed for computational models of biological systems. They are either tailored to a particular biological problem (i.e. problem specific) or generic (i.e. problem independent). An example of a problem specific modelling approach for cancer systems biology is described by Chaudhary et al. (Chaudhary et al., 2013). Conversely some of the most employed problem independent multiscale modelling approaches are described in Table 5.1.

Table 5.1: Several of the most employed problem independent multiscale modelling approaches for constructing computational models of biological systems. For each problem independent multiscale modelling approach considered the table columns record (from left to right) the name of a corresponding software tool, description, supported model types and references.

Software	Description	Model types	Ref.
Chaste	A generic open source multiscale modelling and simulation framework for biological and physiological problems. The current version of the framework contains two modules, namely the Cardiac and the Cell-based module.	Ordinary/partial differential equations (ODE/PDE), rule-based models, Cellular Potts models (CPM), cellular automata, lattice-free models	(Mirams et al., 2013)
Coloured Stochastic Multilevel Multiset Rewriting (CSMMR) model simulator	A multilevel multiset rewriting modelling approach which enables the construction of computational models with parameters, dynamic compartments and multilevel compartmental structures.	CSMMR models	(Oury and Plotkin, 2011)
Compu-Cell3D	A multiscale modelling framework for representing cellular (using Cellular Potts models) and subcellular behaviours (when interfaced with numerical solvers such as BionetSolver).	CPMs, model types (e.g. ODE) supported by various numerical solvers	(Swat et al., 2012)
FLAME	A generic multi-agent modelling platform employed, amongst others, to construct a multiscale 3D model of the epidermis (Adra et al., 2010).	Agent-based models (ABM)	(Kiran et al., 2010)

Software	Description	Model types	Ref.
JAMES II	A multilevel rule-based modelling framework developed to support the construction of computational models of cell biological systems which span multiple levels of organization (Helms et al., 2014).	ML-Rules models	(Maus et al., 2011)
ManyCell	A multiscale cellular modelling environment encoding cells as agents, and subcellular processes as systems of ODEs that are solved using COPASI Web Services (Hoops et al., 2006).	ABMs, ODEs	(Dada and Mendes, 2012)
MOBI and PK-Sim	Commercial multiscale modelling tools employed for developing physiologically-based pharmacokinetic whole-body models.	ODEs, metabolic network models simulated using Dynamic Flux Balance Analysis	(Krauss et al., 2012)
Morpheus	A multiscale cell-based modelling and simulation environment which enables integrating Cellular Potts (for cell behaviour) with ordinary, stochastic and delay differential equation (DDE) based models. Reaction-diffusion systems are also supported and encoded using PDEs.	CPMs, ODEs, DDEs, PDEs	(Starruß et al., 2014)
Multiscale Modelling and Simulation Framework	A domain-independent multiscale modelling framework based on complex automata (Hoekstra et al., 2007), validated against case studies from different domains of science (Borgdorff et al., 2014a). Single scale models can be encoded using different modelling formalisms, are integrated according to the Multiscale Modelling Language (Falcone et al., 2010) specification, and simulated (in a distributed fashion) using the MUSCLE 2 (Borgdorff et al., 2014b) model coupling and simulation library; see (Caiazzo et al., 2011) for an illustrative biomedical application.	Modelling formalism independent	(Chopard et al., 2014)
NetLogo	A multi-agent modelling environment which can be integrated with deterministic continuous (ODE) models via a Matlab extension called MatNet. Illustrative biomedical examples include a model of acute inflammation (An, 2008) and <i>Pseudomonas aeruginosa</i> biofilm formation (Biggs and Papin, 2013).	ABMs, ODEs	(Wilensky, 2015)

Software	Description	Model types	Ref.
Open-CMISS	An open source multiscale modelling framework implemented in Fortran which supports models encoded in Cell Markup Language (CellML) (Lloyd et al., 2004) and Field Markup Language (FieldML) (Christie et al., 2009), standard model representations developed within the VPH and Physiome projects.	CellML models, FieldML models	(Bradley et al., 2011)
PhysioDesigner	A multilevel modelling framework based on the Physiological Hierarchy Markup Language (PHML), a standard modelling language for the integration of single scale models.	PHML models	(Asai et al., 2014)
Snoopy	A unified Petri nets based modelling framework employed to construct both uniscale (Pârvu et al., 2015) and multiscale(Gao et al., 2013; Liu and Heiner, 2013) computational models of biological systems.	Qualitative, deterministic, stochastic and hybrid (coloured) Petri nets	(Heiner et al., 2012)

Using such modelling approaches various multiscale computational models of biological systems have been constructed, covering different organisms (e.g. microorganisms (Biggs and Papin, 2013), plants (Grafahrend-Belau et al., 2013), humans (Krauss et al., 2012)), organ systems (e.g. cardiovascular system (Caiazzo et al., 2011; Formaggia et al., 1999; Laganà et al., 2005), digestive system (Du et al., 2013a; Graudenzi et al., 2014), nervous system (Bouteiller et al., 2011)) and diseases (e.g. thrombus formation (Xu et al., 2010), cancer (Deisboeck et al., 2011; Masoudi-Nejad et al., 2014), Crohn's disease (Dwivedi et al., 2014)).

To use results generated by multiscale computational models outside the *in silico* environment the models need to be validated first. However generic methodologies for multiscale model validation and error quantification have not yet been developed (Hoekstra et al., 2014). Moreover validating multiscale computational models against real-life data is often not possible due to the lack of relevant information from and between all relevant levels of organization (Walpole et al., 2013).

5.2 Multiscale multidimensional spatio-temporal model checking workflow

To overcome these limitations a multiscale multidimensional spatio-temporal model checking methodology is defined which enables the validation of computational models of biological systems in the *in silico* environment with respect to both how numeric and spatial properties change over time and across multiple scales. In contrast to the multidimensional model checking approach described in Chapter 3 which assumes that the modelled systems cover a single scale, the multiscale approach enables reasoning about how changes from different scales relate to one another.

However similarly to the multidimensional model checking approach (see Chapter 3) the multiscale model checking methodology is general and supports computational models encoded using various modelling formalisms because it is not defined relative to models but to model simulation traces. Moreover due to the large state spaces usually associated with complex multiscale computational models of biological systems only approximate probabilistic model checking approaches (see Subsection 2.4.2, Table 2.1) are considered throughout.

The main contributions of this chapter are:

- Definition of multiscale stochastic spatial discrete-event systems (MSSpDES) as theoretical models for describing how a system evolves over time, space and multiple scales. MSSpDESs extend SSpDESs with multiscale architecture (*MA*) graphs that explicitly encode the hierarchical organization of multiscale systems, and a state variable scale and subsystem (*SVSS*) assignment function which maps state variables to scales and subsystems encoded as vertices in *MA* graphs (Section 5.3).
- A formal Probabilistic Bounded Linear Multiscale Spatial Temporal Logic (PBLMSTL) for encoding the multiscale multidimensional spatio-temporal specifications against which the models are validated. One of the most significant extensions of PBLMSTL with respect to PBLSTL are the mechanisms which enable explicitly distinguishing between state variables from different scales (Section 5.5).
- Generalization of the multiscale model checking approach such that it is independent of case study specific spatial entity types and corresponding properties; the generalized approach is called multiscale meta model checking (Section 5.7).

- Implementation of the multiscale multidimensional spatio-temporal meta model checking approach in the model checker Mule (<http://mule.modelchecking.org>) which is freely available online in binary and source code format, and as a Docker image. Mule is an extension of Mudi and therefore supports the same Bayesian and frequentist, hypothesis testing and estimate based approximate probabilistic model checking algorithms (Section 5.8).
- Definition of multiscale spatio-temporal analysis module as a multiscale extension of the spatio-temporal analysis module introduced in Section 3.4 which automatically detects and analyses regions (see Subsubsection 3.4.1.1) and clusters (see Subsubsection 3.4.1.2) from multiple scales. The output of this analysis module is formatted according to the constraints of the Multiscale Spatial Temporal Markup Language (MSTML) introduced here. MSTML adapts the STML standard representation format (see Subsection 3.4.3) to the multiscale scenario (Section 5.4).

The multiscale multidimensional spatio-temporal model checking workflow is depicted in Figure 5.1 and comprises the same steps as the multidimensional model checking approach but adapted to the multiscale context:

1. **Model construction:** Using biological observations and/or relevant references from the literature to construct the multiscale computational model.
2. **Multiscale spatio-temporal analysis:** Each time the model is simulated time series data are generated in which spatial entities from multiple scales are automatically detected and analysed. The output of the multiscale spatio-temporal analysis is formatted according to MSTML.
3. **Formal specification:** The specification of the system is mapped from natural (e.g. English) language into formal PBLMSTL statements.
4. **Model checking:** The model checker takes the processed time series data (formatted according to MSTML) and the PBLMSTL specification as input and decides if the model is valid relative to the specification using the model checking algorithm chosen by the user (e.g. frequentist statistical model checking). In case the model is invalid it is updated and validated again.

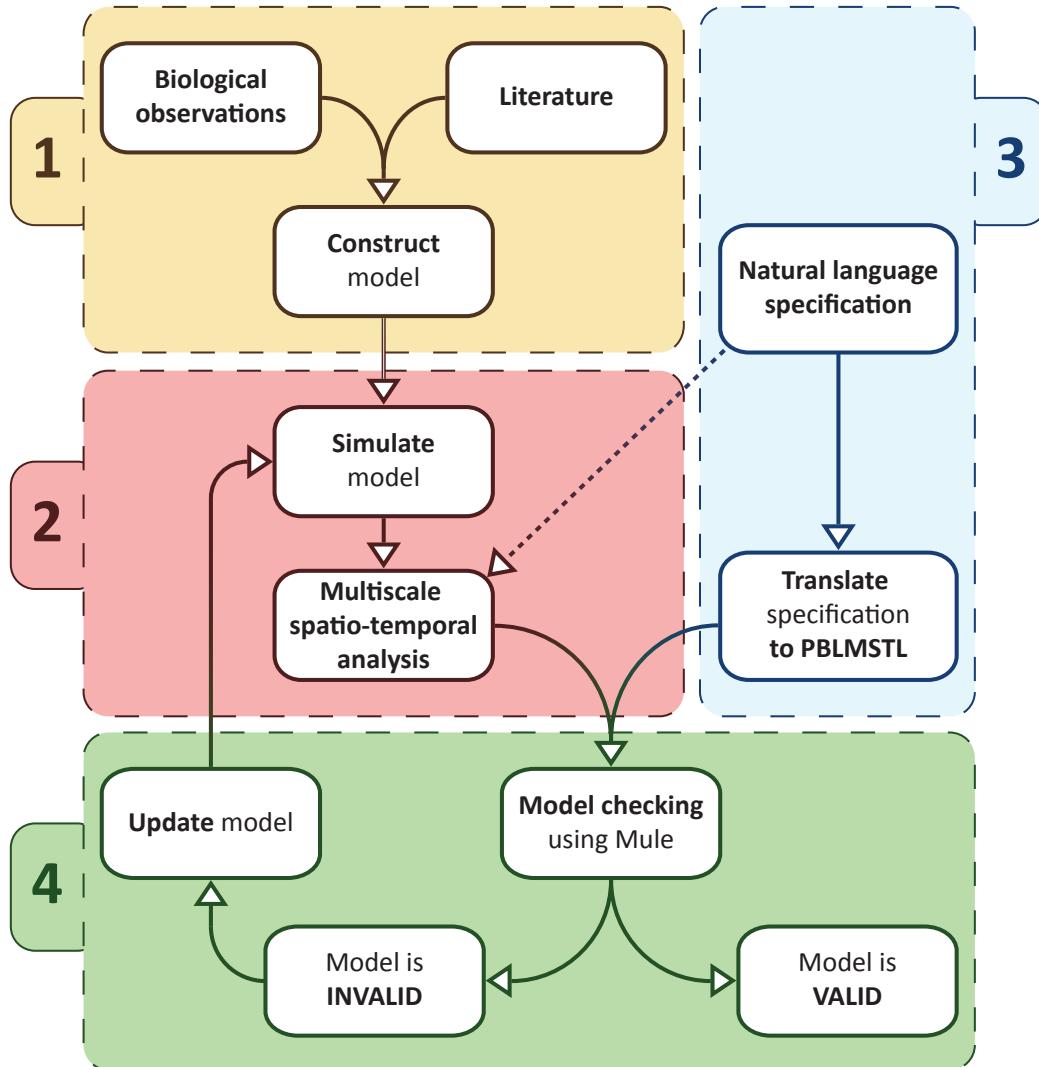


Figure 5.1: Multiscale multidimensional spatio-temporal model checking workflow. The first step (1) in the workflow is using biological observations and/or information from the literature to construct the multiscale computational model of a biological system. Next (2) the model is simulated to produce time series data in which spatial entities from multiple scales are automatically detected and analysed using the multiscale spatio-temporal analysis module. The scales, spatial entity types and spatial measures considered in the multiscale spatio-temporal analysis correspond to the scales, spatial entities and spatial measures described in the natural language specification; the dependency between the natural language specification and the multiscale spatio-temporal analysis is represented by a dashed arrow. Then (3) the specification against which the model is validated is translated manually from natural to formal PBLMSTL language. Finally (4) using the model checker Mule the model is validated relative to the given PBLMSTL specification. If the model is declared invalid then it is updated and steps (2) and (4) are repeated.

5.3 Model construction

The biological systems modelled here are assumed to be inherently complex, stochastic and to span multiple levels of organization (Southern et al., 2008), where each level of organization has an associated spatio-temporal scale. Moreover it is assumed in the following that biological systems which are multilevel (i.e. span multiple levels of biological organization) are inherently multiscale (i.e. span multiple spatio-temporal scales). Therefore the terms multiscale and multilevel,

respectively scale and level are used interchangeably in this thesis. Similarly to the multidimensional model checking approach in the following we assume that spatial domains are discretised and represented in pseudo-3D. However adapting the methodology to other numbers of dimensions requires minor changes which are described later. Moreover we consider here that the modelled systems are discrete-event systems i.e. they transition from the current to the next state only when an event occurs (e.g. a biochemical reaction).

Although SSpDESs (see Subsection 3.3.2) enable reasoning about how stochastic discrete-event systems change over time and space, they are not suitable for multiscale systems due to two main limitations. First of all in an SSpDES it is assumed that the size of the discretised spatial domain, which is encoded by the spatial value assignment function SpV , is the same for all spatial state variables. Secondly SSpDESs do not store any information regarding the scale to which each state variable corresponds. Therefore it is not possible to explicitly distinguish between state variables representing processes occurring at different levels of organization.

For addressing the first limitation the spatial value assignment function SpV in an SSpDES can be replaced with a collection of spatial value assignment functions

$$CSpV = \{SpV \mid SpV \text{ is defined identically as for an SSpDES}\},$$

where each spatial value assignment function $SpV \in CSpV$ corresponds to discretised spatial domains of a particular size $m \times n$. In order to extend the spatial representation from two to, for instance three dimensions, the codomain of each $SpV \in CSpV$ would be $\mathbb{R}^{m \times n \times p}$ instead of $\mathbb{R}^{m \times n}$.

5.3.1 Encoding the hierarchical system structure

To address the second limitation the hierarchical organization (Southern et al., 2008) of biological systems needs to be represented explicitly in the models. Throughout we assume that biological systems can be decomposed in a top-down manner from coarse-grained (e.g. population/organism) to fine-grained (e.g. intracellular/molecular) scales. Moreover at each scale (e.g. organ) one or multiple biological subsystems (e.g. heart and kidney) could be explicitly considered. The number and type of biological subsystems and/or scales considered differs depending on the addressed biological question.

To formally encode this hierarchical top-down structure a rooted (directed) tree (Bondy and Murty, 2010, Chapter 4) is employed called the multiscale architecture graph $MA = (V_{MA}, E_{MA})$, where V_{MA} represents the set of vertices

and E_{MA} the set of directed edges. Each vertex $v \in V_{MA}$ is encoded as a tuple $(sc, subsys)$ where $subsys$ represents a particular biological subsystem (e.g. heart) and sc its corresponding scale (e.g. organ). The root vertex (e.g. (organism, human)) corresponds to the most coarse-grained representation of the biological system considered. Directed edges $(v, v_i) \in E_{MA}$, $i = \overline{1, m}$, link the biological subsystem represented by vertex v to all its m constituent subsystems from finer-grained scales represented by vertices v_i .

The main reason for choosing the rooted directed tree representation is that its structure is inherently hierarchical and represents the organization of biological organisms. Moreover it can be used to encode how the behaviour of a subsystem from a higher scale is determined by the behaviour of one or multiple subsystems from lower scales.

Given a biological system BS the corresponding MA graph is constructed as follows; see Figure 5.2 for an illustrative example. First of all both BS and its corresponding subsystems relevant to the addressed biological question, together with their associated scale, are encoded as a set of vertices V_{MA} (see Figure 5.2(a)). Secondly starting from the vertices at the most coarse-grained scales directed edges are added towards their constituent subsystems. This step is repeated for all finer-grained scales considered until the entire hierarchical structure of BS is explicitly represented. Depending on the hierarchical representation of BS , vertices at the same depth in MA could correspond to different scales (see Figure 5.2(b)). The resulting MA graph should be a directed connected rooted acyclic graph, which means there exists a unique path from the root to every vertex, respectively MA should not contain any cycles.

However for some systems it is possible that the resulting MA graph is a directed acyclic graph but not a rooted directed tree because it contains vertices that have multiple incoming directed edges, which means they are part of more than one biological subsystem. For instance in Figure 5.2(b) the vertex (Tissue, CardiacMuscle) has two parent vertices, namely (OrganSystem, Cardiovascular) and (OrganSystem, Musculoskeletal). Vertices having more than one parent vertex in a directed acyclic graph are similar to classes which inherit from multiple parent classes (i.e. multiple inheritance) in object oriented programming. In our approach multiple inheritance is resolved by transforming the directed acyclic graph in a rooted directed tree. Every vertex with n incoming directed edges is replicated $n - 1$ times such that each instance of the vertex has only one incoming directed edge. In order to distinguish between the n vertex instances either the associated scale or subsystem label needs to be renamed accordingly (see Figure 5.2(c)).

The main advantages of resolving multiple inheritance by transforming a

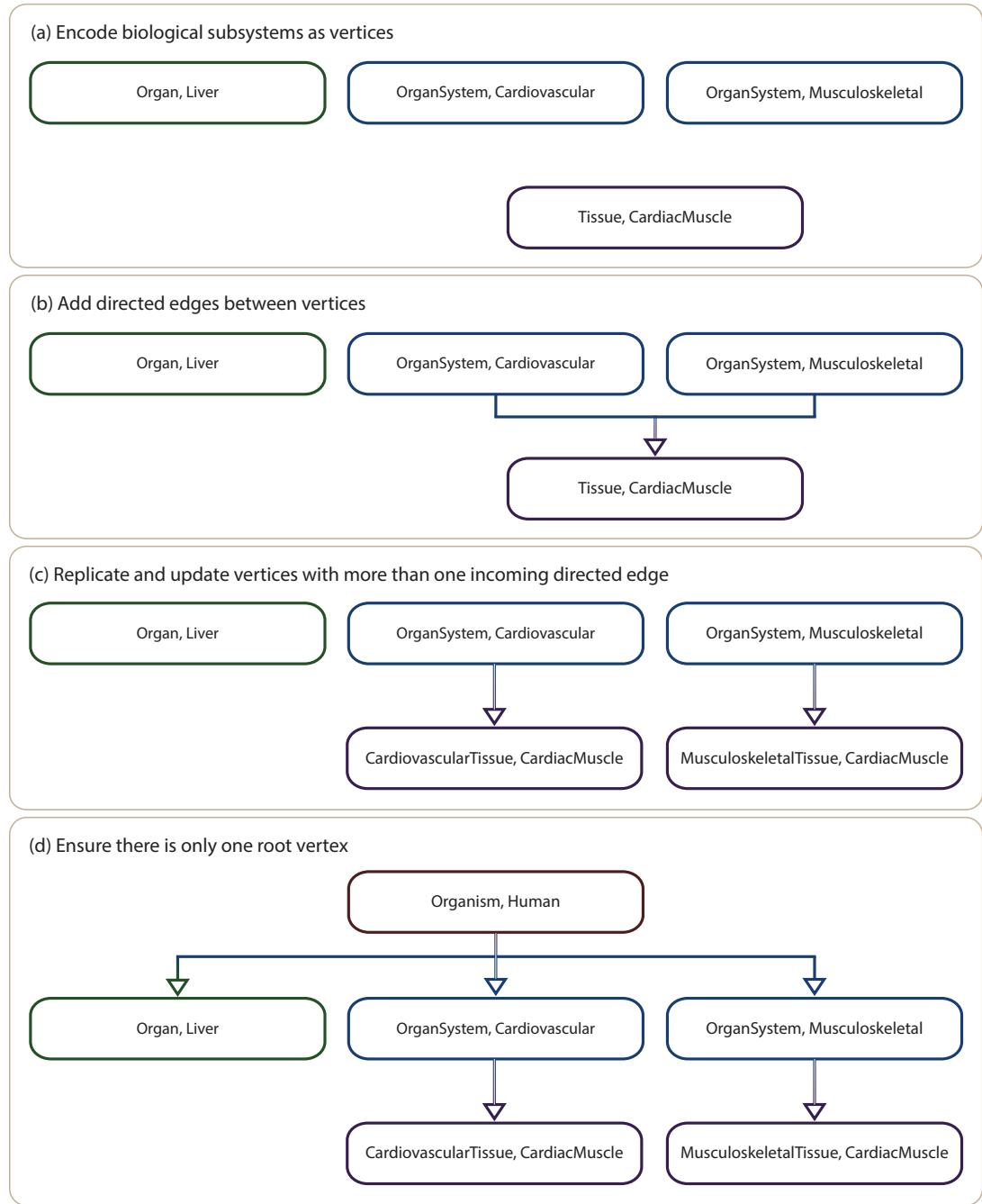


Figure 5.2: Illustrative example on how to construct a multiscale architecture graph. Let us assume that the biological subsystems considered are the human liver, the cardiac muscle tissue, and the cardiovascular and musculoskeletal organ systems. First of all (a) the biological subsystems and their associated scales are encoded as vertices (OrganSystem, Cardiovascular), (OrganSystem, Musculoskeletal), (Organ, Liver) and (Tissue, CardiacMuscle). Next (b) directed edges are added between each system and its constituent subsystems. The considered directed edges are ((OrganSystem, Cardiovascular), (Tissue, CardiacMuscle)) and ((OrganSystem, Musculoskeletal), (Tissue, CardiacMuscle)). The next step (c) is to eliminate vertices with multiple incoming directed edges from the graph. Therefore the vertex (Tissue, CardiacMuscle) is duplicated, each resulting instance (i.e. (CardiovascularTissue, CardiacMuscle) and (MusculoskeletalTissue, CardiacMuscle)) is renamed according to its corresponding parent vertex, and the directed edges are updated accordingly. The last step (d) is to ensure that the MA graph's root vertex is unique. For this purpose the (Organism, Human) vertex and the corresponding directed edges ((Organism, Human), (OrganSystem, Cardiovascular)), ((Organism, Human), (OrganSystem, Musculoskeletal)) and ((Organism, Human), (Organ, Liver)) are added to MA.

directed acyclic graph in a rooted directed tree are clarity and simplicity because each vertex in the tree will have only one parent, and implicitly the path from the root of the tree to each vertex is unique. Conversely the main disadvantage is that vertices need to be replicated and this leads to an increase in the size of the *MA* graph. Alternative approaches for resolving multiple inheritance which do not require increasing the number of vertices in the graph are graph-oriented and linear solutions (Snyder, 1986). In graph-oriented solutions the structure of the directed acyclic graph is preserved, whereas in linear solutions the directed acyclic graph is flattened and transformed into a linear sequence of vertices such that each vertex has at most one parent and/or child vertex. However whenever employing graph-oriented and linear solutions vertex descriptions are lengthier because the parent vertex (i.e. context) considered for each vertex needs to be defined explicitly.

Finally it may happen that in the updated *MA* graph multiple vertices v_1, v_2, \dots, v_p do not have any incoming directed edge and should therefore be labelled as root vertices (see Figure 5.2(c)). However in a rooted directed tree there can only be one root vertex. To address this issue an artificial root vertex v_{root} can be created and added to V_{MA} , whose corresponding scale is higher than that of v_1, v_2, \dots, v_p , and which will be connected via a directed outgoing edge with each vertex v_i , $i = \overline{1, p}$. Although the vertex v_{root} will not be explicitly considered in the model it ensures that the structure of *MA* is tree-like (see Figure 5.2(d)).

Considering that the resulting *MA* graph is a rooted directed tree, a strict partial order $<$ can be defined over the set of vertices V_{MA} , where $v_1 < v_2$, for all $v_1, v_2 \in V_{MA}$, if the unique path from the root to v_1 passes through v_2 . Similarly a non-strict partial order \leq can be defined over V_{MA} , where $v_1 \leq v_2$ if the unique path from the root to v_1 passes through v_2 , or $v_1 = v_2$. One of the main practical benefits of defining these partial orders is that they enable writing expressions for referring to all subsystems v_i of a system v_j ($v_i \leq v_j$), and all ancestor/parent systems v_k of a subsystem v_l ($v_l < v_k$) in a concise manner. Therefore such expressions could be employed to write shorter formal specifications against which MSSpDES models are validated.

To enable mapping both numeric and spatial state variables to particular scales and subsystems encoded as vertices in the *MA* graph the state variable scale and subsystem assignment function *SVSS* is introduced:

$$SVSS : NSV \cup SpSV \rightarrow V_{MA},$$

where *NSV* and *SpSV* are sets of numeric, respectively spatial state variables,

and V_{MA} is the set of vertices corresponding to MA .

5.3.2 Multiscale stochastic spatial discrete-event systems

Using the above notations we define multiscale stochastic spatial discrete-event systems (MSSpDES) as an extension of SSpDEs where the spatial value assignment function SpV is replaced by a collection of spatial value assignment functions $CSpV$, the multiscale architecture graph MA is defined to encode the hierarchical representation of the systems considered, and the state variable scale and subsystem assignment function $SVSS$ is introduced to associate state variables with particular scales and subsystems encoded as vertices in the MA graph.

Definition 13 Multiscale stochastic spatial discrete-event system (MSSpDES)

An MSSpDES \mathcal{M} is a 9-tuple $\langle S, T, \mu, NSV, SpSV, NV, CSpV, MA, SVSS \rangle$ where:

- $S, T, \mu, NSV, SpSV$ and NV have the same semantics as for an SSpDES (see Definition 5);
- $CSpV$ is the collection of spatial value assignment functions;
- MA is the multiscale architecture graph encoding the hierarchical structure of the system considered;
- $SVSS$ is the state variable scale and subsystem assignment function which associates state variables with particular scales and subsystems.

For explanatory purposes an illustrative example of a simple MSSpDES is given below.

Example 9 Illustrative example of an MSSpDES encoding the movement of a unicellular organism

Let us assume that we would like to model the movement of a unicellular microorganism in a fixed size environment (here a discretised rectangular grid of size 2×2). For simplicity the cell can only move up/down and left/right. In order to move the cell requires energy which it can chemically convert from an abstractly denoted nutrient A ; the chemical reaction for converting A to energy is $A \rightarrow Energy$. If nutrient A is available intracellularly then it can be converted directly to energy. Otherwise it has to be assimilated from the environment first;

the cell can only assimilate nutrients from the position of the discretised space which it currently occupies. The probability of the cell to move is 20%, 30% to convert A to energy and 50% to assimilate A from the environment.

Although the system considered in this example is much simpler than a real-life one, it suffices to illustrate the principles of abstractly representing an MSSpDES as follows.

The spatial state variables employed to describe the behaviour of the system are *Cell* — encoding the position of the cell in the discretised space, and *A_extracellular* — representing the distribution of nutrient A in the environment. Conversely the employed numeric state variables are *A_intracellular* — encoding the intracellular availability of nutrient A , and *Energy* — representing the cell's energy supply. The subsystems considered and their corresponding scales are energy production reaction network at the intracellular scale, microorganism at the cellular scale, and growth media at the environment scale. State variables associated with the energy production network (intracellular scale) are *A_intracellular* and *Energy*, respectively *Cell* with the microorganism (cellular scale), and *A_extracellular* with the growth media (environment scale). In the initial state of the system S_0 , depicted in Figure 5.3, the cell is positioned in the lower right part of the environment, *A_extracellular* is uniformly distributed across the entire environment (i.e. $A_{\text{extracellular}}[i, j] = 1$, for all $i, j = \overline{1, 2}$), and the initial levels of *A_intracellular* and *Energy* are zero.

State S_0				
Cell	$A_{\text{extracellular}}$	$A_{\text{intracellular}}$	Energy	
0 0 0 1	1 1 1 1		0 0	

Figure 5.3: Initial state of the MSSpDES encoding the movement of a unicellular organism. *Cell* and *A_extracellular* are the spatial state variables representing the position of the cell, respectively distribution of nutrient A in the environment. *A_intracellular* and *Energy* represent the intracellular availability of nutrient A , respectively energy.

Starting from the initial state S_0 the system can (in)directly transition to any of the states depicted in Figure 5.4.

Given that in S_0 the cell has no supplies of intracellular nutrient A or energy, the only possible action is for it to assimilate A from its environment ($S_0 \rightarrow S_1$, probability 100%). Since only one supply of nutrient A is available the only possible next action is to convert the newly gained intracellular A supply to energy ($S_1 \rightarrow S_2$, probability 100%). Once a supply of energy is available the cell can move either above ($S_2 \rightarrow S_4$) or to its left ($S_2 \rightarrow S_3$). The probability of moving to either of the neighbouring positions is therefore equal to 100% / 2 =

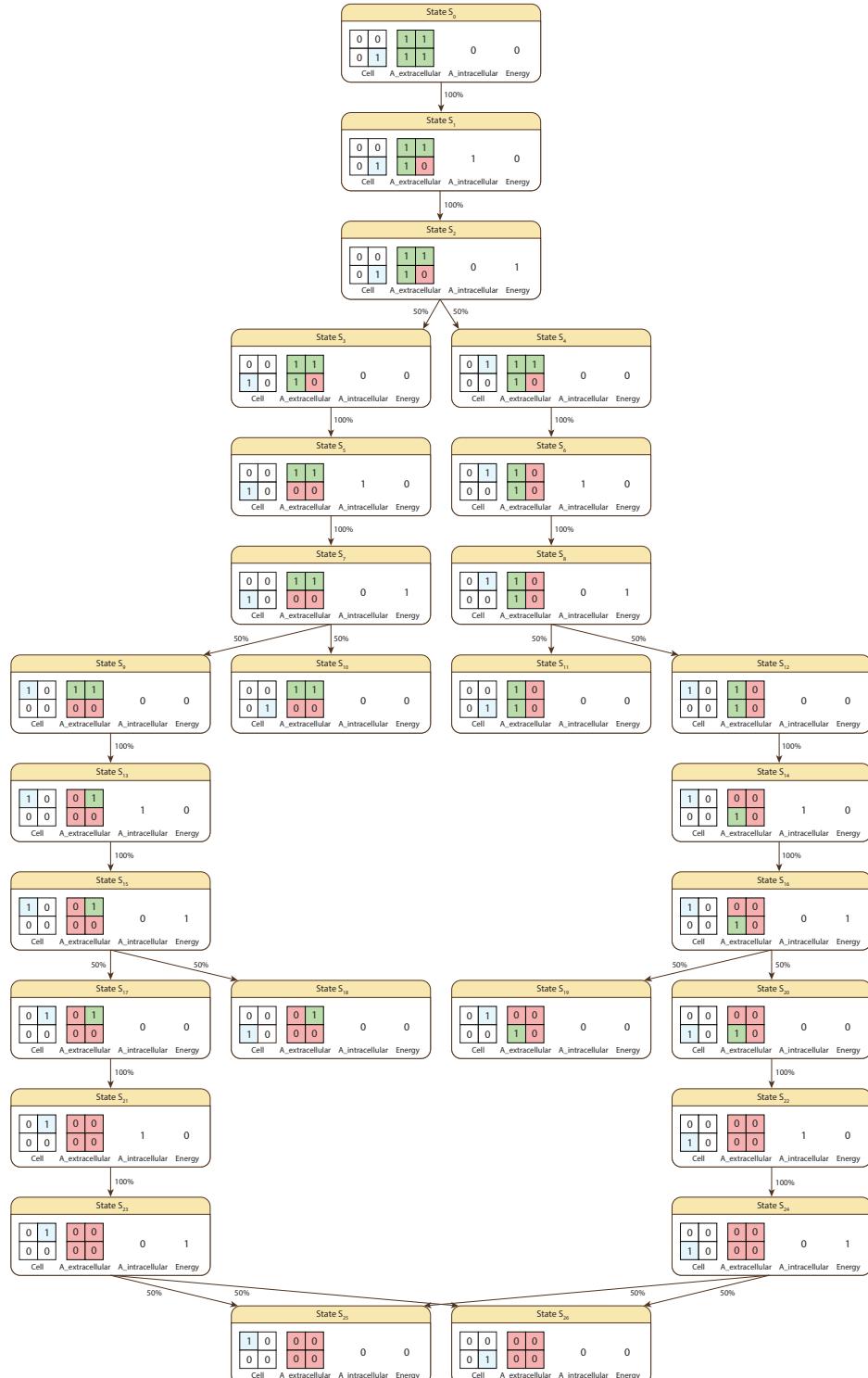


Figure 5.4: The state space of the MSSpDES encoding the movement of a unicellular organism i.e. all possible states which can be reached from the initial state S_0 . *Cell* and $A_{\text{extracellular}}$ are the spatial state variables representing the position of the cell, respectively distribution of nutrient A in the environment. $A_{\text{intracellular}}$ and *Energy* represent the intracellular availability of nutrient A , respectively energy. The percentage associated with the arrows connecting each pair of states represents the probability of transitioning from one state to the other. A high resolution digital copy of the image is made available at http://mule.modelchecking.org/illustrative_example_msspdes_system_state_space.tif?attredirects=0&d=1.

50%. Continuing from either state S_3 or S_4 the cell will try to assimilate new A nutrient supplies, which can be converted to energy and then used to move in the environment. This process is repeated multiple times until the cell reaches a state in which it has no A nutrients available extracellularly/intracellularly, and no supplies of energy (i.e. S_{10} , S_{11} , S_{18} , S_{19} , S_{25} , S_{26}). In such cases the cell becomes dormant and the system reaches its final state.

Using the notations above we formally define the corresponding MSSpDES model $\mathcal{M} = \langle S, T, \mu, NSV, SpSV, NV, CSpV, MA, SVSS \rangle$ as follows:

- $S = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{12}, S_{13}, S_{14}, S_{15}, S_{16}, S_{17}, S_{18}, S_{19}, S_{20}, S_{21}, S_{22}, S_{23}, S_{24}, S_{25}, S_{26}\}$.
- T is the transition rates matrix which records the probability of transitioning between any two system states $s_i, s_j \in S$. Since in our example T is a sparse matrix, only its non-zero entries are explicitly given below: $T[S_0, S_1] = 100\%$, $T[S_1, S_2] = 100\%$, $T[S_2, S_3] = 50\%$, $T[S_2, S_4] = 50\%$, $T[S_3, S_5] = 100\%$, $T[S_4, S_6] = 100\%$, $T[S_5, S_7] = 100\%$, $T[S_6, S_8] = 100\%$, $T[S_7, S_9] = 50\%$, $T[S_7, S_{10}] = 50\%$, $T[S_8, S_{11}] = 50\%$, $T[S_8, S_{12}] = 50\%$, $T[S_9, S_{13}] = 100\%$, $T[S_{12}, S_{14}] = 100\%$, $T[S_{13}, S_{15}] = 100\%$, $T[S_{14}, S_{16}] = 100\%$, $T[S_{15}, S_{17}] = 50\%$, $T[S_{15}, S_{18}] = 50\%$, $T[S_{16}, S_{19}] = 50\%$, $T[S_{16}, S_{20}] = 50\%$, $T[S_{17}, S_{21}] = 100\%$, $T[S_{20}, S_{22}] = 100\%$, $T[S_{21}, S_{23}] = 100\%$, $T[S_{22}, S_{24}] = 100\%$, $T[S_{23}, S_{25}] = 50\%$, $T[S_{23}, S_{26}] = 50\%$, $T[S_{24}, S_{25}] = 50\%$, $T[S_{24}, S_{26}] = 50\%$.
- μ is the function used to compute probabilities associated with cylinder sets $C(\sigma_{finite})$ defined over finite computation path prefixes σ_{finite} . The probability value associated with $C(\sigma_{finite})$ is computed by multiplying the probabilities of the state transitions encoded by σ_{finite} . For instance, if $\sigma_{finite} = \{S_0, S_1, S_2, S_3, S_5, S_7, S_{10}\}$ then $\mu(C(\sigma_{finite})) = P(S_0, S_1) \cdot P(S_1, S_2) \cdot P(S_2, S_3) \cdot P(S_3, S_5) \cdot P(S_5, S_7) \cdot P(S_7, S_{10}) = T[S_0, S_1] \cdot T[S_1, S_2] \cdot T[S_2, S_3] \cdot T[S_3, S_5] \cdot T[S_5, S_7] \cdot T[S_7, S_{10}] = 100\% \cdot 100\% \cdot 50\% \cdot 100\% \cdot 100\% \cdot 50\% = 25\%$.
- $NSV = \{A_intracellular, Energy\}$, and NV is the function used to compute the value of $A_intracellular$ and $Energy$ in a given state of a computation path.
- $SpSV = \{Cell, A_extracellular\}$, and $CSpV = \{SpV\}$ is the collection containing the spatial value assignment function SpV used to evaluate $Cell$ and $A_extracellular$ in a given state of a computation path.

- MA is the multiscale architecture graph depicted in Figure 5.5 encoding the hierarchical organization of the subsystems considered, namely the growth media (environment scale), the microorganism (cellular scale) and the energy production reaction network (intracellular scale).
- $SVSS$ is the state variable scale and subsystem assignment function which associates state variables to particular subsystems encoded as vertices in the MA graph. The values returned by $SVSS$ for the considered state variables are: $SVSS(A_{\text{intracellular}}) = (\text{Intracellular}, \text{EnergyProductionReactionNetwork})$, $SVSS(Energy) = (\text{Intracellular}, \text{EnergyProductionReactionNetwork})$, $SVSS(Cell) = (\text{Cellular}, \text{Microorganism})$, and $SVSS(A_{\text{extracellular}}) = (\text{Environment}, \text{GrowthMedia})$.

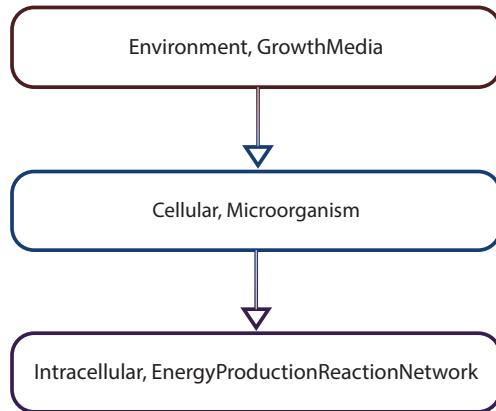


Figure 5.5: The multiscale architecture graph corresponding to the MSSpDES encoding the movement of a unicellular organism. Each vertex in the graph (e.g. (Environment, GrowthMedia)) corresponds to a subsystem (e.g. growth media) and its associated scale (e.g. environment). Directed edges between vertices (e.g. ((Environment, GrowthMedia), (Cellular, Microorganism))) indicate how one subsystem from a coarse-grained scale (e.g. (Environment, GrowthMedia)) can be decomposed in one or multiple subsystems from more fine-grained scales (e.g. (Cellular, Microorganism)).

In spite of the simplicity of the scenario described above the same model development principles apply for more complex multiscale real-life systems. However due to the inherent complexity of such systems the size of the state space is expected to be larger.

Remark 7. The probabilities employed in Example 9 were chosen for explanatory purposes and were not derived from experimental data or the literature. ■

The reason for defining MSSpDESs as extensions of SSpDESs is backwards compatibility. SSpDESs can be represented as MSSpDESs with a collection of spatial value assignment functions containing a single element, and a multiscale architecture graph containing only one vertex to which all state variables are

assigned using the state variable scale and subsystem assignment function. Due to this, multiple SSpDESSs can be easily integrated into a single MSSpDES by gathering all spatial value assignment functions into a single collection, constructing a corresponding multiscale architecture graph and mapping state variables to appropriate vertices in the graph, and adding interactions between submodels.

5.4 Multiscale spatio-temporal analysis

5.4.1 Detection and analysis of spatial entities from multiple scales

Time series data generated by MSSpDES model simulations record how values of both numeric and spatial state variables, potentially corresponding to multiple scales, change over time.

Similarly to the multidimensional scenario, to reason about how numeric properties (e.g. concentrations) or properties of all positions in a discretised spatial domain change over time the values of the corresponding numeric, respectively spatial state variables can be employed without further processing. However in order to reason about how properties of emergent spatial entities potentially occupying only a subset of positions in the discretised space change over time, there is a need for an additional processing step which automatically detects and analyses the spatial entities of interest considering the values of a given set of spatial state variables.

The spatio-temporal analysis module (see Section 3.4) defined for multidimensional spatio-temporal models enables automatically detecting and analysing how specific types of emergent spatial entities change over time. However one of its main limitations is that it does not enable to explicitly distinguish between spatial entities corresponding to different subsystems and/or scales.

To overcome this limitation the multiscale spatio-temporal analysis module extends the spatio-temporal analysis module with relevant pre- and post-processing steps (similarly to the MapReduce (Dean and Ghemawat, 2004) algorithm) as follows.

The pre-processing step is responsible for splitting up time series data corresponding to all spatial state variables in multiple time subseries, where each subseries corresponds to state variables from a single subsystem and scale.

During the main processing step each time subseries can be processed (in parallel) using the existing uniscale spatio-temporal analysis module for detecting, analysing and annotating spatial entities with their corresponding scale

and subsystem. Since the existing spatio-temporal analysis module is reused the collections of spatial entity types and measures considered are {clusters, regions} and {clusteredness, density, area, perimeter, distanceFromOrigin, angle, triangleMeasure, rectangleMeasure, circleMeasure, centroidX, centroidY}.

