OULU 2010
A 560

ACTA UNIVERSITATIS OULUENSIS

Antti Siirtola

ALGORITHMIC MULTIPARAMETERISED VERIFICATION OF SAFETY PROPERTIES

PROCESS ALGEBRAIC APPROACH

FACULTY OF SCIENCE,
DEPARTMENT OF INFORMATION PROCESSING SCIENCE,
UNIVERSITY OF OULU;
INFOTECH OULU,
UNIVERSITY OF OULU

ACTA UNIVERSITATIS OULUENSIS
A Scientiae Rerum Naturalium 560

ANTTI SIIRTOLA

ALGORITHMIC MULTIPARAMETERISED VERIFICATION OF SAFETY PROPERTIES

Process algebraic approach

Academic dissertation to be presented with the assent of the Faculty of Science of the University of Oulu for public defence in OP-sali (Auditorium L10), Linnanmaa, on 8 October 2010, at 12 noon

UNIVERSITY OF OULU, OULU 2010

Acta Univ. Oul. A 560, 2010

Supervised by
Docent Juha Kortelainen
Professor Antti Valmari

Reviewed by
Professor Ranko Lazić
Doctor Antti Puhakka

ISBN 978-951-42-6251-7 (Paperback)
ISBN 978-951-42-6252-4 (PDF)
http://herkules.oulu.fi/isbn978951426252 4/
ISSN 0355-3191 (Printed)
ISSN 1796-220X (Online)
http://herkules.oulu.fi/issn03553191/

Cover design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2010

Siirtola, Antti, Algorithmic multiparameterised verification of safety properties. Process algebraic approach

Faculty of Science, Department of Information Processing Science, University of Oulu, P.O.Box 3000, FI-90014 University of Oulu, Finland; Infotech Oulu, University of Oulu, P.O.Box 4500, FI-90014 University of Oulu, Finland

Acta Univ. Oul. A 560, 2010
Oulu, Finland

Abstract

Due to increasing amount of concurrency, systems have become difficult to design and analyse. In this effort, formal verification, which means proving the correctness of a system, has turned out to be useful. Unfortunately, the application domain of the formal verification methods is often indefinite, tools are typically unavailable, and most of the techniques do not suit especially well for the verification of software systems. These are the questions addressed in the thesis.

A typical approach to modelling systems and specifications is to consider them parameterised by the restrictions of the execution environment, which results in an (infinite) family of finite-state verification tasks. The thesis introduces a novel approach to the verification of such infinite specification-system families represented as labelled transition systems (LTSs). The key idea is to exploit the algebraic properties of the correctness relation. They allow the correctness of large system instances to be derived from that of smaller ones and, in the best case, an infinite family of finite-state verification tasks to be reduced to a finite one, which can then be solved using existing tools.

The main contribution of the thesis is an algorithm that automates the reduction method. A specification and a system are given as parameterised LTSs and the allowed parameter values are encoded using first order logic. Parameters are sets and relations over these sets, which are typically used to denote, respectively, identities of replicated components and relationships between them. Because the number of parameters is not limited and they can be nested as well, one can express multiply parameterised systems with a parameterised substructure, which is an essential property from the viewpoint of modelling software systems. The algorithm terminates on all inputs, so its application domain is explicit in this sense. Other proposed parameterised verification methods do not have both these features. Moreover, some of the earlier results on the verification of parameterised systems are obtained as a special case of the results presented here.

Finally, several natural and significant extensions to the formalism are considered, and it is shown that the problem becomes undecidable in each of the cases. Therefore, the algorithm cannot be significantly extended in any direction without simultaneously restricting some other aspect.

Keywords: cut-off, decidability, formal verification, parameterised verification, process algebra, refinement, trace semantics, undecidability

Preface

"You gotta roll with it
You gotta take your time
You gotta say what you say
Don't let anybody get in your way
'Cause it's all too much for me to take",
Noel Gallagher in the opening lines of Oasis' single Roll with it, 1995.

As long as I remember, I have been interested in mathematics. Long before school age, I used to count numbers from one to a hundred with my grandmother and later on, I set mathematics exams for my parents.

Somewhere between primary and secondary school, I got acquainted with programming when a Commodore VIC-20 computer entered our home. VIC was already outdated at the time we got it and one could not buy programmes for it any longer. Hence, the only way to get new games for it was to write them myself.

Programming became a passion of mine. It appealed to me because I could basically do all the same things as rich international software companies did, all I needed was a single computer. Additionally, there was something similar to mathematics in it. I recall that already as a teenager, I was thinking about the possibility of combining mathematics and programming in the form of mathematical analysis of computer programmes.

When I enrolled in the University of Oulu as an information engineering student, I heard of formal methods in a course on software engineering and learned some very basics of discrete mathematics. I realised that this is what I really wanted to learn. I registered as a mathematics student as well and just about passed all the courses on related subjects: mathematical logic, automata theory, formal languages and abstract algebra. However, these were courses on mathematics, not on computer science, so after passing them I still had no idea of formal software development.

When it was time to start to work on master's thesis, I thought that this is probably my last chance to learn something about the mathematical analysis of computer programmes. Unfortunately, no professor in the department was willing to supervise a thesis on the subject. I also contacted an expert in the field, professor Antti Valmari in Tampere, but he did not have funding for hiring new people at that moment.

Luckily, an acquaintance of mine, Jouni Kokkoniemi, working in the department of information processing science knew his colleague, professor Juha Kortelainen, to be active in the field of theoretical computer science. I contacted Juha and he suggested me a topic on formal verification which was something I had been looking for. He also provided me with an assistant's post in the department.

At first, formal verification techniques looked to me like a silver bullet that would solve all software reliability problems, but soon I learned the limitations of the finite-state methods I reviewed in my master's thesis. The fact presupposed by the methods that an application should be modelled as a single finite-state system (or more generally, as finitely many finite-state systems) appeared to be the most fundamental restriction to me.

Having finished my master's thesis on finite-state temporal logic model checking I enrolled as a graduate student in the department of information processing science in 2004. The problem of applying existing finite-state techniques to the verification of real software systems appeared as a favourable topic to me. I was assured that I could contribute something to the field.

I was fortunate to have Antti Valmari as my second supervisor; he pointed out that I was actually working with a question known as the parameterised verification problem. He also recommended me a book on a process algebra which somehow seemed to provide more tools and suit parameterised and software verification better than state-based formalisms reviewed in my master's thesis. However, I made only a little progress with my research problem. I knew that I needed a practical example to work with.

Things started to evolve when our research group was accepted in Infotech Oulu and I received a position in its graduate school in 2006. Later in the same year, professor Michal Valenta from the technical university of Prague visited Oulu and gave a presentation concerning his work on XML databases. At the end of his talk, he proposed that it might be fruitful to apply formal verification techniques to taDOM protocols. I got immediately interested in the idea and me and Michal started our effort to formally model and verify those protocols.

Nevertheless, in the summer 2007, things did not look bright. Our taDOM paper had been rejected twice and in addition to results presented in the paper, I had not been able to formulate even a single reasonable conjecture. It is a shame to admit now but at that time, I was pretty much assured I will never graduate. I think that the only ones that still thought otherwise were Juha and my wife Susanna.

I decided to take a decent summer vacation which I had not had for years. I spent five weeks with Susanna in my parents' summer cottage, Oluthuone Leskinen and North-East Lapland hiking in highlands. As I got back to work in the autumn, I thought that it still might be possible to do some kind of a Ph.D. thesis within the remaining graduate school period of two and a half years. I abandoned the ambitious idea of a general parameterised verification method and started to work with humbler ones. It turned out to be a fruitful decision as within a few weeks I discovered the idea of the verification technique behind this thesis.

Things started to look bright again. I knew that the idea I had could be worked to a Ph.D. thesis. Moreover, the taDOM paper was accepted for presentation in a conference and my wife started to expect a child which later turned out to be twins.

During the next two years, I rewrote the central parts of the thesis just about a dozen times, I quit counting after the fifth iteration. Each time, Juha pointed out places that caused particular difficulties for a reader and each time I thought it is impossible to simplify the representation. Nevertheless, I always came back with an improved version a few weeks later and often managed to generalise the results as well. This two-year period included both lows and highs in the form of paper rejections and acceptances, scholarship awards and the birth of the twins, Otso and Aino.

I made the final revision of the manuscript in the autumn 2009 based on the comments of my namesake. Among many other notions, he pointed out that the first of my undecidability results, which was the only one at that moment, could be strengthened to its current form. It inspired me to pursue the limits of parameterised verification further which lead to the other four undecidability results. I discovered two of them in a drab department meeting and the other two on the way to my colleague's birthday party. I had shown that my results on parametric verification cannot be much improved.

The manuscript of the thesis was finished in the end of 2009. It took a long time to take the idea of the mathematical analysis of computer programmes to this point but I managed to say what I wanted to and I am satisfied with the result.

> "I think I've I got a feeling I've lost inside",
> Noel Gallagher in the closing line of Roll with it.

Oulu, May 26th, 2010
Antti Siirtola

Acknowledgements

The research is carried out at the Department of Information Processing Science in the University of Oulu and enabled by the true academic freedom prevailing there. Thanks to the funding from Infotech Oulu Graduate School, I could concentrate solely on research during four years. I learned a lot of research and did just about all the work for the thesis during that period. The scholarships granted by the Scholarship Foundation of the University of Oulu and the Foundation of Tauno Tönning have not only been a great financial aid for a young family with two children of the same age but also a considerable source of motivation.

I wish to express my sincere gratitude to my supervisors Dr. Juha Kortelainen and professor Antti Valmari. The support and patience of Juha have been invaluable for finishing this thesis. Even though my first texts were mostly rubbish, he never said that but carefully read and commented on them. On the other hand, also the expertise and outspokenness of Antti were needed to keep the project on track, and his excellent contacts were precious especially at the final stage.

I also want to thank my coauthor, professor Michal Valenta, who introduced taDOM protocols to me. It is the attempts to verify these protocols that gave rise to the theory presented in the thesis.

Moreover, I was lucky to have a pair of excellent reviewers, professor Ranko Lazić and Dr. Antti Puhakka. The works of Ranko on data independence inspired me a lot especially at the early phase of the research, and I have never had such an exhaustive review as Antti did nor as felicitous comments as he gave. I greatly appreciate their effort.

Finally, I wish to thank my friends, relatives and colleagues for offering me a chance to break from research occasionally. I am fortunate to have twins, Aino and Otso, who provide my research-filled life with other content. I address my special gratitude to my wife Susanna for the support and understanding showed to my work which often does not respect working hours. I also appreciate the moments when you spare no sympathy for science and bring me down to earth.

The same

Abbreviations

CCS	*Calculus of Communicating Systems (p. 35)*
CSP	*Communicating Sequential Processes (p. 35)*
DTM	*deterministic Turing machine (p. 125)*
FDR	*Failures-Divergences Refinement (p. 52)*
LOTOS	*Language of Temporal Ordering Specification (p. 35)*
LTL-X	*linear time temporal logic without the next-time operator (p. 100)*
LTS	*labelled transition system (p. 31)*
PVP	*the parameterised verification problem (p. 18)*
RTR	*request-take-release (p. 102)*
taDOMn	*extended Domain Object Model level n (p. 122)*
XML	*Extensible Markup Language (p. 123)*
$\mathbb{A}$	*the set of all the atoms (p. 31)*
$\mathbb{G}$	*the set of all the relation variables (p. 58)*
$\mathbb{N}$	*the set of all the non-negative integers (Chap. 2)*
$\mathbb{P}(A)$	*the set of all the subsets of a set A (Chap. 2)*
$\mathbb{T}$	*the set of all the type variables (p. 58)*
$\mathbb{V}$	*the set of all the visible actions (p. 31)*
$\mathbb{X}$	*the set of all the atom variables (p. 58)*
$\mathbb{Z}_+$	*the set of all the positive integers (Chap. 2)*
a, b	*atoms (p. 31)*
c	*a valuation formula (p. 71)*
d	*a tape symbol (p. 125)*
f, g	*functions (from $\mathbb{A}$ to $\mathbb{A}$) (Chap. 2)*
g^*	*the extension of g to tuples (p. 53)*
$g\phi$	*the valuation obtained from a valuation ϕ by mapping its image atom-wise using a bijection g (p. 80)*
i, j, k, l, n	*(non-negative) integers*
o	*an operation (Chap. 2)*
q, s	*states (p. 31, 125)*
$\hat{q}, \hat{s}$	*initial states (p. 31, 125)*
t	*a trace (p. 33)*

v	*a vertex (p. 118)*
w	*a sequence (Chap. 2)*
x, y	*atom variables (p. 58)*
A, B	*sets (of atoms) (p. 31)*
A^*	*the set of all the sequences over A (Chap. 2)*
C	*a set of colours (p. 118)*
D	*a set of tape symbols (p. 125)*
E	*a set of edges (p. 118)*
G	*a relation (Chap. 2)*
I, J, K	*index sets*
L	*a language (Chap. 2)*
O	*a set of operations (Chap. 2)*
R	*a transition relation (p. 31)*
Q, S	*sets of states (p. 31, 125)*
T, U	*type variables (p. 58)*
V	*a set of vertices (p. 118)*
α, β	*actions (p. 31)*
γ	*a colouring function (p. 118)*
ε	*the empty sequence (Chap. 2)*
σ	*a transition function (p. 125)*
ζ	*a function from* $\mathbb{V}$ *to* $\mathbb{V}$ (Chap. 2)
υ	*a partial function from* $\mathbb{T}$ *to* $\mathbb{Z}_+$ *(Chap. 2)*
π	*path (p. 32)*
ϕ, ψ, θ	*valuations (p. 66)*
τ	*the invisible action (p. 31)*
Δ	*transition relation schema (p. 58)*
Γ	*a set of action schemata (p. 58)*
Λ	*a set of (visible) actions (p. 31)*
Ξ, Π	*relation variables (p. 58)*
Σ	*the alphabet of an LTS (p. 31)*
Υ	*a set of partial functions from* $\mathbb{T}$ *to* $\mathbb{Z}_+$ *(Chap. 2)*
Φ, Ψ, Θ	*sets of valuations (p. 66)*
$\mathcal{F}$	*a function schema (p. 154)*
$\mathcal{G}$	*a coloured graph (p. 118)*
$\mathcal{G}(\phi)$	*the graph of a valuation* ϕ *(p. 119)*

$\mathcal{L}$	*a labelled transition system, LTS (p. 31)*
$\mathcal{P}, \mathcal{Q}, \mathcal{R}$	*LTS schemata (p. 58)*
$\mathcal{S}$	*a set schema (p. 59)*
$\mathrm{at}(\mathcal{P})$	*the set of all the atoms occurring in an LTS schema $\mathcal{P}$ (p. 60)*
$\mathrm{alph}(\mathcal{L})$	*the alphabet of an LTS $\mathcal{L}$ (p. 32)*
$\mathrm{alph}(\mathcal{M})$	*the alphabet of a DTM $\mathcal{M}$ (p. 126)*
$\deg_T(\mathcal{P})$	*the T-degree of an LTS schema $\mathcal{P}$ (p. 96)*
$\deg_T(\mathcal{P}, \phi)$	*the T-degree of an LTS schema $\mathcal{P}$ and a valuation ϕ (p. 96)*
$\mathrm{dom}(f)$	*the domain of a function f (Chap. 2)*
$\mathrm{fa}(\mathcal{L})$	*the set of all the failures of an LTS $\mathcal{L}$ (p. 141)*
$\mathrm{im}(f)$	*the image of a function f (Chap. 2)*
$\mathrm{sig}(c)$	*the signature of a valuation formula c (p. 72)*
$\mathrm{sig}(\mathcal{P})$	*the signature of an LTS schema $\mathcal{P}$ (p. 60)*
$\mathrm{tr}(\mathcal{L})$	*the set of all the traces of an LTS $\mathcal{L}$ (p. 33)*
$\mathrm{typ}_{\mathcal{P}}(\Pi)$	*the set of all the types of a relation variable Π in an LTS schema $\mathcal{P}$ (p. 60)*
$\mathrm{va}(c)$	*the set of all the valuations satisfying a valuation formula c (p. 73)*
$\{a_i\}_{i \in I}$	*an I-indexed set/family (Chap. 2)*
$A_1 \times A_2$	*the Cartesian product of sets A_1 and A_2 (Chap. 2)*
$\bigtimes_{i=1}^{n} A_i$	*the Cartesian product of sets $A_1, A_2, \ldots, A_n$ (Chap. 2)*
$\prod_{i \in I} A_i$	*the generalised Cartesian product of sets in $\{A_i\}_{i \in I}$ (Chap. 2)*
$\lvert A \rvert$	*the size of a set A (Chap. 2)*
$f\lvert_A$	*the restriction of a function f to a set A (Chap. 2)*
$f_1 \circ f_2$	*the composition of functions f_1 and f_2 (Chap. 2)*
$w\lvert_A$	*the sequence obtained from w by preserving exactly the elements in A (Chap. 2)*
$w \setminus A$	*the sequence obtained from w by removing all the elements in A (Chap. 2)*
$w_1 w_2$	*the concatenation of sequences w_1, w_2 (Chap. 2)*
$\lvert w \rvert$	*the length of a sequence w (Chap. 2)*
$L_1 + L_2$	*the sum of languages L_1 and L_2 (Chap. 2)*
$L_1 L_2$	*the concatenation of languages L_1 and L_2 (Chap. 2)*
L^*	*the Kleene star of a language L (Chap. 2)*
$\mathcal{L}_1 \parallel \mathcal{L}_2$	*the parallel composition of LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$ (p. 33)*
$\mathcal{L} \setminus \Lambda$	*an LTS $\mathcal{L}$ after hiding a set of visible actions Λ (p. 34)*

$\zeta(\mathcal{L})$ *an LTS $\mathcal{L}$ renamed by a function ζ (p. 34)*

$[\![\mathcal{P}]\!]_\phi$ *the instance of an LTS schema $\mathcal{P}$ generated by a compatible valuation ϕ (p. 68)*

$\phi[(x_1,\ldots,x_n) \mapsto \Pi]$ *the set of all valuations ϕ' obtained by (re)defining a valuation ϕ for atom variables $x_1, x_2, \ldots, x_n$ such that $(\phi'(x_1), \phi'(x_2), \ldots, \phi'(x_n))$ is in the image of a relation variable Π (p. 67)*

$\phi[x_1 \mapsto T_1, \ldots, x_n \mapsto T_n]$ *the set of all valuations ϕ' obtained by (re)defining a valuation ϕ for atom variables $x_1, x_2, \ldots, x_n$ such that $\phi'(x_i)$ is in the image of a type variable T_i for all $i \in \{1, 2, \ldots, n\}$ (p. 67)*

$\preceq$ *the prefix relation over the sequences (Chap. 2)*

$\succeq_{\text{fa}}$ *the traces-failures refinement relation over the set of LTSs (p. 141)*

$\succeq_{\text{tr}}$ *the traces refinement relation over the set of LTSs (p. 37)*

$=_{\text{fa}}$ *the failures equivalence relation over the set of LTSs (p. 141)*

$=_{\text{tr}}$ *the traces equivalence relation over the set of LTSs (p. 37)*

$\simeq$ *an isomorphism relation over the set of valuations (p. 83)*

$\not\simeq$ *a non-isomorphism relation over the set of valuations (p. 83)*

$\cong$ *an isomorphism relation over the set of coloured graphs (p. 120)*

$\Subset$ *a subvaluation relation over the set of valuations (p. 85)*

$\trianglelefteq$ *a preorder over the set of valuations (p. 94)*

$\triangleright$ *the first cell symbol of a DTM (p. 125)*

$\sharp$ *a type, relation, or an atom variable (p. 58)*

$\sqcup$ *the blank cell symbol of a DTM (p. 125)*

$\top$ *the always true valuation formula (p. 71)*

Contents

Abstract
Preface **5**
Acknowledgements **9**
Abbreviations **11**
Contents **15**
1 Introduction **17**
1.1 Formal Verification . . . 17
1.2 Parameterised Verification . . . 18
1.3 Contribution . . . 23
1.4 The Outline of the Thesis . . . 26
2 Notation and Basic Concepts **27**
3 Process Model **31**
3.1 Labelled Transition Systems . . . 31
3.2 Operators on LTSs . . . 33
3.3 Traces Refinement . . . 37
3.4 Algebra of LTSs . . . 38
4 Process Algebraic Modelling and Verification of Parameterised Systems **43**
4.1 The Precongruence Reduction . . . 43
4.2 Compositional Modelling . . . 46
4.3 Modelling a Parameterised Shared Resource System . . . 48
4.4 Verification by the Precongruence Reduction . . . 52
5 Formalism **57**
5.1 LTS Schemata . . . 58
5.2 Modelling using LTS Schemata . . . 60
5.3 Valuations . . . 66
5.4 Valuation Formulae . . . 70
5.5 The Parameterised Verification Problem . . . 74
6 The Precongruence Reduction for Valuations **79**
6.1 Generalisation by Renaming . . . 80
6.2 Generalisation by Parallel Composition . . . 85

6.3 Infinite Reduction 93
6.4 Comparison with the Related Work 99
7 Algorithmic Precongruence Reduction 105
7.1 Reduction Algorithm 105
7.2 Testing Valuation Isomorphism 118
7.3 The Verification of taDOM+ Tree-Locking Protocols 122
8 Undecidable Extensions 125
8.1 Deterministic Turing Machines 125
8.2 Undecidability with Arbitrary Valuation Formulae 126
8.3 Undecidability with Specifications Involving Hiding 138
8.4 Undecidability for More Expressive Semantics 140
8.5 Undecidability for Systems with Parametric Branching 143
8.6 Undecidability with Systems Involving Renaming 154
9 Conclusions 157
References 159
Appendices 165

1 Introduction

The behaviour of modern computer systems is increasingly non-deterministic, because systems are becoming more and more distributed, hardware is parallelising all the time, and software is becoming more and more concurrent. As the executions of distributed, parallel and concurrent parts can interleave in numerous ways, contemporary computer systems are difficult to design and analyse. Especially testing such systems is intractable, because depending on scheduling and other tasks running concurrently, not every interleaving may be possible on all the execution environments.

We concentrate on highly concurrent software systems where the number of concurrently running processes is not fixed but depends on the resources available. However, we are less interested in the internal states than (observable) behaviour and interprocess communication of such systems. The specifications we consider are so called *safety properties* (Alpern & Schneider 1985), which state that something bad should not happen and therefore correspond to negative testing. In other words, we are not so much interested in whether a system does anything useful than that it does not do anything illegal. For example, mutual exclusion properties of systems where an arbitrary number of processes compete for an access to shared resources fall in this category.

1.1 Formal Verification

(Formal) verification is about proving the correctness of a system with respect to its specification. It can be thought as exhaustive testing, too. A wide variety of formal verification techniques have been introduced and successfully applied to analyse the designs of distributed, parallel and concurrent systems (*e.g.* Clarke *et al.* 1999, Holzmann 2003, Roscoe 1997). Using the methods, several previously unknown bugs have been found while other systems are proved correct with respect to certain properties (*e.g.* Chan *et al.* 1998, Faber & Meyer 2006, Havelund *et al.* 2001, Lowe 1996, Siirtola & Valenta 2008). Unfortunately, the methods can generally handle finite system models only and applying them to realistic designs, especially those of software applications, requires a great deal of ingenuity, which is roughly the problem addressed in the thesis.

In order to get an accessible verification method, its restrictions should be explicit and it ought to be easy to apply. Hence, an accessible verification method should be

both algorithmic and *complete*, which means it ought to terminate with a correct answer on every input. However, in order to automatically verify a system, its behaviour and correctness have to be representable in a finite form that is algorithmically manageable. Unfortunately, the state space of many systems is unbounded. This is especially the case with software applications that often involve dynamic structures. Although hardware systems may have dynamic structure as well, they are built of circuits with static state space, whereas in software applications also the components often have an unbounded state space due to dynamic data structures and references to other objects. Therefore, the verification of software applications is more challenging.

On the other hand, in practice, the resources are always limited, which means a running application can never reach infinitely many states. That is why a typical approach is to model a system parameterised by the restrictions of the execution environment. In practise, the restrictions are depicted at an abstract level, for example as the (maximum) number of replicated components. As the model is instantiated for all the values the parameters are allowed to take, an (infinite) family of similar finite-state specification-system pairs results. The question whether every instance of the parameterised system is correct with respect to the corresponding specification instance is known as the *parameterised verification problem (PVP)*.

To be able to express realistic software system designs involving several different kinds of replicated parts, the use of multiple parameters is a necessity. Moreover, to enable systems with a parameterised substructure, it should be possible to parameterise the states and transitions of components or to nest parameters. Hence, the specific research problem addressed here is to devise a complete verification algorithm that allows the use of multiple and nested parameters, or multiple parameters and components with parameterised states and transitions.

1.2 Parameterised Verification

Unfortunately, there is no complete solution for PVP, because the problem is undecidable in general (Apt & Kozen 1986), even for the systems of a ring (Suzuki 1988) or star (Emerson & Kahlon 2003b) topology. That is why all algorithmic solutions to the problem are somehow restricted.

In general, the process of parameterised verification proceeds as shown in Figure 1. Given an instance of PVP, one first constructs a candidate for its abstract version such that the proposed abstraction can be checked using existing verification algorithms. If

the candidate is created using an abstraction mapping, then the properties of the mapping guarantee that it is a true abstraction, *i.e.* every behaviour of the original instance is covered by the abstraction. Otherwise, this correspondence between the original problem instance and its proposed abstraction has to be checked separately by means provided by the verification method. If the candidate cannot be shown to be an abstraction, then the method is either not applicable or one needs to refine the candidate and try to prove the correspondence again.

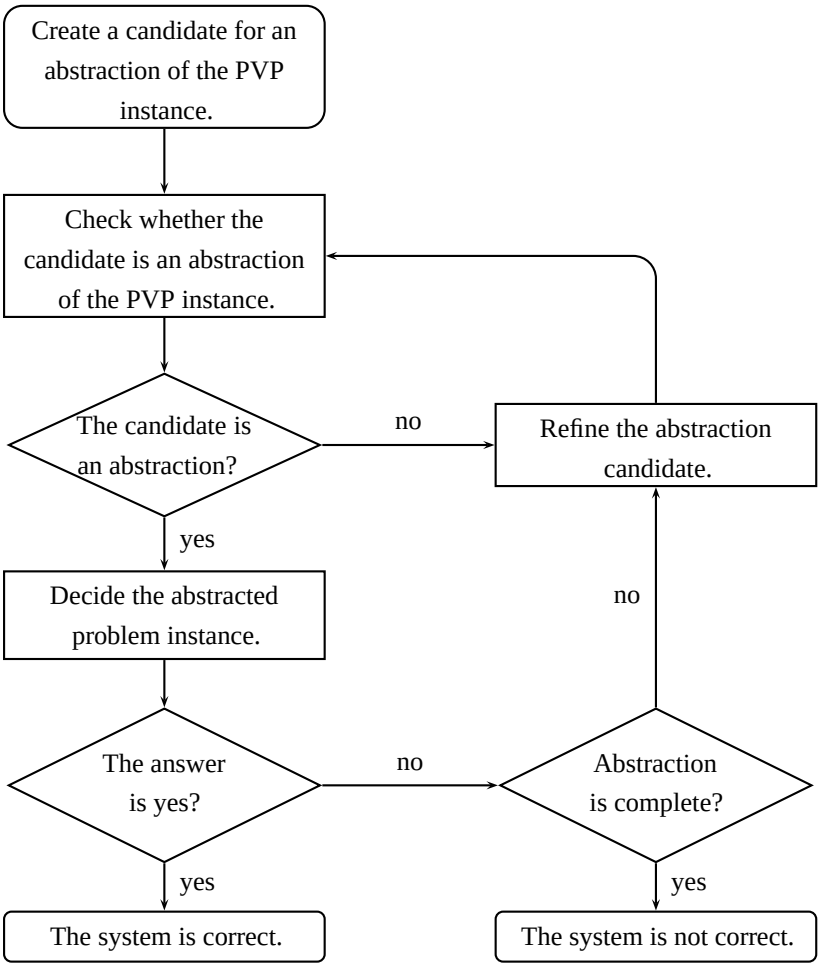

Fig 1. Outline of a generic parameterised verification method.

After obtaining an abstraction of the PVP instance, one can try to decide it. If the answer is positive, then the correspondence between the original problem instance and its abstraction guarantees that the system works correctly in the sense of the specification. However, if the result is negative, then the system may be incorrect or the abstraction too coarse. If the correspondence is mutual (commutative), then we know that it is the system that does not work as specified, but otherwise one has to refine the abstraction and decide it again hoping for a positive answer.

The method template in Figure 1 suggests that there are essentially two different kinds of approaches to PVP: those that involve the inner refinement loop and those that do not. Additionally, we distinguish a class of approaches which includes methods of both kinds: approaches which reduce a parameterised verification task to a finite one by providing bounds to parameter values such that the system is correct for parameter values equal to or below their respective bounds if and only if it is correct for all the allowed values.

The methods of the first kind, which include checking the correspondence between a problem instance and its proposed abstraction, are called *induction methods* (Creese 2001, Kurshan & McMillan 1995, Wolper & Lovinfosse 1990). A crucial part of these approaches is the construction of a *network invariant* which is an abstraction of a (partial) system such that it remains invariant under the addition of components. Typically, a user has to provide a candidate for a network invariant, which is a task that generally requires a lot of ingenuity and in-depth knowledge on the system. Fortunately, the correctness of the guess, which involves induction on the structure of the system, can usually be verified using existing tools.

Once a network invariant is found, one can substitute (the parameter-dependent part of) the system for it, and perform verification, where one can exploit existing tools again. However, if the result is negative, then the system may be incorrect or the network invariant too weak to prove the property. To find out which one is true, a counter-example provided by the tool needs to be examined. It involves manual reasoning which is prone to errors, as the counter-example is related to the abstract system, not the original one.

If too weak a network invariant appears to be the reason for the negative answer, one can try to strengthen it. Unfortunately, because a network invariant must meet conflicting requirements, *i.e.* it has to be abstract enough to represent a system of any size, strong enough for the analysis of correctness, and still finitely representable to be automatically verified, it does not always exist. Therefore, the induction methods

cannot always provide the answer, which means that it is difficult to know in advance whether they are applicable.

The strength of induction methods is that they are generally applicable to various kinds of systems, including multiply parameterised ones and those with parameterised components (Creese 2001, Kurshan *et al.* 1994). Sometimes, it is also possible to construct a network invariant automatically. If a system is a network of identical processes, it may be possible to find a network invariant completely automatically (Grinchtein *et al.* 2006, Li *et al.* 1994, Pyssysalo 1996, Valmari & Tienari 1991). Another possibility is to try to describe a system using a network grammar which also allows the automated discovery of a network invariant (Clarke *et al.* 1997, Shtadler & Grumberg 1990, Lesens *et al.* 2001). Although the approaches overcome some of the inconveniences of induction methods, none of them can guarantee termination, yet.

Another class of parameterised verification techniques with similar pros and cons is *abstraction methods*, where an abstract version of the problem instance is created using an abstraction mapping (Cousot & Cousot 1977). In the early days of parameterised verification, an abstraction mapping had to be provided explicitly (Aggarwal *et al.* 1983), but in modern techniques, it is defined indirectly in the form of a set of predicates (Clarke *et al.* 2006, 2008, Graf & Saïdi 1997, Lahiri *et al.* 2003, Wachter & Westphal 2007, Yahav 2001). By evaluating the predicates on system states, one obtains tuples of truth values that make the state space of the abstract system.

Typically, an abstraction mapping has to be specified manually, but the abstraction of the system (and specification) can be generated automatically. Unfortunately, required tools are often at a prototype level or publicly unavailable. There are also abstraction methods that automatically compute an abstraction mapping, but the result is often not as good as obtained by manual reasoning (Lahiri & Bryant 2004). Moreover, like a network invariant, also an abstraction mapping must meet conflicting requirements: it should preserve enough details for the analysis of correctness but throw away as much information as needed to represent the system and specification in a finite form. Therefore, a suitable abstraction mapping does not always exists, which means that the disadvantages of generic abstraction methods are similar to those of induction methods. On the positive side, the methods in this class are typically not tight to a specific network topology, and even the analysis of multiply parameterised systems is possible (Lahiri & Bryant 2004, Wachter & Westphal 2007).

All the methods referenced above have the drawback that they are not complete. Nevertheless, there are also complete abstraction methods. These approaches are typi-

cally based on *counter abstraction*, where one stores the number of processes in each of the possible states instead of explicitly tracking the states of individual processes (Delzanno 2003, Delzanno *et al.* 2002, German & Sistla 1992, Lubachevsky 1984). It is also possible to count until some integer n and then forget the exact number of processes (Emerson & Namjoshi 1996). The use of counter abstraction presumes that the replicated components of the system are treated symmetrically (Clarke *et al.* 1996, Emerson & Sistla 1996, Ip & Dill 1996) and have a fixed state space. Hence, the application domain of the counter abstraction methods is pretty limited. On the positive side, some of the counter abstraction methods are complete and allow the use of multiple parameters (Delzanno *et al.* 2002) although also incomplete approaches that allow only one parameter are common (Ip & Dill 1999). There are also some tools available for counter abstraction based verification (Delzanno 2003, Ip & Dill 1999).

Counter abstraction techniques of the former kind rely on infinite-state verification algorithms. However, in some cases, such algorithms can be applied directly to parameterised systems and specifications, which can be thought as the application of unit abstraction. This is the case when the model of computation directly reflects the structure of the system. As the states of infinite models of computation are often sequences, arrays or trees over a finite alphabet, they naturally apply to arrays, rings and trees of similar fixed-size processes when the states of a single process are taken as the alphabet (Abdulla *et al.* 2007, 2004, 2006, Bingham & Hu 2005, Kesten *et al.* 2001). Consequently, infinite-state verification can naturally be applied to systems composed of identical processes with a static state space only. Some of the algorithms are implemented (Abdulla *et al.* 2004), but despite few exceptions (Bingham & Hu 2005), they are not guaranteed to terminate.

The final class of parameterised verification methods is called *cut-off results* or, in the context of logic programs, *small model theorems* that provide bounds, cut-offs, for the values of the parameters such that the system is correct for parameter values at most their respective cut-offs if and only if it is correct for all the allowed parameter values. Such results have been established for systems composed of similar fixed-size processes (Emerson & Kahlon 2000, 2003a), systems of a ring topology (Emerson & Kahlon 2002, 2004, Emerson & Namjoshi 2003), networks of homogeneous processes (Clarke *et al.* 2004), systems of identical processes competing for access to a fixed number of shared resources according to prioritised queue policy (Bouajjani *et al.* 2008), and data-independent systems (Lazić 1999, Lazić & Nowak 2000, Wolper 1986). The advantage of these methods is that they are typically complete, often straightforward

to apply, and the remaining instances can usually be checked with the aid of existing verification tools. Moreover, possible counter-examples are executions of the original system, not some abstract construction. The drawback is that the application domain of the methods is typically very narrow. For example, only the result of Emerson & Kahlon (2000) can handle systems with multiple parameters, but neither does it allow parameterised components to be expressed.

There are also methods that establish a cut-off by inductive reasoning (Konnov & Zakharov 2005, Li *et al.* 1994, Pyssysalo 1996, Valmari & Tienari 1991), but, like other induction methods, they are not complete. An exception is the approach of Valmari & Tienari (1991), which is shown to terminate if certain structural conditions are met (Nazari & Thistle 2007). Moreover, all the results concern rings of similar processes, where the only parameter is the size of the ring, and, apart from the result of Pyssysalo (1996), the state space of the processes cannot be parameter-dependent. However, with lots of ingenuity, the method of Valmari & Tienari (1991) can demonstrably handle systems with a linear topology, too (Valmari & Kokkarinen 1998). With a clever formulation of the problem, it might be applicable to multiply parameterised systems as well.

1.3 Contribution

Based on the literature review above, strengths and weaknesses of different kinds of parameterised verification methods are collected in Table 1.

Table 1. Strengths and weaknesses of different types of existing parameterised verification methods.

property	method type			
	induction	abstraction		cut-off
		generic	unit/counter	
multiple parameters	possible	possible	possible	possible
parameterised substructure	possible	possible	no	possible
automation	limited	limited	possible	possible
completeness	no	no	possible	possible

The table shows that induction and generic abstraction methods are not complete, and current approaches based on unit or counter abstraction do not allow parameterising

the substructure of a system. Hence, the only possibility to answer the research problem seems to be a cut-off result that supports multiple and nested parameters, or multiple parameters and components with parametric states and transitions.

Currently, to the best of our knowledge, no cut-off result has both the properties. In this thesis, we provide a cut-off result with both the features by improving the algebra of *labelled transition systems (LTSs)* (Hoare 1985, Roscoe 1997), which suits especially well for the analysis of software systems, or at least we think this way. LTSs are state machines of a kind with emphasis on transition labelled by *actions*. There are also a number of operators that enable systems and specifications to be composed of smaller parts. Typically, one first models one component of each type as an LTS and then uses *renaming* of actions to produce other similar components. After that the component LTSs are put together or *composed in parallel* by synchronising the transitions with the same label. Similarly, the operators allow a specification LTS to be composed of simpler conjunctive parts. Finally, before the correctness of the system is checked, the actions irrelevant to the specification are *hidden* by changing them to the invisible ones.

Verification is done by comparing the semantic value of a system LTS against that of a specification LTS. The simplest practically useful semantics considers an LTS as a set of its *traces*, finite sequences of visible actions that start from the initial state. An LTS is said to be a *traces refinement* of another LTS, if every trace of the former LTS is also a trace of the latter one. A system is considered correct with respect to a specification if the system LTS is a traces refinement of the specification LTS. This way it is possible to prove the absence of illegal behaviour but not the presence of some desired one, which means that the traces refinement is a match for safety property verification.

Unfortunately, LTSs can only express systems with a static state space. Hence, to represent a parameterised system, a different LTS is needed for each possible combination of parameter values, which means a parameterised verification task can be expressed as an infinite family of finite specification-system LTS pairs. Because the pairs are instances of the same parameterised verification task they share a common structure, which can sometimes be exploited to reduce the number of refinement checks needed.

The first major contribution of the thesis is a reduction method for (infinite) families of (finite) LTS pairs representing refinement checking tasks (arising from parameterised specification and system descriptions). The key idea is to exploit the algebraic properties of the traces refinement, mainly the fact that the relation is a precongruence, *i.e.* a reflexive and transitive relation that is preserved under the application of the operators, which sometimes allows the correctness of large specification-system instances to be

derived from that of smaller ones. In such a case, large specification-system instances can be discarded without changing the answer to the problem, and in the best case, only a finite number of (small) refinement checking tasks remains. As the remaining tasks can be decided using existing finite-state refinement checking tools, the approach leads to a parameterised verification method which we call *the precongruence reduction*.

The main contribution of the thesis is an algorithm that automates the precongruence reduction. For that purpose, parallel composition and hiding operators are replaced by their symbolic equivalents and some actions and sets of actions are represented by variables leading to a structure we call an *LTS schema*. There is also a replicated version of the parallel composition, which enables parameterised parts of the system and specification to be expressed. When the values of variables are fixed using a *valuation*, an instance of the LTS schema, an LTS, is obtained. The (infinite) sets of valuations are expressed as a *valuation formula*, which is an expression of first order logic.

In this context, the parameters are variables typically used to denote sets of the identities of replicated components and relationships between them. Hence, the parameterised verification problem can be stated as follows: given a specification and system LTS schema and a valuation formula, determine whether the instance of the system LTS schema is correct with respect to the corresponding instance of the specification LTS schema for all valuations arising from the formula.

The algorithm inputs an instance of the parameterised verification problem of the above kind and reduces the (infinite) set of valuations determined by the valuation formula to a finite one without changing the answer to the problem. The algorithm applies whenever the specification does not involve hiding and existential quantification is not used in the valuation formula, so its application domain is very explicit in this sense. As LTS schemata are defined inductively and the number of variables is not limited, the algorithm can handle multiply parameterised systems with parameterised structure. After the application of the algorithm, the problem can be solved using existing finite-state refinement checkers. Hence, all the goals set for the research are fulfilled.

Informally, the algorithm applies to parameterised systems and safety properties such that the resulting infinite family of specification-system LTS pairs is closed under the removal of a replicated component. For example, systems with a star, bipartite and totally (un)connected topology as well as request-take-release systems (Bouajjani *et al.* 2008) and those with conjunctive guards (Emerson & Kahlon 2000) are such, but those with a ring, linear or tree topology are not. However, if it is possible to capture the behaviour of the system from the viewpoint of any two components connected to

each other in finitely many LTS schemata, then one can study the transitive closures of rings, arrays and forests instead, which are closed under the removal of a replicated component. This is also the case with the shared tree system used as a running example.

Finally, we consider several natural and significant extensions to our result. First, we show that allowing either existential quantification in a valuation formula or hiding in a specification makes the problem undecidable. Hence, both the restrictions explicit in our result are more or less necessary. Next, we consider richer semantics that allows the analysis of deadlocks and *liveness properties* (Alpern & Schneider 1985), *i.e.* checking whether a system actually does something useful. We also consider system LTS schemata equipped with parameterised transitions and those equipped with parametric renaming. We show that all these extensions make the problem undecidable, which means that also the restrictions implicit in our result are hard to overcome. Hence, the result is rather close to the decidability frontier and the algorithm cannot be significantly extended in any direction without restricting some other aspect.

1.4 The Outline of the Thesis

The next chapter introduces the (standard) notation and concepts used throughout the thesis. After that, the algebra of LTSs is revised, and in Chapter 4, the precongruence reduction with the related compositional modelling technique is introduced and justified in the level of LTSs.

In Chapter 5, we present a formalism for expressing parameterised specifications and systems as LTS schemata and sets of parameter values as valuation formulae. In the following chapter, the reduction method is lifted into the level of our formalism and a cut-off result based on the method is proved. An algorithm that automates the application of the result is given in Chapter 7.

Finally, in Chapter 8, several natural and significant extensions to the result and our formalism are considered. It is shown that the problem becomes undecidable in each of the cases. The thesis concludes with a discussion on possible decidable extensions to the theory.

2 Notation and Basic Concepts

We assume basic knowledge on set theory, abstract algebra and predicate logic. However, for the sake of unambiguity, we briefly review some related concepts.

Sets We write $\mathbb{N}$ for the set $\{0,1,2,\ldots\}$ of all the non-negative integers, and $\mathbb{Z}_+$ for the set $\{1,2,3,\ldots\}$ of all the positive integers. If $n \in \mathbb{N}$ and a_i is an element for all $i \in \{1,2,\ldots,n\}$, then $(a_1,a_2,\ldots,a_n)$ is an *n-tuple*. The 0-tuple $()$ is also called *the empty tuple*, and a 1-tuple (a_1) is often abbreviated as the element a_1. The *size* of an n-tuple is n.

Let $n \in \mathbb{N}$ and let A_i be a set for all $i \in \{1,2,\ldots,n\}$. The *Cartesian product (of $A_1,A_2,\ldots,A_n$)*, denoted by $A_1 \times A_2 \times \ldots \times A_n$ or $\bigtimes_{i=1}^{n} A_i$ for short, is the set of all non-empty n-tuples $(a_1,a_2,\ldots,a_n)$ such that $a_i \in A_i$ for all $i \in \{1,2,\ldots,n\}$. Hence, the Cartesian product of zero sets is the empty set. If $A_i = A$ for all $i \in \{1,2,\ldots,n\}$, then the Cartesian product of $A_1,A_2,\ldots,A_n$ can be abbreviated as $\bigtimes_{i=1}^{n} A$.

Relations Let $A_1,A_2,\ldots,A_n$ be sets for some integer $n \in \mathbb{N}$. An *(n-ary) relation (over $A_1,A_2,\ldots,A_n$)* is a subset of $\bigtimes_{i=1}^{n} A_i$. A relation over $A_1,A_2,\ldots,A_n$, where $A_i = A_j$ for all $i,j \in \{1,2,\ldots,n\}$, is simply called an n-ary relation over A. A relation G' is a *subrelation* of G and respectively G is a *superrelation* of G', if G' is a subset of G.

A *binary relation* is a relation over two sets. If G is a binary relation, we write aGb if and only if $(a,b) \in G$. A binary relation G over a set A is

- *reflexive*, if aGa for every $a \in A$;
- *irreflexive*, if there is no $a \in A$ such that aGa;
- *symmetric*, if a_1Ga_2 implies a_2Ga_1;
- *asymmetric*, if a_1Ga_2, a_2Ga_1 implies that a_1 is a_2;
- *transitive*, if a_1Ga_2, a_2Ga_3 implies a_1Ga_3;
- *intransitive*, if there are no distinct $a_1,a_2,a_3 \in G$ such that $a_1Ga_2, a_2Ga_3, a_1Ga_3$.

The *transitive closure* of G is the minimal transitive superrelation of G. Respectively, the *reflexive closure* of G is the minimal reflexive superrelation of G.

A reflexive and transitive relation is a *preorder*. An asymmetric preorder $\leq$ over a set A is a *total order*, if for every $a_1,a_2 \in A$, either $a_1 \leq a_2$ or $a_2 \leq a_1$. A *proper total order* is an irreflexive relation whose reflexive closure is a total order. A symmetric

preorder is an *equivalence*. If $\approx$ is an equivalence over A and $a \in A$, then set the $\{b \in A \mid a \approx b\}$ is called an *equivalence class (of $\approx$)*. If A' is a subset of A, then a *set of representatives (of $\approx$) (in A')*, is a maximal subset of A' that contains at most one element from every equivalence class of $\approx$.

Functions Let A and B be sets. A *partial function f (from A to B)*, denoted by $f : A \mapsto B$, is a binary relation over A and B such that whenever afb_1, afb_2 then $b_1 = b_2$. The *domain of f*, denoted by $\mathrm{dom}(f)$, is the set $\{a \in A \mid \exists b \in B : afb\}$ of all the elements in A related to some element in B. Respectively, the *image of f*, denoted by $\mathrm{im}(f)$ is the set $\{b \in B \mid \exists a \in A : afb\}$ of all the elements in B related to some element in A. Because for every $a \in \mathrm{dom}(f)$ there is a unique element $b \in B$ such that afb, one can refer to b by $f(a)$, for short. A partial function $f : A \mapsto B$ with domain A is a *(total) function (on A)*. If C is a set, then the *restriction of f (to C)*, denoted by $f|_C$, is a function: $A \cap C \mapsto B$ with the domain $\mathrm{dom}(f) \cap C$ such that $f|_C(a) = f(a)$ for every $a \in \mathrm{dom}(f) \cap C$. A function $f : A \mapsto B$ is an *extension* of a function f' to A, if f' is a restriction of f. If $f_1 : A \mapsto B$ and $f_2 : B \mapsto C$ are functions such that $\mathrm{im}(f_1) \subseteq \mathrm{dom}(f_2)$, then *the composition of f_1 and f_2*, denoted by $f_1 \circ f_2$, is a function: $A \mapsto C$ such that $\mathrm{dom}(f_1 \circ f_2) = \mathrm{dom}(f_1)$ and $(f_1 \circ f_2)(a) = f_2(f_1(a))$ for all $a \in \mathrm{dom}(f_1 \circ f_2)$.

If I is an (index) set, then an *I-indexed set or family (of elements in A)* is a function: $I \mapsto A$. We write $\{a_i\}_{i \in I}$ for an I-indexed set a such that $a(i) = a_i$ for all $i \in I$. If $\{A_i\}_{i \in I}$ is an I-indexed set of sets, then the *generalised Cartesian product (of the sets in $\{A_i\}_{i \in I}$)*, denoted by $\prod_{i \in I} A_i$, is the set of all functions $f : I \mapsto \bigcup_{i \in I} A_i$ such that $f(i) \in A_i$ for all $i \in I$. The set $\{A_i\}_{i \in I}$ is a *partition* of a set A if A_i is a non-empty subset of A for all $i \in I$ such that $A = \bigcup_{i \in I} A_i$, and A_i and A_j are disjoint whenever i and j are different elements in I.

Sequences If $n \in \mathbb{N}$, A is a set and $a_1, a_2, \ldots, a_n \in A$, then $a_1 a_2 \ldots a_n$ is a *sequence (of length n) (over A)*. The length of a sequence w is denoted by $|w|$, and the sequence of length zero is denoted by ε. The set of all the sequences of length $k \in \mathbb{N}$ over A is denoted by A^k and the set of all the sequences over A is referred to by A^*. A subset of A^* is called a *language (over A)*. If $w_1 = a_1 a_2 \cdots a_{n_1}$ and $w_2 = b_1 b_2 \cdots b_{n_2}$ are sequences over A, where $a_i \in A$ for all $i \in \{1, 2, \ldots, n_1\}$ and $b_j \in A$ for all $j \in \{1, 2, \ldots, n_2\}$, the *concatenation* of w_1 and w_2, denoted by $w_1 w_2$, is sequence $a_1 a_2 \cdots a_{n_1} b_1 b_2 \cdots b_{n_2}$ obtained by writing w_2 after w_1 a letter by letter. A sequence $w_1 \in A^*$ is a *prefix* of a sequence $w_2 \in A^*$, denoted by $w_1 \preceq w_2$, if there is a sequence $w_1' \in A^*$ such that w_2 is

$w_1 w_1'$. A language L is *prefix-closed*, if whenever $w \in L$ and $w' \preceq w$, then $w' \in L$, too. If $w \in A^*$ and $A' \subseteq A$, then $w \setminus A'$ is a sequence obtained from w by removing all the elements in A', and $w|_{A'}$ denotes the sequence $w \setminus (A \setminus A')$ obtained from w by removing all the elements not in A'.

Algebras Let A be a set. An *(n-ary) operation (on A)*, where $n \in \mathbb{N}$, takes n elements and returns a single element in A. Hence, an operation of a positive arity n is a function: $\bigtimes_{i=1}^{n} A \mapsto A$, and 0-ary operation is a constant, a single element in A. A 2-ary operation is also called a *binary* operation. If $*$ is a binary operation on A and $a_1, a_2 \in A$, we write $a_1 * a_2$ short for $*(a_1, a_2)$. A binary operation $*$ on A is

- *commutative*, if $a_1 * a_2 = a_2 * a_1$ for all $a_1, a_2 \in A$,
- *associative*, if $(a_1 * a_2) * a_3 = a_1 * (a_2 * a_3)$ for all $a_1, a_2, a_3 \in A$, and
- *idempotent*, if $a * a = a$ for all $a \in A$.

Furthermore, an element $a \in A$ is

- an *identity element (with respect to $*$)*, if $a * a' = a' * a = a'$ for all $a' \in A$,
- an *idempotent element (with respect to $*$)*, if $a * a = a$.

A pair (A, O), where A is a set and O a set of operations on A, is an *algebra*. Let (A, O) be an algebra and $\leq$ a preorder over A. The preorder $\leq$ is a *precongruence*, if

$$a_1 \leq b_1, a_2 \leq b_2, \ldots, a_n \leq b_n \text{ implies } o(a_1, a_2, \ldots, a_n) \leq o(b_1, b_2, \ldots, b_n)$$

whenever $n \in \mathbb{N}$, o is an n-ary operation on O, and $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n \in A$. A symmetric precongruence is a *congruence*.

Let L, L_1, L_2 be languages over A. The *concatenation* of L_1 and L_2, denoted by $(L_1 L_2)$, is the set $\{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$ of all sequences that are concatenations of an element in L_1 and an element in L_2. The *sum* of L_1 and L_2, denoted by $(L_1 + L_2)$, is the union $L_1 \cup L_2$. Moreover, we define L^0 as the language $\{\varepsilon\}$ containing the empty sequence only, and L^{k+1} as the concatenation $L^k L$ for all $k \in \mathbb{N}$. The *Kleene star* of L is the infinite union $\bigcup_{k \in \mathbb{N}} L^k$ denoted by L^*. Clearly, the set of all the languages over A equipped with the concatenation, sum and Kleene star forms an algebra. If there is no chance of confusion in which order operations are performed, the parenthesis and brackets can be omitted.

3 Process Model

A *process* is an abstraction of the behaviour of a system. Here, a *system* refers typically to a software one, but in general it can also be a machine or a human actor, and respectively *behaviour* refers to the executions of a software system, or the actions of a machine or a human being. A *process algebra* consists of a set of processes and a set of operators that enable systems and their specifications to be specified and verified by the comparison of processes. Traditionally, systems and specifications are constructed inductively from simple processes by the application of operators, and the algebraic properties of operators are exploited when comparing a system process against a specification one to establish its correctness. For a more complete discussion on process algebras, see Baeten (2005).

3.1 Labelled Transition Systems

Our formal process model is a *labelled transition system*, an LTS. Intuitively, an LTS is a graph the vertices of which are called states, the edges are labelled by actions and they are called transitions, and one of the states is marked as the initial one. However, before LTSs are introduced formally, we define actions, which come in two forms, visible and invisible.

We assume a countably infinite set $\mathbb{A}$ of *atoms* denoted by a, b and their variants. Tuples of atoms are called *actions*. The empty tuple (of atoms), also denoted by τ, is called the *invisible* action and the rest of the actions are *visible*. The set of all the visible actions is referred to by $\mathbb{V}$. In theoretical contexts, actions are denoted by α, β and their variants, and sets of actions by Λ and its variants. For modelling purposes, also other, more informative names for actions are used.

Definition 1 (LTS)**.** A *labelled transition system* is a four-tuple $(S, \Sigma, R, \hat{s})$, where

- S is a non-empty set of *states*,
- $\Sigma \subseteq \mathbb{V}$ is a set of visible actions,
- $R \subseteq S \times (\Sigma \cup \{\tau\}) \times S$ is a set of *transitions*, and
- $\hat{s} \in S$ is the *initial state*.

In theoretical contexts, LTSs are denoted by $\mathcal{L}$ and its variants. For modelling purposes, also other, more informative names are used. If $\mathcal{L} = (S, \Sigma, R, \hat{s})$ is an LTS, then

the set S is called the *state space* of $\mathcal{L}$, the set Σ, also denoted by $\mathrm{alph}(\mathcal{L})$, is said to be the *alphabet* of $\mathcal{L}$, and R is called a *transition relation*. The *size* of an LTS is the sum of the sizes of its state space and alphabet. An LTS of finite size is simply called *finite*.

LTSs are often given graphically, especially when modelling something. The states are typically marked by circles and the transitions by labelled arrows between the states, the arrow-head pointing to the latter state of the transition. The initial state is marked by an open-ended arrow, and the alphabet typically consists of all the transition labels other than τ.

For example, the behaviour of a user u_1 that needs to obtain a lock to use a shared resource is modelled as the LTS $User_1$ of Figure 2. Initially, the user can request a read or write lock or perform some internal operations represented by the invisible action. The read lock allows read access only, whereas the write lock entitles to reading and writing to the resource. However, the read lock can be upgraded by requesting the write lock. After obtaining the lock, the user can repeatedly access the resource, release the lock and return to the initial state, or again perform internal activities. Because the actions are instantaneous, both reading the resource and writing to it are modelled by two actions indicating the beginning and the end of the operation.

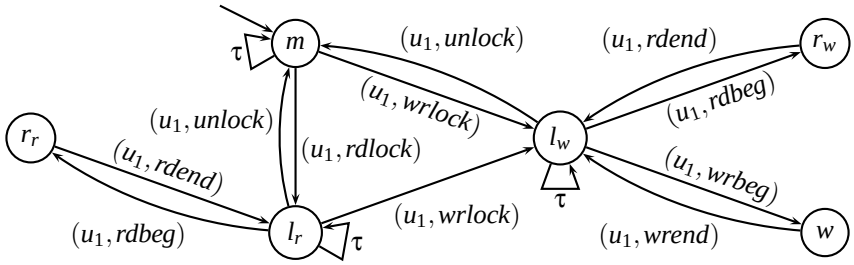

Fig 2. LTS $User_1$ representing the behaviour of the user u_1 from the viewpoint of accessing a shared resource.

In the analysis of LTSs, we are usually interested in the sequences of visible actions reachable from the initial state. Let $\mathcal{L}$ be an LTS. A finite alternating sequence of states and actions, $s_1\alpha_1 s_2 \ldots \alpha_{n-1} s_n$, of $\mathcal{L}$, is a *path (of $\mathcal{L}$) (from s_1) (to s_n)*, if (s_i, α_i, s_{i+1}) is a transition of $\mathcal{L}$ for every $i \in \{1, 2, \ldots, n-1\}$. When there is a path of $\mathcal{L}$ from a state s to a state s', we say that *s' is reachable from s (in $\mathcal{L}$) (by that path)*. A path of $\mathcal{L}$ from s_a to s_b is an *execution (of $\mathcal{L}$) (to s_b)*, if s_a is the initial state. A state s is said to be *reachable (in $\mathcal{L}$)*, if there is an execution of $\mathcal{L}$ to s. The reachable part of an LTS $\mathcal{L}$

is the LTS obtained from $\mathcal{L}$ by restricting the state space and the transition relation to the reachable states. A sequence t of visible actions of an LTS $\mathcal{L}$ is a *trace (of $\mathcal{L}$)*, if there is an execution $s_1\alpha_1 s_2 \ldots \alpha_{n-1} s_n$ of $\mathcal{L}$ such that $t = \alpha_1\alpha_2 \cdots \alpha_{n-1} \setminus \{\tau\}$, *i.e.* t can be obtained from the execution by removing all the states and the invisible actions.

The set of all the traces of an LTS $\mathcal{L}$, the *traces of* $\mathcal{L}$ for short, is referred to by $\mathrm{tr}(\mathcal{L})$. Obviously, $\mathrm{tr}(\mathcal{L})$ is a prefix-closed language over the alphabet of $\mathcal{L}$. For example, the traces of the LTS $User_1$ in Figure 2, is the set

$$\{t' \preceq t \mid t \in ((u_1, wrlock)\mathcal{T}_{lw} + (u_1, rdlock)\mathcal{T}_{lr})^*\} ,$$

where

$$\mathcal{T}_{lw} = (((u_1, wrbeg)(u_1, wrend)) + ((u_1, rdbeg)(u_1, rdend)))^*(u_1, unlock) ,$$
$$\mathcal{T}_{lr} = ((u_1, rdbeg)(u_1, rdend))^*((u_1, unlock) + (u_1, wrlock)\mathcal{T}_{lw}) .$$

3.2 Operators on LTSs

When a system is modelled as an LTS, it is typically built of smaller LTSs representing its parts using a *parallel composition*. The LTSs composed in parallel must take the transitions labelled by the same visible action simultaneously. If there are similar components that are identical modulo the action names, it is convenient to create them from one template by *renaming*. Finally, the actions that are irrelevant to the specification are *hidden*, *i.e.* replaced by the invisible action.

Let $\mathcal{L}_i = (S_i, \Sigma_i, R_i, \hat{s}_i)$ be an LTS for both $i \in \{1,2\}$. The *parallel composition (of $\mathcal{L}_1$ and $\mathcal{L}_2$)*, denoted by $\mathcal{L}_1 \parallel \mathcal{L}_2$, is a four-tuple

$$(S_1 \times S_2, \Sigma_1 \cup \Sigma_2, R_\parallel, (\hat{s}_1, \hat{s}_2)) ,$$

where $R_\parallel$ consists of all tuples $((s_1, s_2), \alpha, (s_1', s_2'))$ such that

- $\alpha \in \Sigma_1 \cap \Sigma_2$ and $(s_i, \alpha, s_i') \in R_i$ for both $i \in \{1,2\}$, or
- $\alpha \in (\Sigma_i \cup \{\tau\}) \setminus \Sigma_j$ and i, j are different elements in $\{1,2\}$ such that $(s_i, \alpha, s_i') \in R_i$ and $s_j = s_j'$.

