Conditional Transition Systems with Upgrades

Harsh Beohar
Universität Duisburg-Essen

Barbara König
Universität Duisburg-Essen

Sebastian Küpper
Universität Duisburg-Essen

Alexandra Silva
University College London

***Abstract*—We introduce a variant of transition systems, where activation of transitions depends on conditions of the environment and upgrades during runtime potentially create additional transitions. Using a cornerstone result in lattice theory, we show that such transition systems can be modelled in two ways: as conditional transition systems (CTS) with a partial order on conditions, or as lattice transition systems (LaTS), where transitions are labelled with the elements from a distributive lattice. We define equivalent notions of bisimilarity for both variants and characterise them via a bisimulation game.**

We explain how conditional transition systems are related to featured transition systems for the modelling of software product lines. Furthermore, we show how to compute bisimilarity symbolically via BDDs by defining an operation on BDDs that approximates an element of a Boolean algebra into a lattice. We have implemented our procedure and provide runtime results.

I. Introduction

Conditional transition systems (CTS) have been introduced in [1] as a model for systems whose behaviour is guarded by different conditions. Before an execution, a condition is chosen by the environment from a pre-defined set of conditions and, accordingly, the CTS is instantiated to a classical labelled transition system (LTS). In this work, we consider *ordered* sets of conditions which allow for a change of conditions during runtime. It is allowed to replace a condition by a smaller condition, called upgrade. An upgrade activates additional transitions compared to the previous instantiation of the system.

Our focus lies on formulating a notion of behavioural equivalence, called *conditional bisimilarity*, that is insensitive to changes in behaviour that may occur due to upgrades. Given two states, we want to determine under which conditions they are behaviourally equivalent. To compute this, we adopt a dual, but equivalent, view from lattice theory due to Birkhoff to represent a CTS by a lattice transition system (LaTS). In general, LaTSs are more compact in nature than their CTS counterparts. Moreover, we also develop an efficient procedure based on matrix multiplication to compute conditional bisimilarity.

Such questions are relevant when we compare a system with its specification or we want to modify a system in such a way that its observable behaviour is invariant. Furthermore, one requires minimisation procedures for transition systems that are potentially very large and need to be made more compact to be effectively used in analysis.

An application of CTSs with upgrades is to model systems that deteriorate over time. Consider a system that is dependent on components that break over time or require calibration, in particular sensor components. In such systems, due to inconsistent sensory data from a sensor losing its calibration, additional behaviour in a system may be enabled (which can be modelled as an upgrade) and chosen nondeterministically.

Another field of interest, which will be explored in more detail, are software product lines (SPLs). SPLs refer to a software engineering method for managing and developing a collection of similar software systems with common features. To ensure correctness of such systems in an efficient way, it is common to specify the behaviour of many products in a single transition system and provide suitable analysis methods based on model-checking or behavioural equivalences (see [3], [6]–[9], [12], [15], [17], [24]).

Featured transition systems (FTS) – a recent extension of conventional transition system proposed by Classen et al.[7] – have become the standard formalism to model an SPL. An important issue usually missing in the theory of FTSs is the notion of self-adaptivity [11], i.e., the view that features or products are not fixed a priori, but may change during runtime. We will show that FTSs can be considered as CTSs without upgrades where the conditions are the powerset of the features. Additionally, we propose to incorporate a notion of upgrades into software product lines, that cannot be captured by FTSs. Furthermore, we also consider deactivation of transitions in Appendix C, to which our techniques can easily be adapted, though some mathematical elegance is lost in the process.

Our contributions are as follows. First, we make the different levels of granularity – features, products and sets of products – in the specification of SPLs explicit and give a theoretical foundation in terms of Boolean algebras and lattices. Second, we present a theory of behavioural equivalences with corresponding games and algorithms and applications to conventional and adaptive SPLs. Third, we present our implementation based on binary decision diagrams (BDDs), which provides a compact encoding of a propositional formula and also show how they can be employed in a lattice-based setting. Lastly, we show how a BDD-based matrix multiplication algorithm provides us with an efficient way to check bisimilarity relative to the naive approach of checking all products separately.

This paper is organised as follows. Section II recalls the fundamentals of lattice theory relevant to this paper. Then, in Section III we formally introduce CTSs and conditional bisimilarity. In Section IV, using the Birkhoff duality, it is shown that CTSs can be represented as lattice transition systems (LaTSs) whose transitions are labelled with the el-

Research partially supported by DFG project BEMEGA and ERC Starting Grant ProFoundNet (grant agreement 679127).

978-1-5386-1925-4/17/$31.00

ements from a distributive lattice. Moreover, the bisimilarity introduced on LaTSs is shown to coincide with the conditional bisimilarity on the corresponding CTSs. In Section V, we show how bisimilarity can be computed using a form of matrix multiplication. Section VI focusses on the translation between an FTS and a CTS, and moreover, a BDD-based implementation of checking bisimilarity is laid out. Lastly, we conclude with a discussion on related work and future work in Section VII. All the proofs can be found in Appendix A.

II. Preliminaries

We now recall some basic definitions concerning lattices, including the well-known Birkhoff's duality result from [13].

Definition 1 (Lattice, Heyting Algebra, Boolean Algebra)**.** *Let $(\mathbb{L}, \sqsubseteq)$ be a partially ordered set. If for each pair of elements $\ell, m \in \mathbb{L}$ there exists a supremum $\ell \sqcup m$ and an infimum $\ell \sqcap m$, we call $(\mathbb{L}, \sqcup, \sqcap)$ a* lattice*. A* bounded lattice *has a top element 1 and a bottom element 0. A lattice is* complete *if every subset of $\mathbb{L}$ has an infimum and a supremum. It is* distributive *if $(\ell \sqcup m) \sqcap n = (\ell \sqcap n) \sqcup (m \sqcap n)$ holds for all $\ell, m, n \in \mathbb{L}$.*

A bounded lattice $\mathbb{L}$ is a Heyting algebra *if for any $\ell, m \in \mathbb{L}$, there is a greatest element ℓ' such that $\ell \sqcap \ell' \sqsubseteq m$. The residuum and negation are defined as $\ell \to m = \bigsqcup\{\ell' \mid \ell \sqcap \ell' \sqsubseteq m\}$ and $\neg\ell = \ell \to 0$. A* Boolean algebra *$\mathbb{L}$ is a Heyting algebra satisfying $\neg\neg\ell = \ell$ for all $\ell \in \mathbb{L}$.*