Finally the post-processing step is responsible for merging the results from all executions of the spatio-temporal analysis module such that spatial entities corresponding to the same time point are grouped together. A graphical depiction of the multiscale spatio-temporal analysis workflow is given in Figure 5.6.

Extending the spatial representation from two to, for instance three dimensions, requires employing appropriate types of spatial entities (e.g. 3D structure) and measures (e.g. volume), and updating the multiscale spatio-temporal analysis module accordingly. Moreover (the value corresponding to) each position in the discretised space is mapped to (the intensity of) a voxel, rather than a pixel in an image.

The output of the multiscale spatio-temporal analysis is time series data describing how the values of the considered spatial measures change over time for each detected spatial entity, scale and subsystem.

5.4.2 Multiscale Spatial Temporal Markup Language

The MSSpDES model simulation results are represented by time series data produced by the multiscale spatio-temporal analysis and time series data describing the evolution over time of numeric state variables values.

To easily share these results across the research community a standard portable representation format is required. STML (see Subsection 3.4.3) is a standard representation format which was designed for the same purpose but it is limited to uniscale spatio-temporal model simulations. Therefore it does not support associating state variables with particular scales and subsystems.

To overcome this limitation we define the Multiscale Spatial Temporal Markup Language (MSTML) as an extension of STML which enables mapping values of both numeric state variables and spatial entities to particular subsystems and their corresponding scales. From a structural point of view this is achieved by adding an optional *scaleAndSubsystem* attribute to both *spatialEntity* and *numericStateVariable* element definitions. The reason for making the attribute optional is to enable both multiscale and multidimensional uniscale datasets to be encoded in the MSTML format.

To easily adapt the multiscale model checking methodology to case study specific spatial entity types and measures — and we will come back to this later

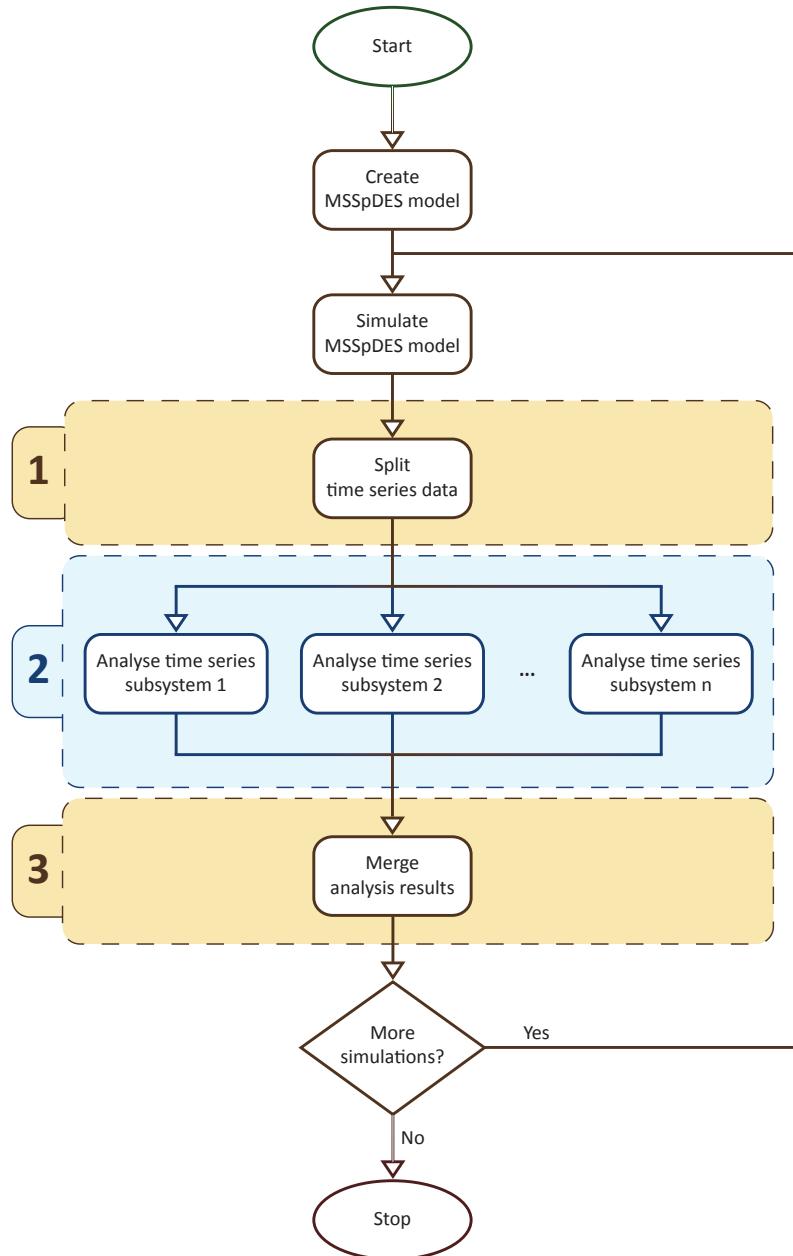


Figure 5.6: The multiscale spatio-temporal analysis workflow. An MSSpDES model of the considered system is constructed and simulated to generate time series data. The time series data subset corresponding to all spatial state variables is split during the pre-processing step into time subseries (1) such that each subseries corresponds to a single subsystem and its corresponding scale. The time subseries are then passed to the uniscale spatio-temporal analysis module (2) which automatically detects, analyses and annotates spatial entities with their corresponding scale and subsystem. The results of the uniscale spatio-temporal analysis are then merged during the post-processing step (3) such that spatial entities corresponding to the same time point are grouped together. If more simulations are required, a new time series dataset is generated, for which steps (1)–(3) are repeated.

in Section 5.7 — the following additional changes were considered when extending STML to MSTML:

- The `pseudo3D` element was removed from the schema, and the `type` attribute, now renamed to `spatialType`, was moved in the parent element `spatialEntity`. All child elements of the `pseudo3D` element are now

Listing 5.1: An example MSTML file recording multiscale spatio-temporal time series data

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <experiment>
3   <timepoint value="1">
4     <spatialEntity spatialType="cluster" scaleAndSubsystem="Organ.Liver">
5       <clusteredness>0.01</clusteredness>
6       <density>0.4</density>
7       <area>15</area>
8       <perimeter>28</perimeter>
9       <distanceFromOrigin>81</distanceFromOrigin>
10      <angle>10.5</angle>
11      <triangleMeasure>0.5</triangleMeasure>
12      <rectangleMeasure>1.0</rectangleMeasure>
13      <circleMeasure>0.1</circleMeasure>
14      <centroidX>703.4999</centroidX>
15      <centroidY>118.087</centroidY>
16    </spatialEntity>
17    <numericStateVariable scaleAndSubsystem="Cellular.Hepatocyte">
18      <name>dysfunction</name>
19      <value>0.1</value>
20    </numericStateVariable>
21  </timepoint>
22  ...
23 </experiment>
```

child elements of the `spatialEntity` element.

- The hierarchical structure of the `centroid` element was flattened. Thus the `centroid` element was replaced by its child elements `x` and `y`, now renamed to `centroidX`, respectively `centroidY`.

All rules and constraints for the structure of MSTML files are formalised in xsd files with the filename format `MSTML_LxVy.xsd` (i.e. Multiscale Spatial Temporal Markup Language Level x, Version y). The latest version of the MSTML format is made available at <http://mule.modelchecking.org/standards>, and an example of a MSTML formatted file is depicted in Listing 5.1.

For model checking purposes the number of MSTML files $\#MSTML$ generated for an MSSpDES model assuming fixed parameter values varies depending if the model is deterministic ($\#MSTML = 1$) or stochastic ($\#MSTML \geq 1$), and if the required level of confidence for the model checking result is high (e.g. 99%) or low (e.g. 70%).

To determine if a model is valid the model checker verifies if its behaviour captured by a corresponding set of MSTML files conforms to a given formal specification.

5.5 Formal specification

The temporal logic employed to write the formal specification needs to enable reasoning about how values of numeric state variables and/or spatial measures, which are the considered state variables, are expected to change over time and multiple scales.

Although BLSTL (see Subsection 3.5.1) was introduced for reasoning about how both numeric and spatial properties change over time, one of its main limitations is that it does not enable to explicitly distinguish between different scales. Therefore it is not possible to relate how changes at one scale reflect at another scale and vice versa.

5.5.1 Bounded Linear Multiscale Spatial Temporal Logic

To address the issue of relating changes between scales Bounded Linear Multiscale Spatial Temporal Logic (BLMSTL) is defined as a multiscale extension of BLSTL which enables to explicitly distinguish between state variables corresponding to different scales and subsystems.

Although, to the best of our knowledge, BLMSTL is the first formal logic defined to encode specifications of how numeric and/or spatial properties of multiscale computational models are expected to change over time, other types of languages have been developed previously to describe multiscale computational model simulations. For instance Helms et al. (Helms et al., 2012) introduced an instrumentation language for specifying what data should be observed during multiscale computational model simulation studies. The main difference between the instrumentation language introduced by Helms et al. and BLMSTL is that the former was mainly designed to describe what data should be collected for the entities in the computational model and/or corresponding computation algorithms, whereas the latter is designed to describe how the values of state variables in the computational model and/or emerging spatial structures are expected to change over time.

Statements which can be expressed in BLSTL, and implicitly BLMSTL, have been previously described in Subsection 3.5.1 and therefore will not be restated here. Instead only the (significant) changes of BLMSTL relative to BLSTL are described in natural language below.

First of all BLMSTL enables referring to scale specific state variables by explicitly associating to each state variable its corresponding scale and subsystem from the considered *MA* graph. This is achieved using either the notation “(scaleAndSubsystem = explicitScale1.explicitSubsystem1)” which reads state

variable corresponding to scale explicitScale1 (e.g. organ), and subsystem explicitSubsystem1 (e.g. heart), or the notation “(scaleAndSubsystem \asymp explicitScale2.explicitSubsystem2)”, $\asymp \in \{<, \leq, =, \geq, >\}$, which reads the collection of state variables whose scale and subsystem is \asymp explicitScale2 and explicitSubsystem2. In the latter case the partial ordering of the scales and subsystems is taken into account as defined by the *MA* graph of the corresponding MSSpDES model.

To reduce the ambiguity in formal BLMSTL specifications numeric state variables, which evaluate to single values, can be associated only with single scales and subsystems (i.e. using “ $=$ ”), whereas spatial measures, which evaluate to as many values as there are spatial entities, can be associated with one or multiple scales and subsystems (i.e. using $\asymp \in \{<, \leq, =, \geq, >\}$).

Secondly since BLMSTL enables distinguishing between state variables from different scales it is possible to define transfer functions over these state variables describing how changes from one scale reflect at another scale and vice versa. To encode such transfer functions, unary (e.g. square root) and binary (e.g. add) arithmetic functions can be employed. For instance if the value of a state variable sv_{cg} from a coarse-grained scale is equal to the arithmetic mean of four state variables $sv_{fg_1}, sv_{fg_2}, sv_{fg_3}, sv_{fg_4}$ from a more fine-grained scale, this can be written as $sv_{cg} = (sv_{fg_1} + sv_{fg_2} + sv_{fg_3} + sv_{fg_4})/4$; in BLMSTL “+” and “/” would be replaced by the arithmetic functions *add*, respectively *div*.

Moreover BLMSTL enables reasoning not only about the values of state variables corresponding to single time points but the distribution of values corresponding to multiple time points. This allows encoding transfer functions that describe how values corresponding to a single time point (e.g. and a coarse-grained scale) relate to the values corresponding to multiple time points (e.g. and a fine-grained scale), or vice versa.

Finally BLMSTL introduces set operators \setminus (difference), \cap (intersection) and \cup (union) which enable reasoning about multiple collections of spatial entities. For instance it is possible to describe (using union) how spatial entities either of type region or of type cluster, and potentially corresponding to different scales and subsystems, change over time.

As in the case of MSSpDESs, for clarity purposes, in BLMSTL we explicitly distinguish between numeric and spatial state variables. The main advantage of this is that state variable type specific functions can be defined. Conversely the main disadvantage is replication because the BLMSTL grammar needs to define structurally similar constructs once for numeric and once for spatial state variables. An alternative approach, which we do not consider here, is to employ a

single state variable type by converting all numeric into spatial state variables. The main advantage of this approach is that the number of constructs in the BLMSTL grammar would be reduced. Conversely the main disadvantage is the lack of clarity in BLMSTL statements because it would no longer be possible to determine directly from the BLMSTL statements syntax if a state variable inherently encodes a single (numeric) or multiple (spatial) real values.

A formal definition of the BLMSTL syntax and semantics, and corresponding illustrative examples are given next.

5.5.1.1 Syntax

To enable comparing BLMSTL with BLSTL, significant syntax changes introduced by the former relative to the latter are emphasized using bold text formatting below.

Definition 14 BLMSTL syntax

The syntax of BLMSTL is given by the following grammar expressed in BNF:

```

⟨logic-property⟩ ::= ⟨temporal-numeric-measure⟩ ⟨comparator⟩
    ⟨temporal-numeric-measure⟩
  | ⟨change-measure⟩(⟨temporal-numeric-measure⟩) ⟨comparator⟩
    ⟨temporal-numeric-measure⟩
  | ~⟨logic-property⟩
  | ⟨logic-property⟩ ∧ ⟨logic-property⟩
  | ⟨logic-property⟩ ∨ ⟨logic-property⟩
  | ⟨logic-property⟩ ⇒ ⟨logic-property⟩
  | ⟨logic-property⟩ ⇔ ⟨logic-property⟩
  | ⟨logic-property⟩ U[⟨unsigned-real-number⟩,⟨unsigned-real-number⟩]
    ⟨logic-property⟩
  | F[⟨unsigned-real-number⟩,⟨unsigned-real-number⟩] ⟨logic-property⟩
  | G[⟨unsigned-real-number⟩,⟨unsigned-real-number⟩] ⟨logic-property⟩
  | X ⟨logic-property⟩
  | X [⟨natural-number⟩] ⟨logic-property⟩
  | ⟨⟨logic-property⟩⟩

⟨temporal-numeric-measure⟩ ::= ⟨real-number⟩
  | ⟨numeric-state-variable⟩
  | ⟨numeric-statistical-measure⟩
  | ⟨unary-numeric-measure⟩(⟨temporal-numeric-measure⟩)

```

```

|  ⟨binary-numeric-measure⟩(⟨temporal-numeric-measure⟩,
  ⟨temporal-numeric-measure⟩)

⟨numeric-statistical-measure⟩ ::= ⟨unary-statistical-measure⟩(
  ⟨numeric-measure-collection⟩)
|  ⟨binary-statistical-measure⟩(⟨numeric-measure-collection⟩,
  ⟨numeric-measure-collection⟩)
|  ⟨binary-statistical-quantile-measure⟩(
  ⟨numeric-measure-collection⟩, ⟨real-number⟩)

⟨numeric-measure-collection⟩ ::= ⟨spatial-measure-collection⟩
|  [⟨unsigned-real-number⟩,⟨unsigned-real-number⟩]
  ⟨numeric-measure⟩

⟨numeric-measure⟩ ::= ⟨primary-numeric-measure⟩
|  ⟨unary-numeric-measure⟩(⟨numeric-measure⟩)
|  ⟨binary-numeric-measure⟩(⟨numeric-measure⟩,⟨numeric-measure⟩)

⟨primary-numeric-measure⟩ ::= ⟨numeric-spatial-measure⟩
|  ⟨real-number⟩
|  ⟨numeric-state-variable⟩

⟨numeric-spatial-measure⟩ ::= ⟨unary-statistical-measure⟩(
  ⟨spatial-measure-collection⟩)
|  ⟨binary-statistical-measure⟩(⟨spatial-measure-collection⟩,
  ⟨spatial-measure-collection⟩)
|  ⟨binary-statistical-quantile-measure⟩(⟨spatial-measure-collection⟩,
  ⟨real-number⟩)

⟨unary-statistical-measure⟩ ::= avg
|  count
|  geomean
|  harmean
|  kurt
|  max
|  median
|  min
|  mode
|  product
|  skew
|  stdev

```

```

|   sum
|   var

⟨binary-statistical-measure⟩ ::= covar

⟨binary-statistical-quantile-measure⟩ ::= percentile
|   quartile

⟨unary-numeric-measure⟩ ::= abs
|   ceil
|   floor
|   round
|   sign
|   sqrt
|   trunc

⟨binary-numeric-measure⟩ ::= add
|   div
|   log
|   mod
|   multiply
|   power
|   subtract

⟨spatial-measure-collection⟩ ::= ⟨spatial-measure⟩(⟨subset⟩)
|   ⟨unary-numeric-measure⟩(⟨spatial-measure-collection⟩)
|   ⟨binary-numeric-measure⟩(⟨spatial-measure-collection⟩,
⟨spatial-measure-collection⟩)

⟨spatial-measure⟩ ::= clusteredness
|   density
|   area
|   perimeter
|   distanceFromOrigin
|   angle
|   triangleMeasure
|   rectangleMeasure
|   circleMeasure
|   centroidX
|   centroidY

```

$\langle \text{subset} \rangle ::= \langle \text{subset-specific} \rangle$

- | $\text{filter}(\langle \text{subset-specific} \rangle, \langle \text{constraint} \rangle)$
- | $\langle \text{subset-operation} \rangle (\langle \text{subset} \rangle, \langle \text{subset} \rangle)$

$\langle \text{subset-specific} \rangle ::= \text{regions}$

- | clusters

$\langle \text{constraint} \rangle ::= \text{scaleAndSubsystem } \langle \text{comparator} \rangle$

$\langle \text{scale-and-subsystem} \rangle$

- | $\langle \text{spatial-measure} \rangle \langle \text{comparator} \rangle \langle \text{filter-numeric-measure} \rangle$
- | $\sim \langle \text{constraint} \rangle$
- | $\langle \text{constraint} \rangle \wedge \langle \text{constraint} \rangle$
- | $\langle \text{constraint} \rangle \vee \langle \text{constraint} \rangle$
- | $\langle \text{constraint} \rangle \Rightarrow \langle \text{constraint} \rangle$
- | $\langle \text{constraint} \rangle \Leftrightarrow \langle \text{constraint} \rangle$
- | $(\langle \text{constraint} \rangle)$

$\langle \text{filter-numeric-measure} \rangle ::= \langle \text{primary-numeric-measure} \rangle$

- | $\langle \text{spatial-measure} \rangle$
- | $\langle \text{unary-numeric-measure} \rangle (\langle \text{filter-numeric-measure} \rangle)$
- | $\langle \text{binary-numeric-measure} \rangle (\langle \text{filter-numeric-measure} \rangle, \langle \text{filter-numeric-measure} \rangle)$

$\langle \text{subset-operation} \rangle ::= \text{difference}$

- | intersection
- | union

$\langle \text{change-measure} \rangle ::= d$

- | r

$\langle \text{real-number} \rangle ::= \langle \text{unsigned-real-number} \rangle$

- | $\langle \text{sign} \rangle \langle \text{unsigned-real-number} \rangle$

$\langle \text{unsigned-real-number} \rangle ::= \langle \text{fractional-part} \rangle$

- | $\langle \text{fractional-part} \rangle \langle \text{exponent-part} \rangle$

$\langle \text{fractional-part} \rangle ::= \langle \text{digit-sequence} \rangle . \langle \text{digit-sequence} \rangle$

- | $. \langle \text{digit-sequence} \rangle$
- | $\langle \text{digit-sequence} \rangle .$
- | $\langle \text{digit-sequence} \rangle$

```

⟨digit-sequence⟩ ::= ⟨digit⟩
| ⟨digit⟩ ⟨digit-sequence⟩

⟨digit⟩ ::= 0
| 1
| 2
| 3
| 4
| 5
| 6
| 7
| 8
| 9

⟨natural-number⟩ ::= ⟨digit-sequence⟩
| + ⟨digit-sequence⟩

⟨exponent-part⟩ ::= e ⟨digit-sequence⟩
| E ⟨digit-sequence⟩
| e ⟨sign⟩ ⟨digit-sequence⟩
| E ⟨sign⟩ ⟨digit-sequence⟩

⟨sign⟩ ::= +
| -

⟨comparator⟩ ::= <
| <=
| =
| >=
| >

⟨numeric-state-variable⟩ ::=
⟨state-variable⟩ ⟨state-variable-scale-and-subsystem⟩

⟨state-variable⟩ ::= {⟨string⟩}

⟨state-variable-scale-and-subsystem⟩ ::= ε
| ⟨scaleAndSubsystem = ⟨scale-and-subsystem⟩⟩