This definition results in a standard Hoare style parallel composition (Hoare 1985). According to it, the LTSs can execute a common visible action α in the parallel composition if and only if both of them agree on its execution, whereas the invisible actions and the visible actions that are only in the alphabet of one LTS are executed individually.

Let $\mathcal{L}$ be an LTS and ζ a function from $\mathbb{V}$ to $\mathbb{V}$. The LTS $\mathcal{L}$ *renamed by* ζ is a four-tuple

$$(S, \{\zeta(\alpha) \mid \alpha \in \Sigma\}, R_\zeta, \hat{s}) ,$$

denoted by $\zeta(\mathcal{L})$, where

$$R_\zeta = \{(s_1, \zeta(\alpha), s_2) \mid (s_1, \alpha, s_2) \in R \wedge \alpha \in \Sigma\} \cup \{(s_1, \tau, s_2) \in R\} .$$

Hence, $\zeta(\mathcal{L})$ is obtained from $\mathcal{L}$ by applying ζ to all the visible actions.

Let $\mathcal{L}$ be an LTS and Λ a set of visible actions. The LTS $\mathcal{L}$ *after hiding* Λ, is a four-tuple

$$(S, \Sigma \setminus \Lambda, R_\Lambda, \hat{s}) ,$$

denoted by $(\mathcal{L} \setminus \Lambda)$, where

$$R_\Lambda = \{(s, \alpha, s') \in R \mid \alpha \notin \Lambda\} \cup \{(s, \tau, s') \mid \exists \alpha \in \Lambda : (s, \alpha, s') \in R\} .$$

Hence, $(\mathcal{L} \setminus \Lambda)$ is obtained from $\mathcal{L}$ by changing the transition labels in Λ to the invisible one and removing the elements in Λ from the alphabet of $\mathcal{L}$.

The structures obtained from LTSs by the parallel composition, hiding and renaming are LTSs. It means that the parallel composition, hiding and renaming are operators, and the set of all the LTSs together with these operators forms an algebra. The operators also preserve finiteness in the following sense.

Lemma 2. *Let $\mathcal{L}_1, \mathcal{L}_2$ be LTSs, ζ a function from $\mathbb{V}$ to $\mathbb{V}$, and Λ a set of visible actions.*

1. *The parallel composition of LTSs $\mathcal{L}_1, \mathcal{L}_2$, the LTS $\mathcal{L}_1$ renamed by ζ and the LTS $\mathcal{L}_1$ after hiding Λ are LTSs.*
2. *If $\mathcal{L}_1, \mathcal{L}_2$ are finite, then $(\mathcal{L}_1 \parallel \mathcal{L}_2)$, $\zeta(\mathcal{L}_1)$ and $(\mathcal{L}_1 \setminus \Lambda)$ are finite, too.*

The claims follow straightforwardly from the definitions.

The lemma implies that we can generalise the parallel composition to any finite totally ordered set of LTSs as follows. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a finite totally ordered index set such that $i_1 < i_2 < \ldots < i_n$, and $\mathcal{L}_i = (S_i, \Sigma_i, R_i, \hat{s}_i)$ an LTS for every $i \in I$. The *parallel composition* of LTSs in the set $\{\mathcal{L}_i\}_{i \in I}$, denoted by $(\|_{i \in I}\ \mathcal{L}_i)$ or $(\|_{j=1}^{n}\ \mathcal{L}_{i_j})$, is defined as

$$((\ldots((\mathcal{L}_{i_1} \parallel \mathcal{L}_{i_2}) \parallel \mathcal{L}_{i_3})\ldots) \parallel \mathcal{L}_{i_n})$$

when I is non-empty. In the case of empty I, $(\|_{i \in I}\ \mathcal{L}_i)$ is defined as a single state LTS $(\{()\}, \emptyset, \emptyset, ())$ with the empty alphabet.

The lemma also means that many parallel composition operators of other process algebras can be expressed with the aid of our operators (Valmari 2000). For example, the parallel composition operator of LOTOS (Language of Temporal Ordering Specification) (Bolognesi & Brinksma 1987) and the parallel composition operators of CSP (Communicating Sequential Processes) (Roscoe 1997) are such. Also the parallel composition operator of CCS (Calculus of Communicating Systems) (Milner 1989) can be constructed using the Hoare style parallel composition but it necessitates the use of *relational renaming*, which allows a single visible action to be mapped to multiple visible actions.

In addition to being very general, our parallel composition has another benefit. By looking at a trace of the parallel composition, one can see which traces the LTSs composed in parallel have executed. This and also the influence of other operators on the traces of LTSs are captured in the following lemma.

Lemma 3. *Let $\mathcal{L}_1, \mathcal{L}_2$ be LTSs, ζ a function from $\mathbb{V}$ to $\mathbb{V}$, and Λ a set of visible actions.*

1. *If t is a sequence over $\mathrm{alph}(\mathcal{L}_1) \cup \mathrm{alph}(\mathcal{L}_2)$, then t is a trace of $(\mathcal{L}_1 \parallel \mathcal{L}_2)$ if and only if $t|_{\mathrm{alph}(\mathcal{L}_i)}$ is a trace of $\mathcal{L}_i$ for both $i \in \{1,2\}$.*
2. *If t is a sequence over $\{\zeta(\alpha) \mid \alpha \in \mathrm{alph}(\mathcal{L}_1)\}$, then t is a trace of $\zeta(\mathcal{L}_1)$ if and only if there is a trace $\alpha_1\alpha_2\cdots\alpha_n$ of $\mathcal{L}_1$ such that $t = \zeta(\alpha_1)\zeta(\alpha_2)\cdots\zeta(\alpha_n)$, where $n \in \mathbb{N}$ and $\alpha_i \in \mathbb{V}$ for all $i \in \{1,2,\ldots,n\}$.*
3. *If t is a sequence over $\mathrm{alph}(\mathcal{L}_1) \setminus \Lambda$, then t is a trace of $(\mathcal{L}_1 \setminus \Lambda)$ if and only if there is a trace t' of $\mathcal{L}_1$ such that $t = t' \setminus \Lambda$.*

See the appendix for a proof.

Now, let us consider a system with two users competing for an access to a shared resource, where the use of the resource is controlled by read and write locks. At most one user can have the write lock at a time, but both the users can hold the read lock simultaneously. Moreover, a user can upgrade its read lock to the write one if the other user does not hold any lock on the resource.

The behaviour of one user is already captured in the LTS $User_1$. A model for the behaviour of other user, denoted by $User_2$, can be obtained from $User_1$ by renaming the actions of the form (u_1, a) to actions (u_2, a). The locking mechanism is modelled as an LTS *Lock* given in Figure 3, and the resource is represented only at the level of actions indicating its usage. The system model, referred to by *Sys*, is obtained by composing the LTSs representing the users and the lock in parallel. Hence, *Sys* is the LTS $((User_1 \parallel User_2) \parallel Lock)$, the reachable part of which is shown in Figure 4.

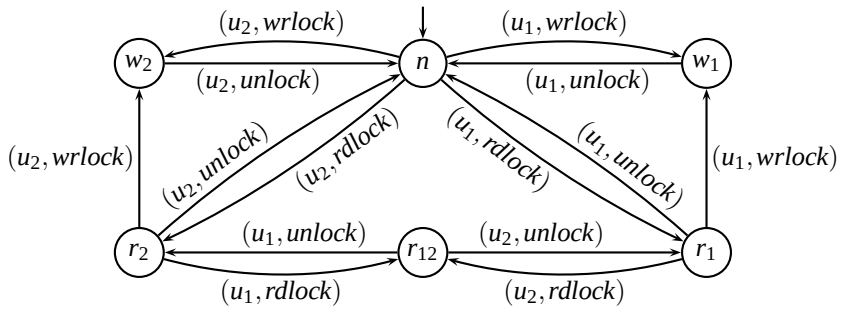

Fig 3. LTS *Lock* **representing the behaviour of the lock that controls an access to the shared resource.**

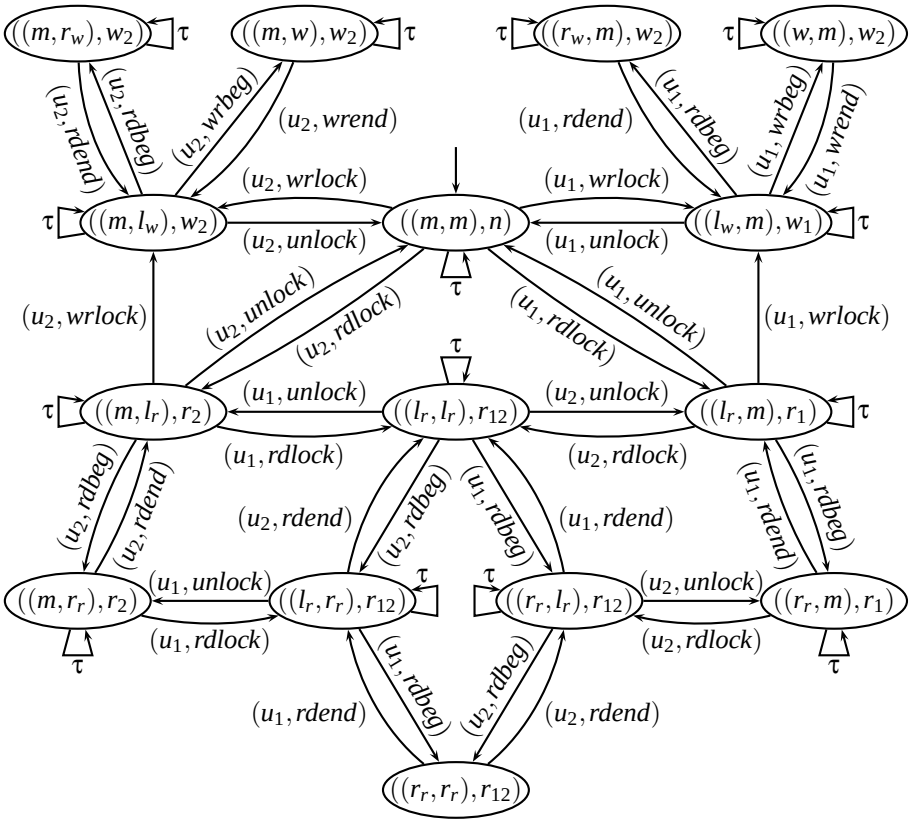

Fig 4. The reachable part of the LTS *Sys*.

3.3 Traces Refinement

In addition to systems, LTSs can naturally express correctness properties, too. Therefore, to carry out verification, one needs to compare LTSs, and a natural way to do it is to compare the traces of LTSs. One writes $\mathcal{L}_1 \succeq_{\text{tr}} \mathcal{L}_2$ to denote that LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$ have the same alphabet and $\text{tr}(\mathcal{L}_2) \subseteq \text{tr}(\mathcal{L}_1)$ (Hoare 1985). Furthermore, we write $\mathcal{L}_1 =_{\text{tr}} \mathcal{L}_2$ to mean that $\mathcal{L}_1 \succeq_{\text{tr}} \mathcal{L}_2$ and $\mathcal{L}_2 \succeq_{\text{tr}} \mathcal{L}_1$.

Lemma 4.

1. *Relation* $\succeq_{\text{tr}}$ *is a preorder in the set of LTSs.*
2. *Relation* $=_{\text{tr}}$ *is an equivalence in the set of LTSs.*

If $\mathcal{L}_1 \succeq_{\text{tr}} \mathcal{L}_2$, we say that $\mathcal{L}_2$ is a *traces refinement* of $\mathcal{L}_1$. Furthermore, if $\mathcal{L}_1 =_{\text{tr}} \mathcal{L}_2$, then $\mathcal{L}_1$ and $\mathcal{L}_2$ are called *traces equivalent* or one says that $\mathcal{L}_1$ is *traces equivalent to* $\mathcal{L}_2$.

In practice, when the specification and the system are modelled respectively as LTSs $\mathcal{L}_{Spec}$ and $\mathcal{L}_{Sys}$, one is usually interested in the related refinement checking problem.

Problem 5 (Traces Refinement)**.**
Instance: LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$.
Question: Is $\mathcal{L}_1 \succeq_{\text{tr}} \mathcal{L}_2$?

If $\mathcal{L}_{Sys}$ is a traces refinement of $\mathcal{L}_{Spec}$, then the system cannot do anything more than the specification does. This way, it is not possible to prove that a system does something, but one can still show the absence of some unwanted behaviour. Therefore, the traces refinement is only applicable in proving the so called *safety properties* (Alpern & Schneider 1985).

Obviously, Traces Refinement cannot be automatically solved in general, but for finite LTSs this is possible (Roscoe 1997).

Proposition 6. *Traces Refinement is decidable for finite LTSs.*

Hence, systems and specifications representable as finite LTSs can be automatically verified.

As an example, consider an LTS *Spec* of Figure 5, which formally states that if one user is writing to the resource, then the other does not access it at all. However, before it makes sense to compare the system model against the specification, the actions related

to locking have to be hidden. Hence, to find out whether the system works correctly, we ask whether *Sys* after hiding the visible actions in the set

$$\Lambda := \{u_1, u_2\} \times \{rdlock, wrlock, unlock\}$$

is a traces refinement of *Spec*.

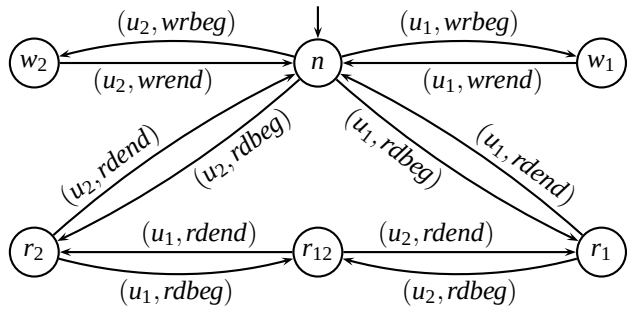

Fig 5. LTS *Spec*, the formal mutual exclusion property.

As both $(Sys \setminus \Lambda)$ and *Spec* are finite LTSs, the verification could be done automatically. However, in this case, it is also easy to manually check that $(Sys \setminus \Lambda)$ and *Spec* have the same alphabet and every trace of $(Sys \setminus \Lambda)$ is a trace of *Spec*. Hence, $(Sys \setminus \Lambda)$ is a traces refinement of *Spec*, and actually also the opposite trace refinement holds, which means that the LTSs are traces equivalent. Because the system cannot do anything more than the specification does, it is not possible that the users access the resource simultaneously. Hence, the system is correct in the sense of the specification.

3.4 Algebra of LTSs

In real-life, the system above would probably have any number of users competing for an access to a number of resources. However, to model such a system and the corresponding specification, one would need two LTSs for each number of users and resources, which leads to the following problem.

Problem 7 (Infinite Family)**.**
Instance: A family $\{(\mathcal{L}_{1,i}, \mathcal{L}_{2,i})\}_{i \in I}$ of pairs of LTSs.
Question: Is $\mathcal{L}_{1,i} \succeq_{\text{tr}} \mathcal{L}_{2,i}$ for all $i \in I$?

Obviously, one cannot solve the problem algorithmically in a general case. Nevertheless, as such systems are very common, it would be useful to be able to solve the

problem in practical situations, at least. Fortunately, in these cases the infinite family $\{(\mathcal{L}_{1,i}, \mathcal{L}_{2,i})\}_{i \in I}$ of pairs of LTSs is not arbitrary, but typically arises from a parameterised specification and system descriptions. Hence, the pairs of LTSs share a common structure, which can be exploited to reduce the number of refinement checks needed. The properties of LTS operators that are given in the following four propositions and derived from the ones in Hoare (1985) and Roscoe (1997) turn out to be useful in this effort.

Proposition 8. *Let $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ be LTSs.*

1. $(\mathcal{L}_1 \parallel \mathcal{L}_2) =_{\mathrm{tr}} (\mathcal{L}_2 \parallel \mathcal{L}_1)$.
2. $((\mathcal{L}_1 \parallel \mathcal{L}_2) \parallel \mathcal{L}_3) =_{\mathrm{tr}} (\mathcal{L}_1 \parallel (\mathcal{L}_2 \parallel \mathcal{L}_3))$.
3. $(\mathcal{L}_1 \parallel \mathcal{L}_2) =_{\mathrm{tr}} \mathcal{L}_1$, *if* $\mathrm{alph}(\mathcal{L}_2) = \emptyset$.
4. $(\mathcal{L}_1 \parallel \mathcal{L}_1) =_{\mathrm{tr}} \mathcal{L}_1$.

The proposition follows straightforwardly from Lemma 3.

Informally, the proposition states that (1) the parallel composition is commutative, and (2) associative, (3) every LTS with the empty alphabet is an identity element and (4) every LTS is an idempotent element with respect to the parallel composition.

Because of commutativity and associativity, the order in which LTSs are composed in parallel is insignificant from the viewpoint of traces equivalence. That is why there is no need to use parenthesis in successive application of the parallel composition, *i.e.* we can simply write $(\mathcal{L}_1 \parallel \mathcal{L}_2 \parallel \mathcal{L}_3)$ instead of $((\mathcal{L}_1 \parallel \mathcal{L}_2) \parallel \mathcal{L}_3)$ or $(\mathcal{L}_1 \parallel (\mathcal{L}_2 \parallel \mathcal{L}_3))$, provided we are interested in the alphabet and the traces only. For the same reason, there is no need to explicitly specify a total order relation over the finite index set when using the generic form of the parallel composition, because all the orders will give the same result from the viewpoint of the alphabet and traces.

Proposition 9. *Let ζ_1, ζ_2 be functions: $\mathbb{V} \mapsto \mathbb{V}$, $\mathcal{L}_1, \mathcal{L}_2$ LTSs, and Λ a set of visible actions.*

1. $\zeta_1(\mathcal{L}_1 \parallel \mathcal{L}_2) = \zeta_1(\mathcal{L}_1) \parallel \zeta_1(\mathcal{L}_2)$, *if ζ_1 is a bijection.*
2. $\zeta_1(\mathcal{L}_1 \setminus \Lambda) = \zeta_1(\mathcal{L}_1) \setminus \{\zeta_1(\alpha) \mid \alpha \in \Lambda\}$, *if ζ_1 is a bijection.*
3. $\zeta_1(\zeta_2(\mathcal{L}_1)) = (\zeta_2 \circ \zeta_1)(\mathcal{L}_1)$.

It is easy to check that in each of these cases, the LTSs on both sides of the equivalence are (structurally) the same. Note that structural equality trivially implies the traces equivalence.

The proposition says that bijective renaming can be pushed inside other operators and successive renaming operators can be combined without affecting the structure of the resulting LTS. Note that Items 1 and 2 do not hold for arbitrary functions even if the equivalence was replaced by the traces refinement. For example, consider an LTS $\mathcal{L}_{ab}$ in Figure 6 with the alphabet $\{a,b\}$, and assume a function $\zeta : \mathbb{V} \mapsto \mathbb{V}$ that preserves an atom c and maps both the atoms a and b to a. The traces of $\zeta(\mathcal{L}_{ac} \parallel \mathcal{L}_{bc})$ is the set $\{t' \preceq t \mid t \in (aac)^*\}$, whereas the traces of $\zeta(\mathcal{L}_{ac}) \parallel \zeta(\mathcal{L}_{bc})$ is the set $\{t' \preceq t \mid t \in (ac)^*\}$. Hence, the sets of the traces of the LTSs are incomparable, which implies that $\zeta(\mathcal{L}_{ac} \parallel \mathcal{L}_{bc})$ is not a traces refinement of $\zeta(\mathcal{L}_{ac}) \parallel \zeta(\mathcal{L}_{bc})$, or vice versa. To see that bijectivity is necessary in the case of Item 2 too, consider LTSs $\zeta(\mathcal{L}_{ab} \setminus \{a\})$ and $\zeta(\mathcal{L}_{ab}) \setminus \{\zeta(a)\}$. The LTSs have different alphabets, which means that they cannot be related by the traces refinement.

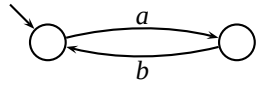

Fig 6. LTS $\mathcal{L}_{ab}$.

Proposition 10. *Let $\mathcal{L}_1, \mathcal{L}_2$ be LTSs, and Λ_1, Λ_2 sets of visible actions.*

1. $(\mathcal{L}_1 \setminus \Lambda_1) \parallel (\mathcal{L}_2 \setminus \Lambda_1) \succeq_{\mathrm{tr}} (\mathcal{L}_1 \parallel \mathcal{L}_2) \setminus \Lambda_1$.
2. $(\mathcal{L}_1 \setminus \Lambda_1) \parallel (\mathcal{L}_1 \setminus \Lambda_2) \succeq_{\mathrm{tr}} \mathcal{L}_1$, *if* $\Lambda_1 \cap \Lambda_2 = \emptyset$.
3. $\mathcal{L}_1 \setminus \Lambda_1 = \mathcal{L}_1$, *if* Λ_1 *and* $\mathrm{alph}(\mathcal{L}_1)$ *are disjoint.*
4. $(\mathcal{L}_1 \setminus \Lambda_1) \setminus \Lambda_2 = \mathcal{L}_1 \setminus (\Lambda_1 \cup \Lambda_2)$.

See the appendix for a proof

The proposition states that (1) pushing hiding inside the parallel composition results in a greater LTS in the preorder, (2) applying first two different hiding operators to an LTS $\mathcal{L}$, and then composing the resulting LTSs in parallel gives an LTS greater than $\mathcal{L}$, provided the hiding sets are disjoint, (3) hiding a set of actions disjoint from the alphabet of an LTS does not change the LTS, and (4) successive hiding operators can be combined or a single hiding operator split without affecting the structure of the resulting LTS.

Note that in the first two cases, the opposite traces refinements, *i.e.* the traces equivalences, do not generally hold. To see it, consider the LTSs $(\mathcal{L}_{ab} \setminus \{b\}) \parallel (\mathcal{L}_{ba} \setminus \{b\})$ and $(\mathcal{L}_{ab} \parallel \mathcal{L}_{ba}) \setminus \{b\}$, where $\mathcal{L}_{ab}$ is the LTS in Figure 6. The set of the traces of the former LTS is a^*, whereas the latter has only the empty one, which means that the traces

equivalence does not hold in the first case. To see it does not hold in the second case either, consider the LTS $\mathcal{L}_{abc}$ in Figure 7 and the LTS $(\mathcal{L}_{abc} \setminus \{b,c\}) \parallel (\mathcal{L}_{abc} \setminus \{a\})$. The traces of the latter LTS contain the arbitrary sequences of a, b and c, but the former LTS cannot execute the same action twice running. Hence, there is no traces equivalence between these LTSs either.

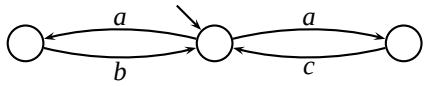

Fig 7. LTS $\mathcal{L}_{abc}$.

Proposition 11. *Let $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}'_1, \mathcal{L}'_2$ be LTSs, ζ a function: $\mathbb{V} \mapsto \mathbb{V}$ and Λ a set of visible actions.*

1. *If $\mathcal{L}_i \succeq_{\mathrm{tr}} \mathcal{L}'_i$ for both $i \in \{1,2\}$, then $\mathcal{L}_1 \parallel \mathcal{L}_2 \succeq_{\mathrm{tr}} \mathcal{L}'_1 \parallel \mathcal{L}'_2$.*
2. *If $\mathcal{L}_1 \succeq_{\mathrm{tr}} \mathcal{L}_2$, then $\zeta(\mathcal{L}_1) \succeq_{\mathrm{tr}} \zeta(\mathcal{L}_2)$.*
3. *If $\mathcal{L}_1 \succeq_{\mathrm{tr}} \mathcal{L}_2$, then $\mathcal{L}_1 \setminus \Lambda \succeq_{\mathrm{tr}} \mathcal{L}_2 \setminus \Lambda$.*

See the appendix for a proof.

The proposition states that the LTS operators preserve the traces refinement, which together with Lemma 4 implies that the traces refinement is a precongruence and the traces equivalence a congruence. The result also means that while composing a system and specification, parallel compositions can be restricted to their reachable parts without affecting the traces and alphabet of the system and specification.

The converses of the claims of Proposition 11 do not hold. To see it, let $\mathcal{L}'$ be a single state LTS with alphabet $\{a,b\}$ and no transitions, $\mathcal{L}_{ab}$ the LTS in Figure 6, and ζ a function: $\mathbb{V} \mapsto \mathbb{V}$ mapping both the atoms a and b to a. Now, $\mathcal{L}_{ab} \parallel \mathcal{L}'$ and $\mathcal{L}_{ba} \parallel \mathcal{L}'$, $\zeta(\mathcal{L}_{ab})$ and $\zeta(\mathcal{L}_{ba})$, as well as $\mathcal{L}_{ab} \setminus \{b\}$ and $\mathcal{L}_{ba} \setminus \{b\}$ are traces equivalent but $\mathcal{L}_{ba}$ is clearly not a traces refinement of $\mathcal{L}_{ab}$.

4 Process Algebraic Modelling and Verification of Parameterised Systems

In the previous chapter, it was mentioned that LTS operators can be made use of not only in modelling but in verification, too. Based on this, we propose a novel method, called the precongruence reduction, to check the correctness of parameterised systems with respect to parameterised specifications that are modelled in a compositional way. The precongruence reduction is first introduced by Siirtola & Kortelainen (2009b) and the results concerning compositional modelling by Siirtola & Kortelainen (2009a).

4.1 The Precongruence Reduction

Suppose that we are given a parameterised system and a related parameterised specification as an instance $\{(Spec_i, Sys_i)\}_{i \in I}$ of Infinite Family, where I denotes an (infinite) set of parameter values, and $Spec_i$ and Sys_i represent respectively the specification and the system with the parameter value $i \in I$. Using existing tools for finite-state systems, we can basically prove the system correct with respect to the specification for any finite number of parameter values. By exploiting the precongruence property, it is possible to generalise the result to the specification-system instances that can be obtained from the examined ones by the application of LTS operators. In the best case, the whole system can be proven correct this way, which leads to a novel way of obtaining cut-offs for parameterised systems and specifications. The method is called *the precongruence reduction* (Siirtola & Kortelainen 2009b) and captured in the following theorem.

Theorem 12. *Let $\{(Spec_i, Sys_i)\}_{i \in I}$ be an instance of Infinite Family, and I' a subset of I such that for every $i \in I \setminus I'$ there are a positive integer n, an n-place function f that is a composition of renaming, hiding and parallel composition operators, and elements $i'_1, i'_2, \ldots, i'_n \in I'$ such that*

$$Spec_i \succeq_{\mathrm{tr}} f(Spec_{i'_1}, Spec_{i'_2}, \ldots, Spec_{i'_n}) \text{ and } f(Sys_{i'_1}, Sys_{i'_2}, \ldots, Sys_{i'_n}) \succeq_{\mathrm{tr}} Sys_i \,.$$

Then the answer to $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$ is the same as the answer to $\{(Spec_i, Sys_i)\}_{i \in I}$.

Proof. Since I' is a subset of I, it is clear that if the answer to $\{(Spec_i, Sys_i)\}_{i \in I}$ is positive then the answer to $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$ is positive, too.

Let us then assume that the answer to $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$ is positive, and let $i \in I \setminus I'$. It means that $Spec_{i'} \succeq_{\mathrm{tr}} Sys_{i'}$ for all $i' \in I'$, and there are a positive integer n, an n-place function f that is a composition of renaming, hiding and parallel composition operators, and $i'_1, i'_2, \ldots, i'_n \in I'$ such that

$$Spec_i \succeq_{\mathrm{tr}} f(Spec_{i'_1}, Spec_{i'_2}, \ldots, Spec_{i'_n}) \text{ and } f(Sys_{i'_1}, Sys_{i'_2}, \ldots, Sys_{i'_n}) \succeq_{\mathrm{tr}} Sys_i \,.$$

Because the traces refinement is preserved under the application of LTS operators (Proposition 11), it is easy to show by induction on the structure of f that

$$f(Spec_{i'_1}, Spec_{i'_2}, \ldots, Spec_{i'_n}) \succeq_{\mathrm{tr}} f(Sys_{i'_1}, Sys_{i'_2}, \ldots, Sys_{i'_n}) \,.$$

By above, it means that

$$Spec_i \succeq_{\mathrm{tr}} f(Spec_{i'_1}, Spec_{i'_2}, \ldots, Spec_{i'_n}) \succeq_{\mathrm{tr}} f(Sys_{i'_1}, Sys_{i'_2}, \ldots, Sys_{i'_n}) \succeq_{\mathrm{tr}} Sys_i \,,$$

which, by the transitivity of the traces refinement (Lemma 4), implies that $Spec_i \succeq_{\mathrm{tr}} Sys_i$. Hence, we have proved that if the answer to $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$ is positive, then the answer to $\{(Spec_i, Sys_i)\}_{i \in I}$ is positive, too. □

In practice, we do not apply the theorem in its full generality. First of all, we assume that the instances $\{(Spec_i, Sys_i)\}_{i \in I}$ of Infinite Family are closed under the application of certain bijective renaming operators. In other words, whenever $\zeta : \mathbb{V} \mapsto \mathbb{V}$ is a bijection (of a certain kind), then $(\zeta(Spec_i), \zeta(Sys_i))$ is in $\{(Spec_i, Sys_i)\}_{i \in I}$ for all $i \in I$. Actually, this is not a restriction, because by Theorem 12, closing an instance of Infinite Family under the application of a bijective renaming operator does not change its answer.

Secondly, we apply the theorem using only functions f that are compositions of bijective renaming and parallel composition operators. Intuitively, the reason is that in order to minimise the cost of solving $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$, we want to pick I' in such a way that $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$ consists of the smallest elements in $\{(Spec_i, Sys_i)\}_{i \in I}$. Hence, we need to be able to approximate big specification-system instances with the aid of smaller ones using the LTS operators. As hiding and non-bijective renaming can make their parameters only smaller, there is intuitively no point to use them in this effort.

From the technical side, the reason for concentrating on the compositions of bijective renaming and parallel composition operators only is that such functions can always be represented in the form $\|_{j=1}^{n} \zeta_j(\cdot_j)$, where n is a positive integer and ζ_j is a bijection:

$\mathbb{V} \mapsto \mathbb{V}$ for every $j \in \{1, 2, \ldots, n\}$. This fact follows from Proposition 9, which says that bijective renaming can be pushed inside parallel compositions and successive bijective renamings can be combined into one. Because we may assume the instances of Infinite Family to be closed under the application of bijective renaming operators needed to construct large specification-system instances, the application of Theorem 12 can be divided into two simpler parts, where bijective renaming operators or (generic forms of) parallel composition are used as functions f.

Hence, given an instance $\{(Spec_i, Sys_i)\}_{i \in I}$ of Infinite Family that is closed under (certain kind of) bijective renaming, we proceed as follows. First, based on our understanding about the verification problem, we pick a finite subset I' of I, and check whether $Spec_{i'} \succeq_{\mathrm{tr}} Sys_{i'}$ for all $i' \in I'$ using existing refinement checking tools. If this is not true, we know that also the answer to $\{(Spec_i, Sys_i)\}_{i \in I}$ is negative.

On the other hand, if it holds, we extend $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$ by adding all LTS pairs $(Spec_i, Sys_i)$, where $i \in I \setminus I'$ and which can be obtained by bijective renaming from those in $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$. It results in an instance $\{(Spec_{i''}, Sys_{i''})\}_{i'' \in I''}$ of Infinite Family, where I'' is obviously a subset of I and typically infinite. By Theorem 12, the answer to this instance is positive, too.

Next, we append to $\{(Spec_{i''}, Sys_{i''})\}_{i'' \in I''}$ all LTS pairs $(Spec_i, Sys_i)$ such that $i \in I \setminus I''$ and $(Spec_i, Sys_i)$ can be approximated as the parallel composition of those in $\{(Spec_{i''}, Sys_{i''})\}_{i'' \in I''}$, *i.e.* there are $n \in \mathbb{Z}_+$ and $i_1, i_2, \ldots, i_n \in I''$ such that

$$Spec_i \succeq_{\mathrm{tr}} Spec_{i_1} \parallel Spec_{i_2} \parallel \cdots \parallel Spec_{i_n} \text{ and } Sys_{i_1} \parallel Sys_{i_2} \parallel \cdots \parallel Sys_{i_n} \succeq_{\mathrm{tr}} Sys_i \,.$$

If we end up with the whole instance $\{(Spec_i, Sys_i)\}_{i \in I}$, by Theorem 12, we know that its answer must be positive as well. Otherwise, we need to refine our guess for I', or the verification technique is not applicable.

Note that one can also apply the procedure in the reverse order. Then one starts with an instance $\{(Spec_i, Sys_i)\}_{i \in I}$ and ends up with an instance $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$, where I' is a subset of I. In other words, one uses the theorem to reduce the parameter set I to its subset. If I' is finite, then the original instance can be solved by deciding $\{(Spec_{i'}, Sys_{i'})\}_{i' \in I'}$ using existing tools. As the parameter values are typically ordered, the approach leads to a novel way of obtaining cut-offs for parameterised systems and specifications. Recall that a cut-off is a parameter value such that a parameterised system can be verified by checking the correctness of the system for the values less than or equal to the cut-off.

Compared with earlier works on cut-offs (Bouajjani *et al.* 2008, Clarke *et al.* 2004, Emerson & Namjoshi 2003, Emerson & Kahlon 2000, 2002, 2003a, 2004, Konnov & Zakharov 2005, Li *et al.* 1994, Pyssysalo 1996, Valmari & Tienari 1991), the fundamental difference is that we take also the specification instances into account. Moreover, in the precongruence reduction, generalisation is justified by the fact that a large specification-system instance can be obtained from finitely many small ones using correctness preserving operations, whereas other methods are based on showing that certain behaviours of a large system instance are covered by the behaviours of a single small instance. An exception is the work of Clarke *et al.* (2004) where multiple small networks are used to cover the behaviours of a large one, but also in this case, the method makes use of the system instances only. Additionally, Clarke *et al.* (2004) provide only an upper bound for the size of a network graph but does not say which networks one should check.

4.2 Compositional Modelling

A successful use of the precongruence reduction typically necessitates a compositional modelling technique that is widely used in practice; a parameterised system and specification have to be constructed from smaller parts using a parallel composition such that each part represents the behaviour of the system or the specification from the viewpoint of certain components.

Ideally, a compositionally modelled system and specification correspond precisely to the original system and specification, respectively. However, when proving a system correct, it is safe to use a system model that covers at least all the behaviours of the original system, and a specification that covers at most the behaviours of the original specification. This way, one can get false negative verification results, but if the system model is correct with respect to the specification model, also the original system is guaranteed to meet its specification. Hence, to successfully use the compositional verification technique, one should know when the parallel composition of system and specification parts results in an LTS greater or smaller in the preorder.

Proposition 13. *Let $\mathcal{L}$ be an LTS, I a finite index set and $\mathcal{L}_i$ an LTS for all $i \in I$ such that* $\mathrm{alph}(\mathcal{L}) = \bigcup_{i\in I} \mathrm{alph}(\mathcal{L}_i)$. *Then*

1. $\mathcal{L} \succeq_{\mathrm{tr}} \|_{i\in I} \mathcal{L}_i$ *if and only if whenever t is a (minimal) sequence over* $\mathrm{alph}(\mathcal{L})$ *such that $t \notin \mathrm{tr}(\mathcal{L})$, there exists $i \in I$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)}$ is not a trace of $\mathcal{L}_i$, and*

2. $\|_{i\in I}\mathcal{L}_i \succeq_{\mathrm{tr}} \mathcal{L}$ *if and only if* $\mathcal{L}_i \succeq_{\mathrm{tr}} \mathcal{L}\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_i))$ *for all* $i\in I$.

Proof. Let us first assume that $\mathcal{L}\succeq_{\mathrm{tr}}\|_{i\in I}\mathcal{L}_i$ and let t be a sequence over $\mathrm{alph}(\mathcal{L})$ such that $t\notin\mathrm{tr}(\mathcal{L})$. Without the loss of generality, we may assume that t is minimal such a trace. If $t|_{\mathrm{alph}(\mathcal{L}_i)}$ was a trace of $\mathcal{L}_i$ for all $i\in I$, then t would a trace of $\|_{i\in I}\mathcal{L}_i$, which is impossible. This can be easily shown by induction on $|I|$ using Lemma 3. Therefore, there must be $i\in I$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)}$ is not a trace of $\mathcal{L}_i$.

Let us then assume that whenever t is a (minimal) sequence over $\mathrm{alph}(\mathcal{L})$ such that $t\notin\mathrm{tr}(\mathcal{L})$, there is $i\in I$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)}$ is not a trace of $\mathcal{L}_i$. Because the alphabet of $\mathcal{L}$ is $\bigcup_{i\in I}\mathrm{alph}(\mathcal{L}_i)$ by assumption, it is sufficient to show that every trace of $\|_{i\in I}\mathcal{L}_i$ is a trace of $\mathcal{L}$, too. If $t\in\mathrm{tr}(\|_{i\in I}\mathcal{L}_i)$, then by Lemma 3, $t|_{\mathrm{alph}(\mathcal{L}_i)}\in\mathrm{tr}(\mathcal{L}_i)$ for all $i\in I$. On the other hand, if $t\notin\mathrm{tr}(\mathcal{L})$, then by above there is $i\in I$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)}$ is not a trace of $\mathcal{L}_i$, which is a contradiction. Therefore, every trace of $\|_{i\in I}\mathcal{L}_i$ must be also a trace of $\mathcal{L}$, which implies that $\mathcal{L}\succeq_{\mathrm{tr}}\|_{i\in I}\mathcal{L}_i$. Hence, the first part of the proposition holds.

To prove the latter claim, let us first assume that $\|_{i\in I}\mathcal{L}_i\succeq_{\mathrm{tr}}\mathcal{L}$. If $j\in I$, then by Proposition 11,

$$(\mathop{\|}_{i\in I}\mathcal{L}_i)\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_j))\succeq_{\mathrm{tr}}\mathcal{L}\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_j))\,.$$

Let us now consider the LTSs $\mathcal{L}_j$ and $(\|_{i\in I}\mathcal{L}_i)\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_j))$. Clearly, their alphabets are the same. Moreover, if t is a trace of the latter LTS, then by Item 3 of Lemma 3, there is a trace t' of $\|_{i\in I}\mathcal{L}_i$ such that

$$t=t'\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_j))=t'|_{\mathrm{alph}(\mathcal{L}_j)}\,.$$

On the other hand, by the same lemma, $t'|_{\mathrm{alph}(\mathcal{L}_j)}$ is also a trace of $\mathcal{L}_j$, which implies that $t\in\mathrm{tr}(\mathcal{L}_j)$. Hence,

$$\mathcal{L}_j\succeq_{\mathrm{tr}}(\mathop{\|}_{i\in I}\mathcal{L}_i)\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_j))\,,$$

which by the transitivity of $\succeq_{\mathrm{tr}}$ means that $\mathcal{L}_j\succeq_{\mathrm{tr}}\mathcal{L}\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_j))$.

To prove the opposite claim, let us assume that $\mathcal{L}_i\succeq_{\mathrm{tr}}\mathcal{L}\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_i))$ for all $i\in I$. By repeated application of Proposition 11, it implies that

$$\mathop{\|}_{i\in I}\mathcal{L}_i\succeq_{\mathrm{tr}}\mathop{\|}_{i\in I}(\mathcal{L}\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_i)))\,.$$

As $\mathrm{alph}(\mathcal{L})=\bigcup_{i\in I}\mathrm{alph}(\mathcal{L}_i)$, the alphabet of the latter LTS is

$$\bigcup_{i\in I}(\mathrm{alph}(\mathcal{L})\setminus(\mathrm{alph}(\mathcal{L})\setminus\mathrm{alph}(\mathcal{L}_i)))=\bigcup_{i\in I}(\mathrm{alph}(\mathcal{L})\cap\mathrm{alph}(\mathcal{L}_i))=\mathrm{alph}(\mathcal{L})\,,$$

i.e. the same as the alphabet of $\mathcal{L}$. Moreover, if t is a trace of $\mathcal{L}$, then by Item 3 of Lemma 3, $t|_{\mathrm{alph}(\mathcal{L}_i)}$ is a trace of $\mathcal{L} \setminus (\mathrm{alph}(\mathcal{L}) \setminus \mathrm{alph}(\mathcal{L}_i))$ for every $i \in I$. By Item 1 of the same lemma, it implies that then t is a trace of $\|_{i \in I}(\mathcal{L} \setminus (\mathrm{alph}(\mathcal{L}) \setminus \mathrm{alph}(\mathcal{L}_i)))$. By the transitivity of the traces refinement, it means that $\|_{i \in I} \mathcal{L}_i \succeq_{\mathrm{tr}} \mathcal{L}$. Hence, also the latter claim holds and the proposition is proved. □

Because in formal verification, it is more important to avoid false positive than false negative verification results, the former part of the proposition applies mainly to specification and the latter one to system instances. Hence, the proposition informally states that when creating the specification from smaller parts, every illegal behaviour must be forbidden by some of the parts, and every part of the system model must cover all the behaviours of the system from the viewpoint of its alphabet. However, if one wants to avoid false negative verification results as well, then it is also necessary that every part of the specification covers all the allowed behaviours of the specification from the viewpoint of its alphabet, and every behaviour not present in the original system is blocked by some of the system parts.

4.3 Modelling a Parameterised Shared Resource System

To see how compositional modelling and the precongruence reduction are applied in practice, let us consider a shared resource system of Figure 8 with an arbitrary number of users and shared resources, where a user may read or write to any resource after obtaining the corresponding lock. The resource itself has no mechanism for concurrency control and several users can hold a lock on the resource simultaneously only if all of them have the read lock. The goal is to formally model the system and the specification, and prove that in our construction it is not possible for a user to access a resource if someone else is writing to it.

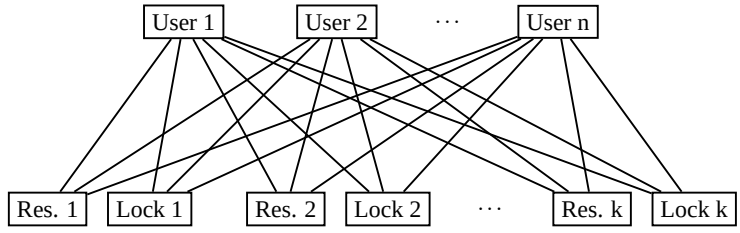

Fig 8. The system of n users and k shared resources with locks.

The system cannot be represented as a single finite LTS, but a different model is needed for each number of users and resources. That is why we take the sets of user and resource identities, denoted respectively by U and R, as the parameters of our model. We assume that U and R are disjoint, finite and non-empty sets of atoms, also disjoint from other atoms used in the model.

The behaviour of a user $u \in U$ from the viewpoint of a single resource $r \in R$ is captured in an LTS $User(u,r)$ in Figure 9. Obtaining the read and write lock on the resource is modelled respectively by actions $(u,r,rdlock)$ and $(u,r,wrlock)$, and releasing the resource by an action $(u,r,unlock)$, where $rdlock, wrlock, unlock$ are atoms denoting respectively obtaining the read and write lock, and releasing the lock. Because the actions are instantaneous, reading the resource is modelled using two of them, namely $(u,r,rdbeg)$ and $(u,r,rdend)$, where *rdbeg* and *rdend* are atoms denoting respectively the beginning and the end of the read event. Similarly, writing to the resource is modelled as actions $(u,r,wrbeg)$ and $(u,r,wrend)$. The invisible actions represent the user's other activities which may take place at any time.

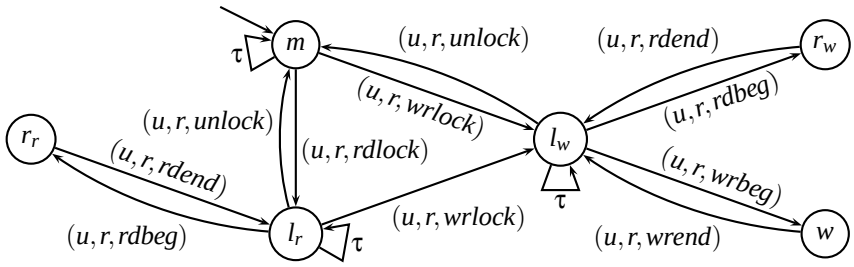

Fig 9. LTS $User(u,r)$ that represents the behaviour of a user u from the viewpoint of a resource r (Siirtola & Kortelainen 2009a, p. 572, published by permission of Springer).

Clearly, the LTS $User(u,r)$ represents the behaviour of the user from the viewpoint of its alphabet. Now, Proposition 13 implies that the behaviour of the user u can be over-approximated by the parallel composition of all LTSs $User(u,r)$ as r ranges over R. As a system or its part can be replaced with a greater LTS in the preorder $\succeq_{\mathrm{tr}}$ without the risk of a false positive verification result, we can adopt $\|_{r \in R}\, User(u,r)$ as the model of the user u.

Similarly, the behaviour of a lock is first modelled from the viewpoint of a user $u \in U$ and a resource $r \in R$, which results in an LTS $Lock_1(r,u)$ in Figure 10. However, as the locking sequences of transactions are mutually dependent, the lock has to be

modelled from the viewpoint of two different users $u_1, u_2 \in U$ and a resource, too. This behaviour is captured in an LTS $Lock_2(r, u_1, u_2)$ in Figure 11. Because both the LTSs obviously capture all behaviours from the viewpoint of their alphabets, by Proposition 13, the behaviour of the lock for the resource r can be over-approximated as the parallel composition

$$\Big(\mathop{\Big\|}_{\substack{(u_1,u_2)\in U\times U\\ u_1\neq u_2}} Lock_2(r,u_1,u_2)\Big) \,\|\, \Big(\mathop{\Big\|}_{u\in U} Lock_1(r,u)\Big)\,,$$

which we thereby take as the lock model.

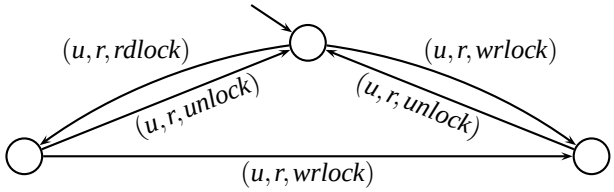

Fig 10. LTS $Lock_1(u,r)$ representing the behaviour of the lock for a resource r from the viewpoint of a user u.

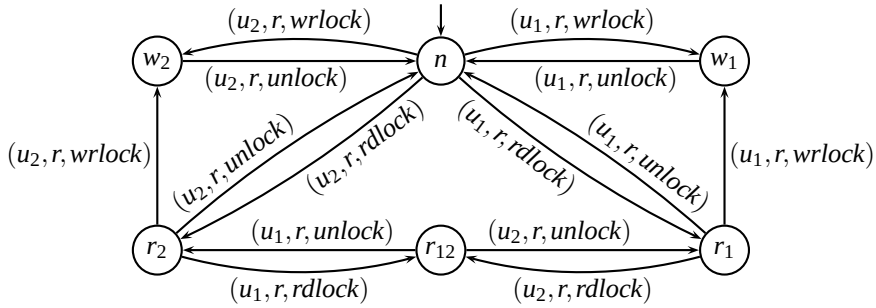

Fig 11. LTS $Lock_2(r,u_1,u_2)$ representing the behaviour of the lock for a resource r from the viewpoint of two different users u_1, u_2 (Siirtola & Kortelainen 2009a, p. 572, published by permission of Springer).

Now, we have modelled the system from the viewpoint of a single user and a single lock. As the user and lock models cover all the actions of the system, we obtain an over-approximation of system behaviours by composing all of the user and lock models in parallel. Because a system can be replaced with a more general one without the risk

of a false positive verification result, we adopt the LTS

$$(\underset{u\in U}{\|}\underset{r\in R}{\|} User(u,r)) \,\|\, (\underset{r\in R}{\|} (\underset{\substack{(u_1,u_2)\in U\times U\\ u_1\neq u_2}}{\|} Lock_2(r,u_1,u_2)) \,\|\, (\underset{u\in U}{\|} Lock_1(r,u))),$$

denoted by $Srs(U,R)$, as the system model.

The specification is formalised in a similar way as the lock; it is first modelled from the viewpoints of a resource, and one and two users, and then all such partial specifications are composed in parallel. This results in an LTS

$$Mtx(U,R) := \underset{r\in R}{\|} ((\underset{\substack{(u_1,u_2)\in U\times U\\ u_1\neq u_2}}{\|} Prop_2(r,u_1,u_2)) \,\|\, (\underset{u\in U}{\|} Prop_1(r,u))),$$

where $Prop_2(r,u_1,u_2)$ and $Prop_1(r,u)$ are LTSs in respectively Figures 12 and 13. By reasoning like above, $Mtx(U,R)$ is an over-approximation of the formal specification. However, the specification can be safely replaced with a smaller LTS in the preorder only. Hence, before we can adopt $Mtx(U,R)$ as the formal specification, we need to make sure that it does not allow any illegal behaviour.

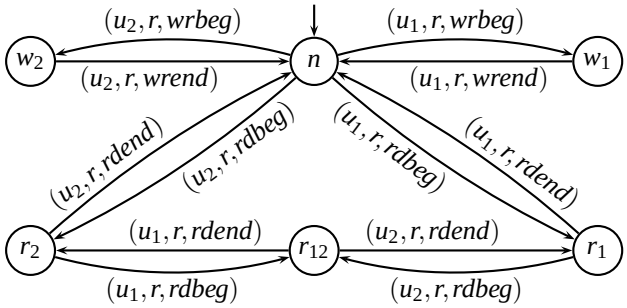

Fig 12. LTS $Prop_2(r,u_1,u_2)$ representing the formal specification from the viewpoint of two different users u_1,u_2 and a resource r (Siirtola & Kortelainen 2009a, p. 572, published by permission of Springer).

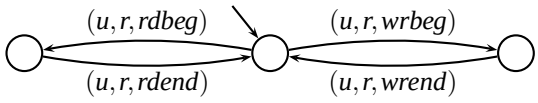

Fig 13. LTS $Prop_1(r,u)$ representing the formal specification from the viewpoint of a user u and a resource r.

To see that $Mtx(U,R)$ does not allow a user to write to a resource that is currently accessed by someone else, suppose the contrary. By Item 1 of Lemma 3, it means that there are $u_1,u_2 \in U$ and $r \in R$ such that $Prop_2(r,u_1,u_2)$ allows u_1 to write to r while u_2 is reading it or writing to it. However, by the construction of $Prop_2(r,u_1,u_2)$ it is clearly impossible, which by Proposition 13 implies that $Mtx(U,R)$ under-approximates the behaviour of the specification. By above, it means that $Mtx(U,R)$ is the precise model of the formal specification.

Finally, we hide the actions irrelevant to the formal specification. Those are the ones related to locking and contained in set

$$La(U,R) := \bigcup_{(u,r)\in U\times R} \{(u,r,rdlock),(u,r,wrlock),(u,r,unlock)\} .$$

Now, we can formally state the related verification problem. We write *Par* for the set of all the possible parameter values, *i.e.* pairs (U,R) such that U and R are disjoint, finite and non-empty sets of atoms, also disjoint from the set

$$\{rdlock, wrlock, unlock, rdbeg, rdend, wrbeg, wrend\} .$$

Hence, we are interested in whether

$$Mtx(U,R) \succeq_{\mathrm{tr}} Srs(U,R) \setminus La(U,R) \text{ for all } (U,R) \in Par.$$

If the refinement holds for all the parameter values, we know that all the behaviours of the system are covered by those of the specification, which means that the system cannot do anything illegal and therefore works as specified.

4.4 Verification by the Precongruence Reduction

Our intuition says that in order to ensure the correctness of the system one should check at least an instance with two users and a resource, and another one with one user and a resource. We write Par' for the set of all pairs $(U',R') \in Par$ such that $|U'| \leq 2$ and $|R'| = 1$. Moreover, for both $i \in \{1,2\}$, let $(U_i,R_i) \in Par'$ such that $|U_i| = i$. Now, using a finite-state refinement checker, like FDR2 (Failures-Divergences Refinement) (Roscoe 1997), one can verify that

$$Mtx(U_i,R_i) \succeq_{\mathrm{tr}} Srs(U_i,R_i) \setminus La(U_i,R_i)$$

holds for both $i \in \{1,2\}$. Hence, at least two instances of the shared resource system, one with two users and a resource, and another one with a single user and a resource, work as specified.

By above, it is actually quite easy to believe that the system works correctly in the presence of any one or two users, and any resource. In other words,

$$Mtx(U,R) \succeq_{\mathrm{tr}} Srs(U,R) \setminus La(U,R)$$

whenever $(U,R) \in Par'$.

To formally justify the claim, let $(U,R) \in Par'$. We show that $Mtx(U,R)$ and $Srs(U,R) \setminus La(U,R)$ can be obtained from respectively $Mtx(U_i,R_i)$ and $Srs(U_i,R_i) \setminus La(U_i,R_i)$, where $i = |U|$, using bijective renaming. By Theorem 12, it proves the claim.

Clearly, there is a bijection $g : \mathbb{A} \mapsto \mathbb{A}$ such that $g(U_i) = U$, $g(R_i) = R$ and g preserves the atoms *rdlock*, *wrlock*, *unlock*, *rdbeg*, *rdend*, *wrbeg* and *wrend*. We write g^* for a function: $\mathbb{V} \mapsto \mathbb{V}$ such that

$$g^*((a_1,a_2,\ldots,a_k)) = (g(a_1),g(a_2),\ldots,g(a_k))$$

for all visible actions $(a_1,a_2,\ldots,a_k)$. It is easy to see that also g^* is a bijection and

$$g^*(Prop_2(r,u_1,u_2)) = Prop_2(g(r),g(u_1),g(u_2)) \,,$$
$$g^*(Prop_1(r,u)) = Prop_1(g(r),g(u))$$

whenever $r \in R_i$ and $u,u_1,u_2 \in U_i$. Therefore, by Proposition 9,

$$\begin{aligned}
&g^*(Mtx(U_i,R_i)) \\
&\quad = g^*(\mathop{\|}_{r\in R_i}((\mathop{\|}_{\substack{(u_1,u_2)\in U_i\times U_i\\ u_1\neq u_2}} Prop_2(r,u_1,u_2)) \,\|\, (\mathop{\|}_{u\in U_i} Prop_1(r,u)))) \\
&\quad = \mathop{\|}_{r\in R_i}((\mathop{\|}_{\substack{(u_1,u_2)\in U_i\times U_i\\ u_1\neq u_2}} g^*(Prop_2(r,u_1,u_2))) \,\|\, (\mathop{\|}_{u\in U_i} g^*(Prop_1(r,u)))) \\
&\quad = \mathop{\|}_{r\in R_i}((\mathop{\|}_{\substack{(u_1,u_2)\in U_i\times U_i\\ u_1\neq u_2}} Prop_2(g(r),g(u_1),g(u_2))) \,\|\, (\mathop{\|}_{u\in U_i} Prop_1(g(r),g(u)))) \\
&\quad = \mathop{\|}_{r\in R}((\mathop{\|}_{\substack{(u_1,u_2)\in U\times U\\ u_1\neq u_2}} Prop_2(r,u_1,u_2)) \,\|\, (\mathop{\|}_{u\in U} Prop_1(r,u))) \\
&\quad = Mtx(U,R) \,.
\end{aligned}$$

Similarly one can show that $g^*(Srs(U_i,R_i)) = Srs(U,R)$, which by Proposition 9 implies that

$$\begin{aligned} g^*(Srs(U_i,R_i) \setminus La(U_i,R_i)) &= g^*(Srs(U_i,R_i)) \setminus \{g^*(\alpha) \mid \alpha \in La(U_i,R_i)\} \\ &= Srs(U,R) \setminus La(U,R)\,. \end{aligned}$$

Hence, we have showed that any specification and system instance with $i \in \{1,2\}$ users and a resource can be obtained from respectively $Mtx(U_i,R_i)$ and $Srs(U_i,R_i) \setminus La(U_i,R_i)$ using bijective renaming. By Theorem 12, it implies that

$$Mtx(U,R) \succeq_{\mathrm{tr}} Srs(U,R) \setminus La(U,R) \text{ for all } (U,R) \in Par'\,. \tag{1}$$

Now, we have generalised our verification results to any system with at most two users and a resource. To generalise the result further, to a system of any size, we show that every specification and system instance can be constructed by composing instances with at most two users and a resource in parallel.

Let $(U,R) \in Par \setminus Par'$. By commutativity, associativity and idempotence of parallel composition (Proposition 8), we may split the parameters R and U to their subsets respectively of size one and two, and take the parallel composition over the subsets instead.

$$\begin{aligned} Mtx(U,R) &= \mathop{\|}_{r\in R} ((\mathop{\|}_{\substack{(u_1,u_2)\in U\times U\\ u_1\neq u_2}} Prop_2(r,u_1,u_2)) \,\|\, (\mathop{\|}_{u\in U} Prop_1(r,u))) \\ &=_{\mathrm{tr}} \mathop{\|}_{\substack{R'\subseteq R\\ |R'|=1}} \mathop{\|}_{r\in R'} \mathop{\|}_{\substack{U'\subseteq U\\ |U'|\leq 2}} ((\mathop{\|}_{\substack{(u_1,u_2)\in U'\times U'\\ u_1\neq u_2}} Prop_2(r,u_1,u_2)) \,\|\, (\mathop{\|}_{u\in U'} Prop_1(r,u))) \\ &=_{\mathrm{tr}} \mathop{\|}_{\substack{R'\subseteq R\\ |R'|=1}} \mathop{\|}_{\substack{U'\subseteq U\\ |U'|\leq 2}} \mathop{\|}_{r\in R'} ((\mathop{\|}_{\substack{(u_1,u_2)\in U'\times U'\\ u_1\neq u_2}} Prop_2(r,u_1,u_2)) \,\|\, (\mathop{\|}_{u\in U'} Prop_1(r,u))) \\ &= \mathop{\|}_{\substack{R'\subseteq R\\ |R'|=1}} \mathop{\|}_{\substack{U'\subseteq U\\ |U'|\leq 2}} Mtx(U',R') =_{\mathrm{tr}} \mathop{\|}_{\substack{(R',U')\in Par'\\ R'\subseteq R, U'\subseteq U}} Mtx(U',R')\,. \end{aligned}$$

Hence, every instance of the specification LTS can be represented as a parallel composition of smaller instances of the specification.

Similarly, when $(U,R) \in Par$, then

$$Srs(U,R) =_{\mathrm{tr}} \mathop{\|}_{\substack{(U',R')\in Par\\ R'\subseteq R, U'\subseteq U}} Srs(U',R')\,,$$

but the corresponding result cannot be generally derived for LTSs involving hiding, because distributing hiding over parallel composition does not preserve the traces equivalence. However, we can still show that

$$\|_{\substack{(R',U')\in Par\\ R'\subseteq R, U'\subseteq U}} (Srs(U',R') \setminus La(U',R')) \succeq_{\mathrm{tr}} Srs(U,R) \setminus La(U,R) \,,$$

which is sufficient for system LTSs, because they can be replaced with more general ones without the risk of false positive verification results.