Example 1. *Given a set of atomic propositions N, consider $\mathbb{B}(N)$, the set of all Boolean expressions over N, i.e., the set of all formulae of propositional logic. We equate every subset $C \subseteq N$ with the evaluation that assigns true to all $f \in C$ and false to all $f \in N \backslash C$. For $b \in \mathbb{B}(N)$, we write $C \models b$ whenever C satisfies b. Furthermore we define $[\![b]\!] = \{C \subseteq N \mid C \models b\} \in \mathcal{P}(\mathcal{P}(N))$. Two Boolean expressions b_1, b_2 are called equivalent whenever $[\![b_1]\!] = [\![b_2]\!]$. Furthermore b_1 implies b_2 ($b_1 \models b_2$), whenever $[\![b_1]\!] \subseteq [\![b_2]\!]$.*

The set $\mathbb{B}(N)$, quotiented by equivalence, is a Boolean algebra, isomorphic to $\mathcal{P}(\mathcal{P}(N))$, where $[\![b_1]\!] \sqcup [\![b_2]\!] = [\![b_1]\!] \cup [\![b_2]\!] = [\![b_1 \vee b_2]\!]$, analogously for $\sqcap, \cap, \wedge$, $\neg[\![b]\!] = \mathcal{P}(N) \backslash [\![b]\!] = [\![\neg b]\!]$, and $[\![b_1]\!] \to [\![b_2]\!] = \mathcal{P}(N) \backslash [\![b_1]\!] \cup [\![b_2]\!] = [\![\neg b_1 \vee b_2]\!]$.

Distributive lattices and Boolean algebras give rise to an interesting duality result, which was first stated for finite lattices by Birkhoff and extended to the infinite case by Priestley [13]. In the sequel we will focus on finite distributive lattices (which are Heyting algebras). We first need the following concepts.

Definition 2. *Let $\mathbb{L}$ be a lattice. An element $n \in \mathbb{L} \backslash \{0\}$ is said to be* (join-)irreducible *if whenever $n = \ell \sqcup m$ for elements $\ell, m \in \mathbb{L}$, it always holds that $n = \ell$ or $n = m$. We write $\mathcal{J}(\mathbb{L})$ for the set of all irreducible elements of $\mathbb{L}$.*

Let $(S, \leq)$ be a partially ordered set. A subset $S' \subseteq S$ is downward-closed*, whenever $s' \in S'$ and $s \leq s'$ implies $s \in S'$. We write $\mathcal{O}(S)$ for the set of all downward-closed subsets of S and $\downarrow s = \{s' \mid s' \leq s\}$ for the downward-closure of $s \in S$.*

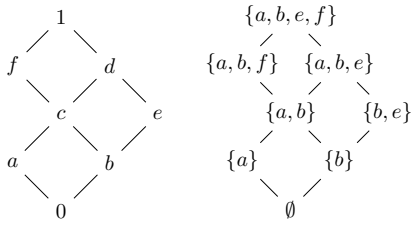

Fig. 1. An example motivating Birkhoff's representation theorem.

Example 2. *For our example of a Boolean algebra $\mathbb{B}(N)$, quotiented by equivalence, the irreducibles are the complete conjunctions of literals, or, alternatively, all sets $C \subseteq N$.*

We can now state the Birkhoff's representation theorem for finite distributive lattices [13].

Theorem 1. *If $\mathbb{L}$ is a finite distributive lattice, then $(\mathbb{L}, \sqcup, \sqcap) \cong (\mathcal{O}(\mathcal{J}(\mathbb{L})), \cup, \cap)$ via the isomorphism $\eta : \mathbb{L} \to \mathcal{O}(\mathcal{J}(\mathbb{L}))$, defined as $\eta(\ell) = \{\ell' \in \mathcal{J}(\mathbb{L}) \mid \ell' \sqsubseteq \ell\}$. Furthermore, given a finite partially ordered set $(S, \leq)$, the downward-closed subsets of S, $(\mathcal{O}(S), \cup, \cap)$ form a distributive lattice, with inclusion ($\subseteq$) as the partial order. The irreducibles of this lattice are all downward-closed sets of the form $\downarrow s$ for $s \in S$.*

Example 3. *Consider the lattice $\mathbb{L} = \{0, a, b, c, d, e, f, 1\}$ with the order depicted in Figure 1. The irreducible elements are a, b, e, f, i.e. exactly those elements that have a unique direct predecessor. On the right we depict the dual representation of the lattice in terms of downward-closed sets of irreducibles, ordered by inclusion. This example suggests an embedding of a distributive lattice $\mathbb{L}$ into a Boolean algebra, obtained by taking the powerset of irreducibles.*

Proposition 1 (Embedding)**.** *A finite distributive lattice $\mathbb{L}$ embeds into the Boolean algebra $\mathbb{B} = \mathcal{P}(\mathcal{J}(\mathbb{L}))$ via the mapping $\eta : \mathbb{L} \to \mathbb{B}$ given by $\eta(\ell) = \{\ell' \in \mathcal{J}(\mathbb{L}) \mid \ell' \sqsubseteq \ell\}$.*

We will simply assume that $\mathbb{L} \subseteq \mathbb{B}$. Since an embedding is a lattice homomorphism, supremum and infimum coincide in $\mathbb{L}$ and $\mathbb{B}$ and we write $\sqcup, \sqcap$ for both versions. Negation and residuum may however differ and we distinguish them via a subscript, writing $\neg_{\mathbb{L}}, \neg_{\mathbb{B}}$ and $\to_{\mathbb{L}}, \to_{\mathbb{B}}$. Given such an embedding, we can approximate elements of a Boolean algebra in the embedded lattice.

Definition 3. *Let a complete distributive lattice $\mathbb{L}$ that embeds into a Boolean algebra $\mathbb{B}$ be given. Then, the* approximation *of $\ell \in \mathbb{B}$ is given by: $\lfloor \ell \rfloor_{\mathbb{L}} = \bigsqcup\{\ell' \in \mathbb{L} \mid \ell' \sqsubseteq \ell\}$.*

If the lattice is clear from the context, we will in the sequel drop the subscript $\mathbb{L}$ and simply write $\lfloor \ell \rfloor$. For instance, in the previous example, the set of irreducibles $\{a, e, f\}$, which is not downward-closed, is approximated by $\lfloor \{a, e, f\} \rfloor = \{a\}$.