⟨string⟩ ::= ⟨character⟩ | ⟨character⟩ ⟨string⟩

⟨character⟩ ::= based on the Unicode character set except “{” and “}”

```

$$\begin{aligned}
\langle \text{scale-and-subsystem} \rangle &::= \langle \text{primary-scale-and-subsystem} \rangle . \\
\langle \text{primary-scale-and-subsystem} \rangle &::= \langle \text{scale-and-subsystem-character} \rangle \\
&\quad | \quad \langle \text{scale-and-subsystem-character} \rangle \\
&\quad | \quad \langle \text{primary-scale-and-subsystem} \rangle \\
\langle \text{scale-and-subsystem-character} \rangle &::= \langle \text{basic-latin-script-character} \rangle \\
&\quad | \quad \langle \text{digit} \rangle \\
\langle \text{basic-latin-script-character} \rangle &::= a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \\
&\quad | \quad p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z \mid A \mid B \mid C \mid D \mid E \mid F \mid G \mid H \mid I \mid J \\
&\quad | \quad K \mid L \mid M \mid N \mid O \mid P \mid Q \mid R \mid S \mid T \mid U \mid V \mid W \mid X \mid Y \mid Z
\end{aligned}$$

Similarly to BLSTL the order of precedence of the operators is given by the definition of the BLMSTL syntax. In the absence of parentheses the logic expressions are evaluated from left to right.

5.5.1.2 Semantics

The semantics of BLMSTL is defined with respect to executions σ of an MSSpDES \mathcal{M} , where the meaning of the notations $|\sigma|$, $\sigma[i]$, σ^i and $\sigma(t)$ are the same as for BLSTL (see Subsubsection 3.5.1.2). Given an execution σ at state s the value of a numeric state variable nsv is computed using $NV(\sigma, s, nsv)$ and its associated scale and subsystem using $SVSS(nsv)$, respectively the value of a spatial state variable $spsv$ is computed using $SpV(\sigma, s, spsv)$ and its associated scale and subsystem using $SVSS(spsv)$.

Moreover in order to minimize the length of the semantics description the full symbol names specific to BLMSTL were replaced with shorter abbreviations as described in Table 5.2. To facilitate comparisons with BLSTL (see Table 3.1) symbol names which were introduced exclusively by BLMSTL are emphasized using bold text formatting.

Definition 15 BLMSTL semantics

Let $\mathcal{M} = \langle S, T, \mu, NSV, SpSV, NV, SpV, MA, SVSS \rangle$ be an MSSpDES and σ an execution of \mathcal{M} . The semantics of BLMSTL for σ is defined as follows:

- $\sigma \models tnm_1 \asymp tnm_2$ if and only if $tnm_1 \asymp tnm_2$, where tnm_1 and $tnm_2 \in \mathbb{R}$, and $\asymp \in \{<, \leq, =, \geq, >\}$;

Table 5.2: Translation of full BLMSTL symbol names to abbreviated forms. The left column contains the full BLMSTL symbol name. The right column contains the corresponding abbreviated form. Symbols which were introduced exclusively in BLMSTL and do not exist in BLSTL are highlighted using bold text formatting.

Full BLMSTL symbol name	Abbreviated BLMSTL symbol name
<code><logic-property></code>	ψ
<code><temporal-numeric-measure></code>	$t\text{nm}$
<code><numeric-statistical-measure></code>	$n\text{stm}$
<code><numeric-measure-collection></code>	$n\text{mc}$
<code><numeric-measure></code>	nm
<code><primary-numeric-measure></code>	$p\text{nm}$
<code><numeric-spatial-measure></code>	$n\text{spm}$
<code><unary-statistical-measure></code>	$u\text{stm}$
<code><binary-statistical-measure></code>	$b\text{stm}$
<code><binary-statistical-quantile-measure></code>	$b\text{stqm}$
<code><unary-numeric-measure></code>	$u\text{nm}$
<code><binary-numeric-measure></code>	$b\text{nm}$
<code><spatial-measure-collection></code>	$s\text{mc}$
<code><spatial-measure></code>	sm
<code><subset></code>	ss
<code><filter-numeric-measure></code>	$f\text{nm}$
<code><comparator></code>	\asymp
<code><change-measure></code>	cm
<code><real-number></code>	re
<code><scale-and-subsystem></code>	$s\text{csbsys}$
<code><state-variable></code>	sv
<code><numeric-state-variable></code>	nsv
<code><spatial-state-variable></code>	$spsv$

- $\sigma \models cm(nm1) \asymp nm2$ **if and only if** $|\sigma| > 1$ and $cm(nm1) \asymp nm2$, where $cm(nm1)$ and $nm2 \in \mathbb{R}$, and $\asymp \in \{<, \leq, =, \geq, >\}$;
- $\sigma \models \sim \psi$ **if and only if** $\sigma \not\models \psi$;
- $\sigma \models \psi_1 \wedge \psi_2$ **if and only if** $\sigma \models \psi_1$ and $\sigma \models \psi_2$;
- $\sigma \models \psi_1 \vee \psi_2$ **if and only if** $\sigma \models \psi_1$ or $\sigma \models \psi_2$;
- $\sigma \models \psi_1 \Rightarrow \psi_2$ **if and only if** $\sigma \models \sim \psi_1$ or $\sigma \models \psi_2$;
- $\sigma \models \psi_1 \Leftrightarrow \psi_2$ **if and only if** $\sigma \models \psi_1 \Rightarrow \psi_2$ and $\sigma \models \psi_2 \Rightarrow \psi_1$;
- $\sigma \models \psi_1 U[a, b] \psi_2$ **if and only if** there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi_2$, and for all $j, j \in [a, i)$, it holds that $\sigma(j) \models \psi_1$ with $a, b \in \mathbb{R}_+$;
- $\sigma \models F[a, b] \psi$ **if and only if** there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi$ with $a, b \in \mathbb{R}_+$;
- $\sigma \models G[a, b] \psi$ **if and only if** for all $i, i \in [a, b]$, it holds that $\sigma(i) \models \psi$ with $a, b \in \mathbb{R}_+$;
- $\sigma \models X \psi$ **if and only if** $|\sigma| > 1$ and $\sigma^1 \models \psi$;
- $\sigma \models X[k] \psi$ **if and only if** $|\sigma| > k$ and $\sigma^k \models \psi$ with $k \in \mathbb{N}$;

- $\sigma \models (\psi)$ if and only if $\sigma \models \psi$.

The tnm symbol represents the category of temporal numeric measures. Considering a model execution σ tnm is evaluated based on one of the definitions below:

- **Real number:** $tnm = re \in \mathbb{R}$;
- **Numeric state variable:** $tnm = nsdv$;
- **Numeric statistical measure:** $tnm = nstm$;
- **Unary numeric measure:** $tnm = unm(tnm')$, where tnm' is a temporal numeric measure;
- **Binary numeric measure:** $tnm = bnm(tnm', tnm'')$, where tnm' and tnm'' are temporal numeric measures.

The values of the unary (unm) and binary (bnm) numeric measures are computed in the same manner as for BLSTL (see Appendix C.2, Tables C.2 and C.3).

The category of numeric statistical measures is represented by the $nstm$ symbol. For a given model execution σ the $nstm$ symbol is evaluated considering one of the definitions below:

- **Unary statistical numeric measure:** $nstm = ustn(nmc)$, where nmc is a numeric measure collection;
- **Binary statistical numeric measure:** $nstm = bstn(nmc', nmc'')$, where nmc' and nmc'' are numeric measure collections;
- **Binary statistical quantile numeric measure:** $nstm = bstqm(nmc', re)$, where nmc' is a numeric measure collection, and re is a real number.

The value of unary statistical ($ustn$), binary statistical ($bstn$) and binary statistical quantile ($bstqm$) measures considering a collection of real values is computed similarly to how values of unary, binary, ternary and quaternary subset measures were computed in BLSTL (see Appendix C.3, Tables C.4, C.5, C.6 and C.7). The main difference between the BLMSTL statistical measures and BLSTL subset measures is that the former are defined considering generic collections of real numbers, whereas the latter were restricted to collections of real

numbers encoding the values of spatial measures evaluated against spatial entities collections. The matching between statistical and subset measures is done based on their name i.e. measures with the same semantics have identical names. Moreover by replacing each pair of (spatial measure, spatial entities collection) parameters from subset measures with a single parameter (collection of real values) in statistical measures, the number of different categories of measures was reduced from four (subset measures) to three (statistical measures). Finally to remove any dependency between statistical measures and case study specific spatial measures the unary BLSTL subset measures “clusteredness” and “density” have no corresponding statistical measures in BLMSTL.

The nmc symbol represents numeric measure collections. Given a model simulation σ nmc is evaluated according to one of the following definitions:

- **Spatial measure collection:** $nmc = smc$;
- **Temporal numeric measure collection:** $nmc = [a, b]$ nm is the collection of numeric measures obtained by evaluating the numeric measure nm against the subtrace $\sigma(i)$, for all $i \in [a, b]$ with $a, b \in \mathbb{R}_+$.

The nm symbol represents the category of real-valued numeric measures. Considering a given execution σ nm is evaluated according to one of the definitions described below:

- **Primary numeric measure:** $nm = pnm$;
- **Unary numeric measure:** $nm = unm(nm')$, where nm' is a numeric measure;
- **Binary numeric measure:** $nm = bnm(nm', nm'')$, where nm' and nm'' are numeric measures.

The pnm symbol corresponds to primary numeric (real-valued) measures. Considering a given execution σ pnm is evaluated according to one of the following definitions:

- **Numeric spatial measure:** $nm = nspm$;
- **Real number:** $nm = re \in \mathbb{R}$;
- **Numeric state variable:** $nm = nsu$.

The $nspm$ symbol represents the category of numeric (real-valued) spatial measures. Considering a given execution σ $nspm$ is evaluated according to one of the definitions described below:

- **Unary statistical spatial measure:** $nspm = ustm(smc)$, where smc is a spatial measure collection;
- **Binary statistical spatial measure:** $nspm = bstm(smc', smc'')$, where smc' and smc'' are spatial measure collections;
- **Binary statistical quantile spatial measure:** $nspm = bstqm(smc, re)$, where smc is a spatial measure collection, and re is a real value.

The smc symbol represents spatial measure collections and is evaluated considering a model execution σ as follows:

- **Primary spatial measure collection:** $smc = sm(ss) = \{value \mid value = sm(ss_i), \forall i, 1 \leq i \leq |ss|\}$, where sm is a spatial measure, and ss is the subset of considered spatial entities against which sm is evaluated;
- **Unary numeric spatial measure collection:** $smc = unm(smc')$, where unm is a unary numeric measure, and smc' is a spatial measure collection;
- **Binary numeric spatial measure collection:** $smc = bnm(smc', smc'')$, where bnm is a binary numeric measure, respectively smc' and smc'' are spatial measure collections.

Spatial measures sm are defined over the set {clusteredness, density, area, perimeter, distanceFromOrigin, angle, triangleMeasure, rectangleMeasure, circleMeasure, centroidX, centroidY} which is identical to the set of spatial measures computed for each detected region/cluster during the multiscale spatio-temporal analysis step of the model checking workflow.

Subsets of spatial entities are represented by the ss symbol. Considering a given execution σ ss is evaluated according to one of the definitions described below:

- **Specific subset:** $ss = specificSubset$, where $specificSubset$ represents the collection of all clusters/regions corresponding to $\sigma[0]$;
- **Filtered specific subset:** $ss = filter(specificSubset, constraint)$, where $specificSubset$ has the semantics defined above, and $constraint$ is a complex logic property that specifies which clusters/regions from $specificSubset$ should be considered (e.g. regions with area > 10);
- **Subset operation result:** $ss = subsetOperation(ss1, ss2)$, where $ss1$ and $ss2$ are spatial entities subsets, and $subsetOperation$ is the subset operation applied to $ss1$ and $ss2$.

Given an execution σ the value of the *specificSubset* symbol is computed using one of the definitions described below:

- **Regions:** $specificSubset = \bigcup_{spsv \in SpSV} \{reg \mid reg \in \text{regionDetectionMechanism}(spsv)\}$ considering the state $\sigma[0]$;
- **Clusters:** $specificSubset = \text{clustersDetectionMechanism}(se)$, where $se = \bigcup_{spsv \in SpSV} \{reg \mid reg \in \text{regionDetectionMechanism}(spsv)\}$ considering the state $\sigma[0]$.

Subsets of collections returned by *specificSubset* can be computed using the *filter* predicate. Considering an execution σ , *filter* is evaluated using the definition described below:

$$\text{filter} = \{e \in specificSubset \mid e \models c, \text{ where } c \text{ is a constraint}\}.$$

The semantics of the constraint satisfaction problem considering a spatial entity e and a constraint c is defined as follows:

- $e \models scaleAndSubsystem \asymp scsubsys$ if and only if there exists a vertex $v_{scsubsys} \in V_{MA}$ encoding the scale and subsystem $scsubsys$, e was annotated with a scale and subsystem $e_{scaleAndSubsystem}$ which has a corresponding vertex $v_{e_{scaleAndSubsystem}} \in V_{MA}$, and $v_{e_{scaleAndSubsystem}} \asymp v_{scsubsys}$, where V_{MA} is the set of vertices in the multiscale architecture graph MA , and $\asymp \in \{<, \leq, =, \geq, >\}$. To determine the truth value of expressions of the form $v_{e_{scaleAndSubsystem}} \asymp v_{scsubsys}$ the partial orders $<$ and \leq defined over the set of vertices V_{MA} are considered. Therefore $v_{e_{scaleAndSubsystem}} < v_{scsubsys}$ holds if the path from the root of MA to $v_{e_{scaleAndSubsystem}}$ passes through $v_{scsubsys}$. Conversely $v_{e_{scaleAndSubsystem}} > v_{scsubsys}$ holds if the path from the root of MA to $v_{scsubsys}$ passes through $v_{e_{scaleAndSubsystem}}$. Expressions $v_{e_{scaleAndSubsystem}} \leq v_{scsubsys}$ and $v_{e_{scaleAndSubsystem}} \geq v_{scsubsys}$ hold if $v_{e_{scaleAndSubsystem}} < v_{scsubsys}$ or $v_{e_{scaleAndSubsystem}} = v_{scsubsys}$, respectively if $v_{e_{scaleAndSubsystem}} > v_{scsubsys}$ or $v_{e_{scaleAndSubsystem}} = v_{scsubsys}$.
- $e \models sm \asymp fnm$ if and only if $sm(e) \asymp fnm$, where $sm(e)$ evaluates the spatial measure sm for the given spatial entity e , $sm \in \{\text{clusteredness}, \text{density}, \text{area}, \text{perimeter}, \text{distanceFromOrigin}, \text{angle}, \text{triangleMeasure}, \text{rectangleMeasure}, \text{circleMeasure}, \text{centroidX}, \text{centroidY}\}$, fnm is a filter numeric measure $\in \mathbb{R}$, and $\asymp \in \{<, \leq, =, \geq, >\}$.
- $e \models \sim c$ if and only if $e \not\models c$.

- $e \models c_1 \wedge c_2$ if and only if $e \models c_1$ and $e \models c_2$.
- $e \models c_1 \vee c_2$ if and only if $e \models c_1$ or $e \models c_2$.
- $e \models c_1 \Rightarrow c_2$ if and only if $e \models \sim c_1$ or $e \models c_2$.
- $e \models c_1 \Leftrightarrow c_2$ if and only if $e \models c_1 \Rightarrow c_2$ and $e \models c_2 \Rightarrow c_1$.
- $e \models (c)$ if and only if $e \models c$.

The f_{nm} symbol represents the (real-valued) numeric measure computed for the *filter* predicate. Given an execution σ and a spatial entity e the value of f_{nm} is computed using one of the definitions given below:

- **Primary numeric measure:** $f_{nm} = p_{nm}$;
- **Spatial measure:** $f_{nm} = sm(e)$, where $sm \in \{\text{clusteredness}, \text{density}, \text{area}, \text{perimeter}, \text{distanceFromOrigin}, \text{angle}, \text{triangleMeasure}, \text{rectangleMeasure}, \text{circleMeasure}, \text{centroidX}, \text{centroidY}\}$;
- **Unary filter numeric measure:** $f_{nm} = unm(f_{nm}')$, where unm is a unary numeric measure, and f_{nm}' is a filter numeric measure;
- **Binary filter numeric measure:** $f_{nm} = bnm(f_{nm}', f_{nm}'')$, where bnm is a binary numeric measure, respectively f_{nm}' and f_{nm}'' are filter numeric measures.

The semantics of the considered subset operation $subsetOperation \in \{\text{difference}, \text{intersection}, \text{union}\}$ is described below with respect to the spatial entities subsets $ss1$ and $ss2$:

- **Difference:** $\text{difference}(ss1, ss2) = \{se \mid se \in ss1 \text{ and } se \notin ss2\}$;
- **Intersection:** $\text{intersection}(ss1, ss2) = \{se \mid se \in ss1 \text{ and } se \in ss2\}$;
- **Union:** $\text{union}(ss1, ss2) = \{se \mid se \in ss1 \text{ or } se \in ss2\}$.

The $cm \in \{d, r\}$ symbol represents the collection of measures which compute the rate at which the value of a temporal numeric measure tnm changes from the current state to the next. Considering a given execution σ , such that $|\sigma| > 1$, and a temporal numeric measure tnm , the logic statement $cm(tnm)$ is evaluated according to one of the definitions below:

- **Derivative:** $cm(tnm) = d(tnm)$, such that

$$d(tnm) = \frac{tnm^1 - tnm^0}{time^1 - time^0};$$

- **Ratio:** $cm(tnm) = r(tnm)$, such that

$$r(tnm) = \frac{\frac{tnm^1}{tnm^0}}{time^1 - time^0},$$

where tnm^i represents the result of evaluating tnm against σ^i , and $time^i$ represents the value of the first time point in σ^i .

Finally the symbol nsv represents real-valued numeric state variables. Given an execution σ nsv is evaluated according to one of the following definitions:

- **Numeric state variable without an associated scale and subsystem:** $nsv = sv$, where sv is the identifier of a numeric state variable having no associated scale and subsystem (i.e. $SVSS(sv) = \emptyset$), and which evaluates to $NV(\sigma, \sigma[0], sv)$;
- **Numeric state variable with an associated scale and subsystem:** $nsv = sv(scaleAndSubsystem = scsubsys)$, where sv is the identifier of a numeric state variable having the associated scale and subsystem $SVSS(sv) = scsubsys$ and value $NV(\sigma, \sigma[0], sv)$.

5.5.1.3 Illustrative examples of BLMSTL statements

Illustrative examples of statements written both in natural language and BLMSTL are provided below. For simplicity purposes the number of explicitly specified scales and subsystems is two in all examples.

- **Natural language:** Always during the time interval [0, 95] if the concentration of EGFR (corresponding to scale and subsystem (Intracellular, RasERKPathway)) increases over 20, then the cancerous cell (corresponding to scale and subsystem (Cellular, Cancerous)) will divide i.e. the cell count will increase.

BLMSTL: $G[0, 95] ((\{EGFR\}(scaleAndSubsystem = Intracellular.RasERKPathway) > 20) \Rightarrow (d(count(density(filter(regions, scaleAndSubsystem = Cellular.Cancerous)))) > 0)).$

- **Natural language:** If the concentration of drug X (corresponding to scale and subsystem (Organism, Human)) eventually increases during time interval [5, 10], then the area of the aorta cross section (corresponding to scale and subsystem (OrganSystem, Aorta)) will be larger during time interval [10, 30] than [0, 10].

BLMSTL: $(F[5, 10] d(\{X\}(scaleAndSubsystem =$

*Organism.Human)) > 0) ⇒
 (min([10, 30] min(area(filter(regions, scaleAndSubsystem =
 OrganSystem.Aorta)))) >
 max([0, 10] max(area(filter(regions, scaleAndSubsystem =
 OrganSystem.Aorta))))).*

- **Natural language:** Always during the time interval [0, 100] the liver dysfunction (corresponding to scale and subsystem (Organ, Liver)) is equal to the average degree of damage suffered by its constituent tissues (corresponding to scales and subsystems \leq (Tissue, DamagedLiverTissue)).
BLMSTL: $G[0, 100] (\{\text{LiverDysfunction}\} (\text{scaleAndSubsystem} = \text{Organ.Liver}) = \text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} \leq \text{Tissue.DamagedLiverTissue}))))$.

To enable explicitly encoding the probability with which a BLMSTL statement is expected to hold, a probabilistic extension of BLMSTL called Probabilistic Bounded Linear Multiscale Spatial Temporal Logic is defined.

5.5.2 Probabilistic Bounded Linear Multiscale Spatial Temporal Logic

Definition 16 Probabilistic Bounded Linear Multiscale Spatial Temporal Logic (PBLMSTL)

A Probabilistic Bounded Linear Multiscale Spatial Temporal Logic property ϕ is a logic property of the form $P_{\bowtie\theta}[\psi]$ where $\bowtie \in \{<, \leq, \geq, >\}$, $\theta \in (0, 1)$ and ψ is a BLMSTL property.

An illustrative example of a natural language probabilistic statement mapped into PBLMSTL is given below:

Natural language: The probability is greater than 0.9 that always during the time interval [0, 95] if the concentration of EGFR (corresponding to scale and subsystem (Intracellular, RasERKPathway)) increases over 20, then the cancerous cell (corresponding to scale and subsystem (Cellular, Cancerous)) will divide i.e. the cell count will increase.

PBLMSTL: $P > 0.9 [G[0, 95] ((\{\text{EGFR}\}(\text{scaleAndSubsystem} = \text{Intracellular.RasERKPathway}) > 20) \Rightarrow (d(\text{count}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Cellular.Cancerous})))) > 0))]$.

Remark 8. The probability employed in the immediately above example was chosen for illustrative purposes and was not derived from experimental data or the literature.

A PBLMSTL property $\phi \equiv P_{\bowtie\theta}[\psi]$ holds for an MSSpDES \mathcal{M} (i.e. $\mathcal{M} \models P_{\bowtie\theta}[\psi]$) if and only if the probability of ψ to hold for an execution of \mathcal{M} is $\bowtie\theta$. Therefore in order to determine the truth value of a PBLMSTL property ϕ the likelihood of ψ being true needs to be computed.

5.6 Model checking

Definition 17 Multiscale multidimensional spatio-temporal model checking problem

The multiscale multidimensional spatio-temporal model checking problem is to automatically verify if an MSSpDES \mathcal{M} satisfies a PBLMSTL property $\phi \equiv P_{\bowtie\theta}[\psi]$.

Due to the high complexity associated with probabilistic computational models of biological systems only approximate probabilistic model checking approaches are considered throughout. As illustrated in Table 2.1 the approaches considered are either Bayesian or frequentist, estimate or hypothesis testing based.

Using approximate probabilistic model checking approaches MSSpDES models can be validated relative to PBLMSTL formal specifications using a finite number of steps. Therefore the multiscale multidimensional spatio-temporal model checking problem is well-defined. The corresponding proof is similar to the one provided for the multidimensional model checking problem (see Subsubsection 3.6.1.3) and therefore will be given in Appendix D.1 rather than the main matter of the thesis. Intuitively the main idea behind the proof is to show that in order to validate an MSSpDES model the number of required model simulations is finite, and that the number of time points considered for each model simulation is bounded. Therefore the PBLMSTL specification is evaluated against a finite number of time points and model simulations, which can be done in a finite number of steps.

5.7 Meta model checking

One of the main limitations of the multiscale model checking methodology, as described up to this point, is that the evolution over time of spatial properties can be described only with respect to the predefined collections of spatial entity types $SET_{considered} = \{\text{clusters, regions}\}$ and spatial measures $SM_{considered} =$

$\{\text{clusteredness}, \text{density}, \text{area}, \text{perimeter}, \text{distanceFromOrigin}, \text{angle}, \text{triangleMeasure}, \text{rectangleMeasure}, \text{circleMeasure}, \text{centroidX}, \text{centroidY}\}$. Therefore the methodology cannot be applied without further modifications to case studies where other types of spatial entities (e.g. 3D spatial structures) and/or measures (e.g. volume) are relevant.

The main reason for this limitation is that several components of the multiscale model checking methodology depend directly on the collections of spatial entity types and measures considered. These components are the multiscale spatio-temporal analysis approach, MSTML and (P)BLMSTL. In case of the multiscale spatio-temporal analysis for each spatial entity type $sety \in SET_{\text{considered}}$ and spatial measure $sm \in SM_{\text{considered}}$ a corresponding spatial detection mechanism and evaluation function are defined. Conversely in MSTML each possible value for the *spatialType* attribute (see *spatialEntity* element) corresponds to a spatial entity type $sety \in SET_{\text{considered}}$. Moreover the information describing the state of a *spatialEntity* element is described by child elements corresponding to the spatial measures $sm \in SM_{\text{considered}}$. Finally (P)BLMSTL enables writing formal specifications that describe how the values of spatial measures $sm \in SM_{\text{considered}}$ evaluated against spatial entities of type $sety \in SET_{\text{considered}}$ are expected to change over time.

In order to overcome this limitation we define a generalized version of the multiscale multidimensional spatio-temporal model checking methodology called multiscale multidimensional spatio-temporal *meta* model checking; the term *meta* is used in multiscale multidimensional spatio-temporal meta model checking similarly to how it is used in meta-programming. From a theoretical point of view multiscale multidimensional spatio-temporal meta model checking enables employing arbitrary spatial entity types and measures for the validation of computational models. Conversely from an implementation point of view a multiscale multidimensional spatio-temporal meta model checking program, similarly to a meta-program, takes program templates and case study specific spatial entity types and measures as input and produces a corresponding case study specific multiscale multidimensional spatio-temporal model checking program as output.

The main difference between the multiscale multidimensional spatio-temporal and multiscale multidimensional spatio-temporal *meta* model checking methodologies is that in the latter $SET_{\text{considered}}$ and $SM_{\text{considered}}$ are replaced with meta collections of spatial entity types SET , respectively spatial measures SM , defined as follows:

- $SET = \{sety \mid sety \text{ is a spatial entity type for which there exists a corresponding spatial detection mechanism } f_{sety}, f_{sety} : SpSV^p \rightarrow \{0, 1\}^{m_1 \times n_1} \times \{0, 1\}^{m_2 \times n_2} \times \dots \times \{0, 1\}^{m_p \times n_p}, \text{ which detects sets of spatial entities } SE \text{ of type } sety \text{ in the discretised spatial domain}\}.$

Considering the spatial state variable tuples $spsvt \in SpSV^p$, f_{sety} computes which positions of the discretised space are occupied (1) by spatial entities or not (0); see Subsection 3.4.1 for examples of spatial detection mechanisms corresponding to the spatial entity types *clusters* and *regions*.

- $SM = \{sm \mid sm \text{ is a spatial measure, } sm : SE \rightarrow SMV \subseteq \mathbb{R}, \text{ where } SE \text{ is a set of spatial entities and } SMV \text{ is the corresponding domain of valid spatial measure values}\};$ see Subsection 3.4.2 for examples of spatial measures corresponding to the spatial entity types *clusters* and *regions*.

These collections are called meta because they provide only a description of the conditions which should hold for each type of spatial entity and measure but do not explicitly define instances thereof. Since SET and SM do not contain specific spatial entity types and spatial measures, the multiscale spatio-temporal analysis, MSTML and (P)BLMSTL can be only partially defined.

The multiscale meta model checking methodology enables the creation of different multiscale model checking methodology instances by replacing SET and SM with case study specific collections of spatial entity types and spatial measures, and updating the multiscale spatio-temporal analysis, MSTML and (P)BLMSTL accordingly. These instances can then be used to validate MSSpDES models considering case study specific types of spatial entities and/or measures. For instance, in order to validate computational models considering a 3D representation of space a corresponding model checking methodology instance could be created that replaces SET and SM with $SET_{3D} = \{cuboid, cylinder, sphere\}$ and $SM_{3D} = \{volume, centroidX, centroidY, centroidZ\}$.

A graphical description of the workflow employed to create instances of multiscale multidimensional spatio-temporal model checking methodologies is provided in Figure 5.7.

The workflow depicted in Figure 5.7 enables automatically creating multiscale model checking methodology instances tailored to specific spatial representations (e.g. 3D), and types of spatial entities and measures. However the workflow does not automatically define the image processing functions required to automatically detect and analyse spatial entities in time series data. Such functions can often be defined based on existing approaches from the image processing literature.

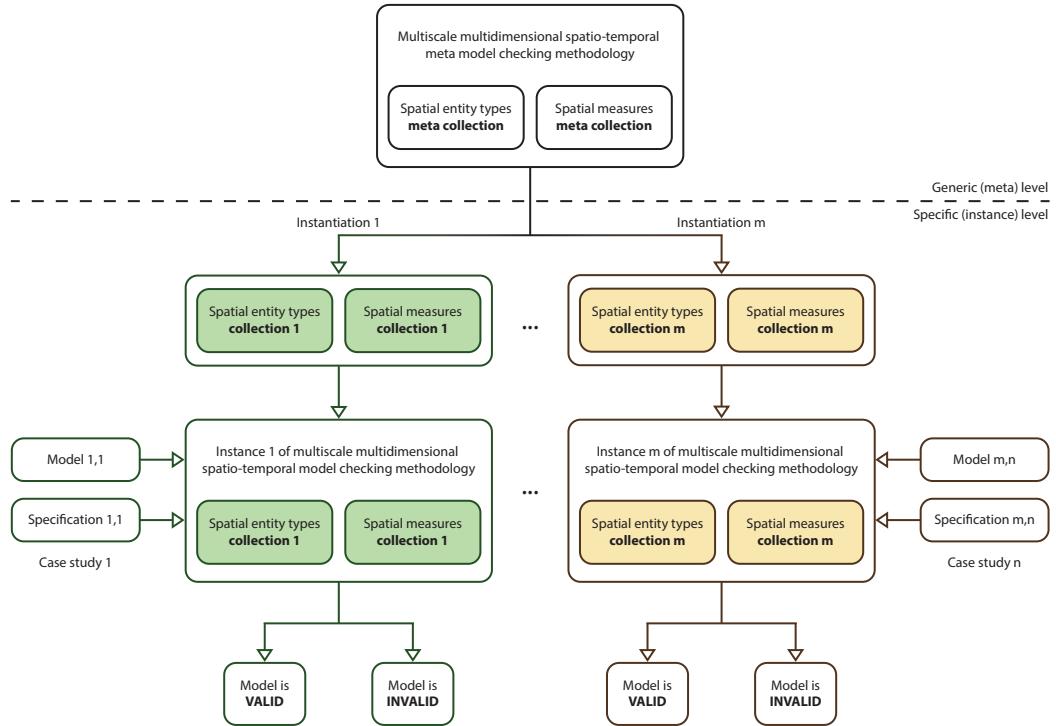


Figure 5.7: Workflow for creating multiscale multidimensional spatio-temporal model checking methodology instances. The workflow comprises two levels, the upper generic (meta) level, and the lower specific (instance) level. The upper level comprises the multiscale multidimensional spatio-temporal meta model checking methodology. Conversely the lower level consists of the specific collections of spatial entity types and measures employed to create multiscale multidimensional spatio-temporal model checking methodology instances. For each pair (e.g. m) of spatial entity types and spatial measures collections considered a corresponding multiscale model checking methodology instance is created. The resulting methodology instances (e.g. m) can then be employed for various case studies (e.g. n) to decide if computational models (e.g. m,n) are valid relative to corresponding formal specifications (e.g. m,n) or not. Rounded rectangles and arrows having the same border/line colour correspond to the same collections of spatial entity types and spatial measures.

Finally following on from Appendix D.1 when validating an MSSpDES model relative to a formal PBLMSTL specification the number of required model simulations, and the number of required state transitions for each model simulation do not depend directly on the considered collections of spatial entity types and spatial measures. Therefore regardless of the instances of *SET* and *SM* considered the multiscale spatio-temporal model checking problem is well-defined.

5.8 Implementation

To automate the validation of computational models relative to how both numeric and spatial properties are expected to change over time and across multiple scales the multiscale spatio-temporal detection and analysis approach and the multiscale multidimensional spatio-temporal meta model checking method were implemented in software tools. The name of the model checking software is Mule and is a concatenation of the first and last two letters of the word “Multiscale”.

Using the model checker Mule and the multiscale spatio-temporal detection and analysis module computational models can be validated relative to PBLMSTL specifications by adapting the workflow given in Figure 3.7 to the multiscale context i.e. the (uniscale) spatio-temporal analysis is replaced by the multiscale spatio-temporal analysis (see Subsection 5.4.1), STML is replaced with MSTML, PBLSTL is replaced with PBLMSTL and Mudi is replaced with Mule.

5.8.1 Multiscale spatio-temporal detection and analysis module

The multiscale spatio-temporal detection and analysis module was implemented as a Bash script which executes the (uniscale) spatio-temporal detection and analysis modules (see Subsection 3.7.1) for each scale and subsystem considered.

Given that scales and subsystems are case study specific the multiscale spatio-temporal detection and analysis Bash script must be adapted for each case study. For usability purposes the script was designed such that only particular steps emphasized by “TODO” comments need to changed between case studies.

5.8.2 Model checker Mule

Mule builds on the implementation of the model checker Mudi (see Subsection 3.7.2). Consequently the use cases for Mule are the same as for Mudi (see Figure 3.8) but adapted to the multiscale context i.e. computational models are validated relative to PBLMSTL instead of PBLSTL specifications, STML files are replaced with MSTML files, and the user can additionally provide as input to Mule an *MA* graph. Similarly the architecture of Mule is the same as the architecture of Mudi (see Figure 3.9) with the exception that a PBLMSTL instead of a PBLSTL logic property parser and evaluator is employed. In addition, similarly to Mudi, Mule was implemented in C++ and supports all approximate probabilistic model checking approaches described in Table 2.1. Moreover all unit tests were implemented using the Google Test unit testing framework.

In contrast to Mudi, Mule can be adapted to case study specific spatial entity types and/or measures. The workflow for generating instances of the multiscale multidimensional spatio-temporal model checker Mule was implemented as described in Figure 5.8. The main idea behind the implementation is to use two instead of one compilation (or translation) steps. The first compilation step takes a description of the spatial entity types and measures as input and produces C++ source code as output. The second compilation step translates the generated C++ source code in binary (i.e. executable) format. Conceptually this approach

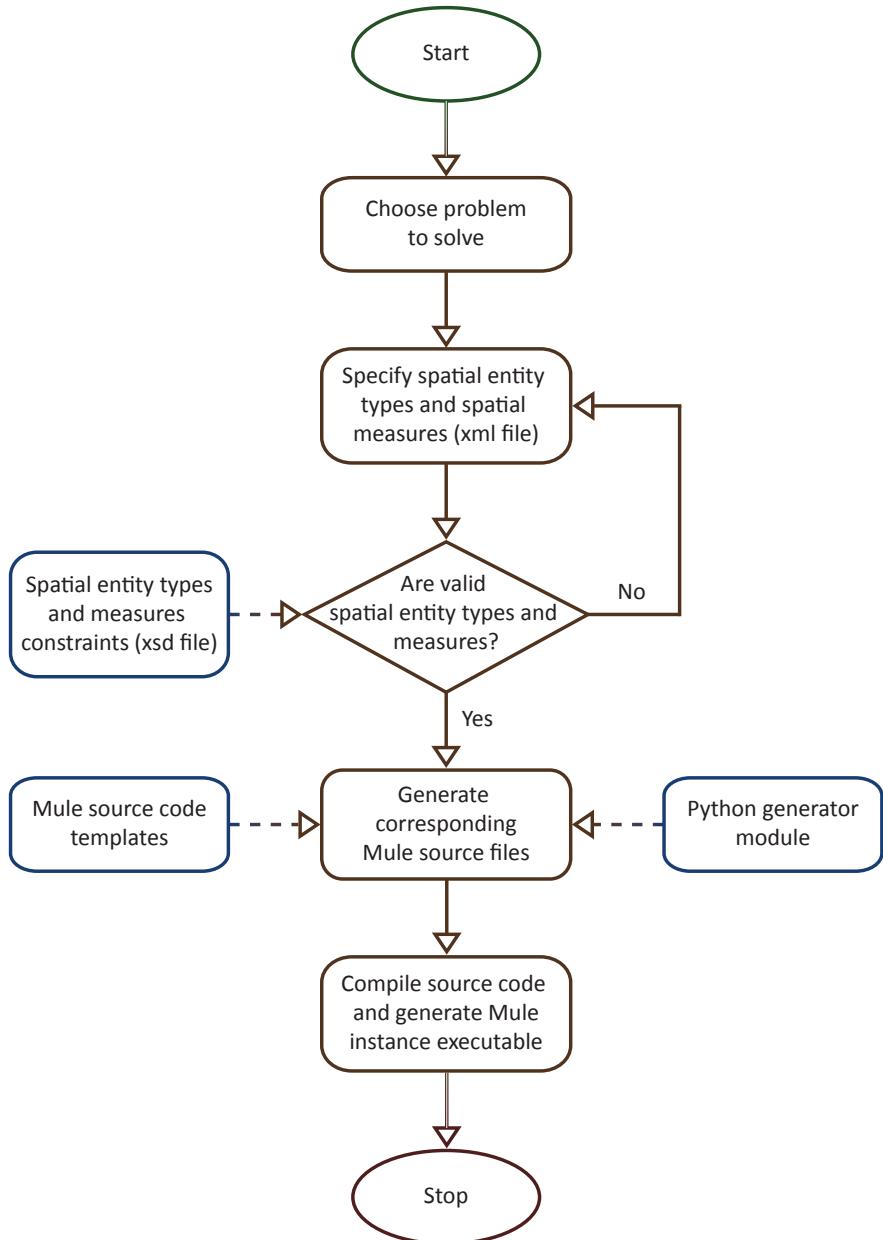


Figure 5.8: Implementation of workflow for generating multiscale multidimensional model checker instances according to user-defined spatial entity types and spatial measures. Starting from the problem one tries to solve, an xml file is created describing the collections of spatial entity types and spatial measures of interest. These collections are then validated with respect to relevant constraints captured by an xsd file; see <http://mule.modelchecking.org/standards> for the latest version of the xsd file. If the xml file validation fails then the specification of the spatial entity types and measures needs to be updated accordingly. Otherwise the xml file is employed by a C++ source code generator written in Python to generate the corresponding Mule source files based on a set of predefined templates. The source files are compiled to produce an executable version of the corresponding Mule instance. This instance can then be employed to validate relevant computational models.

is called “meta” because Mule is an abstract (meta) model checker that can be instantiated according to case study specific spatial entity types and measures. From a practical point of view the user modifies only the description of the spatial entity types and measures, while the source code and the corresponding executables are automatically generated for him/her.

One of the main advantages of the workflow depicted in Figure 5.8 is that it enables generating different instances of Mule corresponding to various spatial entity types and spatial measures without negatively impacting the efficiency of the model checker. Conversely its main disadvantage is that it requires recompiling the model checker whenever the collections of spatial entity types and spatial measures change; this could be especially challenging for non-technical users. In order to address this issue a web service could be set up in the future where users upload the xml specification file and receive the corresponding model checking executable as output.

Similarly to Mudi, Mule can be executed only from the command line but, to improve its usability, a corresponding graphical user interface could be designed and implemented in the future.

The instance of Mule considered throughout this thesis is defined with respect to the collection of spatial entity types $SET_{considered}$ and spatial measures $SM_{considered}$. However, for simplicity purposes the *angle* spatial measure is no longer computed as described in Subsection 3.4.2; the *angle* of a spatial entity is now determined by the lines that pass through the discretised spatial domain's centre point and are tangent to the spatial entity's outer contour.

5.8.3 Availability

The multiscale spatio-temporal detection and analysis Bash scripts corresponding to the case studies against which the efficacy of the model checker Mule is illustrated (see Chapter 6) are made freely available online at https://github.com/IceRage/Mule/tree/master/Multiscale/script/analysis/case_study_specific. Conversely the source code and the executable corresponding to the Mule instance employed throughout the thesis are made freely available online via the official Mule website <http://mule.modelchecking.org>. Moreover a Docker image has been created to provide a self-contained environment for executing/updating model checker instances on all major operating systems without additional setup (except installing Docker (Docker, 2015)). The official Mule website additionally contains the xsd schemas for *MA* graphs, MSTML files and meta model checking configuration files, the computational models, datasets of MSTML files, PBLMSTL specifications and *MA* graphs corresponding to the case studies considered (see Chapter 6), text and video based tutorials on how to download, install and use Mule, respectively on how to generate a case study specific instance of Mule, and a link to the Mule issue tracking webpage.

5.9 Related work

Although the problem of validating multiscale spatio-temporal computational models of biological systems using model checking approaches has not been addressed previously in the literature, the need for reasoning about the evolution of systems across multiple scales has been considered in other fields of science. Relevant approaches from the areas of pattern recognition and spatial information theory are described below. To the best of our knowledge there is no related work on meta model checking approaches.

5.9.1 Pattern recognition

A model checking approach which explicitly distinguishes between multiple spatial scales without (initially) accounting for time was introduced by Grosu (Grosu et al., 2009) for pattern detection. The multiscale representation of space was created by recursively splitting a spatial domain in quadrants (a finite number of times) and representing the resulting hierarchy as a quadtree. For reasoning about spatial subdomains along a linear path through the quadtree a formal logic called Linear Spatial Superposition Logic and a corresponding model checking algorithm were introduced.

More recently both the formal logic and corresponding model checking algorithm were extended by Gol et al. (Gol et al., 2014) to account for branching paths through quadtrees (Tree Spatial Superposition Logic), and by Haghghi et al. (Haghghi et al., 2015) to account for the evolution of the quadtrees over time (SpaTel).

Similarly to BLMSTL SpaTel enables writing formal specifications about how both numeric and spatial properties are expected to change over time and across multiple scales. In SpaTel numeric properties are encoded as binary expressions comparing (using \leq or \geq) the value of a numeric state variable to a real value. Conversely spatial properties are encoded by stating in which of the four possible quadrants (i.e. *NW*, *NE*, *SW*, *SE*) numeric properties should hold. In addition SpaTel enables specifying at which scale(s) numeric and/or spatial properties should hold using the U_k (i.e. bounded until) and \bigcirc (i.e. next) operators. However in contrast to BLMSTL SpaTel does not enable reasoning about general multiscale systems since only one spatial domain is considered and the (quadrants) relation between consecutive levels/scales is fixed. Moreover it is not possible to describe how spatial entities and their properties potentially spanning multiple quadrants of the spatial domain change over time.

5.9.2 Spatial information theory

In the area of spatial information theory several existing directional and topological qualitative uniscale spatial logics (see Subsection 3.8.2) have been adapted to the multiscale context to enable reasoning about how regions evolve in space, time and across multiple scales (Du et al., 2013b, 2014; Su et al., 2011).

The hierarchical organization of the spatial domain considered is usually encoded using tree-like data structures (Plumejeaud et al., 2011) where the root of the tree corresponds to the most coarse-grained representation, and the leaves to the most fine-grained representation. Each region is encoded by a different vertex, and (directional/topological) relations between regions are represented using edges.

The main difference between uniscale and multiscale qualitative spatial logics is that in case of the latter the employed spatial representation is hierarchical instead of flat, where each level in the hierarchy corresponds to a different spatial resolution/scale. Moreover spatial relations can be defined with respect to regions from different scales.

Despite enabling to reason about the evolution of systems over space, time and across multiple scales, such multiscale spatial logics are limited to qualitative descriptions and therefore cannot numerically describe how (biological) systems are expected to change over time.

Summary

This chapter has described a novel methodology and model checker implementation for validating multiscale computational models of biological systems with respect to both how numeric and case study specific spatial properties are expected to change over time and across multiple scales. Biological systems are abstractly represented as MSSpDESs which explicitly encode the hierarchical organization of real-life systems using *MA* graphs, and map state variables to particular scales and subsystems using the *SVSS* assignment function. The simulation output of MSSpDES models is processed by the multiscale spatio-temporal analysis module for automatically detecting and analysing how emergent spatial patterns, potentially from multiple scales, change over time. Formal specifications against which MSSpDES models are validated are encoded using the formal language PBLMSTL introduced here. Given an MSSpDES model and a formal PBLMSTL specification corresponding model checking algorithms decide if the MSSpDES model is valid relative to the given specification. To enable adapting the multiscale

model validation approach to case study specific spatial entity types and measures, a generalization of the multiscale model checking methodology is defined; the resulting approach is called multiscale multidimensional spatio-temporal meta model checking. Implementation details and a comparison with related approaches from the areas of pattern recognition and spatial information theory were provided in the end.

CHAPTER 6

Validation of multiscale computational models of biological systems

Introduction

This chapter illustrates how to employ the multiscale multidimensional spatio-temporal model checking methodology described in Chapter 5 to validate four multiscale computational models of biological systems previously published in the literature. The computational models are of different complexity, were encoded using different modelling formalisms and software, are deterministic, stochastic or hybrid, and represent space explicitly or not. The corresponding case studies are the rat cardiovascular system dynamics, the uterine contractions of labour, the *Xenopus laevis* cell cycle and the acute inflammation of the gut and lung. The computational models have been validated using the model checker Mule against formal PBLMSTL specifications which were derived from the original papers introducing the computational models. Conclusions drawn from validating the computational models are provided in the end.

6.1 Description

The efficiency and applicability of the multiscale multidimensional spatio-temporal meta model checking methodology was assessed considering four systems biology case studies published in the literature. The case studies were chosen such that the corresponding computational models are of different types (i.e. determinis-

tic/hybrid/stochastic), span different levels of organization (e.g. cellular/organ) and are encoded using different modelling formalisms (e.g. ordinary differential equations/cellular automata) and software (e.g. Morpheus/NetLogo); see Table 6.1 for a brief comparison of the computational models considered.