To formally justify the claim, let $(U',R') \in Par'$ such that $U' \subseteq U$ and $R' \subseteq R$. Clearly, $La(U',R')$ is a subset of $La(U,R)$ and the alphabet of $Srs(U',R')$ is disjoint from $La(U,R) \setminus La(U',R')$. By Proposition 10, it implies that

$$\begin{aligned} &Srs(U',R') \setminus La(U,R) \\ &\quad = (Srs(U',R') \setminus (La(U,R) \setminus La(U',R'))) \setminus (La(U,R) \cap La(U',R')) \\ &\quad\quad = Srs(U',R') \setminus (La(U,R) \cap La(U',R')) = Srs(U',R') \setminus La(U',R') \,. \end{aligned}$$

By above and Proposition 8, it means that

$$\begin{aligned} \|_{\substack{(U',R')\in Par'\\ U'\subseteq U, R'\subseteq R}} (Srs(U',R') \setminus La(U',R')) &= \|_{\substack{(U',R')\in Par'\\ U'\subseteq U, R'\subseteq R}} (Srs(U',R') \setminus La(U,R)) \\ &\succeq_{\mathrm{tr}} (\|_{\substack{(U',R')\in Par'\\ U'\subseteq U, R'\subseteq R}} Srs(U',R')) \setminus La(U,R) =_{\mathrm{tr}} Srs(U,R) \setminus La(U,R) \,, \end{aligned}$$

which proves that every instance of the system is a traces refinement of a parallel composition of smaller instances. By Theorem 12 and Property (1), it implies that

$$Mtx(U,R) \succeq_{\mathrm{tr}} Srs(U,R) \setminus La(U,R) \text{ for all } (U,R) \in Par,$$

i.e. the system works correctly irrespective of the number of users and resources.

5 Formalism

To generalise and automate the verification method presented in the previous chapter, the precise structure of parameterised LTSs has to be specified, and a finite representation for the infinite sets of parameters values is needed. For expressing parameterised systems and specifications, we introduce an LTS schema. It is basically an LTS equipped with variables thought as parameters. An instance of an LTS schema is obtained by fixing the values of variables and the result is a standard LTS.

Instances of an LTS schema are generated using a valuation, which is a function mapping variables to values. However, only valuations that meet certain criteria can be used for the purpose. Valuations that fulfil these requirements are called compatible (with the LTS schema).

Sets of valuations that are potentially infinite are represented finitely with the aid of a valuation formula, which is basically an expression of first order logic encoding the restrictions of variable values. In other words, a valuation formula represents all the valuations that map the variables in the formula to values accordant with these restrictions. Therefore, the set of allowed parameter values is given as a valuation formula that covers precisely the variables in the specification and system LTS schema.

Relationships between different constructs of our formalism, LTSs, LTS schemata, valuations and valuation formulae, are summarised in Figure 14. The formalism is first presented by Siirtola & Kortelainen (2009b,a), and polished further here.

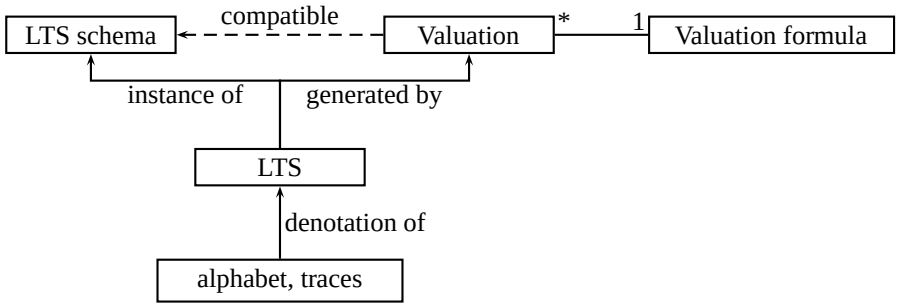

Fig 14. Relationships between different constructs of our formalism.

5.1 LTS Schemata

Based on the example of the previous chapter, we need three kinds of variables to parameterise LTSs. *Atom variables* represent atoms and they are typically used to denote identities of system components, like u and r in $User(u,r)$. *Type variables* denote sets of atoms and they typically represent the sets of the identities of system components of the same kind, like U and R in $La(U,R)$. Finally, *relations variables* denote sets of the tuples of atoms and they are used to describe the topology of a system, and relationships and constraints between its components, like $u_1 \neq u_2$ in the definition of $Srs(U,R)$ and $Mtx(U,R)$.

The sets of all the type, relation and atom variables are respectively denoted by $\mathbb{T}$, $\mathbb{G}$ and $\mathbb{X}$. We assume that the sets are countably infinite and pairwise disjoint. In theoretical contexts, type variables are denoted by upper case letters T, U and their variants, atom variables by lower case letters x, y and their variants, and relation variables by Greek letters Π, Ξ and their variants. For modelling purposes, also other, more informative names are used.

To formally represent LTS schemata, we need to introduce parameterised actions and transitions, called respectively action and transition schemata, first. The purpose of action and transitions schemata is to function as templates from which respectively actions and transitions are generated by substituting atoms for the atom variables. Formally, an *action schema* is a non-empty tuple of atoms and atom variables. For simplicity and without the loss of generality, we may assume that the action schemata are of the form $(x_1,x_2,\ldots,x_k,a_1,a_2,\ldots,a_l)$, where $k+l \in \mathbb{Z}_+$, $x_1,x_2,\ldots,x_k$ are atom variables and $a_1,a_2,\ldots,a_l$ are atoms. A triplet that consists of a state, an action schema and a state, in this order, is a *transition schema*.

Definition 14 (LTS schema)**.**

1. If S is a non-empty set of *states*, Γ a set of action schemata, $\Delta \subseteq S \times (\Gamma \cup \{\tau\}) \times S$ is a set of transition schemata, and $\hat{s} \in S$ is *the initial state*, then $(S,\Gamma,\Delta,\hat{s})$ is an *(elementary) LTS schema*.
2. If $\mathcal{P}_1$ and $\mathcal{P}_2$ are LTS schemata, then $(\mathcal{P}_1 \parallel \mathcal{P}_2)$ is a *(parallel) LTS schema*.
3. If $\mathcal{P}$ is an LTS schema, $T_1,T_2,\ldots,T_k$ are type variables, Π is a relation variable and $x_1,x_2,\ldots,x_k$ are distinct atom variables, where $k \in \mathbb{Z}_+$, then

$$\Big(\parallel_{(x_1,x_2,\ldots,x_k)\in\Pi:T_1\times T_2\times\ldots\times T_k} \mathcal{P}\Big)$$

is a *((Π-)replicated parallel) LTS schema*.

4. If $\mathcal{P}$ is an LTS schema, Γ a set of action schemata, $T_1, T_2, \ldots, T_k$ are type variables and $x_1, x_2, \ldots, x_k$ distinct atom variables, where $k \in \mathbb{N}$, then

$$(\mathcal{P} \setminus \bigcup_{(x_1,x_2,\ldots,x_k)\in T_1\times T_2\times\ldots\times T_k} \Gamma)$$

is a *(hiding) LTS schema*.

Only the expressions obtained by finite application of the steps above are LTS schemata.

In theoretical contexts, LTS schemata are denoted by Calligraphic letters $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$ and their variants, but for modelling purposes also other, more informative names are used. A structure $\bigcup_{(x_1,x_2,\ldots,x_k)\in T_1\times T_2\times\ldots\times T_k}\Gamma$ that is a part of a hiding LTS schema is called a *set schema*. In theoretical contexts, set schemata are denoted by Calligraphic letter $\mathcal{S}$ and its variants, and if k is zero, the set schema can be abbreviated to Γ.

Just like the action and transition schemata are not actions or transitions, neither the LTS schemata are LTSs, but their purpose is to function as templates from which LTSs are generated by substituting atoms, sets of atoms and sets of the tuples of atoms for respectively the atom, type and relation variables. Therefore, neither the (bold) symbols "||", "\", "⋃" are the parallel composition and hiding operators and the set union, but just syntactic connectives that will be interpreted as the very operators above when the values of variables are fixed. Moreover, also the subscripts of "||" and "\" in the replicated parallel and hiding LTS schemata are only syntactic constructs, but they will be interpreted as similar logical conditions when other constructs are assigned their meaning. Additionally, note that there is no structure corresponding to the renaming of LTSs. That is because using a replicated parallel LTS schema one can generate similar components that are identical modulo bijective renaming, and including a structure that allows generic, non-bijective, renaming easily makes the algorithmic analysis of systems impossible. This fact will be justified in more detail in Chapter 8.

If $\mathcal{Q} = (S, \Gamma, \Delta, \hat{s})$ is an elementary LTS schema, then the set S is called the *state space (of $\mathcal{Q}$)*, and the set Γ is the *alphabet schema (of $\mathcal{Q}$)*. An LTS schema $\mathcal{P}$ is *finite*, if every set of states and every set of action schemata occurring in $\mathcal{P}$ is finite. An LTS schema $\mathcal{P}'$ is said to be an *LTS subschema* (of $\mathcal{P}$), if $\mathcal{P}'$ is a subexpression of $\mathcal{P}$. An occurrence of an atom variable x is *bound* in $\mathcal{P}$, if it is within a set schema $\bigcup_{(x_1,x_2,\ldots,x_k)\in T_1\times T_2\times\ldots\times T_k}\Gamma$ or within an LTS subschema $\|_{(x_1,x_2,\ldots,x_k)\in\Pi:T_1\times T_2\times\ldots\times T_k}\mathcal{P}'$ such that $x = x_i$ for some $i \in \{1,2,\ldots,k\}$. An occurrence of x in $\mathcal{P}$ is *free* if it is not

bound. An atom variable x is *free (in $\mathcal{P}$)* if there is a free occurrence of x in $\mathcal{P}$. An LTS schema without free atom variables is said to be *closed*.

The parameters of an LTS schema are the type, relation and free atom variables occurring in it. The set of all the parameters of $\mathcal{P}$ is called the *signature* of $\mathcal{P}$ and it is denoted by $\mathrm{sig}(\mathcal{P})$. The set of all the atoms occurring in $\mathcal{P}$ is referred to by $\mathrm{at}(\mathcal{P})$. If $\|_{(x_1,x_2,\ldots,x_k)\in\Pi:T_1\times T_2\times\ldots\times T_k}\mathcal{P}'$ is an LTS subschema of $\mathcal{P}$, then $T_1 \times T_2 \times \ldots \times T_k$ is a *type of* Π *(in $\mathcal{P}$)*. The set of all the types of Π in $\mathcal{P}$ is denoted by $\mathrm{typ}_{\mathcal{P}}(\Pi)$. Note that $\mathrm{typ}_{\mathcal{P}}(\Pi)$ is non-empty for all relation variables Π in $\mathrm{sig}(\mathcal{P})$ and although it is possible to use several types with a single relation variable, in practice, every relation variable $\Pi \in \mathrm{sig}(\mathcal{P})$ is associated with a precisely one type.

5.2 Modelling using LTS Schemata

As an example, consider a generalisation of the shared resource system, where resources are arranged in the form of a forest. Now, a user obtains a read or write access to the whole subtree of resources after successfully requesting respectively the read or write lock on the root of the subtree and respectively a weaker read or write intention lock on all its ancestors. The purpose of the read (write) intention lock is to indicate the existence of the read (respectively write) lock deeper in the tree. Moreover, several users can hold a lock for a resource simultaneously only if all of them have either intention locks or read related locks. Note that the shared tree system corresponds to the shared resource system when no resource is connected to another one, so the shared tree system is a true generalisation of the shared resource one. Like earlier, our goal is to formally model the system and the specification and to prove that in our construction it is not possible for a user to access a resource if someone else is writing to it.

The replicated components of the system are users and resources. Therefore, we take (distinct) type variables U and R to refer to the sets of, respectively, user and resource identities.

First, we consider the system from the viewpoint of a user u, and reading and writing to a resource r_2 based on the lock on its ancestor r_1, which results in an elementary LTS schema $User_1$ in Figure 15. Here, an ancestor of a resource can also be the resource itself and a proper ancestor is an ancestor other than the resource itself.

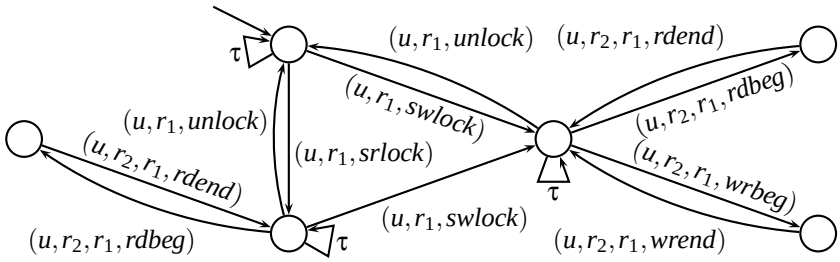

Fig 15. LTS schema $User_1$ representing the behaviour of a user u from the viewpoint of reading and writing to a resource r_2 based on the lock on its (not necessary proper) ancestor resource r_1.

In $User_1$, obtaining the read (write) lock for the subtree of r_1 is modelled by an action schema $(u, r_1, srlock)$ (respectively $(u, r_1, swlock)$) and releasing the lock by an action schema $(u, r_1, unlock)$. The resulting actions will be shared by all the instances of $User_1$ that differ from each other only by the value given to r_2, because all such instances have to be aware of the mode of the lock on r_1.

On the other hand, each instance of $User_1$ should be able to execute actions related to reading and writing independently. To prevent the instances from synchronising on these actions, we include all the atom variables u, r_1, r_2 in the action schemata $(u, r_2, r_1, rdbeg)$, $(u, r_2, r_1, rdend)$, $(u, r_2, r_1, wrbeg)$ and $(u, r_2, r_1, wrend)$ denoting respectively the beginning and the end of reading and the beginning and the end of writing. The action schemata can be thought to mean that the user u_1 reads or writes to the resource r_2 based on respectively the read or write lock on r_1.

Like earlier, the invisible actions are used to represent user's other activities which may take place at any time.

Because the lock requests of a user u on a resource r_2 and its ancestors r_1 are mutually dependent, we need to model the system from the viewpoint of the user u_1 and locking a resource r_2 and its proper ancestor r_1, too. This behaviour is captured in an elementary LTS schema $User_2$ in Figure 16, which introduces three new action schemata: $(u, r_1, irlock)$ (respectively $(u, r_1, iwlock)$) denotes the request of u for the read (respectively write) intention lock on r_1, and an action schema $(u, r_1, nolock)$ informs the user u that r_1 is not locked. Informally, $User_2$ says that the read or read intention lock (respectively the write or write intention lock) can be requested to r_2 only after the read (respectively write) intention lock is granted on r_1, and r_1 can be unlocked only if r_2 is not locked or after r_2 has been unlocked.

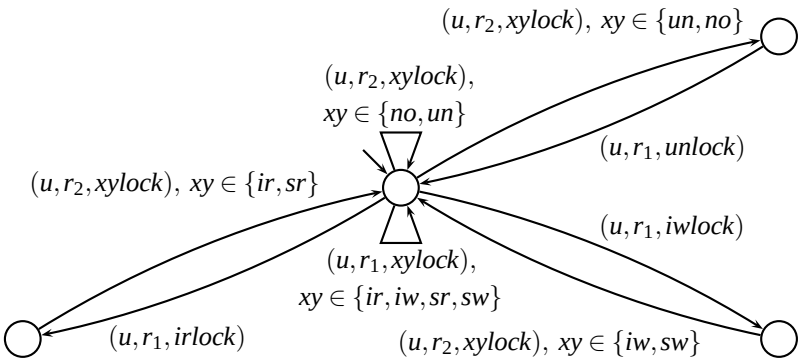

Fig 16. LTS schema $User_2$ representing the behaviour of a user u from the viewpoint of locking a resource r_2 and its proper ancestor r_1.

Finally, as also the lock requests of distinct users on the same resource are mutually dependent, the system has to be modelled from this viewpoint as well. For that purpose, we introduce an elementary LTS schema *Lock* in Figure 17, which stores the lock of a user u_1 on a resource r and restricts the requests of a user u_2 to those that do not violate the locking policy.

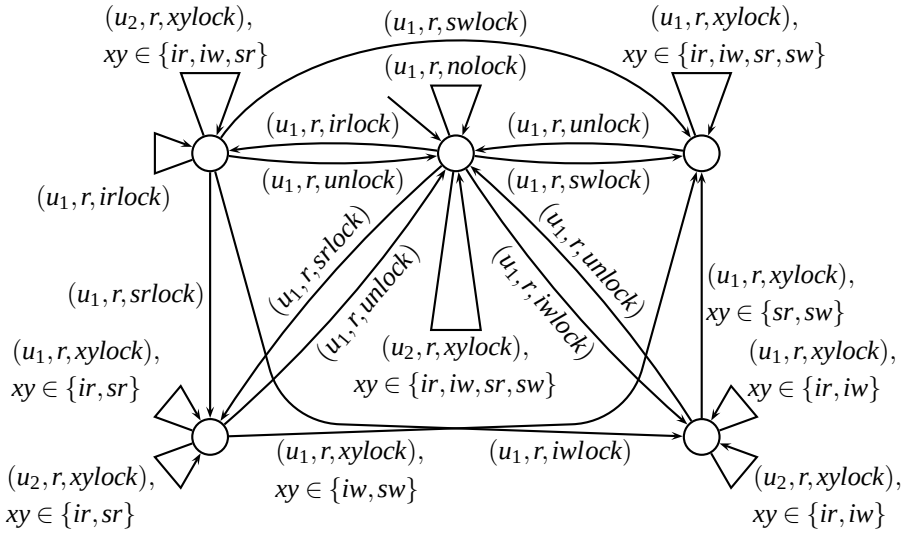

Fig 17. LTS schema *Lock* representing the behaviour of two different users u_1, u_2 from the viewpoint of locking a resource r.

In other words, *Lock* allows any request from u_1 but not all the requests from u_2 if u_1 has a lock on r. If u_1 has an intention lock on r, then u_2 can successfully request intention locks. Moreover, if u_1 has a read-related lock on r, then u_2 can successfully request read-related locks. However, if u_1 has the write lock on r, then no request of u_2 is allowed. Note also that a user cannot have both the read lock and the write intention lock on a resource, but if the user has one of the locks and successfully requests the other, then the lock is converted to the write one.

Clearly, the elementary LTS schemata $User_1$, $User_2$ and *Lock* capture all the behaviours of the system from their point of view and every action the system is supposed to execute is covered by some of them. By informal application of Proposition 13, the behaviour of the system can be over-approximated by composing all the user and lock models in parallel, which results in an LTS schema

$$\Big(\mathop{\|}_{u\in *_U:U}\big(\big(\mathop{\|}_{(r_1,r_2)\in \leq_R:R\times R} User_1\big) \,\|\, \big(\mathop{\|}_{(r_1,r_2)\in <_R:R\times R} User_2\big)\big)\Big) \,\|\, \Big(\mathop{\|}_{r\in *_R:R}\mathop{\|}_{(u_1,u_2)\in \neq_U:U\times U} Lock\Big)\,,$$

denoted by *Sts*, where $*_U$ and $*_R$ are relation variables representing the same sets as respectively U and R, $\neq_U$ denotes the set of all pairs of distinct user identities, $<_R$ represents a forest of resources such that (r_1,r_2) are related by $<_R$ if and only if r_1 is a proper ancestor of r_2, and $\leq_R$ denotes the reflexive closure of $<_R$. As *Sts* over-approximates the behaviour of the system it can be used to prove the system correct but not incorrect, at least without further examination.

To formalise the specification of the shared tree system, note that every illegal behaviour can be traced back to two users u_1,u_2 that write to a resource r_3 simultaneously based on the locks they have on some ancestors r_1,r_2, respectively. Therefore, we first capture the specification in an LTS schema $Prop_2$ in Figure 18 from the viewpoint of u_1,u_2 accessing r_3 based on the locks on respectively r_1,r_2. It says that no matter what locks u_1 and u_2 have on respectively r_1 and r_2, the users can read but not write to r_3 simultaneously. However, to correctly present the specification also in the presence of one user only, we introduce an LTS schema $Prop_1$ in Figure 19. It captures the correctness property from the viewpoint of a single user u that accesses a resource r_3 based on its lock on an ancestor resource r_1.

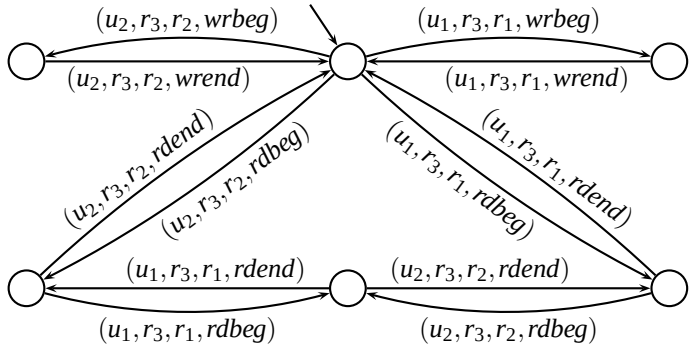

Fig 18. LTS schema $Prop_2$ representing the mutual exclusion property from the viewpoint of two different users u_1, u_2 accessing a resource r_3 based on the locks on respectively ancestors r_1 and r_2.

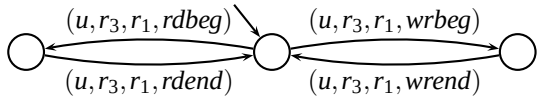

Fig 19. LTS schema $Prop_1$ representing the mutual exclusion property from the viewpoint of a user u accessing a resource r_3 based on the lock on an ancestor r_1.

Because every illegal behaviour is forbidden by some instance of $Prop_2$ and every action the specification is supposed to track is covered by some instance of $Prop_1$, the behaviour of the specification can be under-approximated by composing all the partial specifications together. It results in an LTS schema

$$Mtx := \|_{(r_1,r_2,r_3)\in \leqq_R : R\times R\times R} \left(\left(\|_{(u_1,u_2)\in \neq_U : U\times U} Prop_2 \right) \| \left(\|_{u_1 \in *_U : U} Prop_1 \right) \right),$$

where $\leqq_R$ denotes the set of all triplets (r_1, r_2, r_3) such that r_1 and r_2 are ancestors of r_3.

Finally, because the locking actions are irrelevant to the specification, we hide the actions represented by a set schema

$$La := \bigcup_{(u,r)\in U\times R} \{(u, r, xylock) \mid xy \in \{ir, sr, iw, sw, no, un\}\}$$

in the system model.

In general, when expressing a system as an LTS schema, one typically identifies different kinds of system components first. After that, type variables are chosen to represent the sets of the identities of similar replicated components (the number of

which is parameter dependent). Other components, the number of which is bounded, are typically referred to by atoms chosen for the purpose.

Next, one captures the behaviour of the system in finitely many elementary LTS schemata. Each of them represents the system from the viewpoint of certain behaviours related to finitely many components, and every action the system is supposed to execute must be covered by some of the elementary LTS schemata. If a view involves replicated components, selected atom variables are used to refer to their identities.

Sometimes, in order to avoid introducing excess behaviours, it would be necessary to model the system from the viewpoint of a parameterised number of components, but this is not possible in our formalism. However, there is a natural reason for that; allowing elementary LTS schemata to refer to all the replicated components of a certain kind easily makes automated analysis impossible, as will be pointed out in Chapter 8.

As all the relevant viewpoints are covered, it is time to put the system together. Each elementary LTS schema specified earlier is enclosed within replicated parallel LTS schemata such that every atom variable becomes bound. Obviously, one wants to bind an atom variable to a type variable that represents the components of the same type as the atom variable. Basically, it is sufficient to enclose each elementary LTS schema within a single replicated parallel LTS schemata, but in practice a single replicated parallel LTS schema is used to bind only atom variables that refer to the components of the same type. That is because the values of such atom variables are often interrelated whereas those of atom variables used to denote components of different types are typically not, and relationships between system components are represented by a relation variable that is an integral part of a replicated parallel LTS schema. Splitting a replicated parallel LTS schema into smaller pieces makes specifying the values for relation variables easier and reusing the relation variables possible.

Note that when enclosing an elementary LTS schema within a replicated parallel LTS schema, it is easy to introduce unintentional deadlocks. Consider an elementary LTS schema like $User_1$ that represents the system from a certain point of view and which is enclosed within replicated parallel LTS schemata as described above. Action schemata, like $(u, r_1, swlock)$, that do not involve all the atom variables (occurring in the elementary LTS schema) give rise to actions that are practically always shared by and hence synchronised between several LTSs arising from the elementary LTS schema. However, as the attention is on behaviours local to certain components, it is easy to forget this fact while constructing the elementary LTS schema, which may lead to unintentional deadlocks. Hence, if each LTS arising from an elementary LTS schema is

supposed to execute some actions on its own, like the read and write related actions in the above example, it is generally necessary that the corresponding action schemata involve all the atom variables occurring in the elementary LTS schema.

After binding all the atom variables, the resulting replicated parallel LTS schemata are put together using parallel LTS schemata. If the modelling is carefully done in the sense that each elementary LTS schema captures all the behaviours of the system from its point of view and every action the system is supposed to execute is covered by some of the elementary LTS schemata, by informal application of Proposition 13, the resulting parallel LTS schema over-approximates the behaviour of the system. In other words, it can be used to prove the system correct but generally not incorrect.

Specifications are modelled in a similar way as systems: a specification is first described from the viewpoint of a fixed number of components, and then the resulting elementary LTS schemata are enclosed within (replicated) parallel LTS schemata. The only difference is that now, in order to avoid false positive verification results, one has to check that every illegal behaviour is blocked by some of the elementary LTS schemata. Finally, after the specification is formalised, one encapsulated the system model in a hiding LTS schema to get rid of actions irrelevant to the specification.

5.3 Valuations

Parameter values are formally represented as a valuation. A valuation is a partial function that maps atom, type and relation variables to respectively atoms, sets of atoms and sets of visible actions (sets of the non-empty tuples of atoms), where the atoms occurring in the values of atom and relation variables are restricted to those occurring in the images of type variables.

Definition 15 (Valuation)**.** A *valuation* is a partial function

$$\phi : \mathbb{X} \cup \mathbb{T} \cup \mathbb{G} \mapsto \mathbb{A} \cup \mathbb{P}(\mathbb{A}) \cup \mathbb{P}(\mathbb{V})$$

such that

– $\phi|_{\mathbb{T}}$ maps type variables to finite, non-empty, pairwise disjoint sets of atoms,
– $\phi|_{\mathbb{G}}$ maps relation variables to $\bigcup_{k \in \mathbb{Z}_+} \bigcup_{A_1, A_2, \ldots, A_k \in \mathrm{im}(\phi|_{\mathbb{T}})} \mathbb{P}(A_1 \times A_2 \times \ldots \times A_k)$, and
– $\phi|_{\mathbb{X}}$ maps atom variables to $\bigcup_{A \in \mathrm{im}(\phi|_{\mathbb{T}})} A$.

Valuations are denoted by lower case Greek letters ϕ, ψ, θ and their variants. Sets of valuations are denoted by capital Greek letters Φ, Ψ, Θ and their variants.

A valuation ϕ is said to be *(fully) compatible* (*with* an LTS schema $\mathcal{P}$), if ϕ defines values for all type variables, relation variables and free atom variables in $\mathcal{P}$ without using any atoms that occur in $\mathcal{P}$, and relation variables are mapped to subsets of the values of their types. Formally, it means that

- the signature of $\mathcal{P}$ is a subset of the domain of ϕ,
- the set of all the atoms occurring in $\mathcal{P}$ is disjoint from $\bigcup_{A \in \mathrm{im}(\phi|_{\mathbb{T}})} A$ and
- whenever Π is a relation variable in the domain and $T_1 \times T_2 \times \cdots \times T_k$ is a type of Π in $\mathcal{P}$, then $\phi(\Pi)$ is a subset of $\bigtimes_{i=1}^{k} \phi(T_i)$. (Recall that in practice a relation variable is only associated with a single type.)

A valuation ϕ that satisfies the last two requirements is called *partially compatible with* $\mathcal{P}$. Therefore, a partially compatible valuation can always be extended to a fully compatible one.

Let ϕ be a valuation compatible with an LTS schema $\mathcal{P}$. An *instance* of $\mathcal{P}$ *(generated by* ϕ*)* is denoted by $[\![\mathcal{P}]\!]_\phi$ and basically obtained from $\mathcal{P}$ by first substituting type variables, relation variables and free atom variables according to ϕ, converting symbolic operators to standard ones, and then applying them as usual.

To define $[\![\mathcal{P}]\!]_\phi$ formally, let Π be a relation variable in $\mathrm{sig}(\mathcal{P})$ and $x_1, x_2, \ldots, x_k$ any atom variables. We write $\phi[(x_1, x_2, \ldots, x_k) \mapsto \Pi]$ for the set of all functions ϕ' such that

1. the domain of ϕ' is $\mathrm{dom}(\phi) \cup \{x_1, x_2, \ldots, x_k\}$,
2. $(\phi'(x_1), \phi'(x_2), \ldots, \phi'(x_k)) \in \phi(\Pi)$ and
3. ϕ' maps the elements in $\mathrm{dom}(\phi) \setminus \{x_1, x_2, \ldots, x_k\}$ like ϕ does.

Similarly, $\phi[x_1 \mapsto T_1, x_2 \mapsto T_2, \ldots, x_k \mapsto T_k]$, where $T_1, T_2, \ldots, T_k$ are type variables in $\mathrm{dom}(\phi)$, denotes the set of all the functions satisfying Items 1 and 3 above and mapping x_i to $\phi(T_i)$ whenever $i \in \{1, 2, \ldots, k\}$. In other words, $\phi[(x_1, x_2, \ldots, x_k) \mapsto \Pi]$ denotes the set of all the valuations obtained by (re)defining ϕ for $x_1, x_2, \ldots, x_k$ such that the value of $(x_1, x_2, \ldots, x_k)$ is in $\phi(\Pi)$, and $\phi[x_1 \mapsto T_1, x_2 \mapsto T_2, \ldots, x_k \mapsto T_k]$ is the set of all the valuations obtained by (re)defining ϕ for $x_1, x_2, \ldots, x_k$ such that the value of x_i is in $\phi(T_i)$ for all $i \in \{1, 2, \ldots, k\}$.

Lemma 16. *Let* ϕ *be a valuation,* Π *a relation variable and* $T_1, T_2, \ldots, T_k$ *type variables in* $\mathrm{dom}(\phi)$*, and* $x_1, x_2, \ldots, x_k$ *any atom variables. Then*

1. $\phi[(x_1, x_2, \ldots, x_k) \mapsto \Pi]$ *and* $\phi[x_1 \mapsto T_1, x_2 \mapsto T_2, \ldots, x_k \mapsto T_k]$ *are sets of valuations, and*

2. *if ϕ is compatible with a replicated parallel LTS schema $\|_{(x_1,x_2,\ldots,x_k)\in\Pi:T_1\times T_2\times\ldots\times T_k}\mathcal{P}$, then every valuation in $\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]$ is compatible with $\mathcal{P}$.*

The claims follow straightforwardly from the definitions.

Now, the instance of $\mathcal{P}$ generated by ϕ can be defined inductively as follows.

1. $[\![(S,\Gamma,\Delta,\hat{s})]\!]_\phi = (S,[\![\Gamma]\!]_\phi,[\![\Delta]\!]_\phi,\hat{s})$,
   where $[\![\Gamma]\!]_\phi$ and $[\![\Delta]\!]_\phi$ are obtained from respectively Γ and Δ by substituting $\phi(x)$ for every occurrence of each atom variable x.
2. $[\![(\mathcal{P}_1 \parallel \mathcal{P}_2)]\!]_\phi = [\![\mathcal{P}_1]\!]_\phi \parallel [\![\mathcal{P}_2]\!]_\phi$.
3. $[\![(\mathop{\|}_{(x_1,x_2,\ldots,x_k)\in\Pi:T_1\times T_2\times\ldots\times T_k}\mathcal{P}')]\!]_\phi = \mathop{\|}_{\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]}[\![\mathcal{P}']\!]_{\phi'}$.
4. $[\![(\mathcal{P}'\setminus\bigcup_{(x_1,x_2,\ldots,x_k)\in T_1\times T_2\times\ldots\times T_k}\Gamma)]\!]_\phi = [\![\mathcal{P}']\!]_\phi \setminus \bigcup_{\phi'\in\phi[x_1\mapsto T_1,x_2\mapsto T_2,\ldots,x_k\mapsto T_k]}[\![\Gamma]\!]_{\phi'}$,
   where $[\![\Gamma]\!]_{\phi'}$ is defined similarly as $[\![\Gamma]\!]_\phi$ in the first item.

We may omit parentheses in an LTS schema if it does not affect the order of evaluating an instance.

Lemma 17. *Let $\mathcal{P}$ be an LTS schema and ϕ a valuation compatible with $\mathcal{P}$.*

1. *The instance of $\mathcal{P}$ generated by ϕ is an LTS.*
2. *If $\mathcal{P}$ is finite, then $[\![\mathcal{P}]\!]_\phi$ is finite as well.*

Proof. First, note that if ϕ is a valuation and Γ a set of action schemata such that every atom variable occurring in Γ is in the domain of ϕ, then $[\![\Gamma]\!]_\phi$ is clearly a set of actions, and if Γ is finite, then $[\![\Gamma]\!]_\phi$ is finite, too.

Now, let $\mathcal{P}$ be an LTS schema and ϕ a valuation compatible with $\mathcal{P}$. We argue by induction on the structure of the LTS schema using the lemma as an induction hypothesis.

As the base step, $\mathcal{P}$ is an elementary LTS schema $(S,\Gamma,\Delta,\hat{s})$. Clearly, every atom variable occurring in Γ or in Δ is in the domain of ϕ. Hence, by above, $[\![\Gamma]\!]_\phi$ is a set of visible actions, and as Δ is a subset of $S\times(\Gamma\cup\{\tau\})\times S$, then $[\![\Delta]\!]_\phi$ is a subset of $S\times([\![\Gamma]\!]_\phi\cup\{\tau\})\times S$, which implies that $[\![\mathcal{P}]\!]_\phi$ is an LTS. Furthermore, if $\mathcal{P}$ is finite, meaning that S and Γ are finite, then $[\![\Gamma]\!]_\phi$ is finite as well, which in turn implies that $[\![\mathcal{P}]\!]_\phi$ is a finite LTS.

As the induction step, $\mathcal{P}$ is either a parallel, hiding or replicated parallel LTS schema. If $\mathcal{P}$ is a parallel LTS schema $\mathcal{P}_1 \parallel \mathcal{P}_2$, by the induction hypothesis, $[\![\mathcal{P}_1]\!]_\phi$ and $[\![\mathcal{P}_2]\!]_\phi$

are LTSs. Then, by Lemma 2, also $[\![\mathcal{P}_1]\!]_\phi \parallel [\![\mathcal{P}_2]\!]_\phi = [\![\mathcal{P}]\!]_\phi$ is an LTS. Furthermore, if $\mathcal{P}$ is finite, then both $\mathcal{P}_1$ and $\mathcal{P}_2$ are finite, which by the induction hypothesis implies that $[\![\mathcal{P}_1]\!]_\phi$ and $[\![\mathcal{P}_2]\!]_\phi$ are finite LTSs. By Lemma 2, it means $[\![\mathcal{P}_1]\!]_\phi \parallel [\![\mathcal{P}_2]\!]_\phi = [\![\mathcal{P}]\!]_\phi$ is a finite LTS, too.

If $\mathcal{P}$ is a replicated parallel LTS schema $\|_{(x_1,x_2,\ldots,x_k)\in\Pi:T_1\times T_2\times\ldots\times T_k}\mathcal{P}'$ and ϕ' a function in $\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]$, then, by Lemma 16, ϕ' is a valuation compatible with $\mathcal{P}'$. By the induction hypothesis, it means that $[\![\mathcal{P}']\!]_{\phi'}$ is an LTS. As $\phi(\Pi)$ is finite, by Lemma 2 and the definition of the generic form of the parallel composition,

$$\|_{\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]} [\![\mathcal{P}']\!]_{\phi'} = [\![\mathcal{P}]\!]_\phi$$

is an LTS, too. Furthermore, if $\mathcal{P}$ is finite, then $\mathcal{P}'$ is finite, too. By the induction hypothesis, it implies that $[\![\mathcal{P}']\!]_{\phi'}$ is finite whenever ϕ' is a valuation in $\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]$. As $\phi(\Pi)$ is necessarily finite, by Lemma 2 and the definition of the generic form of the parallel composition, it means that $[\![\mathcal{P}]\!]_\phi$ is finite as well.

Finally, we assume that $\mathcal{P}$ is a hiding LTS schema $\mathcal{P}'\setminus\bigcup_{(x_1,x_2,\ldots,x_k)\in T_1\times T_2\times\ldots\times T_k}\Gamma$. By the induction hypothesis, then $[\![\mathcal{P}']\!]_\phi$ is an LTS. Moreover, if ϕ' is a valuation in $\phi[x_1\mapsto T_1,x_2\mapsto T_2,\ldots,x_k\mapsto T_k]$, then every atom variable occurring in Γ is clearly in the domain of ϕ'. Hence, by above, $[\![\Gamma]\!]_{\phi'}$ is a set of visible actions. By Lemma 2, it means that

$$[\![\mathcal{P}]\!]_\phi = [\![\mathcal{P}']\!]_\phi \setminus \bigcup_{\phi'\in\phi[x_1\mapsto T_1,x_2\mapsto T_2,\ldots,x_k\mapsto T_k]} [\![\Gamma]\!]_{\phi'}$$

is an LTS. Additionally, if $\mathcal{P}$ is finite, then $\mathcal{P}'$ is finite as well. By the induction hypothesis it means that $[\![\mathcal{P}']\!]_\phi$ is a finite LTS, which by Lemma 2 implies that $[\![\mathcal{P}]\!]_\phi$ is a finite LTS, too. □

When generating instances of a specification and system LTS schema, it is certainly sufficient to consider valuations that determine values for precisely the parameters of the LTS schemata. Moreover, as the system and specification LTS schema are typically closed, it is normally sufficient to consider values for type and relation variables only. Furthermore, there is usually no need to restrict the values of type variables anyhow (apart from possibly avoiding the use of atoms occurring in the LTS schemata), because one typically wants to know whether the system works correctly for any number of replicated components. On the other hand, it makes sense to restrict the values of relation variables as they determine the topology of the system and relationships between the

components and, in most of the cases, one is not interested in all the possible topologies and arbitrary connections between the components.

However, the values of relation variables are generally not uniquely determined by those of type variables. For example, when generating a family of processes arranged in the form of a tree, one probably uses a type variable to represent the identities of processes and a relation variable to encode the relationships between processes, *i.e.* the tree topology. Clearly, there are several ways to arrange a set of processes in the form of a tree, which shows that in general, the values of relation variables are not unambiguously defined with the aid of those of type variables.

For example, the instances of the shared tree system and the mutual exclusion property are generated by valuations ϕ with the domain $\{U, R, *_U, *_R, \neq_U, <_R, \leq_R, \leqq_R\}$ such that ϕ maps U and R to finite non-empty disjoint sets of atoms, $\phi(T) = \phi(*_T)$ for both $T \in \{U, R\}$, $\phi(\neq_U)$ is the maximal irreflexive relation over $\phi(U)$, $\phi(<_R)$ is an irreflexive, asymmetric and transitive relation over $\phi(R)$ such that whenever r_1 and r_2 are different ancestors of r_3, then r_1 is an ancestor of r_2, or r_2 is an ancestor of r_1, $\phi(\leq_R)$ is the reflexive closure of $\phi(<_R)$, and $\phi(\leqq_R)$ is the set of all triplets (r_1, r_2, r_3) such that $(r_1, r_3), (r_2, r_3) \in \phi(\leq_R)$. Note that for each value of U and R there are several possible values for $<_R$, but the values of the other relation variables can be unambiguously defined with the aid of U, R and $<_R$.

We write Φ_{Sts} for the set of all the valuations above. Therefore, the verification task related to the shared tree system can be now formalised as the question whether $[\![Mtx]\!]_\phi \succeq_{\mathrm{tr}} [\![Sts \setminus La]\!]_\phi$ for all valuations $\phi \in \Phi_{Sts}$ compatible with *Mtx* and $Sts \setminus La$.

5.4 Valuation Formulae

Normally, one wants to know whether the system works correctly for any number of replicated components, which implies that there are infinitely many instances to check and hence infinitely many valuations to specify. Therefore, in order to automate the reduction method, we still need a finite representation for the potentially infinite sets of valuations. Because there is typically no need to impose restrictions on the values of type variables, the main purpose of such a formalism is to encode the values of relation variables.

A natural choice is to use logics of some sort. For example, the family of all the maximal irreflexive relations G over A, needed to represent the values of $\neq_U$, can be

expressed as the formula

$$\forall a_1, a_2 \in A : (a_1 = a_2 \to (a_1, a_2) \notin G) \wedge (a_1 \neq a_2 \to (a_1, a_2) \in G) \, .$$

The structures used in the formula are quantification, equivalence and membership testing, and standard boolean connectives. By generalising this notion, we end up with a structure we call a valuation formula. It is basically an expression of first order logic equipped with atom, type and relation variables, where one can use universal quantification to let the value of an atom variable to range over the value of a type variable, test the values of atom variables for equality, and test whether the values of atom variables are related by the value of a relation variable.

Definition 18 (Valuation formula)**.**

1. $\top$ is an (elementary) (always true) valuation formula.
2. If $x_0, x_1, x_2, \ldots, x_n$ are atom variables, where $n \in \mathbb{Z}_+$, and Π a relation variable, then $x_0 = x_1$ and $(x_1, x_2, \ldots, x_n) \in \Pi$ are (elementary) valuation formulae.
3. If c_1 and c_2 are valuation formulae, then $(\neg c_1)$ is a (negated) valuation formula, $(c_1 \vee c_2)$ a (disjunctive) valuation formula, and $(c_1 \wedge c_2)$ a (conjunctive) valuation formula.
4. If c is a valuation formula, T a type variable, and x an atom variable, then $(\forall x \in T : c)$ is a (universal) valuation formula.

Only the expressions obtained by finite application of the steps above are *valuation formulae*.

Valuation formulae are denoted by c and its variants.

A valuation formula c' is said to be a *valuation subformula* (of a valuation formula c), if c' is a subexpression of c. A *conjunct* of a valuation formula c is intuitively a maximal valuation subformula c' of c such that c' is not conjunctive. More formally, if c is not conjunctive then its only conjunct is c itself, and if c is $(c_1 \wedge c_2)$, then the conjuncts of c are the conjuncts of c_1 and c_2. Moreover, a valuation formula c is called

1. *existential-free*, if every universal valuation subformula occurs within an even number of negated valuation subformulae in c;
2. *negation-normal*, if whenever $\neg c'$ is a valuation subformula of c, then c' is an elementary valuation formula; and
3. *membership-negated*, if every elementary valuation subformula $(x_1, x_2, \ldots, x_n) \in \Pi$ occurs within an odd number of negated valuation subformulae of c.

An occurrence of an atom variable x is *free* in c if the occurrence is not within a universal valuation subformula $(\forall x \in T : c')$ of c. An atom variable x is *free* in c if there is a free occurrence of x in c. A valuation formula with no free atom variable is *closed*. The set of all the type, relation and free atom variables in c is called the *signature* of c and denoted by $\mathrm{sig}(c)$.

We say that a valuation ϕ is *compatible* with a valuation formula c, if the signature of c is a subset of the domain of ϕ. If ϕ is compatible with c, then $[\![c]\!]_\phi$ denotes a formula obtained from c by substituting $\phi(T)$ for all occurrences of each type variable T, $\phi(\Pi)$ for all occurrences of each relation variable Π, and $\phi(x)$ for all free occurrences of each atom variable x in $\mathrm{sig}(c)$. The formula $[\![c]\!]_\phi$ is called the *instance (of c) (generated by ϕ)* and it is evaluated in the usual way.

Formally, $[\![c]\!]_\phi$ is either true or false, defined inductively in the structure of the valuation formula as follows.

1. $[\![\top]\!]_\phi$ is true.
2. $[\![x = y]\!]_\phi$ is true if and only if $\phi(x) = \phi(y)$.
3. $[\![(x_1, x_2, \ldots, x_n) \in \Pi]\!]_\phi$ is true if and only if $(\phi(x_1), \phi(x_2), \ldots, \phi(x_n)) \in \phi(\Pi)$.
4. $[\![(\neg c')]\!]_\phi$ is true if and only if $[\![c']\!]_\phi$ is not.
5. $[\![(c_1 \vee c_2)]\!]_\phi$ is true if and only if $[\![c_1]\!]_\phi$ or $[\![c_2]\!]_\phi$ is true.
6. $[\![(c_1 \wedge c_2)]\!]_\phi$ is true if and only if $[\![c_1]\!]_\phi$ and $[\![c_2]\!]_\phi$ are true.
7. $[\![(\forall x \in T : c')]\!]_\phi$ is true if and only if $[\![c']\!]_{\phi'}$ is true for all $\phi' \in \phi[x \mapsto T]$.

The parentheses can be omitted from a valuation formula if it does not affect the value of the instances of the valuation formula.

Additionally, we introduce some standard abbreviations. Whenever n is a positive integer, $x_0, x_1, x_2, \ldots, x_n$ are atom variables, T is a type variable, Π a relation variable, and c, c_1, c_2 are valuation formulae, we write

- $x_0 \neq x_1$ short for $\neg(x_0 \neq x_1)$,
- $(x_1, x_2, \ldots, x_n) \notin \Pi$ short for $\neg((x_1, x_2, \ldots, x_n) \in \Pi)$,
- $(c_1 \to c_2)$ short for $((\neg c_1) \vee c_2)$,
- $(c_1 \leftrightarrow c_2)$ short for $((c_1 \to c_2) \wedge (c_2 \to c_1))$,
- $(\forall x_1, x_2, \ldots, x_n \in T : c)$ short for $(\forall x_1 \in T : \forall x_2 \in T : \cdots : \forall x_n \in T : c)$, and
- $(\exists x_1, x_2, \ldots, x_n \in T : c)$ short for $\neg(\forall x_1 \in T : \forall x_2 \in T : \cdots : \forall x_n \in T : \neg c)$.

Moreover, if $I = \{i_1, i_2, \ldots, i_n\}$ is a finite totally ordered set such that $i_1 < i_2 < \ldots < i_n$, and c_i is a valuation formula for every $i \in I$, then the *conjunction (of $c_{i_1}, c_{i_2}, \ldots, c_{i_n}$)*,

denoted by $\bigwedge_{i\in I} c_i$ or $\bigwedge_{j=1}^{n} c_{i_j}$, is defined as the valuation formula

$$((\ldots((c_1 \wedge c_2) \wedge c_3)\ldots) \wedge c_n)$$

for non-empty I, and as $\top$ in the case I is empty. Note that there is no need to explicitly specify the total order if one is only interested in the instances of a conjunction, because an instance of a conjunction evaluates to the same truth value no matter which order is chosen.

It is said that a valuation ϕ *satisfies* a valuation formula c, if ϕ is compatible with c and $[\![c]\!]_\phi$ is true. Now, the set of valuations represented by c, denoted by $\mathrm{va}(c)$, is naturally defined as the set of all valuations ϕ with the domain $\mathrm{sig}(c)$ such that ϕ satisfies c. In other words, a valuation is in the set represented by a valuation formula, if the valuation assigns values to precisely the variables in $\mathrm{sig}(c)$ such that the formula evaluates to true.

As already stated, the main purpose of a valuation formula is to specify allowed values for relation variables based on the values of type variables. It is done by encoding the restrictions which the values of relation variables have to satisfy using the structures of valuation formulae.

As an example, consider the set Φ_{Sts} of valuations related to the shared tree system. It can be expressed as $\mathrm{va}(c_{Sts})$, where c_{Sts} is a valuation formula

$$\begin{aligned}
&(\forall r \in R : (r,r) \notin <_R) \wedge (\forall r_1, r_2 \in R : ((r_1,r_2) \notin <_R \vee (r_2,r_1) \notin <_R)) \wedge \\
&(\forall r_1, r_2, r_3 \in R : (((r_1,r_2) \in <_R \wedge (r_2,r_3) \in <_R) \rightarrow (r_1,r_3) \in <_R) \wedge \\
&((r_1 \neq r_2 \wedge (r_1,r_3) \in <_R \wedge (r_2,r_3) \in <_R) \rightarrow ((r_1,r_2) \in <_R \vee (r_2,r_1) \in <_R))) \wedge \\
&(\forall r_1, r_2 \in R : (r_1,r_2) \in \leq_R \leftrightarrow (r_1 = r_2 \vee (r_1,r_2) \in <_R)) \wedge \\
&(\forall r_1, r_2, r_3 \in R : (r_1,r_2,r_3) \in \leqq_R \leftrightarrow ((r_1,r_3) \in \leq_R \wedge (r_2,r_3) \in \leq_R)) \wedge \\
&(\forall r \in R : r \in *_R) \wedge (\forall u \in U : u \in *_U) \wedge (\forall u_1, u_2 \in U : (u_1,u_2) \in \neq_U \leftrightarrow u_1 \neq u_2)\ .
\end{aligned}$$

The first three lines say that $<_R$ represents an irreflexive, asymmetric and transitive relation such that if two different atoms a_1, a_2 are smaller than a_3 then either a_1 is smaller than a_2 or a_2 is smaller than a_1 in the sense of $<_R$. The fourth line states that $\leq_R$ represents the reflexive closure of $<_R$, and the fifth line says that the value of $\leqq_R$ is the set of all triplets (a_1, a_2, a_3) such that a_1 and a_2 are smaller than or equal to a_3 (in the sense of $\leq_R$). The last line states that the values of T and $*_T$ must match, for both $T \in \{U, R\}$, and $\neq_U$ denotes the maximal irreflexive relation over the value of U. Obviously, the correctness of the shared tree system can be now stated as the question

whether $[\![Mtx]\!]_\phi \succeq_{\mathrm{tr}} [\![Sts \setminus La]\!]_\phi$ for all valuations $\phi \in \mathrm{va}(c_{Sts})$ compatible with *Mtx* and $Sts \setminus La$.

5.5 The Parameterised Verification Problem

We may now assume that a (parameterised) system and its specification are given as LTS schemata and the allowed parameter values are encoded as a set of valuations expressed with the aid of a valuation formula.

Problem 19 (Parameterised Traces Refinement)**.**
Instance: LTS schemata $\mathcal{Q}$ and $\mathcal{P}$, and a set Φ of valuations.
Question: Is $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$ for all valuations $\phi \in \Phi$ compatible with $\mathcal{Q}$ and $\mathcal{P}$?

In other words, we are interested in instances $\mathcal{Q}$, $\mathcal{P}$, Φ of Parameterised Traces Refinement such that $\Phi = \mathrm{va}(c)$ for some valuation formula c.

Without the loss of generality, we may assume that the signature of c is $\mathrm{sig}(\mathcal{Q}) \cup \mathrm{sig}(\mathcal{P})$, *i.e.* the valuations in $\mathrm{va}(c)$ specify values precisely to the parameters of $\mathcal{Q}$ and $\mathcal{P}$. Moreover, we also assume the LTS schemata to be finite, because they are used to represent real systems which obviously cannot reach infinitely many states. Besides, allowing infinite LTS schemata would make cut-offs useless, because the instances of infinite LTS schemata are typically infinite too, and even though one could obtain the best possible cut-off, there would still be an infinite verification task to solve.

Without the loss of generality, we may also assume that the LTS schemata are closed, because finitely many atom variables can be eliminated by introducing new relation variables. As a result, the original problem instance splits into finitely many new ones, and the answer to the original one is positive if and only if the answer to all the new instances is positive.

Proposition 20. *Let $\mathcal{Q}$ and $\mathcal{P}$ be finite LTS schemata and c a valuation formula with the signature $\mathrm{sig}(\mathcal{Q}) \cup \mathrm{sig}(\mathcal{P})$. Moreover, let $x_1, x_2, \ldots, x_k$ be the free atom variables in the LTS schemata, $\Pi_1, \Pi_2, \ldots, \Pi_k$ distinct relation variables not occurring in the LTS schemata, and $T_1, T_2, \ldots, T_k$ type variables in $\mathrm{sig}(c)$, where $k \in \mathbb{N}$. Then*

$$\mathcal{Q}_{T_1,T_2,\ldots,T_k} := \mathop{\|}_{x_1 \in \Pi_1 : T_1} \mathop{\|}_{x_2 \in \Pi_2 : T_2} \cdots \mathop{\|}_{x_k \in \Pi_k : T_k} \mathcal{Q}$$

and

$$\mathcal{P}_{T_1,T_2,\ldots,T_k} := \mathop{\Vert}_{x_1\in\Pi_1:T_1} \mathop{\Vert}_{x_2\in\Pi_2:T_2} \cdots \mathop{\Vert}_{x_k\in\Pi_k:T_k} \mathcal{P}$$

are closed and finite LTS schemata and

$$c_{T_1,T_2,\ldots,T_k} := \bigwedge_{i=1}^{k} (\forall y_1, y_2 \in T_i : ((y_1 \in \Pi \wedge y_2 \in \Pi_i) \to y_1 = y_2)) \wedge$$
$$(\forall x_1 \in T_1 : \forall x_2 \in T_2 : \cdots : \forall x_k \in T_k : ((\bigwedge_{i=1}^{k} x_i \in \Pi_i) \to c))$$

is a valuation formula with the signature $\mathrm{sig}(\mathcal{Q}_{T_1,T_2,\ldots,T_k}) \cup \mathrm{sig}(\mathcal{P}_{T_1,T_2,\ldots,T_k})$ *such that* $c_{T_1,T_2,\ldots,T_k}$ *is existential-free if and only if* c *is existential-free and the answer to the instance* $\mathcal{Q}, \mathcal{P}, c$ *of Parameterised Traces Refinement is positive if and only if the answer to the instance* $\mathcal{Q}_{U_1,U_2,\ldots,U_k}$, $\mathcal{P}_{U_1,U_2,\ldots,U_k}$, $c_{U_1,U_2,\ldots,U_k}$ *is positive for all type variables* $U_1, U_2, \ldots, U_k \in \mathrm{sig}(c)$.

Proof. Clearly, for all type variables $T_1, T_2, \ldots, T_k \in \mathrm{sig}(c)$, the structures $\mathcal{Q}_{T_1,T_2,\ldots,T_k}$ and $\mathcal{P}_{T_1,T_2,\ldots,T_k}$ are closed and finite LTS schemata and $c_{T_1,T_2,\ldots,T_k}$ is a valuation formula with the signature $\mathrm{sig}(\mathcal{Q}_{T_1,T_2,\ldots,T_k}) \cup \mathrm{sig}(\mathcal{P}_{T_1,T_2,\ldots,T_k})$ such that $c_{T_1,T_2,\ldots,T_k}$ is existential-free if and only if c is.

To prove the rest of the claim, let us first suppose that the answer to $\mathcal{Q}$, $\mathcal{P}$, c is positive. Moreover, let $T_1, T_2, \ldots, T_k$ be type variables in $\mathrm{sig}(c)$ and ψ a compatible valuation in $\mathrm{va}(c_{T_1,T_2,\ldots,T_k})$. As ψ satisfies the first conjuncts of $c_{T_1,T_2,\ldots,T_k}$, it means that ψ maps Π_i to the empty set or a singleton for every $i \in \{1,2,\ldots,k\}$. If $\psi(\Pi_j)$ is the empty set for some $j \in \{1,2,\ldots,k\}$, then it is obvious that $[\![\mathcal{Q}_{T_1,T_2,\ldots,T_k}]\!]_\psi \succeq_{\mathrm{tr}} [\![\mathcal{P}_{T_1,T_2,\ldots,T_k}]\!]_\psi$ holds. On the other hand, if $\phi(\Pi_i)$ is a singleton $\{a_i\}$ for all $i \in \{1,2,\ldots,k\}$, then by the compatibility, $a_i \in \phi(T_i)$ for all $i \in \{1,2,\ldots,k\}$. As ψ satisfies the last conjunct of $c_{T_1,T_2,\ldots,T_k}$, the valuation $\phi := \psi \cup \bigcup_{i=1}^{k} \{(x_i, a_i)\}$ satisfies c. It means that $\phi|_{\mathrm{sig}(c)}$ is a compatible valuation in $\mathrm{va}(c)$, which, by the assumption, implies that $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$ and $[\![\mathcal{Q}_{T_1,T_2,\ldots,T_k}]\!]_\psi \succeq_{\mathrm{tr}} [\![\mathcal{P}_{T_1,T_2,\ldots,T_k}]\!]_\psi$. In other words, the answer to the instance $\mathcal{Q}_{U_1,U_2,\ldots,U_k}$, $\mathcal{P}_{U_1,U_2,\ldots,U_k}$, $c_{U_1,U_2,\ldots,U_k}$ is positive whenever $U_1, U_2, \ldots, U_k$ are type variables in $\mathrm{sig}(c)$.

Next, we assume that the answer to $\mathcal{Q}$, $\mathcal{P}$, c is negative. It means that there is a valuation $\phi \in \mathrm{va}(c)$ such that $[\![\mathcal{P}]\!]_\phi$ is not a traces refinement of $[\![\mathcal{Q}]\!]_\phi$. By the definition of a valuation, there are type variables $T_1, T_2, \ldots, T_k \in \mathrm{sig}(c)$ such that $\phi(x_i) \in \phi(T_i)$ for all $i \in \{1,2,\ldots,k\}$. It means that $\psi := \phi \cup \bigcup_{i=1}^{k} \{(\Pi_i, \{\phi(x_i)\})\}$ is a compatible

valuation such that it satisfies $c_{T_1,T_2,\ldots,T_k}$ and $[\![\mathcal{P}_{T_1,T_2,\ldots,T_k}]\!]_\phi$ is not a traces refinement of $[\![\mathcal{Q}_{T_1,T_2,\ldots,T_k}]\!]_\phi$. As the restriction of ψ to the signature of $c_{T_1,T_2,\ldots,T_k}$ has these properties as well, it implies that the answer to the instance $\mathcal{Q}_{T_1,T_2,\ldots,T_k}$, $\mathcal{P}_{T_1,T_2,\ldots,T_k}$, $c_{T_1,T_2,\ldots,T_k}$ is negative, too. □

Although the instances of Parameterised Traces Refinement, where the set of valuations is given with the aid of a valuation formula, can be finitely represented and therefore algorithmically manipulated, the problem is undecidable in general. The reason is that one can simulate a deterministic Turing machine as a family of systems with a ring topology, and a ring topology can be represented as a valuation formula. This claim will be justified more rigorously later in Chapter 8.

To overcome the problem, we concentrate on verification tasks expressible using existential-free valuation formulae. As will be seen later in conjunction with Lemma 31, it implies that we can treat only specification-system families that are closed under the removal of a replicated component. For example, systems with a star, bipartite and totally (un)connected topology are such, but those with a ring, linear or tree topology are not. However, if it is possible to capture the behaviour of the system from the viewpoint of any two components connected to each other in finitely many LTS schemata, then one can study the transitive closures of rings, arrays and forests instead, which are closed under the removal of a replicated component.

This is also the case with our shared tree system. Its behaviour is uniform from the viewpoint of a resource and its proper ancestor, which implies that we can capture the behaviour from the viewpoint of any two resources connected to each other in a single LTS schema, namely $User_2$. Therefore, the topology of our system model is actually not a tree but the transitive closure of a forest.

Additionally, we have to assume that the specification LTS schema does not have a hiding LTS subschema. Although existential-free valuation formulae cannot make a distinction between the linear and ring systems, the use of hiding enables us to create topology dependent specifications. It means that we can exclude linear systems, and hence analyse the behaviour of deterministic Turing machines, by altering the specification in such a way that it can be only violated by systems with a ring topology. Again, a more precise construction is given in Chapter 8.

As the use of hiding is, typically, only necessary in a system LTS schema, the restriction concerning specification LTS schemata is not very serious. However, there are

also more severe constraints implicitly present in the problem formulation, but all of them are more or less necessary, as will be pointed out in Chapter 8.