Lemma 1. *Let $\mathbb{L}$ be a complete distributive lattice that embeds into a Boolean algebra $\mathbb{B}$. For $\ell, m \in \mathbb{B}$, we have $\lfloor \ell \sqcap m \rfloor = \lfloor \ell \rfloor \sqcap \lfloor m \rfloor$ and furthermore that $\ell \sqsubseteq m$ implies $\lfloor \ell \rfloor \sqsubseteq \lfloor m \rfloor$. If $\ell, m \in \mathbb{L}$, then $\lfloor \ell \sqcup \neg m \rfloor = m \to_{\mathbb{L}} \ell$.*

Note that in general it does not hold that $\lfloor \ell \sqcup m \rfloor = \lfloor \ell \rfloor \sqcup \lfloor m \rfloor$

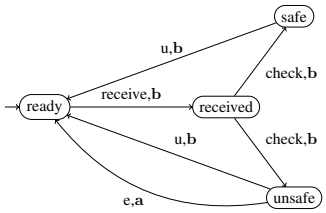

Fig. 2. Adaptive routing protocol with the alphabet $A = \{\text{receive}, \text{check}, \text{u}, \text{e}\}$.

and $\lfloor \ell \sqcup \neg m \rfloor = \lfloor m \rfloor \to_{\mathbb{L}} \lfloor \ell \rfloor$ for arbitrary $\ell, m \in \mathbb{B}$. To witness why these equations fail to hold, take $\ell = \{a, e\}$ and $m = \{b, f\}$ in the previous example as counterexample.

III. Conditional Transition Systems

In this section we introduce conditional transition systems together with a notion of behavioural equivalence based on bisimulation. In [1], such transition systems were already investigated in a coalgebraic setting, where the set of conditions was trivially ordered. In the sequel, we will always use CTS for the variant with upgrades defined as follows:

Definition 4. *A* conditional transition system *(CTS) over an alphabet A and a finite ordered set of conditions $(\Phi, \leq)$ is a triple (X, A, f), where X is a set of states and $f : X \times A \to (\Phi \to \mathcal{P}(X))$ is a function mapping every ordered pair in $X \times A$ to a monotone function of type $(\Phi, \leq) \to (\mathcal{P}(X), \supseteq)$. As usual, we write $x \xrightarrow{a,\varphi} y$ whenever $y \in f(x, a)(\varphi)$.*

Intuitively, a CTS evolves as follows. Before the system starts acting, it is assumed that a condition $\varphi \in \Phi$ is chosen arbitrarily which may represent a selection of a valid product of the system. Now all the transitions that have a condition greater than or equal to φ are activated, while the remaining transitions are inactive. Henceforth, the system behaves like a standard transition system; until at any point in the computation, the condition is changed to a smaller one (say, φ') signifying a selection of a valid, upgraded product. This, in turn, has a propelling effect in the sense that now (de)activation of transitions depends on the new condition φ', rather than on the old condition φ. Note that due to the monotonicity restriction we have that $x \xrightarrow{a,\varphi} y$ and $\varphi' \leq \varphi$ imply $x \xrightarrow{a,\varphi'} y$. That is, active transitions remain active during an upgrade, but new transitions may become active. In Appendix C, we weaken this requirement by discussing a mechanism for deactivating transitions via priorities on the alphabet.

Example 4. *Consider an example (simplified from [11]) of an adaptive routing protocol modelled as a CTS in Figure 2. The system has two products: the* basic *system, denoted* $\mathbf{b}$*, with no encryption feature and the* advanced *system, denoted* $\mathbf{a}$*, with an encryption feature. The ordering on the products is* $\mathbf{a} < \mathbf{b}$*. Transitions that are present due to monotonicity are omitted.*

Initially, the system is in state 'ready' and is waiting to receive a message. Once a message is received there is a check whether the system's environment is safe or unsafe, leading to non-deterministic branching. If the encryption feature is present, then the system can send an encrypted message (e) from the unsafe state only; otherwise, the system sends an unencrypted message (u) regardless of the state being 'safe' or 'unsafe'. Note that such a behaviour description can easily be encoded by a transition function. E.g., $f(\textit{received}, \textit{check})(\mathbf{b}) = \{\textit{safe}, \textit{unsafe}\}$ and $f(\textit{received}, a)(x) = \emptyset$ (for $x \in \{\mathbf{a}, \mathbf{b}\}$ and $a \in A \setminus \{\textit{check}\}$) specifies the transitions that can be fired from the received state to the (un)safe states.

Next, we turn our attention towards (strong) bisimulation relations for CTSs which consider the ordering of conditions in their transfer properties.

Definition 5. *Let (X, A, f), (Y, A, g) be two CTSs over the same set of conditions $(\Phi, \leq)$. For a condition $\varphi \in \Phi$, we define $f_\varphi(x, a) = f(x, a)(\varphi)$ to denote the traditional* (A-)labelled transition system *induced by a CTS (X, A, f). Two states $x \in X, y \in Y$ are* conditionally bisimilar *under a condition $\varphi \in \Phi$, denoted $x \sim_\varphi y$, if there is a family of relations $R_{\varphi'} \subseteq X \times Y$ (for every $\varphi' \leq \varphi$) such that*

(i) each relation $R_{\varphi'}$ is a traditional bisimulation relation between $f_{\varphi'}$ and $g_{\varphi'}$,
(ii) whenever $\varphi' \leq \varphi''$, we have $R_{\varphi'} \supseteq R_{\varphi''}$, and
(iii) R_φ relates x and y, i.e., $(x, y) \in R_\varphi$.

Example 5. *Consider the CTS illustrated in Figure 2 where the condition $\mathbf{b}$ of the transition 'received $\xrightarrow{\textit{check},\mathbf{b}}$ unsafe' is replaced by $\mathbf{a}$. Let $\textit{ready}_1$ and $\textit{ready}_2$ denote the initial states of the system before and after the above modification, respectively. Then, we find $\textit{ready}_1 \sim_{\mathbf{a}} \textit{ready}_2$; however, $\textit{ready}_1 \not\sim_{\mathbf{b}} \textit{ready}_2$. To see why the latter fails to hold, let $R_{\mathbf{b}}$ be the bisimulation relation in the traditional sense between the states $\textit{ready}_1, \textit{ready}_2$ under condition $\mathbf{b}$. Then, one finds that the states $\textit{unsafe}_1, \textit{safe}_2$ are bisimilar in the traditional sense, i.e., $(\textit{unsafe}_1, \textit{safe}_2) \in R_{\mathbf{b}}$. However, the two states cannot be related by any traditional bisimulation relation under condition $\mathbf{a}$; thus, violating Condition 2 of Definition 5.*

Indeed, the two systems behave differently. In the first, it is possible to perform actions receive, check (arrive in state unsafe), do an upgrade, and send an encrypted message (e), which is not feasible in the second system because the check transition forces the system to be in the safe state before doing the upgrade. However, without upgrades, the above systems would be bisimilar for both products.