Table 6.1: Considered multiscale systems biology computational models against which the multiscale multidimensional spatio-temporal meta model checking methodology and implementation were validated. Each model (M1–M4) has an associated description and type (i.e. deterministic, stochastic or hybrid), was encoded using specific modelling formalisms and software, represents space explicitly or not (Y = Yes, N = No), spans different levels of organization, and has a corresponding reference paper and download link.

	M1	M2	M3	M4
Description	Rat cardiovascular system dynamics	Uterine contractions of labour	<i>Xenopus laevis</i> cell cycle	Acute inflammation of gut and lung
Model type	Deterministic	Deterministic	Hybrid	Stochastic
Modelling formalism(s)	Ordinary differential equations (ODE)	Cellular automata (CA)	ODEs + Cellular Potts model (CPM)	Agent based modelling (ABM)
Modelling software	JSim	Mathematica	Morpheus	NetLogo
Explicit spatial representation	N	Y	Y	Y
Levels of organization	Cellular + Organ system	Cellular + Tissue	Intracellular + Cellular	Cellular + Tissue + Organ
Case study reference	(Beard et al., 2012)	(Young and Barendse, 2014)	(Ferrell Jr. et al., 2011)	(An, 2008)
Model download link	http://wiki.virtualrat.org/VPRWiki/images/6/67/Workflow_Model_Files.rar	http://s3-eu-west-1.amazonaws.com/files.figshare.com/1720626/Supporting_Information_S1	http://imc.zih.tu-dresden.de/wiki/morpheus/doku.php?id=examples:multiplescale#odes_in_cpm_cellcycle_and_proliferation	http://bionetgen.org/SCAI-wiki/images/7/7d/GutLungAxis2.1.nlogo

For generalizability purposes two of the models considered in Table 6.1 are deterministic (i.e. M1 and M2), one is hybrid (i.e. M3) and one is stochastic (i.e. M4). As per Definition 13 the proposed model checking approach assumes that the computational models considered are represented as MSSpDESs. Translating stochastic computational models to an MSSpDES representation is straightforward. However in order to translate deterministic/hybrid computational models to an MSSpDES representation two potential changes are required.

First of all, in order to preserve the decidability of the model checking algorithms the behaviour of any continuous-time model component needs to be described by a finite sequence of states, which can be encoded in the MSSpDES

representation. Our assumption is that considering a discrete approximation of the model behaviour is potentially appropriate for biological systems since the *in vitro* observations used to build the model were also recorded considering a finite number of discrete time points (He et al., 2008; Liu et al., 2009a; Palaniappan et al., 2013). For minimizing losses in accuracy the time step employed during the discretisation process was set equal to the time step Δh chosen originally by the model authors for the *in silico* numerical model simulators; in case of adaptive time stepping the minimum time step could be considered.

Secondly, a probability value needs to be associated to each state transition encoded in the models. In case of deterministic state transitions a probability value of 1 is associated, and in case of stochastic state transitions the probabilities initially encoded in the (hybrid) models are preserved.

One of the main differences between stochastic/hybrid and deterministic models translated to an MSSpDES representation is that in case of the former, the probability of the system under consideration to transition from the current to the next state is a real value $p \in [0, 1]$, and in case of the latter $p \in \{0, 1\}$. Similarly, the probability of a BLMSTL logic statement to hold for a stochastic/hybrid model is a real value $p \in [0, 1]$, while in case of a deterministic model it is a real value $p \in \{0, 1\}$.

The other steps of the multiscale model checking workflow (i.e. multiscale spatio-temporal analysis, formal specification, model checking) do not need to be modified when employing deterministic/hybrid models because they are defined relative to simulation traces rather than the models themselves.

The natural language and corresponding formal specifications, against which the models were validated, have been derived from the original papers introducing the case studies. Quotes from the original papers have been employed to create *initial* natural language statements describing the expected system behaviour. The initial natural language statements were then rephrased to match the constructs and structure typical to formal PBLMSTL statements; the resulting statements are called *rephrased* natural language statements. Finally the rephrased natural language statements were manually mapped into corresponding PBLMSTL statements. Where insufficient information was available (e.g. probabilities) the numeric values employed in the formal specification are quantitative approximations of the corresponding natural language descriptions (e.g. with high probability $\Rightarrow 0.9$). The main purpose of the PBLMSTL statements considered is to illustrate the expressivity of the methodology and not to predict previously unknown biologically relevant properties.

The computational models have been simulated, analysed and validated using

the same regular desktop computer (Linux x64, Intel Core i5-2500 CPU @1.6 GHz, 16 GB DDR3 RAM memory). To assess the performance of the approach execution times have been recorded for all relevant steps of the model checking workflow. Moreover, for comparison purposes, the case studies and the corresponding computational models will not be described individually but in parallel considering the steps of the model checking workflow (i.e. model construction, multiscale spatio-temporal analysis, formal specification, model checking).

For reproducibility purposes the *MA* graph, the generated MSTML file(s), and the formal PBLMSTL specification corresponding to all computational models are made available at <http://mule.modelchecking.org/case-studies>.

6.2 Model construction

6.2.1 Rat cardiovascular system dynamics

The cardiovascular system comprises the heart, blood and blood vessels, and is the organ system responsible for delivering oxygen and nutrients to, and removing waste products from the entire organism. Its dynamics changes in case of a transient increase of the thoracic pressure (e.g. by performing the Valsalva manoeuvre) which leads to reduced blood flow in the right atrium, reduced cardiac output and decreased aortic pressure (Beard et al., 2012).

In order to describe the behavioural changes of the cardiovascular system during the Valsalva manoeuvre Beard et al. built a multiscale non-spatial ODE model (Beard et al., 2012) by integrating two previously existing models. The first model is an abstract representation of the cardiovascular system (Smith et al., 2004). Conversely the second model encodes the baroreflex mechanism (Bügenhagen et al., 2010) which is employed to maintain the blood pressure of an organism at approximately constant levels. One of the main advantages of the integrated multiscale model is that it enables relating changes at the entire cardiovascular system level with changes at the baroreflex mechanism level and vice versa, which was not possible when employing the constituent models separately. The hierarchical organization of the resulting model is encoded by the *MA* graph depicted in Figure 6.1.

For validation purposes the numeric state variables considered at the organ system scale are the thoracic pressure and the heart rate, and the aortic pressure at the cellular scale.

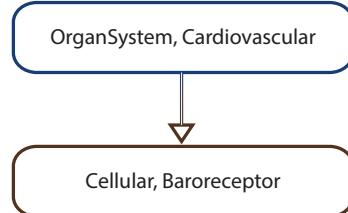


Figure 6.1: *MA* graph representing the multiscale organization of the rat cardiovascular system dynamics computational model.

6.2.2 Uterine contractions of labour

Although it is known that usually during human labour regions across the entire uterus contract in a coordinated fashion the underlying mechanisms by which an initial local contraction propagates to the entire organ level are not fully understood (Young and Barendse, 2014).

One hypothesis is that a positive feedback loop is created between the tissue level contractions and the intrauterine pressure as follows: an initial tissue level contraction increases the intrauterine pressure and adds tension to the neighbouring regions, which in response start to contract, thus increasing the intrauterine pressure even further and adding tension to their corresponding neighbouring regions which also start to contract, and the entire process is repeated until all contractile regions across the entire organ are recruited.

In order to test this hypothesis Young and Barendse developed a corresponding predictive deterministic computational model (Young and Barendse, 2014). The model was encoded as a cellular automaton in Mathematica and spans two levels of organization, the organ level for the uterus, and the tissue level for the uterine regions; see Figure 6.2 for the corresponding *MA* graph.

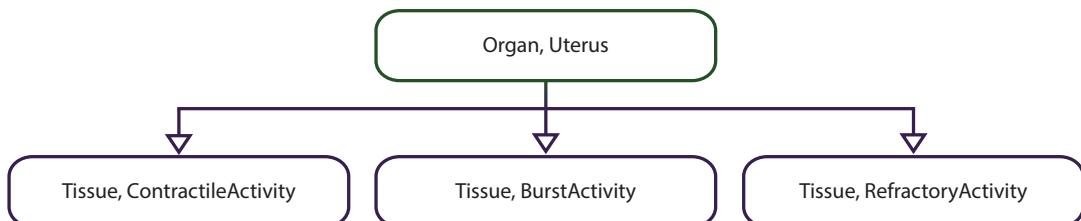


Figure 6.2: *MA* graph representing the multiscale organization of the uterine contractions of labour computational model.

At the organ (i.e. uterus) scale the numeric state variable considered is the intrauterine pressure and space is encoded explicitly as a 4×4 grid, where each grid position represents a tissue (i.e. uterine region). Conversely at the tissue level there is no explicit representation of space and the recorded numeric state variables are the contractile, burst and refractory activities of the uterine regions.

6.2.3 *Xenopus laevis* cell cycle

The cell cycle is a fundamental biological process which is responsible for the replication/division of cells and is involved in the development and partial renewal of organisms. Its complexity usually increases with the complexity of the organism considered. Therefore it is studied in lower and less complex organisms such as the *Xenopus laevis* frog.

To gain a better understanding of the *Xenopus laevis* embryonic cell cycle and how it affects cellular population growth the developers of the modelling software Morpheus (Starruß et al., 2014) built a corresponding multiscale computational model (Starruß and Back, 2014). The computational model describes how three proteins CDK1, Plk1 and APC regulate the cell cycle at the intracellular level using ODEs (Ferrell Jr. et al., 2011), respectively how cells divide and are displaced in 2D space at the cellular level using a CPM. The corresponding *MA* graph is depicted in Figure 6.3.

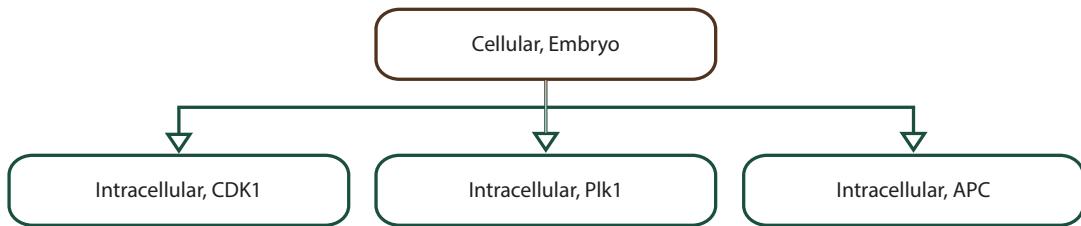


Figure 6.3: *MA* graph representing the multiscale organization of the *Xenopus laevis* cell cycle computational model.

At the cellular level space is represented explicitly as a 52×52 grid recording the spatial distribution of the population of cells. Conversely at the intracellular level there is no explicit representation of space and the numeric state variables considered are the concentrations of CDK1, Plk1 and APC.

6.2.4 Acute inflammation of the gut and lung

There is no single definition of inflammation in the literature (Scott et al., 2004) but in this thesis we will interpret it as the response of a biological system to bodily damaging stimuli. Depending on the intensity of the stimulus an inflammatory response initiated in one organ can propagate to other organs and eventually lead to multiple organ failure (An, 2008).

To gain a better understanding of the relation between inflammatory responses and multiple organ failure, G. An (An, 2008) built a multiscale agent-based computational model using the software NetLogo which describes how the inflammation of either the gut (i.e. gut ischemia) or lung (i.e. pneumonia) could potentially lead to the failure of both organs. The levels of organization considered in the

computational model are cellular (for representing endothelial and epithelial cells), tissue (for representing the organ luminal space, the blood vessel luminal space, and the endothelial and epithelial layers), and organ (for representing the gut and lung); see Figure 6.4 for the corresponding *MA* graph.

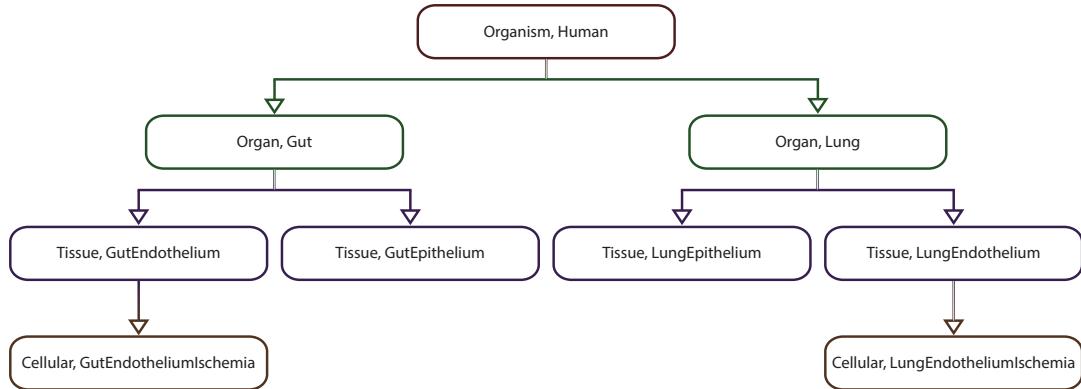


Figure 6.4: *MA* graph representing the multiscale organization of the acute inflammation of the gut and lung computational model.

The organism level is not modelled explicitly and the corresponding vertex (Organism, Human) was added to the *MA* graph in Figure 6.4 only to ensure that its structure is tree-like. At the organ level space is not represented explicitly and the numeric state variables considered represent the amount of solute which leaked into the gut and lung. Conversely at the tissue level space is represented explicitly as a 31×31 grid where each grid position represents a cell. The tissue level numeric state variables considered for both gut and lung are the total concentration of cytoplasm and cell wall occludin, and the total cell damage by-product. At the cellular level the numeric state variables considered encode the level of ischemia for both gut and lung endothelial cells.

6.3 Multiscale spatio-temporal analysis

The computational models M1–M4 were simulated and the simulation outputs were translated to MSTML. The translation operation comprises converting the simulation output to csv format, generating an MSTML subfile for each considered time point, numeric state variable and spatial region comprising one or multiple grid positions, and in the end merging all subfiles into a single MSTML file. Execution times for the model simulation and subsequent translation steps corresponding to all computational models are provided in Table 6.2.

The most time consuming step for the rat cardiovascular system dynamics (i.e. ≈ 37.22 s) and the acute inflammation of the gut and lung (i.e. ≈ 329.60 s) case studies was the model simulation due to the large number of time points considered

Table 6.2: Model simulation and analysis execution times for the rat cardiovascular system dynamics, the uterine contractions of labour, the *Xenopus laevis* cell cycle and the acute inflammation of the gut and lung case studies. The steps considered are model simulation, conversion of the simulation output to csv format, generating an MSTML subfile for each considered time point, numeric state variable and spatial region comprising one or multiple grid positions, and merging subfiles into a single MSTML file. Depending on the computational model type (i.e. deterministic/stochastic/hybrid) and the formal specification against which it was validated, the number of considered model simulations, respectively time points per model simulation differed. Computational models are distinguished by their model id (i.e. M1–M4). The number of model simulations considered was 1 for computational models M1 and M2, 1500 for M3, and 500 for M4. The number of time points recorded for each model simulation was 30001 for computational model M1, 330 for M2, 103 for M3, and 1000 for M4.

	Execution time (seconds)			
	M1	M2	M3	M4
Model simulation	37.2183	1.1339	1.7860	329.5979
Convert simulation output to csv format	0.3333	0.0185	1.3125	2.6153
Generate MSTML subfiles	25.5179	25.1534	12.0642	64.8183
Merge subfiles into single MSTML file	31.2068	0.4352	1.6562	2.8784

(i.e. 30001), and the stochastic nature and high complexity associated with the model. Conversely the most time consuming step for the uterine contractions of labour (i.e. 25.1534s) and *Xenopus laevis* cell cycle (i.e. 12.0642s) case studies was generating the MSTML subfiles due to the spatial regions which have been automatically detected and analysed for each spatial state variable considered.

The least time consuming step for all case studies was converting the model simulation output to csv format.

6.4 Formal specification

The generated MSTML files representing the behaviour of the computational models are employed during the evaluation of the formal specification described below using quotes from the original paper introducing the models, derived natural language statements and corresponding PBLMSTL statements.

6.4.1 Rat cardiovascular system dynamics

1. **Quote from paper:** “During the interval marked Valsalva the thoracic pressure is increased from the baseline value of -4 mmHg to the value of 16 mmHg. After 10 s of elevated pressure, thoracic pressure is returned to the baseline value” (Beard et al., 2012).

Natural language: The probability is greater than 0.9 that after initiating the Valsalva manoeuvre (time = 5000 ms) the thoracic pressure increases from the baseline value -4 to 16 for 10 seconds (time interval [5001 ms, 14999 ms]), and then drops back to the baseline value -4. The corresponding rephrased natural language statement is that the probability is greater than 0.9 that

after initiating the Valsava manoeuvre (time = 5000 ms) the thoracic pressure $\{P_{th}\}$ (corresponding to scale and subsystem OrganSystem.Cardiovascular) increases from the baseline value -4 to 16 for 10 seconds (time interval [5001 ms, 14999 ms]), and then drops back to the baseline value -4.

PBLMSTL: $P > 0.9 [(G [0, 5000] (\{P_{th}\} (scaleAndSubsystem = OrganSystem.Cardiovascular) = -4)) \wedge (G [5001, 14999] (\{P_{th}\} (scaleAndSubsystem = OrganSystem.Cardiovascular) = 16)) \wedge (G [15000, 30000] (\{P_{th}\} (scaleAndSubsystem = OrganSystem.Cardiovascular) = -4))].$

2. **Quote from paper:** “During the initial phase of the response” ... “the increase in aortic pressure causes a transient decrease in heart rate via the baroreflex” (Beard et al., 2012).

Natural language: The probability is greater than 0.9 that during the initial phase of the response (time interval [5001 ms, 6500 ms]) the aortic pressure increases and the heart rate decreases. The corresponding rephrased natural language statement is that the probability is greater than 0.9 that during the initial phase of the response (time interval [5001 ms, 6500 ms]) the aortic pressure $\{P_{ao}\}$ (corresponding to scale and subsystem Cellular.Baroreceptor) increases and the heart rate $\{HR\}$ (corresponding to scale and subsystem OrganSystem.Cardiovascular) decreases. Since the values of $\{P_{ao}\}$ and $\{HR\}$ are continuously oscillating we check if the maximum $\{P_{ao}\}$ value in time interval [5001 ms, 6500 ms] is larger than the maximum $\{P_{ao}\}$ value in time interval [4800 ms, 5000 ms], and if the minimum $\{HR\}$ value in time interval [5001 ms, 6500 ms] is smaller than the minimum $\{HR\}$ value in time interval [4800 ms, 5000 ms].

PBLMSTL: $P > 0.9 [(max([5001, 6500] \{P_{ao}\} (scaleAndSubsystem = Cellular.Baroreceptor)) > max([4800, 5000] \{P_{ao}\} (scaleAndSubsystem = Cellular.Baroreceptor))) \wedge (min([5001, 6500] \{HR\} (scaleAndSubsystem = OrganSystem.Cardiovascular)) < min([4800, 5000] \{HR\} (scaleAndSubsystem = OrganSystem.Cardiovascular)))].$

3. **Quote from paper:** “Following the initial response of increased” (aortic) “pressure, pressure begins to drop as a result of elevated thoracic pressure impeding venous flow to the heart. Heart rate increases in response to

the reduction in pressure predicted over the second phase of the Valsalva interval” (Beard et al., 2012).

Natural language: The probability is less than 0.1 that after the initial response phase (time interval [5001 ms, 6500 ms]) the aortic pressure continues to increase or stay constant, and the heart rate continues to decrease or stay constant throughout the remainder of the Valsalva interval (time interval [6501 ms, 14999 ms]). The corresponding rephrased natural language statement is that the probability is less than 0.1 that after the initial response phase (time interval [5001 ms, 6500 ms]) the aortic pressure $\{P_{ao}\}$ (corresponding to scale and subsystem Cellular.Baroreceptor) continues to increase or stay constant, and the heart rate $\{HR\}$ (corresponding to scale and subsystem OrganSystem.Cardiovascular) continues to decrease or stay constant throughout the remainder of the Valsalva interval (time interval [6501 ms, 14999 ms]). This statement can be rewritten as the minimum $\{P_{ao}\}$ value in time interval [6501 ms, 14999 ms] is greater or equal to the maximum $\{P_{ao}\}$ value in time interval [5001 ms, 6500 ms], and the maximum $\{HR\}$ value in time interval [6501 ms, 14999 ms] is smaller or equal to the minimum $\{HR\}$ value in time interval [5001 ms, 6500 ms] with probability less than 0.1.

PBLMSTL: $P < 0.1 [(min([6501, 14999] \{P_{ao}\}(scaleAndSubsystem = Cellular.Baroreceptor)) >= max([5001, 6500] \{P_{ao}\}(scaleAndSubsystem = Cellular.Baroreceptor))) \wedge (max([6501, 14999] \{HR\}(scaleAndSubsystem = OrganSystem.Cardiovascular)) <= min([5001, 6500] \{HR\}(scaleAndSubsystem = OrganSystem.Cardiovascular)))]$.

6.4.2 Uterine contractions of labour

4. **Quote from paper:** As indicated by the first rule in the “Formal presentation of the rules of the simulation” subsection of (Young and Barendse, 2014) the intrauterine pressure (*pressure*) is computed as the sum of all contractile activities (*act*) divided by the total number of regions (*#regions*) and their corresponding constant anatomic sensitivities (*anatomysens*):

$$pressure(t) = \sum act(i, j) / (\#regions * anatomysens(i, j)),$$

where t represents the current time point, respectively i and j encode Ox and Oy coordinates in the discretised 2D spatial domain.

Since both the total number of uterus regions and the associated anatomic sensitivities are constant throughout individual model simulations, the intrauterine pressure is directly proportional to the contractile activities of the uterine regions.

Natural language: The probability is greater than 0.9 that the intrauterine pressure increases/decreases with the contractile activity of uterine regions. The corresponding rephrased natural language statement is that the intrauterine pressure {Pressure} (corresponding to scale and subsystem Organ.Uterus) increases/decreases with the contractile activities (denoted in PBLMSTL as densities) of the uterine regions (corresponding to scale and subsystem Tissue.ContractileActivity) with probability greater than 0.9.

PBLMSTL: $P > 0.9 [G [1, 329] (((d(\{Pressure\}) (scaleAndSubsystem = Organ.Uterus)) > 0) \wedge ((d(sum(density(filter(regions, scaleAndSubsystem = Tissue.ContractileActivity)))) > 0))) \vee ((d(\{Pressure\}) (scaleAndSubsystem = Organ.Uterus)) < 0) \wedge ((d(sum(density(filter(regions, scaleAndSubsystem = Tissue.ContractileActivity)))) < 0))) \vee (d(\{Pressure\})(scaleAndSubsystem = Organ.Uterus)) = 0)].$

5. **Quote from paper:** “When a region is experiencing an action potential burst, the contractile activity is calculated by multiplying the passive tension by the action potential multiplier (a factor > 1)” (Young and Barendse, 2014).

Following on from the quote(s) corresponding to statement 4 the intrauterine pressure increases/decreases with the contractile activity.

Natural language: The probability is less than 0.1 that the intrauterine pressure decreases when the entire uterus experiences an action potential burst. The corresponding rephrased natural language statement is that the probability is less than 0.1 that the intrauterine pressure {Pressure} (corresponding to scale and subsystem Organ.Uterus) decreases when the entire uterus comprising all 16 regions (corresponding to scale and subsystem Tissue.BurstActivity) experiences an action potential burst.

PBLMSTL: $P < 0.1 [F [1, 329] ((d(\{Pressure\}) (scaleAndSubsystem = Organ.Uterus)) < 0) \wedge (min(area(filter(regions, scaleAndSubsystem =$

Tissue.BurstActivity))) = 16))].

6. **Quote from paper:** “When a region is in the refractory period, the tension” (or pressure) “is decreased by multiplying the passive tension by a factor < 1 (refractory multiplier)” (Young and Barendse, 2014).

Natural language: The probability is greater than 0.9 that the intrauterine pressure decreases when the entire uterus is in the refractory period. The corresponding rephrased natural language statement is that the probability is greater than 0.9 that the intrauterine pressure {Pressure} (corresponding to scale and subsystem Organ.Uterus) decreases when the entire uterus comprising all 16 regions (corresponding to scale and subsystem Tissue.RefractoryActivity) is in the refractory period.

PBLMSTL: $P > 0.9 [G [1, 329] (((max(area(filter(regions, scaleAndSubsystem = Tissue.RefractoryActivity))) = 0)) \Rightarrow ((d(\{Pressure\}(scaleAndSubsystem = Organ.Uterus)) < 0) \vee ((d(\{Pressure\}(scaleAndSubsystem = Organ.Uterus)) = 0) \wedge (\{Pressure\}(scaleAndSubsystem = Organ.Uterus) = 0.5))))].$

6.4.3 *Xenopus laevis* cell cycle

7. **Quote from paper:** “The activation of CDK1 drives the cell into mitosis” (Ferrell Jr. et al., 2011).

Natural language: The probability is greater than 0.9 that whenever the concentration of CDK1 reaches very high levels (in our case >96% of its maximum value) all cells will divide. The corresponding rephrased natural language statement is that the probability is greater than 0.9 that if the concentration (denoted in PBLMSTL as density) of CDK1 (corresponding to scale and subsystem Intracellular.CDK1) increases above 0.96 then all cells will divide i.e. the sum of the (densities · areas) of all regions covered by cells (corresponding to scale and subsystem Cellular.Embryo) will increase. The value of 96% corresponds to the normalized threshold concentration 0.5 for CDK1 ($[CDK1] \in [0, 0.515]$) chosen by the developers of the multiscale model (Starruß and Back, 2014) to trigger cellular division. Moreover the time interval considered in the PBLMSTL statements corresponds to the time interval considered in the model simulation.

PBLMSTL: $P > 0.9 [G [0, 100] (((count(density(filter(regions, scaleAndSubsystem = Intracellular.CDK1 \wedge density < 0.96))) = count(density(filter(regions, scaleAndSubsystem =$

$$\begin{aligned}
& \text{Intracellular.CDK1)))) \wedge \\
& (X (\text{count}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \\
& \text{Intracellular.CDK1} \wedge \text{density} > 0.96))) = \\
& \text{count}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \\
& \text{Intracellular.CDK1)))))) \Rightarrow \\
& (d(\text{sum}(\text{multiply}(\text{area}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \\
& \text{Cellular.Embryo})), \text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \\
& \text{Cellular.Embryo})))))) > 0)).]
\end{aligned}$$

8. **Quote from paper:** “the activation of APC, which generally lags behind CDK1, drives the cell back out of mitosis” (Ferrell Jr. et al., 2011).

Natural language: The probability is greater than 0.9 that whenever the average concentration of APC increases and reaches its local maximum value no cell will divide. The corresponding rephrased natural language statement is that the probability is greater than 0.9 that if the average concentration (represented in PBLMSTL as density) of APC (corresponding to scale and subsystem Intracellular.APC) reaches a local maximum value i.e. increases and then decreases, then no cell will divide i.e. the sum of the (densities · areas) of all regions covered by cells (corresponding to scale and subsystem Cellular.Embryo) will remain constant.

The time interval considered in the PBLMSTL statements corresponds to the time interval considered in the model simulation.

$$\begin{aligned}
& \text{PBLMSTL: } P > 0.9 [G [0, 100] (((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \\
& \text{scaleAndSubsystem} = \text{Intracellular.APC})))) > 0) \wedge \\
& (X (d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \\
& \text{Intracellular.APC})))) < 0))) \Rightarrow \\
& (X (d(\text{sum}(\text{multiply}(\text{area}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \\
& \text{Cellular.Embryo})), \text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \\
& \text{Cellular.Embryo})))))) = 0))).]
\end{aligned}$$

9. **Quote from paper:** Figure 5C in the reference paper considered (Ferrell Jr. et al., 2011) illustrates the oscillatory behaviour of the concentrations of APC, CDK1 and Plk1. This behaviour is additionally emphasized in the figure caption “Time course of the system, showing sustained limit cycle oscillations” (Ferrell Jr. et al., 2011).

Natural language: The probability is greater than 0.9 that the average concentrations of CDK1, Plk1 and APC increase and then decrease (i.e. oscillate) over time at least three times. The corresponding rephrased natural language statement is that the probability is greater than 0.9 that

the average concentrations (represented in PBLMSTL as densities) of CDK1, Plk1 and APC (corresponding to scale and subsystem Intracellular.CDK1, Intracellular.Plk1, and Intracellular.APC) increase and then decrease over time at least three times.

The minimum number of oscillations (in our case three) was chosen according to the number of oscillations displayed in Figure 5C of the reference paper considered (Ferrell Jr. et al., 2011). Moreover the time interval considered in the PBLMSTL statements corresponds to the time interval considered in the model simulation.

PBLMSTL: $P > 0.9 [(F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.CDK1})))) > 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.CDK1})))) < 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.CDK1})))) > 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.CDK1})))) < 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.CDK1})))) > 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.CDK1})))) < 0))))))))))) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.Plk1})))) > 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.Plk1})))) < 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.Plk1})))) > 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.Plk1})))) < 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.Plk1})))) > 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.Plk1})))) < 0))))))))))) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.APC})))) > 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.APC})))) < 0) \wedge (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \text{scaleAndSubsystem} = \text{Intracellular.APC})))) > 0)$

$$\begin{aligned}
 & scaleAndSubsystem = Intracellular.APC)))) > 0) \wedge \\
 & (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \\
 & scaleAndSubsystem = Intracellular.APC)))) < 0) \wedge \\
 & (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \\
 & scaleAndSubsystem = Intracellular.APC)))) > 0) \wedge \\
 & (F [0, 100] ((d(\text{avg}(\text{density}(\text{filter}(\text{regions}, \\
 & scaleAndSubsystem = Intracellular.APC)))) < 0))))))))))].
 \end{aligned}$$

6.4.4 Acute inflammation of the gut and lung

10. Quotes from paper:

- “tight junction (TJ) proteins are involved in the integrity of gut epithelial barrier function” ... “The TJ proteins that seem to be most affected in this situation are occludin ...” (An, 2008).
- “pulmonary epithelial cells behave very similarly to gut epithelial cells with respect to tight junction metabolism and epithelial barrier function” (An, 2008).
- “The impaired systemic oxygenation due to pulmonary leak arises from pulmonary epithelial barrier failure” (An, 2008).
- “impaired oxygenation into the endothelial lumen, which is summed across the surface of the model to produce a measure of systemic arterial oxygen content. This value will now represent the baseline ”oxy” level for all other systemic endothelial agents” (An, 2008).
- “ischemia was modeled as a percentage of the total endothelial surface rendered ”ischemic,” a state defined in the rules for the endothelial cell agents as an ”oxy” level < 60” (An, 2008).

Natural language: The probability is greater than 0.9 that if the level of cytoplasm occludin in the lung decreases then eventually the number of ischemic endothelial lung cells will increase. The corresponding rephrased natural language statement is that the probability is greater than 0.9 that if the value of $\{\text{LungOccludinCytoplasm}\}$ (corresponding to scale and subsystem $\text{Tissue.LungEpithelium}$) decreases then eventually the total area of the regions defined by ischemic endothelial lung cells (corresponding to scale and subsystem $\text{Cellular.LungEndotheliumIschemia}$) will increase.

PBLMSTL: $P > 0.9 [F [1, 999] ((d(\{\text{LungOccludinCytoplasm}\})$
 $scaleAndSubsystem = \text{Tissue.LungEpithelium})) < 0) \Rightarrow$

$$(F [1, 999] (d(sum(area(filter(regions, scaleAndSubsystem = Cellular.LungEndotheliumIschemia)))) > 0))).$$

11. Quotes from paper:

- “this variable is termed ”cell-damage-byproduct,” and it is calculated as a function of total endothelial damage” (An, 2008).
- “the levels of ”cell-damage-byproduct” will be the proxy for the unidentified compound that is produced in the ischemic gut and circulated to the lung, leading to inflammation of pulmonary endothelium” (An, 2008).

Natural language: The probability is greater than 0.9 that always an increase of the cell damage by-product in the gut will lead to an increase of the cell damage by-product in the lung. The corresponding rephrased natural language statement is that the probability is greater than 0.9 that always if the value of {GutCellDamageByproduct} (corresponding to scale and subsystem Tissue.GutEndothelium) increases, then eventually the value of {LungCellDamageByproduct} (corresponding to scale and subsystem Tissue.LungEndothelium) increases.

PBLMSTL: $P > 0.9 [G [1, 999] ((d(\{GutCellDamageByproduct\})(scaleAndSubsystem = Tissue.GutEndothelium)) > 0) \Rightarrow (F [1, 999] (d(\{LungCellDamageByproduct\})(scaleAndSubsystem = Tissue.LungEndothelium)) > 0))].$

12. Quotes from paper:

- “tight junction (TJ) proteins are involved in the integrity of gut epithelial barrier function” ... “The TJ proteins that seem to be most affected in this situation are occludin ...” (An, 2008).
- “A luminal compound that diffuses in response to TJ barrier failure” ... “is represented by ”gut-leak,” which is equal to the ”solute”” ... “that penetrates the failed barrier” (An, 2008).

Natural language: The probability is greater than 0.9 that if the level of cell wall occludin in the gut decreases then eventually the amount of solute leaking in the gut lumen will increase. The corresponding rephrased natural language statement is that the probability is greater than 0.9 that if the value of {GutOccludinCellwall} (corresponding to scale and subsystem Tissue.GutEpithelium) decreases then eventually the value of {GutLeak} (corresponding to scale and subsystem Organ.Gut) will increase.

PBLMSTL: $P > 0.9 [F [1, 999] ((d(\{GutOcludinCellwall\}) scaleAndSubsystem = Tissue.GutEpithelium)) < 0) \Rightarrow (F [1, 999] (d(\{GutLeak\})(scaleAndSubsystem = Organ.Gut)) > 0))].$

6.5 Model checking

Each PBLMSTL statement (stored in a separate file) was evaluated 500 times against the relevant MSTML file(s) considering the corresponding *MA* graphs. The main reason for evaluating each PBLMSTL statement against the corresponding MSTML file(s) 500 times is to compute the variation of the model checker execution time between runs for the deterministic computational models (M1 and M2), respectively the variation of the model checker results for the hybrid (M3) and stochastic (M4) computational models. In the case of the deterministic computational models (M1 and M2) the probabilistic black-box model checking algorithm was employed because it does not place a lower bound on the required number of model simulations and therefore is suitable for computational models which are simulated only once. Conversely, in case of the hybrid (M3) and stochastic (M4) computational models the frequentist statistical model checking algorithm was employed setting the probability of both type I and type II errors equal to 5%. The output of the statistical analysis of the model checking results is summarized in Table 6.3.

Empirical evidence shows that all computational models are valid relative to the formal specifications derived from the original papers introducing the models. Due to the deterministic nature of computational models M1 and M2, the corresponding model checking results were obtained by considering a single MSTML file, and therefore were identical across all 500 model checker executions. The main difference between the PBLMSTL statements considered is that in case of statements 1, 2, 4 and 6 the estimated probability p for them to hold, computed as #true MSTML divided by #total MSTML, was $p = (1 / 1) = 1$, respectively for the PBLMSTL statements 3 and 5 it was $p = (0 / 1) = 0$. However since the associated probabilistic specification for the PBLMSTL statements 1, 2, 4 and 6 was $p > 0.9$ (i.e. $1 > 0.9$), respectively $p < 0.1$ (i.e. $0 < 0.1$) for the PBLMSTL statements 3 and 5, all PBLMSTL statements hold. Conversely in case of the hybrid, respectively stochastic computational models M3 and M4, the model checking results were obtained by considering multiple MSTML files, and therefore in some cases (see Table 6.3, row corresponding to SId 7) variations between model checker executions were observed.

The average execution times corresponding to the validation of the deter-

Table 6.3: Model checking statistical analysis results for the rat cardiovascular system dynamics, the uterine contractions of labour, the *Xenopus laevis* cell cycle and the acute inflammation of the gut and lung case studies. Entries in the “MId” and “SId” columns represent the numeric identifiers associated with each computational model and its corresponding PBLMSTL statements. The “% true PBLMSTL” column describes what percentage of the 500 executions concluded that the PBLMSTL statement is true. “#total MSTML” represents the total number of MSTML files evaluated for the PBLMSTL statement; columns “#true MSTML” and “#false MSTML” represent the number of MSTML files for which the PBLMSTL statement was evaluated true, respectively false. “Execution time” presents the average model checking execution time for each PBLMSTL evaluation using the “minutes:seconds” format. “ μ ” and “ σ ” represent the mean and standard deviation.

MId	SId	% true PBLMSTL	#total MSTML		#true MSTML		#false MSTML		Execution time	
			μ	σ	μ	σ	μ	σ	μ	σ
1	1	100	1	0	1	0	0	0	0:17.67	0:0.12
	2	100	1	0	1	0	0	0	0:17.61	0:0.13
	3	100	1	0	0	0	1	0	0:17.80	0:0.36
2	4	100	1	0	1	0	0	0	0:0.55	0:0.01
	5	100	1	0	0	0	1	0	0:0.54	0:0.01
	6	100	1	0	1	0	0	0	0:0.54	0:0.01
3	7	100	28.79	2.04	28.61	1.62	0.19	0.44	0:35.35	0:2.44
	8	100	28	0	28	0	0	0	0:34.29	0:0.09
	9	100	28	0	28	0	0	0	0:35.36	0:0.99
4	10	100	28	0	28	0	0	0	1:27.39	0:0.72
	11	100	28	0	28	0	0	0	1:30.27	0:2.23
	12	100	28	0	28	0	0	0	1:27.03	0:0.65

ministic computational models M1 and M2 were smaller than for the hybrid, respectively stochastic computational models M3 and M4. This is due to the difference in the number of MSTML files considered which was one for computational models M1 and M2, respectively ≈ 28 for computational models M3 and M4. Moreover the variation in the average model checker execution times between the computational models M1 and M2, respectively M3 and M4 is due to the difference in the number of time points considered per model simulation which was 30001 for M1 and 330 for M2, respectively 103 for M3 and 1000 for M4. Average model checker execution times corresponding to the same computational model but different PBLMSTL statements were approximately equal throughout because most of the execution time is spent on reading the MSTML file(s) from disk and not the evaluation of the PBLMSTL statements.

By storing the PBLMSTL statements corresponding to a computational model in separate files each MSTML file read by the model checker from disk is evaluated against only one rather than all PBLMSTL statements. Therefore in order to reduce the average model checker execution time all PBLMSTL statements corresponding to the same computational model could be written into a single file. A comparison between average execution times obtained for 500 model

checker executions considering all PBLMSTL statements written into single, respectively multiple separate files are provided in Table 6.4. Regardless of the computational model considered the average execution time was approximately three times smaller when storing PBLMSTL statements in single rather than multiple separate files. The main reason for this is that the total number of MSTML files read from disk, which takes up most of the model checker execution time, was reduced by a factor equal to the number of PBLMSTL statements considered (i.e. 3).

Table 6.4: Comparison of average model checker execution times when PBLMSTL statements corresponding to a computational model are stored in a single, respectively multiple separate files. The “MId” column records the numeric identifiers associated with each computational model. Average model checker execution times corresponding to PBLMSTL statements stored in a single, respectively multiple separate files are provided in columns “Single file” and “Separate files”.

MId	Execution times (minutes:seconds)	
	Single file	Separate files
1	0:17.902	0:53.072
2	0:00.560	0:01.625
3	0:36.302	1:45.003
4	1:27.505	4:24.682

A comparison between the average execution times recorded for simulating the model, translating the output to MSTML and validating it using model checking is given in Figure 6.5.

The most time consuming step in the model checking workflow for both the cardiovascular system dynamics and acute inflammation of the gut and lung case studies is the model simulation. This is due to the large number of time points considered in case of the former, and the high complexity associated with the stochastic computational model in case of the latter. Conversely for the uterine contractions of labour case study the most time consuming step in the model checking workflow is generating the MSTML subfiles due to the additional need to automatically detect and analyse spatial regions of three types (i.e. corresponding to the contractile, burst and refractory activities) for each simulation time point. In contrast, the most time consuming step in the model checking workflow for the *Xenopus laevis* cell cycle case study is model checking due to the need to evaluate each PBLMSTL statement against multiple MSTML files. The least time consuming step in the model checking workflow for all case studies is converting the simulation output to csv format.

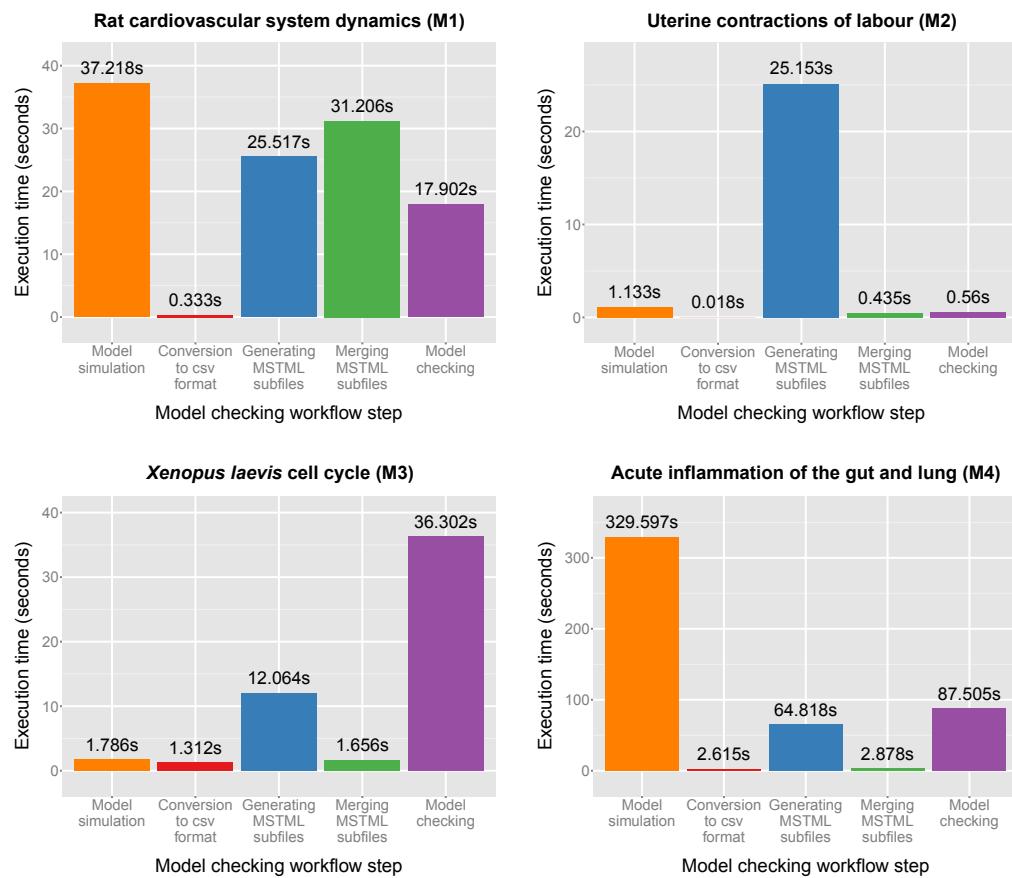


Figure 6.5: Average execution times (measured in seconds) corresponding to the validation of the rat cardiovascular system dynamics, the uterine contractions of labour, the *Xenopus laevis* cell cycle and the acute inflammation of the gut and lung computational models. Execution times were recorded for the computational model simulation, converting the output to csv format, generating MSTML subfiles for each considered time point, numeric state variable and spatial entity, merging the subfiles into a single MSTML file, and model checking.

6.6 Discussion

The multiscale multidimensional spatio-temporal meta model checking methodology enables the validation of multiscale computational models of biological systems relative to specifications describing the expected system behaviour. Computational models are either encoded directly as MSSpDESs or translated from a high-level modelling formalism into an equivalent MSSpDES representation. Time series data representing the model behaviour are pre-generated or generated on demand by the model checker and are processed by the multiscale spatio-temporal analysis module to automatically detect and analyse spatial entities across multiple levels of organization. The processed time series data are stored in MSTML files. Formal specifications against which the models are validated are encoded in PBLMSTL, and depending on the chosen model checking algorithm, are evaluated multiple times against MSTML files to determine the (in)correctness of the model. Although only the probabilistic black box (see rat cardiovascular system dynamics

and uterine contractions of labour case studies) and frequentist statistical model checking algorithms (see *Xenopus laevis* cell cycle and acute inflammation of the gut and lung case studies) were employed here, additional frequentist (i.e. based on Chernoff-Hoeffding bounds) and Bayesian (i.e. hypothesis testing, mean and variance estimate based) model checking algorithms are supported. To automate the entire validation process the approach was implemented in the model checker Mule which is made freely available online (source code, binary, Docker image) at <http://mule.modelchecking.org>.