Because the correctness is understood as the traces refinement, only the analysis of safety properties is possible. However, the algorithmic checking of deadlock and liveness properties is out of the question without further restrictions on specification and system LTS schemata and valuation formulae. The reason is that also information on deadlocks can be used to distinct between systems with a linear and ring topology; one can alter the system and the specification in such a way that only system instances with a ring topology can deadlock and the specification can be only violated by such instances.

Finally, the definition of LTS schemata imposes restrictions on specifications and systems. Probably the most obvious shortcomings of the formalism are the lack of renaming LTS schemata and parametric branching within an elementary LTS schema. By parametric branching, we mean a construct that enables expressing systems where the number of paths between certain states is parameter-dependent but the length of the paths is not. From the modelling point of view, both renaming LTS schemata and parametric branching are useful constructs, but including either of them enables a deterministic Turing machine to be simulated as a family of two-stack machines. Also this claim will be rigorously justified in Chapter 8. Hence, without further restrictions on the system topology, the algorithmic analysis of systems with a richer structure is impossible.

To summarise, we have focused our attention on instances $\mathcal{Q}$, $\mathcal{P}$, va(c) of Parameterised Traces Refinement, where $\mathcal{Q}$ and $\mathcal{P}$ are closed and finite, $\mathcal{Q}$ does not have a hiding LTS subschema, and c is an existential-free valuation formula with the signature $\mathrm{sig}(\mathcal{Q}) \cup \mathrm{sig}(\mathcal{P})$. We will show that with these restrictions the problem can always be solved algorithmically by the principle of the precongruence reduction.

6 The Precongruence Reduction for Valuations

In order to show how the precongruence reduction can be applied to parameterised verification tasks expressed as instances of Parameterised Traces Refinement, it is appropriate to slightly relax the restrictions established in the previous chapter. Here, we consider instances $\mathcal{Q}, \mathcal{P}, \Phi$ such that $\mathcal{Q}$ does not involve hiding and Φ consists of valuations with the same domain. Moreover, because valuations that are not compatible with both the LTS schemata do not affect the answer to the verification task, we assume that Φ contains only valuations that are compatible with $\mathcal{Q}$ and $\mathcal{P}$.

Recall that the precongruence reduction is applied in two phases. First, Φ is reduced to its subset Φ' such that for every $\phi \in \Phi \setminus \Phi'$ there is a finite non-empty subset Θ of Φ' such that $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{tr}} \|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta$ and $\|_{\theta\in\Theta}[\![\mathcal{P}]\!]_\theta \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$. In other words, one removes valuations ϕ from Φ such that $[\![\mathcal{Q}]\!]_\phi$ can be under-approximated and $[\![\mathcal{P}]\!]_\phi$ over-approximated as a parallel composition of instances of respectively $\mathcal{Q}$ and $\mathcal{P}$ generated by a finite non-empty set $\Theta \subseteq \Phi$ of other valuations.

After that, Φ' is reduced to its subset Ψ such that for every $\phi \in \Phi' \setminus \Psi$ there is a valuation $\psi \in \Psi$ and a bijection $\zeta : \mathbb{V} \mapsto \mathbb{V}$ satisfying $\zeta([\![\mathcal{Q}]\!]_\phi) \succeq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ and $[\![\mathcal{P}]\!]_\psi \succeq_{\mathrm{tr}} \zeta([\![\mathcal{P}]\!]_\phi)$. Hence, one removes valuations ϕ from Φ' such that $[\![\mathcal{Q}]\!]_\phi$ can be under-approximated and $[\![\mathcal{P}]\!]_\phi$ over-approximated with the aid of bijectively renamed instances of respectively $\mathcal{Q}$ and $\mathcal{P}$ generated by another valuation $\psi \in \Phi'$.

To see when the reduction results in a finite set of valuations, we first characterise valuations which can be removed in two phases above. In other words, we want to characterise LTS schemata $\mathcal{Q}$ and $\mathcal{P}$ such that $\mathcal{Q}$ has no hiding LTS subschema, and

1. compatible valuations ϕ, ψ which have the same domain and for which there is a bijection $\zeta : \mathbb{V} \mapsto \mathbb{V}$ such that $\zeta([\![\mathcal{Q}]\!]_\phi) \succeq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ and $[\![\mathcal{P}]\!]_\psi \succeq_{\mathrm{tr}} \zeta([\![\mathcal{P}]\!]_\phi)$, and
2. compatible valuations ϕ and finite non-empty sets Θ of compatible valuations with the domain $\mathrm{dom}(\phi)$ such that $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{tr}} \|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta$ and $\|_{\theta\in\Theta}[\![\mathcal{P}]\!]_\theta \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$.

After that, we derive a sufficient condition for instances of Parameterised Traces Refinement for which the set of valuations expressed with the aid of a valuation formula can be reduced to a finite one without changing the answer to the verification question. We also specify a reduced instance for each instance satisfying the condition and compare our method with other cut-off results.

The preliminary versions of the results in this chapter are presented by Siirtola & Kortelainen (2009b,a). Here, the treatment is more thorough and the results slightly better in the sense of reduction. Moreover, the proofs are included and especially the comparison with the related work is more detailed.

6.1 Generalisation by Renaming

First, we characterise LTS schemata $\mathcal{Q}$ and $\mathcal{P}$, where $\mathcal{Q}$ has no hiding LTS subschema, and compatible valuations ϕ and ψ with the same domain for which there is a bijection $\zeta : \mathbb{V} \mapsto \mathbb{V}$ such that $[\![\mathcal{Q}]\!]_\psi \succeq_{\mathrm{tr}} \zeta([\![\mathcal{Q}]\!]_\phi)$ and $\zeta([\![\mathcal{P}]\!]_\phi) \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\psi$. Because the alphabets of the LTSs on both sides of the traces refinement have to match, it means that the alphabets of $[\![\mathcal{Q}]\!]_\psi$ and $[\![\mathcal{P}]\!]_\psi$ can be obtained from the alphabets of respectively $[\![\mathcal{Q}]\!]_\phi$ and $[\![\mathcal{P}]\!]_\phi$ by a bijective mapping of visible actions. It suggests that maybe also the valuation ψ could be obtained from the valuation ϕ by mapping the image of ψ to the image of ϕ bijectively atom-wise.

This notion motivates the following definition. Let ϕ be a valuation and $g : \mathbb{A} \mapsto \mathbb{A}$ a bijection. We write $g\phi$ for a function with the domain $\mathrm{dom}(\phi)$ such that

- $(g\phi)(T) = \{g(a) \mid a \in \phi(T)\}$ for all type variables T in the domain,
- $(g\phi)(\Pi) = \{g^*(\alpha) \mid \alpha \in \phi(\Pi)\}$ for all relation variables Π in the domain, and
- $(g\phi)(x) = g(\phi(x))$ for all atom variables x in the domain.

Recall that g^* denotes a function: $\mathbb{V} \mapsto \mathbb{V}$ such that

$$g^*((a_1, a_2, \ldots, a_k)) = (g(a_1), g(a_2), \ldots, g(a_k))$$

for all visible actions $(a_1, a_2, \ldots, a_k)$.

Lemma 21. *Let ϕ be a valuation, $g : \mathbb{A} \mapsto \mathbb{A}$ a bijection, $\mathcal{P}$ an LTS schema, Π a relation variable and $T_1, T_2, \ldots, T_k$ type variables in* $\mathrm{dom}(\phi)$*, and $x_1, x_2, \ldots, x_k$ any atom variables.*

1. *The function $g\phi$ is a valuation.*
2. *If g preserves the atoms in* $\mathrm{at}(\mathcal{P})$ *and ϕ is (partially) compatible with $\mathcal{P}$, then $g\phi$ is (partially) compatible with $\mathcal{P}$.*
3. *If $\phi' \in \phi[(x_1, x_2, \ldots, x_k) \mapsto \Pi]$, then $g\phi' \in (g\phi)[(x_1, x_2, \ldots, x_k) \mapsto \Pi]$.*
4. *If $\psi' \in (g\phi)[(x_1, x_2, \ldots, x_k) \mapsto \Pi]$, then there is $\phi' \in \phi[(x_1, x_2, \ldots, x_k) \mapsto \Pi]$ such that $g\phi' = \psi$.*

5. *If* $\phi' \in \phi[x_1 \mapsto T_1, \ldots, x_k \mapsto T_k]$, *then* $g\phi' \in (g\phi)[x_1 \mapsto T_1, \ldots, x_k \mapsto T_k]$.
6. *If* $\psi' \in (g\phi)[x_1 \mapsto T_1, \ldots, x_k \mapsto T_k]$, *then there is* $\phi' \in \phi[x_1 \mapsto T_1, \ldots, x_k \mapsto T_k]$ *such that* $g\phi' = \psi$.

The claims follow straightforwardly from definitions.

Valuations that can be obtained from each other by a bijective mapping of atoms in the above way have the property we are looking for: the instances of an LTS schema generated by them can be obtained from each other by bijective renaming. This is formally captured in the following lemma.

Lemma 22. *Let* $\mathcal{P}$ *be an LTS schema,* ϕ *a valuation compatible with* $\mathcal{P}$*, and* $g : \mathbb{A} \mapsto \mathbb{A}$ *a bijection that preserves the atoms in* $\mathrm{at}(\mathcal{P})$*. Then* g^* *is a bijection:* $\mathbb{V} \mapsto \mathbb{V}$ *such that* $g^*(\llbracket \mathcal{P} \rrbracket_\phi) = \llbracket \mathcal{P} \rrbracket_{g\phi}$.

Proof. It is easy to see that g^* is a bijection: $\mathbb{V} \mapsto \mathbb{V}$. By Lemma 21, $g\phi$ is a valuation compatible with $\mathcal{P}$. Hence, $g^*(\llbracket \mathcal{P} \rrbracket_\phi)$ and $\llbracket \mathcal{P} \rrbracket_\psi$ are LTSs. To prove that $g^*(\llbracket \mathcal{P} \rrbracket_\phi) = \llbracket \mathcal{P} \rrbracket_\psi$ we argue by induction on the structure of $\mathcal{P}$.

In the base step, we assume that $\mathcal{P}$ is an elementary LTS schema $(S, \Gamma, \Delta, \hat{s})$. The LTSs $g^*(\llbracket \mathcal{P} \rrbracket_\phi)$ and $\llbracket \mathcal{P} \rrbracket_{g\phi}$ clearly have the same state space and the initial state. Furthermore, the alphabet of $g^*(\llbracket \mathcal{P} \rrbracket_\phi)$ is

$$\begin{aligned}
&\{g^*(\alpha) \mid \alpha \in \llbracket \Gamma \rrbracket_\phi\} \\
&\qquad = \{(g(\phi(x_1)), \ldots, g(\phi(x_k)), g(a_1), \ldots, g(a_l)) \mid (x_1, \ldots, x_k, a_1, \ldots, a_l) \in \Gamma\} \\
&\qquad = \{((g\phi)(x_1), \ldots, (g\phi)(x_k), a_1, \ldots, a_l) \mid (x_1, \ldots, x_k, a_1, \ldots, a_l) \in \Gamma\} \\
&\qquad = \llbracket \Gamma \rrbracket_{g\phi} ,
\end{aligned}$$

which is exactly the alphabet of $\llbracket \mathcal{P} \rrbracket_{g\phi}$. Similarly, the transition relation of $g^*(\llbracket \mathcal{P} \rrbracket_\phi)$ is the same as $\llbracket \Delta \rrbracket_{g\phi}$, the transition relation of $\llbracket \mathcal{P} \rrbracket_{g\phi}$. Hence, $\llbracket \mathcal{P} \rrbracket_{g\phi}$ and $g^*(\llbracket \mathcal{P} \rrbracket_\phi)$ are the same LTS.

In the induction step, $\mathcal{P}$ is a parallel, replicated parallel or hiding LTS schema. If $\mathcal{P}$ is a parallel LTS schema $\mathcal{P}_1 \parallel \mathcal{P}_2$, then by Item 1 of Proposition 9 and the induction hypothesis,

$$\begin{aligned}
g^*(\llbracket \mathcal{P}_1 \parallel \mathcal{P}_2 \rrbracket_\phi) = g^*(\llbracket \mathcal{P}_1 \rrbracket_\phi \parallel \llbracket \mathcal{P}_2 \rrbracket_\phi) = \\
g^*(\llbracket \mathcal{P}_1 \rrbracket_\phi) \parallel g^*(\llbracket \mathcal{P}_2 \rrbracket_\phi) = \llbracket \mathcal{P}_1 \rrbracket_{g\phi} \parallel \llbracket \mathcal{P}_2 \rrbracket_{g\phi} = \llbracket \mathcal{P}_1 \parallel \mathcal{P}_2 \rrbracket_{g\phi} .
\end{aligned}$$

Next, let $\mathcal{P}$ be a replicated parallel LTS schema $\|_{(x_1,x_2,\ldots,x_k)\in\Pi:T_1\times T_2\times\ldots\times T_k}\mathcal{P}'$. First, note that by Item 1 of Proposition 9,

$$g^*(\llbracket \|_{(x_1,x_2,\ldots,x_k)\in\Pi:T_1\times T_2\times\ldots\times T_k}\mathcal{P}'\rrbracket_\phi) = g^*(\|_{\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]}\llbracket\mathcal{P}'\rrbracket_{\phi'}) = \|_{\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]} g^*(\llbracket\mathcal{P}'\rrbracket_{\phi'})\,.$$

Next, if ϕ' is a valuation in $\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]$, then by Lemma 16 it is compatible with $\mathcal{P}'$. By the induction hypothesis, it means that $g^*(\llbracket\mathcal{P}'\rrbracket_{\phi'}) = \llbracket\mathcal{P}'\rrbracket_{g\phi'}$. Hence, by Item 1 of Proposition 11,

$$\|_{\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]} g^*(\llbracket\mathcal{P}'\rrbracket_{\phi'}) = \|_{\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]}\llbracket\mathcal{P}'\rrbracket_{g\phi'}\,.$$

Finally, by Lemma 21,

$$\|_{\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi]}\llbracket\mathcal{P}'\rrbracket_{g\phi'} = \|_{\psi'\in(g\phi)[(x_1,x_2,\ldots,x_k)\mapsto\Pi]}\llbracket\mathcal{P}'\rrbracket_{\psi'} = \llbracket\|_{(x_1,x_2,\ldots,x_k)\in\Pi:T_1\times T_2\times\ldots\times T_k}\mathcal{P}'\rrbracket_{g\phi}\,.$$

Hence, the claim holds also in the case $\mathcal{P}$ is a replicated parallel LTS schema.

Finally, if $\mathcal{P}$ is a hiding LTS schema $\mathcal{P}'\setminus\bigcup_{(x_1,x_2,\ldots,x_k)\in T_1\times T_2\times\ldots\times T_k}\Gamma$, then by Proposition 9,

$$\begin{aligned}
&g^*(\llbracket\mathcal{P}'\setminus\bigcup_{(x_1,x_2,\ldots,x_k)\in T_1\times T_2\times\ldots\times T_k}\Gamma\rrbracket_\phi)\\
&= g^*(\llbracket\mathcal{P}'\rrbracket_\phi\setminus\bigcup_{\phi'\in\phi[x_1\mapsto T_1,x_2\mapsto T_2,\ldots,x_k\mapsto T_k]}\llbracket\Gamma\rrbracket_{\phi'})\\
&= g^*(\llbracket\mathcal{P}'\rrbracket_\phi)\setminus\{g^*(\alpha)\mid\alpha\in\bigcup_{\phi'\in\phi[x_1\mapsto T_1,x_2\mapsto T_2,\ldots,x_k\mapsto T_k]}\llbracket\Gamma\rrbracket_{\phi'}\}\\
&= g^*(\llbracket\mathcal{P}'\rrbracket_\phi)\setminus\bigcup_{\phi'\in\phi[x_1\mapsto T_1,x_2\mapsto T_2,\ldots,x_k\mapsto T_k]}\{g^*(\alpha)\mid\alpha\in\llbracket\Gamma\rrbracket_{\phi'}\}\,.
\end{aligned}$$

Now, the induction hypothesis is clearly applicable to $\mathcal{P}'$, which means that $g^*(\llbracket\mathcal{P}'\rrbracket_\phi)$ and $\llbracket\mathcal{P}'\rrbracket_{g\phi}$ are the same LTS. Moreover, just like in the case of an elementary LTS schema, it is easy to see that $\{g^*(\alpha)\mid\alpha\in\llbracket\Gamma\rrbracket_{\phi'}\} = \llbracket\Gamma\rrbracket_{g\phi'}$ whenever ϕ' is a valuation

in $\phi[x_1 \mapsto T_1, x_2 \mapsto T_2, \ldots, x_k \mapsto T_k]$. Therefore, by Lemma 21,

$$\begin{aligned} g^*([\![\mathcal{P}']\!]_\phi) \setminus & \bigcup_{\phi' \in \phi[x_1 \mapsto T_1, x_2 \mapsto T_2, \ldots, x_k \mapsto T_k]} \{g^*(\alpha) \mid \alpha \in [\![\Gamma]\!]_{\phi'}\} \\ &= [\![\mathcal{P}']\!]_{g\phi} \setminus \bigcup_{\phi' \in \phi[x_1 \mapsto T_1, x_2 \mapsto T_2, \ldots, x_k \mapsto T_k]} [\![\Gamma]\!]_{g\phi'} \\ &= [\![\mathcal{P}']\!]_{g\phi} \setminus \bigcup_{\psi' \in (g\phi)[x_1 \mapsto T_1, x_2 \mapsto T_2, \ldots, x_k \mapsto T_k]} [\![\Gamma]\!]_{\psi'} \\ &= [\![\mathcal{P}' \setminus \bigcup_{(x_1, x_2, \ldots, x_k) \in T_1 \times T_2 \times \ldots \times T_k} \Gamma]\!]_{g\phi} . \end{aligned}$$

Hence, also the last case is clear, which by the induction principle means that the lemma is correct. □

We say that valuations ϕ and ψ are *isomorphic* or that ϕ is *isomorphic* to ψ, denoted by $\phi \simeq \psi$, if $g\phi = \psi$ for some bijection $g : \mathbb{A} \mapsto \mathbb{A}$. If ϕ and ψ are not isomorphic, then they are *non-isomorphic*, which is denoted by $\phi \not\simeq \psi$.

Lemma 23. *Relation $\simeq$ is an equivalence over the set of valuations.*

The claim follows straightforwardly from definitions.

Lemma 22 implies that the instances generated by isomorphic valuations can be obtained from each other by the application of a bijective renaming operator determined by the valuations. By Theorem 12, it means that one can always generalise the answer to an instance $\mathcal{Q}, \mathcal{P}, \Phi$ of Parameterised Traces Refinement to other instances obtained by adding valuations isomorphic to some valuations $\phi \in \Phi$ such that ϕ is compatible with $\mathcal{Q}$ and $\mathcal{P}$.

Proposition 24. *Let $\mathcal{Q}, \mathcal{P}, \Phi$ be an instance of Parameterised Traces Refinement, and Ψ a subset of Φ such that for every valuation $\phi \in \Phi$ there is an isomorphic valuation $\psi \in \Psi$ that is compatible with $\mathcal{Q}$ and $\mathcal{P}$. Then the instances $\mathcal{Q}, \mathcal{P}, \Phi$ and $\mathcal{Q}, \mathcal{P}, \Psi$ have the same answer.*

Proof. Let $\mathcal{Q}, \mathcal{P}, \Phi$ and Ψ as assumed in the Proposition. Because Ψ is a subset of Φ, it is clear that the answer to $\mathcal{Q}, \mathcal{P}, \Psi$ is positive whenever the answer to $\mathcal{Q}, \mathcal{P}, \Phi$ is.

Let us then assume that the answer to $\mathcal{Q}, \mathcal{P}, \Psi$ is positive. If ϕ is a valuation in $\Phi \setminus \Psi$ compatible with $\mathcal{Q}$ and $\mathcal{P}$, then by the assumption there is an isomorphic valuation $\psi \in \Psi$ that is compatible with both the LTS schemata. By definition, it means there is a bijection $g : \mathbb{A} \mapsto \mathbb{A}$ such that $\phi = g\psi$. As both the valuations are compatible

with $\mathcal{P}$ and $\mathcal{Q}$, we may assume that g preserves the atoms in $\mathrm{at}(\mathcal{Q}) \cup \mathrm{at}(\mathcal{P})$. By Lemma 22, then

$$[\![\mathcal{Q}]\!]_\phi = [\![\mathcal{Q}]\!]_{g\psi} = g^*([\![\mathcal{Q}]\!]_\psi) \text{ and } g^*([\![\mathcal{P}]\!]_\psi) = [\![\mathcal{P}]\!]_{g\psi} = [\![\mathcal{P}]\!]_\phi \,.$$

By Theorem 12, it implies that the answer to the instance $\mathcal{Q}$, $\mathcal{P}$, Φ', where Φ' consists of all the valuations in Φ compatible with $\mathcal{Q}$ and $\mathcal{P}$, is positive, too. Because adding valuations that are not compatible with both the LTS schemata do not affect the answer to the instance, the answer to the instance $\mathcal{Q}$, $\mathcal{P}$, Φ is positive, too.

Hence, the instances $\mathcal{Q}$, $\mathcal{P}$, Φ and $\mathcal{Q}$, $\mathcal{P}$, Ψ have the same answer, and the proposition is correct. □

The proposition implies that there is no need to check both the valuations that differ only in the atoms chosen for the values of variables. However, to be able to pick representative valuations, we should know which atoms occur in the values of variables. Fortunately, the answer is immediate: any atoms that do not occur in the LTS schemata $\mathcal{Q}$ or $\mathcal{P}$ will do nicely, if the set of valuations is given as a valuation formula.

Lemma 25. *If c is a valuation formula and ϕ, ψ are isomorphic valuations compatible with c, then $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true.*

The claim follows by induction on the structure of the valuation formula using the lemma as an induction hypothesis.

The lemma together with Proposition 24 means that when the valuations are specified using a valuation formula the choice of atoms for the values of type variables is insignificant as long as the atoms do not occur in the system or specification LTS schema. For example, to verify all the instances of the system with at most two users and three resources, it is sufficient to pick any five different atoms a_1, a_2, b_1, b_2, b_3 for user and resource identities that do not occur in *Mtx* and $Sts \setminus La$, and check whether $[\![Mtx]\!]_\phi \succeq_{\mathrm{tr}} [\![Sts \setminus La]\!]_\phi$ for all compatible valuations $\phi \in \Phi_{Srs}$ such that $\phi(U) = \{a_1\}, \{a_1, a_2\}$ and $\phi(R) = \{b_1\}, \{b_1, b_2\}, \{b_1, b_2, b_3\}$.

Actually, the proposition allows even better reduction, because only 14 of the total of 40 such valuations are non-isomorphic. One possible choice for a set of representative valuations is depicted in Figure 20, where arrows represent connections between resources in such a way that the resource at the beginning of an arrow is a proper ancestor of the resource in the other end. Hence, the value of $<_R$ consists of all the pairs of connected resources. The values of other relation variables are not explicitly shown,

because they can be obtained unambiguously from those of U, R and $<_R$ shown in the figure. Hence, to verify all the instances of the system with at most two users and three resources, it is sufficient to check those generated by the valuations in Figure 20.

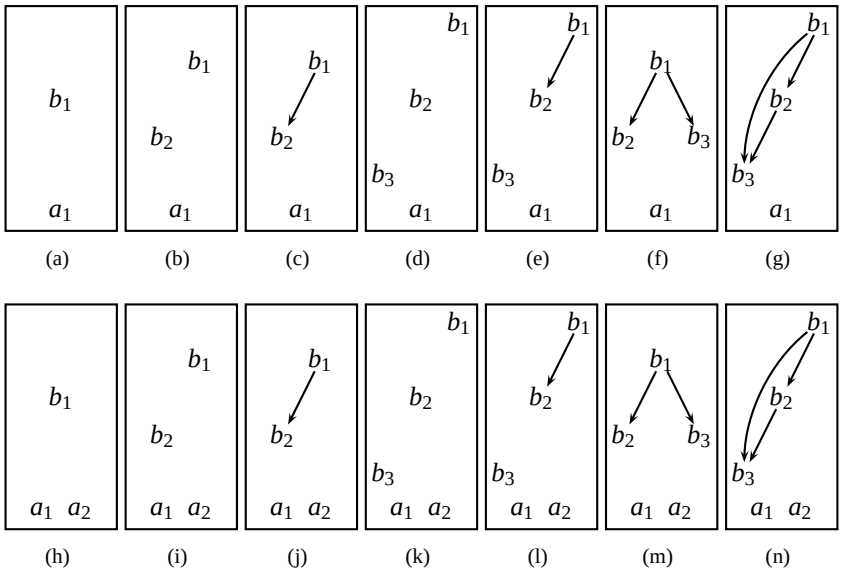

Fig 20. Representative valuations with at most two users and three resources.

6.2 Generalisation by Parallel Composition

Next, we consider how to approximate an instance of an LTS schema as a parallel composition of other instances of the LTS schema. In other words, we want to characterise LTS schemata $\mathcal{Q}$ and $\mathcal{P}$, where $\mathcal{Q}$ does not have a hiding LTS subschema, compatible valuations ϕ, and finite non-empty sets Θ of compatible valuations with the domain $\mathrm{dom}(\phi)$ such that $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{tr}} \|_{\theta\in\Theta} [\![\mathcal{Q}]\!]_\theta$ and $\|_{\theta\in\Theta} [\![\mathcal{P}]\!]_\theta \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$. Because the alphabets on both sides of the traces refinement have to match, the alphabet of the instance generated by a valuation in Θ should be a subset of the alphabet of the instance generated by ϕ. Intuitively, it means that a valuation in Θ can contain only a subset of the information about ϕ.

The notion above motivates the following definition. We say that a valuation ψ is a *subvaluation* of a valuation ϕ, denoted by $\psi \sqsubseteq \phi$, if the valuations have the same domain,

1. $\psi(U) \subseteq \phi(U)$ for all type variables U in the domain,
2. $\psi(\Xi) \subseteq \phi(\Xi)$ for all relation variables Ξ in the domain, and
3. $\psi(y) = \phi(y)$ for all atom variables y in the domain.

Lemma 26. *Let ϕ be a valuation, ψ a subvaluation of ϕ, $\mathcal{P}$ an LTS schema, Π a relation variable in* $\mathrm{dom}(\phi)$*, and $x_1, x_2, \ldots, x_k$ any atom variables.*

1. *If ϕ is compatible with $\mathcal{P}$, then ψ is compatible with $\mathcal{P}$, too.*
2. *If $\phi' \in \phi[(x_1, x_2, \ldots, x_k) \mapsto \Pi]$ and $\psi' \in \psi[(x_1, x_2, \ldots, x_k) \mapsto \Pi]$, then ψ' is a subvaluation of ϕ' if and only if $\phi'(x_i) = \psi'(x_i)$ for all $i \in \{1, 2, \ldots, k\}$.*

Proof. To prove that ψ is compatible with $\mathcal{P}$ as well, we need to show that ψ defines values for all the variables in $\mathrm{sig}(\mathcal{P})$ without using the atoms in $\mathrm{at}(\mathcal{P})$, and $\psi(\Pi)$ is a subset of $\psi(T_1) \times \psi(T_2) \times \ldots \times \psi(T_n)$ whenever Π is a relation variable in $sig(\mathcal{P})$ and $T_1 \times T_2 \times \ldots \times T_n$ is a type of Π in $\mathcal{P}$.

Because ϕ and ψ have the same domain, and the atoms occurring in the image of ψ are those occurring in the image of ϕ, it is obvious that ψ defines values for all the variables in $\mathrm{sig}(\mathcal{P})$ without using the atoms in $\mathrm{at}(\mathcal{P})$.

If Π is a relation variable in $sig(\mathcal{P})$ and $T_1 \times T_2 \times \ldots \times T_n$ is a type of Π in $\mathcal{P}$, then, by the fact that ψ is a subvaluation of ϕ,

$$\psi(\Pi) \subseteq \phi(\Pi) \subseteq \phi(T_1) \times \phi(T_2) \times \ldots \times \phi(T_n) .$$

On the other hand, as $\Pi \in \mathrm{dom}(\psi)$, by the definition of a valuation, there are type variables $U_1, U_2, \ldots, U_k \in \mathrm{dom}(\psi)$ such that

$$\psi(\Pi) \subseteq \psi(U_1) \times \psi(U_2) \times \ldots \times \psi(U_k) \subseteq \phi(U_1) \times \phi(U_2) \times \ldots \times \phi(U_k) .$$

If $\psi(\Pi)$ is empty, then obviously $\psi(\Pi)$ is a subset of $\psi(T_1) \times \psi(T_2) \times \ldots \times \psi(T_n)$. Otherwise, k and n have to match and

$$\psi(\Pi) \subseteq (\phi(T_1) \cap \phi(U_1)) \times (\phi(T_2) \cap \phi(U_2)) \times \ldots \times (\phi(T_n) \cap \phi(U_n)) .$$

Because a valuation maps type variables to disjoint sets of atoms, it implies that T_i and U_i must denote the same type variable for every $i \in \{1, 2, \ldots, n\}$. Hence, by above, $\psi(\Pi)$ is a subset of $\psi(T_1) \times \psi(T_2) \times \ldots \times \psi(T_n)$.

The latter claim follows straightforwardly from definitions. □

It is easy to see that if every $\theta \in \Theta$ is a subvaluation of ϕ, then the alphabets of $\|_{\theta \in \Theta} [\![\mathcal{Q}]\!]_\theta$ and $\|_{\theta \in \Theta} [\![\mathcal{P}]\!]_\theta$ are subsets of respectively the alphabets of $[\![\mathcal{Q}]\!]_\phi$ and $[\![\mathcal{P}]\!]_\phi$. However, to guarantee also the opposite inclusion, it is not sufficient just to require that

$\mathbf{P}_0(\phi,\Theta)$: for every type and relation variable $\sharp \in \mathrm{dom}(\phi)$ and every $\alpha \in \phi(\sharp)$ there is $\theta \in \Theta$ such that $\alpha \in \theta(\sharp)$.

To see this, suppose that $\mathcal{R}'$ is an elementary LTS schema with an alphabet schema $\{(x_1,x_2)\}$ and $\mathcal{R}$ is an LTS schema $\|_{x_1\in\Pi_1:T_1}\|_{x_2\in\Pi_2:T_2}\mathcal{R}'$. Now, the alphabets of $[\![\mathcal{R}]\!]_\phi$ and $\|_{\theta\in\Theta}[\![\mathcal{R}]\!]_\theta$ match only if $\theta(\Pi_i)\subseteq\phi(\Pi_i)$ for both $i\in\{1,2\}$, and for all $a_1\in\phi(\Pi_1)$ and $a_2\in\phi(\Pi_2)$ there is some $\theta\in\Theta$ such that $a_1\in\theta(\Pi_1)$ and $a_2\in\theta(\Pi_2)$. This is also true when Π_1 and Π_2 denote the same relation variable or T_1 and T_2 are the same type variable. Hence, a strictly stronger assumption than $\mathbf{P}_0$ is needed to make the alphabets of $\|_{\theta\in\Theta}[\![\mathcal{R}]\!]_\theta$ and $[\![\mathcal{R}]\!]_\phi$ equal.

The example suggests that we should pay attention to actions $\alpha_1,\alpha_2,\ldots,\alpha_k$ which can be picked from the values ϕ assigns to respectively relation variables $\Pi_1,\Pi_2,\ldots,\Pi_k$ nested in $\mathcal{P}$ or $\mathcal{Q}$, and make sure that there is always some valuation $\theta\in\Theta$ such that $\alpha_i\in\theta(\Pi_i)$ for all $i\in\{1,2,\ldots,k\}$. To put it more formally, we say that LTS schemata $\mathcal{R}_0,\mathcal{R}_1,\ldots,\mathcal{R}_n$ are *nested*, if $\mathcal{R}_i$ is an LTS subschema of $\mathcal{R}_{i-1}$ for all $i\in\{1,2\ldots,n\}$. Moreover, relation variables $\Pi_1,\Pi_2,\ldots,\Pi_n$ are called *nested* in an LTS schema $\mathcal{R}$, if there are nested LTS subschemata $\mathcal{R}_1,\mathcal{R}_2,\ldots,\mathcal{R}_n$ of $\mathcal{R}$ such that $\mathcal{R}_i$ is a Π_i-replicated parallel LTS schema for each $i\in\{1,2,\ldots,n\}$. Hence, to make sure that the alphabets of $\|_{\theta\in\Theta}[\![\mathcal{R}]\!]_\theta$ and $[\![\mathcal{R}]\!]_\phi$ match, for both $\mathcal{R}\in\{\mathcal{Q},\mathcal{P}\}$, we require the following.

$\mathbf{P}(\mathcal{R},\phi,\Theta)$: Whenever $\Pi_1,\Pi_2,\ldots,\Pi_k$ are relation variables nested in $\mathcal{R}$ and $\alpha_1,\ldots,\alpha_k$ are actions in respectively $\phi(\Pi_1),\phi(\Pi_2),\ldots,\phi(\Pi_k)$, then there is $\theta\in\Theta$ such that $\alpha_i\in\theta(\Pi_i)$ for all $i\in\{1,2,\ldots,k\}$.

Now, if $\mathcal{R}$ is an LTS schema, every $\theta\in\Theta$ is a subvaluation of ϕ, and $\mathbf{P}(\mathcal{R},\phi,\Theta)$ holds, then the alphabets of $[\![\mathcal{R}]\!]_\phi$ and $\|_{\theta\in\Theta}[\![\mathcal{R}]\!]_\theta$ match. Actually, the assumption also implies a traces refinement or a traces equivalence between $\|_{\theta\in\Theta}[\![\mathcal{R}]\!]_\theta$ and $[\![\mathcal{R}]\!]_\phi$ depending on whether $\mathcal{R}$ involves respectively hiding or not.

To see this, note that the precise form of an instance of an elementary LTS subschema $\mathcal{R}'$ of $\mathcal{R}$ depends on the values assigned to atom variables occurring in $\mathcal{R}'$. The values of free atom variables are obtained directly from the valuation and the values of other atom variables are picked from the values of relation variables nested in $\mathcal{R}$. Assuming every $\theta\in\Theta$ to be a subvaluation of ϕ guarantees that the values of free atom variables match and whenever one can pick some values for other atom variables while evaluating $\|_{\theta\in\Theta}[\![\mathcal{R}]\!]_\theta$ then one can pick the same values for the variables during the evaluation of $[\![\mathcal{R}]\!]_\phi$ as well. The assumption $\mathbf{P}(\mathcal{R},\phi,\Theta)$ guarantees that also the oppo-

site holds, which implies that both $[\![\mathcal{R}]\!]_\phi$ and $\|_{\theta\in\Theta}[\![\mathcal{R}]\!]_\theta$ are composed from the same LTSs.

In the case of the specification LTS schema $\mathcal{Q}$ which does not have a hiding LTS subschema, it means that both $[\![\mathcal{Q}]\!]_\phi$ and $\|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta$ are constructed from the same instances of the same elementary LTS schemata using the parallel composition only. The only difference in the construction of the LTSs is that the instances of the elementary LTS schemata may be composed in parallel in a different order, some instances may occur in $\|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta$ more often than in $[\![\mathcal{Q}]\!]_\phi$, and $\|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta$ may contain more LTSs with the empty alphabet. However, because parallel composition is commutative, associative and idempotent, and every LTS with the empty alphabet is an identity element, all the differences are insignificant from the viewpoint of the traces equivalence. Therefore, $[\![\mathcal{Q}]\!]_\phi$ and $\|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta$ are necessarily traces equivalent.

Lemma 27. *Let $\mathcal{Q}$ be an LTS schema without a hiding LTS subschema, ϕ a valuation compatible with $\mathcal{Q}$, and Θ a finite non-empty set of subvaluations of ϕ such that* $\mathbf{P}(\mathcal{Q},\phi,\Theta)$. *Then* $[\![\mathcal{Q}]\!]_\phi =_{\mathrm{tr}} \|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta$.

Proof. We argue by induction on the structure of $\mathcal{Q}$ using the lemma as an induction hypothesis.

In the base step, we assume that $\mathcal{Q}$ is an elementary LTS schema $(S,\Gamma,\Delta,\hat{s})$. Because $\theta(y) = \phi(y)$ whenever θ is a valuation in Θ and y an atom variable in $\mathrm{sig}(\mathcal{Q})$, the LTSs $[\![\mathcal{Q}]\!]_\phi$ and $[\![\mathcal{Q}]\!]_\theta$ have not only the same state space and the initial state, but the same alphabet and the transition relation. Hence, $[\![\mathcal{Q}]\!]_\phi$ and $[\![\mathcal{Q}]\!]_\theta$ are the same LTS for every $\theta\in\Theta$. By the idempotence of the parallel composition, it means that $\|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta =_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$.

In the induction step, $\mathcal{Q}$ is either a parallel or replicated parallel LTS schema. If $\mathcal{Q}$ is a parallel LTS schema $\mathcal{Q}_1 \parallel \mathcal{Q}_2$, then $[\![\mathcal{Q}]\!]_\phi =_{\mathrm{tr}} [\![\mathcal{Q}_1]\!]_\phi \parallel [\![\mathcal{Q}_2]\!]_\phi$, and the induction hypothesis is applicable to both $[\![\mathcal{Q}_1]\!]_\phi$ and $[\![\mathcal{Q}_2]\!]_\phi$. By the associativity and commutativity of the parallel composition and by the congruence of the traces equivalence, it implies that

$$\begin{aligned}\|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta = \|_{\theta\in\Theta}([\![\mathcal{Q}_1]\!]_\theta \parallel [\![\mathcal{Q}_2]\!]_\theta) &=_{\mathrm{tr}} (\|_{\theta\in\Theta}[\![\mathcal{Q}_1]\!]_\theta) \parallel (\|_{\theta\in\Theta}[\![\mathcal{Q}_2]\!]_\theta)\\ &=_{\mathrm{tr}} [\![\mathcal{Q}_1]\!]_\phi \parallel [\![\mathcal{Q}_2]\!]_\phi = [\![\mathcal{Q}]\!]_\phi\,.\end{aligned}$$

Hence, the claim holds in the case $\mathcal{Q}$ is a parallel LTS schema.

Next, it is assumed that $\mathcal{Q}$ is a replicated parallel LTS schema

$$\|_{(x_1,x_2,\ldots,x_k)\in\Pi_0:T_1\times T_2\times\ldots\times T_k} \mathcal{Q}' .$$

If $\phi(\Pi_0)$ is empty, then also $\theta(\Pi_0)$ is empty whenever θ is a valuation in Θ, which means that $[\![\mathcal{Q}]\!]_\theta$ and $[\![\mathcal{Q}]\!]_\phi$ are the same LTS for every $\theta\in\Theta$. By the idempotence of the parallel composition, it implies that $\|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta =_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$.

If $\phi(\Pi_0)$ is not empty, let ϕ' be a valuation in $\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$. Moreover, let α_0 denote the action $(\phi'(x_1),\phi'(x_2),\ldots,\phi'(x_k))$ and let $\Theta_{\phi'}$ be the set of all valuations $\theta'\in\bigcup_{\theta\in\Theta}\theta[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$ such that $\theta'(x_i)=\phi'(x_i)$ for all $i\in\{1,2,\ldots,k\}$. By Lemmas 26 and 16, the valuations in $\Theta_{\phi'}$ are subvaluations of ϕ' and compatible with $\mathcal{Q}'$.

To show that $\mathbf{P}(\mathcal{Q}',\phi',\Theta_{\phi'})$, too, let $\Pi_1,\Pi_2,\cdots,\Pi_n$ be nested relation variables in $\mathcal{Q}'$ and $\alpha_1,\alpha_2,\ldots,\alpha_n$ actions in respectively $\phi'(\Pi_1),\phi'(\Pi_2),\ldots,\phi'(\Pi_n)$. Now, the relation variables $\Pi_0,\Pi_1,\cdots,\Pi_n$ are nested in $\mathcal{Q}$ and $\alpha_i\in\phi(\Pi_i)$ for all $i\in\{0,1,\ldots,n\}$, which by $\mathbf{P}(\mathcal{Q},\phi,\Theta)$ means that there is a valuation $\theta\in\Theta$ such that $\alpha_i\in\theta(\Pi_i)$ for all $i\in\{0,1,\ldots,n\}$. As $\alpha_0\in\theta(\Pi_0)$, there is a valuation $\theta'\in\theta[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$ such that $\theta'(x_i)=\phi'(x_i)$ for all $i\in\{1,2,\ldots,k\}$. Therefore, θ' is a valuation in $\Theta_{\phi'}$ such that $\alpha_i\in\theta'(\Pi_i)$ for every $i\in\{1,2,\ldots,n\}$, which also implies that $\Theta_{\phi'}$ is a non-empty set. Hence, by the induction hypothesis, $\|_{\theta'\in\Theta_{\phi'}}[\![\mathcal{Q}']\!]_{\theta'} =_{\mathrm{tr}} [\![\mathcal{Q}']\!]_{\phi'}$ for every valuation $\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$.

By definition, if θ' is a valuation in $\Theta_{\phi'}$ for some $\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$, then θ' is obviously in $\theta[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$ for some $\theta\in\Theta$. However, also the opposite is true. If $\theta'\in\theta[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$ for some $\theta\in\Theta$, then, because θ is a subvaluation of ϕ, there is a unique valuation $\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$ such that $\theta'(x_i)=\phi'(x_i)$ for all $i\in\{1,2,\ldots,k\}$, *i.e.* $\theta'\in\Theta_{\phi'}$. In other words, the sets $\Theta_{\phi'}$, where $\phi'\in\phi[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$, form a partition of $\bigcup_{\theta\in\Theta}\theta[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]$. By above and Proposition 8, it implies that

$$\begin{aligned}
\|_{\theta\in\Theta}[\![\mathcal{Q}]\!]_\theta &= \|_{\theta\in\Theta}\,(\|_{\theta'\in\theta[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]}[\![\mathcal{Q}']\!]_{\theta'}) \\
&\overset{(\mathrm{P8.3})}{=_{\mathrm{tr}}} \|_{\substack{\theta\in\Theta\\ \theta(\Pi_0)\neq\emptyset}}\,(\|_{\theta'\in\theta[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]}[\![\mathcal{Q}']\!]_{\theta'}) \\
&\overset{(\mathrm{P8.4\&P8.2\&P8.1})}{=_{\mathrm{tr}}} \|_{\theta'\in\bigcup_{\theta\in\Theta}\theta[(x_1,x_2,\ldots,x_k)\mapsto\Pi_0]}[\![\mathcal{Q}']\!]_{\theta'}
\end{aligned}$$

$$
\begin{aligned}
&= \mathop{\|}_{\theta' \in \bigcup_{\phi' \in \phi[(x_1,x_2,\ldots,x_k) \mapsto \Pi_0]} \Theta_{\phi'}} [\![\mathcal{Q}']\!]_{\theta'} \\
&\stackrel{\text{(P8.2\&P8.1)}}{=_{\mathrm{tr}}} \mathop{\|}_{\phi' \in \phi[(x_1,x_2,\ldots,x_k) \mapsto \Pi_0]} \Big(\mathop{\|}_{\theta' \in \Theta_{\phi'}} [\![\mathcal{Q}']\!]_{\theta'} \Big) \\
&\stackrel{\text{(P11.1)}}{=_{\mathrm{tr}}} \mathop{\|}_{\phi' \in \phi[(x_1,x_2,\ldots,x_k) \mapsto \Pi_0]} [\![\mathcal{Q}']\!]_{\phi'} = [\![\mathcal{Q}]\!]_{\phi} \, .
\end{aligned}
$$

Hence, the claim holds also in the case $\mathcal{Q}$ is a replicated parallel LTS schema.

By the induction principle, the lemma is correct. □

Note that the lemma does not hold for LTS schemata that have a hiding LTS subschema. To see this, assume that $\mathcal{R}$ is an LTS schema

$$
\Big(\mathop{\|}_{x \in \Pi : T} (\{s_1, s_2\}, \{x, a\}, \{(s_1, x, s_2), (s_2, a, s_1)\}, s_1) \Big) \setminus \{a\} \, .
$$

If $\phi(\Pi) = \{a_1, a_2, \ldots, a_n\}$, where $n \geq 2$ is an integer, $a_1, a_2, \ldots, a_n$ are distinct atoms different from a, and $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ is a set of subvaluations of ϕ such that $\theta_i(\Pi) = \{a_i\}$ for all $i \in \{1, 2, \ldots, n\}$, then $\mathbf{P}(\mathcal{R}, \phi, \Theta)$ is satisfied but $\|_{\theta \in \Theta} [\![\mathcal{R}]\!]_\theta$ and $[\![\mathcal{R}]\!]_\phi$ are not traces equivalent. For example, a_1^* and a_2^* are included in the traces of $\|_{\theta \in \Theta} [\![\mathcal{R}]\!]_\theta$ but not in the traces of $[\![\mathcal{R}]\!]_\phi$. The reason is that hiding takes place earlier in $\|_{\theta \in \Theta} [\![\mathcal{R}]\!]_\theta$ than in $[\![\mathcal{R}]\!]_\phi$, and therefore there is no synchronisation between the instances of elementary LTS schemata in the former LTS, which means that the instances can execute totally independently. In the latter LTS, however, all the instances have to synchronise on the action a, which makes it impossible for them to execute totally independently.

In the light of the fact that the use of hiding in specifications makes the problem undecidable, the observation above is natural. Fortunately, for system LTS schemata $\mathcal{P}$, which typically involve hiding, it is sufficient to establish a traces refinement between $\|_{\theta \in \Theta} [\![\mathcal{P}]\!]_\theta$ and $[\![\mathcal{P}]\!]_\phi$. To see that it can be done, note that now both the LTSs are constructed from the same instances of the same elementary LTS schemata using the parallel composition and hiding. Moreover, in both $[\![\mathcal{P}]\!]_\phi$ and $\|_{\theta \in \Theta} [\![\mathcal{P}]\!]_\theta$, the same actions are hidden but in the latter LTS hiding is distributed in smaller parts, *i.e.* there is less synchronisation between instances of elementary LTS schemata than in $[\![\mathcal{P}]\!]_\phi$. In the light of Proposition 10, it implies that $[\![\mathcal{P}]\!]_\phi$ is a traces refinement of $\|_{\theta \in \Theta} [\![\mathcal{P}]\!]_\theta$.

Lemma 28. *Let $\mathcal{P}$ be an LTS schema, ϕ a valuation compatible with $\mathcal{P}$, and Θ a finite non-empty set of subvaluations of ϕ such that $\mathbf{P}(\mathcal{P}, \phi, \Theta)$. Then $\|_{\theta \in \Theta} [\![\mathcal{P}]\!]_\theta \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$.*

Proof. We argue by induction on the structure of $\mathcal{P}$ using the lemma as an induction hypothesis.

The cases where $\mathcal{P}$ is an elementary, parallel or replicated parallel LTS schema are handled similarly as in the proof of Lemma 27. One just uses the traces refinement instead of the traces equivalence. Therefore, it is sufficient to consider only the case where $\mathcal{P}$ is a hiding LTS schema.

Hence, let us assume that $\mathcal{P}$ is a hiding LTS schema $\mathcal{P}' \setminus \bigcup_{(x_1,x_2,\ldots,x_n)\in T_1\times T_2\times\ldots\times T_n} \Gamma$. We write Λ_ψ for the set $\bigcup_{\psi'\in\psi[x_1\mapsto T_1,x_2\mapsto T_2,\ldots,x_n\mapsto T_n]} [\![\Gamma]\!]_{\psi'}$ whenever ψ is a valuation compatible with $\mathcal{P}$. By the induction hypothesis, $\|_{\theta\in\Theta} [\![\mathcal{P}']\!]_\theta \succeq_{\mathrm{tr}} [\![\mathcal{P}']\!]_\phi$, which, by Item 1 of Proposition 10, and the precongruence of the traces refinement means that

$$\|_{\theta\in\Theta} ([\![\mathcal{P}']\!]_\theta \setminus \Lambda_\phi) \succeq_{\mathrm{tr}} (\|_{\theta\in\Theta} [\![\mathcal{P}']\!]_\theta) \setminus \Lambda_\phi \succeq_{\mathrm{tr}} [\![\mathcal{P}']\!]_\phi \setminus \Lambda_\phi = [\![\mathcal{P}]\!]_\phi \,.$$

If $\alpha \in \Lambda_\phi$, then there is an action schema $(y_1,\ldots,y_k,a_1,\ldots,a_l) \in \Gamma$ and a valuation $\phi' \in \phi[x_1 \mapsto T_1,\ldots,x_n \mapsto T_n]$ such that $\alpha = (\phi'(y_1),\ldots,\phi'(y_k),a_1,\ldots,a_l)$. If θ is a valuation in Θ such that $\alpha \notin \Lambda_\theta$, then there must be $i \in \{1,2,\ldots,k\}$ and a type variable U such that $\phi'(y_i) \in \phi(U) \setminus \theta(U)$. By induction on the structure of $\mathcal{P}'$ it is easy to show that then $\alpha \notin \mathrm{alph}([\![\mathcal{P}']\!]_\theta)$ either. Therefore, $\mathrm{alph}([\![\mathcal{P}']\!]_\theta)$ and $\Lambda_\phi \setminus \Lambda_\theta$ are disjoint whenever $\theta \in \Theta$, which, by Items 4 and 3 of Proposition 10, implies that

$$\begin{aligned}[\![\mathcal{P}']\!]_\theta \setminus \Lambda_\phi &= [\![\mathcal{P}']\!]_\theta \setminus ((\Lambda_\phi \setminus \Lambda_\theta) \cup (\Lambda_\phi \cap \Lambda_\theta)) \\ &= ([\![\mathcal{P}']\!]_\theta \setminus (\Lambda_\phi \setminus \Lambda_\theta)) \setminus (\Lambda_\phi \cap \Lambda_\theta) = [\![\mathcal{P}']\!]_\theta \setminus (\Lambda_\phi \cap \Lambda_\theta) \,.\end{aligned}$$

Because the valuations in Θ are subvaluations of ϕ, the set Λ_θ is clearly a subset of Λ_ϕ for all $\theta \in \Theta$, which by above, implies that $[\![\mathcal{P}']\!]_\theta \setminus \Lambda_\phi$ and $[\![\mathcal{P}']\!]_\theta \setminus \Lambda_\theta$ are the same LTS for every $\theta \in \Theta$. Therefore, by the precongruence of the traces refinement and above,

$$\|_{\theta\in\Theta} [\![\mathcal{P}]\!]_\theta = \|_{\theta\in\Theta} ([\![\mathcal{P}']\!]_\theta \setminus \Lambda_\theta) = \|_{\theta\in\Theta} ([\![\mathcal{P}']\!]_\theta \setminus \Lambda_\phi) \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi \,,$$

and the claim holds also in the case $\mathcal{P}$ is a hiding LTS schema.

By the induction principle, the lemma is correct. □

Lemmas 27 and 28 imply that one can represent the instance of a specification LTS schema and over-approximate the instance of a system LTS schema generated by a valuation ϕ as the parallel compositions of the instances generated by certain subvaluations of ϕ. By Theorem 12, it means that one can always add valuations ϕ to an instance $\mathcal{Q}, \mathcal{P}, \Phi$ of Parameterised Traces Refinement without changing its answer, if there are

sufficiently many and sufficiently big subvaluations of ϕ in Φ and $\mathcal{Q}$ does not involve hiding.

Proposition 29. *Let $\mathcal{Q}$, $\mathcal{P}$, Φ be an instance of Parameterised Traces Refinement such that $\mathcal{Q}$ has no hiding LTS subschema, and let Φ' be a subset of Φ. If for every $\phi \in \Phi \setminus \Phi'$ there is a finite non-empty set $\Theta \subseteq \Phi'$ of subvaluations of ϕ such that $\mathbf{P}(\mathcal{Q}, \phi, \Theta)$ and $\mathbf{P}(\mathcal{P}, \phi, \Theta)$, then the instances $\mathcal{Q}$, $\mathcal{P}$, Φ and $\mathcal{Q}$, $\mathcal{P}$, Φ' have the same answer.*

Proof. Because Φ' is a subset of Φ, it is clear that the answer to $\mathcal{Q}$, $\mathcal{P}$, Φ' is positive whenever the answer to $\mathcal{Q}$, $\mathcal{P}$, Φ is.

Let us then assume that the answer to $\mathcal{Q}$, $\mathcal{P}$, Φ' is positive. If ϕ is a valuation in $\Phi \setminus \Phi'$ compatible with $\mathcal{Q}$ and $\mathcal{P}$, then by the assumption there is a finite non-empty set $\Theta \subseteq \Phi'$ of subvaluations of ϕ such that $\mathbf{P}(\mathcal{Q}, \phi, \Theta)$ and $\mathbf{P}(\mathcal{P}, \phi, \Theta)$. By Lemmas 27 and 28, it means that $[\![\mathcal{Q}]\!]_\phi =_{\mathrm{tr}} \|_{\theta \in \Theta} [\![\mathcal{Q}]\!]_\theta$ and $\|_{\theta \in \Theta} [\![\mathcal{P}]\!]_\theta \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$. By Theorem 12, it implies that the answer to the instance $\mathcal{Q}$, $\mathcal{P}$, Φ'', where Φ'' consists of all the valuations in Φ compatible $\mathcal{Q}$ and $\mathcal{P}$, is positive, too. Because adding valuations that are not compatible with both the LTS schemata do not affect the answer to an instance, also the answer to the instance $\mathcal{Q}$, $\mathcal{P}$, Φ is positive.

Hence, the instances $\mathcal{Q}$, $\mathcal{P}$, Φ and $\mathcal{Q}$, $\mathcal{P}$, Ψ have the same answer, and the proposition is correct. □

In the case of the shared tree system, the proposition implies that one can derive the correctness of the whole system from instances with at most two users and three resources. To see it, you must first note that maximal sequences of nested relation variables in *Mtx* and $Sts \setminus La$ are $*_U, \leq_R$; $*_U, <_R$; $*_R, \neq_U$; $\leqq_R, \neq_U$ and $\leqq_R, *_U$. Next, let ϕ be a valuation in $\mathrm{va}(c_{Sts})$ such that $|\phi(U)| > 2$ or $|\phi(R)| > 3$, and let Θ be the set of all subvaluations θ of ϕ such that $|\theta(U)| \leq 2$ and $|\theta(R)| \leq 3$. The set Θ is clearly finite and non-empty. To see that it satisfies Condition **P**, too, consider for example the sequence $\leqq_R, \neq_U$ of relation variables nested in *Mtx*. It is easy to see that whenever $(b'_1, b'_2, b'_3) \in \phi(\leqq_R)$ and $(a'_1, a'_2) \in \phi(\neq_U)$, then there is a valuation $\theta \in \Theta$ such that $(b'_1, b'_2, b'_3) \in \theta(\leqq_R)$ and $(a'_1, a'_2) \in \theta(\neq_U)$. Similarly, one can see that an analogous result holds for other sequences of nested relation variables, too, which means that $\mathbf{P}(Mtx, \phi, \Theta)$ and $\mathbf{P}(Sts \setminus La, \phi, \Theta)$ are satisfied. By Proposition 29, it implies that the shared resource system is correct with respect to the mutual exclusion property irrespective of the values of the parameters, if and only if all the instances of the system with at most two users and three resources work as specified.

6.3 Infinite Reduction

We have now converted the principle of the precongruence reduction to the context of LTS schemata and valuations in the form of Propositions 24 and 29. Next, we show that it can be successfully applied to instances $\mathcal{Q}$, $\mathcal{P}$, va(c) of Parameterised Traces Refinement, where $\mathcal{Q}$ and $\mathcal{P}$ are finite and closed, $\mathcal{Q}$ has no hiding LTS subschema, and c is an existential-free valuation formula with the signature $\mathrm{sig}(\mathcal{Q}) \cup \mathrm{sig}(\mathcal{P})$. In other words, we prove that when you first reduce va(c) to its subset Φ' using Proposition 29, and then Φ' to its subset using Proposition 24, you can always obtain a finite set Ψ of valuations.

In practice, it is not only important to be able to reduce va(c) to its finite subset Ψ, but also to minimise the cost of deciding the resulting instance $\mathcal{Q}$, $\mathcal{P}$, Ψ. That is why, in both the steps, we want to maximise reduction, or in other words, to minimise Φ' and Ψ, and especially the valuations included in the sets.

Minimisation of Φ' is limited by Condition **P**. To see how small Φ' and its elements can be, let ϕ be a valuation in va(c), $\Pi_1, \Pi_2, \cdots, \Pi_k$ nested relation variables in $\mathcal{P}$ or $\mathcal{Q}$, and $\alpha_1, \alpha_2, \ldots, \alpha_k$ actions in respectively $\phi(\Pi_1), \phi(\Pi_2), \ldots, \phi(\Pi_k)$. Now, there should be a subvaluation θ of ϕ in Φ' such that $\alpha_i \in \theta(\Pi_i)$ for all $i \in \{1, 2, \ldots, k\}$. A minimal valuation that has this property is a subvaluation θ of ϕ which maps Π_i to $\{\alpha_i\}$ for all $i \in \{1, 2, \ldots, k\}$, other relation variables in $\mathrm{dom}(\theta)$ to the empty set, and each type variable $T \in \mathrm{dom}(\theta)$ to a minimal non-empty set containing all atoms $a \in \phi(T)$ such that a occurs in some of the actions $\alpha_1, \alpha_2, \ldots, \alpha_k$. Because $\mathcal{Q}$ and $\mathcal{P}$ are closed, it means that θ maps each type variable $T \in \mathrm{dom}(\theta)$ to a singleton or to the set of all atoms $a \in \phi(T)$ occurring in $\alpha_1, \alpha_2, \ldots, \alpha_k$.

Unfortunately, such a minimal subvaluation does not always exist. For example, in the case of the shared tree system, $*_R, \neq_U$ are relation variables nested in *Mtx*. Whenever ϕ is a valuation in va(c_{Sts}) and b, a_1, a_2 are atoms such that $b \in \phi(*_R)$ and $(a_1, a_2) \in \phi(\neq_U)$, then the minimal subvaluation θ of ϕ of the above kind maps R to $\{b\}$, U to $\{a_1, a_2\}$ and $\neq_U$ to $\{(a_1, a_2)\}$. The valuation θ is not in va(c_{Sts}), but the verification method works still very well in the case of the shared tree system, because there is a slightly bigger subvaluation $\theta' \in \mathrm{va}(c_{Sts})$ which is otherwise like θ but maps $\neq_U$ to $\{(a_1, a_2), (a_2, a_1)\}$. Therefore, the minimisation of valuations in the above way may be too aggressive.

Note that θ' can be obtained from ϕ in a simple way: just remove an action α from the values of variables whenever α cannot be expressed as a tuple of atoms b, a_1, a_2 oc-

curring in the actions $b, (a_1, a_2)$ picked from the values of relation variables. It suggests that instead of minimising the values of all the variables, we should minimise only the values of type variables and restrict the values of relation variables accordingly.

This observation motivates the following definition. A valuation ψ is *equal to or smaller than* a valuation ϕ, denoted by $\psi \trianglelefteq \phi$, if ψ is a subvaluation of ϕ such that

$$\psi(\Pi) = \phi(\Pi) \cap (\bigcup_{k \in \mathbb{Z}_+} \bigcup_{A_1, A_2, \ldots, A_k \in \mathrm{im}(\psi|_{\mathbb{T}})} A_1 \times A_2 \times \ldots \times A_k) \, ,$$

for every relation variable Π in the domain. A valuation ψ is said to be *(strictly) smaller than* ϕ, if $\psi \trianglelefteq \phi$ and $\psi \neq \phi$.