We end this section by adapting the classical bisimulation game to conditional transition systems; thus, incorporating our intuitive explanation of upgrades with the notion of bisimilarity.

Definition 6 (Bisimulation Game)**.** *Given two CTSs (X, A, f) and (Y, A, g) over a poset $(\Phi, \leq)$, a state $x \in X$, a state $y \in Y$, and a condition $\varphi \in \Phi$, the bisimulation game is a round-based two-player game that uses both CTSs as game boards. Let (x, y, φ) be a game instance indicating that x, y are marked and the current condition is φ. The game progresses to the next game instance as follows:*

- *Player 1 is the first one to move. Player 1 can decide to make an upgrade, i.e., replace the condition φ by a smaller one (say $\varphi' \leq \varphi$, for some $\varphi' \in \Phi$).*
- *Player 1 can choose the marked state $x \in X$ (or $y \in Y$) and performs a transition $x \xrightarrow{a,\varphi'} x'$ ($y \xrightarrow{a,\varphi'} y'$).*
- *Player 2 then has to simulate the last step, i.e., if Player 1 made a step $x \xrightarrow{a,\varphi'} x'$, Player 2 is required to make step $y \xrightarrow{a,\varphi'} y'$ and vice-versa.*
- *In turn, the new game instance is (x', y', φ').*

Player 1 wins if Player 2 cannot simulate the last step performed by Player 1. Player 2 wins if the game never terminates or Player 1 cannot make another step.

So bisimulation is characterised as follows: Player 2 has a winning strategy for a game instance (x, y, φ) if and only if $x \sim_\varphi y$. The proof and the computation of the winning strategies for both players are given in Appendix A-B.

IV. Lattice Transition Systems

Recall from Section II that there is a duality between partial orders and distributive lattices. In fact, as we will show below, this result can be lifted to the level of transition systems as follows: a conditional transition system over a poset is equivalent to a transition system whose transitions are labelled by the downward-closed subsets of the poset.

Definition 7. *A* lattice transition system *(LaTS) over a finite distributive lattice $\mathbb{L}$ and an alphabet A is a triple (X, A, α) with a set of states X and a transition function $\alpha : X \times A \times X \to \mathbb{L}$. A LaTS (X, A, α) is* finite *if the sets X, A are finite.*

Note that superficially, lattice transition systems resemble weighted automata [14]. However, while in weighted automata the lattice annotations are seen as weights that are accumulated, in CTSs they play the role of guards that control which transitions can be taken. Furthermore, the notions of behavioural equivalence are quite different.

Given a CTS (X, A, f) over $(\Phi, \leq)$, we can easily construct a LaTS over $\mathcal{O}(\Phi)$ by defining $\alpha(x, a, x') = \{\varphi \in \Phi \mid x' \in f(x, a)(\varphi)\}$ for $x, x' \in X$, $a \in A$. Due to monotonicity, $\alpha(x, a, x')$ is always downward-closed. Similarly, a LaTS can be converted into a CTS by using the Birkhoff duality and by taking the irreducibles as conditions.

Theorem 2. *The set of all CTSs over a set of conditions Φ is isomorphic to the set of all LaTSs over the lattice whose elements are the downward-closed subsets of Φ.*

So every LaTS over a finite distributive lattice gives rise to a CTS in our sense (cf. Definition 4) and since finite Boolean algebras are finite distributive lattices, conditional transition systems in the sense of [1] are CTSs in our sense as well. We chose the definition of a CTS using posets instead of the dual view using lattices, because this view yields a natural description which models transitions in terms of conditions (product versions), though when computing with CTSs we often choose the lattice view. By adopting this view, conditional bisimulations can be computed symbolically and hence more efficiently (cf. Section VI-B).

Definition 8. *Let (X, A, α) and (Y, A, β) be any two LaTSs over a lattice $\mathbb{L}$. A conditional relation R, i.e., a function of type $R : X \times Y \to \mathbb{L}$ is a* lattice bisimulation *for α, β if and only if the following transfer properties are satisfied.*

(i) For all $x, x' \in X$, $y \in Y$, $a \in A$, $\ell \in \mathcal{J}(\mathbb{L})$ whenever $x \xrightarrow{a,\ell} x'$ and $\ell \sqsubseteq R(x, y)$, there exists $y' \in Y$ such that $y \xrightarrow{a,\ell} y'$ and $\ell \sqsubseteq R(x', y')$.

(ii) Symmetric to (i) with the roles of x and y interchanged.

In the above, we write $x \xrightarrow{a,\ell} x'$, whenever $\ell \sqsubseteq \alpha(x, a, x')$.

For $\varphi \in \Phi$, a transition $x \xrightarrow{a,\varphi} x'$ exists in the CTS if and only if there is a transition $x \xrightarrow{a,\downarrow\varphi} x'$ in the corresponding LaTS. Hence they are denoted by the same symbol.

Theorem 3. *Let (X, A, f) and (Y, A, g) be any two CTSs over Φ. Two states $x \in X, y \in Y$ are conditionally bisimilar under a condition φ if and only if there is a lattice bisimulation R between the corresponding LaTSs such that $\varphi \in R(x, y)$.*

Incidentally, the order in $\mathbb{L}$ gives rise to a natural order on lattice bisimulations. For any two lattice bisimulations $R_1, R_2 : X \times Y \to \mathbb{L}$, we write $R_1 \sqsubseteq R_2$ if and only if $R_1(x, y) \sqsubseteq R_2(x, y)$ for all $x \in X, y \in Y$. As a result, taking the element-wise supremum of a family of lattice bisimulations is again a lattice bisimulation. Therefore, the greatest lattice bisimulation for a LaTS always exists, just like in the traditional case.

Lemma 2. *Let $R_i \in X \times Y \to \mathbb{L}, i \in I$ be lattice bisimulations for a pair of LaTSs (X, A, α) and (Y, A, β). Then $\bigsqcup\{R_i \mid i \in I\}$ is a lattice bisimulation.*

V. Computation of Lattice Bisimulation

The goal of this section is to present an algorithm that computes the greatest lattice bisimulation between a given pair of LaTSs. In particular, we first characterise lattice bisimulation as a post-fixpoint of an operator F on the set of all conditional relations. Then, we show that this operator F is monotone with respect to the ordering relation $\sqsubseteq$; thereby, ensuring that the greatest bisimulation always exists by applying the well-known Knaster-Tarski fixpoint theorem. Moreover, on finite lattices and finite sets of states, the usual fixpoint iteration starting with the trivial conditional relation (i.e., the constant 1-matrix over $\mathbb{L}$) can be used to compute the greatest lattice bisimulation. Lastly, we give a translation of F in terms of matrices using a form of matrix multiplication found in the literature of residuated lattices [4] and database design [18].