6.6.1 Model validation and experimental data analysis

The multiscale multidimensional spatio-temporal meta model checking methodology is generic and supports computational models encoded using various high-level modelling formalisms because it is defined relative to time series data rather than the computational models used to produce them.

The only requirement is that the computational models can be translated from the high-level modelling formalism used initially to encode them into an equivalent MSSpDES representation. This is illustrated by the case studies considered which were formally encoded using ODEs (rat cardiovascular system dynamics), CAs (uterine contractions of labour), CPMs (*Xenopus laevis* cell cycle), ABMs (acute inflammation of the gut and lung) or combinations thereof.

In addition since the multiscale multidimensional spatio-temporal meta model checking methodology is defined relative to time series data, in principle, it could be employed to analyse experimental time series data recorded in the wet lab against given formal specifications, but this is beyond the scope of this thesis.

6.6.2 Automatic reconfiguration according to case study specific spatial entity types and measures

The meta model checking concept enables automatically reconfiguring the model checking method and its implementation according to case study specific spatial entity types (e.g. 3D spatial structure) and/or properties (e.g. minimum distance to a fixed point) not covered by our multiscale spatio-temporal analysis module. The only requirement is that a corresponding multiscale spatio-temporal analysis module is defined which can automatically detect and analyse the considered types of spatial entities. Such modules can be usually defined based on existing functions from the image processing literature.

The meta model checking approach was implemented using two instead of one compilation step. The first compilation step automatically translates a

configuration file recording the considered spatial entity types and properties into C++ source code. The second compilation step translates the C++ source code into an executable. The main reason for employing an additional compilation step rather than loading the configuration file at runtime and dynamically updating the behaviour of the model checker accordingly is that the latter approach could potentially negatively impact the performance of the model checker.

Using the meta model checking concept it is also possible to adapt the multiscale model checking approach to other domains of science where multiscale computational models are employed (e.g. astrophysics, energy, engineering, environmental science and materials science (Groen et al., 2014)). However this is beyond the scope of this thesis.

In addition the meta model checking concept could be additionally employed to reconfigure the model checker according to non-spatial (atomic) properties (e.g. second order derivative) previously not considered, and to employ representations different from rooted trees to encode the hierarchical structure of multiscale systems.

6.6.3 Scalability

The scalability of the entire multiscale model validation workflow depends on the scalability of the model simulation, multiscale spatio-temporal analysis and model checking steps. The execution time of the model simulation depends on the complexity of the system considered. Conversely the execution times of both the multiscale spatio-temporal analysis and the model checker depend on the size of the simulation output. In addition, the model checker execution time also depends on the formal specification. Our expectation is that scaling up to more complex systems will lead to an increase of the computational model complexity but not necessarily the size of the simulation output and/or formal specification. Therefore the expected scalability bottleneck of the entire model checking workflow is the model simulation and not the model validation step. This is supported by empirical evidence obtained from the case studies; the ratio between the maximum and minimum execution times for the model simulation step was ≈ 290 , ≈ 5 for the multiscale spatio-temporal analysis and ≈ 156 for the model checking step. In addition it would be possible to speed up the model checking step by evaluating MSTML files against the formal specification in parallel rather than sequentially as it is done now.

Summary

The efficacy and usefulness of the multiscale multidimensional spatio-temporal meta model checking methodology was illustrated against four multiscale computational models of biological systems encoding the rat cardiovascular system dynamics, the uterine contractions of labour, the *Xenopus laevis* cell cycle and the acute inflammation of the gut and lung. One of the conclusions drawn is that the methodology is generic and supports computational models encoded using various high-level modelling formalisms because it is defined relative to time series data and not the models used to generate them. Moreover using the meta model checking approach the methodology can be reconfigured automatically according to case study specific spatial entity types and measures, which can correspond to different numbers of spatial dimensions, or are characteristic to other domains of science. Finally empirical evidence shows that the approach scales well and therefore is expected to be also applicable to computational models of more complex systems.

7

CHAPTER

Conclusions, open problems and future work

This concluding chapter highlights the main contributions of the thesis and its potential practical applications. Corresponding open problems that could be addressed in the future are described in the end.

7.1 Summary and conclusions

Due to the decreasing costs of computational power and increasing amount of biological data available, there is an unprecedented opportunity to build and use multiscale (spatial) computational models of complex biological systems to gain a systems level understanding of how biological organisms function.

However insights gained from such computational models can be employed for real-life applications only if the models have been validated first. One of the most frequently employed *in silico* approaches for validating computational models of biological systems is called model checking. The main limitation of the existing model checking approaches (see Chapter 2) is that they cannot be employed to validate (multiscale) spatial computational models of complex biological systems. The main reason for this is that they do not consider how properties of (emergent) spatial structures (e.g. area of multicellular population) change over time and across multiple scales (e.g. cellular and organ). Moreover validating such computational models in the *in vitro* environment is often difficult due to the lack of biological data from all scales, and the interactions between scales.

This thesis addresses the limitations of existing model checking approaches by introducing a novel multiscale multidimensional spatio-temporal meta model check-

ing methodology that enables validating both uniscale and multiscale (spatial) computational models of complex biological systems relative to formal specifications describing how both numeric properties (e.g. concentrations) and properties of (emergent) spatial structures are expected to change over time and/or across multiple scales.

The description of the methodology is separated into two parts. The first part presents the multidimensional spatio-temporal model checking methodology which enables the validation of uniscale spatial computational models (see Chapters 3 and 4), whereas the second part builds on the first and describes the multiscale multidimensional spatio-temporal meta model checking methodology that enables the validation of both uniscale and multiscale computational models (see Chapters 5 and 6).

7.1.1 Multidimensional spatio-temporal model checking

The multidimensional spatio-temporal model checking methodology (see Chapter 3) was developed to enable the validation of spatial computational models of biological systems with respect to both how numeric properties and properties of (emergent) spatial structures are expected to change over time. Existing model checking approaches cannot be employed to validate such models because they only consider how numeric properties change over time.

Using the multidimensional spatio-temporal model checking approach, spatial computational models are validated relative to a given formal PBLSTL specification as follows. First of all the computational model is simulated to generate time series data. The resulting time series data is passed as input to the spatio-temporal analysis module which can automatically detect and analyse how specific types of spatial entities change over time. The output of the spatio-temporal analysis module is merged with time series data encoding the evolution of numeric properties over time and is formatted according to the standard xml-based data representation format STML. The formal PBLSTL specification and the STML files encoding the modelled system behaviour are taken as input by the model checker which uses an approximate probabilistic model checking algorithm to determine if the computational model is valid relative to the formal specification (considering the minimum confidence level specified by the user). The output of the model checker indicates if the model is valid relative to the given formal specification, and records for which STML files the formal specification evaluated true, respectively false.

The novel components introduced by the multidimensional spatio-temporal

model checking methodology are:

- The theoretical model SSpDES for encoding how stochastic (biological) systems evolve over time and space;
- The spatio-temporal analysis modules for automatically detecting and analysing how spatial properties of clusters/regions change over time;
- The standard representation format STML for recording time series data describing how both numeric and spatial properties change over time;
- The quantitative probabilistic spatio-temporal logic PBLSTL for encoding formal specifications describing how systems are expected to change over time and space.

The approximate probabilistic model checking algorithms considered were introduced in the papers referenced in Table 2.1.

To enable the automatic validation of computational models, the multidimensional model checking approach was implemented in the model checker Mudi which is made freely available online at <http://mudi.modelchecking.org>.

The applicability and efficacy of the model checker was illustrated against two uniscale spatial computational models encoding phase variation in bacterial colony growth and the chemotactic aggregation of cells (see Chapter 4).

The computational model corresponding to the former case study describes how sector-like patterns (i.e. regions) emerge in bacterial colonies with phase variable genes, and how their properties change over time. Understanding the potential relationship between the development of sector-like patterns and the mutation and/or fitness rates of bacteria could enable automatically predicting the latter by only observing the former. In principle the parameters of the computational model could be changed until the *in silico* generated sector-like patterns match the ones observed *in vitro/in vivo*. If such a parameter combination is found, the corresponding mutation and/or fitness rates of the bacteria could be directly extracted from the computational model. Using the multidimensional spatio-temporal model checking approach it would be possible to verify if the sector-like patterns develop as expected, and if they evolve similarly in the *in silico* and in the *in vitro* environment. Due to the lack of experimental data it was not possible to check if a relationship between sector-like patterns and the mutation and/or fitness rates of the bacteria truly exists, and this could be a potential direction for future work.

Conversely the computational model corresponding to the latter case study describes the chemotactic aggregation of a population of cells randomly distributed

in the environment. This case study does not have a specific intended practical application. It was chosen mainly to illustrate the applicability of the multidimensional model checking approach for the automatic detection and analysis of clusters in time series data. The constituent elements of the clusters are not explicitly constrained. Therefore other types of entities different from cells (e.g. molecules, tissues etc.) are supported.

Following on from the considered case studies, one of the main advantages of the model checker Mudi is that it supports computational models encoded using various modelling formalisms because it is defined relative to time series data and not the computational models used to generate them. For instance the computational model corresponding to the phase variation case study was encoded using Coloured Stochastic Petri Nets, whereas the computational model corresponding to the chemotactic aggregation of cells case study was encoded as a Cellular Potts model integrated with a system of partial differential equations. Therefore the model checker Mudi could be integrated with various existing model development workflows, and enable computational biologists to efficiently build reliable spatial computational models of biological systems.

Conversely one of the main disadvantages of the multidimensional spatio-temporal model checking approach is that it enables validating spatial computational models only relative to a fixed set of spatial entity types and measures. Therefore any computational model where different types of spatial entities (e.g. 3D structure) and/or measures (e.g. volume) are of interest cannot be validated using this approach. Moreover the multidimensional spatio-temporal model checking approach does not enable to explicitly distinguish between different scales. Therefore multiscale computational models cannot be validated by considering how changes at one scale reflect at another scale and vice versa. These limitations are addressed by the multiscale multidimensional spatio-temporal meta model checking methodology described below.

7.1.2 Multiscale multidimensional spatio-temporal meta model checking

The multiscale multidimensional spatio-temporal meta model checking methodology (see Chapter 5) was developed to enable the validation of both uniscale and multiscale (spatial) computational models relative to formal specifications describing how numeric properties and/or properties of case study specific (emergent) spatial structures are expected to change over time and across multiple scales. Therefore the main advantage of this methodology, compared to the multidimen-

sional spatio-temporal model checking approach, is that it can be automatically reconfigured to enable the validation of spatial computational models considering spatial entity types and/or measures specific to particular case studies, and it is possible to explicitly distinguish between different scales.

The procedure for validating a multiscale (spatial) computational model relative to a formal PBLMSTL specification using the multiscale multidimensional spatio-temporal meta model checking approach comprises the following steps. First of all the case study specific spatial entity types and measures need to be specified in a configuration file. This configuration file is then taken as input by a module which automatically generates a corresponding instance of the meta model checker Mule. Secondly the computational model is simulated to generate time series data. These time series data are then processed and translated to MSTML, which is a multiscale extension of STML. In contrast to the uniscale multidimensional model checking approach, spatial entities are detected and analysed using multiscale rather than uniscale spatio-temporal analysis modules. The formal PBLMSTL specification and the collection of MSTML files representing the modelled system behaviour are taken as input by the generated instance of the meta model checker Mule which decides if the formal specification holds for the computational model. Similarly to Mudi, the output of the generated meta model checker instance indicates if the computational model is valid or not, and for which MSTML files the formal PBLMSTL specification evaluated true, respectively false.

The novel components introduced by the multiscale multidimensional spatio-temporal meta model checking methodology are:

- The theoretical model MSSpDES for encoding how multiscale stochastic (biological) systems evolve over time and space;
- The multiscale spatio-temporal analysis modules for automatically detecting and analysing how case study specific spatial entities and their properties change over time;
- The standard representation format MSTML for recording time series data describing both how numeric and spatial properties change over time and across multiple scales;
- The quantitative probabilistic multiscale spatio-temporal logic PBLMSTL for encoding formal specifications describing how systems are expected to change over time, space, and across multiple scales;

- The meta model checking approach which enables automatically reconfiguring the model checker according to case study specific spatial entity types and/or measures.

Similarly to the multidimensional spatio-temporal model checking approach, the approximate probabilistic model checking algorithms considered were introduced in the papers referenced in Table 2.1.

The multiscale multidimensional spatio-temporal meta model checking approach was implemented in the model checker Mule which is made freely available online at <http://mule.modelchecking.org> in executable and source code format, as well as a Docker image.

The efficacy and applicability of the meta model checker Mule was illustrated against four systems biology computational models previously published in the literature encoding the rat cardiovascular system dynamics, the uterine contractions of labour, the *Xenopus laevis* cell cycle and the acute inflammation of the gut and lung (see Chapter 6). The potential practical applications of the computational models have been previously described by their original authors, and therefore will not be restated here. Moreover the formal specifications against which the computational models were validated have been derived from the original papers introducing the models. The main reason for choosing computational models that encode different biological systems and/or organisms was to illustrate the wide applicability of the multiscale multidimensional spatio-temporal meta model checking approach.

Following on from the validation of the four computational models, and due to the same reasons as for the multidimensional spatio-temporal model checking approach, the multiscale multidimensional spatio-temporal meta model checking methodology enables the validation of computational models encoded using different modelling formalisms. Moreover the methodology is generic because it can be automatically reconfigured according to case study specific types of spatial entities (e.g. 3D structure) and/or measures (e.g. volume).

Using the multiscale multidimensional spatio-temporal meta model checking methodology computational biologists will be able to efficiently construct reliable multiscale computational models of complex biological systems. These computational models could then be used in systems biology to gain a systems level understanding of biological systems, and to generate new hypotheses for driving experiments in the wet-lab. Moreover such computational models could be potentially translated into systems medicine where they could be employed to generate patient specific predictions of how a disease and its treatment reflect across multiple levels of biological organization (Boissel et al., 2015).

7.2 Open problems and future work

The multiscale multidimensional spatio-temporal meta model checking methodology could be extended in the future to address the following open problems.

7.2.1 Analysis of time series data recorded in the *in vitro* environment

The model checking approach is defined relative to time series data and not computational models. Therefore, in principle, it could be employed to check if a given formal PBLMSTL specification holds for any time series data, regardless if it was generated through computational model simulation or recorded in the wet-lab.

Since in this thesis model checking was employed to validate computational models, formal PBLMSTL specifications were evaluated only against *in silico* generated time series data.

However the ability to check if the same formal PBLMSTL specification holds for *in silico* and *in vitro/in vivo* generated time series data could be employed as a comparison measure between computational models and the corresponding living organisms. Such a comparison measure could be useful for systems biology applications to ensure that the computational model behaves similarly to the corresponding real-life system, and for synthetic biology applications to check if the behaviour of a computational model employed as a prototype/design is reproduced in a reliable manner by the corresponding synthetically modified living organism.

7.2.2 Validation of computational models from other domains of science

Considering that the model checking methodology can be automatically reconfigured according to case study specific types of spatial entities and measures, it could be potentially employed to validate multiscale computational models of non-biological systems.

Other domains of science where multiscale computational models are employed, and therefore the model checking methodology could be potentially useful, include astrophysics, chemistry, engineering and environmental science. As highlighted by Hoekstra (Hoekstra et al., 2014), there is a need for a general multiscale model validation framework in these scientific domains.

Therefore building on from the model checking methodology introduced here a potential long term aim could be the development of a unified general-purpose multiscale computational model checking framework that can be employed to validate multiscale computational models from a wide range of scientific disciplines. Such a unified model validation framework could enable verifying the reliability of multiscale computational models in a standard reproducible manner.

7.2.3 Parameter estimation, model construction and robustness computation

Existing model checking approaches have been employed in systems biology to solve model validation, parameter estimation, model construction and robustness computation problems. Conversely, in this thesis the efficacy of the multiscale multidimensional spatio-temporal meta model checking methodology was illustrated only against model validation problems.

Therefore an additional direction for future work could be extending the methodology to address the other three classes of problems. If successful, these extensions could enable fitting (semi-)automatically the parameters and/or structure of models to experimental data, and estimating their robustness to perturbations.

A specific practical application of the extended methodology could be the (semi-)automatic construction of multiscale computational models that are fitted to the characteristics of specific (groups of) individuals, and could be used to deliver personalized treatments.

7.2.4 Distributed multiscale model checking web service

Due to the increasing availability of experimental data and decreasing costs of computational power more complex multiscale computational models of biological systems could be potentially developed in the (near) future.

In order to ensure that the execution of the model checkers Mudi and Mule will scale well with the increasing complexity of the computational models, formal PBLMSTL specifications could be evaluated against time series data in parallel rather than sequentially as they are now. Therefore the execution of the model checkers could be distributed across multiple processing units/devices.

Moreover, a cloud based multiscale multidimensional spatio-temporal meta model checking web service could be set up to enable users to validate their computational models online without the need to install additional software.

7.2.5 Alternative model representations and spatio-temporal analysis modules

It is assumed throughout that computational models are encoded as MSSpDESs which means that any computational model encoded using a different formalism must be translated to a corresponding MSSpDES representation subject to potential approximation errors (e.g. consider continuous computational models). To overcome this limitation alternative representations could be employed instead.

Similarly, the spatio-temporal analysis modules are currently restricted to pseudo-3D spatial entity types and measures but could be extended in the future to other numbers of dimensions. Such modules can often be implemented using existing functions from the image processing literature.

7.2.6 Usability improvement

Finally users of the model checkers Mudi and Mule could potentially find it difficult to write PBLSTL, respectively PBLMSTL specifications due to the low-level constructs employed in these formal languages.

To improve the usability of the model checkers and enable writing complex statements in a clear and concise manner high-level language constructs could be added to PBLSTL and PBLMSTL. For instance the oscillatory behaviour of a numeric state variable X could be encoded using a high-level function `oscillate(X, n)` instead of specifying explicitly that the value of X increases/decreases and then decreases/increases n times; see PBLMSTL statement 9 in Subsection 6.4.3 for an example.

In addition a questionnaire could be designed to investigate how difficult/easy it is for computational modellers to use the model checkers Mudi and Mule, what changes (if any) are required, and what new features should be added in the future. Moreover in order to ensure that the model checkers continue to address computational modellers' needs on the long term a multiscale computational model validation working group could be additionally set up. Such a working group could enable having relevant discussions/meetings with users on a regular basis, and promoting the model checkers within the research community.

APPENDIX A

Approximate probabilistic model checking approaches

A description of the approximate probabilistic model checking approaches considered in this thesis is provided below.

In order for such approaches to be employed the following properties must be satisfied (Reijsbergen et al., 2014):

1. The computational model can be simulated according to a well-defined probability measure;
2. The resulting computation paths are generated in a finite amount of time; similarly the considered formal specification ψ can be evaluated against each computation path in a finite amount of time;
3. There is either no nondeterminism in the model or it is resolved by a well-defined policy or scheduler.

For consistency purposes model simulation evaluations are represented throughout as a sequence X_1, X_2, \dots, X_n of iid Bernoulli variables.

A.1 Chernoff-Hoeffding bounds based model checking

Approximate probabilistic model checking (Hérault et al., 2004) is a simulation-based approach which estimates the true probability p of a logic property being true.

The approximation error of the method is controlled using a derived form of the Chernoff-Hoeffding inequalities (Hoeffding, 1963):

$$P[|\bar{X} - p| > \epsilon] < 2e^{-\frac{N\epsilon^2}{4}}, \quad (\text{A.1})$$

where \bar{X} is the sample mean,

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i,$$

and $0 < \epsilon < 1$.

Equation A.1 states that the probability of \bar{X} to deviate from the true probability p more than ϵ is bounded above by $\delta = 2e^{-\frac{N\epsilon^2}{4}}$. The number of simulations N required to meet the constraints of Equation A.1 is computed with respect to parameters ϵ and δ :

$$N = \frac{4}{\epsilon^2} \log \left(\frac{2}{\delta} \right),$$

where $0 < \epsilon, \delta < 1$. Therefore ϵ and δ are input parameters of the algorithm.

A detailed description of the approach and corresponding examples are given by Hérault et al. (Hérault et al., 2004). From the point of view of the computational complexity the algorithm is linear with respect to the value of δ and quadratic with respect to the value of ϵ .

A.2 Frequentist statistical model checking

Frequentist statistical model checking methods (Younes et al., 2006; Younes, 2005b) verify if a logic property ϕ holds for a model \mathcal{M} using acceptance sampling tests (Younes and Simmons, 2002).

Let us assume that ϕ is a logic property of the form $P_{\geq\theta}[\psi]$. The null hypothesis $H_0 : p \geq \theta$ is tested against the alternative hypothesis $H_1 : p < \theta$ and model simulations are evaluated until one of the hypotheses is accepted. In case ϕ is of the form $P_{\leq\theta}[\psi]$ the roles of the null and alternative hypotheses switch. Moreover in terms of hypothesis testing $P_{>\theta}[\psi]$ is equivalent to $P_{\geq\theta}[\psi]$, and $P_{<\theta}[\psi]$ is equivalent to $P_{\leq\theta}[\psi]$.

The approximation error of this method is determined by the strength $\langle\alpha, \beta\rangle$ of the acceptance sampling test, where

- $\alpha = P[H_1 \text{ is accepted} \mid H_0 \text{ is true}]$ (Probability of type I error);

- $\beta = P[H_0 \text{ is accepted} \mid H_1 \text{ is true}]$ (Probability of type II error).

In case the probability θ specified in the logic property ϕ is close to the true probability p a large number of simulations is required to validate a hypothesis and it is impossible to ensure a low probability of type I and type II errors simultaneously; see (Younes and Simmons, 2006) for details.

Therefore the hypothesis testing problem constraints are relaxed. An indifference region $(p - \delta, p + \delta)$ of width 2δ is introduced where neither of the two hypotheses is true. In this new setting three hypotheses are considered:

- The null hypothesis $H'_0 : p \geq \theta + \delta$;
- The alternative hypothesis: $H'_1 : p < \theta - \delta$;
- The undecided hypothesis: $H_2 : \theta - \delta \leq p < \theta + \delta$.

Using two acceptance sampling tests it is possible to decide if $\phi \equiv P_{\geq\theta}[\psi]$ holds:

Test 1 with strength $\langle\alpha, \gamma\rangle$ ($H_0 : p \geq \theta, H'_1 : p < \theta - \delta$);

Test 2 with strength $\langle\gamma, \beta\rangle$ ($H'_0 : p \geq \theta + \delta, H_1 : p < \theta$),

where γ represents the probability of undecided results. Whenever H_0 and H'_0 are accepted ϕ is declared to hold. Conversely if H_1 and H'_1 are accepted then ϕ is declared not to hold. Otherwise the validity of ϕ is undecided.

Two types of acceptance sampling plans can be employed to determine the true hypothesis:

- Single acceptance sampling plan;
- Sequential acceptance sampling plan.

A.2.1 Single acceptance sampling plan

Single sampling plan methods compute the values of the acceptance number c and the smallest number of required simulations n which ensure that the strength of the test $\langle\alpha, \beta\rangle$ is guaranteed. The number of simulations n is fixed in the beginning and the hypothesis H_0 is accepted if

$$\sum_{i=1}^n x_i > c,$$

where x_i represents an observation of the i -th Bernoulli variable ($1 = \text{true}, 0 = \text{false}$). Otherwise the hypothesis H_1 is true. Values for c and n can be obtained

from a precomputed table of values (Grubbs, 1949) or approximated using binary search (Younes, 2005b, p. 21).

The disadvantage of employing single sampling plans is that the number of required simulations is fixed and not updated while evaluating the simulations. This means that although sufficient evidence is available to validate one of the hypotheses the method will not stop early. For instance if the first $c < n$ simulations have been evaluated true H_0 is validated and further simulations are irrelevant to the final result. However since n is fixed the remaining $n - c$ simulations will be generated and evaluated as well.

A.2.2 Sequential acceptance sampling plan

Sequential acceptance sampling plans address this issue by verifying after each simulation evaluation if sufficient evidence is available to validate one of the hypotheses. An efficient sequential acceptance sampling plan is Wald's sequential probability ratio test (Wald, 1945).

After evaluating each simulation a value is computed

$$f_m = \prod_{i=1}^m \frac{P[X_i = x_i \mid p = p_1]}{P[X_i = x_i \mid p = p_0]} = \frac{p_1^d(1-p_1)^{m-d}}{p_0^d(1-p_0)^{m-d}},$$

where m is the number of simulations evaluated so far, $d = \sum_{i=1}^m x_i$ is the number of true evaluations, $p_0 = \theta + \delta$ and $p_1 = \theta - \delta$. The hypothesis H_0 is accepted if $f_m \leq B$, and hypothesis H_1 is accepted if $f_m \geq A$. Otherwise, if $B \leq f_m \leq A$, then insufficient evidence is available and additional simulations are required.

In practical applications an approximation of the optimal A and B values is used in order to reduce the overall complexity of the method (Wald, 1945, Section 3.4):

$$A = \frac{1-\beta}{\alpha}; \quad B = \frac{\beta}{1-\alpha}.$$

The strength of the test $\langle \alpha', \beta' \rangle$ given by the approximated A and B values closely matches the initial strength $\langle \alpha, \beta \rangle$. Wald (Wald, 1945) has shown that

$$\alpha' \leq \frac{\alpha}{1-\beta}; \quad \beta' \leq \frac{\beta}{1-\alpha}; \quad \alpha' + \beta' \leq \alpha + \beta,$$

which means that at least one of the inequalities $\alpha' \leq \alpha$ and $\beta' \leq \beta$ must hold. Moreover if the values of α and β are small (e.g. less than 5%) then $\frac{\alpha}{1-\beta} \approx \alpha$ and $\frac{\beta}{1-\alpha} \approx \beta$ which means both inequalities hold.

The input parameters of the algorithm are α , β and δ . A detailed description of the statistical model checking algorithm is given by Younes and Simmons (Younes and Simmons, 2006) and an example of a model checker implementing the algorithm is described by Younes (Younes, 2005c).

A.3 Statistical black-box model checking

Statistical black-box model checking, initially introduced by Sen et al. (Sen et al., 2004) and further extended by Younes (Younes, 2005a), verifies if a logic property ϕ holds for a model \mathcal{M} using statistical hypothesis testing based on p-values. In contrast to statistical model checking (see Section A.2) a fixed number of simulations is provided and the model cannot be simulated on demand.

Let us denote the sum of all Bernoulli variables by $Y = \sum_{i=1}^n X_i$. Then Y has a binomial distribution with the cumulative distribution function:

$$F(c; n, p) = \sum_{i=0}^c \binom{n}{i} p^i (1-p)^{n-i}.$$

If ϕ is a logic property of the form $P_{\geq\theta}[\psi]$ then the null hypothesis $H_0 : p \geq \theta$ is tested against the alternative hypothesis $H_1 : p < \theta$. A p-value is computed for each hypothesis and the hypothesis with the lowest p-value is accepted (Younes, 2005a):

$$\begin{aligned} p_{H_0} &= 1 - F(d-1; n, \theta); \\ p_{H_1} &= F(d; n, \theta), \end{aligned}$$

where n is the number of Bernoulli variables, $d = \sum_{i=1}^m x_i$ is the number of true evaluations, and θ is the probability specified within the logic property ϕ . In case the p-values are equal the alternative hypothesis H_1 is accepted.

More details and usage examples regarding the extended statistical black-box model checking method are given by Younes (Younes, 2005a).

A.4 Bayesian mean and variance estimate based model checking

Bayesian mean and variance estimate based model checking (Langmead, 2009) verifies if a logic property ϕ holds for a model \mathcal{M} by estimating the true probability p of ϕ being true. In contrast to the frequentist model checking approach (see

Section A.1) the present approach uses prior information during the estimation process.

Simulation evaluations are represented as iid Bernoulli variables with the probability of the logic property ϕ being true equal to p . Therefore we can assume that the posterior has a Bernoulli distribution with parameter p . The conjugate prior of a Bernoulli distribution is a beta distribution with shape parameters α and β . Thus the prior information considered during the estimation process is represented by a beta distribution. If prior information is unavailable an unbiased prior can be used ($\alpha = 1$, $\beta = 1$). Both shape parameters are provided by the user as input to the algorithm.

Considering the user-defined beta distribution shape parameters α and β the algorithm updates the estimate of the true probability ρ and variance ν after evaluating ϕ for each newly generated sample. The formulae for computing the estimates ρ and ν are:

$$\rho = \frac{k + \alpha}{\alpha + \beta + n} ;$$

$$\nu = \frac{(\alpha + k)(n - k + \beta)}{(\alpha + n + \beta)^2(\alpha + n + \beta + 1)} ,$$

where n represents the number of generated samples, k represents the number of samples for which ϕ was evaluated true, and $\alpha > 0$ and $\beta > 0$ are the beta distribution shape parameters.

New samples are generated and the estimates ρ and ν are updated until the condition $\nu < T$ is true, where $T > 0$ is a user-defined threshold value provided as input to the algorithm. Considering that the logic property ϕ is of the form $P_{\geq\theta}[\psi]$ ($P_{\leq\theta}[\psi]$) ϕ will be evaluated true if $\rho \geq \theta$ ($\rho \leq \theta$).

A detailed description of the algorithm and usage examples are given by Langmead (Langmead, 2009).

A.5 Bayesian statistical model checking

Bayesian statistical model checking (Jha et al., 2009a,b) verifies if a logic property ϕ holds for a model \mathcal{M} using statistical hypothesis testing. In contrast to frequentist statistical model checking approaches (see Section A.2) the present approach employs prior information for validating one of the hypotheses.

Let us assume that ϕ is a logic property of the form $P_{\geq\theta}[\psi]$. The null hypothesis $H_0 : p \geq \theta$ is tested against the alternative hypothesis $H_1 : p < \theta$ and model simulations are evaluated until one of the hypotheses is accepted. In case ϕ is of

the form $P_{\leq \theta}[\psi]$ the roles of the null and alternative hypotheses switch.

A measure of relative confidence in H_0 with respect to H_1 is defined called Bayes factor \mathcal{B} . The value of \mathcal{B} considering a sequence of simulation evaluations $s = (x_1, x_2, \dots, x_n)$ and hypotheses H_0 and H_1 is computed as follows:

$$\mathcal{B} = \frac{P(s | H_0)}{P(s | H_1)} = \frac{P(H_0 | s)P(H_1)}{P(H_1 | s)P(H_0)}.$$

Similarly to the Bayesian mean and variance estimation based model checking approach the posterior is assumed to have a Bernoulli distribution. Therefore the conjugate prior has a beta distribution with shape parameters $\alpha > 0$ and $\beta > 0$. Both beta distribution shape parameters are provided as input to the algorithm.

Considering these assumptions Jha et al. (Jha et al., 2009b) show that the value of \mathcal{B} can be computed with respect to the cumulative beta distribution function:

$$\mathcal{B} = \frac{1}{F_{(x+\alpha, n-x+\beta)}(\theta)} - 1,$$

where n represents the total number of simulation evaluations, x represents the number of simulations for which ϕ was evaluated true, and

$$F_{(\alpha', \beta')}(\theta) = I_\theta(\alpha', \beta')$$

is the cumulative beta distribution function with shape parameters $\alpha' = x + \alpha$ and $\beta' = n - x + \beta$ such that $I_\theta(\alpha', \beta')$ is the regularized incomplete beta function.

The null hypothesis H_0 is accepted if $\mathcal{B} > T$ where T is a user-defined threshold value provided as input to the algorithm. Conversely the alternative hypothesis H_1 is accepted if $\mathcal{B} < 1/T$. Otherwise if $1/T \leq \mathcal{B} \leq T$ insufficient evidence is available and additional simulations need to be generated and evaluated. A threshold value $T = 10^{-2}$ suggests to provide decisive evidence against H_0 and in favour of H_1 (Jeffreys, 1961, Appendix B). Conversely a threshold value $T = 10^2$ suggests to provide decisive evidence in favour of H_0 .

A detailed description of the algorithm and usage examples are given by Jha et al. (Jha et al., 2009b). Moreover an example of applying Bayesian statistical model checking to Simulink/Stateflow is given by Zuliani et al. (Zuliani et al., 2010).

B APPENDIX

Existing model checking approaches for computational models of biological systems

A concise description of the reviewed references underpinning Section 2.5 is provided in Table B.1. None of the approaches consider how biological systems evolve in space and/or across multiple levels of organization.

Table B.1: Description of the existing model checking approaches employed for the validation of computational models of biological systems. The first column records references considered from the literature. The modelling formalism employed to encode the computational model is given in column two; if the computational model was represented in a software-specific modelling language the name of the software is additionally provided in parentheses. The third column specifies the temporal logic used to write the formal specification. The employed model checking algorithm type (AP - Approximate probabilistic, EN - Exhaustive non-probabilistic, EP - Exhaustive probabilistic) is provided in column four. The name of the model checker and the case studies against which it was validated are provided in columns five and six. Finally the model checking application type (MC - Model construction, MV - Model validation, PI - Parameter identification, RC - Robustness computation) is recorded in column seven. Throughout if the value corresponding to a particular column is unknown the NME (Not mentioned explicitly) abbreviation is used instead.

Reference	Model	Logic	Alg	Tool	Case study	App
(Antoniotti et al., 2003)	XS-System	ASySA (based on CTL)	EN	XSSYS	Repressilator, Purine metabolism	MV
(Ballarini et al., 2014)	Generalized stochastic Petri nets	HASL	AP	COSMOS	Wnt/ β -catenin signalling pathway	MV
(Ballarini and Guerriero, 2010)	Continuous Time Markov Chain (PRISM)	CSL, LTL	EP	PRISM	Generic biochemical systems	MV

Reference	Model	Logic	Alg	Tool	Case study	App
(Ballarini et al., 2012)	Generalized stochastic Petri nets	HASL	AP	COSMOS	Single gene expression with stochastic delayed dynamics	MV
(Ballarini et al., 2009)	Continuous Time Markov Chain (PRISM)	CSL	EP	PRISM	3-way oscillator	MV
(Barbuti et al., 2012)	Interval Discrete-Time Markov Chain	PCTL fragment	EP	PRISM	Tumour growth	MV
(Barnat et al., 2009b)	Piecewise linear differential equations	apLTL	EN	GeNeSim (based on DiVinE)	Synthetic genetic regulatory networks	MV
(Barnat et al., 2010b)	Piecewise multi-affine ordinary differential equations	LTL	EN	NME	Ammonium transport from external environment into the cells of <i>Escherichia coli</i>	PI, RC
(Barnat et al., 2009a)	Piecewise multi-affine ordinary differential equations	LTL	EN	BioDiVinE (based on DiVinE)	Ammonium transport in <i>Escherichia coli</i>	MV
(Barnat et al., 2010a)	Ordinary differential equations	LTL	EN	DiVinE	Sporulation of a soil bacteria <i>Bacillus subtilis</i> , <i>Escherichia coli</i> ammonium assimilation	MV
(Bartocci et al., 2010)	Oscillator timed automata	BOSL	EN	BOSL model checker	Pacemaker cells	MV

Reference	Model	Logic	Alg	Tool	Case study	App
(Bartocci et al., 2015)	Continuous Time Markov Chain, Stochastic Hybrid Automata	STL	AP	Breach	Schlögl bistable system, Circadian clock of <i>Ostreococcus tauri</i> , feed-forward motif of a gene regulatory network	PI, RC