It turns out that all the sets of valuations arising from existential-free valuation formulae are downward closed respect to $\trianglelefteq$. In other words, whenever a valuation is in such a set, then all the smaller valuations are in the set as well. We prove the result for negation-normal valuation formulae first.

Lemma 30. *Let c be a negation-normal existential-free valuation formula and ϕ, ψ valuations compatible with c such that $\psi \trianglelefteq \phi$. If $[\![c]\!]_\phi$ is true, then $[\![c]\!]_\psi$ is true, too.*

Proof. To prove the claim, we show a slightly stronger claim which implies the lemma. We prove by induction on the structure of c that whenever $[\![c]\!]_\phi$ is true, then $[\![c]\!]_\psi$ is true as well, and if c has no universal valuation subformula, then $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true.

In the base step, c is an elementary valuation formula. If c is $\top$, then obviously $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true.

If c is $x_1 = x_2$, where x_1, x_2 are atom variables, then $\phi(x_1) = \phi(x_2)$ if and only if $\psi(x_1) = \psi(x_2)$, because $\phi(y) = \psi(y)$ for all atom variables y in the domain. Hence, $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true.

The last case in the base step is that c is $(x_1, x_2, \ldots, x_n) \in \Pi$, where $x_1, x_2, \ldots, x_n$ are atom variables and Π a relation variable. Because ψ is equal to or smaller than ϕ, clearly $(\psi(x_1), \psi(x_2), \ldots, \psi(x_n)) \in \psi(\Pi)$ implies $(\phi(x_1), \phi(x_2), \ldots, \phi(x_n)) \in \phi(\Pi)$. On the other hand, for every $i \in \{1, 2, \ldots, n\}$, $\phi(x_i) = \psi(x_i)$ and there is $A \in \mathrm{im}(\psi)$ such that $\psi(x_i) \in A$. Hence, if $(\phi(x_1), \phi(x_2), \ldots, \phi(x_n)) \in \phi(\Pi)$, then

$$(\psi(x_1), \psi(x_2), \ldots, \psi(x_n)) \in \bigcup_{k \in \mathbb{Z}_+} \bigcup_{A_1, A_2, \ldots, A_k \in \mathrm{im}(\psi|_{\mathbb{T}})} (\phi(\Pi) \cap \bigtimes_{i=1}^{k} A_i) =$$

$$\phi(\Pi) \cap (\bigcup_{k \in \mathbb{Z}_+} \bigcup_{A_1, A_2, \ldots, A_k \in \mathrm{im}(\psi|_{\mathbb{T}})} A_1 \times A_2 \times \ldots \times A_k) = \psi(\Pi) \, .$$

Therefore, $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true.

In the induction step, c is either a negated, disjunctive, conjunctive or universal valuation formula. If c is a negated valuation formula $\neg c'$, then c' is an elementary valuation formula and clearly negation-normal. By the induction hypothesis, it implies that $[\![c']\!]_\phi$ is true if and only if $[\![c']\!]_\psi$ is true. Hence, $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true.

Let us then assume that c is $c_1 \odot c_2$, where $\odot$ is either $\vee$ or $\wedge$. By the induction hypothesis, it implies that if $i \in \{1,2\}$ and $[\![c_i]\!]_\phi$ is true, then $[\![c_i]\!]_\psi$ is true, too. Therefore, if $[\![c]\!]_\phi$ is true, then $[\![c]\!]_\psi$ is also true. If c has no universal valuation subformula, then c_1 and c_2 do not have such a valuation subformula either. By the induction hypothesis, it implies that for both $i \in \{1,2\}$, $[\![c_i]\!]_\phi$ is true if and only if $[\![c_i]\!]_\psi$ is true. Hence, $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true.

Finally, if c is a universal valuation formula $\forall x \in T : c'$ and $[\![c]\!]_\phi$ is true, then $[\![c']\!]_{\phi'}$ is true for all valuations $\phi' \in \phi[x \mapsto T]$. If ψ' is a valuation in $\psi[x \mapsto T]$, then there is a valuation $\phi' \in \phi[x \mapsto T]$ such that $\phi'(x) = \psi'(x)$. Clearly, ψ' is compatible with c' and equal to or smaller than ϕ'. As $[\![c']\!]_{\phi'}$ is true, by the induction hypothesis, $[\![c']\!]_{\psi'}$ is true as well. Hence, if $[\![c]\!]_\phi$ is true, then $[\![c]\!]_\psi$ is true as well.

Hence, by the induction principle, the lemma holds. □

To understand why the lemma holds, suppose that ϕ is a valuation satisfying a negation-normal existential-free valuation formula c, and consider what happens if one makes ϕ smaller by removing all the actions involving an atom a from the values of variables such that $a \neq \phi(x)$ for all atom variables $x \in \mathrm{dom}(\phi)$. It is clear that if universal quantification over some set gives a positive result, then universal quantification over its subset must give a positive result, too. Because type variables are only used in universal quantification, one cannot make the instance of a negation-normal existential-free valuation formula false by removing a from the value of a type variable. On the other hand, the reduced instance of c cannot state anything on the relationship of a to other atoms anymore, which means it is also safe to cut down the values of relation variables by removing all the actions involving a.

Now, it is easy to prove the corresponding result for any existential-free valuation formula.

Lemma 31. *Let c be an existential-free valuation formula and ϕ, ψ valuations compatible with c such that $\psi \trianglelefteq \phi$. If $[\![c]\!]_\phi$ is true, then $[\![c]\!]_\psi$ is true, too.*

Proof. Using De Morgan's rules, we convert c into an existential-free negation-normal valuation formula c' with the domain $\mathrm{sig}(c)$ such that every valuation θ is compatible with c and satisfies c, if and only if θ is compatible with c' and satisfies c'. By Lemma 30, we know that whenever $[\![c']\!]_\phi$ is true, then $[\![c']\!]_\psi$ is true as well, which implies that if ϕ satisfies c, then also ψ satisfies c. □

The lemma cannot be extended to arbitrary valuation formulae, which feels very natural because the verification problem we consider becomes undecidable when such valuation formulae are allowed. As an example, consider a closed valuation formula $c := \neg\forall x, y \in T : x = y$ that is not existential-free. Clearly, a valuation ϕ with the domain $\{T\}$ such that $|\phi(T)| = 2$ satisfies the formula meaning that ϕ is in $\mathrm{va}(c)$. Now, there is a valuation ϕ' with the domain $\{T\}$ such that $\phi'(T)$ is a singleton subset of $\phi(T)$. The valuation ϕ' is then equal to or smaller than ϕ but $[\![c]\!]_{\phi'}$ is not true. Hence, ϕ' is not in $\mathrm{va}(c)$ and therefore $\mathrm{va}(c)$ is not downward closed with respect to $\trianglelefteq$.

In the light of above, we know now that whenever a valuation is in the set represented by an existential-free valuation formula, then all the smaller ones in the sense of $\trianglelefteq$ are also in the set. Informally, it means that by restricting our attention to existential-free valuation formulae, we can consider only specification-system families that are closed under the removal of a replicated component. On the other hand, by earlier observations, it also means that we need to preserve only valuations $\phi \in \mathrm{va}(c)$ that map each type variable T in the domain to a singleton or to a set A for which there are relation variables $\Pi_1, \Pi_2, \ldots, \Pi_k$ nested in $\mathcal{Q}$ or $\mathcal{P}$, and actions $\alpha_1 \in \phi(\Pi_1)$, $\alpha_2 \in \phi(\Pi_2), \ldots, \alpha_k \in \phi(\Pi_k)$ such that every atom in A occurs in $\alpha_1, \alpha_2, \ldots, \alpha_k$.

To put it more formally, let T be a type variable, $\mathcal{R}$ an LTS schema, and θ a valuation. The maximum number of different atoms $a \in \theta(T)$ occurring in any actions $\alpha_1 \in \phi(\Pi_1), \alpha_2 \in \phi(\Pi_2), \ldots, \alpha_n \in \phi(\Pi_n)$ such that $\Pi_1, \Pi_2, \ldots, \Pi_n$ are relation variables nested in $\mathcal{R}$ is called *the T-degree (of $\mathcal{R}$ and θ)* and denoted by $\deg_T(\mathcal{R}, \theta)$. The maximum number of occurrences of T in the types of relation variables nested in $\mathcal{R}$, called *the T-degree (of $\mathcal{R}$)* and denoted by $\deg_T(\mathcal{R})$, gives an upper bound for $\deg_T(\mathcal{R}, \theta)$. Hence, the valuations $\phi \in \mathrm{va}(c)$ that are preserved and therefore form the reduced set Φ' are exactly those that map each type variable T in the domain to a singleton or to a set of size $\max\{\deg_T(\mathcal{Q}, \phi), \deg_T(\mathcal{P}, \phi)\}$.

As an example, you may consider the shared tree system again. Recall that the maximal sequences of relation variables nested in $Sts \setminus La$ are $*_R, *_U$; $*_U, \leq_R$; $*_U, <_R$ and $*_R, \neq_U$, and the maximal sequences of relation variables nested in *Mtx* are $\leq_R, \neq_U$

and $\leqq_R, *_U$. Now, the U-degree of either LTS schema and a valuation $\phi \in \mathrm{va}(c_{Sts})$ is one or two depending on whether $\phi(U)$ is respectively a singleton or not. Also the R-degree of the system LTS schema and valuations $\phi \in \mathrm{va}(c_{Sts})$ is either one or two depending on whether $\phi(<_R)$ is respectively empty or not. However, the R-degree of *Mtx* and valuations $\phi \in \mathrm{va}(c_{Sts})$ is either one, two or three depending on how many different atoms may occur in an action in $\phi(\leqq_R)$. Therefore, to prove the system correct, it is sufficient to concentrate on valuations $\phi \in \mathrm{va}(c_{Sts})$ such that $|\phi(U)| \leq 2$ and either $\phi(R)$ is a singleton, $\phi(R)$ is of size two and $\phi(<_R)$ is non-empty, or $\phi(R)$ is a set of three atoms all of which occur in some action in $\phi(\leqq_R)$.

Having obtained Φ' minimising Ψ using Proposition 24 is simple: pick all the valuations in Φ' that are compatible with $\mathcal{Q}$ and $\mathcal{P}$, and then remove all isomorphic copies, *i.e.* preserve exactly one valuation per equivalence class of $\sim$.

It is quite easy to see that the set Ψ determined as above is finite. Consider the valuations in Ψ. The size of the values of type variables is bounded, so each type variable can have only finitely many essentially different (non-isomorphic) values. Moreover, because all valuations $\psi \in \Psi$ are compatible with the LTS schemata and every relation variable $\Pi \in \mathrm{dom}(\psi)$ occurs in $\mathcal{Q}$ or $\mathcal{P}$, the values of Π are limited to subsets of the values of its types. Hence, also every relation variable can have only finitely many non-isomorphic values. As all the valuations in Ψ are non-isomorphic and have the same finite domain which contains no atom variable, it is clear Ψ cannot be infinite.

Theorem 32. *Let $\mathcal{Q}$ and $\mathcal{P}$ be closed and finite LTS schemata and c an existential-free valuation formula such that $\mathcal{Q}$ has no hiding LTS subschema and* $\mathrm{sig}(c) = \mathrm{sig}(\mathcal{Q}) \cup \mathrm{sig}(\mathcal{P})$. *A maximal set Ψ of non-isomorphic valuations in the set*

$$\{\phi \in \mathrm{va}(c) \mid \phi \textit{ is compatible with } \mathcal{Q} \textit{ and } \mathcal{P}\textit{, and } \forall T \in \mathrm{dom}(\phi|_{\mathbb{T}}) : |\phi(T)| = \max\{1, \deg_T(\mathcal{Q}, \phi), \deg_T(\mathcal{P}, \phi)\}\}$$

is finite and the answer to both the instances $\mathcal{Q}$, $\mathcal{P}$, Ψ and $\mathcal{Q}$, $\mathcal{P}$, $\mathrm{va}(c)$ *is the same.*

Proof. Let Φ' denote the set

$$\{\phi \in \mathrm{va}(c) \mid \forall T \in \mathrm{dom}(\phi|_{\mathbb{T}}) : |\phi(T)| = \max\{1, \deg_T(\mathcal{Q}, \phi), \deg_T(\mathcal{P}, \phi)\}\}$$

and let Ψ' be the set of all the valuations in Φ' compatible with $\mathcal{Q}$ and $\mathcal{P}$. Now, Ψ is a maximal set of non-isomorphic valuations in Ψ'. Because Ψ is clearly a subset of $\mathrm{va}(c)$, the answer to $\mathcal{Q}$, $\mathcal{P}$, Ψ is positive whenever the answer to $\mathcal{Q}$, $\mathcal{P}$, $\mathrm{va}(c)$ is.

To prove the contrary, assume that the answer to $\mathcal{Q}$, $\mathcal{P}$, Ψ is positive. Let ϕ be a valuation in $\mathrm{va}(c) \setminus \Phi'$, and let Θ denote the set of all valuations $\theta \trianglelefteq \phi$ such that $|\theta(T)| = \max\{1, \deg_T(\mathcal{Q}, \theta), \deg_T(\mathcal{P}, \theta)\}$ for all type variables $T \in \mathrm{dom}(\theta)$. Clearly, Θ is a finite non-empty set of subvaluations of ϕ. Moreover, by Lemma 31, Θ is a subset of $\mathrm{va}(c)$, which implies that Θ is also a subset of Φ'.

To see that Θ also satisfies Condition **P**, let $\Pi_1, \Pi_2, \ldots, \Pi_k$ be a sequence of nested relation variables in either of the LTS schemata, and $\alpha_1, \alpha_2, \ldots, \alpha_k$ actions in respectively $\phi(\Pi_1), \phi(\Pi_2), \ldots, \phi(\Pi_k)$. For every type variable $T \in \mathrm{dom}(\phi)$, let B_T denote the set of all atoms $a \in \phi(T)$ such that a occurs in some of the actions $\alpha_1, \alpha_2, \ldots, \alpha_k$. Now, let θ be a valuation equal to or smaller than ϕ such that for each type variable T in the domain, $\theta(T) = B_T$ when B_T is non-empty, and $\theta(T) = \{b\}$ for some $b \in \phi(T)$ when B_T is empty. Therefore, $\alpha_1 \in \theta(\Pi_1), \alpha_2 \in \theta(\Pi_2), \ldots, \alpha_k \in \theta(\Pi_k)$, and

$$|\theta(T)| = \max\{1, |B_T|\} = \max\{1, \deg_T(\mathcal{Q}, \theta), \deg_T(\mathcal{P}, \theta)\}$$

for every type variable $T \in \mathrm{dom}(\theta)$. It implies that θ is in Θ, which means that $\mathbf{P}(\mathcal{Q}, \phi, \Theta)$ and $\mathbf{P}(\mathcal{P}, \phi, \Theta)$. Therefore, by Proposition 29, the answer to the instance $\mathcal{Q}$, $\mathcal{P}$, Φ' is positive, and because Ψ' is obtained from Φ' by removing all the valuations that are not compatible with both $\mathcal{Q}$ and $\mathcal{P}$, also the answer to $\mathcal{Q}$, $\mathcal{P}$, Ψ' must be positive.

Next, if ψ is a valuation in $\Psi' \setminus \Psi$, there is an isomorphic valuation in Ψ that is compatible with $\mathcal{Q}$ and $\mathcal{P}$. By Proposition 24, it implies that the answer to $\mathcal{Q}$, $\mathcal{P}$, Ψ is positive, too.

Hence, we proved that the answer to both the instances $\mathcal{Q}$, $\mathcal{P}$, Ψ and $\mathcal{Q}$, $\mathcal{P}$, $\mathrm{va}(c)$ is the same. We still have to show that Ψ is finite.

Let Υ denote the set of all functions $\upsilon : \mathrm{sig}(c) \cap \mathbb{T} \mapsto \mathbb{Z}_+$ such that $\upsilon(T)$ is at most $\max\{1, \deg_T(\mathcal{Q}), \deg_T(\mathcal{P})\}$ for all the type variables $T \in \mathrm{sig}(c)$. Because the signature of c is finite, Υ is a finite set. For every $\upsilon \in \Upsilon$, let us then write Φ_υ for the set of all valuations $\phi \in \mathrm{va}(c)$ compatible with $\mathcal{Q}$ and $\mathcal{P}$ such that $|\phi(T)| = \upsilon(T)$ for each type variable $T \in \mathrm{dom}(\phi)$. Because Ψ' is clearly a subset of $\bigcup_{\upsilon \in \Upsilon} \Phi_\upsilon$ and Υ is finite, to prove that Ψ is finite, it is sufficient to show that a maximal set Ψ_υ of non-isomorphic valuations in Φ_υ is finite for each $\upsilon \in \Upsilon$.

Let $\upsilon \in \Upsilon$ and let ϕ_υ be a valuation with the domain $\mathrm{sig}(c)$ such that ϕ_υ is compatible with $\mathcal{Q}$ and $\mathcal{P}$, and $|\phi_\upsilon(T)| = \upsilon(T)$ for all type variables $T \in \mathrm{sig}(c)$. If ϕ is a valuation in Φ_υ, then there is a valuation ψ_ϕ such that ϕ is isomorphic to ψ_ϕ and $\psi_\phi(T) = \phi_\upsilon(T)$ for all type variables $T \in \mathrm{sig}(c)$. Now, let Ψ'_υ denote the set $\{\psi_\phi \mid \phi \in \Psi_\upsilon\}$, which is

as large as Ψ_υ because all the valuations in Ψ_υ are pairwise non-isomorphic. It will be shown that Ψ'_υ is finite, which implies that Ψ_υ is finite, too.

Clearly, the valuations in Ψ'_υ agree on the values of type variables. Furthermore, every valuation $\psi \in \Psi'_\upsilon$ has the same finite domain $\mathrm{sig}(c)$, which means that every relation variable $\Pi \in \mathrm{dom}(\psi)$ occurs in $\mathcal{Q}$ or $\mathcal{P}$. As valuations in Ψ'_υ are compatible with both the LTS schemata, it implies that ψ maps each relation variable Π in the domain to a subset of the finite set $\bigtimes_{i=1}^{k} \phi_\upsilon(T_i)$, where $T_1 \times T_2 \times \cdots \times T_k$ is a type of Π. Hence, for each relation variable there are only finitely many possible values. Furthermore, because the valuation formula is closed, there is no atom variable in $\mathrm{sig}(c)$. Therefore, Ψ'_υ and Ψ_υ are finite sets, which implies that Ψ is finite, too. □

Recall that in the case of the shared tree system, the U-degree of both $Sts \setminus La$ and *Mtx* is two, and the R-degree is two for the system and three for the specification LTS schema. Hence, to prove the system correct, it is sufficient to consider compatible valuations that map U and R to sets of size at most respectively two and three. Also recall that only 14 such valuations are pairwise non-isomorphic, for example those depicted in Figure 20. However, Valuations (b), (d), (e), (f), (i), (k), (l) and (m) map the type variable R to a set greater than their (and the LTS schemas') R-degree, which implies that there is no need to consider those valuations. Hence, by Theorem 32, to prove the shared tree system correct irrespective of the number of users and the size and shape of the forest, it is sufficient to check a total of six of its instances generated by for example Valuations (a), (c), (g), (h), (j) and (n).

The instances were encoded in CSP modelling language and verification was performed using a finite-state refinement checker FDR2. All the instances were found to be correct, which implies that the shared tree system works as specified.

6.4 Comparison with the Related Work

Probably the most closely related research is done by Emerson & Kahlon (2000) and Bouajjani *et al.* (2008), whose results partly overlap those presented here. The former group considers multiparameterised systems

$$MPS := (\,\|_{j=1}^{n_1} P_{1,j}) \,\|\, (\,\|_{j=1}^{n_2} P_{2,j}) \,\|\, \ldots \,\|\, (\,\|_{j=1}^{n_k} P_{k,j})\,,$$

where k is a positive integer, $n_1, n_2, \ldots, n_k$ are parameters ranging over positive integers, and for each $i = \{1, 2, \ldots, k\}$, structures $P_{i,1}, P_{i,2}, \ldots, P_{i,n_i}$ are processes created from

a template TP_i by subscripting the states with respectively $1,2,\ldots,n_i$. The processes execute with interleaving semantics but transitions are allowed to have a guard, which must evaluate to true before a transition can be fired. A *guard* is basically a set of states, and guards come in two forms. A *conjunctive* guard g evaluates to true if each of the other processes is in some of the states in g, whereas a *disjunctive* guard evaluates to true if at least one other process is in some of the states in g.

The properties considered by Emerson & Kahlon (2000) are related to the states of one or two arbitrary processes and expressed in linear time temporal logic without the next-time operator (LTL-X), which enables expressing both safety and liveness properties. Hence, the verification problem they consider is if a multiparameterised system is correct with respect to an LTL-X specification for all parameter values, *i.e.* whenever $n_1,n_2,\ldots,n_k \in \mathbb{Z}_+$.

Disjunctive guards are naturally modelled with the aid of parameterised branching introduced in Chapter 8, and hence cannot be handled in our formalism. On the other hand, parameters cannot be nested, and the state space of the processes and the system topology are fixed, so for example the shared tree system cannot be represented using the formalism of Emerson & Kahlon (2000). However, systems with conjunctive guards can be treated in our formalism in the sense that one can create an LTS schema whose instances have the same reachability properties as the original system.

To model a multiparameterised system as an LTS schema, for every $i \in \{1,2,\ldots,k\}$, we introduce a type variable T_i to represent the set of the identities of processes generated from the template TP_i. Moreover, we pick $k+1$ atom variables $x,y_1,y_2,\ldots,y_k$, and a unique atom a_t for each transition t occurring in the process templates. The atom variable x refers to the current process whereas $y_1,y_2,\ldots,y_k$ are meant to denote processes of other types or other processes of the same type generated from respectively templates $TP_1,TP_2,\ldots,TP_k$.

A process generated from a template TP_i, where $i \in \{1,2,\ldots,k\}$, is modelled as an elementary LTS schema $\mathcal{P}_{MPS,i}$, whose state space and initial state are the same as those of TP_i. Each transition t of TP_i from a state s_1 to a state s_2, gives rise to a transition schema $(s_1,(x,a_t),s_2)$, and whenever g is the guard of a transition t' of a process template TP_j, where $j \in \{1,2,\ldots,k\}$, and $s \in g$ is a state of TP_i, also $(s,(y_i,a_{t'}),s)$ is a transition schema of $\mathcal{P}_{MPS,i}$. These are the only transition and action schemata of $\mathcal{P}_{MPS,i}$, and their purpose is to make all the instances synchronise on precisely the actions arising from conjunctively guarded transitions.

Because atom variables x and y_j occurring in $\mathcal{P}_i$ are meant to represent the identities of different processes whenever $i \in \{1,2,\ldots,k\}$, the LTS schema can be used in the presence of two or more processes only. To cover the case when $n_i = 1$ for each $i \in \{1,2,\ldots,k\}$, too, we introduce an LTS schema $\mathcal{P}'_{MPS,i}$ which is obtained from $\mathcal{P}_{MPS,i}$ by removing all the transition schemata not involving x.

Now, to build the system, we introduce unique relation variables $*_i, \neq_i$ for each $i \in \{1,2,\ldots,k\}$. As usual, $*_i$ denotes the sets of the identities of processes and $\neq_i$ represents the set of all the pairs of identities of different processes generated from the template TP_i. Hence, a multiparameterised system can be expressed as an LTS schema

$$\mathcal{P}_{MPS} := \mathcal{P}_1 \parallel \mathcal{P}_2 \parallel \cdots \parallel \mathcal{P}_k \,,$$

where $\mathcal{P}_i$ is

$$\mathop{\Vert}_{y_1 \in *_1 : T_1} \cdots \mathop{\Vert}_{y_{i-1} \in *_{i-1} : T_{i-1}} \mathop{\Vert}_{y_{i+1} \in *_{i+1} : T_{i+1}} \cdots \mathop{\Vert}_{y_k \in *_k : T_k} \Big(\Big(\mathop{\Vert}_{(x,y_i) \in \neq_i : T_i \times T_i} \mathcal{P}_{MPS,i} \Big) \parallel \Big(\mathop{\Vert}_{x \in *_i : T_i} \mathcal{P}'_{MPS,i} \Big) \Big)$$

for all $i \in \{1,2,\ldots,k\}$. Moreover, the set of all the valuations that give the intended meanings to the type and relation variables above is represented by an existential-free valuation formula

$$c_{MPS} := \bigwedge_{i=1}^{k} \big((\forall x \in T_i : x \in *_i) \wedge (\forall x_1, x_2 \in T_i : ((x_1, x_2) \in \neq_i \leftrightarrow x_1 \neq x_2)) \big) \,.$$

It is not difficult to see that if a guard g of a process $P_{i,j}$, where $i \in \{1,2,\ldots,k\}$ and $j \in \{1,2,\ldots,n_i\}$, evaluates to true in *MPS* for some $n_1, n_2, \ldots, n_k \in \mathbb{Z}_+$, then an action (b, a_g) can be executed in $[\![\mathcal{P}_{MPS}]\!]_\phi$, where ϕ is a compatible valuation in $\mathrm{va}(c_{MPS})$ such that $b \in \phi(T_i)$ and $|\phi(T_l)| = n_l$ for all $l \in \{1,2,\ldots,k\}$. It implies that the traces of $[\![\mathcal{P}_{MPS}]\!]_\phi$ can be identified with the sequences of transitions of *MPS* executable from the initial state, and the reachability properties of $[\![\mathcal{P}_{MPS}]\!]_\phi$ and *MPS* are the same.

Although multiparameterised systems can be modelled in our formalism, the properties analysable in our framework and that of Emerson & Kahlon (2000) are not precisely the same. That is because LTL-X can express liveness properties and our specifications relate to traces of arbitrary $k \in \mathbb{N}$ processes. However, in both the cases, reachability properties related to one or two arbitrary processes can be verified, because one can easily add transition schemata in our model that express certain states to be reached.

The cut-offs obtained in these two frameworks are only slightly different. In the case of conjunctive guards and finite behaviours, Emerson & Kahlon (2000) prove a cut-off

of two for the parameters n_i, $i \in \{1,2,\ldots,k\}$, such that the property involves processes generated from the template TP_i, and one for the rest. If the problem is encoded as an LTS schema and a valuation formula as suggested above, Theorem 32 gives a slightly worse cut-off; it says that two processes of each kind are needed. However, this is natural, because our theorem applies also to other kinds of systems and easily adapts to properties involving any finite number of processes.

Bouajjani *et al.* (2008) consider request-take-release (RTR) systems with an arbitrary number of identical processes competing for an access to a finite set R of shared resources. There are requests of low and high priority and a process may request or release any set of resources at once. Requests are queued and always the first one that can be fulfilled is granted, taking account of priorities of course. Because processes can be thought as LTSs communicating with the resource manager using the request, take, and release actions, an RTR system can be easily modelled as an LTS schema in the same way as the shared resource system.

The conversion of a process in our formalism is straightforward. It can be thought as an elementary LTS schema $\mathcal{P}_{pr}$ with an alphabet schema

$$\{(p,req,R'),(p,preq,R'),(p,take,R'),(p,rel,R') \mid R' \subseteq R, R' \neq \emptyset\} ,$$

where p is an atom variable capturing the identity of the process. The action schemata of the form (p,req,R') and $(p,preq,R')$ denote respectively the low- and high-priority requests of the process p for the set R' of resources. Similarly, the action schema $(p,take,R')$ means that the set R' of resources is granted to the process p, and (p,rel,R') indicates that the process p releases the resources in the set R'.

A process can always have at most one pending request. That is why the resource management policy from the viewpoint of two distinct processes p_1 and p_2 can be captured in an elementary LTS schema $\mathcal{P}_{res}$. As all the violations of the policy can be traced back to two processes served in a wrong order, by the principle of compositional modelling, the overall resource management policy is obtained as an LTS schema $\|_{(p_1,p_2)\in\neq_T:T\times T} \mathcal{P}_{res}$, where T is a type variable denoting the set of process identities and $\neq_T$ is a relation variables with the usual meaning.

An RTR system can be now represented as the LTS schema

$$\mathcal{P}_{RTR} := (\|_{p\in *_T:T} \mathcal{P}_{pr}) \,\|\, (\|_{(p_1,p_2)\in\neq_T:T\times T} \mathcal{P}_{res}) ,$$

where $*_T$ is a relation variables with the usual meaning. Furthermore, it should be clear that the set of valuations generating the system instances can be expressed as an

existential-free valuation formula c. As the conversion of a process is straightforward and the model of the resource management policy is provably correct, it is obvious that the instances of $\mathcal{P}_{RTR}$ have exactly the same reachability properties as the instances of the RTR system.

The properties considered by Bouajjani *et al.* (2008) are related to a finite number k of processes and expressed using LTL-X. By taking the LTS schema $\mathcal{P}_{RTR}$ as a model of the RTR system, we can analyse safety properties concerning any $k \in \mathbb{N}$ processes expressible as LTS schemata without hiding. Hence, the properties analysable in our framework and that of Bouajjani *et al.* (2008) are not precisely the same, but as explained earlier, reachability properties related $k \in \mathbb{N}$ arbitrary processes can be verified in both the formalisms. In the case of finite behaviours, the cut-off provided by Bouajjani *et al.* (2008) is k, which is practically the same as $\max\{2,k\}$ given by Theorem 32.

Other complete methods that enable parameterised verification by bounding the values of the parameters are more or less incomparable to ours. Emerson & Kahlon (2003a) consider systems with arbitrary many processes generated from the same template and properties related to one or two processes expressible in LTL-X. The processes execute with interleaving semantics and communicate using broadcasts and guarded transitions. Because the systems are singly parameterised, and the guards can be disjunctive, the results are incomparable to ours.

The same applies to data-independence results of Wolper (Wolper 1986), and Lazić and Nowak (Lazić 1999, Lazić & Nowak 2000). These methods treat systems that can send and receive the data values of an arbitrary large or infinite type, but the number of concurrently running components cannot be parameterised. As the receiving of data is naturally expressed using parameterised branching, also data-independence results are incomparable to ours.

The results of Emerson & Namjoshi (2003), Emerson & Kahlon (2002, 2004), and Nazari & Thistle (2007) are related to rings of processes communicating through token passing. The models of computation are (close to) LTSs, but the results are still incomparable to ours, because applying them outside rings is obviously difficult and applying our method to rings requires additional modelling work.

The same can be said about networks of homogeneous fixed-size processes communicating through token passing considered by Clarke *et al.* (2004). First, the authors show how the verification of a large network can be converted to that of finitely many reduced ones. Based on this idea, it is then proved that the verification of a family of

networks can be converted to the verification of finitely many reduced ones. However, the result provides only an upper bound for the size of the graphs of reduced networks, but no method of determining the reduced networks. Moreover, as only one process type is allowed and a family of networks does not have to be closed under the removal of a process, the method is incomparable to ours.

7 Algorithmic Precongruence Reduction

Theorem 32 characterises instances of Parameterised Traces Refinement that can be reduced to a finite verification task using the precongruence reduction. The theorem also specifies a reduced instance for each of those instances, but does not provide a mechanical procedure for determining one.

Automating the application of the theorem is not that straightforward as in the case of other cut-off results, where parameters are simple typically representing the numbers of replicated components (Bouajjani *et al.* 2008, Emerson & Kahlon 2000, 2002, 2003a, 2004, Emerson & Namjoshi 2003). Here, also the topology of a system and the relationships between components can be parameterised. To the best of our knowledge, a similar setting is only previously considered by Clarke *et al.* (2004), who study networks of homogeneous fixed-size processes communicating through token passing. Their result applies to systems of any topology, but they only provide an upper bound for the size of network graphs only and no method for determining the networks below the cut-off.

Next, we give a practical implementation-ready algorithm that inputs an instance $\mathcal{Q}$, $\mathcal{P}$, $\mathrm{va}(c)$ of Parameterised Traces Refinement satisfying our restrictions, and computes a finite subset Ψ of $\mathrm{va}(c)$ such that solving the instance $\mathcal{Q}$, $\mathcal{P}$, Ψ gives the same result as solving $\mathcal{Q}$, $\mathcal{P}$, $\mathrm{va}(c)$. To the best of our knowledge, this is the first algorithm that can handle complex parameters defining the topology of a system, and which not only determines a cut-off but also computes the needed parameter values below the cut-off.

7.1 Reduction Algorithm

Based on the proof, it is obvious that the application of Theorem 32 can be automated. Because the system and specification LTS schemata are assumed to be closed and finite, we have to consider values for finitely many type and relation variables only. It implies that there are only finitely many non-isomorphic valuations to consider, as Theorem 32 implicitly establishes an upper bound for the size of the values of type variables T, which is the maximum of 1 and the T-degrees of the LTS schemata, and restricts the values of relation variables to subsets of the values of their types. Now, such a finite set of valuations can be algorithmically generated, because, by Lemma 25, the choice

of atoms for the values of variables is insignificant as long as the atoms do not occur in the system or specification LTS schema. Moreover, while generating the valuations, we can remove isomorphs and valuations ϕ that do not satisfy the formula

$$\forall T \in \mathrm{dom}(\phi|_{\mathbb{T}}) : |\phi(T)| = \max\{1, \deg_T(\mathcal{Q}, \phi), \deg_T(\mathcal{P}, \phi)\} \,. \tag{2}$$

For example, to algorithmically apply Theorem 32 to the shared tree system, we first pick two and three distinct atoms a_1, a_2 and b_1, b_2, b_3 outside those occurring in *Mtx* and $Sts \setminus La$ for the values of respectively U and R, and then go through all valuations ϕ with the domain $\{U, R, *_U, *_R, \neq_U, <_R, \leq_R, \leqq_R\}$ such that $\phi(U) = \phi(*_U) = \{a_1\}, \{a_1, a_2\}$; $\phi(R) = \phi(*_R) = \{b_1\}, \{b_1, b_2\}, \{b_1, b_2, b_3\}$; $\phi(\neq_U)$ and $\phi(\leqq_R)$ are subsets of respectively $\phi(U) \times \phi(U)$ and $\phi(R) \times \phi(R) \times \phi(R)$; and $\phi(<_R)$ and $\phi(\leq_R)$ are subsets of $\phi(R) \times \phi(R)$. While walking through the valuations, we pick those that satisfy c_{Sts}, and remove isomorphs and valuations ϕ that do not satisfy Formula (2).

Unfortunately, only for the values of $\leqq_R$, there are $2^{1 \cdot 1 \cdot 1} + 2^{2 \cdot 2 \cdot 2} + 2^{3 \cdot 3 \cdot 3}$, *i.e.* over 100 million, possibilities to consider. It is an unacceptably high number, especially when the number of valuations needed to output is only six. On the other hand, the values of the relation variables $*_U$, $*_R$, $\neq_U$, $\leq_R$ and $\leqq_R$ can be unambiguously defined with the aid of the values of U, R and $<_R$. That is why we call relation variables $*_U$, $*_R$, $\neq_U$, $\leq_R$ and $\leqq_R$ *dependent*. Hence, by removing dependent relation variables from a valuation formula, we can greatly improve the performance of computing the valuations.

Unfortunately, determining and removing dependent relation variables may be difficult in general. However, if the value of a relation variable Π can be unambiguously defined with the aid of the values of other variables, then it should not be difficult to specify it in the form

$$\forall x_1 \in T_1 : \forall x_2 \in T_2 : \cdots : \forall x_n \in T_n : ((x_1, x_2, \ldots, x_n) \in \Pi \leftrightarrow c''') \,, \tag{3}$$

where n is a positive integer, $x_1, x_2, \ldots, x_n$ are atom variables, $T_1, T_2, \ldots, T_n$ are type variables, and c''' is a valuation formula not involving Π.

Let $\mathcal{Q}$ and $\mathcal{P}$ be closed and finite LTS schemata such that $\mathcal{Q}$ has no hiding LTS subschema, c an existential-free valuation formula with the signature $\mathrm{sig}(\mathcal{Q}) \cup \mathrm{sig}(\mathcal{P})$, and $c_1, c_2, \ldots, c_k$ the conjuncts of c, where k is a positive integer. Suppose that ϕ is a valuation in $\mathrm{va}(c)$ compatible with $\mathcal{P}$ and $\mathcal{Q}$, Π is a relation variable in $\mathrm{sig}(c)$, and $T_1, T_2, \ldots, T_n$ are type variables such that $T_1 \times T_2 \times \cdots \times T_n$ is the only type of Π, which implies that $\phi(\Pi)$ must be a subset of $\phi(T_1) \times \phi(T_2) \times \ldots \times \phi(T_n)$. Now, if there is

$j \in \{1,2,\ldots,k\}$ such that c_j is of Form (3), then $\phi(\Pi)$ can be uniquely determined with the aid of the values of other variables. Furthermore, if all the other variables in $\mathrm{sig}(c_j)$ except Π occur in $c_1, c_2, \ldots, c_{j-1}, c_{j+1}, c_{j+2}, \ldots, c_k$, then $c_0 = \bigwedge_{i=1, i\neq j}^{k} c_i$ can be used to compute valuations satisfying c. That is because whenever ϕ is a valuation in $\mathrm{va}(c)$, then $\phi|_{\mathrm{sig}(c_0)}$ is a valuation in $\mathrm{va}(c_0)$ containing all information needed to construct ϕ. This notion gives rise to Algorithms 1 and 2.

Algorithm 1. Removes conjuncts involving dependent relation variables from a valuation formula and returns the reduced valuation formula with the removed conjuncts.

procedure *RemRed*(c)
  let $c_1, c_2, \ldots, c_k$ be the conjuncts of c
  if there is $j \in \{1,2,\ldots,k\}$ such that
    $c_j = (\forall x_1 \in T_1 : \forall x_2 \in T_2 : \cdots : \forall x_n \in T_n : ((x_1, x_2, \ldots, x_n) \in \Pi \leftrightarrow c_j'))$,
    $\{T_1 \times T_2 \times \cdots \times T_n\} = \mathrm{typ}_{\mathcal{P}}(\Pi) \cup \mathrm{typ}_{\mathcal{Q}}(\Pi)$,
    $\Pi \notin \mathrm{sig}(c_j')$ and $\bigcup_{i=1, i\neq j}^{k} \mathrm{sig}(c_i) = \mathrm{sig}(c) \setminus \{\Pi\}$
  then
    let $(c_0, c'') :=$ *RemRed*$(\bigwedge_{i=1, i\neq j}^{k} c_i)$
    return $(c_0, c'' \wedge c_j)$
  else
    return $(c, \top)$
  end if

Algorithm 2. Extends a valuation satisfying the reduced valuation formula to a valuation satisfying the original valuation formula based on the removed conjuncts.

procedure *Ext*(ϕ, c')
  if $c' = c'' \wedge (\forall x_1 \in T_1 : \cdots : \forall x_n \in T_n : ((x_1, \ldots, x_n) \in \Pi \leftrightarrow c'''))$ then
    let $\phi :=$ *Ext*(ϕ, c'')
    let $\Lambda := \{(\phi'(x_1), \ldots, \phi'(x_n)) \mid \phi' \in \phi[x_1 \to T_1, \ldots, x_n \to T_n], [\![c''']\!]_{\phi'}$ is true$\}$
    let $\phi := \phi \cup \{(\Pi, \Lambda)\}$
  end if
  return ϕ

Algorithm 1 divides a valuation formula c into a pair (c_0, c'), where the former valuation formula c_0 is free from dependent relation variables specified in the above way and the latter one contains all the removed conjuncts. The algorithm applies the reduction above recursively until there is no conjunct of Form (3) left. Respectively, Algorithm 2 recursively extends a valuation $\phi \in \mathrm{va}(c_0)$ to a valuation in $\mathrm{va}(c)$ when provided with the valuation formula containing the conjuncts removed from c. The fact that the reduced valuation formula c_0 can be used to compute valuations satisfying the original valuation formula c is captured in the following proposition.

Proposition 33. *Let $\mathcal{Q}$ and $\mathcal{P}$ be LTS schemata and c a valuation formula. Then $RemRed(c)$ returns a pair (c_0, c') of valuation formulae such that*

1. *whenever $\phi \in \mathrm{va}(c)$, then $\phi|_{\mathrm{sig}(c_0)} \in \mathrm{va}(c_0)$;*
2. *whenever $\phi \in \mathrm{va}(c)$ is compatible with $\mathcal{Q}$ and $\mathcal{P}$, then $\phi = Ext(\phi|_{\mathrm{sig}(c_0)}, c')$;*
3. *whenever $\phi_1, \phi_2 \in \mathrm{va}(c_0)$ such that $\phi_1 \simeq \phi_2$, then $Ext(\phi_1, c') \simeq Ext(\phi_2, c')$.*

Proof. Let c be a valuation formula. Because the length of c decreases in every call to $RemRed(c)$, the algorithm eventually terminates and obviously returns a pair (c_0, c') of valuation formulae. Similarly, you can see that *Ext* terminates on all inputs and outputs a valuation.

To prove the first claim, we argue by induction on the number of conjuncts in c' that c_0 is a conjunction of conjuncts of c and when $\phi \in \mathrm{va}(c)$, then $\phi|_{\mathrm{sig}(c_0)} \in \mathrm{va}(c_0)$.

In the base step, c' contains only the trivial conjunct c', which implies that c' is $\top$ and hence c_0 is c. Now, c_0 is clearly a conjunction of conjuncts of c, and if $\phi \in \mathrm{va}(c)$, then $\phi|_{\mathrm{sig}(c_0)} = \phi \in \mathrm{va}(c) = \mathrm{va}(c_0)$. Hence the base step is clear.

In the induction step, c' has more than one conjunct, which implies that c' is $c'' \wedge c_1$, where c_1 is a conjunct

$$(\forall x_1 \in T_1 : \cdots : \forall x_n \in T_n : ((x_1, \ldots, x_n) \in \Pi \leftrightarrow c''')) \tag{4}$$

of c such that $T_1 \times \cdots \times T_n$ is the only type of Π in the LTS schemata $\mathcal{Q}$ and $\mathcal{P}$, every variable in $\mathrm{sig}(c) \setminus \{\Pi\}$ occurs in a conjunct of c other than c_1, and Π does not occur in c''' nor in any conjunct of c other than c'''. It means that $RemRed(c)$ calls itself with a parameter c_0', which is the conjunction of all the conjuncts of c except c_1, and returns (c_0, c''). Moreover, if $\phi \in \mathrm{va}(c)$, then obviously $\phi|_{\mathrm{sig}(c_0')} \in \mathrm{va}(c_0')$. Because c'' has strictly less conjuncts than c', by the induction hypothesis, c_0 is a conjunction of conjuncts of c_0' and $\phi|_{\mathrm{sig}(c_0')}|_{\mathrm{sig}(c_0)} \in \mathrm{va}(c_0)$, which implies that c_0 is a conjunction of

conjuncts of c and $\phi|_{\mathrm{sig}(c_0)} \in \mathrm{va}(c_0)$. As also the induction step is clear, by the induction principle, the first claim holds.

To prove the second claim, we argue by induction on the number of conjuncts in c' that whenever $\phi \in \mathrm{va}(c)$ is partially compatible with $\mathcal{Q}$ and $\mathcal{P}$, then $\phi = \mathit{Ext}(\phi|_{\mathrm{sig}(c_0)}, c')$.

In the base step, c' has only one conjunct, namely c' itself, which implies that c' is $\top$ and hence c_0 is c. Therefore, whenever $\phi \in \mathrm{va}(c)$, then $\mathit{Ext}(\phi|_{\mathrm{sig}(c_0)}, c') = \mathit{Ext}(\phi, \top) = \phi$, and the base step is clear.

In the induction step, we assume that c' is $c'' \wedge c_1$, where c_1 is as above. It means that $\mathit{RemRed}(c)$ calls itself with a parameter c_0', which is the conjunction of all the conjuncts of c except c_1, and returns (c_0, c''). If $\phi \in \mathrm{va}(c)$, then, by the proof of the first claim, we know that $\phi|_{\mathrm{sig}(c_0')}$ is a valuation in $\mathrm{va}(c_0')$ partially compatible with $\mathcal{Q}$ and $\mathcal{P}$. Now, the induction hypothesis implies that $\phi|_{\mathrm{sig}(c_0')}$ is the valuation returned by $\mathit{Ext}(\phi|_{\mathrm{sig}(c_0')}|_{\mathrm{sig}(c_0)}, c'')$. By the proof of the first claim, we also know that $\phi|_{\mathrm{sig}(c_0')}|_{\mathrm{sig}(c_0)} = \phi|_{\mathrm{sig}(c_0)}$, which implies that $\mathit{Ext}(\phi|_{\mathrm{sig}(c_0)}, c'') = \phi|_{\mathrm{sig}(c_0')}$. Moreover, because $\phi(\Pi)$ is a subset of $\phi(T_1) \times \phi(T_2) \times \ldots \times \phi(T_n)$, ϕ satisfies c_1, and Π does not occur in c''', then

$$\begin{aligned}\phi(\Pi) &= \{(\phi'(x_1), \ldots, \phi'(x_n)) \mid \phi' \in \phi[x_1 \to T_1, \ldots, x_n \to T_n], [\![c''']\!]_{\phi'} \text{ is true}\} \\ &= \{(\phi'(x_1), \ldots, \phi'(x_n)) \mid \phi' \in \phi|_{\mathrm{sig}(c_0')}[x_1 \to T_1, \ldots, x_n \to T_n], [\![c''']\!]_{\phi'} \text{ is true}\} .\end{aligned}$$

Hence, $\mathit{Ext}(\phi|_{\mathrm{sig}(c_0)}, c')$ returns $\phi|_{\mathrm{sig}(c_0')} \cup \{(\Pi, \phi(\Pi))\}$, which is nothing but ϕ and means that also the induction step holds. By the induction principle, the second claim is correct.

For the last item, we prove a stronger claim that whenever $\phi_1, \phi_2 \in \mathrm{va}(c_0)$ such that $g\phi_1 = \phi_2$ for some bijection $g : \mathbb{A} \mapsto \mathbb{A}$, then $g\mathit{Ext}(\phi_1, c') = \mathit{Ext}(\phi_2, c')$. As above, we argue by induction on the number of conjuncts in c'.

The base step follows directly from the second claim, which says that ϕ_i is returned by $\mathit{Ext}(\phi_i, c')$ for both $i \in \{1, 2\}$.

In the induction step, we assume that c' is $c'' \wedge c_1$, where c_1 is as above. It means that $\mathit{RemRed}(c)$ calls itself with a parameter c_0', which is the conjunction of all the conjuncts of c except c_1, and returns (c_0, c''). By the induction hypothesis, $g\mathit{Ext}(\phi_1, c'') = \mathit{Ext}(\phi_2, c'')$. Now, if $i \in \{1, 2\}$ and ψ_i is a valuation returned by $\mathit{Ext}(\phi_i, c'')$, then $\mathit{Ext}(\phi_i, c')$ returns a valuation $\psi_i \cup \{(\Pi, \Lambda_i)\}$, where

$$\Lambda_i = \{(\phi_i'(x_1), \ldots, \phi_i'(x_n)) \mid \phi_i' \in \phi_i[x_1 \to T_1, \ldots, x_n \to T_n], [\![c''']\!]_{\phi_i'} \text{ is true}\} .$$

By Lemma 25 and Items 5 and 6 of Lemma 21,

$$\begin{aligned}
&\{g^*(\phi_1'(x_1),\ldots,\phi_1'(x_n)) \mid \phi_1' \in \phi_1[x_1 \to T_1,\ldots,x_n \to T_n], [\![c''']\!]_{\phi_1'} \text{ is true}\}\\
&= \{(g(\phi_1'(x_1)),\ldots,g(\phi_1'(x_n))) \mid \phi_1' \in \phi_1[x_1 \to T_1,\ldots,x_n \to T_n], [\![c''']\!]_{g\phi_1'} \text{ is true}\}\\
&= \{(\phi_2'(x_1),\ldots,\phi_2'(x_n)) \mid \phi_2' \in \phi_2[x_1 \to T_1,\ldots,x_n \to T_n], [\![c''']\!]_{\phi_2'} \text{ is true}\}\,,
\end{aligned}$$

which implies that $gExt(\phi_1,c') = Ext(\phi_2,c')$, which means that also the induction step is clear. By the induction principle, also the last claim is correct. □

The first claim of the proposition states that if ϕ is a valuation satisfying a valuation formula c and we reduce c to a valuation formula c_0 using Algorithm 1, then restricting ϕ to $\mathrm{sig}(c_0)$ gives a valuation satisfying the reduced formula. According to the second claim, if we now extend $\phi|_{\mathrm{sig}(c_0)}$ using Algorithm 2, we end up with ϕ. Hence, if one wants to generate valuations in $\mathrm{va}(c)$ it is sufficient to create corresponding smaller valuations in $\mathrm{va}(c_0)$ and then extend them to valuations in $\mathrm{va}(c)$. This procedure is practical if there is some extra work, like generating excess valuations and performing certain checks, to be done to find the correct valuations. According to the last claim, especially isomorphs can be removed before extending valuations.

Now, if we apply the algorithm *RemRed* to c_{Sts} all the relation variables except $\leqq_R$, $*_R$, $*_U$ are removed as redundant. Moreover, if the values of $*_U$ and $*_R$ are encoded in the form $\forall x \in T : (x \in *_T \leftrightarrow \top)$, where $T \in \{U,R\}$, and there is a conjunct $\forall u \in U : \top$ whose only purpose is to introduce an additional occurrence of U in the valuation formula, also $*_R$ and $*_U$ are removed. It means that we have to go through only $2 \cdot (2^{1\cdot1} + 2^{2\cdot2} + 2^{3\cdot3}) = 1060$ valuations to find six ones we need. On the other hand, a lot of useless valuations are still generated. Those are valuations ϕ, which do not satisfy the reduced valuation formula c_0 (nor the original one), have an isomorphic counterpart already generated, or for which there is a type variable T such that the size of $\phi(T)$ is not equal to the maximum of $1, \deg_T(\mathcal{Q},\phi), \deg_T(\mathcal{P},\phi)$.

A traditional approach to solving classification problems like this is based on the recursive generation of a search tree (Kaski & Östergård 2006). An obvious idea is to start from the smallest valuation and gradually enlarge the values of variables as we go deeper in the tree. For each combination of the values of type variables below the cut-off (modulo the choice of atoms), we generate a separate search tree. Each time, the root of the search tree is a valuation that maps the relation variables to the empty set, and the children of any valuation ϕ are those that are obtained from ϕ by adding an action to the value of a relation variable.

Clearly, a search tree generated like this contains all the desired valuations, but the same valuation may be generated multiple times. To avoid it, we assume that the sets of the relation variables and the visible actions (the tuples of atoms) are totally ordered. Now, when computing the children of a valuation ϕ, we allow only the value of the greatest relation variable Ξ such that $\phi(\Xi) \neq \emptyset$ to be appended using the actions greater than those already in $\phi(\Xi)$, plus the (empty) values of relation variables greater Ξ. This way we still get all the valuations but none of them is generated twice.

In order to generate only valuations that satisfy the valuation formula c_0, we should be able to skip over valuations that do not satisfy it and to prune the search when a valuation cannot be made larger without violating c_0. Unfortunately, we do not know when a valuation can be skipped, but for certain valuation formulae we know when to prune the search. These are the valuation formulae that result in sets of valuations downward-closed with respect to the subvaluation order. (To bring back the definition of a subvaluation, see Page 85.) It turns out that a valuation formula has this property if it is both existential-free and membership-negated. As in the case of Lemma 31, we prove the result for negation-normal valuation formulae first.

Lemma 34. *Let c be a negation-normal existential-free membership-negated valuation formula, and ϕ, ψ valuations compatible with c such that $\psi \sqsubseteq \phi$. If ϕ satisfies c, then ψ satisfies c, too.*

Proof. To prove the claim, we show a slightly stronger claim which implies the lemma. We prove by induction on the number of membership-negated valuation subformulae of c that ψ satisfies c, if ϕ satisfies c, and if c is $\top$ or $y_1 = y_2$, then $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true.

In the base step, c is $\top$, $y_1 = y_2$ or $\neg(x_1, x_2, \ldots, x_n) \in \Pi$, where $y_1, y_2, x_1, x_2, \ldots, x_n$ are atom variables and Π a relation variable. If c is $\top$, then obviously $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true. If c is $y_1 = y_2$ and $[\![c]\!]_\phi$ is true, then $\phi(y_1) = \phi(y_2)$ if and only if $\psi(y_1) = \psi(y_2)$, because $\phi(y) = \psi(y)$ for all atom variables y in the domain. Hence, $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true. Finally, if c is $\neg(x_1, x_2, \ldots, x_n) \in \Pi$, then $(\phi(x_1), \phi(x_2), \ldots, \phi(x_n)) \notin \phi(\Pi)$ clearly implies $(\psi(x_1), \psi(x_2), \ldots, \psi(x_n)) \notin \psi(\Pi)$, because $\psi \sqsubseteq \phi$. Hence, if $[\![c]\!]_\phi$ is true, then $[\![c]\!]_\psi$ is true, too.

In the induction step, c is either a negated, disjunctive, conjunctive or universal valuation formula with more than one membership-negated valuation subformula. If c is a negated valuation formula $\neg c'$, then c' must be $\top$ or $y_1 = y_2$, where y_1, y_2 are atom

variables. By the induction hypothesis, $[\![c']\!]_\phi$ is true if and only if $[\![c']\!]_\psi$ is true, which implies that $[\![c]\!]_\phi$ is true if and only if $[\![c]\!]_\psi$ is true.

Let us then assume that c is $c_1 \odot c_2$, where $\odot$ is either $\vee$ or $\wedge$. By the induction hypothesis, if $i \in \{1,2\}$ and $[\![c_i]\!]_\phi$ is true, then $[\![c_i]\!]_\psi$ is true, too. Therefore, if $[\![c]\!]_\phi$ is true, then $[\![c]\!]_\psi$ is true, too.

Finally, if c is an existential-free universal valuation formula $\forall x \in T : c'$ and $[\![c]\!]_\phi$ is true, then $[\![c']\!]_{\phi'}$ is true for all valuations $\phi' \in \phi[x \mapsto T]$. If ψ' is a valuation in $\psi[x \mapsto T]$, then there is a valuation $\phi' \in \phi[x \mapsto T]$ such that $\phi'(x) = \psi'(x)$. By Lemma 26, ϕ', ψ' are compatible with c', and clearly $\psi' \sqsubseteq \phi'$. As $[\![c']\!]_{\phi'}$ is true, by the induction hypothesis, $[\![c']\!]_{\psi'}$ is true, too. Hence, if $[\![c]\!]_\phi$ is true, then $[\![c]\!]_\psi$ is true as well.

By the induction principle, we know that the lemma holds. □

To understand why the lemma holds for negation-normal existential-free membership-negated valuation formulae c, you may suppose that ϕ is a valuation satisfying c and consider what happens if you remove an atom a from the value of a type variable such that $a \neq \phi(x)$ for all atom variables $x \in \mathrm{dom}(\phi)$, or an action from the value of a relation variable. First, note that if the universal quantification over some set gives a positive result, then the universal quantification over its subset must give a positive result, too. Because type variables are used in universal quantification only, you cannot make the instance of an existential-free valuation formula false by removing a from the value of a type variable. On the other hand, membership-negated valuation formulae can only state the absence of some actions in the values of relation variables. Hence, if you remove an action from the value of a relation variable, the formula must still be satisfied.

Lemma 35. *Let c be an existential-free membership-negated valuation formula, and ϕ, ψ compatible valuations such that $\psi \sqsubseteq \phi$. If ϕ satisfies c, then ψ satisfies c, too.*

Proof. Using De Morgan's rules, we convert c into an existential-free membership-negated negation-normal valuation formula c' with the domain $\mathrm{sig}(c)$ such that every valuation θ is compatible with c and satisfies c, if and only if θ is compatible with c' and satisfies c'. By Lemma 34, we know that whenever $[\![c']\!]_\phi$ is true, then $[\![c']\!]_\psi$ is true as well, which implies that if ϕ satisfies c, then also ψ satisfies c. □

The lemma implies that when one is searching for valuations satisfying an existential-free membership-negated valuation formula, you can prune the search as soon as a

valuation that does not satisfy the valuation formula is encountered. The idea can also be extended to a pruning heuristic for an arbitrary valuation formula c_0.

As earlier, we may assume that c_0 is of the form $\bigwedge_{i=1}^{n} c_i$, where n is a positive integer and $c_1, c_2, \ldots, c_n$ are the conjuncts of c_0. We pick all $i \in \{1, 2, \ldots, n\}$ such that c_i is existential-free and membership-negated to form a set I. Now, $c' := \bigwedge_{i \in I} c_i$ is an existential-free and membership-negated valuation formula satisfied by every valuation in $\mathrm{va}(c_0)$. Hence, if a valuation does not satisfy c' then the search can be pruned because neither c_0 can be satisfied by making the valuation bigger.

Combining the recursive generation of a search tree with the removal of dependent relation variables and the pruning heuristics results in Algorithm 3 which inputs an instance $\mathcal{Q}$, $\mathcal{P}$, $\mathrm{va}(c)$ of Parameterised Traces Refinement accordant with the restrictions set above and reduces the set of $\mathrm{va}(c)$ of valuations to its finite subset Ψ without changing the answer to the verification task.