A. A Fixpoint Approach

Throughout this section, we let $\alpha : X \times A \times X \to \mathbb{L}$, $\beta : Y \times A \times Y \to \mathbb{L}$ denote any two LaTSs, $\mathbb{L}$ denote a finite distributive lattice, and $\mathbb{B}$ denote the Boolean algebra that this lattice embeds into.

Definition 9. *Recall the residuum operator $\to$ on a lattice and define three operators $F_1, F_2, F : (X \times Y \to \mathbb{L}) \to$*

$(X \times Y \to \mathbb{L})$ *in the following way (for* $R \in X \times Y \to \mathbb{L}$*,* $x \in X$*,* $y \in Y$*):*

$$F_1(R)(x,y) = \prod_{a \in A, x' \in X} \Big(\alpha(x,a,x') \to \big(\bigsqcup_{y' \in Y} (\beta(y,a,y') \sqcap R(x',y')) \big) \Big),$$

$$F_2(R)(x,y) = \prod_{a \in A, y' \in Y} \Big(\beta(y,a,y') \to \big(\bigsqcup_{x' \in X} (\alpha(x,a,x') \sqcap R(x',y')) \big) \Big),$$

$$F(R)(x,y) = F_1(R)(x,y) \sqcap F_2(R)(x,y).$$

Note that the above definition is provided for a distributive lattice, viewing it in classical two-valued Boolean algebra results in the well-known transfer properties of a bisimulation.

Theorem 4. *A conditional relation* R *is a lattice bisimulation if and only if* $R \sqsubseteq F(R)$*.*

Next, it is easy to see that F is a monotone operator with respect to the ordering $\sqsubseteq$ on $\mathbb{L}$ since the infimum and supremum are both monotonic, and moreover, the residuum operation is monotonic in the second component. As a result, we can use the following fixpoint iteration to compute the greatest bisimulation while working with finite lattices and finite sets of states.

Algorithm 1. Let (X, A, α) and (Y, A, β) be two finite LaTSs. Fix R_0 as $R_0(x,y) = 1$ for all $x \in X, y \in Y$. Then, compute $R_{i+1} = F(R_i)$ for all $i \in \mathbb{N}_0$ until $R_i \sqsubseteq R_{i+1}$. Lastly, return R_i as the greatest bisimulation.

Suppose $\alpha = \beta$, it is not hard to see that the fixpoint iteration must stabilise after at most $|X|$ steps, since each R_i induces equivalence relations for all conditions φ and refinements regarding φ are immediately propagated to every $\varphi' \geq \varphi$. An equivalence relation can be refined at most $|X|$ times, limiting the number of iterations.

B. Lattice Bisimilarity is Finer than Boolean Bisimilarity

We now show the close relation of the notions of bisimilarity for a LaTS defined over a finite distributive lattice $\mathbb{L}$ and a Boolean algebra $\mathbb{B}$. As usual, let (X, A, α) and (Y, A, β) be any two LaTSs together with the restriction that the lattice $\mathbb{L}$ embeds into the Boolean algebra $\mathbb{B}$. Moreover, let $F_{\mathbb{L}}$ and $F_{\mathbb{B}}$ be the monotonic operators as defined in Definition 9 over the lattice $\mathbb{L}$ and the Boolean algebra $\mathbb{B}$, respectively. We say that R is an $\mathbb{L}$-bisimulation (resp. $\mathbb{B}$-bisimulation) whenever $R \sqsubseteq F_{\mathbb{L}}(R)$ (resp. $R \sqsubseteq F_{\mathbb{B}}(R)$).

Proposition 2.

(i) If $R : X \times Y \to \mathbb{L}$*, then* $\lfloor F_{\mathbb{B}}(R) \rfloor = F_{\mathbb{L}}(R)$*.*
(ii) Every $\mathbb{L}$*-bisimulation is also a* $\mathbb{B}$*-bisimulation.*
(iii) A $\mathbb{B}$*-bisimulation* $R : X \times Y \to \mathbb{B}$ *is an* $\mathbb{L}$*-bisimulation whenever all the entries of* R *are in* $\mathbb{L}$*.*

However, even though the two notions of bisimilarity are closely related, they are not identical, i.e., it is not true that whenever a state x is bisimilar to a state y in $\mathbb{B}$ that it is also bisimilar in $\mathbb{L}$ (see Example 4 where we encounter a $\mathbb{B}$-bisimulation, which is not an $\mathbb{L}$-bisimulation).

C. Matrix Multiplication

An alternative way to represent a LaTS (X, A, α) is to view the transition function α as a family of matrices $\alpha_a : X \times X \to \mathbb{L}$ (one for each action $a \in A$) with $\alpha_a(x,x') = \alpha(x,a,x')$, for every $x, x' \in X$. We use standard matrix multiplication (where $\sqcup$ is used for addition and $\sqcap$ for multiplication), as well as a special form of matrix multiplication [4], [18].

Definition 10 ($\otimes$-multiplication)**.** *Given an* $X \times Y$*-matrix* $U \colon X \times Y \to \mathbb{L}$ *and a* $Y \times Z$*-matrix* $V \colon Y \times Z \to \mathbb{L}$*, we define the* $\otimes$*-multiplication of* U *and* V *as follows:*

$$U \otimes V \colon X \times Z \to \mathbb{L}$$

$$(U \otimes V)(x,z) = \prod_{y \in Y} \big(U(x,y) \to_{\mathbb{L}} V(y,z) \big) \ .$$

Theorem 5. *Let* $R : X \times Y \to L$ *be a conditional relation between a pair of LaTSs* (X, A, α) *and* (Y, A, β)*. Then,* $F(R) = \prod_{a \in A} ((\alpha_a \otimes (R \cdot {\beta_a}^T)) \sqcap (\beta_a \otimes (\alpha_a \cdot R)^T)^T)$*, where* A^T *denotes the transpose of a matrix* A*.*

We end this section by making an observation on LaTSs over a Boolean algebra. In a Boolean algebra, it is well-known that the residuum operator can be replaced by the negation and join operators. Thus, in this case, using only the standard matrix multiplication and (componentwise) negation we get $U \otimes V = \neg(U \cdot (\neg V))$ which further simplifies $F(R)$ as:

$$F(R) = \prod_{a \in A} \big(\neg(\alpha_a \cdot \neg(R \cdot \beta_a^T)) \sqcap \neg(\neg(\alpha_a \cdot R) \cdot \beta_a^T) \big) \ .$$

This reduction is especially relevant to software product lines with no upgrade features.