(Batt et al., 2008)	Piecewise multi-affine differential equations	LTL	EN	RoVerGeNe (based on NuSMV)	Synthetic gene network in <i>Escherichia coli</i>	PI
(Batt et al., 2007a)	Piecewise multi-affine differential equations	LTL	EN	RoVerGeNe (based on NuSMV)	Synthetic gene network in <i>Escherichia coli</i>	MV, PI
(Batt et al., 2005)	Piecewise linear differential equations	CTL	EN	NuSMV	Gene regulatory network controlling the nutritional stress response in <i>Escherichia coli</i>	MV
(Batt et al., 2007b)	Piecewise multi-affine differential equations	LTL	EN	RoVerGeNe (based on NuSMV)	Synthetic transcriptional cascade built in <i>Escherichia coli</i>	PI
(Bérenguier et al., 2013)	Logic regulatory graph	NME	EN	NuSMV	Regulatory network controlling T-helper cell differentiation	MV
(Bernot et al., 2004)	Regulatory graph	CTL	EN	SMBioNet (based on SMV)	The mucus production in <i>Pseudomonas aeruginosa</i>	MV
(Braz et al., 2013)	Continuous Time Markov Chain (PRISM)	CSL	EP	PRISM	Palytoxin effects on the sodium-potassium pump, a trans-membrane ionic transport system	MV

Reference	Model	Logic	Alg	Tool	Case study	App
(Brim et al., 2013a)	Continuous Time Markov Chain (PRISM)	CSL	EP	Based on PRISM	Generic system of biochemical reactions exhibiting bistability, Mammalian cell cycle gene regulatory control	PI
(Calder et al., 2006)	Continuous Time Markov Chain (PRISM)	CSL	EP	PRISM	RKIP inhibited ERK signalling pathway	MV
(Calzone et al., 2006)	Rule-based (BIOCHAM)	CTL, LTL, PLTL	EN, EP	NuSMV, BIOCHAM, APMC	Cell cycle control	MC
(Cavaliere et al., 2014)	Membrane systems	PLASMA formal logic (based on BLTL)	AP	NME (based on PLASMA-lab)	The Role of Estrogen in Cellular Mitosis and DNA Damage	MV
(Češka et al., 2014)	Continuous Time Markov Chain (PRISM)	CSL	EP	PRISM	SIR epidemic, DNA walkers	PI
(Česka et al., 2014)	Continuous Time Markov Chain (PRISM)	CSL	EP	PRISM	Gene regulation of mammalian cell cycle, Response of two-component signalling systems	RC
(Chabrier and Fages, 2003)	Rule-based (BIOCHAM)	CTL	EN	NuSMV, DMC	Mammalian cell cycle and gene expression regulation	MV
(Chabrier-Rivier et al., 2004)	Rule-based (BIOCHAM)	CTL	EN	NuSMV	Mammalian cell cycle	MV
(Ciocchetta et al., 2009)	Bio-PEPA	CSL	EP	PRISM	General genetic networks with a negative feedback	MV

Reference	Model	Logic	Alg	Tool	Case study	App
(Clarke et al., 2008)	Rule-based (BioNetGen)	PBLTL	AP	BioLab	The dynamics of the T-cell receptor signalling network	MV
(David et al., 2012)	Stochastic hybrid automata, Domain specific	Weighted MITL, MITL	AP	UPPAAL-SMC, PLASMA	A genetic circadian oscillator	MV
(Donaldson and Gilbert, 2008a)	Ordinary differential equations	PLTLC	AP	MC2	MAPK signalling pathway	PI
(Fages and Rizk, 2009)	Ordinary differential equations	QFCTL	EN	BIOCHAM	Budding yeast cell cycle, MAPK signalling pathway, synthetic gene transcriptional cascade system	MC, PI, RC
(Giacobbe et al., 2015)	Weighted regulatory graph	LTL	EN	NME	Gene regulatory networks	PI, RC
(Gilbert et al., 2007)	Petri nets	CTL, CSL, LTLc	EN, EP	Model Checking Kit, PRISM, BIOCHAM	The MAPK signalling pathway	MV
(Gong and Feng, 2014)	Extended Boolean networks	CTL	EN	SMV	The role of Endoplasmic Reticulum-Golgi-regulated signalling pathway on the cell cycle	MV
(Gong et al., 2012)	Rule-based (BioNetGen)	BLTL	AP	NME	The HMGB1 signalling pathway	MV
(Guerriero, 2009)	Bio-PEPA	CSL	EP	PRISM	gp130/JAK/-STAT signalling pathway	MV

Reference	Model	Logic	Alg	Tool	Case study	App
(Heath et al., 2008)	Continuous Time Markov Chain (PRISM)	CSL	EP	PRISM	FGF signalling pathway	MV
(Heiner et al., 2008)	Petri nets	CTL, CSL, (P)LTLc	EN, EP, AP	BIOCHAM, idd-ctl, Model Checking Kit	ERK/MAPK signalling pathway	MV
(Hussain et al., 2014a)	Stochastic discrete-event system	PBLTL	AP	NME	Glucose and insulin metabolism	PI
(Islam et al., 2015)	Ordinary differential equations	NME	EN	NME	Tap withdrawal in <i>Caenorhabditis elegans</i>	PI
(Jha and Langmead, 2011)	Continuous Time Markov Chain	PBLTL, PBMLT	AP	NME	FGF signalling pathway and cell cycle	PI
(Koh et al., 2012)	Stochastic discrete-event system	PLTL	AP	NME	Cell fate determination model of gustatory neurons of <i>Caenorhabditis elegans</i>	MV
(Kwiatkowska et al., 2007)	Continuous/Discrete Time Markov Chain (PRISM)	PCTL, CSL	EP	PRISM	FGF signalling pathway	MV
(Kwiatkowska et al., 2008)	Continuous Time Markov Chain (PRISM)	CSL	EP	PRISM	MAPK signalling pathway	MV
(Lakin et al., 2012)	Continuous Time Markov Chain (Visual DSD)	CTL, CSL	EN, EP	PRISM	DNA strand displacement devices	MV

Reference	Model	Logic	Alg	Tool	Case study	App
(Li et al., 2011)	Hybrid functional Petri nets with extensions	PLTL	AP	MIRACH	Neuronal cell fate decision model in <i>Caenorhabditis elegans</i>	PI
(Li et al., 2009)	Hybrid functional Petri nets with extensions	ATL	EN	MOCHA	Biological pathways involved in <i>Caenorhabditis elegans</i> vulval development	MV
(Liu et al., 2012)	Dynamic Bayesian Networks	PBL	AP	NME	The EGF-NGF pathway, the segmentation clock network, the MLC-phosphorylation pathway	MV
(Liu et al., 2014a)	Hybrid automata	$\mathcal{L}_{\mathbb{R}, \mathcal{F}}$	EN	dReal	Cardiac cell action potential	PI
(Liu et al., 2015)	Nonlinear hybrid automata	NME	EN	dReach	Prostate cancer	PI
(Madsen et al., 2012)	Continuous Time Markov Chain (iBioSim)	CSL	EP	NME	Genetic toggle switch in <i>Escherichia coli</i>	MV
(Mancini et al., 2015)	Ordinary differential equations	LTL	AP	NME	Female menstrual cycle	PI
(Maria et al., 2009)	Rule-based (BIOCHAM)	Constraint-LTL	EN	BIOCHAM	Coupling models of Mammalian cell cycle, the p53-based DNA-damage repair network, and irinotecan metabolism	MV, PI
(Miskov-Zivanov et al., 2013)	Generalized Boolean networks	BLTL	AP	NME	T cell differentiation control network	MV

Reference	Model	Logic	Alg	Tool	Case study	App
(Monteiro et al., 2008)	Piecewise linear differential equations	CTL, μ -calculus	EN	NuSMV, CADP	Regulatory network controlling the carbon starvation response in <i>Escherichia coli</i>	MV
(Monteiro et al., 2014)	Logic regulatory graph	ARCTL	EN	NuSMV	T-helper lymphocyte differentiation	MV
(Palaniappan et al., 2013)	Ordinary differential equations	PBLTL	AP	NME	Segmentation clock pathway, Thrombin-dependent MLC-pathway	PI
(Rizk et al., 2008)	Rule-based (BIOCHAM)	QFLTL(R)	EN	BIOCHAM	The budding yeast cell cycle, MAPK signalling pathway	PI, RC
(Rizk et al., 2009)	Ordinary differential equations	QFLTL	EN	BIOCHAM	Synthetic transcriptional cascade built in <i>Escherichia coli</i>	RC
(Schivo et al., 2012)	Timed automata	TCTL subset	EN	UPPAAL	Crosstalk between EGF and NGF in PC-12 cells	MV
(Siebert and Bockmayr, 2006)	Timed automata	TCTL subset	EN	UPPAAL	Regulatory network of bacteriophage λ	MV
(Van Goethem et al., 2013)	Timed automata	TCTL subset	EN	UPPAAL	G1/S cell cycle in mammalian cells	MV
(Yordanov and Belta, 2011)	Piecewise multi-affine differential equations	LTL	EN	RoVerGeNe (based on NuSMV)	Synthetic gene networks	MV

C APPENDIX

Multidimensional spatio-temporal model checking supplementary materials

C.1 Mapping between subalgorithms of region detection mechanism and OpenCV functions

Table C.1: Mapping between subalgorithms employed by Algorithm 1 and functions from the open source Computer Vision library OpenCV. The left column describes the signature of the subalgorithms employed by Algorithm 1. The right column describes the signature of the corresponding OpenCV function(s).

Subalgorithm signature	OpenCV function signature
ChangeBrightnessAndContrast(image, alpha, beta)	convertTo(image, -1, alpha, beta)
MorphologicalCloseOperation(image, morphCloseNrOfIter)	morphologyEx(image, outputImage, MORPH_CLOSE, Mat(), Point(-1, -1), morphCloseNrOfIter)
GaussianBlur(image, kernelSize, standardDev)	GaussianBlur(image, outputImage, kernelSize, standardDev)
Threshold(image, thresholdValue)	threshold(image, outputImage, thresholdValue, 255, THRESH_BINARY)

<pre> DetectAndApproximateContours(image, approximationLevel) </pre>	<pre> findContours(image, contours, contoursHierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_NONE, Point()) approxPolyDP(image, outputImage, approximationLevel, true) </pre>
--	---

C.2 Numeric measures for encoding formal specifications

Unary and binary numeric measures which can be employed for encoding formal BLSTL specifications are described in the tables below. In each table body row the name of the numeric measure is provided in column 1, the description in column 2 and the semantics definition in column 3.

Table C.2: Unary numeric measures.

Name	Description	Semantics
<i>abs</i>	Returns the absolute value of a number n .	$abs(n) = n $.
<i>ceil</i>	Rounds the number n upward, returning the smallest integral value that is not less than n .	$ceil(n) = \lceil n \rceil$.
<i>floor</i>	Rounds the number n downward, returning the largest integral value that is not greater than n .	$floor(n) = \lfloor n \rfloor$.
<i>round</i>	Returns the integral value that is nearest to number n , with halfway cases rounded away from zero.	$round(n) = \begin{cases} \lfloor n + 0.5 \rfloor, & \text{if } n \geq 0 \\ \lceil n - 0.5 \rceil, & \text{otherwise.} \end{cases}$
<i>sign</i>	Returns the sign of a number n .	$sign(n) = \begin{cases} 1, & \text{if } n > 0 \\ 0, & \text{if } n = 0 \\ -1, & \text{otherwise.} \end{cases}$
<i>sqrt</i>	Returns the square root of a number n .	$sqrt(n) = \sqrt{n}$.
<i>trunc</i>	Rounds the number n toward zero.	$trunc(n) = sign(n) \lfloor n \rfloor$.

Table C.3: Binary numeric measures.

Name	Description	Semantics
<i>add</i>	Returns the sum of two numbers n_1 and n_2 .	$add(n_1, n_2) = n_1 + n_2$.
<i>div</i>	Returns the integer part of the division n_1/n_2 .	$div(n_1, n_2) = \lfloor n_1/n_2 \rfloor$.
<i>log</i>	Returns the logarithm of a number n in the given base b .	$log(n, b) = \log_b(n), n > 0, b > 0, b \neq 1$.
<i>mod</i>	Returns the remainder of the division n_1/n_2 .	$mod(n_1, n_2) = n_1 - (n_2 \cdot div(n_1, n_2))$.
<i>multiply</i>	Returns the multiplication of two numbers n_1 and n_2 .	$multiply(n_1, n_2) = n_1 n_2$.
<i>power</i>	Returns the base b raised at the power e .	$pow(b, e) = b^e$.
<i>subtract</i>	Returns the difference between two numbers n_1 and n_2 .	$subtract(n_1, n_2) = n_1 - n_2$.

C.3 Subset measures for encoding formal specifications

Unary, binary, ternary and quaternary subset measures which can be employed for encoding formal BLSTL specifications are described in the tables below. In each table body row the name of the subset measure is provided in column 1, the description in column 2 and the semantics definition in column 3.

Table C.4: Unary subset measures.

Name	Description	Semantics
<i>count</i>	Returns the number of spatial entities in <i>subset</i> .	$count(subset) = subset $.
<i>clusteredness</i>	Returns the clusteredness of a set of clusters/regions.	As described in Subsection 3.4.2.
<i>density</i>	Returns the density of a set of clusters/regions.	As described in Subsection 3.4.2.

Table C.5: Binary subset measures.

Name	Description	Semantics
------	-------------	-----------

<i>avg</i>	Returns the arithmetic mean considering the given <i>subset</i> and spatial measure <i>sm</i> .	$avg(subset, sm) = \frac{1}{n} \sum_{i=1}^n sm(subset_i),$ where $n = subset .$
<i>geomean</i>	Returns the geometric mean considering the given <i>subset</i> and spatial measure <i>sm</i> .	$geomean(subset, sm) = \left(\prod_{i=1}^n sm(subset_i) \right)^{\frac{1}{n}},$ where $n = subset .$
<i>harmean</i>	Returns the harmonic mean considering the given <i>subset</i> and spatial measure <i>sm</i> .	$harmean(subset, sm) = \frac{n}{\sum_{i=1}^n \frac{1}{sm(subset_i)}},$ where $n = subset .$
<i>kurt</i>	Returns the kurtosis considering the given <i>subset</i> and spatial measure <i>sm</i> .	$kurt(subset, sm) = \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^n \left(\frac{sm(subset_i) - avg(subset, sm)}{stdev(subset, sm)} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)},$ where $n = subset \geq 4.$
<i>max</i>	Returns the maximum considering the given <i>subset</i> and spatial measure <i>sm</i> .	$max(subset, sm) = \max_{i=1,n} sm(subset_i),$ where $n = subset .$
<i>median</i>	Returns the median considering the given <i>subset</i> and spatial measure <i>sm</i> .	$median(subset, sm) =$ middle value in the ordered list of spatial measures values $sm(subset_1), sm(subset_2), \dots, sm(subset_n)$, where $n = subset .$
<i>min</i>	Returns the minimum considering the given <i>subset</i> and spatial measure <i>sm</i> .	$min(subset, sm) = \min_{i=1,n} sm(subset_i),$ where $n = subset .$
<i>mode</i>	Returns the mode considering the given <i>subset</i> and spatial measure <i>sm</i> .	$mode(subset, sm) =$ value which appears most often in the list of spatial measures values $sm(subset_1), sm(subset_2), \dots, sm(subset_n)$, where $n = subset .$
<i>product</i>	Returns the product considering the given <i>subset</i> and spatial measure <i>sm</i> .	$product(subset, sm) = \prod_{i=1}^n sm(subset_i),$ where $n = subset .$
<i>skew</i>	Returns the skewness considering the given <i>subset</i> and spatial measure <i>sm</i> .	$skew(subset, sm) = \frac{n}{(n-1)(n-2)} \sum_{i=1}^n \left(\frac{sm(subset_i) - avg(subset, sm)}{stdev(subset, sm)} \right)^3,$ where $n = subset .$
<i>stdev</i>	Returns the standard deviation considering the given <i>subset</i> and spatial measure <i>sm</i> .	$stdev(subset, sm) = \sqrt{\frac{\sum_{i=1}^n (sm(subset_i) - avg(subset, sm))^2}{(n-1)}},$ where $n = subset .$
<i>sum</i>	Returns the sum considering the given <i>subset</i> and spatial measure <i>sm</i> .	$sum(subset, sm) = \sum_{i=1}^n sm(subset_i),$ where $n = subset .$
<i>var</i>	Returns the variance considering the given <i>subset</i> and spatial measure <i>sm</i> .	$var(subset, sm) = \frac{\sum_{i=1}^n (sm(subset_i) - avg(subset, sm))^2}{(n-1)},$ where $n = subset .$

Table C.6: Ternary subset measures.

Name	Description	Semantics
<i>percentile</i>	Returns the pc -th ($0 \leq pc \leq 100$) percentile considering the given <i>subset</i> and spatial measure <i>sm</i> .	$percentile(subset, sm, pc) = i$ -th spatial measure value in the ordered list of n spatial measures values $sm(subset_1), sm(subset_2), \dots, sm(subset_n)$, where $n = subset $, and $i = \lfloor \frac{pc}{100}n + \frac{1}{2} \rfloor$.
<i>quartile</i>	Returns the i -th ($i \in \{25, 50, 75\}$) quartile considering the given <i>subset</i> and spatial measure <i>sm</i> .	Let v be the ordered list of spatial measures values $sm(subset_1), sm(subset_2), \dots, sm(subset_n)$, where $n = subset $. Moreover let $m = median(subset, sm)$, L be the sublist of values in v smaller than m , and U be the sublist of values in v greater than m . $quartile(subset, sm, i) = median$ of L if $i = 25$, m if $i = 50$, and $median$ of U if $i = 75$.

Table C.7: Quaternary subset measures.

Name	Description	Semantics
<i>covar</i>	Returns the covariance considering <i>subset₁</i> and <i>subset₂</i> , and the spatial measures <i>sm₁</i> and <i>sm₂</i> .	$covar(subset_1, sm_1, subset_2, sm_2) = \frac{1}{n-1} \sum_{i=1}^n ((sm_1(subset_{1i}) - avg(subset_1, sm_1)) (sm_2(subset_{2i}) - avg(subset_2, sm_2))),$ where $n = min(subset_1 , subset_2)$.

C.4 Improved frequentist statistical model checking

The algorithms OSM A/B described by Koh et al. (Koh et al., 2012) initialise variables with invalid values. First of all the notations relevant for describing the initialisation error will be explained. Then a brief proof will be provided illustrating the presence of the error followed by the proposed solution which is considered in our approach. Let us assume for the remainder of this section that the logic property to be verified ϕ is of the form $P_{\bowtie\theta}[\psi], \theta \in (0, 1)$.

C.4.1 Notations

The width of the indifference region (p_1, p_0) is equal to 2δ , where

$$p_1 = \theta - \delta;$$

$$p_0 = \theta + \delta,$$

and $0 \leq \delta \leq 1$. Using δ two values f_n and f'_n are computed to decide if ϕ holds:

$$\begin{aligned} f_n &= d \left(\log \frac{\theta - \delta}{\theta} \right) + (n - d) \left(\log \frac{1 - (\theta - \delta)}{1 - \theta} \right); \\ f'_n &= d \left(\log \frac{\theta}{\theta + \delta} \right) + (n - d) \left(\log \frac{1 - \theta}{1 - (\theta + \delta)} \right), \end{aligned} \quad (\text{C.1})$$

where n represents the total number of evaluated model simulations and d the number of simulations for which ϕ evaluated true.

The nominator and denominator of each fraction in Equation C.1 represent probability values $\in (0, 1)$. This additionally ensures that the values provided to the logarithms are positive and therefore valid. Thus the following inequalities must hold:

$$0 < \theta - \delta < 1; \quad (\text{C.2a})$$

$$0 < \theta < 1; \quad (\text{C.2b})$$

$$0 < 1 - (\theta - \delta) < 1; \quad (\text{C.2c})$$

$$0 < 1 - \theta < 1; \quad (\text{C.2d})$$

$$0 < \theta + \delta < 1; \quad (\text{C.2e})$$

$$0 < 1 - (\theta + \delta) < 1. \quad (\text{C.2f})$$

Considering that $\theta \in (0, 1)$ the δ independent inequalities C.2b and C.2d hold always.

C.4.2 Description of initialisation error

The cause of the error in the OSM A/B algorithms is the initialisation of δ with the value 1. We will prove this by assuming that the initialisation is valid ($\delta = 1$) and employing proof by contradiction.

Proof 4 Initialisation error in OSM A/B algorithms

The first instruction in the OSM A/B algorithms is the initialisation of δ with the value 1. This value is then passed to the IncrementalYounesB function.

After computing the value of four variables (A_1, B_1, A_2, B_2) the main repeat loop is entered. A new sample is generated and evaluated, and the values of the variables n and d are updated. Afterwards the values of f_n and f'_n are computed according to Equation C.1. To prove that the initialisation of δ is invalid it is sufficient to show that one of the C.2x inequalities does not hold. For instance according to inequality C.2a:

$$0 < \theta - \delta < 1.$$

In our case $\delta = 1$. Therefore inequality C.2a is evaluated as follows:

$$\begin{aligned} 0 &< \theta - \delta < 1 \\ \Leftrightarrow & & & (\text{replace } \delta \text{ with } 1) \\ 0 &< \theta - 1 < 1 \\ \Leftrightarrow & & & (+1) \\ 0 + 1 &< \theta - 1 + 1 < 1 + 1 \\ \Leftrightarrow & & & (\text{evaluate arithmetic expressions}) \\ 1 &< \theta < 2 \\ \Rightarrow & & & (\text{conclusion from inequality}) \\ \theta &\in (1, 2). \end{aligned}$$

However $\theta \in (0, 1)$ which contradicts $\theta \in (1, 2)$. Hence the initialisation $\delta = 1$ is invalid. \square

As a side note one of the arithmetic expressions which are invalid when δ is initialised with the value 1 is $\log \frac{\theta-\delta}{\theta}$ because $\theta - \delta < 0$ which means that a negative value is provided to the logarithm.

C.4.3 Solution

In both OSM A/B algorithms the width of the indifference region 2δ is reduced to half whenever an undecided result is obtained. The method employed by the algorithms is to start with the maximum valid δ value and then decrease it until a true/false result can be obtained.

The domain of all valid δ values is computed based on inequalities C.2a-C.2f; see Table C.8 for the interval of valid δ values computed for each inequality.

By intersecting all intervals provided in Table C.8 the domain of valid δ values is obtained:

Table C.8: The valid interval of δ values corresponding to inequalities C.2a-C.2f

Nr.	Inequality	Valid interval of δ values
C.2a	$0 < \theta - \delta < 1$	$\delta \in (\theta - 1, \theta) \cap (0, 1) = (0, \theta)$
C.2b	$0 < \theta < 1$	$\delta \in (0, 1)$
C.2c	$0 < 1 - (\theta - \delta) < 1$	$\delta \in (\theta - 1, \theta) \cap (0, 1) = (0, \theta)$
C.2d	$0 < 1 - \theta < 1$	$\delta \in (0, 1)$
C.2e	$0 < \theta + \delta < 1$	$\delta \in (-\theta, 1 - \theta) \cap (0, 1) = (0, 1 - \theta)$
C.2f	$0 < 1 - (\theta + \delta) < 1$	$\delta \in (-\theta, 1 - \theta) \cap (0, 1) = (0, 1 - \theta)$

$$\begin{aligned}
 D_\delta &= (0, \theta) \cap (0, 1) \cap (0, \theta) \cap (0, 1) \cap (0, 1 - \theta) \cap (0, 1 - \theta) \\
 &\Leftrightarrow \quad \text{(remove duplicates)} \\
 D_\delta &= (0, \theta) \cap (0, 1) \cap (0, 1 - \theta) \\
 &\Leftrightarrow \quad \text{(remove enclosing interval } (0, 1)) \\
 D_\delta &= (0, \theta) \cap (0, 1 - \theta),
 \end{aligned}$$

which is equivalent to

$$D_\delta = (0, \min(\theta, 1 - \theta)).$$

A value δ_{init} smaller than the maximum value defined in D_δ should be employed during the initialisation step of the improved statistical model checking algorithm. Thus

$$\begin{aligned}
 \delta_{init} &< \max_{v \in D_\delta}(v) \\
 &\Leftrightarrow \quad \text{(expand rhs. of the equation)} \\
 \delta_{init} &< \max_{v \in (0, \min(\theta, 1 - \theta))}(v) \\
 &\Leftrightarrow \quad \text{(expand rhs. of the equation)} \\
 \delta_{init} &< \min(\theta, 1 - \theta) \\
 &\Rightarrow \quad (\delta_{init} \text{ must be smaller than } \min(\theta, 1 - \theta)) \\
 \delta_{init} &= \min(\theta, 1 - \theta) - \epsilon,
 \end{aligned}$$

where $0 < \epsilon \ll \min(\theta, 1 - \theta)$. For implementation purposes the value of ϵ can be chosen as follows:

$$0 < \epsilon = \frac{1}{k} \min(\theta, 1 - \theta) < \min(\theta, 1 - \theta),$$

where $k \gg 1$ is a user-defined or hard coded finite constant. Thus

$$\begin{aligned}
 \delta_{init} &= \min(\theta, 1 - \theta) - \frac{1}{k} \min(\theta, 1 - \theta) \\
 &\Leftrightarrow \quad \text{(arithmetic operations)} \\
 \delta_{init} &= \frac{k}{k} \min(\theta, 1 - \theta) - \frac{1}{k} \min(\theta, 1 - \theta) \\
 &\Leftrightarrow \quad \text{(arithmetic operations)} \\
 \boxed{\delta_{init} = \frac{k-1}{k} \min(\theta, 1 - \theta)}.
 \end{aligned}$$

Proving that $\delta_{init} \in D_\delta$ for a finite constant value $k \gg 1$ is trivial. Moreover

$$\begin{aligned}
 &\lim_{k \rightarrow \infty} \delta_{init} \\
 &= \quad \text{(replace } \delta_{init} \text{ with its value)} \\
 &\lim_{k \rightarrow \infty} \left(\frac{k-1}{k} \min(\theta, 1 - \theta) \right) \\
 &= \quad \text{(extract } k \text{ independent terms outside limit)} \\
 &\min(\theta, 1 - \theta) \lim_{k \rightarrow \infty} \left(\frac{k-1}{k} \right) \\
 &= \quad \text{(compute value of limit)} \\
 &\min(\theta, 1 - \theta)(1) \\
 &= \quad \text{(evaluate arithmetic expression)} \\
 &\min(\theta, 1 - \theta).
 \end{aligned}$$

The value of δ_{init} gets closer to $\min(\theta, 1 - \theta)$ as the value of k approaches ∞ . Thus high values of k should be employed in the implementation.

C.5 Proof that the semantics of a BLSTL statement can be defined based on a finite prefix of an infinite execution

Proof 5 **BLSTL semantics based on finite prefix of infinite execution**
 We will prove the results of Lemma 1 recursively on the structure of the logic property ψ as described below:

1. $\sigma \models nspm \asymp nm$ if and only if $\hat{\sigma} \models nspm \asymp nm$.

Proof.

- a) $\sigma \models nspm \asymp nm$ **if and only if** $nspm \asymp nm$.
- b) $\hat{\sigma} \models nspm \asymp nm$ **if and only if** $nspm \asymp nm$.
- c) By Definition 12 $[\psi] = 0$ which means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0$. Hence the symbols $nspm$ and nm are evaluated to the same values for both σ and $\hat{\sigma}$.
- d) From 1a, 1b and 1c it follows that $\sigma \models nspm \asymp nm$ **if and only if** $\hat{\sigma} \models nspm \asymp nm$.

□

2. $\sigma \models nsv \asymp nm$ **if and only if** $\hat{\sigma} \models nsv \asymp nm$ (Proof is similar to the one provided for 1).
3. $\sigma \models d(nm_1) \asymp nm_2$ **if and only if** $\hat{\sigma} \models d(nm_1) \asymp nm_2$.

Proof.

- a) $\sigma \models d(nm_1) \asymp nm_2$ **if and only if** $|\sigma| > 1$ and $d(nm_1) \asymp nm_2$, such that $d(nm_1) = \frac{nm_1^1 - nm_1^0}{time^1 - time^0}$, where nm_i^j represents the result of evaluating nm_i against σ^j , and $time^k$ represents the value of the first time point in σ^k .
- b) $\hat{\sigma} \models d(nm_1) \asymp nm_2$ **if and only if** $|\hat{\sigma}| > 1$ and $d(nm_1) \asymp nm_2$.
- c) By Definition 12 $[\psi] = 1$ which means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0$ and $\hat{s}_1 = s_1$. Hence the symbols nm_1^0, nm_1^1 and nm_2 are evaluated to the same values for both σ and $\hat{\sigma}$.
- d) From 3c it follows that $\frac{nm_1^1 - nm_1^0}{time^1 - time^0} \asymp nm_2$ (considering σ) **if and only if** $\frac{nm_1^1 - nm_1^0}{time^1 - time^0} \asymp nm_2$ (considering $\hat{\sigma}$).
- e) From 3a and 3d it follows that $\sigma \models d(nm_1) \asymp nm_2$ **if and only if** $\frac{nm_1^1 - nm_1^0}{time^1 - time^0} \asymp nm_2$ (considering $\hat{\sigma}$).
- f) From 3b and 3e it follows that $\sigma \models d(nm_1) \asymp nm_2$ **if and only if** $\hat{\sigma} \models d(nm_1) \asymp nm_2$.

□

4. $\sigma \models \sim \psi$ **if and only if** $\hat{\sigma} \models \sim \psi$.

Proof.

- a) $\sigma \models \sim \psi$ **if and only if** $\sigma \not\models \psi$.

- b) $\hat{\sigma} \models \sim \psi$ if and only if $\hat{\sigma} \not\models \psi$.
- c) By Definition 12 $[\sim \psi] = [\psi]$ which means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ . Hence the semantics of ψ considering σ is equivalent to the semantics of ψ considering $\hat{\sigma}$.
- d) From 4c it follows that $\sigma \not\models \psi$ if and only if $\hat{\sigma} \not\models \psi$.
- e) From 4a and 4d it follows that $\sigma \models \sim \psi$ if and only if $\hat{\sigma} \not\models \psi$.
- f) From 4b and 4e it follows that $\sigma \models \sim \psi$ if and only if $\hat{\sigma} \models \sim \psi$.

□

5. $\sigma \models \psi_1 \wedge \psi_2$ if and only if $\hat{\sigma} \models \psi_1 \wedge \psi_2$.

Proof.

- a) $\sigma \models \psi_1 \wedge \psi_2$ if and only if $\sigma \models \psi_1$ and $\sigma \models \psi_2$.
- b) $\hat{\sigma} \models \psi_1 \wedge \psi_2$ if and only if $\hat{\sigma} \models \psi_1$ and $\hat{\sigma} \models \psi_2$.
- c) By Definition 12 $[\psi_1 \wedge \psi_2] = \max([\psi_1], [\psi_2])$ which means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of both ψ_1 and ψ_2 . Hence the semantics of ψ_1 and ψ_2 is the same considering both σ and $\hat{\sigma}$.
- d) From 5c it follows that $\sigma \models \psi_1$ if and only if $\hat{\sigma} \models \psi_1$, and $\sigma \models \psi_2$ if and only if $\hat{\sigma} \models \psi_2$.
- e) From 5d it follows that $\sigma \models \psi_1$ and $\sigma \models \psi_2$ if and only if $\hat{\sigma} \models \psi_1$ and $\hat{\sigma} \models \psi_2$.
- f) From 5a and 5e it follows that $\sigma \models \psi_1 \wedge \psi_2$ if and only if $\hat{\sigma} \models \psi_1$ and $\hat{\sigma} \models \psi_2$.
- g) From 5b and 5f it follows that $\sigma \models \psi_1 \wedge \psi_2$ if and only if $\hat{\sigma} \models \psi_1 \wedge \psi_2$.

□

6. $\sigma \models \psi_1 \vee \psi_2$ if and only if $\hat{\sigma} \models \psi_1 \vee \psi_2$ (Proof is similar to the one provided for 5).
7. $\sigma \models \psi_1 \Rightarrow \psi_2$ if and only if $\hat{\sigma} \models \psi_1 \Rightarrow \psi_2$ (Proof is similar to the one provided for 5).

8. $\sigma \models \psi_1 \Leftrightarrow \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 \Leftrightarrow \psi_2$ (Proof is similar to the one provided for 5).
9. $\sigma \models \psi_1 U[a, b] \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 U[a, b] \psi_2$.

Proof.

- a) $\sigma \models \psi_1 U[a, b] \psi_2$ **if and only if** there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi_2$, and for all $j, j \in [a, i)$, it holds that $\sigma(j) \models \psi_1$.
- b) $\hat{\sigma} \models \psi_1 U[a, b] \psi_2$ **if and only if** there exists $i', i' \in [a, b]$, such that $\hat{\sigma}(i') \models \psi_2$, and for all $j', j' \in [a, i')$, it holds that $\hat{\sigma}(j') \models \psi_1$.
- c) By Definition 12 $[\psi_1 U[a, b] \psi_2] = b + \max([\psi_1], [\psi_2])$. This means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of both ψ_1 and ψ_2 considering any execution suffix $\sigma(h)/\hat{\sigma}(h), h \in [a, b]$.
- d) From 9c it follows that for any suffix execution $\sigma(h)/\hat{\sigma}(h), h \in [a, b]$ the semantics of ψ_1 and ψ_2 is the same.
- e) From 9d it follows that there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi_2$ **if and only if** there exists $i', i' \in [a, b], i' = i$, such that $\hat{\sigma}(i') \models \psi_2$.
- f) From 9d it follows that for all $j, j \in [a, i)$, it holds that $\sigma(j) \models \psi_1$ **if and only if** for all $j', j' \in [a, i'), i' = i, j' = j$, it holds that $\hat{\sigma}(j') \models \psi_1$.
- g) From 9e and 9f it follows that there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi_2$ and for all $j, j \in [a, i)$, it holds that $\sigma(j) \models \psi_1$ **if and only if** there exists $i', i' \in [a, b], i' = i$, such that $\hat{\sigma}(i') \models \psi_2$ and for all $j', j' \in [a, i')$, it holds that $\hat{\sigma}(j') \models \psi_1$.
- h) From 9a and 9g it follows that $\sigma \models \psi_1 U[a, b] \psi_2$ **if and only if** there exists $i', i' \in [a, b]$, such that $\hat{\sigma}(i') \models \psi_2$ and for all $j', j' \in [a, i')$, it holds that $\hat{\sigma}(j') \models \psi_1$.
- i) From 9b and 9h it follows that $\sigma \models \psi_1 U[a, b] \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 U[a, b] \psi_2$.

□

10. $\sigma \models F[a, b] \psi$ **if and only if** $\hat{\sigma} \models F[a, b] \psi$.

Proof.

- a) $\sigma \models F[a, b] \psi$ **if and only if** there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi$.
- b) $\hat{\sigma} \models F[a, b] \psi$ **if and only if** there exists $i', i' \in [a, b]$, such that $\hat{\sigma}(i') \models \psi$.
- c) By Definition 12 $[F[a, b] \psi] = b + [\psi]$. This means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ considering any execution suffix $\sigma(h)/\hat{\sigma}(h)$, $h \in [a, b]$.
- d) From 10c it follows that the semantics of ψ is equivalent for suffix executions $\sigma(h)$ and $\hat{\sigma}(h)$, for all $h, h \in [a, b]$.
- e) From 10d it follows that there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi$ **if and only if** there exists $i', i' \in [a, b], i' = i$, such that $\hat{\sigma}(i') \models \psi$.
- f) From 10a and 10e it follows that $\sigma \models F[a, b] \psi$ **if and only if** there exists $i', i' \in [a, b]$, such that $\hat{\sigma}(i') \models \psi$.
- g) From 10b and 10f it follows that $\sigma \models F[a, b] \psi$ **if and only if** $\hat{\sigma} \models F[a, b] \psi$.

□

11. $\sigma \models G[a, b] \psi$ **if and only if** $\hat{\sigma} \models G[a, b] \psi$.

Proof.

- a) $\sigma \models G[a, b] \psi$ **if and only if** for all $i, i \in [a, b]$, it holds that $\sigma(i) \models \psi$.
- b) $\hat{\sigma} \models G[a, b] \psi$ **if and only if** for all $i', i' \in [a, b]$, it holds that $\hat{\sigma}(i') \models \psi$.
- c) By Definition 12 $[G[a, b] \psi] = b + [\psi]$. This means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ considering any execution suffix $\sigma(h)/\hat{\sigma}(h)$, $h \in [a, b]$.
- d) From 11c it follows that the semantics of ψ is equivalent for suffix executions $\sigma(h)$ and $\hat{\sigma}(h)$, for all $h, h \in [a, b]$.
- e) From 11d it follows that for all $i, i \in [a, b]$, it holds that $\sigma(i) \models \psi$ **if and only if** for all $i', i' \in [a, b], i' = i$, it holds that $\hat{\sigma}(i') \models \psi$.
- f) From 11a and 11e it follows that $\sigma \models G[a, b] \psi$ **if and only if** for all $i', i' \in [a, b]$, it holds that $\hat{\sigma}(i') \models \psi$.

- g) From 11b and 11f it follows that $\sigma \models G[a, b] \psi$ if and only if $\hat{\sigma} \models G[a, b] \psi$.

□

12. $\sigma \models X \psi$ if and only if $\hat{\sigma} \models X \psi$.

Proof.

- a) $\sigma \models X \psi$ if and only if $|\sigma| > 1$ and $\sigma^1 \models \psi$.
- b) $\hat{\sigma} \models X \psi$ if and only if $|\hat{\sigma}| > 1$ and $\hat{\sigma}^1 \models \psi$.
- c) By Definition 12 $[X \psi] = 1 + [\psi]$. This means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0$, $\hat{s}_1 = s_1$, ..., $\hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ considering the execution suffix $\sigma^1/\hat{\sigma}^1$.
- d) From 12c it follows that the semantics of ψ is equivalent for suffix executions σ^1 and $\hat{\sigma}^1$.
- e) From 12d it follows that $\sigma^1 \models \psi$ if and only if $\hat{\sigma}^1 \models \psi$.
- f) From 12a and 12e it follows that $\sigma \models X \psi$ if and only if $\hat{\sigma} \models X \psi$.
- g) From 12b and 12f it follows that $\sigma \models X \psi$ if and only if $\hat{\sigma} \models X \psi$.

□

13. $\sigma \models X[k] \psi$ if and only if $\hat{\sigma} \models X[k] \psi$.

Proof.

- a) $\sigma \models X[k] \psi$ if and only if $|\sigma| > k$ and $\sigma^k \models \psi$.
- b) $\hat{\sigma} \models X[k] \psi$ if and only if $|\hat{\sigma}| > k$ and $\hat{\sigma}^k \models \psi$.
- c) By Definition 12 $[X[k] \psi] = k + [\psi]$. This means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0$, $\hat{s}_1 = s_1$, ..., $\hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ considering the execution suffix $\sigma^k/\hat{\sigma}^k$.
- d) From 13c it follows that the semantics of ψ is equivalent for suffix executions σ^k and $\hat{\sigma}^k$.
- e) From 13d it follows that $\sigma^k \models \psi$ if and only if $\hat{\sigma}^k \models \psi$.
- f) From 13a and 13e it follows that $\sigma \models X[k] \psi$ if and only if $\hat{\sigma} \models X[k] \psi$.

g) From 13b and 13f it follows that $\sigma \models X[k] \psi$ **if and only if** $\hat{\sigma} \models X[k] \psi$.

□

14. $\sigma \models (\psi)$ **if and only if** $\hat{\sigma} \models (\psi)$.

Proof.

- a) $\sigma \models (\psi)$ **if and only if** $\sigma \models \psi$.
- b) $\hat{\sigma} \models (\psi)$ **if and only if** $\hat{\sigma} \models \psi$.
- c) By Definition 12 $\lceil(\psi)\rceil = \lceil\psi\rceil$. This means that according to the assumptions of Lemma 1 $\hat{s}_0 = s_0$, $\hat{s}_1 = s_1$, ..., $\hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ .
- d) From 14c it follows that the semantics of ψ is equivalent for both σ and $\hat{\sigma}$.
- e) From 14d it follows that $\sigma \models \psi$ **if and only if** $\hat{\sigma} \models \psi$.
- f) From 14a and 14e it follows that $\sigma \models (\psi)$ **if and only if** $\hat{\sigma} \models \psi$.
- g) From 14b and 14f it follows that $\sigma \models (\psi)$ **if and only if** $\hat{\sigma} \models (\psi)$.

□

□

D APPENDIX

Multiscale multidimensional spatio-temporal meta model checking supplementary materials

D.1 Proof that the multiscale multidimensional spatio-temporal model checking problem is well-defined

To show that the model checking problem is well-defined we will first prove that the number of required model simulations and state transitions within each simulation are finite.

D.1.1 Finite number of required simulations

In Subsubsection 3.6.1.1 it was shown that, considering the approximate probabilistic model checking approaches described in Table 2.1, a finite number of model simulations are sufficient to determine if a logic statement holds. This property is inherent to the model checking approaches considered and does not directly depend on the model representation and/or formal logic employed to write the system specification. Therefore it will not be restated here.

D.1.2 Finite number of state transitions

Bounded temporal logic (including BLMSTL) properties can be evaluated against model simulations which cover only a finite interval of time. The upper bound of this interval can be computed based on the temporal operators/functions contained by the evaluated logic properties. Let us denote the upper bound corresponding to a generic BLMSTL logic property ψ by $\lceil \psi \rceil$.

Definition 18 Model simulation time upper bound for BLMSTL logic statement

The upper bound $\lceil \psi \rceil \in \mathbb{R}_+$ corresponding to a BLMSTL logic statement ψ considering an execution σ and the abbreviations introduced in Table 5.2 is defined recursively on the structure of the logic statement as follows:

- $\lceil tnm_1 \asymp tnm_2 \rceil = \max(\lceil tnm_1 \rceil, \lceil tnm_2 \rceil)$;
- $\lceil cm(tnm_1) \asymp tnm_2 \rceil = \max(1 + \lceil tnm_1 \rceil, \lceil tnm_2 \rceil) \leq 1 + \max(\lceil tnm_1 \rceil, \lceil tnm_2 \rceil)$ because the value of tnm_1 is computed considering both $\sigma[0]$ and $\sigma[1]$;
- $\lceil \sim \psi \rceil = \lceil \psi \rceil$;
- $\lceil \psi_1 \wedge \psi_2 \rceil = \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil \psi_1 \vee \psi_2 \rceil = \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil \psi_1 \Rightarrow \psi_2 \rceil = \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil \psi_1 \Leftrightarrow \psi_2 \rceil = \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil \psi_1 U[a, b] \psi_2 \rceil = \max(b - 1 + \lceil \psi_1 \rceil, b + \lceil \psi_2 \rceil) \leq b + \max(\lceil \psi_1 \rceil, \lceil \psi_2 \rceil)$;
- $\lceil F[a, b] \psi \rceil = b + \lceil \psi \rceil$;
- $\lceil G[a, b] \psi \rceil = b + \lceil \psi \rceil$;
- $\lceil X \psi \rceil = 1 + \lceil \psi \rceil$;
- $\lceil X[k] \psi \rceil = k + \lceil \psi \rceil$;
- $\lceil (\psi) \rceil = \lceil \psi \rceil$.
- The upper bound $\lceil tnm \rceil$ corresponding to the temporal numeric measure tnm is defined recursively on the structure of the temporal numeric measure as follows:
 - $\lceil re \rceil = 0$, because the value of re is employed directly;

- $\lceil nsv \rceil = 0$, because the value of nsv is computed considering only $\sigma[0]$;
 - $\lceil nstm \rceil$ which is computed as described below;
 - $\lceil unm(tnm) \rceil = \lceil tnm \rceil$;
 - $\lceil bnm(tnm_1, tnm_2) \rceil = \max(\lceil tnm_1 \rceil, \lceil tnm_2 \rceil)$.
- The upper bound $\lceil nstm \rceil$ corresponding to the numeric statistical measure $nstm$ is defined recursively on the structure of the numeric statistical measure as follows:
 - $\lceil ustn(nmc) \rceil = \lceil nmc \rceil$;
 - $\lceil bstn(nmc_1, nmc_2) \rceil = \max(\lceil nmc_1 \rceil, \lceil nmc_2 \rceil)$;
 - $\lceil bstqm(nmc, re) \rceil = \lceil nmc \rceil$.
 - The upper bound $\lceil nmc \rceil$ corresponding to the numeric measure collection nmc is defined recursively on the structure of the numeric measure collection as follows:
 - $\lceil [a, b]nm \rceil = b + \lceil nm \rceil$;
 - $\lceil smc \rceil = 0$, because the value of smc is computed considering only $\sigma[0]$.
 - The upper bound $\lceil nm \rceil$ corresponding to the numeric measure nm is defined recursively on the structure of the numeric measure as follows:
 - $\lceil pnm \rceil = 0$, because the value of pnm is computed considering only $\sigma[0]$;
 - $\lceil unm(nm) \rceil = \lceil nm \rceil$;
 - $\lceil bnm(nm_1, nm_2) \rceil = \max(\lceil nm_1 \rceil, \lceil nm_2 \rceil)$.

Thus the minimum upper bound for the simulation time interval to be covered by model executions when verifying a BLMSTL logic property ψ is $\lceil \psi \rceil$.

Lemma 3 BLMSTL semantics based on finite prefix of infinite execution

Let us assume that a BLMSTL logic property ψ is verified against an infinite execution $\sigma = \{(s_0, t_0), (s_1, t_1), (s_2, t_2), \dots\}$. Moreover let us denote a finite prefix of σ by $\hat{\sigma} = \{(\hat{s}_0, \hat{t}_0), (\hat{s}_1, \hat{t}_1), \dots, (\hat{s}_m, \hat{t}_m)\}$, where

$$\hat{s}_i = s_i \text{ and } \hat{t}_i = t_i, \forall i = \overline{0, m} \text{ with } \sum_{i=0}^m t_i \geq \lceil \psi \rceil \text{ and } \sum_{i=0}^{m-1} t_i < \lceil \psi \rceil.$$

Then $\sigma \models \psi$ if and only if $\hat{\sigma} \models \psi$.

Proof 6 BLMSTL semantics based on finite prefix of infinite execution

We will prove the results of Lemma 3 recursively on the structure of the logic property ψ as described below:

1. $\sigma \models tnm_1 \asymp tnm_2$ if and only if $\hat{\sigma} \models tnm_1 \asymp tnm_2$.

Proof.

- a) $\sigma \models tnm_1 \asymp tnm_2$ if and only if $tnm_1 \asymp tnm_2$.
- b) $\hat{\sigma} \models tnm_1 \asymp tnm_2$ if and only if $tnm_1 \asymp tnm_2$.
- c) By Definition 18 $[\psi] = \max(tnm_1, tnm_2)$ which means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of both tnm_1 and tnm_2 . Hence both tnm_1 and tnm_2 are evaluated to the same values for both σ and $\hat{\sigma}$.
- d) From 1c it follows that $tnm_1 \asymp tnm_2$ (considering σ) if and only if $tnm_1 \asymp tnm_2$ (considering $\hat{\sigma}$).
- e) From 1a and 1d it follows that $\sigma \models tnm_1 \asymp tnm_2$ if and only if $tnm_1 \asymp tnm_2$ (considering $\hat{\sigma}$).
- f) From 1b and 1e it follows that $\sigma \models tnm_1 \asymp tnm_2$ if and only if $\hat{\sigma} \models tnm_1 \asymp tnm_2$.

□

2. $\sigma \models cm(tnm_1) \asymp tnm_2$ if and only if $\hat{\sigma} \models cm(tnm_1) \asymp tnm_2$.

Proof.

- a) $\sigma \models cm(tnm_1) \asymp tnm_2$ if and only if $|\sigma| > 1 + \lceil tnm_1 \rceil$ and $cm(tnm_1) \asymp tnm_2$, such that $cm \in \{d, r\}$, $d(tnm_1) = \frac{tnm_1^1 - tnm_1^0}{time^1 - time^0}$ and $r(tnm_1) = \frac{tnm_1^1}{tnm_1^0}$, where tnm_i^j represents the result of evaluating tnm_i against σ^j , and $time^k$ represents the value of the first time point in σ^k .
- b) $\hat{\sigma} \models cm(tnm_1) \asymp tnm_2$ if and only if $|\hat{\sigma}| > 1 + \lceil tnm_1 \rceil$ and $cm(tnm_1) \asymp tnm_2$.

- c) By Definition 18 $\lceil \psi \rceil = 1 + \max(\lceil tnm_1 \rceil, \lceil tnm_2 \rceil)$ which means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of tnm_1^0, tnm_1^1 and tnm_2 . Hence the symbols tnm_1^0, tnm_1^1 and tnm_2 are evaluated to the same values for both σ and $\hat{\sigma}$.
- d) From 2c it follows that $\frac{tnm_1^1 - tnm_1^0}{time^1 - time^0} \asymp tnm_2$, respectively $\frac{\frac{tnm_1^1}{tnm_1^0}}{\frac{time^1 - time^0}{tnm_1^0}} \asymp tnm_2$ (considering σ) **if and only if** $\frac{tnm_1^1 - tnm_1^0}{time^1 - time^0} \asymp tnm_2$, respectively $\frac{\frac{tnm_1^1}{tnm_1^0}}{\frac{time^1 - time^0}{tnm_1^0}} \asymp tnm_2$ (considering $\hat{\sigma}$).
- e) From 2a and 2d it follows that $\sigma \models cm(tnm_1) \asymp tnm_2$ **if and only if** $\frac{tnm_1^1 - tnm_1^0}{time^1 - time^0} \asymp tnm_2$, respectively $\frac{\frac{tnm_1^1}{tnm_1^0}}{\frac{time^1 - time^0}{tnm_1^0}} \asymp tnm_2$ (considering $\hat{\sigma}$).
- f) From 2b and 2e it follows that $\sigma \models cm(tnm_1) \asymp tnm_2$ **if and only if** $\hat{\sigma} \models cm(tnm_1) \asymp tnm_2$.

□

3. $\sigma \models \sim \psi$ **if and only if** $\hat{\sigma} \models \sim \psi$.

Proof.

- a) $\sigma \models \sim \psi$ **if and only if** $\sigma \not\models \psi$.
- b) $\hat{\sigma} \models \sim \psi$ **if and only if** $\hat{\sigma} \not\models \psi$.
- c) By Definition 18 $\lceil \sim \psi \rceil = \lceil \psi \rceil$ which means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ . Hence the semantics of ψ is the same considering both σ and $\hat{\sigma}$.
- d) From 3c it follows that $\sigma \not\models \psi$ **if and only if** $\hat{\sigma} \not\models \psi$.
- e) From 3a and 3d it follows that $\sigma \models \sim \psi$ **if and only if** $\hat{\sigma} \not\models \psi$.
- f) From 3b and 3e it follows that $\sigma \models \sim \psi$ **if and only if** $\hat{\sigma} \models \sim \psi$.

□

4. $\sigma \models \psi_1 \wedge \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 \wedge \psi_2$.

Proof.

- a) $\sigma \models \psi_1 \wedge \psi_2$ **if and only if** $\sigma \models \psi_1$ and $\sigma \models \psi_2$.
- b) $\hat{\sigma} \models \psi_1 \wedge \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1$ and $\hat{\sigma} \models \psi_2$.
- c) By Definition 18 $[\psi_1 \wedge \psi_2] = \max([\psi_1], [\psi_2])$ which means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of both ψ_1 and ψ_2 . Hence the semantics of ψ_1 and ψ_2 is the same considering both σ and $\hat{\sigma}$.
- d) From 4c it follows that $\sigma \models \psi_1$ **if and only if** $\hat{\sigma} \models \psi_1$, and $\sigma \models \psi_2$ **if and only if** $\hat{\sigma} \models \psi_2$.
- e) From 4d it follows that $\sigma \models \psi_1$ and $\sigma \models \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1$ and $\hat{\sigma} \models \psi_2$.