Algorithm 3. Reduces a set of valuations, which is expressed as an existential-free valuation formula and related to an instance of Parameterised Traces Refinement, to a finite one without changing the answer to the verification task.

procedure *Reduce*$(\mathcal{Q}, \mathcal{P}, c)$
  for all $T \in \mathrm{sig}(c) \cap \mathbb{T}$ let $c := (\forall x \in T : \top) \wedge c$
  let $(c_0, c') := \mathit{RemRed}(c)$
  let c_0' be the conjunction of all the membership-negated conjuncts of c_0
  let $\Psi := \emptyset$
  for all $T \in \mathrm{sig}(c_0) \cap \mathbb{T}$ let $l_T := \max\{1, \deg_T(\mathcal{Q}), \deg_T(\mathcal{P})\}$
  for all $\upsilon \in \prod_{T \in \mathrm{sig}(c_0) \cap \mathbb{T}} \{1, 2, \ldots, l_T\}$ do
    let $\hat{\phi}$ be a valuation with the domain c_0 such that
      $\hat{\phi}(T)$ is a set of $\upsilon(T)$ atoms disjoint from $\mathrm{at}(\mathcal{Q}) \cup \mathrm{at}(\mathcal{P})$ for all $T \in \mathrm{sig}(c_0) \cap \mathbb{T}$, and
      $\hat{\phi}(\Pi)$ is the empty set for all $\Pi \in \mathrm{sig}(c_0) \cap \mathbb{G}$
    let $\Phi := \mathit{Search}(\hat{\phi}, \emptyset)$
    remove valuations ϕ from Φ such that ϕ does not satisfy c or for some $T \in \mathrm{dom}(\phi|_{\mathbb{T}})$
      $|\phi(T)| \neq \max\{1, \deg_T(\mathcal{Q}, \mathit{Ext}(\phi, c')), \deg_T(\mathcal{P}, \mathit{Ext}(\phi, c'))\}$
    remove isomorphs from Φ
    let $\Psi := \Psi \cup \{\mathit{Ext}(\phi, c') \mid \phi \in \Phi\}$
  end do
  return Ψ

Algorithm 3. Continued.

subprocedure *Search*(ϕ, Φ)
  if ϕ satisfies c_0' then for all $\phi' \in$ *Children*(ϕ) let $\Phi :=$ *Search*$(\phi', \Phi \cup \{\phi\})$
  return Φ

subprocedure *Children*(ϕ)
  let $\Phi := \emptyset$ and let Ξ be the smallest relation variable
  if $\phi(\Pi) \neq \emptyset$ for some relation variable $\Pi \in \mathrm{dom}(\phi)$ then
    let Ξ be the greatest relation variable in $\mathrm{dom}(\phi)$ such that $\phi(\Xi) \neq \emptyset$
  for all relation variables $\Pi \in \mathrm{dom}(\phi)$ not smaller than Ξ do
    for all $\alpha \in \bigcap_{T_1 \times T_2 \times \ldots \times T_n \in \mathrm{typ}_{\mathcal{Q}}(\Pi) \cup \mathrm{typ}_{\mathcal{P}}(\Pi)} \phi(T_1) \times \phi(T_2) \times \ldots \times \phi(T_n)$ such that
      α is greater than any element in $\phi(\Pi)$ do
        let ϕ' be a valuation equal to ϕ except that $\phi'(\Pi) = \phi(\Pi) \cup \{\alpha\}$
        let $\Phi := \Phi \cup \{\phi'\}$
    end do
  end do
  return Φ

Theorem 36. *Let $\mathcal{Q}$ and $\mathcal{P}$ be closed and finite LTS schemata such that $\mathcal{Q}$ has no hiding LTS subschema, and c an existential-free valuation formula with the domain* $\mathrm{sig}(\mathcal{Q}) \cup \mathrm{sig}(\mathcal{P})$. *Then the algorithm Reduce$(\mathcal{Q}, \mathcal{P}, c)$ returns a finite subset Ψ of* $\mathrm{va}(c)$ *such that the instances $\mathcal{Q}$, $\mathcal{P}$, Ψ and $\mathcal{Q}$, $\mathcal{P}$,* $\mathrm{va}(c)$ *of Parameterised Traces Refinement have the same answer.*

Proof. Let us use the notation of the algorithm. First, note that the algorithm terminates because all the for-loops are finite and the value of

$$\sum_{\Pi \in \mathrm{sig}(c_0) \cap \mathbb{G}} (\,|\bigcap_{T_1 \times T_2 \times \ldots \times T_n \in \mathrm{typ}_{\mathcal{Q}}(\Pi) \cup \mathrm{typ}_{\mathcal{P}}(\Pi)} \phi(T_1) \times \phi(T_2) \times \ldots \times \phi(T_n)| - |\phi(\Pi)|)$$

decreases in recursive calls to *Search*. Hence, the algorithm *Reduce* terminates with the input $\mathcal{Q}$, $\mathcal{P}$, c and hence returns a set Ψ of valuations, which is necessarily finite.

Next, note that the first line of the algorithm does not change the set $\mathrm{va}(c)$. Then, by Lemma 33, it is obvious that Ψ is a subset of $\mathrm{va}(c)$. Hence, if the answer to $\mathcal{Q}$, $\mathcal{P}$, $\mathrm{va}(c)$ is positive, then also the answer to $\mathcal{Q}$, $\mathcal{P}$, Ψ is. To prove the opposite, we show

that Ψ contains a maximal set of non-isomorphic valuations in the set

$$\{\phi \in \mathrm{va}(c) \mid \phi \text{ is compatible with } \mathcal{Q} \text{ and } \mathcal{P}, \text{ and}$$
$$\forall T \in \mathrm{dom}(\phi|_{\mathbb{T}}) : |\phi(T)| = \max\{1, \deg_T(\mathcal{Q},\phi), \deg_T(\mathcal{P},\phi)\}\}\,, \quad (5)$$

which by Theorem 32 implies that the instances $\mathcal{Q}$, $\mathcal{P}$, Ψ and $\mathcal{Q}$, $\mathcal{P}$, $\mathrm{va}(c)$ of Parameterised Traces Refinement have the same answer.

Let ϕ be a valuation in Set (5). It means that $\phi \in \mathrm{va}(c)$, which by Proposition 33 implies that $\phi|_{\mathrm{sig}(c_0)}$, denoted by ϕ', is a valuation in $\mathrm{va}(c_0)$ partially compatible with $\mathcal{Q}$ and $\mathcal{P}$. Moreover, as c_0' is a conjunction of conjuncts of c_0, then ϕ' satisfies also c_0'. Next, we show that there is a partially compatible valuation isomorphic to ϕ' in a set Φ returned by *Search*.

Whenever ϕ_1 is a valuation with the domain $\mathrm{dom}(c_0)$ partially compatible with $\mathcal{Q}$ and $\mathcal{P}$ such that $\phi_1(T)$ is a set of size at most l_T for all type variables T in the domain, we write $\hat{\phi}_1$ for a (root) valuation generated by the algorithm such that $\hat{\phi}_1(T)$ and $\phi_1(T)$ are sets of the same size for every type variable T and $\hat{\phi}_1(\Pi)$ is the empty set for every relation variable Π in the domain. We prove that whenever ϕ_1 is such a valuation satisfying c_0' and g is a bijection: $\mathbb{A} \mapsto \mathbb{A}$ such that g preserves the atoms in $\mathrm{at}(\mathcal{Q}) \cup \mathrm{at}(\mathcal{P})$ and $\{g(a) \mid a \in \phi_1(T)\} = \hat{\phi}_1(T)$ for all type variables T in the domain, then $g\phi_1$ is a valuation in $\mathit{Search}(\hat{\phi}_1, \emptyset)$ partially compatible with $\mathcal{Q}$ and $\mathcal{P}$. We argue by induction on $\sum_{\Pi \in \mathrm{sig}(c_0) \cap \mathbb{G}} |\phi_1(\Pi)|$.

In the base step, ϕ_1 maps type variables T in the domain to sets of size at most l_T and relation variables Π to the empty set. If g is a bijection: $\mathbb{A} \mapsto \mathbb{A}$ such that g preserves the atoms in $\mathrm{at}(\mathcal{Q}) \cup \mathrm{at}(\mathcal{P})$ and $\{g(a) \mid a \in \phi_1(T)\} = \hat{\phi}_1(T)$ for all type variables T in the domain, then $g\phi_1 = \hat{\phi}_1$, which means that ϕ_1 and $\hat{\phi}_1$ are isomorphic. Because ϕ_1 satisfies c_0', then by Lemma 25, $\hat{\phi}_1$ satisfies c_0', too. Therefore, $\hat{\phi}_1$ is a partially compatible valuation in $\mathit{Search}(\hat{\phi}_1, \emptyset)$, and the base step is clear.

In the induction step, there is a relation variable $\Pi \in \mathrm{sig}(c_0)$ such that $\phi_1(\Pi)$ is not empty. We may assume that Π is the greatest such a variable in the total order. Let g be a bijection: $\mathbb{A} \mapsto \mathbb{A}$ such that g preserves the atoms in $\mathrm{at}(\mathcal{Q}) \cup \mathrm{at}(\mathcal{P})$ and $\{g(a) \mid a \in \phi_1(T)\} = \hat{\phi}_1(T)$ for all type variables T in the domain, α an action in $\phi_1(\Pi)$ such that $g^*(\alpha)$ is maximal in the total order, and ϕ_1' the maximal subvaluation of ϕ_1 such that $\phi_1'(\Pi) = \phi_1(\Pi) \setminus \{\alpha\}$. Because ϕ_1' is a subvaluation of ϕ_1 and ϕ_1 satisfies the existential-free membership-negated valuation formula c_0', by Lemma 35, also ϕ_1' satisfies c_0'. Moreover, clearly $\{g(a) \mid a \in \phi_1'(T)\} = \hat{\phi}_1'(T)$ for all type variables T in the domain, which by the induction hypothesis means that $g\phi_1'$ is a partially compati-

ble valuation in $\mathit{Search}(\hat{\phi}'_1, \emptyset)$. As ϕ'_1 satisfies c'_0, by Lemma 25, also $g\phi'_1$ satisfies it. Therefore, also the children of $g\phi'_1$ are computed. By the choice of α, we know that $g\phi_1 \in \mathit{Children}(g\phi'_1)$. Moreover, because ϕ_1 satisfies c'_0, by Lemma 25 again, also $g\phi_1$ satisfies it, which implies that $g\phi_1 \in \mathit{Search}(\hat{\phi}'_1, \emptyset)$. Furthermore, as ϕ_1 is partially compatible with $\mathcal{Q}$ and $\mathcal{P}$, g preserves the atoms in $\mathrm{at}(\mathcal{Q}) \cup \mathrm{at}(\mathcal{P})$, and $\hat{\phi}'_1$ is clearly the same valuation as $\hat{\phi}_1$, by Lemma 26, $g\phi_1$ is a partially compatible valuation in $\mathit{Search}(\hat{\phi}_1, \emptyset)$. Therefore, also the induction step is clear.

Recall that we assumed ϕ to be a valuation in Set (5), and ϕ' to be the restriction of ϕ to $\mathrm{sig}(c_0)$ which is a partially compatible valuation in $\mathrm{va}(c_0)$. By the induction principle, we know that whenever g is a bijection: $\mathbb{A} \mapsto \mathbb{A}$ such that g preserves the atoms in $\mathrm{at}(\mathcal{Q}) \cup \mathrm{at}(\mathcal{P})$ and $\{g(a) \mid a \in \phi'(T)\} = \hat{\phi}'(T)$ for all type variables T in the domain, then $g\phi'$ is a valuation in $\mathit{Search}(\hat{\phi}', \emptyset)$ partially compatible with $\mathcal{Q}$ and $\mathcal{P}$. Because such a bijection g exists, also the valuation $g\phi'$ exists. As $\phi' \in \mathrm{va}(c_0)$, by Lemma 25, also $g\phi' \in \mathrm{va}(c_0)$. Moreover, as ϕ' and $g\phi'$ are clearly isomorphic, by Proposition 33, also $\psi := \mathit{Ext}(\phi', c')$ and $\mathit{Ext}(g\phi', c')$ are isomorphic, too. On the other hand, by the same proposition $\mathit{Ext}(g\phi', c') = \phi$, which by Lemma 25 and the fact that $\phi \in \mathrm{va}(c)$ implies that $\psi \in \mathrm{va}(c)$ as well. Furthermore, as ϕ fulfils Equation (2), also ψ satisfies it. Hence, there is a compatible valuation isomorphic to ϕ in the set Ψ returned by the algorithm.

In other words, for every valuation in Set (5), there is an isomorphic valuation in Ψ that is compatible with $\mathcal{Q}$ and $\mathcal{P}$, which implies that the algorithm returns a set that contains a maximal set of non-isomorphic valuations in Set (5). Hence, the theorem is correct. □

The purpose of the first line of the algorithm is just to introduce an additional occurrence of every type variable in the valuation formula. It guarantees that the removal of a conjunct of Form (3) is not blocked due to the fact that some type variable occurs in that conjunct only. Also note that valuations ϕ that do not satisfy Equation (2) can only be removed after the search tree is generated and the valuations are extended. That is because even though a valuation does not satisfy the equation, some greater valuations in the sense of $\Subseteq$ may satisfy it. Hence, the unsatisfaction of Equation (2) cannot be used as a pruning criterion. Moreover, before the equation can be evaluated one needs to know the values of all the relation variables. Hence, valuations ϕ that do not satisfy Equation (2) can be detected only after they are extended.

Also the isomorphs are removed only after a search tree is generated. Although it would be tempting to prune the search as soon as a valuation isomorphic to a previously generated one is detected, you cannot do so, because the subtrees of isomorphic valuations are generally not isomorphic. Therefore, the pruning techniques reviewed by Kaski & Östergård (2006) cannot be applied.

When applied to the shared resource system, the algorithm generates a total of six search trees, one for each combination of the values of U and R below the cut-off modulo the choice of atoms. The search trees for valuations that map R to a singleton are simple, both of them consist of two vertices of which the root satisfies c_{Sts} and Equation (2). The search tree for the valuations mapping R and U to sets of size two is shown in Figure 21, where valuations satisfying c_{Sts} are highlighted. However, the root does not satisfy Equation (2), and the other two highlighted valuations are isomorphic, which means that there is only one more valuation to check. The search tree for valuations mapping R and U to sets of size respectively two and one is obviously similar to the search tree in Figure 21 despite the number of users. The search trees for valuations mapping R to a set of size three consist of 77 vertices, of which 16 satisfy c_{Sts}. However, only six isomorphic ones fulfil Equation (2). Hence, the algorithm generates a total of 174 vertices, performs isomorphism test for 18 of them, and returns six valuations, which we find acceptable.

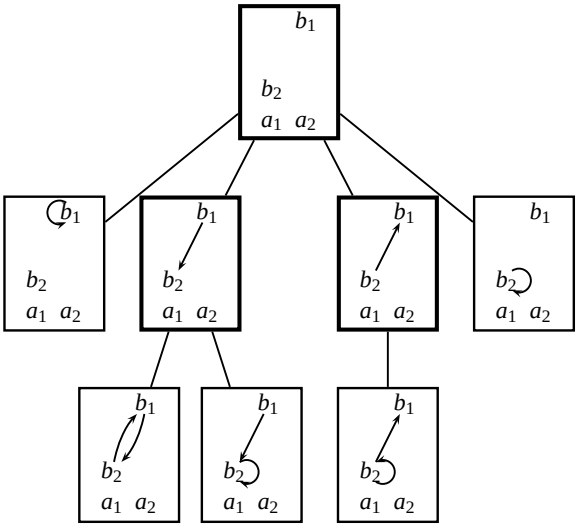

Fig 21. The search tree for valuations with two users and two resources.

Despite this rather positive result, the worst-case running time of the algorithm is obviously exponential in the number of relation variables and in the number of type variables with a degree greater than one, too. Actually, the running time is doubly exponential in the number of type variables with a degree greater than one occurring in the same type of a relation variable.

7.2 Testing Valuation Isomorphism

The algorithm is quite straightforward to implement except for one thing, namely the isomorphism test between valuations. In the context of (coloured) graphs, it is often done by first converting both the graphs into a canonical form which is one of the isomorphic graphs and which is the same to precisely all the isomorphic ones, and then checking whether the canonical forms of the graphs are the same.

While generating non-isomorphic graphs, canonical forms encountered so far are typically stored in an ordered list. When a new graph is generated, it is sufficient to search the list for the corresponding canonical graph, and insert the canonical form in the list if it is not already there.

You can use the same idea in the generation of non-isomorphic valuations as well. However, it is not that obvious how the canonical form of a valuation can and should be computed. On the other hand, computing the canonical form of a coloured graph is a well-studied problem for which there are efficient tools as well (McKay 2007). That is why we wish to convert valuations into coloured graphs and perform isomorphism reduction on coloured graphs instead of valuations.

Definition 37 (Coloured graph)**.** A *coloured graph* is a four-tuple (V, C, γ, E), where

- V is a set of *vertices*,
- C is a finite set of *colours*,
- $\gamma : V \mapsto C$ is a *colouring function*, and
- $E \subseteq V \times V$ is a set of *edges*.

Coloured graphs are denoted by a capital Calligraphic letter $\mathcal{G}$ and its variants.

Let ϕ be a valuation with a finite domain that contains no atom variable. We convert a valuation ϕ into a coloured graph as follows. First, the atoms and the tuples of atoms occurring in the image of ϕ are taken as vertices. Then, we make sure that also the first half and last half of every vertex of length at least two are included in the vertex set. Simultaneously, for each such a vertex, we add an edge from its first part and an edge

to its last part. After that, the vertices are coloured by a set that consists of the length of the vertex and those variables in $\mathrm{dom}(\phi)$ the value of which includes the vertex. Finally, we add one more colour, a set of all the relation variables mapped to the empty set, which obviously is not used to colour any vertex.

Formally, the *graph of* ϕ, denoted by $\mathcal{G}(\phi)$, is a four-tuple (V, C, γ, E), where

1. V is the smallest set such that
   a) $\bigcup_{\Lambda \in \mathrm{im}(\phi)} \Lambda \subseteq V$, and
   b) if $v = (a_1, a_2, \ldots, a_n) \in V$ such that $n \geq 2$, then
   $$(a_1, a_2, \ldots, a_{\lceil n/2 \rceil}), (a_{\lceil n/2 \rceil + 1}, a_{\lceil n/2 \rceil + 2}, \ldots, a_n) \in V \, ;$$
2. C is the smallest set such that
   a) $\{\Pi \in \mathrm{dom}(\phi) \mid \phi(\Pi) = \emptyset\} \in C$, and
   b) $\{n\} \cup \{\sharp \in \mathrm{dom}(\phi) \mid (a_1, \ldots, a_n) \in \phi(\sharp)\} \in C$ for all $(a_1, \ldots, a_n) \in V$;
3. $\gamma : V \mapsto C$ is a function such that
   $$\gamma((a_1, \ldots, a_n)) = \{n\} \cup \{\sharp \in \mathrm{dom}(\phi) \mid (a_1, \ldots, a_n) \in \phi(\sharp)\}$$
   for all $(a_1, \ldots, a_n) \in V$;
4. E is the smallest set such that if $v = (a_1, a_2, \ldots, a_n)$ is in V and $n \geq 2$, then
   $$((a_1, a_2, \ldots, a_{\lceil n/2 \rceil}), v), (v, (a_{\lceil n/2 \rceil + 1}, a_{\lceil n/2 \rceil + 2}, \ldots, a_n)) \in E \, .$$

It is easy to see that the graph of a valuation is a coloured graph from which the valuation can be reconstructed as follows.

Lemma 38. *Let ϕ be a valuation with a finite domain that contains no atom variable. Then the graph of ϕ is a coloured graph (V, C, γ, E) such that*

– $\mathrm{dom}(\phi)$ *is the set of all the relation and type variables occurring in C, and*
– $\phi(\sharp) = \{(a_1, a_2, \ldots, a_n) \in V \mid \sharp \in \gamma((a_1, a_2, \ldots, a_n))\}$ *for all* $\sharp \in \mathrm{dom}(\phi)$.

The claim follows straightforwardly from the definition of the graph of a valuation.

As an example, consider a valuation $\phi \in \mathrm{va}(c_{Sts})$ such that $\phi(U) = \phi(*_U) = \{a_1, a_2\}$, $\phi(R) = \phi(*_R) = \{b_1, b_2\}$, $\phi(<_R) = \{(b_1, b_2)\}$, $\phi(\neq_U)$ is the set of all the different pairs of the elements in $\phi(U)$, $\phi(\leq_R)$ is the reflexive closure of $\phi(<_R)$, and $\phi(\leqq_R)$ is the set of all triplets (b'_1, b'_2, b'_3) such that $(b'_1, b'_3), (b'_2, b'_3) \in \phi(\leq_R)$. The valuation corresponds to Valuation (j) in Figure 20 and its graph is shown in Figure 22.

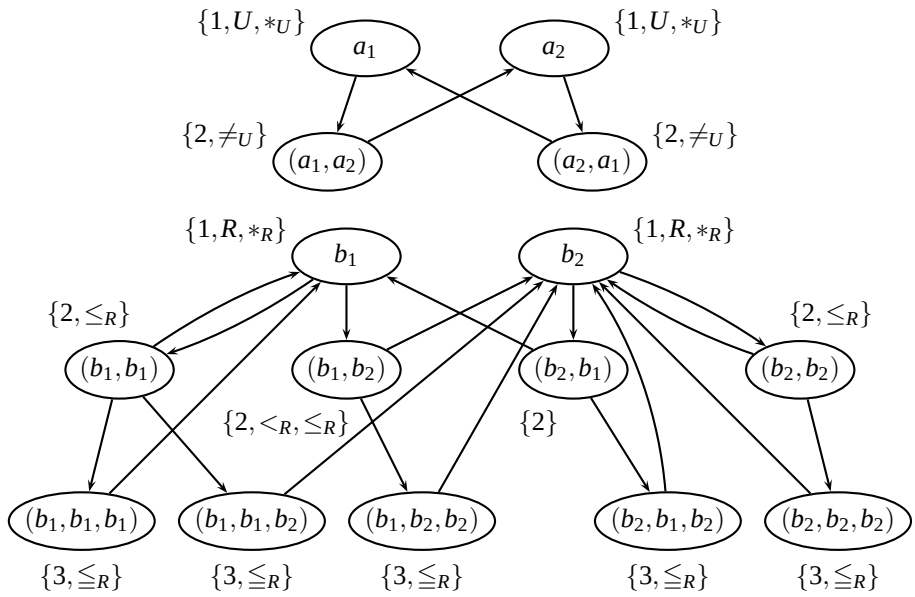

Fig 22. The graph of a valuation corresponding to an instance of the shared tree system with two users and two resources.

Now, we have provided a conversion from valuations to coloured graphs. However, in order to be able to exploit graph isomorphism algorithms for valuation isomorphism testing, we still have to show that the valuation isomorphism problem is equivalent to testing the isomorphism of their graphs.

Let $\mathcal{G}_i = (V_i, C_i, \gamma_i, E_i)$ be a coloured graph for both $i \in \{1,2\}$, and $f : V_1 \mapsto V_2$ a bijection. We write $f\mathcal{G}_1 = \mathcal{G}_2$, if

1. $V_2 = \{f(v) \mid v \in V_1\}$,
2. $C_1 = C_2$,
3. $\gamma_2(f(v)) = \gamma_1(v)$ for all $v \in V_1$, and
4. $E_2 = \{(f(v_1), f(v_2)) \mid (v_1, v_2) \in E_1\}$.

If $f\mathcal{G}_1 = \mathcal{G}_2$ for some bijection $f : V_1 \mapsto V_2$, we say that $\mathcal{G}_1$ and $\mathcal{G}_2$ are *isomorphic* or that $\mathcal{G}_1$ *is isomorphic to* $\mathcal{G}_2$, denoted by $\mathcal{G}_1 \cong \mathcal{G}_2$.

Let ϕ and ψ be valuations with a finite domain that contains no atom variable. It is easy to see that if g is a bijection: $\mathbb{A} \mapsto \mathbb{A}$ such that $\psi = g\phi$, then the restriction of g^* (recall that g^* is the extension of g to tuples) to the vertices of $\mathcal{G}(\phi)$ is a bijec-

tion f such that $\mathcal{G}(\psi) = f\mathcal{G}(\phi)$. Moreover, if f is a bijection such that $\mathcal{G}(\psi) = f\mathcal{G}(\phi)$ and g is a bijection: $\mathbb{A} \mapsto \mathbb{A}$ such that $g(a) = f(a)$ for all atoms $a \in \mathrm{dom}(f)$, then $(g(a_1), g(a_2), \ldots, g(a_n)) = f((a_1, a_2, \ldots, a_n))$ for all vertices $(a_1, a_2, \ldots, a_n)$ of $\mathcal{G}(\phi)$. This can be easily shown by induction on n, as for every vertex v of the length $i > 1$ there is precisely one incoming edge from a vertex v_1 of the length $i-1$ and exactly one outgoing edge to a vertex v_2 of the length $i-1$ such that and v_1 and v_2 form respectively the first and last part of v. After that, using Lemma 38, it is easy to see that $\psi = g\phi$. As such a bijection g clearly exists, it proves that valuations are isomorphic if and only if their graphs are.

Proposition 39. *Let ϕ and ψ be valuations with a finite domain that contains no atom variable. The valuations ϕ and ψ are isomorphic, if and only if $\mathcal{G}(\phi)$ and $\mathcal{G}(\psi)$ are isomorphic.*

Proof. Let $\mathcal{G}_1 = (V_1, C_1, \gamma_1, E_1)$ and $\mathcal{G}_2 = (V_2, C_2, \gamma_2, E_2)$ be the coloured graphs of respectively valuations ϕ and ψ.

Let us first suppose that ϕ_1 and ϕ_2 are isomorphic, which means that their domains are the same and that there is a bijection $g : \mathbb{A} \mapsto \mathbb{A}$ such that $\phi_2(T) = \{g(a) \mid a \in \phi_1(T)\}$ for all type variables T in the domain, and $\phi_2(\Pi) = \{g^*(\alpha) \mid \alpha \in \phi_1(\Pi)\}$ for all relation variables Π in the domain. To see that then the graphs of ϕ and ψ are isomorphic as well, one first shows that if $v \in V_1$ then $g^*(v) \in V_2$ by induction on the number of times the steps 1.a) and 1.b) in the definition of the graph of a valuation are applied to see that $v \in V_1$. Similarly, one shows that if $v \in V_2$ then there is $v' \in V_1$ such that $v = g^*(v')$. Hence, $g^*|_{V_1}$ is a bijection: $V_1 \mapsto V_2$ such that $V_2 = \{g^*|_{V_1}(v) \mid v \in V_1\}$. It is also easy to see that $C_1 = C_2$, $\gamma_2(g^*|_{V_1}(v)) = \gamma_1(v)$ for all $v \in V_1$, and

$$E_2 = \{(g^*|_{V_1}(v_1), g^*|_{V_1}(v_2)) \mid (v_1, v_2) \in E_1\}\,.$$

Hence, the graphs of ϕ and ψ are isomorphic.

Let us then suppose that $\mathcal{G}_1$ and $\mathcal{G}_1$ are isomorphic, which means that there is a bijection $f : V_1 \mapsto V_2$ such that $V_2 = \{v \in V_1 \mid f(v)\}$, $\gamma_2(f(v)) = \gamma_1(v)$ for all $v \in V_1$, $C_1 = C_2$, and $E_2 = \{(f(v_1), f(v_2)) \mid (v_1, v_2) \in E_1\}$. Note that the vertices of $\mathcal{G}_1$ and $\mathcal{G}_2$ are visible actions, *i.e.* tuples of atoms, and because $\gamma_2(f(v)) = \gamma_1(v)$ for all $v \in V_1$, the function f maps each vertex in V_1 to a tuple of the same size. Hence, there is a bijection $g : \mathbb{A} \mapsto \mathbb{A}$ such that $g(a) = f(a)$ for all atoms $a \in \mathrm{dom}(f)$.

To prove that ϕ_1 and ϕ_2 are isomorphic, we first show that $f(v) = g^*(v)$ for all $v \in V_1$ by induction on the length of v. As the base step is clear by definition, we consider the

induction step where v is $(a_1, a_2, \ldots, a_n)$ such that $n \geq 2$. By above, it is known that $f(v)$ is an n-tuple $(b_1, b_2, \ldots, b_n)$ of atoms. By the definition of the graph of ϕ,

$$((a_1, a_2, \ldots, a_{\lceil n/2 \rceil}), v) \text{ and } (v, (a_{\lceil n/2 \rceil + 1}, a_{\lceil n/2 \rceil + 2}, \ldots, a_n))$$

are edges of $\mathcal{G}_1$. By the induction hypothesis and the isomorphism of coloured graphs, it means that then

$$((g(a_1), \ldots, g(a_{\lceil n/2 \rceil})), (b_1, \ldots, b_n)) \text{ and } ((b_1, \ldots, b_n), (g(a_{\lceil n/2 \rceil + 1}), \ldots, g(a_n)))$$

are edges of $\mathcal{G}_2$. However, it is possible only if $b_i = g(a_i)$ for all $i \in \{1, 2, \ldots, n\}$, or in other words, $f(v) = g^*(v)$. By the induction principle, $f(v) = g^*(v)$ for all $v \in V_1$.

Now, we are ready to show that ϕ_1 and ϕ_2 are isomorphic. As the graphs of the valuations have the same set of colours, ϕ_1 and ϕ_2 must have the same domain. If $\sharp$ is a relation or type variable in the domain of the valuations, then by Lemma 38 and the isomorphism of coloured graphs,

$$\begin{aligned} \{g^*(\alpha) \mid \alpha \in \phi_1(\sharp)\} &= \{g^*(v) \mid v \in V_1, \sharp \in \gamma_1(v)\} \\ &= \{g^*(v) \mid v \in V_1, \sharp \in \gamma_2(g^*(v))\} = \{v \in V_2 \mid \sharp \in \gamma_2(v)\} = \phi_2(\sharp)\,. \end{aligned}$$

As the domains of ϕ_1 and ϕ_2 contain no atom variable, it proves that ϕ_1 and ϕ_2 are isomorphic, and the claim holds. □

The result means that existing tools can be used for the generation of canonical valuations and isomorphism testing. Probably the best and also the best-known tool for the purpose is nauty (McKay 2007). It uses a slightly different definition of a coloured graph, where vertices are numbered from one to $n \in \mathbb{Z}_+$ and the set of colours is an ordered partition of the vertex set. This is not a problem though as the coloured graphs in our sense can always be converted into such a format. One just has to define a total order in the set of all the possible colours (which can be done in several ways) to create an ordered partition of the vertex set. Additionally, one has to store the original set of colours with information on unused ones to distinct between coloured graphs that are equivalent modulo renaming of colours.

7.3 The Verification of taDOM+ Tree-Locking Protocols

We have applied the results to verify taDOM2+ and taDOM3+ (extended Document Object Model level 2 and 3, respectively) tree-locking protocols (Haustein & Härder 2008)

used in XML (Extensible Markup Language) databases (Siirtola 2010). The protocols closely resemble the shared tree system but are more complex, and users access trees via *transactions*, which can be thought as the sequences of read and write actions on vertices. However, as the essential issue is correct isolation between transactions, it is sufficient to consider the protocols from the viewpoint of transitions and vertices.

The new lock modes are the main reason for added complexity. They enable locking a single vertex or a vertex together with its children. It means that the protocols look different from the viewpoint of a vertex and its parent than from the viewpoint of the vertex and its other proper ancestor. Therefore, the edges of a forest have to be divided in two classes, parent and ancestor edges. *Parent edges* are used to instantiate the elementary LTS schema representing the behaviour of a protocol from the viewpoint of a vertex and its parent, and respectively *ancestor edges* are used to instantiate the elementary LTS schema representing the behaviour of a protocol from the viewpoint of a vertex and its other proper ancestor. However, in order to express the structure of the protocol models without existential quantification, we had to allow a vertex to have incoming ancestor edges without a single incoming parent edge, which corresponds to a situation where the parent of a vertex is abstracted away but other ancestors are preserved. Hence, if (n_1, n_2) is a parent edge, then n_1 is necessarily the parent of n_2 in the forest as well, but the opposite is not generally true.

Another difference is that for practical purposes, locking proceeds in taDOM+ protocols in the opposite direction: the target vertex is locked first and the root last. Also this issue complicates the model, because in order to make sure that a vertex cannot be accessed before the whole path of vertices up to the root is successfully locked, we had to introduce action schemata denoting the beginning and ending of locking shared by all the protocol views involving locking action schemata. Moreover, as in the case of taDOM+ protocols one can obtain an access to a vertex also by checking and finding out that some of its ancestors is appropriately locked, we had to introduce action schemata that enable the mode of a vertex lock to be read, too.

The property we considered is known as *repeatable-read* (Ramakrishnan & Gehrke 2002) in the context of databases. It says that reading a vertex within a transaction should always give the same result unless the transaction itself has updated the vertex. In other words, if a transaction accesses a vertex, no other transaction should be able to write to the vertex before the transaction ends, and if a transaction starts writing to a vertex, no other transaction should be able to access the vertex before the transaction

ends. The same property is earlier proved by Siirtola & Valenta (2008) but using a theory tailored for the protocols.

As the structure of taDOM+ protocol models is similar to that of the shared tree system, also the cut-off obtained by Theorem 32 is alike. To prove the tree-locking protocols correct for any number of transactions and any size and shape of a (tree) database, it is sufficient to consider instances with at most two transactions and three vertices. However, because there are two kinds of edges instead of one, there are more instances to check. These instances are created by valuations obtained from Valuations (a), (c), (g), (h), (j), (n) in Figure 20 by dividing the edges into the parent and ancestor ones. As the edge from b_1 to b_3 present in Valuations (g) and (n) can only be an ancestor edge, there were a total of 14 instances to check per protocol.

The instances were encoded in CSP modelling language and verification was carried out using the finite-state refinement checker FDR2. All the instances were found to be correct, which implies that taDOM2+ and taDOM3+ tree-locking protocols have the repeatable-read property irrespective of the number of transactions and the size and shape of a database. The largest refinement check involved examining nearly 350 million states and 3 billion transitions and took approximately eight gigabytes of memory and three and a half hours to complete on a relatively fast computation server.

8 Undecidable Extensions

Finally, we consider several natural extensions to the theory. We will show that allowing the use of valuation formulae that are not existential-free, specifications involving hiding, richer semantics, and stronger LTS schemata all make the problem undecidable. In each of the cases, the lack of algorithmic solutions is shown by reducing a famous undecidable problem, the halting of a deterministic Turing machine with the empty input, to a parameterised verification task expressible in the extended formalism. Therefore, the result is very close to the decidability frontier and hence cannot be significantly extended in any direction without simultaneously restricting some other aspect.

8.1 Deterministic Turing Machines

Informally, a deterministic Turing machine, a DTM for short, is a finite-state machine operating on an infinite tape of data (Papadimitriou 1994). The left end of the tape is fixed and it is marked by a special symbol $\triangleright$ that cannot be erased. The symbol is followed by a finite input string and the rest of the cells are initialised to the *blank symbol* denoted by $\sqcup$.

Initially, the read-write head of a DTM is scanning the first element in the tape. Based on the current state and the contents of the cell at the read-write head, the machine deterministically computes the next state, the symbol to be written in the position of the read-write head in the tape, and the direction where the read-write head moves. The machine operates like this until it halts, *i.e.* reaches one of the special states, *halt*, *yes* or *no* called respectively *the halting state*, *the accepting state*, and *the rejecting state*. After a DTM is halted, the output of the computation can be read from the tape by omitting the first $\triangleright$ symbol and the infinite sequence of trailing blank symbols.

Definition 40 (DTM)**.** *A deterministic Turing machine* is a four-tuple $(Q, D, \delta, \hat{q})$, where

- Q is a finite set of *states* such that $\mathit{halt}, \mathit{yes}, \mathit{no} \notin Q$,
- D is a finite set of *tape symbols* such that $\sqcup, \triangleright \in D$,
- $\delta : Q \times D \mapsto (Q \cup \{\mathit{halt}, \mathit{yes}, \mathit{no}\}) \times D \times \{\leftarrow, \rightarrow, -\}$ is a *transition function* such that whenever $\delta(q, \triangleright) = (q', d', m)$ then $d' = \triangleright$ and $m = \rightarrow$, and
- $\hat{q} \in Q$ is *the initial state*.

DTMs are denoted by $\mathcal{M}$ and its variants.

Let $\mathcal{M} = (Q, D, \delta, \hat{s})$ be a DTM. The set D of tape symbols is called the *alphabet (of $\mathcal{M}$)* and denoted by $\text{alph}(\mathcal{M})$, like the alphabet of an LTS. A *configuration (of $\mathcal{M}$)* is a triplet (q, w_1, w_2), where q is a state of $\mathcal{M}$, $w_1 \in \triangleright\text{alph}(\mathcal{M})^*$, and $w_2 \in \text{alph}(\mathcal{M})^*$. It is said that a configuration (q', w_1', w_2') *follows* a configuration $(q, w_1 d_1, w_2)$, where $d_1 \in \text{alph}(\mathcal{M})$, if

- $\delta(q, d_1) = (q', d_1', -)$, $w_1' = w_1 d_1'$ and $w_2' = w_2$,
- $\delta(q, d_1) = (q', d_1', \leftarrow)$, $w_1' = w_1$ and $w_2' = d_1' w_2$,
- $\delta(q, d_1) = (q', d_1', \rightarrow)$, $w_1' = w_1 d_1' d_2$ and $w_2 = d_2 w_2'$ for some $d_2 \in \text{alph}(\mathcal{M})$, or
- $\delta(q, d_1) = (q', d_1', \rightarrow)$, $w_1' = w_1 d_1' \sqcup$ and $w_2 = w_2' = \varepsilon$.

A finite sequence $(q_1, w_{1,1}, w_{1,2}), (q_2, w_{2,1}, w_{2,2}), \ldots, (q_k, w_{k,1}, w_{k,2})$ of configurations of $\mathcal{M}$ such that $(q_{i+1}, w_{i+1,1}, w_{i+1,2})$ follows $(q_i, w_{i,1}, w_{i,2})$ for each $i \in \{1, 2, \ldots, k-1\}$ is called a *run (of $\mathcal{M}$) (from $(q_1, w_{1,1}, w_{1,2})$) (to $(q_k, w_{k,1}, w_{k,2})$)*. The *scope* of a run $(q_1, w_{1,1}, w_{1,2}), (q_2, w_{2,1}, w_{2,2}), \ldots, (q_k, w_{k,1}, w_{k,2})$ is $\max\{|w_{i,1} w_{i,2}| \mid i \in \{1, 2, \ldots, k\}\}$, the maximum number of cells effectively used in the run. An *input (of $\mathcal{M}$)* is a finite sequence $w \in (\text{alph}(\mathcal{M}) \setminus \{\sqcup\})^*$ of non-blank tape symbols. It is said that a DTM $\mathcal{M}$ *halts with the input w*, if there is a run of $\mathcal{M}$ from $(\hat{q}, \triangleright, w)$ to a configuration (q, w_1, w_2) such that q is the halting, accepting or rejecting state.

Problem 41 (Halting with Empty Input)**.**
Instance: A DTM $\mathcal{M}$.
Question: Does $\mathcal{M}$ halt with the input ε?

Theorem 42. *Halting with Empty Input is undecidable (Papadimitriou 1994).*

8.2 Undecidability with Arbitrary Valuation Formulae

First, we consider verification tasks expressible with the aid of arbitrary valuation formulae. The fact that such problems cannot be algorithmically solved is shown by constructing a system with a ring topology that simulates a DTM with a circular tape of an arbitrary length. The system is first modelled from the viewpoint of a cell and its neighbours and then all such instances are composed in parallel in the form of a ring.

The behaviour of a DTM from the viewpoint of a single cell is captured in elementary LTS schema *Cell*. An instance of *Cell* stores the symbol in the corresponding cell and communicates with its neighbours solely by passing a token that carries information on the state of the DTM. Furthermore, each instance can update its contents and make progress only when it possesses a token.

The instances of *Cell* synchronously initialise themselves such that after the operation each of them contains either the blank symbol, or the tape head symbol plus a token. Each step of the DTM is simulated by a single action and whenever the read-write head moves, the token with DTM state information is passed on to the neighbouring LTS schema. If an accepting, rejecting or halting state is reached, the instance with the token may execute a specific halting action and deadlock. To make sure that the read-write head does not cross the right end of the string, each instance is allowed to receive the token initially from the left-hand side neighbour and later on from the neighbour to which the token was last passed. Hence, if the token tries to go round the ring, the system deadlocks.

To specify *Cell* formally, we assume that for each state $q \in Q \cup \{halt, yes, no\}$ there is a unique atom, which for the sake of simplicity is denoted by the same letter as the state itself. We also assume a unique atom *init* denoting the initialisation of the instances of *Cell*, and an element $0 \notin Q \cup \{halt, yes, no\}$ representing the lack of a token. The state space of *Cell* consists of all tuples (q, d, m), where $q \in Q \cup \{halt, yes, no, 0\}$ stores the state of the DTM or denotes the lack of the token, $d \in D$ records the contents of the corresponding cell, and m is one of the symbols $\leftarrow, \rightarrow, -$ denoting respectively the token being on the left, on the right, or owned by the cell. The initial state of *Cell* is $(0, \sqcup, -)$.

We refer to the corresponding cell, and its left- and right-hand side neighbour by respectively atom variables c, c_l and c_r. We use the action schema

- *init* to denote initialisation,
- $(c, halt)$ to represent halting,
- $(c, c_l, q), (c, c_r, q)$, where $q \in Q \cup \{yes, no, halt\}$, to denote passing a token with information on the state q to respectively the left-hand and right-hand side neighbour, and
- $(c_l, c, q), (c_r, c, q)$, where $q \in Q \cup \{yes, no, halt\}$, to represent the reception of a token with information on the state q from respectively the left-hand and right-hand side neighbour.

The alphabet schema of *Cell* consists of all the action schemata above, and the transition schemata of *Cell* are

- $((0, \sqcup, -), init, (0, \sqcup, \leftarrow)), ((0, \sqcup, -), init, (\hat{q}, \triangleright, -))$,
- $((q, d, -), \tau, (q', d', -))$, whenever $\sigma(q, d) = (q', d', -)$,
- $((q, d, -), (c, c_l, q'), (0, d', \leftarrow))$, whenever $\sigma(q, d) = (q', d', \leftarrow)$,

– $((q,d,-),(c,c_r,q'),(0,d',\rightarrow))$, whenever $\sigma(q,d)=(q',d',\rightarrow)$,
– $((0,d,\leftarrow),(c_l,c,q),(q,d,-))$, for every $q \in Q \cup \{halt,yes,no\}$ and $d \in D$,
– $((0,d,\rightarrow),(c_r,c,q),(q,d,-))$, for every $q \in Q \cup \{halt,yes,no\}$ and $d \in D$,
– $((q,d,-),(c,halt),(q,d,\rightarrow))$, for all $q \in \{halt,yes,no\}$ and $d \in D$.

Our DTM model can be now expressed as an LTS schema $\|_{(c_l,c,c_r)\in\Pi:T\times T\times T}\, Cell$, denoted by $DTM_{\mathcal{M}}$, where T is a type variable denoting the set of cell identifiers and Π is a relation variable representing all the triplets of the identifiers of successive cells. Hence, we are interested in valuations ϕ with the domain $\{T,\Pi\}$ such that

– $\phi(T)=\{a_0,a_1,\ldots,a_{n-1}\}$ for some positive integer n,
– $\phi(\Pi)=\{(a_{(i-1) \bmod n},a_i,a_{(i+1) \bmod n}) \mid i \in \{0,1,\ldots,n-1\}\}$, when $n \geq 2$, and
– $\phi(\Pi)=\emptyset$, when $n=1$.

The set of all such valuations is denoted by Φ_{DTM}.

Whenever ϕ is a valuation in Φ_{DTM} compatible with $DTM_{\mathcal{M}}$ and $n=|\phi(T)|$ is greater than one, then a state of $[\![DTM_{\mathcal{M}}]\!]_\phi$ can be thought as an n-tuple the i^{th} component of which denotes a state of $[\![Cell]\!]_{\phi_i}$, where $i \in \{0,1,\ldots,n-1\}$ and ϕ_i is a valuation in $\phi[(c_l,c,c_r) \rightarrow \Pi]$ such that $\phi_i(c)=a_i$. It is intuitively obvious that $[\![DTM_{\mathcal{M}}]\!]_\phi$ simulates the DTM $\mathcal{M}$ with a tape of length up to n. However, it is a tedious task to prove it formally.

Lemma 43. *Let ρ be a run of a DTM $\mathcal{M}$ from the configuration $(\hat{q},\triangleright,\sqcup^{n-1})$ to a configuration $(q,d_0d_1\ldots d_k,d_{k+1}d_{k+2}\ldots d_{n-1})$, where $\hat{q}$ is the initial state of $\mathcal{M}$ and $n \geq 2$ is an integer greater than or equal to the scope of ρ. Then there is a valuation $\phi \in \Phi_{DTM}$ compatible with $DTM_{\mathcal{M}}$ such that*

$$((0,d_0,\rightarrow),\ldots,(0,d_{k-1},\rightarrow),(q,d_k,-),(0,d_{k+1},\leftarrow),\ldots,(0,d_{n-1},\leftarrow))$$

is a reachable state of $[\![DTM_{\mathcal{M}}]\!]_\phi$.

Proof. We show a slightly strengthened version of the lemma. We claim that if ρ is a run of a DTM $\mathcal{M}$ from the configuration $(\hat{q},\triangleright,\sqcup^{n-1})$ to a configuration

$$(q',d'_0d'_1\ldots d'_{k'},d'_{k'+1}d'_{k'+2}\ldots d'_{n-1})\,, \tag{6}$$

where $\hat{q}$ is the initial state of $\mathcal{M}$, and $n \geq 2$ is an integer greater than or equal to the scope of ρ, then $d'_0=\triangleright$ and there is a valuation $\phi \in \Phi_{DTM}$ compatible with $DTM_{\mathcal{M}}$ such that

$$s'=((0,d'_0,\rightarrow),\ldots,(0,d'_{k'-1},\rightarrow),(q',d'_{k'},-),(0,d'_{k'+1},\leftarrow),\ldots,(0,d'_{n-1},\leftarrow))$$

is a reachable state of $[\![DTM_{\mathcal{M}}]\!]_\phi$. We argue by induction on the length of the run ρ.

In the base step the length of ρ is one, which implies that

$$s' = ((\hat{q}, \triangleright, -), (0, \sqcup, \leftarrow), (0, \sqcup, \leftarrow), \ldots, (0, \sqcup, \leftarrow)) \,.$$

Clearly, s' is a state of $[\![DTM_{\mathcal{M}}]\!]_\phi$ reachable by the path $\hat{s}\ init\ s'$, where $\hat{s}$ denotes the initial state of $[\![DTM_{\mathcal{M}}]\!]_\phi$, and ϕ is a valuation in Φ_{DTM} compatible with $DTM_{\mathcal{M}}$ such that ϕ maps T to a set of size n.

In the induction step, ρ is $\rho'(q', d_0' d_1' \ldots d_{k'}', d_{k'+1}' d_{k'+2}' \ldots d_{n-1}')$, where the length of ρ' is at least one. If the last element in ρ' is

$$(q, d_0 d_1 \ldots d_k, d_{k+1} d_{k+2} \ldots d_{n-1}) \,, \tag{7}$$

then Configuration (6) follows Configuration (7), and by the induction hypothesis, $d_0 = \triangleright$ and there is a valuation ϕ in Φ_{DTM} such that

$$s = ((0, d_0, \rightarrow), \ldots, (0, d_{k-1}, \rightarrow), (q, d_k, -), (0, d_{k+1}, \leftarrow), \ldots, (0, d_{n-1}, \leftarrow))$$

is a reachable state of $[\![DTM_{\mathcal{M}}]\!]_\phi$. To complete the proof, we show that there is an action $\alpha \in \text{alph}([\![DTM_{\mathcal{M}}]\!]_\phi) \cup \{\tau\}$ such that (s, α, s') is a transition of $[\![DTM_{\mathcal{M}}]\!]_\phi$ and $d_0' = \triangleright$.

There are three possibilities how Configuration (6) can follow Configuration (7). First, we assume that $\delta(q, d_k) = (q', d_k', -)$, which implies that $k' = k$ and $d_i' = d_i$ for all $i \in \{0, 1, \ldots, n-1\}$ different from k. As $d_0 = \triangleright$, k cannot be zero, which in turn means that $d_0' = d_0 = \triangleright$. By definition, $[\![Cell]\!]_{\phi_k}$ has a transition $((q, d_k, -), \tau, (q', d_k', -))$, which implies that

$$(s, \tau, ((0, d_0, \rightarrow), \ldots, (0, d_{k-1}, \rightarrow), (q', d_k', -), (0, d_{k+1}, \leftarrow), \ldots, (0, d_{n-1}, \leftarrow)))$$

is a transition of $[\![DTM_{\mathcal{M}}]\!]_\phi$. However, this is the same transition as (s, τ, s'), which means that s' is a reachable state of $[\![DTM_{\mathcal{M}}]\!]_\phi$, and the induction hypothesis holds.

Next, if $\delta(q, d_k) = (q', d_k', \leftarrow)$, then $k' = k - 1$ and $d_i' = d_i$ for all $i \in \{0, 1, \ldots, n-1\}$ different from k. Because $d_0 = \triangleright$, k cannot be zero, which in turn means that $d_0' = d_0 = \triangleright$. Moreover, by definition, $[\![Cell]\!]_{\phi_k}$ and $[\![Cell]\!]_{\phi_{k-1}}$ have respectively transitions $((q, d_k, -), \alpha, (0, d_k', \leftarrow))$ and $((0, d_{k-1}, \rightarrow), \alpha, (q', d_{k-1}, -))$, where $\alpha = (a_k, a_{k-1}, q')$. Because α is not in the alphabet of $[\![Cell]\!]_{\phi_j}$ whenever j is an element in $\{0, 1, \ldots, n-1\}$ different from $k-1$ and k, (s, α, s'') is a transition $[\![DTM_{\mathcal{M}}]\!]_\phi$, where s'' is the state

$$((0, d_0, \rightarrow), \ldots, (0, d_{k-2}, \rightarrow), (q', d_{k-1}, -), (0, d_k', \leftarrow), (0, d_{k+1}, \leftarrow), \ldots, (0, d_{n-1}, \leftarrow)) \,.$$

However, s'' is the same state as s', so s' is a reachable state of $[\![DTM_{\mathcal{M}}]\!]_\phi$, and the induction hypothesis holds.

Finally, it is assumed that $\delta(q,d_k) = (q',d'_k,\rightarrow)$, which implies that $k' = k+1$ and $d'_i = d_i$ for all $i \in \{0,1,\ldots,n-1\}$ different from k. Now, k cannot be $n-1$, because otherwise the scope of ρ would be greater than n. Note also that by the definition of a DTM, $d'_0 = d_0 = \triangleright$, even if $k = 0$. Furthermore, the LTSs $[\![Cell]\!]_{\phi_k}$ and $[\![Cell]\!]_{\phi_{k+1}}$ have respectively transitions $((q,d_k,-),\alpha,(0,d'_k,\rightarrow))$ and $((0,d_{k+1},\leftarrow),\alpha,(q',d_{k+1},-))$, where α is the action (a_k,a_{k+1},q'). Because α is not in the alphabet of $[\![Cell]\!]_{\phi_j}$ whenever j is an element in $\{0,1,\ldots,n-1\}$ different from k and $k+1$, (s,α,s'') is a transition $[\![DTM_{\mathcal{M}}]\!]_\phi$, where s'' is now the state

$$((0,d_0,\rightarrow),\ldots,(0,d_{k-1},\rightarrow),(0,d'_k,\rightarrow),(q',d_{k+1},-),(0,d_{k+2},\leftarrow),\ldots,(0,d_{n-1},\leftarrow)) .$$

Again, s'' is the same state as s', which implies that s' is a reachable state of $[\![DTM_{\mathcal{M}}]\!]_\phi$, and the induction hypothesis holds.

Hence, by the induction principle, the lemma is correct. □

However, it is not sufficient just to be able to simulate $\mathcal{M}$ using $DTM_{\mathcal{M}}$, we also have to show that $[\![DTM_{\mathcal{M}}]\!]_\phi$, where $\phi \in \Phi_{DTM}$, executes halting actions, *i.e.* actions of the form $(a,halt)$ where a is an atom, only if $\mathcal{M}$ halts. For that purpose, we show that the executions of $[\![DTM_{\mathcal{M}}]\!]_\phi$ correspond to the runs of $\mathcal{M}$, too. Proving correspondence in this direction is somewhat more difficult than the other, because the initialisation is non-deterministic; any number of instances of *Cell* can be initially given a token. Fortunately, $[\![DTM_{\mathcal{M}}]\!]_\phi$ exhibits a great deal of symmetry, which means that we can always assume that if some instance initially gets a token, the first instance $[\![Cell]\!]_{\phi_0}$ is among them.

Lemma 44. *Let $\mathcal{M}$ be a DTM and $\phi \in \Phi_{DTM}$ a valuation compatible with $DTM_{\mathcal{M}}$ such that $n \geq 2$ is an integer equal to $|\phi(T)|$. If a state $(s'_0,s'_1,\ldots,s'_{n-1})$ is reachable from a state $(s_0,s_1,\ldots,s_{n-1})$ in $[\![DTM_{\mathcal{M}}]\!]_\phi$, then the state $(s'_1,s'_2,\ldots,s'_{n-1},s'_0)$ is reachable from the state $(s_1,s_2,\ldots,s_{n-1},s_0)$ in $[\![DTM_{\mathcal{M}}]\!]_\phi$.*

Proof. To prove the claim, we show that if $((s_0,s_1,\ldots,s_{n-1}),\alpha,(s'_0,s'_1,\ldots,s'_{n-1}))$ is a transition of $[\![DTM_{\mathcal{M}}]\!]_\phi$, then there is an action α' in $\mathrm{alph}([\![DTM_{\mathcal{M}}]\!]_\phi) \cup \{\tau\}$ such that $((s_1,s_2,\ldots,s_{n-1},s_0),\alpha',(s'_1,s'_2,\ldots,s'_{n-1},s'_0))$ is a transition of $[\![DTM_{\mathcal{M}}]\!]_\phi$, which implies the lemma.

First note that the state space of $[\![Cell]\!]_{\phi_i}$ is the same for all $i \in \{0,1,\ldots,n-1\}$. Hence, whenever $(s_0,s_1,\ldots,s_{n-1})$ is a state of $[\![DTM_{\mathcal{M}}]\!]_\phi$, then $(s_1,s_2,\ldots,s_{n-1},s_0)$ is

a state of $[\![DTM_{\mathcal{M}}]\!]_\phi$, too. Next, if $((s_0,s_1,\ldots,s_{n-1}),\alpha,(s'_0,s'_1,\ldots,s'_{n-1}))$ is a transition of $[\![DTM_{\mathcal{M}}]\!]_\phi$, then α is either $\tau, (a_i, halt), (a_i, a_{(i-1) \bmod n}, q)$ or $(a_i, a_{(i+1) \bmod n}, q)$, where $i \in \{0,1,\ldots,n-1\}$ and $q \in Q \cup \{yes, no, halt\}$. Now, if we take α' to be respectively either $\tau, init, (a_{(i-1) \bmod n}, halt), (a_{(i-1) \bmod n}, a_{(i-2) \bmod n}, q)$ or $(a_{(i-1) \bmod n}, a_i, q)$, it is straightforward to check that then $((s_1,s_2,\ldots,s_{n-1},s_0),\alpha',(s'_1,s'_2,\ldots,s'_{n-1},s'_0))$ is a transition of $[\![DTM_{\mathcal{M}}]\!]_\phi$, which completes the proof. □

Now, to prove the opposite simulation result, it is sufficient to show that an execution of $[\![DTM_{\mathcal{M}}]\!]_\phi$, where $\phi \in \Phi_{DTM}$ and the first cell initially gets the token, corresponds to a run of $\mathcal{M}$.

Lemma 45. *Let $\mathcal{M}$ be a DTM, $\phi \in \Phi_{DTM}$ a valuation compatible with $DTM_{\mathcal{M}}$ such that $n \geq 2$ is an integer equal to $|\phi(T)|$, and*

$$((q_0,d_0,m_0),(q_1,d_1,m_1),\ldots,(q_{n-1},d_{n-1},m_{n-1}))$$

a state reachable from state

$$((\hat{q},\triangleright,-),(0,\sqcup,\leftarrow),\ldots,(0,\sqcup,\leftarrow),(\hat{q},\hat{d}_l,\hat{m}_l),\ldots,(\hat{q},\hat{d}_{n-1},\hat{m}_{n-1}))$$

in $[\![DTM_{\mathcal{M}}]\!]_\phi$, where $l \leq n$ is an integer such that $\hat{d}_{l \bmod n} = \triangleright$ and $\hat{m}_{l \bmod n} \neq \leftarrow$. Then there is a unique integer $k \in \{0,1,\ldots,l-1\}$ such that $q_k \neq 0$, and there is a run from the configuration $(\hat{q},\triangleright,\sqcup^{l-1})$ to the configuration

$$(q_k, d_0 d_1 \ldots d_k, d_{k+1} d_{k+2} \ldots d_{l-1}) .$$

The purpose of l in the lemma is to encode the number of cells that can communicate with the first one using token passing. Intuitively, it is achieved if we define l to be the next cell (after the first one) initialised to the tape head symbol. However, if the first cell is the only one initialised to the tape head symbol, then the definition does not lead to the desired result. That is why we pick l to be the smallest positive integer, which is necessarily at most n, such that the cell at the place l mod n is initialised to the tape head symbol.

Proof. We show a strengthened version of the lemma. We claim that if $\mathcal{M}$ is a DTM, $\phi \in \Phi_{DTM}$ a valuation compatible with $DTM_{\mathcal{M}}$ such that $n \geq 2$ is an integer equal to $|\phi(T)|$, and

$$s = ((q'_0,d'_0,m'_0),(q'_1,d'_1,m'_1),\ldots,(q'_{n-1},d'_{n-1},m'_{n-1}))$$

a state reachable by a path π from a state

$$s_0 = ((\hat{q}, \triangleright, -), (0, \sqcup, \leftarrow), \ldots, (0, \sqcup, \leftarrow), (\hat{q}, \hat{d_l}, \hat{m}_l), \ldots, (\hat{q}, \hat{d}_{n-1}, \hat{m}_{n-1}))$$

in $[\![DTM_{\mathcal{M}}]\!]_\phi$, where $l \leq n$ is an integer such that $\hat{d}_{l \bmod n} = \triangleright$ and $\hat{m}_{l \bmod n} \neq \leftarrow$, then $d'_0 = d'_{l \bmod n} = \triangleright$, $m'_0, m'_{l \bmod n} \neq \leftarrow$, there is a unique integer $k' \in \{0, 1, \ldots, l-1\}$ such that $q'_{k'} \neq 0$, and there is a run from the configuration $(\hat{q}, \triangleright, \sqcup^{l-1})$ to the configuration

$$(q'_{k'}, d'_0 d'_1 \ldots d'_{k'}, d'_{k'+1} d'_{k'+2} \ldots d'_{l-1}) .$$

We argue by induction on the length of the path π.

In the base step, π is s_0, and clearly the claim holds. As the induction step, π is $\pi' \alpha s'$ for some path π' and action $\alpha \in \text{alph}([\![DTM_{\mathcal{M}}]\!]_\phi) \cup \{\tau\}$. If

$$s = ((q_0, d_0, m_0), (q_1, d_1, m_1), \ldots, (q_{n-1}, d_{n-1}, m_{n-1}))$$

is the last state in π', then (s, α, s') is a transition of $[\![DTM_{\mathcal{M}}]\!]_\phi$, and s is reachable from s_0 by path π' the length of which is strictly smaller than the length of π. By the induction hypothesis, it means that $d_0 = \triangleright$, $m_0 \neq \leftarrow$, $d_{l \bmod n} = \triangleright$, $m_{l \bmod n} \neq \leftarrow$, there is a unique integer $k \in \{0, 1, \ldots, l-1\}$ such that $q_k \neq 0$, and there is a run from the configuration $(\hat{q}, \triangleright, \sqcup^{l-1})$ to the configuration

$$(q_k, d_0 d_1 \ldots d_k, d_{k+1} d_{k+2} \ldots d_{l-1}) . \tag{8}$$

To complete the proof, we have to show that $d'_0 = \triangleright$, $m'_0 \neq \leftarrow$, $d'_{l \bmod n} = \triangleright$, $m'_{l \bmod n} \neq \leftarrow$, there is a unique integer $k' \in \{0, 1, \ldots, l-1\}$ such that $q'_{k'} \neq 0$, and

$$(q'_{k'}, d'_0 d'_1 \ldots d'_{k'}, d'_{k'+1} d'_{k'+2} \ldots d'_{l-1}) \tag{9}$$

is a configuration that follows Configuration (8) or that Configurations (8) and (9) are the same. Note that α cannot be *init*, so there are four cases to consider.

First, let us suppose that α is $(a_i, halt)$ for some $i \in \{0, 1, \ldots, n-1\}$. Because $\alpha \in \text{alph}([\![Cell]\!]_{\phi_i})$ and $\alpha \notin \text{alph}([\![Cell]\!]_{\phi_j})$ for all $j \in \{0, 1, \ldots, n-1\}$ different from i, then $((q_i, d_i, m_i), (a_i, halt), (q'_i, d'_i, m'_i))$ must be a transition of $[\![Cell]\!]_{\phi_i}$, and $q'_j = q_j$, $d'_j = d_j$, $m'_j = m_j$ for all $j \in \{0, 1, \ldots, n-1\}$ different from i. However, by the definition of *Cell*, then also q'_i and d'_i must be equal to respectively q_i and d_i, and $m'_i = \rightarrow$. Therefore, $d'_0 = d'_{l \bmod n} = \triangleright$, $m'_0, m'_{l \bmod n} \neq \leftarrow$ and if we choose k' to be k, then k' is the unique integer in $\{0, 1, \ldots, l-1\}$ such that $q'_{k'} \neq 0$ and Configurations (8) and (9) are the same. Hence, the induction hypothesis holds.