VI. Application and Implementation

A. Featured Transition Systems

A Software Product Line (SPL) is commonly described as "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets [artifacts] in a prescribed way" [10]. The idea of designing a set of software systems that share common functionalities in a collective way is becoming prominent in the field of software engineering (cf. [21]). In this section we show that a featured transition system (FTS) [12] – a well-known formal model that is expressive enough to specify an SPL – is a special instance of a CTS.

Definition 11. *A* featured transition system *(FTS) over a finite set of* features N *is a tuple* $\mathcal{F} = (X, A, T, \gamma)$*, where* X *is a finite set of states,* A *is a finite set of actions and* $T \subseteq X \times A \times X$ *is the set of transitions. Finally,* $\gamma : T \to \mathbb{B}(N)$ *assigns a Boolean expression over* N *to each transition.*

FTSs are often accompanied by a so-called *feature diagram* [7], [9], [11], a Boolean expression $d \in \mathbb{B}(N)$ that specifies admissible feature combinations. Given a subset of features

$C \subseteq N$ (called *configuration* or *product*) such that $C \models d$ and an FTS $\mathcal{F} = (X, A, T, \gamma)$, a state $x \in X$ can perform an a-transition to a state $y \in X$ in the configuration C, whenever $(x, a, y) \in T$ and $C \models \gamma(x, a, y)$.

It is easy to see that an FTS is a CTS, where the conditions are subsets of N satisfying d with the discrete order. Moreover, an FTS is a special case of an LaTS due to Theorem 2 and $\mathcal{O}(\llbracket d \rrbracket, =) = \mathcal{P}(\llbracket d \rrbracket)$. Given an FTS $\mathcal{F} = (X, A, T, \gamma)$ and a feature diagram d, then the corresponding LaTS is (X, A, α) with $\alpha(x, a, y) = \llbracket \gamma(x, a, y) \wedge d \rrbracket$, if $(x, a, y) \in T$; $\alpha(x, a, y) = \emptyset$, if $(x, a, y) \notin T$.

Furthermore, we can extend the notion of FTSs by fixing a subset of upgrade features $U \subseteq N$ that induces the following ordering on configurations $C, C' \in \llbracket d \rrbracket$:

$$C \leq C' \iff \forall f \in U (f \in C' \Rightarrow f \in C) \wedge \forall f \in (N \backslash U) (f \in C' \iff f \in C).$$

Intuitively, the configuration C can be obtained from C' by "switching" on one or several upgrade features $f \in U$. Notice that it is this upgrade ordering on configurations which gives rise to the partially ordered set of conditions in the definition of a CTS. Hence, in the following we will consider the lattice $\mathcal{O}(\llbracket d \rrbracket, \leq)$ (i.e., the set of all downward-closed subsets of $\llbracket d \rrbracket$).

B. BDD-Based Representation

In this section, we discuss our implementation of lattice bisimulation using a special form of binary decision diagrams (BDDs) called *reduced and ordered binary decision diagrams* (ROBDDs). Our implementation can handle adaptive SPLs that allow upgrade features, using finite distributive lattices. Note that non-adaptive SPLs based on Boolean algebras are a special case. BDD-based implementations of FTSs without upgrades have already been mentioned in [8], [12].

A *binary decision diagram* (BDD) is a rooted, directed, and acyclic graph which serves as a representation of a Boolean function. Every BDD has two distinguished terminal nodes 1 and 0, representing the logical constants *true* and *false*. The inner nodes are labelled by the atomic propositions of a Boolean expression $b \in \mathbb{B}(N)$ represented by the BDD, such that on each path from the root to the terminal nodes, every variable of the Boolean formula occurs at most once. Each inner node has exactly two distinguished outgoing edges called *high* and *low* representing the case that the atomic proposition of the inner node has been set to *true* or *false*. Given a BDD for a Boolean expression $b \in \mathbb{B}(N)$ and a configuration $C \subseteq N$ (representing an evaluation of the atomic propositions), we can check whether $C \models b$ by following the path from the root node to a terminal node. At a node labelled $f \in N$ we go to the *high*-successor if $f \in C$ and to the *low*-successor if $f \notin C$. If we arrive at the terminal node labelled 1 we have established that $C \models b$, otherwise $C \not\models b$.

We use a special class of BDDs called ROBDDs (see [2] for more details) in which the order of the variables occurring in the BDD is fixed and redundancy is avoided. If both the child nodes of a parent node are identical, the parent node is dropped from the BDD and isomorphic parts of the BDD are merged. The advantage of ROBDDs is that two equivalent Boolean formulae are represented by exactly the same ROBDD (if the order of the variables is fixed). Furthermore, there are polynomial-time implementations for the basic operations – negation, conjunction, and disjunction. These are however sensitive to the ordering of atomic propositions and an exponential blowup cannot be ruled out, but often it can be avoided.

Consider a Boolean expression b with $\llbracket b \rrbracket = \{\emptyset, \{f_2, f_3\}, \{f_0, f_1\}, \{f_0, f_1, f_2, f_3\}\}$ and the ordering on the atomic propositions as f_0, f_1, f_2, f_3. Figure 3 shows the corresponding ROBDD representation for b, where the inner nodes, terminal nodes, and high (low) edges are depicted as circles, rectangles, and solid (dashed) lines, respectively.

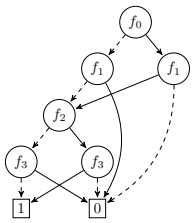

Fig. 3. BDD for b.

Formally, an ROBDD b over a set of features N is an expression in one of the following forms: 0, or 1, or (f, b_1, b_0). Here, $0, 1$ denote the two terminal nodes and the triple (f, b_1, b_0) denotes an inner node with variable $f \in N$ and b_0, b_1 as the *low*- and *high*-successors, respectively. If $b = (f, b_1, b_0)$, we write $root(b) = f$, $high(b) = b_1$, and $low(b) = b_0$.

Note that the elements of the Boolean algebra $\mathcal{P}(\mathcal{P}(N))$ *correspond exactly* to ROBDDs over N. We now discuss how ROBDDs can be used to specify and manipulate elements of the lattice $\mathcal{O}(\llbracket d \rrbracket, \leq)$. In particular, computing the infimum (conjunction) and the supremum (disjunction) in the lattice $\mathcal{O}(\llbracket d \rrbracket, \leq)$ is standard, since this lattice can be embedded into $\mathcal{P}(\mathcal{P}(N))$ and the infimum and supremum operations coincide in both structures. Thus, it remains to characterize the lattice elements and the residuum operation.