- f) From 4a and 4e it follows that $\sigma \models \psi_1 \wedge \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1$ and $\hat{\sigma} \models \psi_2$.
- g) From 4b and 4f it follows that $\sigma \models \psi_1 \wedge \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 \wedge \psi_2$.

□

5. $\sigma \models \psi_1 \vee \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 \vee \psi_2$ (Proof is similar to the one provided for 4).
6. $\sigma \models \psi_1 \Rightarrow \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 \Rightarrow \psi_2$ (Proof is similar to the one provided for 4).
7. $\sigma \models \psi_1 \Leftrightarrow \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 \Leftrightarrow \psi_2$ (Proof is similar to the one provided for 4).
8. $\sigma \models \psi_1 U[a, b] \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 U[a, b] \psi_2$.

Proof.

- a) $\sigma \models \psi_1 U[a, b] \psi_2$ **if and only if** there exists $i, i \in [a, b]$, such that $\sigma(i) \models \psi_2$, and for all $j, j \in [a, i)$, it holds that $\sigma(j) \models \psi_1$.
- b) $\hat{\sigma} \models \psi_1 U[a, b] \psi_2$ **if and only if** there exists $i', i' \in [a, b]$, such that $\hat{\sigma}(i') \models \psi_2$, and for all $j', j' \in [a, i')$, it holds that $\hat{\sigma}(j') \models \psi_1$.
- c) By Definition 18 $[\psi_1 U[a, b] \psi_2] = b + \max([\psi_1], [\psi_2])$. This means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of both ψ_1 and ψ_2 considering any execution suffix $\sigma(h)/\hat{\sigma}(h)$, $h \in [a, b]$.

- d) From 8c it follows that for any suffix execution $\sigma(h)/\hat{\sigma}(h)$, $h \in [a, b]$ the semantics of ψ_1 and ψ_2 is the same.
- e) From 8d it follows that there exists i , $i \in [a, b]$, such that $\sigma(i) \models \psi_2$ **if and only if** there exists i' , $i' \in [a, b]$, $i' = i$, such that $\hat{\sigma}(i') \models \psi_2$.
- f) From 8d and 8e it follows that for all j , $j \in [a, i]$, it holds that $\sigma(j) \models \psi_1$ **if and only if** for all j' , $j' \in [a, i']$, $i' = i$, $j' = j$, it holds that $\hat{\sigma}(j') \models \psi_1$.
- g) From 8e and 8f it follows that there exists i , $i \in [a, b]$, such that $\sigma(i) \models \psi_2$ and for all j , $j \in [a, i]$, it holds that $\sigma(j) \models \psi_1$ **if and only if** there exists i' , $i' \in [a, b]$, $i' = i$, such that $\hat{\sigma}(i') \models \psi_2$ and for all j' , $j' \in [a, i']$, it holds that $\hat{\sigma}(j') \models \psi_1$.
- h) From 8a and 8g it follows that $\sigma \models \psi_1 U[a, b] \psi_2$ **if and only if** there exists i' , $i' \in [a, b]$, such that $\hat{\sigma}(i') \models \psi_2$ and for all j' , $j' \in [a, i']$, it holds that $\hat{\sigma}(j') \models \psi_1$.
- i) From 8b and 8h it follows that $\sigma \models \psi_1 U[a, b] \psi_2$ **if and only if** $\hat{\sigma} \models \psi_1 U[a, b] \psi_2$.

□

9. $\sigma \models F[a, b] \psi$ **if and only if** $\hat{\sigma} \models F[a, b] \psi$.

Proof.

- a) $\sigma \models F[a, b] \psi$ **if and only if** there exists i , $i \in [a, b]$, such that $\sigma(i) \models \psi$.
- b) $\hat{\sigma} \models F[a, b] \psi$ **if and only if** there exists i' , $i' \in [a, b]$, such that $\hat{\sigma}(i') \models \psi$.
- c) By Definition 18 $[F[a, b] \psi] = b + [\psi]$. This means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0$, $\hat{s}_1 = s_1$, ..., $\hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ considering any execution suffix $\sigma(h)/\hat{\sigma}(h)$, $h \in [a, b]$.
- d) From 9c it follows that the semantics of ψ is equivalent for suffix executions $\sigma(h)$ and $\hat{\sigma}(h)$, for all h , $h \in [a, b]$.
- e) From 9d it follows that there exists i , $i \in [a, b]$, such that $\sigma(i) \models \psi$ **if and only if** there exists i' , $i' \in [a, b]$, $i' = i$, such that $\hat{\sigma}(i') \models \psi$.

- f) From 9a and 9e it follows that $\sigma \models F[a, b] \psi$ if and only if there exists $i', i' \in [a, b]$, such that $\hat{\sigma}(i') \models \psi$.
- g) From 9b and 9f it follows that $\sigma \models F[a, b] \psi$ if and only if $\hat{\sigma} \models F[a, b] \psi$.

□

10. $\sigma \models G[a, b] \psi$ if and only if $\hat{\sigma} \models G[a, b] \psi$.

Proof.

- a) $\sigma \models G[a, b] \psi$ if and only if for all $i, i \in [a, b]$, it holds that $\sigma(i) \models \psi$.
- b) $\hat{\sigma} \models G[a, b] \psi$ if and only if for all $i', i' \in [a, b]$, it holds that $\hat{\sigma}(i') \models \psi$.
- c) By Definition 18 $[G[a, b] \psi] = b + [\psi]$. This means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ considering any execution suffix $\sigma(h)/\hat{\sigma}(h)$, $h \in [a, b]$.
- d) From 10c it follows that the semantics of ψ is equivalent for suffix executions $\sigma(h)$ and $\hat{\sigma}(h)$, for all $h, h \in [a, b]$.
- e) From 10d it follows that for all $i, i \in [a, b]$, it holds that $\sigma(i) \models \psi$ if and only if for all $i', i' \in [a, b], i' = i$, it holds that $\hat{\sigma}(i') \models \psi$.
- f) From 10a and 10e it follows that $\sigma \models G[a, b] \psi$ if and only if for all $i', i' \in [a, b]$, it holds that $\hat{\sigma}(i') \models \psi$.
- g) From 10b and 10f it follows that $\sigma \models G[a, b] \psi$ if and only if $\hat{\sigma} \models G[a, b] \psi$.

□

11. $\sigma \models X \psi$ if and only if $\hat{\sigma} \models X \psi$.

Proof.

- a) $\sigma \models X \psi$ if and only if $|\sigma| > 1$ and $\sigma^1 \models \psi$.
- b) $\hat{\sigma} \models X \psi$ if and only if $|\hat{\sigma}| > 1$ and $\hat{\sigma}^1 \models \psi$.
- c) By Definition 18 $[X \psi] = 1 + [\psi]$. This means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0, \hat{s}_1 = s_1, \dots, \hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ considering the execution suffix $\sigma^1/\hat{\sigma}^1$.

- d) From 11c it follows that the semantics of ψ is equivalent for suffix executions σ^1 and $\hat{\sigma}^1$.
- e) From 11d it follows that $\sigma^1 \models \psi$ if and only if $\hat{\sigma}^1 \models \psi$.
- f) From 11a and 11e it follows that $\sigma \models X \psi$ if and only if $\hat{\sigma}^1 \models \psi$.
- g) From 11b and 11f it follows that $\sigma \models X \psi$ if and only if $\hat{\sigma} \models X \psi$.

□

12. $\sigma \models X[k] \psi$ if and only if $\hat{\sigma} \models X[k] \psi$.

Proof.

- a) $\sigma \models X[k] \psi$ if and only if $|\sigma| > k$ and $\sigma^k \models \psi$.
- b) $\hat{\sigma} \models X[k] \psi$ if and only if $|\hat{\sigma}| > k$ and $\hat{\sigma}^k \models \psi$.
- c) By Definition 18 $[X[k] \psi] = k + [\psi]$. This means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0$, $\hat{s}_1 = s_1$, ..., $\hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ considering the execution suffix $\sigma^k/\hat{\sigma}^k$.
- d) From 12c it follows that the semantics of ψ is equivalent for suffix executions σ^k and $\hat{\sigma}^k$.
- e) From 12d it follows that $\sigma^k \models \psi$ if and only if $\hat{\sigma}^k \models \psi$.
- f) From 12a and 12e it follows that $\sigma \models X[k] \psi$ if and only if $\hat{\sigma}^k \models \psi$.
- g) From 12b and 12f it follows that $\sigma \models X[k] \psi$ if and only if $\hat{\sigma} \models X[k] \psi$.

□

13. $\sigma \models (\psi)$ if and only if $\hat{\sigma} \models (\psi)$.

Proof.

- a) $\sigma \models (\psi)$ if and only if $\sigma \models \psi$.
- b) $\hat{\sigma} \models (\psi)$ if and only if $\hat{\sigma} \models \psi$.
- c) By Definition 18 $[(\psi)] = [\psi]$. This means that according to the assumptions of Lemma 3 $\hat{s}_0 = s_0$, $\hat{s}_1 = s_1$, ..., $\hat{s}_m = s_m$, where the value of m is determined such that sufficient time points are recorded for the evaluation of ψ .
- d) From 13c it follows that the semantics of ψ is equivalent for both σ and $\hat{\sigma}$.

- e) From 13d it follows that $\sigma \models \psi$ if and only if $\hat{\sigma} \models \psi$.
- f) From 13a and 13e it follows that $\sigma \models (\psi)$ if and only if $\hat{\sigma} \models \psi$.
- g) From 13b and 13f it follows that $\sigma \models (\psi)$ if and only if $\hat{\sigma} \models (\psi)$.

□

□

Lemma 4 Finite number of state transitions to evaluate BLSTL logic statement

The number of state transitions required to verify a BLMSTL logic property is finite.

Proof 7 Finite number of state transitions to evaluate BLSTL logic statement

From Lemma 3 it follows that a BLMSTL logic property ψ can be verified against a model simulation σ based on a finite prefix $\hat{\sigma}$. The minimum time interval captured by $\hat{\sigma}$ is bounded and can be computed using Definition 18. Since we assume the time divergence property holds for all the systems considered only a finite number of state transitions can occur in a bounded interval of time. □

D.1.3 Well-defined model checking problem

Theorem 2 Well-defined multiscale multidimensional spatio-temporal model checking problem

The multiscale multidimensional spatio-temporal model checking problem is well-defined.

Proof 8 Well-defined multiscale multidimensional spatio-temporal model checking problem

In Appendix Subsection D.1.1 it was shown that the number of model simulations required to verify if a PBLMSTL logic property ϕ holds is finite. Moreover according to Lemmas 3 and 4 only a finite prefix and a finite number of state transitions have to be considered for each model simulation. Thus the evaluation of ϕ is reduced to the problem of evaluating atomic properties over a finite number of states for each model simulation, which is decidable. Hence the model checking problem is well-defined. □

References

- Adra, S., Sun, T., MacNeil, S., Holcombe, M., and Smallwood, R. (2010). Development of a Three Dimensional Multiscale Computational Model of the Human Epidermis. *PLoS ONE*, 5(1):e8511.
- Aiello, M., Pratt-Hartmann, I., and Benthem, J. F. A. K. v. (2007). *Handbook of spatial logics*. Springer.
- Allen, J. F. and Hayes, P. J. (1989). Moments and points in an interval-based temporal logic. *Computational Intelligence*, 5(3):225–238.
- Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P. H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. (1995). The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34.
- Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235.
- An, G. (2008). Introduction of an agent-based multi-scale modular architecture for dynamic knowledge representation of acute inflammation. *Theoretical Biology and Medical Modelling*, 5(1):11.
- An, G., Mi, Q., Dutta-Moscato, J., and Vodovotz, Y. (2009). Agent-based models in translational systems biology. *Wiley Interdisciplinary Reviews. Systems Biology and Medicine*, 1(2):159–171.
- Andersen, M. E. and Krewski, D. (2009). Toxicity Testing in the 21st Century: Bringing the Vision to Life. *Toxicological Sciences*, 107(2):324–330.
- Andrianantoandro, E., Basu, S., Karig, D. K., and Weiss, R. (2006). Synthetic biology: new engineering rules for an emerging discipline. *Molecular Systems Biology*, 2(1).
- Antoniotti, M., Policriti, A., Ugel, N., and Mishra, B. (2003). Model building and model checking for biochemical processes. *Cell Biochemistry and Biophysics*, 38(3):271–286.
- Arbelaitz, O., Gurrutxaga, I., Muguerza, J., Pérez, J. M., and Perona, I. (2013). An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256.
- Arditi, R., Tyutyunov, Y., Morgulis, A., Govorukhin, V., and Senina, I. (2001). Directed Movement of Predators and the Emergence of Density-Dependence in Predator–Prey Models. *Theoretical Population Biology*, 59(3):207–221.
- Asai, Y., Abe, T., Oka, H., Okita, M., Hagiwara, K.-i., Ghosh, S., Matsuoka, Y., Kurachi, Y., Nomura, T., and Kitano, H. (2014). A Versatile Platform for Multilevel Modeling of Physiological Systems: SBML-PHML Hybrid Modeling and Simulation. *Advanced Biomedical Engineering*, 3:50–58.
- Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill,

- D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M., and Sherlock, G. (2000). Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29.
- Aziz, A., Sanwal, K., Singhal, V., and Brayton, R. (2000). Model-checking Continuous-time Markov Chains. *ACM Trans. Comput. Logic*, 1(1):162–170.
- Aziz, A., Sanwal, K., Singhal, V., and Brayton, R. K. (1996). Verifying Continuous Time Markov Chains. In *Proceedings of the 8th International Conference on Computer Aided Verification*, CAV ’96, pages 269–276, London, UK. Springer-Verlag.
- Aziz, A., Singhal, V., Balarin, F., Brayton, R. K., and Sangiovanni-Vincentelli, A. L. (1995). It usually works: The temporal logic of stochastic systems. In Wolper, P., editor, *Computer Aided Verification*, number 939 in Lecture Notes in Computer Science, pages 155–165. Springer Berlin Heidelberg.
- Baeten, J. C. M., Basten, T., and Reniers, M. A. (2010). *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press.
- Baier, C. (1998). *On Algorithmic Verification Methods for Probabilistic Systems*. Habilitation, Mannheim, Germany.
- Baier, C. and Grosser, M. (2005). Recognizing ω -regular languages with probabilistic automata. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, pages 137–146.
- Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. The MIT Press.
- Baier, C., Katoen, J.-P., and Hermanns, H. (1999). Approximative Symbolic Model Checking of Continuous-Time Markov Chains. In Baeten, J. C. M. and Mauw, S., editors, *CONCUR’99 Concurrency Theory*, number 1664 in Lecture Notes in Computer Science, pages 146–161. Springer Berlin Heidelberg.
- Ballarini, P., Djafri, H., Duflot, M., Haddad, S., and Pekergin, N. (2011). COSMOS: A Statistical Model Checker for the Hybrid Automata Stochastic Logic. In *2011 Eighth International Conference on Quantitative Evaluation of Systems (QEST)*, pages 143–144.
- Ballarini, P., Gallet, E., Gall, P. L., and Manceny, M. (2014). Formal Analysis of the Wnt/ β -catenin through Statistical Model Checking. In Margaria, T. and Steffen, B., editors, *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, number 8803 in Lecture Notes in Computer Science, pages 193–207. Springer Berlin Heidelberg.
- Ballarini, P. and Guerriero, M. L. (2010). Query-based verification of qualitative trends and oscillations in biochemical systems. *Theoretical Computer Science*, 411(20):2019–2036.
- Ballarini, P., Mäkelä, J., and Ribeiro, A. S. (2012). Expressive Statistical Model Checking of Genetic Networks with Delayed Stochastic Dynamics. In Gilbert, D. and Heiner, M., editors, *Computational Methods in Systems Biology*, Lecture Notes in Computer Science, pages 29–48. Springer Berlin Heidelberg.
- Ballarini, P., Mardare, R., and Mura, I. (2009). Analysing Biochemical Oscillation through Probabilistic Model Checking. *Electronic Notes in Theoretical Computer Science*, 229(1):3–19.
- Balter, A., Merks, R. M. H., Popławski, N. J., Swat, M., and Glazier, J. A. (2007). The Glazier-Graner-Hogeweg Model: Extensions, Future Directions, and Opportunities for Further Study. In Anderson, D. A. R. A., Chaplain, P.

- M. A. J., and Rejniak, D. K. A., editors, *Single-Cell-Based Models in Biology and Medicine*, Mathematics and Biosciences in Interaction, pages 151–167. Birkhäuser Basel.
- Barbuti, R., Levi, F., Milazzo, P., and Scatena, G. (2012). Probabilistic model checking of biological systems with uncertain kinetic rates. *Theoretical Computer Science*, 419:2–16.
- Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Pardini, G., and Tesei, L. (2011). Spatial P systems. *Natural Computing*, 10(1):3–16.
- Barnat, J., Brim, L., Černá, I., Dražan, S., Fabriková, J., Lánfk, J., Šafránek, D., and Ma, H. (2009a). BioDiVinE: A Framework for Parallel Analysis of Biological Models. *Electronic Proceedings in Theoretical Computer Science*, 6:31–45.
- Barnat, J., Brim, L., Černá, I., Dražan, S., Fabriková, J., and Šafránek, D. (2009b). On algorithmic analysis of transcriptional regulation by LTL model checking. *Theoretical Computer Science*, 410(33–34):3128–3148.
- Barnat, J., Brim, L., Havel, V., Havlíček, J., Kriho, J., Lenčo, M., Ročkai, P., Štill, V., and Weiser, J. (2013). DiVinE 3.0 – An Explicit-State Model Checker for Multithreaded C & C++ Programs. In Sharygina, N. and Veith, H., editors, *Computer Aided Verification*, number 8044 in Lecture Notes in Computer Science, pages 863–868. Springer Berlin Heidelberg.
- Barnat, J., Brim, L., and Šafránek, D. (2010a). High-performance analysis of biological systems dynamics with the DiVinE model checker. *Briefings in Bioinformatics*, 11(3):301–312.
- Barnat, J., Brim, L., Šafránek, D., and Vejnár, M. (2010b). Parameter Scanning by Parallel Model Checking with Applications in Systems Biology. In *Second International Workshop on Parallel and Distributed Methods in Verification, 2010 Ninth International Workshop on, and High Performance Computational Systems Biology*, pages 95–104.
- Bartocci, E., Bortolussi, L., Nenzi, L., and Sanguinetti, G. (2013). On the Robustness of Temporal Properties for Stochastic Models. *Electronic Proceedings in Theoretical Computer Science*, 125:3–19.
- Bartocci, E., Bortolussi, L., Nenzi, L., and Sanguinetti, G. (2015). System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*.
- Bartocci, E., Corradini, F., Merelli, E., and Tesei, L. (2010). Detecting synchronisation of biological oscillators by model checking. *Theoretical Computer Science*, 411(20):1999–2018.
- Batt, G., Belta, C., and Weiss, R. (2007a). Model Checking Genetic Regulatory Networks with Parameter Uncertainty. In Bemporad, A., Bicchi, A., and Buttazzo, G., editors, *Hybrid Systems: Computation and Control*, number 4416 in Lecture Notes in Computer Science, pages 61–75. Springer Berlin Heidelberg.
- Batt, G., Belta, C., and Weiss, R. (2008). Temporal Logic Analysis of Gene Networks Under Parameter Uncertainty. *IEEE Transactions on Automatic Control*, 53(Special Issue):215–229.
- Batt, G., Ropers, D., Jong, H. d., Geiselmann, J., Mateescu, R., Page, M., and Schneider, D. (2005). Validation of qualitative models of genetic regulatory networks by model checking: analysis of the nutritional stress response in

- Escherichia coli. *Bioinformatics*, 21(suppl 1):i19–i28.
- Batt, G., Yordanov, B., Weiss, R., and Belta, C. (2007b). Robustness analysis and tuning of synthetic gene networks. *Bioinformatics*, 23(18):2415–2422.
- Bauer, A. L., Beauchemin, C. A. A., and Perelson, A. S. (2009). Agent-based modeling of host-pathogen systems: The successes and challenges. *Information Sciences*, 179(10):1379–1389.
- Beard, D. A., Neal, M. L., Tabesh-Saleki, N., Thompson, C. T., Bassingwaite, J. B., Shimoyama, M., and Carlson, B. E. (2012). Multiscale Modeling and Data Integration in the Virtual Physiological Rat Project. *Annals of Biomedical Engineering*, 40(11):2365–2378.
- Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø., and Herrgard, M. J. (2007). Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nature Protocols*, 2(3):727–738.
- Behrmann, G., David, A., Larsen, K. G., Pettersson, P., and Yi, W. (2011). Developing UPPAAL over 15 years. *Software: Practice and Experience*, 41(2):133–142.
- Bérenguier, D., Chaouiya, C., Monteiro, P. T., Naldi, A., Remy, E., Thieffry, D., and Tichit, L. (2013). Dynamical modeling and analysis of large cellular regulatory networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 23(2):025114.
- Bernard, C. (1865). *Introduction à l'étude de la médecine expérimentale*. J. B. Bailliére et Fils, Paris, France.
- Bernard, S. (2013). How to Build a Multiscale Model in Biology. *Acta Biotheoretica*, 61(3):291–303.
- Bernot, G., Comet, J.-P., Richard, A., and Guespin, J. (2004). Application of formal methods to biological regulatory networks: extending Thomas' asynchronous logical approach with temporal logic. *Journal of Theoretical Biology*, 229(3):339–347.
- Besozzi, D., Cazzaniga, P., Pescini, D., and Mauri, G. (2008). Modelling metapopulations with stochastic membrane systems. *Biosystems*, 91(3):499–514.
- Biggs, M. B. and Papin, J. A. (2013). Novel Multiscale Modeling Tool Applied to *Pseudomonas aeruginosa* Biofilm Formation. *PLoS ONE*, 8(10):e78011.
- Blinov, M. L., Faeder, J. R., Goldstein, B., and Hlavacek, W. S. (2004). BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291.
- Boissel, J.-P., Auffray, C., Noble, D., Hood, L., and Boissel, F.-H. (2015). Bridging Systems Medicine and Patient Needs. *CPT: Pharmacometrics & Systems Pharmacology*, 4(3):135–145.
- Bondy, A. and Murty, U. S. R. (2010). *Graph Theory*. Graduate Texts in Mathematics. Springer, New York, 2008 edition.
- Borgdorff, J., Belgacem, M. B., Bona-Casas, C., Fazendeiro, L., Groen, D., Hoenen, O., Mizeranschi, A., Suter, J. L., Coster, D., Coveney, P. V., Dubitzky, W., Hoekstra, A. G., Strand, P., and Chopard, B. (2014a). Performance of distributed multiscale simulations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2021):20130407.
- Borgdorff, J., Mamonski, M., Bosak, B., Kurowski, K., Belgacem, M. B., Chopard, B., Groen, D., Coveney, P. V., and Hoekstra, A. G. (2014b). Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and

- Environment. *Journal of Computational Science*, 5(5):719 – 731.
- Bouteiller, J.-M., Allam, S., Hu, E., Greget, R., Ambert, N., Keller, A., Bischoff, S., Baudry, M., and Berger, T. (2011). Integrated Multiscale Modeling of the Nervous System: Predicting Changes in Hippocampal Network Activity by a Positive AMPA Receptor Modulator. *IEEE Transactions on Biomedical Engineering*, 58(10):3008–3011.
- Boyer, B., Corre, K., Legay, A., and Sedwards, S. (2013). PLASMA-lab: A Flexible, Distributable Statistical Model Checking Library. In Joshi, K., Siegle, M., Stoelinga, M., and D’Argenio, P. R., editors, *Quantitative Evaluation of Systems*, number 8054 in Lecture Notes in Computer Science, pages 160–164. Springer Berlin Heidelberg.
- Bradley, C., Bowery, A., Britten, R., Budelmann, V., Camara, O., Christie, R., Cookson, A., Frangi, A. F., Gamage, T. B., Heidlauf, T., Krittian, S., Ladd, D., Little, C., Mithraratne, K., Nash, M., Nickerson, D., Nielsen, P., Nordbø, Ø., Omholt, S., Pashaei, A., Paterson, D., Rajagopal, V., Reeve, A., Röhrle, O., Safaei, S., Sebastián, R., Steghöfer, M., Wu, T., Yu, T., Zhang, H., and Hunter, P. (2011). OpenCMISS: A multi-physics & multi-scale computational infrastructure for the VPH/Physiome project. *Progress in Biophysics and Molecular Biology*, 107(1):32–47.
- Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA.
- Braz, F., Cruz, J., Faria-Campos, A., and Campos, S. (2013). Probabilistic Model Checking Analysis of Palytoxin Effects on Cell Energy Reactions of the Na⁺/K⁺-ATPase. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(6):1530–1541.
- Bresolin, D., Sala, P., Della Monica, D., Montanari, A., and Sciavicco, G. (2010). A Decidable Spatial Generalization of Metric Interval Temporal Logic. In *2010 17th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 95–102.
- Brim, L., Češka, M., Dražan, S., and Šafránek, D. (2013a). Exploring Parameter Space of Stochastic Biochemical Systems Using Quantitative Model Checking. In Sharygina, N. and Veith, H., editors, *Computer Aided Verification*, number 8044 in Lecture Notes in Computer Science, pages 107–123. Springer Berlin Heidelberg.
- Brim, L., Češka, M., and Šafránek, D. (2013b). Model Checking of Biological Systems. In Bernardo, M., Vink, E. d., Pierro, A. D., and Wiklicky, H., editors, *Formal Methods for Dynamical Systems*, number 7938 in Lecture Notes in Computer Science, pages 63–112. Springer Berlin Heidelberg.
- Bryant, R. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.
- Büchi, J. R. (1962). On a Decision Method in Restricted Second Order Arithmetic. In *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11, Stanford, California, USA. Stanford University Press.
- Bugenhagen, S. M., Cowley, A. W., and Beard, D. A. (2010). Identifying physiological origins of baroreflex dysfunction in salt-sensitive hypertension in the Dahl SS rat. *Physiological Genomics*, 42(1):23–41.

- Bui, T. H. and Nymeyer, A. (2009). Formal Verification Based on Guided Random Walks. In Leuschel, M. and Wehrheim, H., editors, *Integrated Formal Methods*, number 5423 in Lecture Notes in Computer Science, pages 72–87. Springer Berlin Heidelberg.
- Bulychev, P., David, A., Larsen, K. G., Mikućionis, M., Poulsen, D. B., Legay, A., and Wang, Z. (2012). UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. *Electronic Proceedings in Theoretical Computer Science*, 85:1–16.
- Cai, H. and Devreotes, P. N. (2011). Moving in the right direction: How eukaryotic cells migrate along chemical gradients. *Seminars in Cell & Developmental Biology*, 22(8):834–841.
- Caiazzo, A., Evans, D., Falcone, J.-L., Hegewald, J., Lorenz, E., Stahl, B., Wang, D., Bernsdorf, J., Chopard, B., Gunn, J., Hose, R., Krafczyk, M., Lawford, P., Smallwood, R., Walker, D., and Hoekstra, A. (2011). A Complex Automata approach for in-stent restenosis: Two-dimensional multiscale modelling and simulations. *Journal of Computational Science*, 2(1):9–17.
- Calder, M., Vysemirsky, V., Gilbert, D., and Orton, R. (2006). Analysis of Signalling Pathways Using Continuous Time Markov Chains. In Priami, C. and Plotkin, G., editors, *Transactions on Computational Systems Biology VI*, number 4220 in Lecture Notes in Computer Science, pages 44–67. Springer Berlin Heidelberg.
- Calzone, L., Chabrier-Rivier, N., Fages, F., and Soliman, S. (2006). Machine Learning Biochemical Networks from Temporal Logic Properties. In Priami, C. and Plotkin, G., editors, *Transactions on Computational Systems Biology VI*, number 4220 in Lecture Notes in Computer Science, pages 68–94. Springer Berlin Heidelberg.
- Cassandras, C. G. and Lafourne, S. (2008). *Introduction to Discrete Event Systems*. Springer Science & Business Media.
- Cavaliere, M., Mazza, T., and Sedwards, S. (2014). Statistical Model Checking of Membrane Systems with Peripheral Proteins: Quantifying the Role of Estrogen in Cellular Mitosis and DNA Damage. In Frisco, P., Gheorghe, M., and Pérez-Jiménez, M. J., editors, *Applications of Membrane Computing in Systems and Synthetic Biology*, number 7 in Emergence, Complexity and Computation, pages 43–63. Springer International Publishing.
- Ceccarelli, M., Cerulo, L., and Santone, A. (2014). De novo reconstruction of gene regulatory networks from time series data, an approach based on formal methods. *Methods*, 69(3):298–305.
- Češka, M., Dannenberg, F., Kwiatkowska, M., and Paoletti, N. (2014). Precise Parameter Synthesis for Stochastic Biochemical Systems. In Mendes, P., Dada, J. O., and Smallbone, K., editors, *Computational Methods in Systems Biology*, number 8859 in Lecture Notes in Computer Science, pages 86–98. Springer International Publishing.
- Česka, M., Šafránek, D., Dražan, S., and Brim, L. (2014). Robustness Analysis of Stochastic Biochemical Systems. *PLoS ONE*, 9(4):e94553.
- Chabrier, N. and Fages, F. (2003). Symbolic Model Checking of Biochemical Networks. In Priami, C., editor, *Computational Methods in Systems Biology*, number 2602 in Lecture Notes in Computer Science, pages 149–162. Springer

- Berlin Heidelberg, Rovereto, Italy.
- Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., and Schächter, V. (2004). Modeling and querying biomolecular interaction networks. *Theoretical Computer Science*, 325(1):25–44.
- Chaudhary, S. U., Shin, S.-Y., Lee, D., Song, J.-H., and Cho, K.-H. (2013). ELECANS—an integrated model development environment for multiscale cancer systems biology. *Bioinformatics*, 29(7):957–959.
- Cheng, A. A. and Lu, T. K. (2012). Synthetic biology: an emerging engineering discipline. *Annual review of biomedical engineering*, 14:155–178.
- Chopard, B., Borgdorff, J., and Hoekstra, A. G. (2014). A framework for multiscale modelling. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 372(2021).
- Christie, G. R., Nielsen, P. M. F., Blackett, S. A., Bradley, C. P., and Hunter, P. J. (2009). FieldML: concepts and implementation. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1895):1869–1884.
- Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In Brinksma, E. and Larsen, K. G., editors, *Computer Aided Verification*, number 2404 in Lecture Notes in Computer Science, pages 359–364. Springer Berlin Heidelberg.
- Cimatti, A., Clarke, E. M., Giunchiglia, F., and Roveri, M. (1999). NuSMV: A New Symbolic Model Verifier. In *Proceedings of the 11th International Conference on Computer Aided Verification*, CAV ’99, pages 495–499, London, UK. Springer-Verlag.
- Ciocchetta, F., Gilmore, S., Guerriero, M. L., and Hillston, J. (2009). Integrated Simulation and Model-Checking for the Analysis of Biochemical Systems. *Electronic Notes in Theoretical Computer Science*, 232:17–38.
- Clarke, E., Donzé, A., and Legay, A. (2010). On simulation-based probabilistic model checking of mixed-analog circuits. *Formal Methods in System Design*, 36(2):97–113.
- Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2001). Progress on the State Explosion Problem in Model Checking. In Wilhelm, R., editor, *Informatics*, number 2000 in Lecture Notes in Computer Science, pages 176–194. Springer Berlin Heidelberg.
- Clarke, E. M. (2008). The Birth of Model Checking. In Grumberg, O. and Veith, H., editors, *25 Years of Model Checking*, number 5000 in Lecture Notes in Computer Science, pages 1–26. Springer Berlin Heidelberg.
- Clarke, E. M. and Emerson, E. A. (1982). Design and synthesis of synchronization skeletons using branching time temporal logic. In Kozen, D., editor, *Logics of Programs*, number 131 in Lecture Notes in Computer Science, pages 52–71. Springer Berlin Heidelberg.
- Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263.
- Clarke, E. M., Enders, R., Filkorn, T., and Jha, S. (1996). Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1-2):77–

- 104.
- Clarke, E. M., Faeder, J. R., Langmead, C. J., Harris, L. A., Jha, S. K., and Legay, A. (2008). Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-Cell Receptor Signaling Pathway. In Heiner, M. and Uhrmacher, A. M., editors, *Computational Methods in Systems Biology*, number 5307 in Lecture Notes in Computer Science, pages 231–250. Springer Berlin Heidelberg.
- Clarke, E. M., Grumberg, O., and Hamaguchi, K. (1997). Another Look at LTL Model Checking. *Formal Methods in System Design*, 10(1):47–71.
- Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press.
- Clementini, E., Felice, P. D., and Hernández, D. (1997). Qualitative representation of positional information. *Artificial Intelligence*, 95(2):317–356.
- Condotta, J.-F. (2000). The Augmented Interval and Rectangle Networks. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, pages 571–579, Breckenridge, Colorado, USA.
- Corrias, A., Pathmanathan, P., Gavaghan, D. J., and Buist, M. L. (2012). Modelling tissue electrophysiology with multiple cell types: applications of the extended bidomain framework. *Integrative Biology*, 4(2):192–201.
- Courcoubetis, C. and Yannakakis, M. (1995). The Complexity of Probabilistic Verification. *J. ACM*, 42(4):857–907.
- Courtot, M., Juty, N., Knüpfer, C., Waltemath, D., Zhukova, A., Dräger, A., Dumontier, M., Finney, A., Golebiewski, M., Hastings, J., Hoops, S., Keating, S., Kell, D. B., Kerrien, S., Lawson, J., Lister, A., Lu, J., Machne, R., Mendes, P., Pocock, M., Rodriguez, N., Villeger, A., Wilkinson, D. J., Wimalaratne, S., Laibe, C., Hucka, M., and Novère, N. L. (2011). Controlled vocabularies and semantics in systems biology. *Molecular Systems Biology*, 7(1).
- Dada, J. O. and Mendes, P. (2011). Multi-scale modelling and simulation in systems biology. *Integrative biology: quantitative biosciences from nano to macro*, 3(2):86–96.
- Dada, J. O. and Mendes, P. (2012). ManyCell: A Multiscale Simulator for Cellular Systems. In Gilbert, D. and Heiner, M., editors, *Computational Methods in Systems Biology*, Lecture Notes in Computer Science, pages 366–369. Springer Berlin Heidelberg.
- Danos, V., Feret, J., Fontana, W., Harmer, R., and Krivine, J. (2007). Rule-Based Modelling of Cellular Signalling. In Caires, L. and Vasconcelos, V. T., editors, *CONCUR 2007 – Concurrency Theory*, number 4703 in Lecture Notes in Computer Science, pages 17–41. Springer Berlin Heidelberg.
- D’Argenio, P. R. and Katoen, J.-P. (2005). A theory of stochastic systems part I: Stochastic automata. *Information and Computation*, 203(1):1–38.
- David, A., Larsen, K. G., Legay, A., Mikucionis, M., Poulsen, D. B., and Sedwards, S. (2012). Runtime Verification of Biological Systems. In Margaria, T. and Steffen, B., editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, number 7609 in Lecture Notes in Computer Science, pages 388–404. Springer Berlin Heidelberg.
- Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Operating Systems Design & Implementation*, volume 6 of *OSDI’04*, pages 10–10, Berkeley, CA,

- USA. USENIX Association.
- Deisboeck, T. S., Wang, Z., Macklin, P., and Cristini, V. (2011). Multiscale Cancer Modeling. *Annual Review of Biomedical Engineering*, 13(1):127–155.
- Deutsch, A. and Dormann, S. (2007). *Cellular Automaton Modeling of Biological Pattern Formation: Characterization, Applications, and Analysis*. Springer Science & Business Media.
- Di Ventura, B., Lemerle, C., Michalodimitrakis, K., and Serrano, L. (2006). From in vivo to in silico biology and back. *Nature*, 443(7111):527–533.
- Docker (2015). Docker - Build, Ship, and Run Any App, Anywhere. <https://www.docker.com/>, Last accessed on: 2015-03-31.
- Donaldson, R. and Gilbert, D. (2008a). A Model Checking Approach to the Parameter Estimation of Biochemical Pathways. In Heiner, M. and Uhrmacher, A. M., editors, *Computational Methods in Systems Biology*, number 5307 in Lecture Notes in Computer Science, pages 269–287. Springer Berlin Heidelberg.
- Donaldson, R. and Gilbert, D. (2008b). A Monte Carlo model checker for probabilistic LTL with numerical constraints. Report, University of Glasgow.
- Du, P., O’Grady, G., Gao, J., Sathar, S., and Cheng, L. K. (2013a). Toward the Virtual Stomach: Progress in Multi-scale Modeling of Gastric Electrophysiology and Motility. *Wiley interdisciplinary reviews. Systems biology and medicine*, 5(4):481–493.
- Du, S., Feng, C.-C., and Wang, Q. (2013b). Multi-scale qualitative location: A direction-based model. *Computers, Environment and Urban Systems*, 41:151–166.
- Du, S., Feng, C.-C., Wang, Q., and Guo, L. (2014). Multi-Scale Qualitative Location: A Topology-Based Model. *Transactions in GIS*, 18(4):604–631.
- Dwivedi, G., Fitz, L., Hegen, M., Martin, S., Harrold, J., Heatherington, A., and Li, C. (2014). A Multiscale Model of Interleukin-6–Mediated Immune Regulation in Crohn’s Disease and Its Application in Drug Discovery and Development. *CPT: Pharmacometrics & Systems Pharmacology*, 3(1):1–9.
- Emerson, E. A. (1995). Temporal and Modal Logic. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. MIT Press, Cambridge, MA, USA.
- Emerson, E. A. (2008). The Beginning of Model Checking: A Personal Perspective. In Grumberg, O. and Veith, H., editors, *25 Years of Model Checking*, number 5000 in Lecture Notes in Computer Science, pages 27–45. Springer Berlin Heidelberg.
- Emerson, E. A. and Halpern, J. Y. (1986). “Sometimes” and “Not Never” Revisited: On Branching Versus Linear Time Temporal Logic. *J. ACM*, 33(1):151–178.
- Emerson, E. A. and Lei, C.-L. (1987). Modalities for model checking: branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306.
- Emerson, E. A., Mok, A. K., Sistla, A. P., and Srinivasan, J. (1992). Quantitative temporal reasoning. *Real-Time Systems*, 4(4):331–352.
- Emerson, E. A. and Sistla, A. P. (1996). Symmetry and model checking. *Formal Methods in System Design*, 9(1-2):105–131.
- Endy, D. (2005). Foundations for engineering biology. *Nature*, 438(7067):449–453.
- Engl, H. W., Flamm, C., Kügler, P., Lu, J., Müller, S., and Schuster, P. (2009).

- Inverse problems in systems biology. *Inverse Problems*, 25(12):123014.
- Ermentrout, G. B. and Edelstein-Keshet, L. (1993). Cellular Automata Approaches to Biological Modeling. *Journal of Theoretical Biology*, 160(1):97–133.
- Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, USA.
- Ewald, R. and Uhrmacher, A. M. (2014). SESSL: A Domain-specific Language for Simulation Experiments. *ACM Trans. Model. Comput. Simul.*, 24(2):11:1–11:25.
- Fages, F. and Rizk, A. (2009). From Model-Checking to Temporal Logic Constraint Solving. In Gent, I. P., editor, *Principles and Practice of Constraint Programming - CP 2009*, number 5732 in Lecture Notes in Computer Science, pages 319–334. Springer Berlin Heidelberg.
- Fages, F. and Soliman, S. (2008). Formal Cell Biology in Biocham. In Bernardo, M., Degano, P., and Zavattaro, G., editors, *Formal Methods for Computational Systems Biology*, number 5016 in Lecture Notes in Computer Science, pages 54–80. Springer Berlin Heidelberg.
- Falcone, J.-L., Chopard, B., and Hoekstra, A. (2010). MML: towards a Multiscale Modeling Language. *Procedia Computer Science*, 1(1):819–826.
- Feng, C. and Hillston, J. (2014). PALOMA: A Process Algebra for Located Markovian Agents. In Norman, G. and Sanders, W., editors, *Quantitative Evaluation of Systems*, pages 265–280. Springer International Publishing.
- Ferrell Jr., J. E., Tsai, T. Y.-C., and Yang, Q. (2011). Modeling the Cell Cycle: Why Do Certain Circuits Oscillate? *Cell*, 144(6):874–885.
- Ferrer, J., Prats, C., and López, D. (2008). Individual-based Modelling: An Essential Tool for Microbiology. *Journal of Biological Physics*, 34(1-2):19–37.
- Finkbeiner, B. and Sipma, H. (2001). Checking Finite Traces using Alternating Automata. *Electronic Notes in Theoretical Computer Science*, 55(2):147–163.
- Fisher, J. and Piterman, N. (2014). Model Checking in Biology. In Kulkarni, V. V., Stan, G.-B., and Raman, K., editors, *A Systems Theoretic Approach to Systems and Synthetic Biology I: Models and System Characterizations*, pages 255–279. Springer Netherlands.
- Florin, G., Fraize, C., and Natkin, S. (1991). Stochastic Petri nets: Properties, applications and tools. *Microelectronics Reliability*, 31(4):669–697.
- Formaggia, L., Nobile, F., Quarteroni, A., and Veneziani, A. (1999). Multiscale modelling of the circulatory system: a preliminary analysis. *Computing and Visualization in Science*, 2(2-3):75–83.
- Freeman, H. and Shapira, R. (1975). Determining the Minimum-area Encasing Rectangle for an Arbitrary Closed Curve. *Commun. ACM*, 18(7):409–413.
- Gao, Q., Gilbert, D., Heiner, M., Liu, F., Maccagnola, D., and Tree, D. (2013). Multiscale Modeling and Analysis of Planar Cell Polarity in the Drosophila Wing. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(2):337–351.
- Gärtner, B. (1999). Fast and Robust Smallest Enclosing Balls. In Nešetřil, J., editor, *Algorithms - ESA '99*, number 1643 in Lecture Notes in Computer Science, pages 325–338. Springer Berlin Heidelberg.
- Gerth, R., Peled, D., Vardi, M. Y., and Wolper, P. (1996). Simple On-the-fly

- Automatic Verification of Linear Temporal Logic. In *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, page 3–18, London, UK. Chapman & Hall, Ltd.
- Ghosh, S., Matsuoka, Y., Asai, Y., Hsin, K.-Y., and Kitano, H. (2011). Software for systems biology: from tools to integrated platforms. *Nature Reviews Genetics*, 12(12):821–832.
- Giacobbe, M., Guet, C. C., Gupta, A., Henzinger, T. A., Paixao, T., and Petrov, T. (2015). Model Checking Gene Regulatory Networks. *arXiv:1410.7704 [cs, q-bio]*.
- Gilbert, D., Heiner, M., and Lehrack, S. (2007). A Unifying Framework for Modelling and Analysing Biochemical Pathways Using Petri Nets. In Calder, M. and Gilmore, S., editors, *Computational Methods in Systems Biology*, number 4695 in Lecture Notes in Computer Science, pages 200–216. Springer Berlin Heidelberg.
- Gilbert, D., Heiner, M., Liu, F., and Saunders, N. (2013). Colouring Space - A Coloured Framework for Spatial Modelling in Systems Biology. In Colom, J.-M. and Desel, J., editors, *Application and Theory of Petri Nets and Concurrency*, number 7927 in Lecture Notes in Computer Science, pages 230–249. Springer Berlin Heidelberg.
- Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361.
- Godefroid, P. (1991). Using partial orders to improve automatic verification methods. In Clarke, E. M. and Kurshan, R. P., editors, *Computer Aided Verification*, number 531 in Lecture Notes in Computer Science, pages 176–185. Springer Berlin Heidelberg.
- Godefroid, P. and Khurshid, S. (2002). Exploring Very Large State Spaces Using Genetic Algorithms. In Katoen, J.-P. and Stevens, P., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, number 2280 in Lecture Notes in Computer Science, pages 266–280. Springer Berlin Heidelberg.
- Gol, E. A., Bartocci, E., and Belta, C. (2014). A Formal Methods Approach to Pattern Synthesis in Reaction Diffusion Systems. *arXiv:1409.5671 [cs]*.
- Gong, H. and Feng, L. (2014). Computational analysis of the roles of ER-Golgi network in the cell cycle. *BMC Systems Biology*, 8(Suppl 4):S3.
- Gong, H., Zuliani, P., Komuravelli, A., Faeder, J. R., and Clarke, E. M. (2012). Computational Modeling and Verification of Signaling Pathways in Cancer. In Horimoto, K., Nakatsui, M., and Popov, N., editors, *Algebraic and Numeric Biology*, number 6479 in Lecture Notes in Computer Science, pages 117–135. Springer Berlin Heidelberg.
- Google (2015). Google Test. <https://github.com/google/googletest>, Last accessed on: 2015-11-30.
- Grafahrend-Belau, E., Junker, A., Eschenröder, A., Müller, J., Schreiber, F., and Junker, B. H. (2013). Multiscale metabolic modeling: dynamic flux balance analysis on a whole plant scale. *Plant Physiology*.
- Graner and Glazier (1992). Simulation of biological cell sorting using a two-dimensional extended Potts model. *Physical review letters*, 69(13):2013–2016.
- Graudenzi, A., Caravagna, G., De Matteis, G., and Antoniotti, M. (2014). Investigating the Relation between Stochastic Differentiation, Homeostasis and

- Clonal Expansion in Intestinal Crypts via Multiscale Modeling. *PLoS ONE*, 9(5):e97272.
- Groen, D., Zasada, S. J., and Coveney, P. V. (2014). Survey of Multiscale and Multiphysics Applications and Communities. *Computing in Science Engineering*, 16(2):34–43.
- Groote, J. F., Kouters, T. W. D. M., and Osaiweran, A. (2012). Specification Guidelines to Avoid the State Space Explosion Problem. In Arbab, F. and Sirjani, M., editors, *Fundamentals of Software Engineering*, number 7141 in Lecture Notes in Computer Science, pages 112–127. Springer Berlin Heidelberg.
- Grosu, R. and Smolka, S. A. (2005). Monte Carlo Model Checking. In Halbwachs, N. and Zuck, L. D., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, number 3440 in Lecture Notes in Computer Science, pages 271–286. Springer Berlin Heidelberg.
- Grosu, R., Smolka, S. A., Corradini, F., Wasilewska, A., Entcheva, E., and Bartocci, E. (2009). Learning and Detecting Emergent Behavior in Networks of Cardiac Myocytes. *Commun. ACM*, 52(3):97–105.
- Grubbs, F. E. (1949). On Designing Single Sampling Inspection Plans. *The Annals of Mathematical Statistics*, 20(2):242–256.
- Guerriero, M. L. (2009). Qualitative and Quantitative Analysis of a Bio-PEPA Model of the Gp130/JAK/STAT Signalling Pathway. In Priami, C., Back, R.-J., and Petre, I., editors, *Transactions on Computational Systems Biology XI*, number 5750 in Lecture Notes in Computer Science, pages 90–115. Springer Berlin Heidelberg.
- Guzman, J. d. and Kaiser, H. (2015). Boost Spirit. http://www.boost.org/doc/libs/1_55_0/libs/spirit/doc/html/index.html, Last accessed on: 2015-11-30.
- Haghghi, I., Jones, A., Kong, Z., Bartocci, E., Grosu, R., and Belta, C. (2015). SpaTeL: A Novel Spatial-temporal Logic and Its Applications to Networked Systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC ’15, pages 189–198, New York, NY, USA. ACM.
- Hansson, H. and Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535.
- Hardy, S. and Robillard, P. N. (2004). Modeling and simulation of molecular biology systems using petri nets: modeling goals of various approaches. *Journal of Bioinformatics and Computational Biology*, 02(04):619–637.