If α is τ, then there is $i \in \{0,1,\ldots,n-1\}$ such that $((q_i,d_i,m_i),\tau,(q_i',d_i',m_i'))$ is a transition of $[\![Cell]\!]_{\phi_i}$ and $q_j' = q_j$, $d_j' = d_j$, $m_j' = m_j$ for every $j \in \{0,1,\ldots,n-1\}$ different from i. By the definition of *Cell*, it implies that $q_i \in Q$, $q_i' \in Q \cup \{yes,no,halt\}$, $m_i' = m_i = -$ and $\delta(q_i,d_i) = (q_i',d_i',-)$. Because $q_0 = q_{l \bmod n} = \triangleright$, i cannot be 0 or l mod n. Therefore, $d_0' = d_0 = \triangleright$, $m_0' = m_0 \neq \leftarrow$, $d_{l \bmod n}' = d_{l \bmod n} = \triangleright$ and $m_{l \bmod n}' = m_{l \bmod n} \neq \leftarrow$. If $l \leq i < n$, then we choose k' to be k, which means that k' is the unique integer in $\{0,1,\ldots,l-1\}$ such that $q_{k'}' \neq 0$ and Configurations (8) and (9) are the same, hence, the induction hypothesis clearly holds. On the other hand, if $0 \leq i < l$, then i must be equal to k, because $q_i \in Q$ and there is at most one such element in $\{0,1,\ldots,l-1\}$. Furthermore, as $q_i' \in Q \cup \{yes,no,halt\}$, by choosing k' to be k makes it the unique element in $\{0,1,\ldots,l-1\}$ such that $q_{k'}' \neq 0$. It also means that

$$(q_{k'}', d_0 d_1 \ldots d_{k'-1} d_{k'}', d_{k'+1} d_{k'+2} \ldots d_{l-1})$$

is a configuration that follows Configuration (8). However, this the same configuration as Configuration (9), which means that the induction hypothesis holds.

If α is $(a_i, a_{(i-1) \bmod n}, q)$ for some $i \in \{0,1,\ldots,n-1\}$ and $q \in Q \cup \{yes,no,halt\}$, then

$$((q_i,d_i,m_i),\alpha,(q_i',d_i',m_i'))$$

and

$$((q_{(i-1) \bmod n}, d_{(i-1) \bmod n}, m_{(i-1) \bmod n}), \alpha, (q_{(i-1) \bmod n}', d_{(i-1) \bmod n}', m_{(i-1) \bmod n}'))$$

are transitions of respectively $[\![Cell]\!]_{\phi_i}$ and $[\![Cell]\!]_{\phi_{(i-1) \bmod n}}$. Because $(a_i, a_{(i-1) \bmod n}, q)$ is in the alphabet of $[\![Cell]\!]_{\phi_j}$ if and only if $j \in \{i, (i-1) \bmod n\}$, it implies that $q_j' = q_j$, $d_j' = d_j$, and $m_j' = m_j$ for every $j \in \{0,1,\ldots,n-1\}$ different from i and $(i-1)$ mod n. By the definition of *Cell*, then $m_i = m_{(i-1) \bmod n}' = -$, $m_{(i-1) \bmod n} = \rightarrow$, $m_i' = \leftarrow$, $q_i' = q_{(i-1) \bmod n} = 0$, $\delta(q_i,d_i) = (q,d_i',\leftarrow)$, $q_i \in Q$, $q_{(i-1) \bmod n}' = q$, and $d_{(i-1) \bmod n} = d_{(i-1) \bmod n}'$. As $d_0 = d_{l \bmod n} = \triangleright$, i cannot be 0 or l mod n. Hence, $d_0' = d_{l \bmod n}' = \triangleright$ and $m_0', m_{l \bmod n}' \neq \leftarrow$. If $l < i < n$, then we choose k' to be k, which means that k' is the unique integer in $\{0,1,\ldots,l-1\}$ such that $q_{k'}' \neq 0$ and Configurations (8) and (9) are the same, hence, the induction hypothesis holds. On the other hand, if $0 < i < l$, we reason like above to see that then i must be equal to k and by choosing k' to be $i-1$ makes it the unique element in $\{0,1,\ldots,l-1\}$ such that $q_{k'}' \neq 0$. Moreover, the choice implies that

$$(q_{i-1}', d_0 d_1 \ldots d_{i-1}, d_i' d_{i+1} d_{i+2} \ldots d_{l-1})$$

is a configuration that follows Configuration (8). However, the configuration above is the same as Configuration (9), which means that the induction hypothesis holds.

Finally, we assume that α is $(a_i, a_{(i+1) \bmod n}, q)$ for some $i \in \{0,1,\ldots,n-1\}$ and $q \in Q \cup \{yes, no, halt\}$, which means that

$$((q_i, d_i, m_i), \alpha, (q'_i, d'_i, m'_i))$$

and

$$((q_{(i+1) \bmod n}, d_{(i+1) \bmod n}, m_{(i+1) \bmod n}), \alpha, (q'_{(i+1) \bmod n}, d'_{(i+1) \bmod n}, m'_{(i+1) \bmod n}))$$

are transitions of respectively $[\![Cell]\!]_{\phi_i}$ and $[\![Cell]\!]_{\phi_{(i+1) \bmod n}}$. Because $(a_i, a_{(i+1) \bmod n}, q)$ is in the alphabet of $[\![Cell]\!]_{\phi_j}$ if and only if $j \in \{i, (i+1) \bmod n\}$, then $q'_j = q_j$, $d'_j = d_j$, and $m'_j = m_j$ for every $j \in \{0,1,\ldots,n-1\}$ different from i and $(i+1) \bmod n$. By the definition of *Cell*, it means that $m_i = m'_{(i+1) \bmod n} = -$, $m_{(i+1) \bmod n} = \leftarrow$, $m'_i = \rightarrow$, $q'_i = q_{(i+1) \bmod n} = 0$, $\delta(q_i, d_i) = (q, d'_i, \rightarrow)$, $q_i \in Q$, $q = q'_{(i+1) \bmod n}$, and $d_{(i+1) \bmod n} = d'_{(i+1) \bmod n}$. By the definition of a DTM, $d'_0 = d'_{l \bmod n} = \triangleright$ even if $i \in \{0, l \bmod n\}$, and clearly $m'_0, m'_{l \bmod n} \neq \leftarrow$, too. Moreover, as $m_{(i+1) \bmod n} = \leftarrow$, it implies that i cannot be $l-1$ or $n-1$. Now, we reason like earlier to see that the induction hypothesis holds. The case when $l \leq i < n-1$ is clear. If $0 \leq i < l-1$ then i must be equal to k and the choice $k' = i+1$ makes k' the only element in $\{0,1,\ldots,l-1\}$ such that $q'_{k'} \neq 0$ and

$$(q'_{i+1}, d_0 d_1 \ldots d_{i-1} d'_i d_{i+1}, d_{i+2} d_{i+3} \ldots d_{l-1})$$

a configuration that follows Configuration (8). However, the configuration above is the same as Configuration (9), which means that the induction hypothesis holds in this case as well.

Hence, by the induction principle, the lemma holds. □

The lemmas above imply that a DTM $\mathcal{M}$ and the LTS schema $DTM_{\mathcal{M}}$ can simulate each other in the sense that $\mathcal{M}$ halts if and only if there is a valuation $\phi \in \Phi_{DTM}$ compatible with $DTM_{\mathcal{M}}$ such that $[\![DTM_{\mathcal{M}}]\!]_\phi$ executes a halting action. Hence, the halting problem can be converted to a parameterised verification task. However, to prove the desired undecidability result, we still have to show that such a parameterised verification task can be expressed as an instance $\mathcal{Q}, \mathcal{P}, \mathrm{va}(c)$ of Parameterised Traces Refinement, where $\mathcal{Q}$ does not have a hiding LTS subschema, and c is a valuation formula.

Certainly, the main effort in this task is to encode the set Φ_{DTM} of valuations as a valuation formula.[1] In order to do it, we first define an existential-free valuation formula

$$\begin{aligned} c_{lin} = \forall x_1,x_2,x_3,x_4,x_5,x_1',x_2',x_3' \in T : ((x_1,x_2,x_3) \in \Pi \to (x_1 \neq x_2 \wedge x_2 \neq x_3)) \wedge \\ (((x_1,x_2,x_3) \in \Pi \wedge (x_1,x_2',x_3') \in \Pi) \to (x_2 = x_2' \wedge x_3 = x_3')) \wedge \\ (((x_1,x_2,x_3) \in \Pi \wedge (x_2,x_3',x_4) \in \Pi) \to x_3 = x_3') \wedge \\ (((x_1,x_2,x_3) \in \Pi \wedge (x_1',x_2,x_3') \in \Pi) \to (x_1 = x_1' \wedge x_3 = x_3')) \wedge \\ (((x_1,x_2,x_3) \in \Pi \wedge (x_1',x_2',x_3) \in \Pi) \to (x_1 = x_1' \wedge x_2 = x_2')) \wedge \\ (((x_1,x_2,x_3) \in \Pi \wedge (x_2',x_3,x_4) \in \Pi) \to x_2 = x_2') \wedge \\ (((x_1,x_2,x_3) \in \Pi \wedge (x_3,x_4,x_5) \in \Pi) \to ((x_2,x_3,x_4) \in \Pi)) , \end{aligned}$$

which says that a cell cannot be succeeded by itself, each cell has unique successors, neighbours, and predecessors, and if two cells x_2 and x_4 have a common neighbour x_3, then x_2 and x_4 are the neighbours of x_3. Hence, $\mathrm{va}(c_{lin})$ denotes a set of valuations each of which represents a set of rings and open chains of cells.

Intuitively, each valuation in $\mathrm{va}(c_{lin})$ can be partitioned into a finite non-empty set of rings and open chains of cells. To specify this fact more formally, recall that rings are represented by valuations in Φ_{DTM}. Respectively, open chains of cells are represented by valuations ϕ with the domain $\{T,\Pi\}$ such that

- $\phi(T) = \{a_0, a_1, \ldots, a_{n-1}\}$ for some positive integer n, and
- $\phi(\Pi) = \{(a_{(i-1) \bmod n}, a_i, a_{(i+1) \bmod n}) \mid i \in \{1,2,\ldots,n-2\}\}$.

We write Φ_{lin} for the set of all such valuations that encode the open chains of cells. Now, the valuations in $\mathrm{va}(c_{lin})$ can be defined with the aid of those in $\Phi_{DTM} \cup \Phi_{lin}$ as follows.

Lemma 46.

1. *Every valuation in* $\Phi_{DTM} \cup \Phi_{lin}$ *is also a valuation in* $\mathrm{va}(c_{lin})$.
2. *For every valuation* $\phi \in \mathrm{va}(c_{lin})$ *there are valuations* $\phi_1, \phi_2, \ldots, \phi_n \in \Phi_{DTM} \cup \Phi_{lin}$, *where* $n \in \mathbb{Z}_+$, *such that* $\{\phi_i(T)\}_{i=1}^n$ *is a partition of* $\phi(T)$ *and* $\bigcup_{i=1}^n \phi_i(\Pi) = \phi(\Pi)$.

The claims follow straightforwardly from definitions.

To simulate a DTM, we need to exclude the open chains of instances of *Cell*, because such structures can execute halting actions any time. That is why we impose an

[1] The fact that it is actually possible was pointed out by Antti Valmari.

additional requirement that each cell should have at least one successor, which can be expressed as a valuation formula

$$c_{rng} = c_{lin} \wedge \forall x_1 \in T : \exists x_2, x_3 \in T : (x_1, x_2, x_3) \in \Pi \,.$$

Now, the set $\mathrm{va}(c_{rng})$ consists of valuations each of which represents a set of rings. Hence, Φ_{DTM} is included in the set $\mathrm{va}(c_{rng})$ and every valuation in $\mathrm{va}(c_{rng})$ can be divided into finitely many valuations in Φ_{DTM}.

Lemma 47.

1. *Every valuation in Φ_{DTM} is also a valuation in* $\mathrm{va}(c_{rng})$.
2. *For every valuation $\phi \in \mathrm{va}(c_{rng})$ there are valuations $\phi_1, \phi_2, \ldots, \phi_n \in \Phi_{DTM}$, where $n \in \mathbb{Z}_+$, such that $\{\phi_i(T)\}_{i=1}^n$ is a partition of $\phi(T)$ and $\bigcup_{i=1}^n \phi_i(\Pi) = \phi(\Pi)$.*

The claims follow straightforwardly from definitions and Lemma 46.

Another complication is to come up with a specification which can detect halting and only halting. For that purpose, we introduce a single state elementary LTS schema *NoHlt* with the alphabet schema $\{(c, halt)\}$ and no transition schema. Now, the instances of the LTS schema $\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NoHlt$ have all the halting actions in their alphabet but cannot execute a single action. Hence, if the initialisation and token actions represented by a set schema

$$Tok := \bigcup_{(c_1,c_2)\in T\times T} \{init, (c_1, c_2, q) \mid q \in Q \cup \{yes, no, halt\}\}$$

are hidden in $DTM_{\mathcal{M}}$, we can take $\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NoHlt$ as a specification, because halting can then be detected as an incorrect behaviour.

Theorem 48. *A DTM $\mathcal{M}$ halts with the empty input if and only if the answer to the instance*

$$\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NoHlt, DTM_{\mathcal{M}} \setminus Tok, \mathrm{va}(c_{rng})$$

of Parameterised Traces Refinement is negative.

Proof. First, suppose that $\mathcal{M}$ halts with the empty input. Then, there is a run ρ of $\mathcal{M}$ from the configuration $(\hat{q}, \triangleright, \sqcup^{n-1})$ to a configuration $(q, d_0 d_1 \ldots d_k, d_{k+1} d_{k+2} \ldots d_{n-1})$, where $\hat{q}$ is the initial state of $\mathcal{M}$, and $q \in \{halt, yes, no\}$. We may assume that n is an integer greater than one and greater than the scope of ρ. Then, by Lemma 43, there is a valuation $\phi \in \Phi_{DTM}$ compatible with $DTM_{\mathcal{M}}$ such that a state

$$s := ((0, d_0, \rightarrow), \ldots, (0, d_{k-1}, \rightarrow), (q, d_k, -), (0, d_{k+1}, \leftarrow), \ldots, (0, d_{n-1}, \leftarrow))$$

is reachable in $[\![DTM_{\mathcal{M}}]\!]_\phi$ by some execution π. It means that there is a state s' such that $\pi\,(a_k, halt)\,s'$ is an execution of $[\![DTM_{\mathcal{M}}]\!]_\phi$, and therefore $[\![DTM_{\mathcal{M}} \setminus Tok]\!]_\phi$ has a non-empty trace of halting actions.

On the other hand, ϕ is also compatible with $\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NoHlt$, and the only trace of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NoHlt]\!]_\phi$ is the empty sequence. Because by Lemma 47, ϕ is also in $\mathrm{va}(c_{rng})$, it implies that the answer to the instance

$$\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NoHlt, DTM_{\mathcal{M}} \setminus Tok, \mathrm{va}(c_{rng})$$

of Parameterised Traces Refinement is negative.

Next, let us suppose that the answer to the instance of Parameterised Traces Refinement above is negative. Because, for every compatible valuation $\phi \in \mathrm{va}(c_{rng})$, the alphabets of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NoHlt]\!]_\phi$ and $[\![DTM_{\mathcal{M}} \setminus Tok]\!]_\phi$ match, and the former LTS has the empty trace only, there must be a compatible valuation $\phi \in \mathrm{va}(c_{rng})$ and $a \in \phi(T)$ such that $(a, halt)$ is a trace of $[\![DTM_{\mathcal{M}} \setminus Tok]\!]_\phi$. Therefore, $[\![DTM_{\mathcal{M}}]\!]_\phi$ has a trace containing the action $(a, halt)$.

By Lemma 47, we may split ϕ into finitely many subvaluations in Φ_{DTM}. Let Θ be the set of all such subvaluations. Clearly, $\mathbf{P}(DTM_{\mathcal{M}}, \phi, \Theta)$, which by Lemma 27 implies that $[\![DTM_{\mathcal{M}}]\!]_\phi =_{\mathrm{tr}} \|_{\theta\in\Theta} [\![DTM_{\mathcal{M}}]\!]_\theta$. By Lemma 3, it means that there is $\theta \in \Theta$ such that $[\![DTM_{\mathcal{M}}]\!]_\theta$ has a trace t containing the action $(a, halt)$. Hence, there is an execution $\pi\,(a, halt)\,(s'_0, s'_1, \ldots, s'_{n-1})$ of $[\![DTM_{\mathcal{M}}]\!]_\theta$, where n is the size of $\theta(T)$ and $s'_j := (q'_j, d'_j, m'_j)$ is a state of $[\![Cell]\!]_{\theta_j}$ for every $j \in \{0, 1, \ldots, n-1\}$. By the definition of *Cell*, it implies that there is $i \in \{1, 2, \ldots, n\}$ such that q'_i is either *yes*, *no* or *halt*.

Let us then consider the transitions of the form $((\hat{s}, \hat{s}, \ldots, \hat{s}), \alpha, (s_0, s_1, \ldots, s_{n-1}))$, where $\hat{s}$ is the initial state of *Cell*. Now, α must be *init*, and s_j has to be $(0, \sqcup, \leftarrow)$ or $(\hat{q}, \triangleright, -)$ for each $j \in \{0, 1, \ldots, n-1\}$. If $s_j = (0, \sqcup, \leftarrow)$ for all $j \in \{0, 1, \ldots, n-1\}$, then $[\![DTM_{\mathcal{M}}]\!]_\theta$ deadlocks after executing the transition. Hence, the state $(s'_0, s'_1, \ldots, s'_{n-1})$ must be reachable from a state $(s_0, s_1, \ldots, s_{n-1})$ such that $s_j = (\hat{q}, \triangleright, -)$ for some $j \in \{0, 1, \ldots, n-1\}$.

By above, we know that a state $((q_0, d_0, m_0), (q_1, d_1, m_1), \ldots (q_{n-1}, d_{n-1}, m_{n-1}))$ of $[\![DTM_{\mathcal{M}}]\!]_\theta$, where $q_k \in \{yes, no, halt\}$ for some $k \in \{0, 1, \ldots, n-1\}$, is reachable from a state $(s_0, s_1, \ldots, s_{n-1})$, where $s_j \in \{(0, \sqcup, \leftarrow), (\hat{q}, \triangleright, -)\}$ for all $j \in \{0, 1, \ldots, n-1\}$, and $s_i = (\hat{q}, \triangleright, -)$ for some $i \in \{0, 1, \ldots, n-1\}$. If $i \neq k$ we may assume that i and k are chosen in a way such that $s_{i+1 \bmod n} = s_{i+2 \bmod n} = \ldots = s_k = (0, \sqcup, \leftarrow)$. By Lemma 44, we may rotate the components of the states which allows us to assume that $i = 0$. Now, we pick l to be the smallest positive integer such that $s_{l \bmod n} = (\hat{q}, \triangleright, -)$. Note that such

l exists and it is necessarily at most n. By Lemma 45, it means that there is a unique $k' \in \{0,1,\ldots,l-1\}$ such that $q_{k'} \neq 0$, and a run from the configuration $(\hat{q}, \triangleright, \sqcup^{l-1})$ to the configuration $(q_{k'}, d_0 d_1 \ldots d_{k'}, d_{k'+1} d_{k'+2} \ldots d_{l-1})$. Because $q_k \neq 0$ and k is smaller than l, it implies that $k = k'$. In other words, $\mathcal{M}$ halts with the empty input. □

The result leads immediately to the undecidability of Parameterised Traces Refinement in the case of the specifications without hiding and the sets of valuations expressed as a valuation formula.

Corollary 49. *Let $\mathcal{Q}$ and $\mathcal{P}$ be finite and closed LTS schemata such that $\mathcal{Q}$ has no hiding LTS subschema, and c a valuation formula. The question whether $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$ for all compatible valuations $\phi \in \mathrm{va}(c)$ is undecidable.*

Hence, the cut-off result and the reduction algorithm cannot be extended to arbitrary valuation formulae without imposing additional restrictions on specification or system LTS schemata.

8.3 Undecidability with Specifications Involving Hiding

Next, we will show that keeping a valuation formula existential-free but allowing the use of hiding in a specification makes the problem undecidable, too. Obviously, the valuation formula c_{rng} cannot be used because it is not existential-free. On the other hand, the existential-free valuation formula c_{lin} allows the open chains of cells along with the rings, which results in false halting alerts, because for example an open chain of a single cell can always execute halting actions (if it has any). In general, the problem is that there is less synchronisation between the cells in a chain than those in a ring. However, the fact that chains of cells may have more behaviours than rings, can be made use of on the specification side, which enables us to create a specification which allows only the cells in chains to execute halting actions.

To create such a specification, note that one can test if the cells are in a chain by checking whether each of them can execute a certain action before its successor. This notion leads to a specification $(\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} \mathit{NHRng}) \setminus \mathit{Lin}$, where *NHRng* is an elementary LTS schema in Figure 23 and *Lin* is a set schema $\bigcup_{c\in T}\{(c, \mathit{linear})\}$. It is not difficult to see that the instances of *NHRng* related to rings deadlock immediately, whereas the instances related to open chains of cells can agree on the execution of all the actions. It implies that the halting problem can be stated as an instance of Parameterised

Traces Refinement, where the set of valuations is expressed with the aid of an existential-free valuation formula.

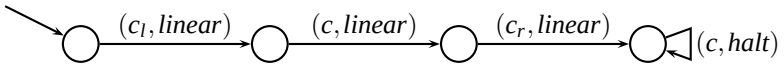

Fig 23. LTS schema *NHRng*.

Theorem 50. *A DTM $\mathcal{M}$ halts with the empty input if and only if the answer to the instance*

$$(\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng) \setminus Lin, DTM_{\mathcal{M}} \setminus Tok, \mathrm{va}(c_{lin})$$

of Parameterised Traces Refinement is negative.

Proof. The proof that the halting of $\mathcal{M}$ with the empty input implies the negative answer to the instance

$$(\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng) \setminus Lin, DTM_{\mathcal{M}} \setminus Tok, \mathrm{va}(c_{lin})$$

of Parameterised Traces Refinement is similar to the corresponding case of Theorem 48.

Also the opposite proof is analogous despite a few details. First, note that for every compatible valuation $\psi \in \mathrm{va}(c_{lin})$,

$$\mathrm{alph}([\![(\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng) \setminus Lin]\!]_\psi) = \mathrm{alph}([\![DTM_{\mathcal{M}} \setminus Tok]\!]_\psi)\,.$$

Hence, if the answer to the instance of Parameterised Traces Refinement above is negative, then there is a compatible valuation $\phi \in \mathrm{va}(c_{lin})$, an atom $a \in \phi(T)$, and a sequence $t \in \{(b, halt) \mid b \in \phi(T)\}^*$ such that $t(a, halt)$ is a minimal trace of $[\![DTM_{\mathcal{M}} \setminus Tok]\!]_\phi$ which is not a trace of $[\![(\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng) \setminus Lin]\!]_\phi$, too. By Lemma 3, then the LTS $[\![DTM_{\mathcal{M}}]\!]_\phi$ has a trace containing the action $(a, halt)$. Moreover, by Lemma 46, we may split ϕ into a finitely many subvaluations in Φ_{DTM} and Φ_{lin}. Let Θ be the set of all such subvaluations. Obviously, $\mathbf{P}(DTM_{\mathcal{M}}, \phi, \Theta)$, which by Lemma 27 implies that $[\![DTM_{\mathcal{M}}]\!]_\phi =_{\mathrm{tr}} \|_{\theta\in\Theta}[\![DTM_{\mathcal{M}}]\!]_\theta$. By Lemma 3, it means that there is $\theta \in \Theta$ such that $[\![DTM_{\mathcal{M}}]\!]_\theta$ has a trace containing the action $(a, halt)$. Next, we show that θ is in Φ_{DTM}, in which case the rest of the proof proceeds as in the case of Theorem 48.

By Lemma 3, there is a trace t' of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_\phi$ such that t is obtained from t' by removing all actions $(b, linear)$, where $b \in \phi(T)$. Like above, it is

obvious that $\mathbf{P}(\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng, \phi, \Theta)$, which, by Lemma 27, implies that

$$[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_\phi =_{\mathrm{tr}} \|_{\theta\in\Theta} [\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_\theta .$$

By Lemma 3, for every $\theta' \in \Theta$, there is a trace $t_{\theta'}$ of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_{\theta'}$ such that $t_{\theta'}$ is obtained from t' by removing all the actions not in the alphabet of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_{\theta'}$. If $\theta \in \Phi_{lin}$, there is $t'_\theta \in \{(b, linear) \mid b \in \theta(T)\}^*(a, halt)$ such that $t_\theta t'_\theta \in \mathrm{tr}([\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_\theta)$. In other words, the trace t_θ can be extended to the trace $t_\theta t'_\theta$ which is obtained from $t't'_\theta$ by removing all the actions not in the alphabet of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_\theta$. Moreover, by Lemma 46, the alphabets of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_\theta$ and $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_{\theta'}$ are disjoint for all $\theta' \in \Theta \setminus \{\theta\}$, which means that also the trace $t_{\theta'}$ is obtained from $t't'_\theta$ by removing all the actions not in the alphabet of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_{\theta'}$, whenever $\theta' \in \Theta \setminus \{\theta\}$. By Lemma 3, it implies that $t't'_\theta$ is a trace of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng]\!]_\phi$ and therefore $t(a, halt)$ is a trace of $[\![\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng \setminus Lin]\!]_\phi$. However, by the choice of t, it is not possible, so θ must be a valuation in Φ_{DTM}. Now, the rest of the proof proceeds as in the case of Theorem 48. □

The result means that Parameterised Traces Refinement is undecidable when arbitrary specification LTS schemata are allowed, even if the sets of valuations are restricted to those expressible as existential-free valuation formulae.

Corollary 51. *Let $\mathcal{Q}$ and $\mathcal{P}$ be finite and closed LTS schemata, and c an existential-free valuation formula. The question whether $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$ for all compatible valuations $\phi \in \mathrm{va}(c)$ is undecidable.*

Hence, the cut-off result and the reduction algorithm cannot be extended to arbitrary specification LTS schemata either without imposing additional restrictions on system LTS schemata or valuation formulae.

8.4 Undecidability for More Expressive Semantics

So far, we have only considered the verification of safety properties. However, one might be interested in the detection of deadlocks and the analysis of liveness properties as well (Alpern & Schneider 1985). Nevertheless, as we will show next, the result cannot be extended in this direction either, because allowing such specifications makes the problem undecidable.

The weakest deadlock-preserving congruence in the set of LTSs is the *(stable) failures equivalence* (Valmari 1995). Let $\mathcal{L}$ be an LTS. A pair (t,Λ), where t is a trace of $\mathcal{L}$ and $\Lambda \subseteq \text{alph}(\mathcal{L})$, is a *(stable) failure* of $\mathcal{L}$, if there is an execution π of $\mathcal{L}$ to a state s of $\mathcal{L}$ such that

– t can be obtained from π by removing all the states and the invisible actions, and
– (s,α,s') is not a transition of $\mathcal{L}$ for any state s' and action $\alpha \in \Lambda \cup \{\tau\}$.

The set of all the stable failures of $\mathcal{L}$ is denoted by $\text{fa}(\mathcal{L})$. We say that an LTS $\mathcal{L}_2$ is a *failures refinement* of an LTS $\mathcal{L}_1$, denoted by $\mathcal{L}_1 \succeq_{\text{fa}} \mathcal{L}_2$, if the alphabets of $\mathcal{L}_1$ and $\mathcal{L}_2$ are the same and $\text{fa}(\mathcal{L}_2) \subseteq \text{fa}(\mathcal{L}_1)$. The LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$ are *failures equivalent*, if $\mathcal{L}_1$ is a failures refinement of $\mathcal{L}_2$ and $\mathcal{L}_2$ is a failures refinement of $\mathcal{L}_1$.

To show that the cut-off result and the reduction algorithm cannot be extended to failures semantics, we use a similar idea as in the case of specifications involving hiding. However, now the test for chains has to be done with the aid of failures, because one cannot use hiding in the specification anymore. That is why we move the actions expressing linearity to the system model and hide them to introduce loops of the invisible actions to instances in a chain. We write *Cell*′ for an elementary LTS schema obtained from *Cell* by adding transition schemata $(s,(c_l,\mathit{linear}),s'),(s',(c,\mathit{linear}),s''),(s'',(c_r,\mathit{linear}),s)$ for every state s, where s',s'' are new states unique to s. Hence, our DTM model is now an LTS schema $\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} \mathit{Cell}'$ denoted by $\mathit{DTM}'_{\mathcal{M}}$, and the system LTS schema is $\mathit{DTM}'_{\mathcal{M}} \setminus \mathit{Tok}'$, where *Tok*′ is a set schema

$$\mathit{Tok}' := \bigcup_{(c_1,c_2)\in T\times T} \{\mathit{init},(c_1,c_2,q),(c_1,\mathit{linear}) \mid q \in Q \cup \{\mathit{yes},\mathit{no},\mathit{halt}\}\} .$$

The construction guarantees that an instance of *Cell*′ can deadlock after executing a halting action if and only if it is a part of a ring.

Now, to detect halting in a ring, we create a specification that does not allow an instance of *Cell*′ to deadlock after executing a halting action. This behaviour is captured in an elementary LTS schema *NHRng*′ in Figure 24. Hence, if we take the LTS schema $\|_{(c_l,c,c_r)\in\Pi:T\times T\times T} \mathit{NHRng}'$ as a specification, then halting in a ring can be detected as a violation of correctness.

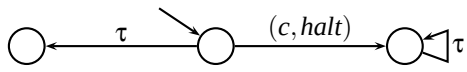

Fig 24. LTS schema *NHRng*′**.**

Theorem 52. *A DTM $\mathcal{M}$ halts with the empty input if and only if*

$$[\![\mathop{\Vert}_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng']\!]_\phi \succeq_{\mathrm{fa}} [\![DTM'_\mathcal{M} \setminus Tok']\!]_\phi$$

does not hold for some compatible valuation $\phi \in \mathrm{va}(c_{lin})$.

Proof. First, we proceed like in the proof of Theorem 48, to see that when $\mathcal{M}$ halts with the empty input, then there is a valuation $\phi \in \Phi_{DTM}$ compatible with $DTM_\mathcal{M}$ such that a state

$$s := ((0,d_0,\rightarrow),\ldots,(0,d_{k-1},\rightarrow),(q,d_k,-),(0,d_{k+1},\leftarrow),\ldots,(0,d_{n-1},\leftarrow)) ,$$

where $q \in \{yes, no, halt\}$, is reachable in $[\![DTM_\mathcal{M}]\!]_\phi$ by some execution π. We may pick ϕ in such a way that it is compatible with $[\![DTM'_\mathcal{M}]\!]_\phi$, too. Then, by construction, the state s is reachable by π in $[\![DTM'_\mathcal{M}]\!]_\phi$ as well, which means that the state

$$s' := ((0,d_0,\rightarrow),\ldots,(0,d_{k-1},\rightarrow),(q,d_k,\rightarrow),(0,d_{k+1},\leftarrow),\ldots,(0,d_{n-1},\leftarrow))$$

is reachable by the execution $\pi\,(a_k, halt)\,s'$ in $[\![DTM'_\mathcal{M}]\!]_\phi$. Now, it is easy to see that $[\![DTM'_\mathcal{M}]\!]_\phi$ deadlocks after $\pi\,(a_k, halt)\,s'$, which implies that $((a_k, halt), \Lambda)$ is a failure of $[\![DTM'_\mathcal{M} \setminus Tok']\!]_\phi$ whenever Λ is a subset of $\{(b, halt) \mid b \in \phi(T)\}$.

On the other hand, ϕ is also compatible with $\Vert_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng'$, but the LTS $[\![\Vert_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng']\!]_\phi$ has only failures (ε,Λ) such that Λ is a subset of $\{(b, halt) \mid b \in \phi(T)\}$. Because by Lemma 46, ϕ is also in $\mathrm{va}(c_{lin})$, it implies that ϕ is the compatible valuation in $\mathrm{va}(c_{lin})$ such that

$$[\![\mathop{\Vert}_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng']\!]_\psi \succeq_{\mathrm{fa}} [\![DTM'_\mathcal{M} \setminus Tok']\!]_\psi \tag{10}$$

does not hold.

Next, let us suppose that there is a compatible valuation $\phi \in \mathrm{va}(c_{lin})$ such that Relation (10) does not hold. Because, for every compatible valuation $\psi \in \mathrm{va}(c_{lin})$, the alphabets of $[\![\Vert_{(c_l,c,c_r)\in\Pi:T\times T\times T} NHRng']\!]_\psi$ and $[\![DTM'_\mathcal{M} \setminus Tok']\!]_\psi$ match and the former LTS has all failures (ε,Λ) such that Λ is a subset of $\{(b, halt) \mid b \in \phi(T)\}$, the LTS $[\![DTM'_\mathcal{M} \setminus Tok']\!]_\phi$ must have a failure $(t,\emptyset)$ such that t is a trace containing an action $((a, halt),\emptyset)$ for some $a \in \phi(T)$. It implies that also $[\![DTM'_\mathcal{M}]\!]_\phi$ has a failure $(t',\emptyset)$ such that t' is a trace containing the action $(a, halt)$.

By Lemma 46, we may split ϕ into a finitely many subvaluations in Φ_{DTM} and Φ_{lin}. Let Θ be the set of all such subvaluations. However, in this case, no valuation $\theta \in \Theta$

can be in Φ_{lin}, because otherwise $[\![DTM'_{\mathcal{M}}]\!]_\phi$ could always execute linearity actions, which would make $\mathrm{fa}([\![DTM'_{\mathcal{M}} \setminus Tok']\!]_\phi)$ empty. Hence, Θ is a subset of Φ_{DTM}. Clearly, $\mathbf{P}(DTM'_{\mathcal{M}}, \phi, \Theta)$, which by Lemma 27 implies that $[\![DTM'_{\mathcal{M}}]\!]_\phi =_{\mathrm{tr}} \|_{\theta\in\Theta}[\![DTM'_{\mathcal{M}}]\!]_\theta$. By Lemma 3, it means that there is $\theta \in \Theta$ such that $[\![DTM'_{\mathcal{M}}]\!]_\theta$ has a trace t'' containing the action $(a, halt)$. By construction, t'' is also a trace of $[\![DTM_{\mathcal{M}}]\!]_\theta$, and the rest of the proof proceeds as in the case of Theorem 48. □

The result implies that we can neither extend the cut-off result nor the reduction algorithm to semantics that enables the analysis of deadlocks without imposing additional restrictions on specification or system LTS schemata, or on valuation formulae used to express the sets of valuations.

Corollary 53. *Let $\mathcal{Q}$ and $\mathcal{P}$ be finite and closed LTS schemata such that $\mathcal{Q}$ has no hiding LTS subschema, and c an existential-free valuation formula. The question whether $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{fa}} [\![\mathcal{P}]\!]_\phi$ for all compatible $\phi \in \mathrm{va}(c)$ is undecidable.*

As checking liveness properties necessitates information on failures and livelocks, extending the algorithm to liveness properties is also impossible, without further restrictions on specification or system LTS schemata, or valuation formulae. On the other hand, if one is only interested in safety properties and livelocks, then the extension appears to be possible. It looks like all the key results hold for the weakest livelock-preserving congruence (Puhakka & Valmari 1999), but proving it will be left as a topic for future research.

8.5 Undecidability for Systems with Parametric Branching

We have shown that one cannot allow hiding in the specification without making the problem undecidable or restricting the result from some other aspect. However, we do not know if it is possible to strengthen the system LTS schema. Next, we will consider the most obvious extensions to our LTS schema formalism and prove that they make the problem undecidable.

Probably the most serious limitation of our process model is the lack of a structure that enables expressing systems where the number of paths between certain states is parameter-dependent but the lengths of the paths are not. Consequently, we refer to such a construct by *parametric branching (with joining)*.

With the aid of a replicated parallel LTS schema, you cannot generally parameterise the number of alternative execution paths without parameterising their lengths as well.

The exceptions are the simplest cycles where the branches immediately return to the state from which they originated. We have used such a trivial parametric branching in Section 6.4 to model the conjunctive guards of systems originally considered by Emerson & Kahlon (2000). Hence, in the simplest kind of parametric branching not supported by our formalism, branches are joined in the state that immediately follows branching. This can be achieved by allowing type variables to be used in place of atom variables in a transition schema.

An *action set schema* is a non-empty tuple $(\sharp_1, \sharp_2, \ldots, \sharp_k, a_1, a_2, \ldots, a_l)$, where $k + l$ is a positive integer, $\sharp_1, \sharp_2, \ldots, \sharp_k$ are atom or type variables, and $a_1, a_2, \ldots, a_l$ are atoms. An *extended elementary LTS schema* is an elementary LTS schema where action set schemata are used in place of action schemata, and respectively an *extended LTS schema* is an LTS schema where extended elementary LTS schemata are used in place of elementary LTS schemata. The concepts and notation, like the signature, finiteness and compatibility, are defined for extended LTS schemata analogously to LTS schemata.

An instance of an extended LTS schema $\mathcal{P}$ generated by a compatible valuation ϕ is obtained in an obvious way as follows. If $\mathcal{P}$ is a parallel, replicated parallel or hiding LTS schema, then $[\![\mathcal{P}]\!]_\phi$ is defined as in the case of LTS schemata. When $\mathcal{P}$ is an extended elementary LTS schema, then $[\![\mathcal{P}]\!]_\phi$ is defined like in the case of an elementary LTS schema but an action set schema $(\sharp_1, \sharp_2, \ldots, \sharp_k, a_1, a_2, \ldots, a_l)$ occurring in $\mathcal{P}$ is interpreted as a set of all actions $(b_1, b_2, \ldots, b_k, a_1, a_2, \ldots, a_l)$ such that $b_i = \phi(\sharp_i)$ for each $i \in \{1, 2, \ldots, k\}$ such that $\sharp_i$ is an atom variable, and $b_i \in \phi(\sharp_i)$ for each $i \in \{1, 2, \ldots, k\}$ such that $\sharp_i$ is a type variable.

To prove that the use of extended elementary LTS schemata in a system model results in undecidability, we obviously need a different model for a DTM. Now, instead of a ring system, we model a DTM as a two-stack machine, where one stack is used to store the contents of the tape on the left of the read-write head, and the other stack stores the contents of the tape on the right of the read-write head, respectively. The control part stores the state of the simulated DTM together with the contents of the cell at the read-write head. The stacks can be modelled as standard LTS schemata, but reading and writing to them necessitates the use of action set schemata.

To present the parts of the two-stack machine formally, let $\mathcal{M}$ be a DTM $(Q, D, \delta, \hat{q})$. First, we model the left stack from the viewpoint of two different memory locations c_1 and c_2. The behaviour is captured in an elementary LTS schema Tape_l which allows data to be pushed into and popped from the locations c_1 and c_2 in respectively this and the opposite order.

Formally, the states of Tape_l are pairs $(d_1, d_2) \in (D \cup \{-\}) \times (D \cup \{-\})$, where d_1 and d_2 represent the contents of respectively c_1 and c_2. The symbol $-$ denotes an unused memory location and it is assumed to be different from the tape symbols. Therefore, the initial state of Tape_l is obviously $(-,-)$.

The alphabet schema of Tape_l is the set of all action schemata $(c_i, \mathit{lft}, \mathit{psh}, d)$ and $(c_i, \mathit{lft}, \mathit{pop}, d)$, where $i \in \{1,2\}$, denoting respectively pushing d into and popping d from a location represented by c_i. Finally, the transition schemata of Tape_l are

- $((-,-),(c_1,\mathit{lft},\mathit{psh},d),(d,-))$ for every $d \in D$,
- $((d,-),(c_1,\mathit{lft},\mathit{pop},d),(-,-))$ for every $d \in D$,
- $((d_1,-),(c_2,\mathit{lft},\mathit{psh},d_2),(d_1,d_2))$ for all $d_1, d_2 \in D$,
- $((d_1,d_2),(c_2,\mathit{lft},\mathit{pop},d_2),(d_1,-))$ for all $d_1, d_2 \in D$.

To create the model of the left stack, we pick a type variable T representing the set of stack locations, and relation variable Ξ which denotes an irreflexive, asymmetric, and transitive relation over stack locations. Now, the model of the left stack can be obtained as the LTS schema $\|_{(c_1,c_2)\in\Xi:T\times T} \mathit{Tape}_l$, which allows data to be pushed into stack locations in the order specified by Ξ and popped from in the opposite order. Note that our model of the left stack not only includes instances of Tape_l where c_1 and c_2 represent adjacent locations but all instances of Tape_l such that c_1 represents a location lower in the stack array than c_2. It does not affect the behaviour of the model but allows its topology to be expressed without the use of existential quantification.

The right stack is created in a similar way. However, we allow the blank cell symbol to be popped from the stack when it is empty, which corresponds to moving the read-write head to the right to a cell not visited before. Hence, the right stack can be modelled as an LTS schema $\|_{(c_1,c_2)\in\Xi:T\times T} \mathit{Tape}_r$, where Tape_r is an elementary LTS schema obtained from Tape_l by substituting *rgh* for every occurrence of *lft*, and adding a transition schema $((-,-),(c_1,\mathit{rgh},\mathit{pop},\sqcup),(-,-))$.

Note that our stack models accept the push actions related only to the first unused location and pop actions related only to the last non-empty location. Hence, to be able to push data into and pop it from a stack we should be able to pick respectively the first unused and the last non-empty locations, which necessitates the use of action set schemata.

The control part of $\mathcal{M}$ is naturally modelled as an extended elementary LTS schema, which we denote by $\mathit{Ctrl}_{\mathcal{M}}$. As $\mathit{Ctrl}_{\mathcal{M}}$ stores the state of $\mathcal{M}$ and the symbol at the position of the read-write head, an execution step of $\mathcal{M}$, during which the read-write head

is not moved, can be modelled as an internal transition between the states of $Ctrl_{\mathcal{M}}$. However, modelling other execution steps of $\mathcal{M}$ requires communication between both the stacks, which means that two transition schemata are needed to represent them. First, the symbol at the position of the read-write head has to be pushed into one stack, and then the read-write head is moved by popping a symbol from the other stack. Additionally, we allow $Ctrl_{\mathcal{M}}$ to execute the action *halt*, denoting halting, whenever an accepting, rejecting, or a halting state is reached.

Formally, the states of $Ctrl_{\mathcal{M}}$ are pairs $(q,d) \in (Q \cup \{halt, yes, no\}) \times (D \cup \{\leftarrow, \rightarrow\})$, where q is the state of the simulated DTM, and d denotes the symbol at the position of the read-write head either explicitly or implicitly by pointing to the topmost element of either of the stacks. The initial state of $Ctrl_{\mathcal{M}}$ is obviously $(\hat{q}, \triangleright)$. The alphabet schema is the smallest set of action set schemata such that the transition schemata of $Ctrl_{\mathcal{M}}$ are

- $((q,d), \tau, (q',d'))$ whenever $\delta(q,d) = (q',d',-)$,
- $((q,d), (T, rgh, psh, d'), (q', \leftarrow))$ whenever $\delta(q,d) = (q',d',\leftarrow)$,
- $((q,\leftarrow), (T, lft, pop, d), (q,d))$ for every $q \in Q \cup \{halt, yes, no\}$ and every $d \in D$,
- $((q,d), (T, lft, psh, d'), (q', \rightarrow))$ whenever $\delta(q,d) = (q',d',\rightarrow)$,
- $((q,\rightarrow), (T, rgh, pop, d), (q,d))$ for every $q \in Q \cup \{halt, yes, no\}$ and every $d \in D$,
- $((q,d), halt, (q,d))$ for every $q \in \{halt, yes, no\}$ and every $d \in D$.

The model of the whole DTM $\mathcal{M}$ can be now obtained by composing all its parts in parallel, which results in an extended LTS schema

$$DTM''_{\mathcal{M}} := \mathop{\|}_{(c_1,c_2) \in \Xi : T \times T} (Tape_l \parallel Ctrl_{\mathcal{M}} \parallel Tape_r) .$$

The instances we are interested in are generated by valuations ϕ with the domain $\{T, \Xi\}$ such that

- $\phi(T) = \{a_1, a_2, \ldots, a_n\}$ for some positive integer n, and
- $\phi(\Xi)$ is a proper total order $\{(a_{i_1}, a_{i_2}) \mid 1 \leq i_1 < i_2 \leq n\}$ over $\phi(T)$.

The set of all such valuations is denoted by Φ'_{DTM}.

It is rather easy to see that every run of $\mathcal{M}$ can be simulated by an execution of the instance of $DTM''_{\mathcal{M}}$ generated by some compatible valuation in Φ'_{DTM}, and vice versa. Whenever $\phi \in \Phi'_{DTM}$ is compatible with $DTM''_{\mathcal{M}}$ and i_1, i_2 are positive integers such that $1 \leq i_1 < i_2 \leq |\phi(T)|$, we write $[\![Tape_l]\!]_{\phi_{i_1,i_2}}$ and $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ for respectively LTSs $[\![Tape_l]\!]_{\phi'}$ and $[\![Tape_r]\!]_{\phi'}$ such that $\phi' \in \phi[(c_1,c_2) \rightarrow \Xi]$ and $\phi'(c_i) = a_{i_i}$ for both

$i \in \{1,2\}$. With this notation, the concept of the mutual simulation can be now captured formally as follows.

Lemma 54. *Let* ρ *be a run of a DTM* $\mathcal{M}$ *from the configuration* $(\hat{q}, \triangleright, \varepsilon)$ *to a configuration* $(q, d_{k-1}d_{k-2}\dots d_0 d, d_{n-1}d_{n-2}\dots d_{k+1})$*, where* $\hat{q}$ *is the initial state of* $\mathcal{M}$*,* n *is the scope of* ρ*, and* k *is a non-negative integer less than* n*. Whenever* $\phi \in \Phi'_{DTM}$ *is a valuation compatible with* $DTM''_{\mathcal{M}}$ *such that* $\max\{2,n\} \leq |\phi(T)|$*, there is a state* s *reachable in* $[\![DTM''_{\mathcal{M}}]\!]_\phi$ *such that the component of* s *corresponding to*

- *the state of* $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ *is* (q,d)*,*
- *the state of* $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ *is* (d_{k+i_1}, d_{k+i_2}) *for all* $1 \leq i_1 < i_2 \leq |\phi(T)|$*, and*
- *the state of* $[\![Tape_l]\!]_{\phi_{i_1,i_2}}$ *is* (d_{k-i_1}, d_{k-i_2}) *for all* $1 \leq i_1 < i_2 \leq |\phi(T)|$*,*

where $d_i = -$ *whenever* $i < 0$ *and* $i \geq n$*.*

Proof. We argue by induction on the length of ρ using the lemma as an induction hypothesis strengthened by the claim that the first symbol in the tape is always $\triangleright$.

In the base step, $\rho = (\hat{q}, \triangleright, \varepsilon)$, which implies that $n = 1$, $k = 0$, $q = \hat{q}$, and $d = \triangleright$. Hence, the first symbol in the tape is $\triangleright$. Moreover, whenever $\phi \in \Phi'_{DTM}$ is a valuation compatible with $DTM''_{\mathcal{M}}$ such that the size of $\phi(T)$ is at least two, the initial state of $[\![DTM''_{\mathcal{M}}]\!]_\phi$, which is clearly reachable, satisfies the conditions concerning the states of the component LTSs. Hence, the base step is clear.

In the induction step, $\rho = \rho'(q, d_{k-1}d_{k-2}\dots d_0 d, d_{n-1}d_{n-2}\dots d_{k+1})$, where ρ' is a run of $\mathcal{M}$. Moreover, we may assume that the last configuration in ρ' is

$$(q', d'_{k'-1}d'_{k'-2}\dots d'_0 d', d'_{n'-1}d'_{n'-2}\dots d'_{k'+1})\,.$$

By the induction hypothesis, the first symbol in the tape after the run ρ is $\triangleright$, and whenever $\phi \in \Phi'_{DTM}$ is a compatible valuation such that $\max\{2,n\} \leq |\phi(T)|$, then there is a state s'' reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$ such that the component of s'' corresponding to

- the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q', d'),
- the state of $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ is $(d'_{k'+i_1}, d'_{k'+i_2})$ for all $1 \leq i_1 < i_2 \leq |\phi(T)|$, and
- the state of $[\![Tape_l]\!]_{\phi_{i_1,i_2}}$ is $(d'_{k'-i_1}, d'_{k'-i_2})$ for all $1 \leq i_1 < i_2 \leq |\phi(T)|$,

where $d'_i = -$ whenever $i < 0$ and $i \geq n'$. Now, there are four cases to consider.

First of all, if $\sigma(q', d') = (q, d, -)$, then $n' = n$, $k' = k$, and $d'_i = d_i$ for every $i \in \{1, 2, \dots, k'-1, k'+1, k'+2, \dots, n'-1\}$. By definition, k' cannot be zero, and $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ has a transition $((q', d'), \tau, (q, d))$. It means that the first symbol in the tape

is still $\triangleright$, and there is a state s reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$ such that the component of s corresponding to the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q,d), but otherwise s is like s''. By above, it implies that s satisfies the conditions concerning the states of the component LTSs, and the induction hypothesis holds.

Secondly, if $\sigma(q',d') = (q,d_{n-1},\leftarrow)$, then $n' = n$, $k' = k+1$, $d'_{k'-i} = d_{k-i}$ for every $i \in \{1,2,\ldots,k\}$, $d'_0 = d$, and $d'_{k'+i} = d_{k+i}$ for every $i \in \{1,2,\ldots,n-k'-1\}$. Moreover, k' cannot be zero, which means that the first symbol in the tape is still $\triangleright$. Furthermore, $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ has a transition $((q',d'),(a_i,rgh,psh,d_{n-1}),(q,\leftarrow))$ for every $i \in \{1,2,\ldots,|\phi(T)|\}$, but only $(a_{n-k'},rgh,psh,d_{n-1})$ is enabled at the state s''. It means that there is a state s' reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$ such that the component of s' corresponding to

- the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is $(q,\leftarrow)$,
- the state of $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ is (d_{k+i_1},d_{k+i_2}) for all $1 \le i_1 < i_2 \le |\phi(T)|$,

where $d_i = -$ whenever $i \ge n$, and otherwise s' is like s''. Additionally, $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ has a transition $((q,\leftarrow),(a_j,lft,pop,d''),(q,d''))$ for every $j \in \{1,2,\ldots,|\phi(T)|\}$ and $d'' \in D$, but only $(a_{k'},lft,pop,d'_0)$ is enabled at the state s'. It means that there is a state s reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$ such that the component of s corresponding to

- the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q,d),
- the state of $[\![Tape_l]\!]_{\phi_{i_1,i_2}}$ is (d_{k-i_1},d_{k-i_2}) for all $1 \le i_1 < i_2 \le |\phi(T)|$,

where $d'_i = -$ whenever $i < 0$, and otherwise s is like s'. By above, it implies that s satisfies the conditions concerning the states of the component LTSs, and the induction hypothesis holds.

Thirdly, if $\sigma(q',d') = (q,d_0,\rightarrow)$ and $k'+1 < n'$, then $n' = n$, $k' = k-1$, $d'_{k'-i} = d_{k-i}$ for all $i \in \{1,2,\ldots,k-1\}$, $d'_{n-1} = d$, and $d'_{k'+i} = d_{k+i}$ for all $i \in \{1,2,\ldots,n-k-1\}$. Note that the first symbol in the tape is preserved even if $k' = 0$. Moreover, $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ has a transition $((q',d'),(a_i,lft,psh,d_0),(q,\rightarrow))$ for every $i \in \{1,2,\ldots,|\phi(T)|\}$, but only $(a_{k'+1},lft,psh,d_0)$ is enabled at the state s''. It means that there is a state s' reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$ such that the component of s' corresponding to

- the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is $(q,\rightarrow)$,
- the state of $[\![Tape_l]\!]_{\phi_{i_1,i_2}}$ is (d_{k-i_1},d_{k-i_2}) for all $1 \le i_1 < i_2 \le |\phi(T)|$,

where $d_i = -$ whenever $i < 0$, and otherwise s' is like s''. Furthermore, $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ has also a transition $((q,\rightarrow),(a_j,rgh,pop,d''),(q,d''))$ for every $j \in \{1,2,\ldots,|\phi(T)|\}$ and

$d'' \in D$, but only $(a_{n-k'-1}, rgh, pop, d'_{n-1})$ is enabled at the state s'. It means that there is a state s reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$ such that the component of s corresponding to

- the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q,d),
- the state of $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ is (d_{k+i_1}, d_{k+i_2}) for all $1 \leq i_1 < i_2 \leq |\phi(T)|$,

where $d_i = -$ whenever $i \geq n$, and otherwise s is like s'. By above, it implies that s satisfies the conditions concerning the states of the component LTSs, and the induction hypothesis holds.

The last possibility is that $\sigma(q', d') = (q, d_0, \rightarrow)$ and $k' + 1 = n'$. Then $n = n' + 1$, $k' = k - 1$, $d'_{k'-i} = d_{k-i}$ for all $i \in \{1, 2, \ldots, k-1\}$, and $d = \sqcup$. We proceed like in the previous case to see that the first symbol in the tape is preserved even if $k' = 0$, and to find a state s' reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$. Like above, $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ has a transition $((q, \rightarrow), (a_j, rgh, pop, d''), (q, d''))$ for every $j \in \{1, 2, \ldots, |\phi(T)|\}$ and $d'' \in D$, but now only $(a_1, rgh, pop, \sqcup)$ is enabled at the state s'. It means that a state s like in the previous case is reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$. As s satisfies the conditions concerning the states of the component LTSs, also the induction step is clear. Hence, by the induction principle, the lemma is correct. □

Lemma 55. *Let $\mathcal{M}$ be a DTM $(Q, D, \delta, \hat{q})$, $\phi \in \Phi'_{DTM}$ a valuation compatible with $DTM''_{\mathcal{M}}$, n an integer equal to $|\phi(T)|$ and greater than one, and s a state reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$. Then there are a state $q \in Q$, an element $d \in D \cup \{\leftarrow, \rightarrow\}$, integers $k_l, k_r \in \{1, 2, \ldots, n+1\}$, and elements $d_{r,1}, d_{r,2}, \ldots, d_{r,k_r-1}, d_{l,1}, d_{l,2}, \ldots, d_{l,k_l-1} \in D$ and $d_{r,k_r}, d_{r,k_r+1}, \ldots, d_{r,n}, d_{l,k_l}, d_{l,k_l+1}, \ldots, d_{l,n} \in \{-\}$ such that the component of s which corresponds to*

- *the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q,d),*
- *the state of $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ is (d_{r,i_1}, d_{r,i_2}) for all $1 \leq i_1 < i_2 \leq n$, and*
- *the state of $[\![Tape_l]\!]_{\phi_{i_1,i_2}}$ is (d_{l,i_1}, d_{l,i_2}) for all $1 \leq i_1 < i_2 \leq n$.*

Moreover, there is a run of $\mathcal{M}$ from the configuration $(\hat{q}, \triangleright, \varepsilon)$ to the configuration

- *$(q, d_{l,1} d_{l,2} \ldots d_{l,k_l-1} d, d_{r,k_r-1} d_{r,k_r-2} \ldots d_{r,1})$ when $d \in D$,*
- *$(q, d_{l,1} d_{l,2} \ldots d_{l,k_l-1}, d_{r,k_r-1} d_{r,k_r-2} \ldots d_{r,1})$ when $d = \leftarrow$, and*
- *$(q, d_{l,1} d_{l,2} \ldots d_{l,k_l-1} d_{r,k_r-1}, d_{r,k_r-2} d_{r,k_r-3} \ldots d_{r,1})$ when $d = \rightarrow$ and $k_r > 1$, and*
- *$(q, d_{l,1} d_{l,2} \ldots d_{l,k_l-1} \sqcup, \varepsilon)$ when $d = \rightarrow$ and $k_r = 1$.*

Proof. We argue by induction on the length of an execution π by which the state s is reachable using the lemma as an induction hypothesis strengthened by the claim that the first symbol in the tape is $\triangleright$.

In the base step, π is the initial state of $[\![DTM''_{\mathcal{M}}]\!]_\phi$. Choosing $q = \hat{q}$, $d = \triangleright$ and $k_r = k_l = 1$ satisfies the claim on the states of component LTSs. Obviously, there is also a run from the configuration $(\hat{q}, \triangleright, \varepsilon)$ to itself and the first symbol in the tape is $\triangleright$, which means that the base step is complete.

In the induction step, π is $\pi'\alpha s$, where π' is an execution of $[\![DTM''_{\mathcal{M}}]\!]_\phi$ to a state s'. By the induction hypothesis, there are a state $q' \in Q$, an element $d' \in D \cup \{\leftarrow, \rightarrow\}$, integers $k'_l, k'_r \in \{1, 2, \ldots, n+1\}$, elements $d'_{r,1}, d'_{r,2}, \ldots, d'_{r,k'_r-1}, d'_{l,1}, d'_{l,2}, \ldots, d'_{l,k'_l-1} \in D$, and $d'_{r,k'_r}, d'_{r,k'_r+1}, \ldots, d'_{r,n}, d'_{l,k'_l}, d'_{l,k'_l+1}, \ldots, d'_{l,n} \in \{-\}$ such that the component of s' corresponding to

- the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q', d'),
- the state of $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ is (d'_{r,i_1}, d'_{r,i_2}) for all $1 \leq i_1 < i_2 \leq n$, and
- the state of $[\![Tape_l]\!]_{\phi_{i_1,i_2}}$ is (d'_{l,i_1}, d'_{l,i_2}) for all $1 \leq i_1 < i_2 \leq n$.

Moreover, there is a run of $\mathcal{M}$ from the configuration $(\hat{q}, \triangleright, \varepsilon)$ to the configuration

- $(q', d'_{l,1} d'_{l,2} \ldots d'_{l,k'_l-1} d', d'_{r,k'_r-1} d'_{r,k'_r-2} \ldots d'_{r,1})$ when $d' \in D$,
- $(q', d'_{l,1} d'_{l,2} \ldots d'_{l,k'_l-1}, d'_{r,k'_r-1} d'_{r,k'_r-2} \ldots d'_{r,1})$ when $d' = \leftarrow$, and
- $(q', d'_{l,1} d'_{l,2} \ldots d'_{l,k'_l-1} d'_{r,k'_r-1}, d'_{r,k'_r-2} d'_{r,k'_r-3} \ldots d'_{r,1})$ when $d' = \rightarrow$ and $k'_r > 1$, and
- $(q', d'_{l,1} d'_{l,2} \ldots d'_{l,k'_l-1} \sqcup, \varepsilon)$ when $d' = \rightarrow$ and $k'_r = 1$

such that first symbol in the tape is $\triangleright$. Now, there are six cases to consider.

First of all, if α is *halt*, then s' is s, and the claim is immediately true.