We say that an ROBDD b is *downward-closed* w.r.t. $\leq$ (or simply, downward-closed) if the set of configurations $\llbracket b \rrbracket$ is downward-closed w.r.t. $\leq$. The following lemma characterises when an ROBDD b is downward-closed. It follows from the fact that $F \in \mathcal{P}(\mathcal{P}(N))$ is downward-closed if and only if for all $C \in F, f \in U$ we have $C \cup \{f\} \in F$.

Lemma 3. *An ROBDD is downward-closed if and only if for each node labelled with a upgrade feature, the* low*-successor implies the* high*-successor.*

Next, we compute the residuum in $\mathcal{O}(\llbracket d \rrbracket, \leq)$ by using the residuum operation of the Boolean algebra $\mathcal{P}(\mathcal{P}(N))$. For this, we first describe how to approximate an element of the Boolean algebra (represented as an ROBDD) in the lattice $\mathcal{O}(\mathcal{P}(N), \leq)$.

In the above algorithm, for each non-terminal node that carries a label in U (line 3), we replace the *high*-successor with the conjunction of the *low* and the *high*-successor using the procedure described above. Since this might result in a BDD that is not reduced, we apply the $build$ procedure appropriately, which simply transforms a given ordered BDD into an ROBDD. The result of the algorithm $\llbracket b \rrbracket$ coincides with the approximation $\lfloor b \rfloor$ of the ROBDD b seen as an element of the Boolean algebra $\mathcal{P}(\mathcal{P}(N))$ (Definition 3).

Algorithm 1 Approximation $\lfloor\!\lfloor b \rfloor\!\rfloor$ of an ROBDD b in the lattice $\mathcal{O}(\mathcal{P}(N), \leq)$

Input: An ROBDD b over a set of features N and a set of upgrade features $U \subseteq N$.
Output: An ROBDD $\lfloor\!\lfloor b \rfloor\!\rfloor$, which is the best approximation of b in the lattice.

```
1: procedure ⌊⌊b⌋⌋
2:     if b is a leaf then return b
3:     else if root(b) ∈ U then return
4:         build(root(b), ⌊⌊high(b)⌋⌋, ⌊⌊high(b)⌋⌋ ∧ ⌊⌊low(b)⌋⌋)
5:     else return build(root(b), ⌊⌊high(b)⌋⌋, ⌊⌊low(b)⌋⌋)
6:     end if
7: end procedure
```

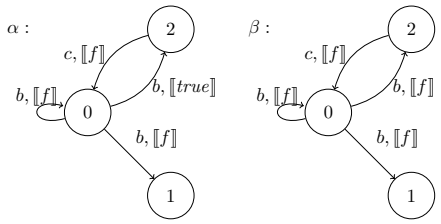

Fig. 4. Components for α and β, where f is viewed as a Boolean expression indicating the presence of feature f.

Lemma 4. *For an ROBDD b, $\lfloor\!\lfloor b \rfloor\!\rfloor$ is downward-closed. Furthermore, $\lfloor\!\lfloor b \rfloor\!\rfloor \models b$ and there is no other downward-closed ROBDD b' such that $\lfloor\!\lfloor b \rfloor\!\rfloor \models b' \models b$. Hence $\lfloor\!\lfloor b \rfloor\!\rfloor = \lfloor b \rfloor$.*

For each node in the BDD we compute at most one supremum, which is quadratic. Hence the entire runtime of the approximation procedure is at most cubic. Finally, we discuss how to compute the residuum in $\mathcal{O}(\llbracket d \rrbracket, \leq)$.

Proposition 3. *Let b_1, b_2 be two ROBDD which represent elements of $\mathcal{O}(\llbracket d \rrbracket, \leq)$, i.e., b_1, b_2 are both downward-closed and $b_1 \models d$, $b_2 \models d$. (i) $\lfloor \neg b_1 \vee b_2 \vee \neg d \rfloor \wedge d$ is the residuum $b_1 \to b_2$ in the lattice $\mathcal{O}(\llbracket d \rrbracket, \leq)$. (ii) If d is downward-closed, then this simplifies to $b_1 \to b_2 = \lfloor \neg b_1 \vee b_2 \rfloor \wedge d$.*

Here, $\neg$ is the negation in the Boolean algebra $\mathcal{P}(\mathcal{P}(N))$.

C. Implementation and Runtime Results

We have implemented an algorithm that computes the lattice bisimulation relation based on the matrix multiplication (see Theorem 5) in a generic way. Specifically, this implementation is independent of how the irreducible elements are encoded, ensuring that no implementation details of operations such as matrix multiplication can interfere with the runtime results. For our experiments we instantiated it in two possible ways: with bit vectors representing feature combinations and with ROBDDs as outlined above. Our results show a significant advantage when we use BDDs to compute lattice bisimilarity. The implementation is written in C# and uses the CUDD package by Fabio Somenzi via the interface PAT.BDD [22].

To show that the use of BDDs can potentially lead to an exponential gain in speed when compared to the naive bit-vector implementation, we executed the algorithm on a family of increasingly larger LaTSs over an increasingly larger number of features, where all features are upgrade features. Let F be a set of features. Our example contains, for each feature $f \in F$, one disconnected component in both LaTSs that is depicted in Figure 4: the component for α on the left, the component for β is on the right. The only difference between the two is in the guard of the transition from state 0 to state 2.

The quotient of the times taken without BDDs and with BDDs is growing exponentially by a factor of about 2 for each additional feature (see the table in Appendix B). Due to fluctuations, an exact rate cannot be given. By the eighteenth iteration (i.e. 18 features and copies of the basic component), the implementation using BDDs needed 17 seconds, whereas the version without BDDs took more than 96 hours. The nineteenth iteration exceeded the memory for the implementation without BDDs, but terminated within 22 seconds with BDDs.

VII. Conclusion, Related Work, and Future Work

In this paper, we endowed CTSs with an order on conditions to model systems whose behaviour can be upgraded by replacing the current condition by a smaller one. Corresponding verification techniques based on behavioural equivalences can be important for SPLs where an upgrade to a more advanced version of the same software should occur without unexpected behaviour. To this end, we proposed an algorithm, based on matrix multiplication, that allows to compute the greatest bisimulation of two given CTSs. Interestingly, the duality between lattices and downward-closed sets of posets, as well as the embedding into a Boolean algebra proved to be fruitful when developing it and proving its correctness.