- Harrison, P. G. and Strulo, B. (1995). Stochastic Process Algebra for Discrete Event Simulation. In Baccelli, F., Jean-Marie, A., and Mitrani, I., editors, *Quantitative Methods in Parallel Systems*, Esprit Basic Research Series, pages 18–37. Springer Berlin Heidelberg.
- He, F., Yeung, L. F., and Brown, M. (2008). Discrete-Time Model Representations for Biochemical Pathways. In Castillo, O., Xu, L., and Ao, S.-I., editors, *Trends in Intelligent Systems and Computer Engineering*, number 6 in Lecture Notes in Electrical Engineering, pages 255–271. Springer US.
- Heath, A. P. and Kavraki, L. E. (2009). Computational challenges in systems biology. *Computer Science Review*, 3(1):1–17.
- Heath, J., Kwiatkowska, M., Norman, G., Parker, D., and Tymchyshyn, O. (2008). Probabilistic model checking of complex biological pathways. *Theoretical*

- Computer Science*, 391(3):239–257.
- Heiner, M., Gilbert, D., and Donaldson, R. (2008). Petri Nets for Systems and Synthetic Biology. In Bernardo, M., Degano, P., and Zavattaro, G., editors, *Formal Methods for Computational Systems Biology*, number 5016 in Lecture Notes in Computer Science, pages 215–264. Springer Berlin Heidelberg.
- Heiner, M., Herajy, M., Liu, F., Rohr, C., and Schwarick, M. (2012). Snoopy – A Unifying Petri Net Tool. In Haddad, S. and Pomello, L., editors, *Application and Theory of Petri Nets*, number 7347 in Lecture Notes in Computer Science, pages 398–407. Springer Berlin Heidelberg, Hamburg, Germany.
- Heiner, M., Rohr, C., and Schwarick, M. (2013). – Model Checking and Reachability Analysis Done Efficiently. In Colom, J.-M. and Desel, J., editors, *Application and Theory of Petri Nets and Concurrency*, number 7927 in Lecture Notes in Computer Science, pages 389–399. Springer Berlin Heidelberg, Milano, Italy.
- Helms, T., Himmelsbach, J., Maus, C., Röwer, O., Schützel, J., and Uhrmacher, A. M. (2012). Toward a Language for the Flexible Observation of Simulations. In *Proceedings of the Winter Simulation Conference*, WSC ’12, pages 418:1–418:12, Berlin, Germany. Winter Simulation Conference.
- Helms, T., Maus, C., Haack, F., and Uhrmacher, A. M. (2014). Multi-level Modeling and Simulation of Cell Biological Systems with ML-rules: A Tutorial. In *Proceedings of the 2014 Winter Simulation Conference*, WSC ’14, pages 177–191, Piscataway, NJ, USA. IEEE Press.
- Henzinger, T. (1996). The theory of hybrid automata. In *Proceedings of Eleventh Annual IEEE Symposium on Logic in Computer Science, 1996*, pages 278–292.
- Hérault, T., Lassaigne, R., Magniette, F., and Peyronnet, S. (2004). Approximate Probabilistic Model Checking. In Steffen, B. and Levi, G., editors, *Verification, Model Checking, and Abstract Interpretation*, number 2937 in Lecture Notes in Computer Science, pages 73–84. Springer Berlin Heidelberg.
- Hermanns, H., Katoen, J.-P., Meyer-Kayser, J., and Siegle, M. (2000). Towards Model Checking Stochastic Process Algebra. In Grieskamp, W., Santen, T., and Stoddart, B., editors, *Integrated Formal Methods*, number 1945 in Lecture Notes in Computer Science, pages 420–439. Springer Berlin Heidelberg.
- Hillen, T. and Painter, K. J. (2009). A user’s guide to PDE models for chemotaxis. *Journal of Mathematical Biology*, 58(1-2):183–217.
- Hoeffding, W. (1963). Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30.
- Hoekstra, A., Chopard, B., and Coveney, P. (2014). Multiscale modelling and simulation: a position paper. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2021):20130377.
- Hoekstra, A. G., Lorenz, E., Falcone, J.-L., and Chopard, B. (2007). Towards a Complex Automata Framework for Multi-scale Modeling: Formalism and the Scale Separation Map. In Shi, Y., Albada, G. D. v., Dongarra, J., and Sloot, P. M. A., editors, *Computational Science – ICCS 2007*, number 4487 in Lecture Notes in Computer Science, pages 922–930. Springer Berlin Heidelberg.
- Holzhüter, H.-G., Drasdo, D., Preusser, T., Lippert, J., and Henney, A. M. (2012). The virtual liver: a multidisciplinary, multilevel challenge for systems biology. *Wiley Interdisciplinary Reviews. Systems Biology and Medicine*, 4(3):221–235.
- Holzmann, G. (1997). The model checker SPIN. *IEEE Transactions on Software*

- Engineering*, 23(5):279–295.
- Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. (2006). COPASI—a COmplex PAthway SImlator. *Bioinformatics*, 22(24):3067–3074.
- Hu, J., Lygeros, J., and Sastry, S. (2000). Towards a Theory of Stochastic Hybrid Systems. In Lynch, N. and Krogh, B. H., editors, *Hybrid Systems: Computation and Control*, number 1790 in Lecture Notes in Computer Science, pages 160–173. Springer Berlin Heidelberg.
- Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., Arkin, A. P., Bornstein, B. J., Bray, D., Cornish-Bowden, A., Cuellar, A. A., Dronov, S., Gilles, E. D., Ginkel, M., Gor, V., Goryanin, I. I., Hedley, W. J., Hodgman, T. C., Hofmeyr, J.-H., Hunter, P. J., Juty, N. S., Kasberger, J. L., Kremling, A., Kummer, U., Novère, N. L., Loew, L. M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E. D., Nakayama, Y., Nelson, M. R., Nielsen, P. F., Sakurada, T., Schaff, J. C., Shapiro, B. E., Shimizu, T. S., Spence, H. D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., and Wang, J. (2003). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531.
- Hunter, P. J. and Borg, T. K. (2003). Integration from proteins to organs: the Physiome Project. *Nature Reviews Molecular Cell Biology*, 4(3):237–243.
- Hussain, F., Jha, S. K., Jha, S., and Langmead, C. J. (2014a). Parameter discovery in stochastic biological models using simulated annealing and statistical model checking. *International Journal of Bioinformatics Research and Applications*, 10(4):519–539.
- Hussain, F., Ramanathan, A., Pullum, L., and Jha, S. (2014b). EpiSpec: A formal specification language for parameterized agent-based models against epidemiological ground truth. In *2014 IEEE 4th International Conference on Computational Advances in Bio and Medical Sciences (ICCAKS)*, pages 1–6, Miami, FL. IEEE.
- Ideker, T., Galitski, T., and Hood, L. (2001). A NEW APPROACH TO DECODING LIFE: Systems Biology. *Annual Review of Genomics and Human Genetics*, 2(1):343–372.
- Ip, C. N. and Dill, D. L. (1996). Better verification through symmetry. *Formal Methods in System Design*, 9(1-2):41–75.
- Islam, M. A., DeFrancisco, R., Fan, C., Grosu, R., Mitra, S., and Smolka, S. A. (2015). Model Checking Tap Withdrawal in C. Elegans. *arXiv:1503.06480 [cs, q-bio]*.
- Itseez (2013). OpenCV documentation. <http://docs.opencv.org>, Last accessed on: 2013-12-13.
- Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666.
- Jeffreys, S. H. (1961). *The Theory of Probability*. Oxford University Press, 3 edition.
- Jha, S. and Ramanathan, A. (2012). Quantifying Uncertainty in Epidemiological Models. In *2012 ASE/IEEE International Conference on BioMedical Computing (BioMedCom)*, pages 80–85.
- Jha, S. K., Clarke, E. M., Langmead, C. J., Legay, A., Platzer, A., and Zuliani,

- P. (2009a). A Bayesian Approach to Model Checking Biological Systems. In Degano, P. and Gorrieri, R., editors, *Computational Methods in Systems Biology*, number 5688 in Lecture Notes in Computer Science, pages 218–234. Springer Berlin Heidelberg.
- Jha, S. K., Clarke, E. M., Langmead, C. J., Legay, A., Platzer, A., and Zuliani, P. (2009b). Statistical Model Checking for Complex Stochastic Models in Systems Biology. Technical report, Carnegie Mellon University.
- Jha, S. K. and Langmead, C. J. (2011). Synthesis and infeasibility analysis for stochastic models of biochemical systems using statistical model checking and abstraction refinement. *Theoretical Computer Science*, 412(21):2162–2187.
- Jilkine, A. and Edelstein-Keshet, L. (2011). A Comparison of Mathematical Models for Polarization of Single Eukaryotic Cells in Response to Guided Cues. *PLoS Comput Biol*, 7(4):e1001121.
- Jin, T. (2013). Gradient sensing during chemotaxis. *Current Opinion in Cell Biology*, 25(5):532–537.
- John, M., Ewald, R., and Uhrmacher, A. M. (2008). A Spatial Extension to the π Calculus. *Electronic Notes in Theoretical Computer Science*, 194(3):133–148.
- John, M., Lhoussaine, C., Niehren, J., and Uhrmacher, A. M. (2010). The Attributed Pi-Calculus with Priorities. In Priami, C., Breitling, R., Gilbert, D., Heiner, M., and Uhrmacher, A. M., editors, *Transactions on Computational Systems Biology XII*, pages 13–76. Springer Berlin Heidelberg.
- John, M., Lhoussaine, C., Niehren, J., and Versari, C. (2011). Biochemical Reaction Rules with Constraints. In Barthe, G., editor, *Programming Languages and Systems*, number 6602 in Lecture Notes in Computer Science, pages 338–357. Springer Berlin Heidelberg.
- Kaazempur-Mofrad, M. R., Bathe, M., Karcher, H., Younis, H. F., Seong, H. C., Shim, E. B., Chan, R. C., Hinton, D. P., Isasi, A. G., Upadhyaya, A., Powers, M. J., Griffith, L. G., and Kamm, R. D. (2003). Role of simulation in understanding biological systems. *Computers & Structures*, 81(8–11):715–726.
- Kaletta, T. and Hengartner, M. O. (2006). Finding function in novel targets: *C. elegans* as a model organism. *Nature Reviews Drug Discovery*, 5(5):387–399.
- Katoen, J.-P., Zapreev, I. S., Hahn, E. M., Hermanns, H., and Jansen, D. N. (2011). The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104.
- Kauffman, S. (1969). Homeostasis and Differentiation in Random Genetic Control Networks. *Nature*, 224(5215):177–178.
- Kell, D. B. and Knowles, J. D. (2006). The Role of Modeling in Systems Biology. In *System Modeling in Cellular Biology*. The MIT Press.
- Kiran, M., Richmond, P., Holcombe, M., Chin, L. S., Worth, D., and Greenough, C. (2010). FLAME: Simulating Large Populations of Agents on Parallel Hardware Architectures. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS ’10, page 1633–1636, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
- Kitano, H. (2002a). Computational systems biology. *Nature*, 420(6912):206–210.
- Kitano, H. (2002b). Systems Biology: A Brief Overview. *Science*, 295(5560):1662–1664.

- Kitano, H. (2007). Towards a theory of biological robustness. *Molecular Systems Biology*, 3(1):137.
- Kleinsteuer, N., Dix, D., Rountree, M., Baker, N., Sipes, N., Reif, D., Spencer, R., and Knudsen, T. (2013). A Computational Model Predicting Disruption of Blood Vessel Development. *PLoS Comput Biol*, 9(4):e1002996.
- Koh, C. H., Nagasaki, M., Saito, A., Li, C., Wong, L., and Miyano, S. (2011). MIRACH: efficient model checker for quantitative biological pathway models. *Bioinformatics*, 27(5):734–735.
- Koh, C. H., Palaniappan, S. K., Thiagarajan, P. S., and Wong, L. (2012). Improved statistical model checking methods for pathway analysis. *BMC Bioinformatics*, 13(Suppl 17):S15.
- Kohl, P. and Noble, D. (2009). Systems biology and the virtual physiological human. *Molecular Systems Biology*, 5:292.
- Kondo, S. and Miura, T. (2010). Reaction-Diffusion Model as a Framework for Understanding Biological Pattern Formation. *Science*, 329(5999):1616–1620.
- Konur, S. (2010). A Survey on Temporal Logics. arXiv e-print 1005.3199, Department of Computer Science, University of Liverpool.
- Kor, A.-L. and Bennett, B. (2013). A Hybrid Reasoning Model for “Whole and Part” Cardinal Direction Relations. *Advances in Artificial Intelligence*, 2013:1–20.
- Koymans, R. (1990). Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299.
- Krauss, M., Schaller, S., Borchers, S., Findeisen, R., Lippert, J., and Kuepfer, L. (2012). Integrating Cellular Metabolism into a Multiscale Whole-Body Model. *PLoS Comput Biol*, 8(10):e1002750.
- Kugler, H., Larjo, A., and Harel, D. (2010). Biocharts: a visual formalism for complex biological systems. *Journal of The Royal Society Interface*, 7(48):1015–1024.
- Kurachi, Y. (2014). High Definition Physiology project. <http://hd-physiology.jp/>, Last accessed on: 2014-04-07.
- Kurshan, R. P. and McMillan, K. (1989). A Structural Induction Theorem for Processes. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, PODC ’89, page 239–247, New York, NY, USA. ACM.
- Kwiatkowska, M., Norman, G., and Parker, D. (2007). Stochastic Model Checking. In Bernardo, M. and Hillston, J., editors, *Formal Methods for Performance Evaluation*, number 4486 in Lecture Notes in Computer Science, pages 220–270. Springer Berlin Heidelberg.
- Kwiatkowska, M., Norman, G., and Parker, D. (2008). Using Probabilistic Model Checking in Systems Biology. *SIGMETRICS Perform. Eval. Rev.*, 35(4):14–21.
- Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of Probabilistic Real-Time Systems. In Gopalakrishnan, G. and Qadeer, S., editors, *Computer Aided Verification*, number 6806 in Lecture Notes in Computer Science, pages 585–591. Springer Berlin Heidelberg, Snowbird, UT, USA.
- Laganà, K., Balossino, R., Migliavacca, F., Pennati, G., Bove, E. L., de Leval, M. R., and Dubini, G. (2005). Multiscale modeling of the cardiovascular system: application to the study of pulmonary and coronary perfusions in the

- univentricular circulation. *Journal of Biomechanics*, 38(5):1129–1141.
- Lakin, M. R., Parker, D., Cardelli, L., Kwiatkowska, M., and Phillips, A. (2012). Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of The Royal Society Interface*, page rsif20110800.
- Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., Funke, R., Gage, D., Harris, K., Heaford, A., Howland, J., Kann, L., Lehoczky, J., LeVine, R., McEwan, P., McKernan, K., Meldrim, J., Mesirov, J. P., Miranda, C., Morris, W., Naylor, J., Raymond, C., Rosetti, M., Santos, R., Sheridan, A., Sougnez, C., Stange-Thomann, N., Stojanovic, N., Subramanian, A., Wyman, D., Rogers, J., Sulston, J., Ainscough, R., Beck, S., Bentley, D., Burton, J., Clee, C., Carter, N., Coulson, A., Deadman, R., Deloukas, P., Dunham, A., Dunham, I., Durbin, R., French, L., Grafham, D., Gregory, S., Hubbard, T., Humphray, S., Hunt, A., Jones, M., Lloyd, C., McMurray, A., Matthews, L., Mercer, S., Milne, S., Mullikin, J. C., Mungall, A., Plumb, R., Ross, M., Shownkeen, R., Sims, S., Waterston, R. H., Wilson, R. K., Hillier, L. W., McPherson, J. D., Marra, M. A., Mardis, E. R., Fulton, L. A., Chinwalla, A. T., Pepin, K. H., Gish, W. R., Chissoe, S. L., Wendl, M. C., Delehaunty, K. D., Miner, T. L., Delehaunty, A., Kramer, J. B., Cook, L. L., Fulton, R. S., Johnson, D. L., Minx, P. J., Clifton, S. W., Hawkins, T., Branscomb, E., Predki, P., Richardson, P., Wenning, S., Slezak, T., Doggett, N., Cheng, J.-F., Olsen, A., Lucas, S., Elkin, C., Uberbacher, E., Frazier, M., Gibbs, R. A., Muzny, D. M., Scherer, S. E., Bouck, J. B., Sodergren, E. J., Worley, K. C., Rives, C. M., Gorrell, J. H., Metzker, M. L., Naylor, S. L., Kucherlapati, R. S., Nelson, D. L., Weinstock, G. M., Sakaki, Y., Fujiyama, A., Hattori, M., Yada, T., Toyoda, A., Itoh, T., Kawagoe, C., Watanabe, H., Totoki, Y., Taylor, T., Weissenbach, J., Heilig, R., Saurin, W., Artiguenave, F., Brottier, P., Bruls, T., Pelletier, E., Robert, C., Wincker, P., Rosenthal, A., Platzer, M., Nyakatura, G., Taudien, S., Rump, A., Smith, D. R., Doucette-Stamm, L., Rubenfield, M., Weinstock, K., Lee, H. M., Dubois, J., Yang, H., Yu, J., Wang, J., Huang, G., Gu, J., Hood, L., Rowen, L., Madan, A., Qin, S., Davis, R. W., Federspiel, N. A., Abola, A. P., Proctor, M. J., Roe, B. A., Chen, F., Pan, H., Ramser, J., Lehrach, H., Reinhardt, R., McCombie, W. R., Bastide, M. d. l., Dedhia, N., Blöcker, H., Hornischer, K., Nordsiek, G., Agarwala, R., Aravind, L., Bailey, J. A., Bateman, A., Batzoglou, S., Birney, E., Bork, P., Brown, D. G., Burge, C. B., Cerutti, L., Chen, H.-C., Church, D., Clamp, M., Copley, R. R., Doerks, T., Eddy, S. R., Eichler, E. E., Furey, T. S., Galagan, J., Gilbert, J. G. R., Harmon, C., Hayashizaki, Y., Haussler, D., Hermjakob, H., Hokamp, K., Jang, W., Johnson, L. S., Jones, T. A., Kasif, S., Kaspryzk, A., Kennedy, S., Kent, W. J., Kitts, P., Koonin, E. V., Korf, I., Kulp, D., Lancet, D., Lowe, T. M., McLysaght, A., Mikkelsen, T., Moran, J. V., Mulder, N., Pollara, V. J., Ponting, C. P., Schuler, G., Schultz, J., Slater, G., Smit, A. F. A., Stupka, E., Szustakowski, J., Thierry-Mieg, D., Thierry-Mieg, J., Wagner, L., Wallis, J., Wheeler, R., Williams, A., Wolf, Y. I., Wolfe, K. H., Yang, S.-P., Yeh, R.-F., Collins, F., Guyer, M. S., Peterson, J., Felsenfeld, A., Wetterstrand, K. A., Myers, R. M., Schmutz, J., Dickson, M., Grimwood, J., Cox, D. R., Olson, M. V., Kaul, R., Raymond, C., Shimizu, N., Kawasaki, K., Minoshima, S., Evans, G. A., Athanasiou, M., Schultz, R.,

- Patrinos, A., and Morgan, M. J. (2001). Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921.
- Langmead, C. (2009). Generalized Queries and Bayesian Statistical Model Checking in Dynamic Bayesian Networks: Application to Personalized Medicine. In *Proc. of the 8th International Conference on Computational Systems Bioinformatics (CSB)*, pages 201–212, California. Life Sciences Society.
- Lee, S. K., Chou, H., Ham, T. S., Lee, T. S., and Keasling, J. D. (2008). Metabolic engineering of microorganisms for biofuels production: from bugs to synthetic biology to fuels. *Current Opinion in Biotechnology*, 19(6):556–563.
- Legay, A., Delahaye, B., and Bensalem, S. (2010). Statistical Model Checking: An Overview. In Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., and Tillmann, N., editors, *Runtime Verification*, number 6418 in Lecture Notes in Computer Science, pages 122–135. Springer Berlin Heidelberg.
- Lewis, H. (1990). A logic of concrete time intervals. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, 1990*, pages 380–389.
- Li, C., Nagasaki, M., Koh, C. H., and Miyano, S. (2011). Online model checking approach based parameter estimation to a neuronal fate decision simulation model in *Caenorhabditis elegans* with hybrid functional Petri net with extension. *Molecular Biosystems*, 7(5):1576–1592.
- Li, C., Nagasaki, M., Ueno, K., and Miyano, S. (2009). Simulation-based model checking approach to cell fate specification during *Caenorhabditis elegans* vulval development by hybrid functional Petri net with extension. *BMC Systems Biology*, 3(1):42.
- Lichtenstein, O. and Pnueli, A. (1985). Checking That Finite State Concurrent Programs Satisfy Their Linear Specification. In *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL '85*, pages 97–107, New York, NY, USA. ACM.
- Lieschke, G. J. and Currie, P. D. (2007). Animal models of human disease: zebrafish swim into view. *Nature Reviews Genetics*, 8(5):353–367.
- Liu, B., Hagiescu, A., Palaniappan, S. K., Chattopadhyay, B., Cui, Z., Wong, W.-F., and Thiagarajan, P. S. (2012). Approximate probabilistic analysis of biopathway dynamics. *Bioinformatics*, 28(11):1508–1516.
- Liu, B., Kong, S., Gao, S., Zuliani, P., and Clarke, E. M. (2014a). Parameter Synthesis for Cardiac Cell Hybrid Models Using δ -Decisions. In Mendes, P., Dada, J. O., and Smallbone, K., editors, *Computational Methods in Systems Biology*, number 8859 in Lecture Notes in Computer Science, pages 99–113. Springer International Publishing.
- Liu, B., Kong, S., Gao, S., Zuliani, P., and Clarke, E. M. (2015). Towards Personalized Prostate Cancer Therapy Using Delta-reachability Analysis. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC '15*, pages 227–232, New York, NY, USA. ACM.
- Liu, B., Thiagarajan, P. S., and Hsu, D. (2009a). Probabilistic Approximations of Signaling Pathway Dynamics. In Degano, P. and Gorrieri, R., editors, *Computational Methods in Systems Biology*, number 5688 in Lecture Notes in Computer Science, pages 251–265. Springer Berlin Heidelberg.
- Liu, F., Blätke, M.-A., Heiner, M., and Yang, M. (2014b). Modelling and

- simulating reaction–diffusion systems using coloured Petri nets. *Computers in Biology and Medicine*, 53:297–308.
- Liu, F. and Heiner, M. (2013). Multiscale modelling of coupled Ca²⁺ channels using coloured stochastic Petri nets. *IET Systems Biology*, 7(4):106–113.
- Liu, W., Li, S., and Renz, J. (2009b). Combining RCC-8 with Qualitative Direction Calculi: Algorithms and Complexity. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI’09, pages 854–859, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Lloyd, C. M., Halstead, M. D. B., and Nielsen, P. F. (2004). CellML: its future, present and past. *Progress in Biophysics and Molecular Biology*, 85(2–3):433–450.
- Machado, D., Costa, R. S., Rocha, M., Ferreira, E. C., Tidor, B., and Rocha, I. (2011). Modeling formalisms in Systems Biology. *AMB Express*, 1(1):1–14.
- Madsen, C., Myers, C., Roehner, N., Winstead, C., and Zhang, Z. (2012). Utilizing stochastic model checking to analyze genetic circuits. In *2012 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 379–386.
- Mallet, D. G. and De Pillis, L. G. (2006). A cellular automata model of tumor–immune system interactions. *Journal of Theoretical Biology*, 239(3):334–350.
- Mancini, T., Tronci, E., Salvo, I., Mari, F., Massini, A., and Melatti, I. (2015). Computing Biological Model Parameters by Parallel Statistical Model Checking. In Ortuño, F. and Rojas, I., editors, *Bioinformatics and Biomedical Engineering*, number 9044 in Lecture Notes in Computer Science, pages 542–554. Springer International Publishing.
- Marée, A. F. M., Grieneisen, V. A., and Hogeweg, P. (2007). The Cellular Potts Model and Biophysical Properties of Cells, Tissues and Morphogenesis. In Anderson, D. A. R. A., Chaplain, P. M. A. J., and Rejniak, D. K. A., editors, *Single-Cell-Based Models in Biology and Medicine*, Mathematics and Biosciences in Interaction, pages 107–136. Birkhäuser Basel.
- Maria, E. D., Fages, F., and Soliman, S. (2009). On Coupling Models Using Model-Checking: Effects of Irinotecan Injections on the Mammalian Cell Cycle. In Degano, P. and Gorrieri, R., editors, *Computational Methods in Systems Biology*, number 5688 in Lecture Notes in Computer Science, pages 142–157. Springer Berlin Heidelberg.
- Markram, H. (2012). The Human Brain Project. *Scientific American*, 306(6):50–55.
- Masoudi-Nejad, A., Bidkhori, G., Hosseini Ashtiani, S., Najafi, A., Bozorgmehr, J. H., and Wang, E. (2014). Cancer systems biology and modeling: Microscopic scale and multiscale approaches. *Seminars in Cancer Biology*.
- Maus, C., Rybacki, S., and Uhrmacher, A. M. (2011). Rule-based multi-level modeling of cell biological systems. *BMC Systems Biology*, 5(1):166.
- McKinsey, J. and Tarski, A. (1944). The Algebra of Topology. *The Annals of Mathematics*, 45(1):141–191.
- Merks, R. M. H. and Glazier, J. A. (2005). A cell-centered approach to developmental biology. *Physica A: Statistical Mechanics and its Applications*, 352(1):113–130.

- Miller, S. P., Whalen, M. W., and Cofer, D. D. (2010). Software model checking takes off. *Communications of the ACM*, 53(2):58–64.
- Mirams, G. R., Arthurs, C. J., Bernabeu, M. O., Bordas, R., Cooper, J., Corrias, A., Davit, Y., Dunn, S.-J., Fletcher, A. G., Harvey, D. G., Marsh, M. E., Osborne, J. M., Pathmanathan, P., Pitt-Francis, J., Southern, J., Zemzemi, N., and Gavaghan, D. J. (2013). Chaste: An Open Source C++ Library for Computational Physiology and Biology. *PLoS Comput Biol*, 9(3):e1002970.
- Miskov-Zivanov, N., Zuliani, P., Clarke, E. M., and Faeder, J. R. (2013). Studies of Biological Networks with Statistical Model Checking: Application to Immune System Cells. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics, '13*, pages 728–729, New York, NY, USA. ACM.
- Mone, G. (2014). New models in cosmetics replacing animal testing. *Communications of the ACM*, 57(4):20–21.
- Montanari, A., Puppis, G., and Sala, P. (2009). A decidable spatial logic with cone-shaped cardinal directions. In *Computer Science Logic*, pages 394–408.
- Monteiro, P. T., Ropers, D., Mateescu, R., Freitas, A. T., and Jong, H. d. (2008). Temporal logic patterns for querying dynamic models of cellular interaction networks. *Bioinformatics*, 24(16):i227–i233.
- Monteiro, P. T., Wassim, A.-J., Thieffry, D., and Chaouiya, C. (2014). Model Checking Logical Regulatory Networks. In *Discrete Event Systems*, volume 12, pages 170–175, Ecole Normale Supérieure de Cachan, Cachan, France. International Federation of Automatic Control.
- Moreira, J. and Deutsch, A. (2002). Cellular automaton models of tumor development: a critical review. *Advances in Complex Systems*, 05(02n03):247–267.
- Nenzi, L. (2014). Verification of stochastic and spatial behaviours of Complex Systems. In *11th Summer School on Modelling and Verification of Parallel Processes*, Nantes, France.
- Nenzi, L. and Bortolussi, L. (2014). Specifying and Monitoring Properties of Stochastic Spatio-Temporal Systems in Signal Temporal Logic. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2014)*, pages 66–73, Bratislava, Slovakia. ICST.
- Nikolić, D., Priami, C., and Zunino, R. (2012). A Rule-Based and Imperative Language for Biochemical Modeling and Simulation. In Eleftherakis, G., Hinckey, M., and Holcombe, M., editors, *Software Engineering and Formal Methods*, number 7504 in Lecture Notes in Computer Science, pages 16–32. Springer Berlin Heidelberg.
- Noble, D. (1960). Cardiac Action and Pacemaker Potentials based on the Hodgkin-Huxley Equations. *Nature*, 188(4749):495–497.
- Noble, D. (2008). Claude Bernard, the first systems biologist, and the future of physiology. *Experimental Physiology*, 93(1):16–26.
- Norris, J. R. (1998). *Markov Chains*. Cambridge University Press.
- Norton, K.-A. and Popel, A. S. (2014). An agent-based model of cancer stem cell initiated avascular tumour growth and metastasis: the effect of seeding frequency and location. *Journal of The Royal Society Interface*, 11(100):20140640.
- Novère, N. L., Hucka, M., Mi, H., Moodie, S., Schreiber, F., Sorokin, A., Demir, E., Wegner, K., Aladjem, M. I., Wimalaratne, S. M., Bergman, F. T., Gauges,

- R., Ghazal, P., Kawaji, H., Li, L., Matsuoka, Y., Villéger, A., Boyd, S. E., Calzone, L., Courtot, M., Dogrusoz, U., Freeman, T. C., Funahashi, A., Ghosh, S., Jouraku, A., Kim, S., Kolpakov, F., Luna, A., Sahle, S., Schmidt, E., Watterson, S., Wu, G., Goryanin, I., Kell, D. B., Sander, C., Sauro, H., Snoep, J. L., Kohn, K., and Kitano, H. (2009). The Systems Biology Graphical Notation. *Nature Biotechnology*, 27(8):735–741.
- Øhrstrøm, P. and Hasle, P. F. V. (1995). *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Springer Science & Business Media.
- O'Rourke, J., Aggarwal, A., Maddila, S., and Baldwin, M. (1986). An optimal algorithm for finding minimal enclosing triangles. *Journal of Algorithms*, 7(2):258–269.
- Orth, J. D., Thiele, I., and Palsson, B. Ø. (2010). What is flux balance analysis? *Nature Biotechnology*, 28(3):245–248.
- Oury, N. and Plotkin, G. D. (2011). Coloured Stochastic Multilevel Multiset Rewriting. In *Proceedings of the 9th International Conference on Computational Methods in Systems Biology*, CMSB '11, pages 171–181, New York, NY, USA. ACM.
- Palaniappan, S. K., Gyori, B. M., Liu, B., Hsu, D., and Thiagarajan, P. S. (2013). Statistical Model Checking Based Calibration and Analysis of Bio-pathway Models. In Gupta, A. and Henzinger, T. A., editors, *Computational Methods in Systems Biology*, number 8130 in Lecture Notes in Computer Science, pages 120–134. Springer Berlin Heidelberg.
- Pandey, U. B. and Nichols, C. D. (2011). Human Disease Models in *Drosophila melanogaster* and the Role of the Fly in Therapeutic Drug Discovery. *Pharmacological Reviews*, 63(2):411–436.
- Pârvu, O. and Gilbert, D. (2014a). Automatic validation of computational models using pseudo-3D spatio-temporal model checking. *BMC Systems Biology*, 8(1):124.
- Pârvu, O. and Gilbert, D. (2014b). Implementation of linear minimum area enclosing triangle algorithm. *Computational and Applied Mathematics*, pages 1–16.
- Pârvu, O. and Gilbert, D. (submitted). A novel method to validate multilevel computational models of biological systems using multiscale spatio-temporal meta model checking. *PLoS ONE*.
- Pârvu, O., Gilbert, D., Heiner, M., Liu, F., and Saunders, N. (2013). Modelling and Analysis of Phase Variation in Bacterial Colony Growth. In Gupta, A. and Henzinger, T. A., editors, *Computational Methods in Systems Biology*, number 8130 in LNCS, pages 78–91. Springer Berlin Heidelberg.
- Pârvu, O., Gilbert, D., Heiner, M., Liu, F., Saunders, N., and Shaw, S. (2015). Spatial-Temporal Modelling and Analysis of Bacterial Colonies with Phase Variable Genes. *ACM Trans. Model. Comput. Simul.*, 25(2):13:1–13:25.
- Peled, D. (1994). Combining partial order reductions with on-the-fly model-checking. In Dill, D. L., editor, *Computer Aided Verification*, number 818 in Lecture Notes in Computer Science, pages 377–390. Springer Berlin Heidelberg.
- Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA.
- Petri, C. A. (1962). *Kommunikation mit Automaten*. PhD thesis, Universität

- Hamburg.
- Plumejeaud, C., Mathian, H., Gensel, J., and Grasland, C. (2011). Spatio-temporal analysis of territorial changes from a multi-scale perspective. *International Journal of Geographical Information Science*, 25(10):1597–1612.
- Pnueli, A. (1977). The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Providence, RI, USA. IEEE.
- Prior, A. (1967). *Past, Present and Future*. Oxford University Press.
- Queille, J. P. and Sifakis, J. (1982). Specification and verification of concurrent systems in CESAR. In Dezani-Ciancaglini, M. and Montanari, U., editors, *International Symposium on Programming*, number 137 in Lecture Notes in Computer Science, pages 337–351. Springer Berlin Heidelberg.
- Rafe, V., Rahmani, M., and Rashidi, K. (2013). A Survey on Coping with the State Space Explosion Problem in Model Checking. *International Research Journal of Applied and Basic Sciences*, 4(6):1379–1384.
- Randell, D. A., Cui, Z., and Cohn, A. G. (1992). A Spatial Logic based on Regions and Connection. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 165–176, Cambridge, MA.
- Reijsbergen, D., Boer, P.-T. d., Scheinhardt, W., and Haverkort, B. (2014). On hypothesis testing for statistical model checking. *International Journal on Software Tools for Technology Transfer*, pages 1–19.
- Reijsbergen, D., Boer, P.-T. d., Scheinhardt, W., and Haverkort, B. (2015). Hypothesis testing for statistical model checking. <http://wwwhome.ewi.utwente.nl/~ptdeboer/hyptest-for-smc/>, Last accessed on: 2015-02-02.
- Rizk, A., Batt, G., Fages, F., and Soliman, S. (2008). On a Continuous Degree of Satisfaction of Temporal Logic Formulae with Applications to Systems Biology. In Heiner, M. and Uhrmacher, A. M., editors, *Computational Methods in Systems Biology*, number 5307 in Lecture Notes in Computer Science, pages 251–268. Springer Berlin Heidelberg.
- Rizk, A., Batt, G., Fages, F., and Soliman, S. (2009). A general computational method for robustness analysis with applications to synthetic gene networks. *Bioinformatics*, 25(12):i169–i178.
- Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.
- Ruder, W. C., Lu, T., and Collins, J. J. (2011). Synthetic Biology Moving into the Clinic. *Science*, 333(6047):1248–1252.
- Ruf, J. and Kropf, T. (1997). Symbolic Model Checking for a Discrete Clocked Temporal Logic with Intervals. In *Proceedings of the IFIP WG 10.5 International Conference on Correct Hardware Design and Verification Methods: Advances in Hardware Design and Verification*, pages 146–163, London, UK. Chapman & Hall, Ltd.
- Salaün, L., Ayraud, S., and Saunders, N. J. (2005). Phase variation mediated niche adaptation during prolonged experimental murine infection with Helicobacter pylori. *Microbiology (Reading, England)*, 151(Pt 3):917–923.
- Salaün, L., Snyder, L. A., and Saunders, N. J. (2003). Adaptation by phase variation in pathogenic bacteria. *Advances in applied microbiology*, 52:263–301.

- Saunders, N. J., Moxon, E. R., and Gravenor, M. B. (2003). Mutation rates: estimating phase variation rates when fitness differences are present and their impact on population structure. *Microbiology*, 149(2):485–495.
- Schaff, J., Fink, C. C., Slepchenko, B., Carson, J. H., and Loew, L. M. (1997). A general computational framework for modeling cellular structure and function. *Biophysical Journal*, 73(3):1135–1146.
- Schivo, S., Scholma, J., Wanders, B., Camacho, R., van der Vet, P., Karperien, M., Langerak, R., van de Pol, J., and Post, J. (2012). Modelling biological pathway dynamics with Timed Automata. In *2012 IEEE 12th International Conference on Bioinformatics Bioengineering (BIBE)*, pages 447–453.
- Schnell, S., Grima, R., and Maini, P. (2007). Multiscale Modeling in Biology. *American Scientist*, 95(2):134.
- Scott, A., Khan, K. M., Cook, J. L., and Duronio, V. (2004). What is “inflammation”? Are we ready to move beyond Celsius? *British Journal of Sports Medicine*, 38(3):248–249.
- Selick, H. E., Beresford, A. P., and Tarbit, M. H. (2002). The emerging importance of predictive ADME simulation in drug discovery. *Drug Discovery Today*, 7(2):109–116.
- Sen, K., Viswanathan, M., and Agha, G. (2004). Statistical Model Checking of Black-Box Probabilistic Systems. In Alur, R. and Peled, D. A., editors, *Computer Aided Verification*, number 3114 in Lecture Notes in Computer Science, pages 202–215. Springer Berlin Heidelberg.
- Sheard, T. (2001). Accomplishments and Research Challenges in Metaprogramming. In Taha, W., editor, *Semantics, Applications, and Implementation of Program Generation*, number 2196 in Lecture Notes in Computer Science, pages 2–44. Springer Berlin Heidelberg.
- Siebert, H. and Bockmayr, A. (2006). Incorporating Time Delays into the Logical Analysis of Gene Regulatory Networks. In Priami, C., editor, *Computational Methods in Systems Biology*, number 4210 in Lecture Notes in Computer Science, pages 169–183. Springer Berlin Heidelberg.
- Sloot, P. M. A. and Hoekstra, A. G. (2010). Multi-scale modelling in computational biomedicine. *Briefings in Bioinformatics*, 11(1):142–152.
- Smith, B. W., Chase, J. G., Nokes, R. I., Shaw, G. M., and Wake, G. (2004). Minimal haemodynamic system model including ventricular interaction and valve dynamics. *Medical Engineering & Physics*, 26(2):131–139.
- Snyder, A. (1986). Encapsulation and Inheritance in Object-oriented Programming Languages. In *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications*, OOPSLA '86, pages 38–45, New York, NY, USA. ACM.
- Southern, J., Pitt-Francis, J., Whiteley, J., Stokeley, D., Kobashi, H., Nobes, R., Kadooka, Y., and Gavaghan, D. (2008). Multi-scale computational modelling in biology and physiology. *Progress in Biophysics and Molecular Biology*, 96(1–3):60–89.
- Starruß, J. and Back, W. d. (2014). Morpheus examples. <http://imc.zih.tu-dresden.de/wiki/morpheus/doku.php?id=examples:examples>, Last accessed on: 2014-11-20.
- Starruß, J., Back, W. d., Brusch, L., and Deutsch, A. (2014). Morpheus: a

- user-friendly modeling environment for multiscale and multicellular systems biology. *Bioinformatics*, 30(9):1331–1332.
- Steger, C. (1996). On the Calculation of Moments of Polygons. Technical Report FGBV-96-04, Forschungsgruppe Bildverstehen (FG BV), Informatik IX, Technische Universität München.
- Su, X., He, C., Feng, Q., Deng, X., and Sun, H. (2011). A Supervised Classification Method Based on Conditional Random Fields With Multiscale Region Connection Calculus Model for SAR Image. *IEEE Geoscience and Remote Sensing Letters*, 8(3):497–501.
- Swat, M. H., Thomas, G. L., Belmonte, J. M., Shirinifard, A., Hmeljak, D., and Glazier, J. A. (2012). Multi-Scale Modeling of Tissues Using CompuCell3D. In Anand R. Asthagiri and Adam P. Arkin, editor, *Methods in Cell Biology*, volume 110 of *Computational Methods in Cell Biology*, pages 325–366. Academic Press.
- Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition.
- Tarski, A. (1938). Sentential calculus and topology. (Der Aussagenkalkül und die Topologie.). *Fundam. Math.*, Warszawa,, 31:103–134.
- Thiel, W. and Hummer, G. (2013). Nobel 2013 Chemistry: Methods for computational chemistry. *Nature*, 504(7478):96–97.
- Thorne, B. C., Bailey, A. M., and Peirce, S. M. (2007). Combining experiments with multi-cell agent-based modeling to study biological tissue patterning. *Briefings in Bioinformatics*, 8(4):245–257.
- Toussaint, G. T. (1983). Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon*, volume 83.
- Tran, T. N., Drab, K., and Daszykowski, M. (2013). Revised DBSCAN algorithm to cluster data with dense adjacent clusters. *Chemometrics and Intelligent Laboratory Systems*, 120:92–96.
- Valmari, A. (1991). A stubborn attack on state explosion. In Clarke, E. M. and Kurshan, R. P., editors, *Computer Aided Verification*, number 531 in Lecture Notes in Computer Science, pages 156–165. Springer Berlin Heidelberg.
- Van Goethem, S., Jacquet, J. M., Brim, L., and Šafránek, D. (2013). Timed Modelling of Gene Networks with Arbitrarily Precise Expression Discretization. *Electronic Notes in Theoretical Computer Science*, 293:67–81.
- Vardi, M. Y. and Wolper, P. (1986). An Automata-Theoretic Approach to Automatic Program Verification. In *Proc. 1st Symp. on Logic in Computer Science*, page 332–344, Cambridge.
- Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A., Gocayne, J. D., Amanatides, P., Ballew, R. M., Huson, D. H., Wortman, J. R., Zhang, Q., Kodira, C. D., Zheng, X. H., Chen, L., Skupski, M., Subramanian, G., Thomas, P. D., Zhang, J., Miklos, G. L. G., Nelson, C., Broder, S., Clark, A. G., Nadeau, J., McKusick, V. A., Zinder, N., Levine, A. J., Roberts, R. J., Simon, M., Slayman, C., Hunkapiller, M., Bolanos, R., Delcher, A., Dew, I., Fasulo, D., Flanigan, M., Florea, L., Halpern, A., Hannenhalli, S., Kravitz, S., Levy, S., Mobarry, C., Reinert, K., Remington, K., Abu-Threideh, J., Beasley, E., Biddick, K., Bonazzi, V., Brandon, R., Cargill, M., Chandramouliswaran, I.,

- Charlab, R., Chaturvedi, K., Deng, Z., Francesco, V. D., Dunn, P., Eilbeck, K., Evangelista, C., Gabrielian, A. E., Gan, W., Ge, W., Gong, F., Gu, Z., Guan, P., Heiman, T. J., Higgins, M. E., Ji, R.-R., Ke, Z., Ketchum, K. A., Lai, Z., Lei, Y., Li, Z., Li, J., Liang, Y., Lin, X., Lu, F., Merkulov, G. V., Milshina, N., Moore, H. M., Naik, A. K., Narayan, V. A., Neelam, B., Nusskern, D., Rusch, D. B., Salzberg, S., Shao, W., Shue, B., Sun, J., Wang, Z. Y., Wang, A., Wang, X., Wang, J., Wei, M.-H., Wides, R., Xiao, C., Yan, C., Yao, A., Ye, J., Zhan, M., Zhang, W., Zhang, H., Zhao, Q., Zheng, L., Zhong, F., Zhong, W., Zhu, S. C., Zhao, S., Gilbert, D., Baumhueter, S., Spier, G., Carter, C., Cravchik, A., Woodage, T., Ali, F., An, H., Awe, A., Baldwin, D., Baden, H., Barnstead, M., Barrow, I., Beeson, K., Busam, D., Carver, A., Center, A., Cheng, M. L., Curry, L., Danaher, S., Davenport, L., Desilets, R., Dietz, S., Dodson, K., Doup, L., Ferriera, S., Garg, N., Gluecksmann, A., Hart, B., Haynes, J., Haynes, C., Heiner, C., Hladun, S., Hostin, D., Houck, J., Howland, T., Ibegwam, C., Johnson, J., Kalush, F., Kline, L., Koduru, S., Love, A., Mann, F., May, D., McCawley, S., McIntosh, T., McMullen, I., Moy, M., Moy, L., Murphy, B., Nelson, K., Pfannkoch, C., Pratts, E., Puri, V., Qureshi, H., Reardon, M., Rodriguez, R., Rogers, Y.-H., Romblad, D., Ruhfel, B., Scott, R., Sitter, C., Smallwood, M., Stewart, E., Strong, R., Suh, E., Thomas, R., Tint, N. N., Tse, S., Vech, C., Wang, G., Wetter, J., Williams, S., Williams, M., Windsor, S., Winn-Deen, E., Wolfe, K., Zaveri, J., Zaveri, K., Abril, J. F., Guigó, R., Campbell, M. J., Sjolander, K. V., Karlak, B., Kejariwal, A., Mi, H., Lazareva, B., Hatton, T., Narechania, A., Diemer, K., Muruganujan, A., Guo, N., Sato, S., Bafna, V., Istrail, S., Lippert, R., Schwartz, R., Walenz, B., Yooseph, S., Allen, D., Basu, A., Baxendale, J., Blick, L., Caminha, M., Carnes-Stine, J., Caulk, P., Chiang, Y.-H., Coyne, M., Dahlke, C., Mays, A. D., Dombroski, M., Donnelly, M., Ely, D., Esparham, S., Fosler, C., Gire, H., Glanowski, S., Glasser, K., Glodek, A., Gorokhov, M., Graham, K., Gropman, B., Harris, M., Heil, J., Henderson, S., Hoover, J., Jennings, D., Jordan, C., Jordan, J., Kasha, J., Kagan, L., Kraft, C., Levitsky, A., Lewis, M., Liu, X., Lopez, J., Ma, D., Majoros, W., McDaniel, J., Murphy, S., Newman, M., Nguyen, T., Nguyen, N., Nodell, M., Pan, S., Peck, J., Peterson, M., Rowe, W., Sanders, R., Scott, J., Simpson, M., Smith, T., Sprague, A., Stockwell, T., Turner, R., Venter, E., Wang, M., Wen, M., Wu, D., Wu, M., Xia, A., Zandieh, A., and Zhu, X. (2001). The Sequence of the Human Genome. *Science*, 291(5507):1304–1351.
- Visser, W. and Barringer, H. (2000). Practical CTL* model checking: Should SPIN be extended? *International Journal on Software Tools for Technology Transfer*, 2(4):350–365.
- Visser, W., Barringer, H., Fellows, D., Gough, G., and Williams, A. (1997). Efficient CTL* Model Checking for Analysis of Rainbow Designs. In *Proceedings of the IFIP WG 10.5 International Conference on Correct Hardware Design and Verification Methods: Advances in Hardware Design and Verification*, pages 128–145, London, UK. Chapman & Hall, Ltd.
- Wald, A. (1945). Sequential Tests of Statistical Hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186.
- Walpole, R. E., Papin, J. A., and Peirce, S. M. (2013). Multiscale Computational Models of Complex Biological Systems. *Annual Review of Biomedical Engineering*,

- 15(1):137–154.
- Weber, W. and Fussenegger, M. (2012). Emerging biomedical applications of synthetic biology. *Nature Reviews Genetics*, 13(1):21–35.
- Wilensky, U. (2015). NetLogo. <https://ccl.northwestern.edu/netlogo/>, Last accessed on: 2015-04-14.
- Wilkinson, D. J. (2007). Bayesian methods in bioinformatics and computational systems biology. *Briefings in Bioinformatics*, 8(2):109–116.
- Williams, R. S. B., Boeckeler, K., Gräf, R., Müller-Taubenberger, A., Li, Z., Isberg, R. R., Wessels, D., Soll, D. R., Alexander, H., and Alexander, S. (2006). Towards a molecular understanding of human diseases using *Dictyostelium discoideum*. *Trends in Molecular Medicine*, 12(9):415–424.
- Xu, J. (2007). Formalizing natural-language spatial relations between linear objects with topological and metric properties. *International Journal of Geographical Information Science*, 21(4):377–395.
- Xu, Z., Lioi, J., Mu, J., Kamocka, M. M., Liu, X., Chen, D. Z., Rosen, E. D., and Alber, M. (2010). A Multiscale Model of Venous Thrombus Formation with Surface-Mediated Control of Blood Coagulation Cascade. *Biophysical Journal*, 98(9):1723–1732.
- Yang, A. (2013). On the Common Conceptual and Computational Frameworks for Multiscale Modeling. *Industrial & Engineering Chemistry Research*, 52(33):11451–11462.
- Yordanov, B. and Belta, C. (2011). A formal verification approach to the design of synthetic gene networks. In *2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 4873–4878.
- Younes, H. L. and Simmons, R. G. (2006). Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 204(9):1368–1409.
- Younes, H. L. S. (2005a). Probabilistic Verification for “Black-Box” Systems. In Etessami, K. and Rajamani, S. K., editors, *Computer Aided Verification*, number 3576 in Lecture Notes in Computer Science, pages 253–265. Springer Berlin Heidelberg.
- Younes, H. L. S. (2005b). *Verification and Planning for Stochastic Processes with Asynchronous Events*. Doctor of philosophy, Carnegie Mellon, Pittsburgh.
- Younes, H. L. S. (2005c). Ymer: A Statistical Model Checker. In Etessami, K. and Rajamani, S. K., editors, *Computer Aided Verification*, number 3576 in Lecture Notes in Computer Science, pages 429–433. Springer Berlin Heidelberg.
- Younes, H. L. S., Kwiatkowska, M., Norman, G., and Parker, D. (2006). Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8(3):216–228.
- Younes, H. L. S. and Simmons, R. G. (2002). Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling. In Brinksma, E. and Larsen, K. G., editors, *Computer Aided Verification*, number 2404 in Lecture Notes in Computer Science, pages 223–235. Springer Berlin Heidelberg.
- Young, R. C. and Barendse, P. (2014). Linking Myometrial Physiology to Intrauterine Pressure; How Tissue-Level Contractions Create Uterine Contractions of Labor. *PLoS Comput Biol*, 10(10):e1003850.
- Zuliani, P. (2014). Statistical model checking for biological applications. *Interna-*

- tional Journal on Software Tools for Technology Transfer, pages 1–10.
- Zuliani, P., Platzer, A., and Clarke, E. M. (2010). Bayesian statistical model checking with application to Simulink/Stateflow verification. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, HSCC ’10, pages 243–252, New York, NY, USA. ACM.