Secondly, if α is τ, then $\sigma(q', d') = (q, d, -)$ for $q \in Q$ and $d \in D$ such that the component of s corresponding to the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q, d). The state s must be otherwise like s', which implies that by choosing $k_l = k'_l$, $k_r = k'_r$, and $d_{r,i} = d'_{r,i}$ and $d_{l,i} = d'_{l,i}$ for all $i \in \{1, 2, \ldots, n\}$ the state s satisfies the claim on the states of component LTSs. It also means that the configuration $(q, d_{l,1} d_{l,2} \ldots d_{l,k_l-1} d, d_{r,k_r-1} d_{r,k_r-2} \ldots d_{r,1})$ follows the configuration $(q', d'_{l,1} d'_{l,2} \ldots d'_{l,k'_l-1} d', d'_{r,k'_r-1} d'_{r,k'_r-2} \ldots d'_{r,1})$. Hence, there is a run from the initial configuration $(\hat{q}, \triangleright, \varepsilon)$ to a configuration accordant with the lemma. Finally, k_l must be greater than one, which implies that the first symbol in the tape is preserved.

Thirdly, if α is (a_i, rgh, psh, d''), where $i \in \{1, 2, \ldots, n\}$ and $d'' \in D$, then i must be k'_r and hence k'_r must be at most n. It implies that if we set $k_r = k'_r + 1$, $d_{r,k_r-1} = d''$, and $d_{r,i} = d'_{r,i}$ for all $i \in \{1, 2, \ldots, n\}$ different from $k_r - 1$, then the component of s corresponding to the state of $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ is (d_{r,i_1}, d_{r,i_2}) for all $1 \leq i_1 < i_2 \leq n$. Moreover, d' must be in D and $\sigma(q', d')$ has to be $(q, d'', \leftarrow)$, where $q \in Q$ such that the component

of s corresponding to the state of $[\![Ctrl_{\mathcal{M}}]\!]_{\phi}$ is $(q, \leftarrow)$. The state s must be otherwise like s', which implies that by choosing $k_l = k'_l$ and $d_{l,i} = d'_{l,i}$ for all $i \in \{1, 2, \ldots, n\}$ the state s satisfies the claim on the states of component LTSs. Furthermore, k_l must be greater than one, so $(q, d_{l,1}d_{l,2}\ldots d_{l,k_l-1}, d_{r,k_r-1}d_{r,k_r-2}\ldots d_{r,1})$ is the configuration that follows the configuration $(q', d'_{l,1}d'_{l,2}\ldots d'_{l,k'_l-1}d', d'_{r,k'_r-1}d'_{r,k'_r-2}\ldots d'_{r,1})$. It means that there is a run from the initial configuration $(\hat{q}, \triangleright, \varepsilon)$ to a configuration accordant with the lemma, and the first symbol in the tape is preserved.

Next, if α is (a_i, lft, pop, d), where $i \in \{1, 2, \ldots, n\}$ and $d \in D$, then d' must be $\leftarrow$, i must be $k'_l - 1$ and hence k'_l must be greater than one, and d must be d'_{l,k'_l-1}. It implies that if we set $k_l = k'_l - 1$, $d_{l,k_l} = -$, and $d_{l,i} = d'_{l,i}$ for all $i \in \{1, 2, \ldots, n\}$ different from k_l, then the component of s corresponding to

- the state of $[\![Ctrl_{\mathcal{M}}]\!]_{\phi}$ is (q', d),
- the state of $[\![Tape_l]\!]_{\phi_{i_1,i_2}}$ is (d_{l,i_1}, d_{l,i_2}) for all $1 \leq i_1 < i_2 \leq n$, and

otherwise s is like s'. Hence, by choosing $q = q'$, $k_r = k'_r$ and $d_{r,i} = d'_{r,i}$ for every $i \in \{1, 2, \ldots, n\}$, the state s satisfies the claim on the states of component LTSs. Moreover, because the configuration $(q, d_{l,1}d_{l,2}\ldots d_{l,k_l-1}d, d_{r,k_r-1}d_{r,k_r-2}\ldots d_{r,1})$ is the same as $(q', d'_{l,1}d'_{l,2}\ldots d'_{l,k'_l-1}, d'_{r,k'_r-1}d'_{r,k'_r-2}\ldots d'_{r,1})$, there is a run of from the initial configuration to it. Moreover, the configuration is accordant with the lemma and the first symbol in the tape is still $\triangleright$.

As the fifth case, we assume that α is (a_i, lft, psh, d''), where $i \in \{1, 2, \ldots, n\}$ and $d'' \in D$, then i must be k'_l and hence k'_l must be at most n. It implies that if we set $k_l = k'_l + 1$, $d_{l,k_l-1} = d''$, and $d_{l,i} = d'_{l,i}$ for all $i \in \{1, 2, \ldots, n\}$ different from $k_l - 1$, then the component of s corresponding to the state of $[\![Tape_l]\!]_{\phi_{i_1,i_2}}$ is (d_{l,i_1}, d_{l,i_2}) for all $1 \leq i_1 < i_2 \leq n$. Moreover, $\sigma(q', d')$ must be $(q, d'', \rightarrow)$, where $q \in Q$ such that the component of s corresponding to the state of $[\![Ctrl_{\mathcal{M}}]\!]_{\phi}$ is $(q, \rightarrow)$. Obviously, s is otherwise like s', which means that by choosing $k_r = k'_r$ and $d_{r,i} = d'_{r,i}$ for every $i \in \{1, 2, \ldots, n\}$, the state s satisfies the claim on the states of component LTSs. If k_r is greater than one, then the configuration $(q, d_{l,1}d_{l,2}\ldots d_{l,k_l-1}d_{r,k_r-1}, d_{r,k_r-2}d_{r,k_r-3}\ldots d_{r,1})$ follows the configuration $(q', d'_{l,1}d'_{l,2}\ldots d'_{l,k'_l-1}d', d'_{r,k'_r-1}d'_{r,k'_r-2}\ldots d'_{r,1})$, which implies that there is a run from the initial configuration to a configuration accordant with the lemma. On the other hand, if $k_r = 1$, then the configuration $(q, d_{l,1}d_{l,2}\ldots d_{l,k_l-1}\sqcup, \varepsilon)$ follows the configuration $(q', d'_{l,1}d'_{l,2}\ldots d'_{l,k'_l-1}d', d'_{r,k'_r-1}d'_{r,k'_r-2}\ldots d'_{r,1})$, which also means that there is a run from the initial configuration to a configuration accordant with the lemma. To

complete the case, note that in both the cases the first symbol in the tape is preserved even if $k_l = 1$.

Finally, if α is (a_i, rgh, pop, d), where $i \in \{1,2,\ldots,n\}$ and $d \in D$, then d' must be $\rightarrow$, and there are two cases to consider. If k'_r is greater than one, then i must be $k'_r - 1$ and d must be d'_{r,k'_r-1}. It implies that if we set $k_r = k'_r - 1$, $d_{r,k_r} = -$, and $d_{r,i} = d'_{r,i}$ for all $i \in \{1,2,\ldots,n\}$ different from k_r, then the component of s corresponding to

- the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q', d),
- the state of $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ is (d_{r,i_1}, d_{r,i_2}) for all $1 \leq i_1 < i_2 \leq n$, and

otherwise s is like s'. Hence, by choosing $q = q'$, $k_l = k'_l$ and $d_{l,i} = d'_{l,i}$ for every $i \in \{1,2,\ldots,n\}$, the state s satisfies the claim on the states of component LTSs. It also means that the configuration $(q, d_{l,1}d_{l,2}\ldots d_{l,k_l-1}d, d_{r,k_r}d_{r,k_r-2}\ldots d_{r,1})$ is the same as $(q', d'_{l,1}d'_{l,2}\ldots d'_{l,k'_l-1}d'_{r,k'_r-1}, d'_{r,k'_r-2}d'_{r,k'_r-3}\ldots d'_{r,1})$, which implies that there is a run from the initial configuration to it. Moreover, the configuration is accordant with the lemma and the first symbol in the tape is still $\triangleright$.

On the other hand, if $k'_r = 1$, then $i = 1$ and $d = \sqcup$. It implies that if we set $k_r = 1$ and $d_{r,i} = -$ for all $i \in \{1,2,\ldots,n\}$, then the component of s corresponding to

- the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q', d),
- the state of $[\![Tape_r]\!]_{\phi_{i_1,i_2}}$ is (d_{r,i_1}, d_{r,i_2}) for all $1 \leq i_1 < i_2 \leq n$, and

otherwise s is like s'. Hence, by choosing $q = q'$, $k_l = k'_l$ and $d_{l,i} = d'_{l,i}$ for every $i \in \{1,2,\ldots,n\}$, the state s satisfies the claim on the states of component LTSs. It also means that the configuration $(q, d_{l,1}d_{l,2}\ldots d_{l,k_l-1}d, d_{r,k_r}d_{r,k_r-2}\ldots d_{r,1})$ is the same as $(q', d'_{l,1}d'_{l,2}\ldots d'_{l,k'_l-1}\sqcup, \varepsilon)$, which implies that there is a run from the initial configuration to it. Moreover, the configuration is accordant with the lemma and the first symbol in the tape is $\triangleright$. Therefore, also the induction step is complete, and by the induction principle, the lemma is correct. □

The simulation lemmas above imply that $\mathcal{M}$ can halt with the empty input if and only if there is a valuation $\phi \in \Phi'_{DTM}$ compatible with $DTM''_{\mathcal{M}}$ such that $[\![DTM''_{\mathcal{M}}]\!]_\phi$ can execute the halting action. It means that the halting problem can be converted to a parameterised verification task concerning the system $DTM''_{\mathcal{M}}$ and the set Φ'_{DTM} of valuations.

To express it in our extended formalism, we need to specify the verification task using extended LTS schemata and an existential-free valuation formula. As a specification, we take an LTS schema $\|_{(c_1,c_2)\in\Xi:T\times T} NoHtl'$, where $NoHlt'$ is a single-state

elementary LTS schema with the singleton alphabet schema $\{halt\}$ and no transition schema. This specification enables halting to be detected as a violation of correctness, provided the stack size is at least two. Because the actions represented by

$$IO = \bigcup_{c \in T} \{(c, rgh, push, d), (c, rgh, pop, d), (c, lft, push, d), (c, lft, pop, d) \mid d \in D\}$$

are irrelevant to the specification, we take an extended LTS schema $DTM''_{\mathcal{M}} \setminus IO$ as a system. Finally, because $\phi(\Xi)$ is a proper total order over $\phi(T)$ for every $\phi \in \Phi'_{DTM}$, the set Φ'_{DTM} of valuations generating the system instances can be expressed as an existential-free valuation formula

$$\begin{aligned} c_{ord} = {} & (\forall x \in T : \neg(x,x) \in \Xi) \wedge \\ & (\forall x_1, x_2 \in T : x_1 \neq x_2 \rightarrow ((x_1,x_2) \in \Xi \vee (x_2,x_1) \in \Xi)) \wedge \\ & (\forall x_1, x_2, x_3 \in T : (((x_1,x_2) \in \Xi \wedge (x_2,x_3) \in \Xi) \rightarrow (x_1,x_3) \in \Xi)) \,. \end{aligned}$$

Theorem 56. *A DTM $\mathcal{M}$ halts with the empty input if and only if*

$$[\![\mathop{\Big\|}_{(c_1,c_2) \in \Xi : T \times T} NoHlt']\!]_\phi \succeq_{\mathrm{tr}} [\![DTM''_{\mathcal{M}} \setminus IO]\!]_\phi$$

does not hold for some compatible valuation $\phi \in \mathrm{va}(c_{ord})$.

Proof. First, suppose that $\mathcal{M}$ halts with the empty input. Then, there is a run ρ of $\mathcal{M}$ from the configuration $(\hat{q}, \triangleright, \varepsilon)$ to a configuration $(q, d_0 d_1 \dots d_k, d_{k+1} d_{k+2} \dots d_{n-1})$, where n is the scope of ρ, $\hat{q}$ is the initial state of $\mathcal{M}$ and $q \in \{halt, yes, no\}$. By Lemma 54, there is a valuation $\phi \in \Phi'_{DTM} = \mathrm{va}(c_{ord})$ compatible with $DTM''_{\mathcal{M}}$, and a state s reachable in $[\![DTM''_{\mathcal{M}}]\!]_\phi$ by an execution π such that the component of s corresponding to the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q, d_k). Then also π *halt* s is an execution of $[\![DTM''_{\mathcal{M}}]\!]_\phi$, which means that $[\![DTM''_{\mathcal{M}} \setminus IO]\!]_\phi$ has the trace *halt*. Because ϕ is also compatible with $\|_{(c_1,c_2) \in \Xi : T \times T} NoHlt'$, and the only trace of $[\![\|_{(c_1,c_2) \in \Xi : T \times T} NoHlt']\!]_\phi$ is the empty one, it implies that

$$[\![\mathop{\Big\|}_{(c_1,c_2) \in \Xi : T \times T} NoHlt']\!]_\phi \succeq_{\mathrm{tr}} [\![DTM''_{\mathcal{M}} \setminus IO]\!]_\phi \tag{11}$$

does not hold.

Next, we assume that there is a compatible valuation $\phi \in \mathrm{va}(c_{ord}) = \Phi'_{DTM}$ such that Relation 11 does not hold. Because the alphabets of $[\![\|_{(c_1,c_2) \in \Xi : T \times T} NoHlt']\!]_\phi$ and $[\![DTM''_{\mathcal{M}} \setminus IO]\!]_\phi$ match and the former LTS has only the empty trace, the latter LTS must have the trace *halt*. It implies that there is an execution π of $[\![DTM''_{\mathcal{M}}]\!]_\phi$ to a

state s such that the component of s corresponding to the state of $[\![Ctrl_{\mathcal{M}}]\!]_\phi$ is (q,d), where $q \in \{halt, yes, no\}$ and $d \in D$. By Lemma 55, there is a run of $\mathcal{M}$ from the configuration $(\hat{q}, \triangleright, \varepsilon)$ to a configuration (q, w_1, w_2), which means that $\mathcal{M}$ halts with the empty input. □

The theorem leads immediately to an undecidability result for systems expressed as extended LTS schemata. Hence, extending the cut-off result and the reduction algorithm to systems with parametric branching and joining is impossible without imposing additional restrictions on specification LTS schemata and valuation formulae.

Corollary 57. *Let $\mathcal{Q}$ and $\mathcal{P}$ be finite and closed LTS schemata and c an existential-free valuation formula such that $\mathcal{Q}$ has no hiding LTS subschema and $\mathcal{P}$ may have extended elementary LTS subschemata. The question whether $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$ for all compatible $\phi \in \mathrm{va}(c)$ is undecidable.*

8.6 Undecidability with Systems Involving Renaming

Let us then consider another natural extension to our system model, namely the use of parametric renaming. Also this structure can be expressed with the aid of action set schemata.

A set of pairs, where the first component is an action set schema and the second an action schema, is a *function schema* denoted by a Calligraphic letter $\mathcal{F}$ and its variants. If $\mathcal{F}$ is a function schema and $\mathcal{P}$ an LTS schema, then $\mathcal{F}(\mathcal{P})$ is a *(renaming) LTS schema*. An *LTS schema with renaming* is an LTS schema where renaming LTS schemata are possibly used as LTS subschemata. The concepts and notation, like the signature, finiteness and compatibility, are defined for LTS schemata with renaming analogously to LTS schemata.

If $\mathcal{F}$ is a function schema and ϕ a valuation defined for all the atom and type variables that occur in $\mathcal{F}$, then $[\![\mathcal{F}]\!]_\phi$ denotes a function: $\mathbb{V} \mapsto \mathbb{V}$ defined for all actions α as follows. If $\alpha = (b_1, b_2, \ldots, b_k, a_1, a_2, \ldots, a_l)$ and there is a pair

$$((\sharp_1, \sharp_2, \ldots, \sharp_k, a_1, a_2, \ldots, a_l), (x_1, x_2, \ldots, x_{k'}, a'_1, a'_2, \ldots, a'_{l'})) \in \mathcal{F}$$

such that $b_i = \phi(\sharp_i)$ whenever $i \in \{1, 2, \ldots, k\}$ and $\sharp_i \in \mathbb{X}$, and $b_i \in \phi(\sharp_i)$ whenever $i \in \{1, 2, \ldots, k\}$ and $\sharp_i \in \mathbb{T}$, then

$$\zeta(\alpha) = (\phi(x_1), \phi(x_2), \ldots, \phi(x_{k'}), a'_1, a'_2, \ldots, a'_{l'}) .$$

Otherwise ζ maps α to itself.

The instance of an LTS schema $\mathcal{P}$ with renaming generated by a compatible valuation ϕ is obtained in an obvious way as follows. If $\mathcal{P}$ is a parallel, replicated parallel, or hiding LTS schema, then $[\![\mathcal{P}]\!]_\phi$ is defined as in the case of LTS schemata. If $\mathcal{F}(\mathcal{P})$ is a renaming LTS schema and ϕ a compatible valuation, then $[\![\mathcal{F}(\mathcal{P})]\!]_\phi$ is $[\![\mathcal{F}]\!]_\phi([\![\mathcal{P}]\!]_\phi)$. Hence, the instance of an LTS schema with renaming is obviously an LTS.

The proof for undecidability for systems expressed as LTS schemata with renaming is very similar to that of systems with parametric branching. All we have to do is to modify the model $DTM''_{\mathcal{M}}$ of a DTM $\mathcal{M}$ in such a way that the use of action set schemata in extended elementary LTS schemata is replaced by renaming LTS schemata.

Recall that the use of parametric branching enabled us to pick the first unused location and the last non-empty one for pushing data into and popping it from a stack. The same effect can be achieved by renaming all pushing and popping actions of a stack to single actions, which can be done by applying a function schema

$$\mathcal{F}_l := \{((T, \mathit{lft}, \mathit{psh}, d), (\mathit{lft}, \mathit{psh}, d)), ((T, \mathit{lft}, \mathit{pop}, d), (\mathit{lft}, \mathit{pop}, d)) \mid d \in D\}$$

to the model of the left stack, and a renaming schema $\mathcal{F}_r$ obtained from $\mathcal{F}_l$ by substituting *rgh* to every occurrence of *lft* to the model of the right stack. Now, the control part can be expressed simply as an elementary LTS schema $\mathit{Ctrl}'_{\mathcal{M}}$ obtained from $\mathit{Ctrl}_{\mathcal{M}}$ by dropping all the occurrences of type variables. Hence, a DTM $\mathcal{M}$ can be now expressed as an LTS schema

$$DTM'''_{\mathcal{M}} := (\mathcal{F}_l(\|_{(c_1,c_2)\in\Xi:T\times T} \mathit{Tape}_l)) \parallel \mathit{Ctrl}'_{\mathcal{M}} \parallel (\mathcal{F}_r(\|_{(c_1,c_2)\in\Xi:T\times T} \mathit{Tape}_r)) .$$

with renaming.

It is obvious that results analogous to Lemmas 54 and 55 can be proved for $DTM'''_{\mathcal{M}}$, too. Therefore, the halting problem can be expressed also in the formalism extended by a renaming LTS schema without using hiding in a specification and existential quantification in a valuation formula.

Obviously, the same specification as earlier, NoHlt' namely, can be used to detect halting. It means that the actions in the set

$$IO' = \{(\mathit{rgh}, \mathit{push}, d), (\mathit{rgh}, \mathit{pop}, d), (\mathit{lft}, \mathit{push}, d), (\mathit{lft}, \mathit{pop}, d) \mid d \in D\}$$

have to be hidden in our DTM model. Hence, the system is an LTS schema $DTM'''_{\mathcal{M}} \setminus IO'$ with renaming, and the existential-free valuation formula c_{ord} can be used to generate its instance here as well.

Theorem 58. *A DTM $\mathcal{M}$ halts with the empty input if and only if*

$$[\![\mathop{\Big\|}_{(c_1,c_2)\in\Xi:T\times T} NoHlt']\!]_\phi \succeq_{\mathrm{tr}} [\![DTM'''_{\mathcal{M}} \setminus IO']\!]_\phi$$

does not hold for some compatible valuation $\phi \in \mathrm{va}(c_{ord})$.

The proof is analogous to that of Theorem 56.

The result means that also the use of renaming LTS schemata makes the problem undecidable. Hence, extending the cut-off result and the reduction algorithm to significantly stronger systems seems unlikely without additional restrictions on specification LTS schemata or valuation formulae representing the sets of valuations.

Corollary 59. *Let $\mathcal{Q}$ be a finite and closed LTS schema without a hiding LTS subschema, $\mathcal{P}$ a finite and closed LTS schema with renaming, and c an existential-free valuation formula. The question whether $[\![\mathcal{Q}]\!]_\phi \succeq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$ for all compatible $\phi \in \mathrm{va}(c)$ is undecidable.*

To summarise, we have considered several natural and significant extensions to our theory and showed that they all lead to undecidability. Therefore, our result is very close to the decidability frontier and hence cannot be significantly extended in any direction without simultaneously restricting it from some other aspect.

9 Conclusions

In this thesis, we have presented a novel approach, called the precongruence reduction, to cut down the number of individual verification tasks needed to solve a parameterised refinement checking task for labelled transition systems. In the best case, only a finite number of finite-state verification tasks remains, which implies that the problem can then be solved using existing tools.

We have also provided an algorithm that automates the precongruence reduction. A distinctive feature of the algorithm is that its restrictions are explicit, it always gives the correct answer, and it can handle multiply parameterised systems with a parameterised substructure. Informally, the algorithm applies to safety-property-system families that are closed under the removal of a replicated component, which covers for example systems with a star, bipartite and totally (un)connected topology, and the transitive closures of rings, arrays and forests.

Finally, it is shown that the result behind the algorithm is very close to the decidability frontier. We have considered several obvious and significant extensions to our result and formalism, and shown that they all make the problem undecidable.

An obvious topic for future research is implementing the algorithm and analysing its practical performance. From the theoretical point of view, an obvious topic is to find as weak conditions as possible under which one can allow rings and linear systems to be analysed, the use of hiding in specifications, the analysis of deadlocks and liveness properties, and parametric branching within elementary LTS schemata.

We conjecture that by allowing only inequality and full relations in the specification side, it is possible to use hiding in specifications as well, but with the cost of doubling the bounds for the sizes of type variable values. We also conjecture that forbidding the use of other kinds of relations altogether and requiring the system and specification to be deterministic, we can analyse deadlocks and liveness properties as well, but again with the cost of worse reduction. On the other hand, as already mentioned in Section 8.4, the extension to livelocks only looks straightforward and seems to come without a price.

Systems with a ring, linear, and tree topology are likely to be more difficult to handle. However, with the aid of inductive reasoning, it seems to be possible to cover them in some extent (Siirtola 2010). We conjecture that if any two components (instances of

elementary LTS schemata) have at most one visible action in common, then the system can have any topology. This corresponds to the situation considered by Clarke *et al.* (2004), where processes communicate with (stateless) token passing.

Another interesting scenario worth investigating is that considered by Emerson & Kahlon (2004), where processes communicate with data carrying tokens that can change their state only a bounded number of times. However, we do not currently know how to characterise such systems in a sophisticated way in process algebraic terms. Nevertheless, in order to generate reduced instances automatically, a new inductive formalism for expressing the sets of valuations is very likely needed in both the cases.

Treating with systems involving parametric branching appears to be the most difficult one among the proposed extensions, because the idea of the precongruence reduction does not seem to be applicable then. Moreover, with the aid of this structure, it is extremely easy to order components in such a way that a DTM can be simulated, and the only way to prevent it from happening appears to be forbidding the use of nested parameters. Currently, we do not know how to handle such systems, but converting them first to token passing ones and then trying to apply the related results could be a way to proceed.

References

Abdulla PA, Delzanno G, Henda NB & Rezine A (2007) Regular model checking without transducers (On efficient verification of parameterized systems). In: Grumberg O & Huth M (eds) Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 – April 1, 2007, Proceedings, volume 4424 of *Lecture Notes in Computer Science*, 721–736. Springer-Verlag.

Abdulla PA, Jonsson B, Nilsson M, d'Orso J & Saksena M (2004) Regular model checking for LTL(MSO). In: Alur R & Peled D (eds) Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13–17, 2004, Proceedings, volume 3114 of *Lecture Notes in Computer Science*, 348–360. Springer-Verlag.

Abdulla PA, Legay A, d'Orso J & Rezine A (2006) Tree regular model checking: A simulation-based approach. Journal of Logic and Algebraic Programming 69(1-2): 93–121.

Aggarwal S, Kurshan RP & Sabnani KK (1983) A calculus for protocol specification and validation. Proc. Rudin H & West CH (eds) Protocol Specification, Testing, and Verification, III, Proceedings of the IFIP WG 6.1 Third International Workshop on Protocol Specification, Testing and Verification, organized by IBM Research, Rüschlikon, Switzerland, 31 May – 2 June, 1983, North-Holland, 19–34.

Alpern B & Schneider FB (1985) Defining liveness. Information Processing Letters 21(4): 181–185.

Apt KR & Kozen DC (1986) Limits for automatic verification of finite-state concurrent systems. Information Processing Letters 22(6): 307–309.

Baeten JCM (2005) A brief history of process algebra. Theoretical Computer Science 335(2–3): 131–146.

Bingham JD & Hu AJ (2005) Empirically efficient verification for a class of infinite-state systems. In: Halbwachs N & Zuck LD (eds) Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4–8, 2005, Proceedings, volume 3440 of *Lecture Notes in Computer Science*, 77–92. Springer-Verlag.

Bolognesi T & Brinksma E (1987) Introduction to the ISO specification language LOTOS. Computer Networks and ISDN Systems 14(1): 25–59.

Bouajjani A, Habermehl P & Vojnar T (2008) Verification of parametric concurrent systems with prioritised FIFO resource management. Formal Methods in System Design 32(2): 129–172.

Chan W, Anderson RJ, Beame P, Burns S, Modugno F, Notkin D & Reese JD (1998) Model checking large software specifications. IEEE Transactions on Software Engineering 24(7): 498–520.

Clarke E, Talupur M, Touili T & Veith H (2004) Verification by network decomposition. In: Gardner, P and Yoshida, N (ed) CONCUR 2004 — Concurrency Theory, 15th International Conference, London, UK, August 31 – September 3, 2004, Proceedings, volume 3170 of *Lecture Notes in Computer Science*, 276–291. Springer-Verlag.

Clarke EM, Enders R, Filkorn T & Jha S (1996) Exploiting symmetry in temporal logic model checking. Formal Methods in System Design 9(1-2): 77–104.

Clarke EM, Grumberg O & Jha S (1997) Verifying parameterized networks. ACM Transactions on Programming Languages and Systems 19(5): 726–750.

Clarke EM, Grumberg O & Peled DA (1999) Model Checking. The MIT Press.

Clarke EM, Talupur M & Veith H (2006) Environment abstraction for parameterized verification. In: Emerson EA & Namjoshi KS (eds) Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8–10, 2006, Proceedings, volume 3855 of *Lecture Notes in Computer Science*, 126–141. Springer-Verlag.

Clarke EM, Talupur M & Veith H (2008) Proving ptolemy right: The environment abstraction framework for model checking concurrent systems. In: Ramakrishnan CR & Rehof J (eds) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29–April 6, 2008. Proceedings, volume 4963 of *Lecture Notes in Computer Science*, 33–47. Springer-Verlag.

Cousot P & Cousot R (1977) Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. Proc. POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM Press, New York, NY, Los Angeles, California, 238–252.

Creese SJ (2001) Data independent induction: CSP model checking of arbitrary sized networks. Ph.D. thesis, Oxford University.

Delzanno G (2003) Constraint-based verification of parameterized cache coherence protocols. Formal Methods in System Design 23(3): 257–301.

Delzanno G, Raskin JF & Begin LV (2002) Towards the automated verification of multithreaded Java programs. In: Katoen JP & Stevens P (eds) Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8–12, 2002, Proceedings, volume 2280 of *Lecture Notes in Computer Science*, 173–187. Springer-Verlag, London, UK.

Emerson EA & Kahlon V (2000) Reducing model checking of the many to the few. In: McAllester DA (ed) Automated Deduction — CADE-17, 17th International Conference on Automated Deduction, Pittsburgh, PA, USA, June 17–20, 2000, Proceedings, volume 1831 of *Lecture Notes in Computer Science*, 236–254. Springer-Verlag.

Emerson EA & Kahlon V (2002) Model checking large-scale and parameterized resource allocation systems. In: Katoen JP & Stevens P (eds) Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8–12, 2002, Proceedings, volume 2280 of *Lecture Notes in Computer Science*, 251–265. Springer-Verlag.

Emerson EA & Kahlon V (2003a) Exact and efficient verification of parameterized cache coherence protocols. In: Geist D & Tronci E (eds) Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, L'Aquila, Italy, October 21–24, 2003, Proceedings, volume 2860 of *Lecture Notes in Computer Science*, 247–262. Springer-Verlag.

Emerson EA & Kahlon V (2003b) Model checking guarded protocols. Proc. 18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22–25 June 2003, Ottawa, Canada, Proceedings, IEEE Computer Society, 361–370.

Emerson EA & Kahlon V (2004) Parameterized model checking of ring-based message passing systems. In: Marcinkowski J & Tarlecki A (eds) Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20–24, 2004, Proceedings, volume 3210 of *Lecture Notes in Computer Science*, 325–339. Springer-Verlag.

Emerson EA & Namjoshi KS (1996) Automatic verification of parameterized synchronous systems (extended abstract). In: Alur R & Henzinger TA (eds) Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 – August 3, 1996, Proceedings, volume 1102 of *Lecture Notes in Computer Science*, 87–98. Springer-Verlag.

Emerson EA & Namjoshi KS (2003) On reasoning about rings. International Journal of Foundations of Computer Science 14(4): 527–550.

Emerson EA & Sistla AP (1996) Symmetry and model checking. Formal Methods in System Design 9(1-2): 105–131.

Faber J & Meyer R (2006) Model checking data-dependent real-time properties of the European Train Control System. Proc. Formal Methods in Computer-Aided Design, 6th International Conference, FMCAD 2006, San Jose, California, USA, November 12-16, 2006, Proceedings, IEEE Computer Society, Washington, DC, USA, 76–77.

German SM & Sistla AP (1992) Reasoning about systems with many processes. Journal of ACM 39(3): 675–735.

Graf S & Saïdi H (1997) Construction of abstract state graphs with PVS. In: Grumberg O (ed) Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings, volume 1254 of *Lecture Notes in Computer Science*, 72–83. Springer-Verlag, London, UK.

Grinchtein O, Leucker M & Piterman N (2006) Inferring network invariants automatically. In: Furbach U & Shankar N (eds) Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17–20, 2006, Proceedings, volume 4130 of *Lecture Notes in Computer Science*, 483–497. Springer-Verlag.

Haustein M & Härder T (2008) Optimizing lock protocols for native XML processing. Data & Knowledge Engineering 65(1): 147–173.

Havelund K, Lowry M & Penix J (2001) Formal analysis of a space-craft controller using SPIN. IEEE Transactions on Software Engineering 27(8): 749–765.

Hoare CAR (1985) Communicating Sequential Processes. Prentice-Hall, Inc.

Holzmann GJ (2003) The SPIN Model Checker : Primer and Reference Manual. Addison-Wesley Professional.

Ip CN & Dill DL (1996) Better verification through symmetry. Formal Methods in System Design 9(1-2): 41–75.

Ip CN & Dill DL (1999) Verifying systems with replicated components in Murφ. Formal Methods in System Design 14(3): 273–310.

Kaski P & Östergård PRJ (2006) Classification Algorithms for Codes and Designs. Number 15 in Algorithms and Computation in Mathematics. Springer-Verlag, Berlin Heidelberg.

Kesten Y, Maler O, Marcus M, Pnueli A & Shahar E (2001) Symbolic model checking with rich assertional languages. Theoretical Compututer Science 256(1-2): 93–112.

Konnov IV & Zakharov VA (2005) An approach to the verification of symmetric parameterized distributed systems. Programming and Computer Software 31(5): 225–236.

Kurshan RP & McMillan KL (1995) A structural induction theorem for processes. Information and Compututing 117(1): 1–11.

Kurshan RP, Merritt M, Orda A & Sachs SR (1994) A structural linearization principle for processes. Formal Methods in System Design 5(3): 227–244.

Lahiri SK & Bryant RE (2004) Indexed predicate discovery for unbounded system verification. In: Alur R & Peled D (eds) Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13–17, 2004, Proceedings, volume 3114 of *Lecture Notes in Computer Science*, 135–147. Springer-Verlag.

Lahiri SK, Bryant RE & Cook B (2003) A symbolic approach to predicate abstraction. In: Jr WAH & Somenzi F (eds) Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8–12, 2003, Proceedings, volume 2725 of *Lecture Notes in Computer Science*, 141–153. Springer-Verlag.

Lazić RS (1999) A semantic study of data independence with applications to model checking. Ph.D. thesis, Oxford University.

Lazić RS & Nowak D (2000) A unifying approach to data-independence. In: Palamidessi C (ed) CONCUR 2000 — Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22–25, 2000, Proceedings, volume 1877 of *Lecture Notes in Computer Science*, 581–595. Springer-Verlag.

Lesens D, Halbwachs N & Raymond P (2001) Automatic verification of parameterized networks of processes. Theoretical Compututer Science 256(1-2): 113–144.

Li J, Suzuki I & Yamashita M (1994) A new structural induction theorem for rings of temporal Petri nets. IEEE Transactions on Software Engineering 20(2): 115–126.

Lowe G (1996) Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Margaria T & Steffen B (eds) Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings, volume 1055 of *Lecture Notes in Computer Science*, 147–166. Springer-Verlag.

Lubachevsky BD (1984) An approach to automating the verification of compact parallel coordination programs I. Acta Informatica 21: 125–169.

McKay BD (2007) nauty User's Guide (Version 2.4). Department of Computer Science, Australian National University.

Milner R (1989) Communication and Concurrency. Prentice-Hall, Inc.

Nazari S & Thistle J (2007) Structural conditions for model-checking of parameterized networks. Proc. Basten T, Juhás G & Shukla SK (eds) Seventh International Conference on Application of Concurrency to System Design (ACSD 2007), 10–13 July 2007, Bratislava, IEEE Computer Society, 187–196.

Papadimitriou CM (1994) Computational Complexity. Addison-Wesley, Reading, Massachusetts.

Puhakka A & Valmari A (1999) Weakest-congruence results for livelock-preserving equivalences. In: Baeten JCM & Mauw S (eds) CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24–27, 1999, Proceedings, volume 1664 of *Lecture Notes in Computer Science*, 510–524. Springer-Verlag, London, UK.

Pyssysalo T (1996) An induction theorem for ring protocols of processes described with predicate/transition nets. Research Report A37, Helsinki University of Technology.

Ramakrishnan R & Gehrke J (2002) Database Management Systems. McGraw-Hill.

Roscoe AW (1997) The Theory and Practice of Concurrency. Prentice Hall.

Roscoe AW, Gardiner PHB, Goldsmith MH, Hulance JR, Jackson DM & Scattergood JB (1995) Hierarchical compression for model-checking CSP or how to check 10^{20} dining philosophers for deadlock. In: Brinksma E, Cleaveland R, Larsen KG, Margaria T & Steffen B (eds) Tools and Algorithms for Construction and Analysis of Systems, First International

Workshop, TACAS '95, Aarhus, Denmark, May 19–20, 1995, Proceedings, volume 1019 of *Lecture Notes in Computer Science*, 133–152. Springer-Verlag.

Shtadler Z & Grumberg O (1990) Network grammars, communication behaviors and automatic verification. In: Sifakis J (ed) Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12–14, 1989, Proceedings, volume 407 of *Lecture Notes in Computer Science*, 151–165. Springer-Verlag, New York.

Siirtola A (2010) Cut-offs with network invariants. Proc. 10th International Conference on Application of Concurrency to System Design (ACSD 2010), June 23–25, 2010, Braga, Portugal, IEEE Computer Society.

Siirtola A & Kortelainen J (2009a) Algorithmic verification with multiple and nested parameters. In: Cavalcanti A & Breitman K (eds) Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brazil, December 8–11, 2009, Proceedings, volume 5885 of *Lecture Notes in Computer Science*, 561–580. Springer-Verlag.

Siirtola A & Kortelainen J (2009b) Parameterised process algebraic verification by precongruence reduction. Proc. Edwards S, Lorenz R & Vogler W (eds) 9th International Conference on Application of Concurrency to System Design (ACSD 2009), July 1–3, 2009, Augsburg, Germany, IEEE Computer Society, 158–167.

Siirtola A & Valenta M (2008) Verifying parameterized taDOM+ lock managers. In: Geffert V, Karhumäki J, Bertoni A, Preneel B, Návrat P & Bieliková M (eds) SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 19–25, 2008, Proceedings, volume 4910 of *Lecture Notes in Computer Science*, 460–472. Springer-Verlag.

Suzuki I (1988) Proving properties of a ring of finite-state machines. Information Processing Letters 28(4): 213–214.

Valmari A (1995) The weakest deadlock-preserving congruence. Information Processing Letters 53(6): 341–346.

Valmari A (2000) Composition and abstraction. In: Cassez F, Jard C, Rozoy B & Ryan MD (eds) Modeling and Verification of Parallel Processes, 4th Summer School, MOVEP 2000, Nantes, France, June 19-23, 2000, volume 2067 of *Lecture Notes in Computer Science*, 58–98. Springer-Verlag.

Valmari A & Kokkarinen I (1998) Unbounded verification results by finite-state compositional techniques: 10^{any} states and beyond. Proc. 1st International Conference on Application of Concurrency to System Design (ACSD '98), 23–26 March 1998, Fukushima, Japan, IEEE Computer Society, 75–85.

Valmari A & Tienari M (1991) An improved failures equivalence for finite-state systems with a reduction algorithm. Proc. Jonsson B, Parrow J & Pehrson B (eds) Protocol Specification, Testing and Verification XI, Proceedings of the IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Stockholm, Sweden, 18–20 June 1991, North-Holland, 3–18.

Wachter B & Westphal B (2007) The spotlight principle. In: Cook B & Podelski A (eds) Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14–16, 2007, Proceedings, volume 4349 of *Lecture Notes in Computer Science*, 182–198. Springer-Verlag.

Wolper P (1986) Expressing interesting properties of programs in propositional temporal logic. Proc. POPL '86: Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Princi-

ples of programming languages, ACM, 184–193.
Wolper P & Lovinfosse V (1990) Verifying properties of large sets of processes with network invariants. In: Sifakis J (ed) Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12–14, 1989, Proceedings, volume 407 of *Lecture Notes in Computer Science*, 68–80. Springer-Verlag, New York.
Yahav E (2001) Verifying safety properties of concurrent Java programs using 3-valued logic. SIGPLAN Notices 36(3): 27–40.

Appendix 1 Proofs of Chapter 3

Lemma 3. To prove the first claim, we first claim that whenever $s_1\alpha_1 s_2 \dots \alpha_{n-1} s_n$ is an execution of $\mathcal{L}_1 \parallel \mathcal{L}_2$, then there is an execution $s_{i,1}\alpha_{i,1}s_{i,2}\dots\alpha_{i,n_i-1}s_{i,n_i}$ of $\mathcal{L}_i$ for both $i \in \{1,2\}$ such that $s_n = (s_{1,n_1}, s_{2,n_2})$ and $\alpha_1\alpha_2\cdots\alpha_{n-1}|_{\mathrm{alph}(\mathcal{L}_i)} = \alpha_{i,1}\alpha_{i,2}\cdots\alpha_{i,n_i-1} \setminus \{\tau\}$ for both $i \in \{1,2\}$. We argue by induction on n.

In the base step, n is one, which implies that s_1 is a pair $(\hat{s}_1, \hat{s}_2)$ such that $\hat{s}_i$ is the initial state of $\mathcal{L}_i$ for both $i \in \{1,2\}$. Obviously, $\hat{s}_1$ and $\hat{s}_2$ are executions of respectively $\mathcal{L}_1$ and $\mathcal{L}_2$ satisfying the assumptions. Hence, the base step is clear.

In the induction step, n is greater than one and there are four cases to consider. First, if $\alpha_{n-1} \in \mathrm{alph}(\mathcal{L}_1) \cap \mathrm{alph}(\mathcal{L}_2)$, then for both $i \in \{1,2\}$, there is a transition $(s_{i,n-1}, \alpha_{n-1}, s_{i,n})$ of $\mathcal{L}_i$ such that $s_{n-1} = (s_{1,n-1}, s_{2,n-1})$ and $s_n = (s_{1,n}, s_{2,n})$. By the induction hypothesis, for both $i \in \{1,2\}$, there is an execution π_i of $\mathcal{L}_i$ to the state $s_{i,n-1}$ such that $\alpha_1\alpha_2\cdots\alpha_{n-2}|_{\mathrm{alph}(\mathcal{L}_i)}$ is the sequence obtained from π_i by removing all the states and the invisible actions. It means that for both $i \in \{1,2\}$, $\pi_i\alpha_{n-1}s_{i,n}$ is an execution of $\mathcal{L}_i$ and $\alpha_1\alpha_2\cdots\alpha_{n-1}|_{\mathrm{alph}(\mathcal{L}_i)}$ is the sequence obtained from $\pi_i\alpha_{n-1}s_{i,n}$ by removing all the states and the invisible actions.

Next, we assume that $\alpha_{n-1} \in \mathrm{alph}(\mathcal{L}_1)$ but $\alpha_{n-1} \notin \mathrm{alph}(\mathcal{L}_2)$. Then, there is a transition $(s_{1,n-1}, \alpha_{n-1}, s_{1,n})$ of $\mathcal{L}_1$ and a state $s_{2,n-1}$ of $\mathcal{L}_2$ such that $s_{n-1} = (s_{1,n-1}, s_{2,n-1})$ and $s_n = (s_{1,n}, s_{2,n-1})$. By the induction hypothesis, for both $i \in \{1,2\}$, there is an execution π_i of $\mathcal{L}_i$ to the state $s_{i,n-1}$ such that $\alpha_1\alpha_2\cdots\alpha_{n-2}|_{\mathrm{alph}(\mathcal{L}_i)}$ is the sequence obtained from π_i by removing all the states and the invisible actions. It means that $\pi_1\alpha_{n-1}s_{i,n}$ and π_2 are executions of respectively $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\alpha_1\alpha_2\cdots\alpha_{n-1}|_{\mathrm{alph}(\mathcal{L}_1)}$ and $\alpha_1\alpha_2\cdots\alpha_{n-1}|_{\mathrm{alph}(\mathcal{L}_2)}$ are the respective sequences obtained from them by removing all the states and the invisible actions.

The case when $\alpha_{n-1} \in \mathrm{alph}(\mathcal{L}_2)$ but $\alpha_{n-1} \in \mathrm{alph}(\mathcal{L}_1)$ is symmetric to the previous one. Finally, the case when $\alpha_{n-1} = \tau$ reduces either of the two previous ones. Therefore, also the induction step is clear. By the induction principle, it implies that whenever t is a trace of $(\mathcal{L}_1 \parallel \mathcal{L}_2)$, then $t|_{\mathrm{alph}(\mathcal{L}_i)}$ is a trace of $\mathcal{L}_i$ for both $i \in \{1,2\}$.

To prove the opposite, we claim that whenever $s_{i,1}\alpha_{i,1}s_{i,2}\dots\alpha_{i,n_i-1}s_{i,n_i}$ is an execution of $\mathcal{L}_i$ for both $i \in \{1,2\}$, then for every sequence t over $\mathrm{alph}(\mathcal{L}_1) \cup \mathrm{alph}(\mathcal{L}_2)$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)} = \alpha_{i,1}\alpha_{i,2}\dots\alpha_{i,n_i-1} \setminus \{\tau\}$ for both $i \in \{1,2\}$, there is an execution

$s_1\alpha_1 s_2 \ldots \alpha_{n-1} s_n$ of $\mathcal{L}_1 \parallel \mathcal{L}_2$ such that $s_n = (s_{1,n_1}, s_{2,n_2})$ and $t = \alpha_1\alpha_2\cdots\alpha_{n-1} \setminus \{\tau\}$. We argue by (strong) induction on $n_1 + n_2$.

In the base step, both n_1 and n_2 are one. It implies that $s_{1,1}$ and $s_{2,1}$ are the initial states of respectively $\mathcal{L}_1$ and $\mathcal{L}_2$. If t is a sequence over $\mathrm{alph}(\mathcal{L}_1) \cup \mathrm{alph}(\mathcal{L}_2)$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)} = \varepsilon$ for both $i \in \{1,2\}$, then also t must be the empty sequence. Hence, $(s_{1,1}, s_{2,1})$ is an execution of $\mathcal{L}_1 \parallel \mathcal{L}_2$ accordant with the assumptions, and the base step is clear.

In the induction step, n_1 or n_2 is greater than one and there are six cases to consider. First, we assume that $n_1 > 1$ and α_{1,n_1-1} is the invisible action. If t is a sequence over $\mathrm{alph}(\mathcal{L}_1) \cup \mathrm{alph}(\mathcal{L}_2)$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)} = \alpha_{i,1}\alpha_{i,2}\ldots\alpha_{i,n_i-1} \setminus \{\tau\}$ for both $i \in \{1,2\}$, then $t|_{\mathrm{alph}(\mathcal{L}_1)} = \alpha_{1,1}\alpha_{1,2}\ldots\alpha_{1,n_1-2} \setminus \{\tau\}$. By the induction hypothesis, there is an execution π of $\mathcal{L}_1 \parallel \mathcal{L}_2$ to the state (s_{1,n_1-1}, s_{2,n_2}) such that t is the sequence obtained from π by removing all the states and the invisible actions. As $(s_{n_1-1}, \tau, s_{n_1})$ is a transition of $\mathcal{L}_1$, it means that $\pi\tau(s_{1,n_1}, s_{2,n_2})$ is an execution of $\mathcal{L}_1 \parallel \mathcal{L}_2$ such that t is the sequence obtained from it by removing all the states and the invisible actions. The case when $n_2 > 1$ and $\alpha_{2,n_2-1} = \tau$ is symmetric.

Thirdly, we assume that $n_1, n_2 > 1$ and $\alpha_{1,n_1-1} = \alpha_{2,n_2-1} \in \mathrm{alph}(\mathcal{L}_1) \cap \mathrm{alph}(\mathcal{L}_2)$. If t is a sequence over $\mathrm{alph}(\mathcal{L}_1) \cup \mathrm{alph}(\mathcal{L}_2)$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)} = \alpha_{i,1}\alpha_{i,2}\ldots\alpha_{i,n_i-1} \setminus \{\tau\}$ for both $i \in \{1,2\}$, then $t = t'\alpha_{1,n_1-1}$, and $t'|_{\mathrm{alph}(\mathcal{L}_i)} = \alpha_{i,1}\alpha_{i,2}\ldots\alpha_{i,n_i-2} \setminus \{\tau\}$ for both $i \in \{1,2\}$. By the induction hypothesis, it implies that there is an execution π of $\mathcal{L}_1 \parallel \mathcal{L}_2$ to the state $(s_{1,n_1-1}, s_{2,n_2-1})$ such that t' is the sequence obtained from π by removing all the states and the invisible actions. As $(s_{n_i-1}, \alpha_{n_i-1}, s_{n_i})$ is a transition of $\mathcal{L}_i$ for both $i \in \{1,2\}$, it means that $\pi\alpha_{1,n_1-1}(s_{1,n_1}, s_{2,n_2})$ is an execution of $\mathcal{L}_1 \parallel \mathcal{L}_2$ such that t is the sequence obtained from it by removing all the states and the invisible actions.

Next, let n_1 be greater than one, α_{1,n_1-1} an action in $\mathrm{alph}(\mathcal{L}_1) \setminus \mathrm{alph}(\mathcal{L}_2)$, and either $n_2 = 1$ or $\alpha_{2,n_2-1} \in \mathrm{alph}(\mathcal{L}_1) \cap \mathrm{alph}(\mathcal{L}_2)$. If t is a sequence over $\mathrm{alph}(\mathcal{L}_1) \cup \mathrm{alph}(\mathcal{L}_2)$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)} = \alpha_{i,1}\alpha_{i,2}\ldots\alpha_{i,n_i-1} \setminus \{\tau\}$ for both $i \in \{1,2\}$, then $t = t'\alpha_{1,n_1-1}$ such that $t'|_{\mathrm{alph}(\mathcal{L}_1)} = \alpha_{1,1}\alpha_{1,2}\ldots\alpha_{1,n_1-2} \setminus \{\tau\}$ and $t'|_{\mathrm{alph}(\mathcal{L}_2)} = \alpha_{2,1}\alpha_{2,2}\ldots\alpha_{2,n_2-1} \setminus \{\tau\}$. By the induction hypothesis, it implies that there is an execution π of $\mathcal{L}_1 \parallel \mathcal{L}_2$ to the state (s_{1,n_1-1}, s_{2,n_2}) such that t' is the sequence obtained from π by removing all the states and the invisible actions. As $(s_{n_1-1}, \alpha_{n_1-1}, s_{n_1})$ is a transition of $\mathcal{L}_1$ and $\alpha_{n_1-1} \notin \mathrm{alph}(\mathcal{L}_2)$, it means that $\pi\alpha_{1,n_1-1}(s_{1,n_1}, s_{2,n_2})$ is an execution of $\mathcal{L}_1 \parallel \mathcal{L}_2$ such that t is the sequence obtained from it by removing all the states and the invisible actions. Again, the case when n_2 is greater than one, α_{2,n_2-1} an action in $\mathrm{alph}(\mathcal{L}_2) \setminus \mathrm{alph}(\mathcal{L}_1)$, and either $n_1 = 1$ or $\alpha_{1,n_1-1} \in \mathrm{alph}(\mathcal{L}_2) \cap \mathrm{alph}(\mathcal{L}_1)$ is analogous.

Finally, let us assume that $n_i > 1$ and $\alpha_{i,n_i-1} \in \mathrm{alph}(\mathcal{L}_i) \setminus \mathrm{alph}(\mathcal{L}_{\bar{i}})$ for both $i \in \{1,2\}$, where $\bar{i}$ denotes an element in $\{1,2\}$ different from i. If t is a sequence over $\mathrm{alph}(\mathcal{L}_1) \cup \mathrm{alph}(\mathcal{L}_2)$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)} = \alpha_{i,1}\alpha_{i,2}\ldots\alpha_{i,n_i-1} \setminus \{\tau\}$ for both $i \in \{1,2\}$, then $t = t'\alpha_{j,n_j-1}$ for either $j \in \{1,2\}$ such that $t'|_{\mathrm{alph}(\mathcal{L}_j)} = \alpha_{j,1}\alpha_{j,2}\ldots\alpha_{j,n_j-2} \setminus \{\tau\}$ and $t'|_{\mathrm{alph}(\mathcal{L}_{\bar{j}})} = \alpha_{\bar{j},1}\alpha_{\bar{j},2}\ldots\alpha_{\bar{j},n_{\bar{j}}-1} \setminus \{\tau\}$. To see that there is an execution of $\mathcal{L}_1 \parallel \mathcal{L}_2$ to the state (s_{1,n_1}, s_{2,n_2}) such that t is the sequence obtained from it by removing all the states and the invisible actions, we argue like in the previous case.

Now, we have covered all six possibilities, so the induction step is complete. By the induction principle, it means whenever t is a sequence over $\mathrm{alph}(\mathcal{L}_1) \cup \mathrm{alph}(\mathcal{L}_2)$ such that $t|_{\mathrm{alph}(\mathcal{L}_i)}$ is a trace of $\mathcal{L}_i$ for both $i \in \{1,2\}$, then t is a trace of $\mathcal{L}_1 \parallel \mathcal{L}_2$. Hence, the first claim is proved.

The last two claims of the lemma follow straightforwardly from the definitions of renaming and hiding. □

Proposition 10. To prove the first claim, first note that the alphabets of $(\mathcal{L}_1 \setminus \Lambda_1) \parallel (\mathcal{L}_2 \setminus \Lambda_1)$ and $(\mathcal{L}_1 \parallel \mathcal{L}_2) \setminus \Lambda_1$ are clearly the same. Next, if $t \in \mathrm{tr}((\mathcal{L}_1 \parallel \mathcal{L}_2) \setminus \Lambda_1)$, then by Lemma 3, there is $t' \in \mathrm{tr}(\mathcal{L}_1 \parallel \mathcal{L}_2)$ such that $t = t' \setminus \Lambda_1$. By the same lemma, $t'|_{\mathrm{alph}(\mathcal{L}_i)} \in \mathrm{tr}(\mathcal{L}_i)$ for both $i \in \{1,2\}$, which implies that

$$t|_{\mathrm{alph}(\mathcal{L}_i \setminus \Lambda_1)} = (t' \setminus \Lambda_1)|_{\mathrm{alph}(\mathcal{L}_i) \setminus \Lambda_1} = t'|_{\mathrm{alph}(\mathcal{L}_i)} \setminus \Lambda_1 \in \mathrm{tr}(\mathcal{L}_i \setminus \Lambda_1)$$

for both $i \in \{1,2\}$. It, in turn, means that $t \in \mathrm{tr}((\mathcal{L}_1 \setminus \Lambda_1) \parallel (\mathcal{L}_2 \setminus \Lambda_1))$. Therefore, $(\mathcal{L}_1 \parallel \mathcal{L}_2) \setminus \Lambda_1$ is a traces refinement of $(\mathcal{L}_1 \setminus \Lambda_1) \parallel (\mathcal{L}_2 \setminus \Lambda_1)$.

To prove the second claim, first note that the alphabet of $(\mathcal{L} \setminus \Lambda_1) \parallel (\mathcal{L} \setminus \Lambda_2)$ is precisely the alphabet of $\mathcal{L}$ when Λ_1 and Λ_2 are disjoint. Next, if $t \in \mathrm{tr}(\mathcal{L})$, then $t|_{\mathrm{alph}(\mathcal{L} \setminus \Lambda_i)} = t \setminus \Lambda_i \in \mathrm{tr}(\mathcal{L} \setminus \Lambda_i)$ for both $i \in \{1,2\}$, which by Lemma 3, implies that $t \in \mathrm{tr}((\mathcal{L} \setminus \Lambda_1) \parallel (\mathcal{L} \setminus \Lambda_2))$. Hence, $(\mathcal{L} \setminus \Lambda_1) \parallel (\mathcal{L} \setminus \Lambda_2) \succeq_{\mathrm{tr}} \mathcal{L}$, provided Λ_1 and Λ_2 are disjoint.

The last two claims are trivial, because the LTSs on both sides of the equivalence are obviously the same. □

Proposition 11. Let us use the notation of the lemma. To prove the first claim, note that the alphabets of $\mathcal{L}_1 \parallel \mathcal{L}_2$ and $\mathcal{L}'_1 \parallel \mathcal{L}'_2$ are clearly the same under the given assumptions. If $t \in \mathrm{tr}(\mathcal{L}'_1 \parallel \mathcal{L}'_2)$, then by Lemma 3, $t|_{\mathrm{alph}(\mathcal{L}'_i)} \in \mathrm{tr}(\mathcal{L}'_i)$ for both $i \in \{1,2\}$. Moreover, if $\mathcal{L}_i \succeq_{\mathrm{tr}} \mathcal{L}'_i$, then $t|_{\mathrm{alph}(\mathcal{L}_i)} = t|_{\mathrm{alph}(\mathcal{L}'_i)} \in \mathrm{tr}(\mathcal{L}'_i) \subseteq \mathrm{tr}(\mathcal{L}_i)$ for both $i \in \{1,2\}$, which by Lemma 3 again, means that $t \in \mathrm{tr}(\mathcal{L}_1 \parallel \mathcal{L}_2)$. Hence, the claim holds.

Also in the second case, it is clear that the alphabets of $\zeta(\mathcal{L}_1)$ and $\zeta(\mathcal{L}_2)$ match when $\mathcal{L}_1 \succeq_{\text{tr}} \mathcal{L}_2$. If $t \in \text{tr}(\zeta(\mathcal{L}_2))$, then by Lemma 3, there exists $\alpha_1\alpha_2\cdots\alpha_n \in \text{tr}(\mathcal{L}_2)$ such that $t = \zeta(\alpha_1)\zeta(\alpha_2)\cdots\zeta(\alpha_n)$, where $n \in \mathbb{N}$ and $\alpha_i \in \mathbb{V}$ for all $i \in \{1,2,\ldots,n\}$. Moreover, if $\mathcal{L}_1 \succeq_{\text{tr}} \mathcal{L}_2$, then $\alpha_1\alpha_2\cdots\alpha_n \in \text{tr}(\mathcal{L}_1)$, too. By Lemma 3 again, it means that $t \in \text{tr}(\zeta(\mathcal{L}_1))$, and the claim holds.

To prove the last claim, note that the alphabets of $\mathcal{L}_1 \setminus \Lambda$ and $\mathcal{L}_2 \setminus \Lambda$ are clearly the same when $\mathcal{L}_1 \succeq_{\text{tr}} \mathcal{L}_2$. If $t \in \text{tr}(\mathcal{L}_2 \setminus \Lambda)$, then by Lemma 3, there is $t' \in \text{tr}(\mathcal{L}_2)$ such that $t = t' \setminus \Lambda$. Moreover, if $\mathcal{L}_1 \succeq_{\text{tr}} \mathcal{L}_2$, then $t' \in \text{tr}(\mathcal{L}_1)$, too. By Lemma 3 again, it means that $t \in \text{tr}(\mathcal{L}_1 \setminus \Lambda)$. Hence, the claim holds. □

544. Saari, Eija (2009) Towards minimizing measurement uncertainty in total petroleum hydrocarbon determination by GC-FID

545. Pihlajaniemi, Henna (2009) Success of micropropagated woody landscape plants under northern growing conditions and changing environment

546. Pajunen, Anu (2010) Willow-characterised shrub vegetation in tundra and its relation to abiotic, biotic and anthropogenic factors

547. Salin, Mikko (2010) Protein crystallographic studies of A-TIM—structure based development of new enzymes

548. Kemppainen, Jukka (2010) Behaviour of the boundary potentials and boundary integral solution of the time fractional diffusion equation

549. Varga, Sandra (2010) Significance of plant gender and mycorrhizal symbiosis in plant life history traits

550. Räisänen, Teppo (2010) All for one, one for all. Organizational knowledge creation and utilization using a new generation of IT tools

551. Luimula, Mika (2010) Development and evaluation of the location-aware platform. Main characteristics in adaptable location-aware systems

552. Bisi, Jukka (2010) Suomalaisen susikonfliktin anatomia

553. Poutanen, Hilkka (2010) Developing the role of human resource information systems for the activities of good leadership

554. Saukkoriipi, Jaakko (2010) Theoretical study of the hydrolysis of aluminum complexes

555. Sirviö, Anu (2010) The role of factors promoting genetic diversity within social insect colonies

556. Tahkokorpi, Marjaana (2010) Anthocyanins under drought and drought-related stresses in bilberry (*Vaccinium myrtillus* L.)

557. Piippo, Sari (2010) Grazing tolerance of biennial meadow plants in relation to resource availability

558. Lappalainen, Niina (2010) The responses of ectohydric and endohydric mosses under ambient and enhanced ultraviolet radiation

559. Luojus, Satu (2010) From a momentary experience to a lasting one. The concept of and research on expanded user experience of mobile device

Book orders:
Granum: Virtual book store
http://granum.uta.fi/granum/

UNIVERSITY OF OULU P.O.B. 7500 FI-90014 UNIVERSITY OF OULU FINLAND

ACTA UNIVERSITATIS OULUENSIS

ISBN 978-951-42-6251-7 (Paperback)
ISBN 978-951-42-6252-4 (PDF)
ISSN 0355-3191 (Print)
ISSN 1796-220X (Online)

UNIVERSITY of OULU
OULUN YLIOPISTO