There are two ways in which one can extend CTSs as a specification language: first, in some cases it makes sense to specify that an advanced version offers improved transitions with respect to a basic version. For instance, in our running example, allowing the router to send unencrypted messages in an unsafe environment is superfluous because the advanced version always has the encryption feature. Such a situation can be modelled in a CTS by adding a precedence relation over the set of actions, leading to the deactivation of transitions, which is worked out in Appendix C. The second question is how to incorporate downgrades: one solution could be to work with a pre-order on conditions, instead of an order. This simply means that two conditions $\varphi \neq \psi$ with $\varphi \leq \psi$, $\psi \leq \varphi$ can be merged since they can be exchanged arbitrarily. Naturally, one could study more sophisticated notions of upgrade and downgrade in the context of adaptivity.

As for the related work on adaptive SPLs, literature can be grouped into either empirical or formal approaches; however, given the nature of our work, below we rather concentrate only on the formal ones [6], [11], [15], [23].

Cordy et al. [11] model an adaptive SPL using an FTS which encodes not only a product's transitions, but also how some of the features may change via the execution of a transition. In contrast, we encode adaptivity by requiring a partial order on the products of an SPL and its effect on behaviour evolution

by the monotonicity requirement on the transition function. Moreover, instead of studying the model checking problem as in [11], our focus was on bisimilarity between adaptive SPLs.

In [6], [15], [20], alternative ways to model adaptive SPLs by using the synchronous parallel composition on two separate computational models is presented. Intuitively, one models the static aspect of an SPL, while the other focuses on adaptivity by specifying the dynamic (de)selection of features. For instance, Dubslaff et al. [15] used two separate Markov decision processes (MDP) to model an adaptive SPL. They modelled the core behaviour in an MDP called *feature module*; while dynamic (de)activation of features is modelled separately in a MDP called *feature controller*. In retrospect, our work shows that for monotonic upgrades it is possible to *compactly* represent an adaptive SPL over one computational model (CTSs in our case) rather than a parallel composition of two.

In [23], a process calculus QFLan motivated by concurrent constraint programming was developed. Thanks to an in-built notion of a store, various aspects of an adaptive SPL such as (un)installing a feature and replacing a feature by another feature can be modelled at run-time by operational rules. Although QFLan has constructs to specify quantitative constraints in the spirit of [15], their aim is to obtain statistical evidence by performing simulations.

Behavioural equivalences such as (bi)simulation relations have already been studied in the literature of traditional SPLs. In [12], the authors proposed a definition of simulation relation between any two FTSs (without upgrades) to combat the state explosion problem by establishing a simulation relation between a system and its refined version. In contrast, the authors in [3] used simulation relations to measure the discrepancy in behaviour caused by feature interaction, i.e., whether a feature that is correctly designed in isolation works correctly when combined with the other features or not.

(Bi)simulation relations on lattice Kripke structures were also studied in [19], but in a very different context (in model-checking rather than in the analysis of adaptive SPLs). Disregarding the differences between transition systems and Kripke structures (i.e., forgetting the role of atomic propositions), the definition of bisimulation in [19] is quite similar to our Definition 9 (another similar formula occurs in [12]). However, in [19] the stronger assumption of finite distributive de Morgan algebras is used, the results are quite different and symbolic representations via BDDs are not taken into account. Moreover, representing the lattice elements and computing residuum over them using the BDDs is novel in comparison with [12], [19].

Lastly, Fitting [16] studied bisimulation relations in the setting of unlabelled transition systems and gave an elegant characterisation of bisimulation when transition systems and the relations over states are viewed as matrices. By restricting ourselves to LaTSs over Boolean algebras and fixing our alphabet to be a singleton set, we can establish the following correspondence between Fitting's formulation of bisimulation and lattice bisimulation (see Appendix A-A for the proof).

Theorem 6. *Let (X, α) be a LaTS over an atomic Boolean algebra $\mathbb{B}$. Then, a conditional relation $R : X \times X \to \mathbb{B}$ is a lattice bisimulation for α if and only if $R \cdot \alpha \sqsubseteq \alpha \cdot R$ and $R^T \cdot \alpha \sqsubseteq \alpha \cdot R^T$. Here we interpret α as a matrix of type $X \times X \to \mathbb{L}$ by dropping the occurrence of action labels.*

In hindsight, we are treating general distributive lattices that allow us to conveniently model and reason about upgrades.

Current and future work: In the future we plan to obtain runtime results for systems of varying sizes. In particular, we are interested in real-world applications in the field of SPLs, together with other applications, such as modelling transition systems with access rights or deterioration.

On the more theoretical side of things, we have worked out the coalgebraic concepts for CTSs [5] and compared the matrix multiplication algorithm to the final chain algorithm presented in [1], when applied to CTSs.

Acknowledgements: We thank Filippo Bonchi and Mathias Hülsbusch for interesting discussions on earlier drafts.

REFERENCES

[1] J. Adámek, F. Bonchi, M. Hülsbusch, B. König, S. Milius, and A. Silva. A coalgebraic perspective on minimization and determinization. In *Proc. of FOSSACS '12*, pages 58–73. Springer, 2012. LNCS/ARCoSS 7213.

[2] H. R. Andersen. An introduction to binary decision diagrams. *Course Notes*, 1997.

[3] J. M. Atlee, U. Fahrenberg, and A. Legay. Measuring behaviour interactions between product-line features. In *Proc. of Formalise '15*, pages 20–25, Piscataway, NJ, USA, 2015. IEEE Press.

[4] R. Belohlavek and J. Konecny. Row and column spaces of matrices over residuated lattices. *Fundam. Inf.*, 115(4):279–295, December 2012.

[5] Harsh Beohar, Barbara König, Sebastian Küpper, and Alexandra Silva. A coalgebraic treatment of conditional transition systems with upgrades. arXiv:1612.05002, submitted to LMCS.

[6] P. Chrszon, C. Dubslaff, S. Klüppelholz, and C. Baier. Family-based modeling and analysis for probabilistic systems – featuring ProFeat. In *Proc. of FASE '16*, pages 287–304. Springer, 2016. LNCS 9633.

[7] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin. Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. *IEEE Trans. Softw. Eng.*, 39(8):1069–1089, August 2013.

[8] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay. Symbolic model checking of software product lines. In *Proc. of ICSE '11*. ACM, 2011.

[9] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *Proc. of ICSE '10*. ACM, 2010.

[10] P. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, USA, 2001.

[11] M. Cordy, A. Classen, P. Heymans, A. Legay, and P.-Y. Schobbens. Model checking adaptive software with featured transition systems. In *Assurances for Self-Adaptive Systems*, pages 1–29. Springer, 2013.

[12] M. Cordy, A. Classen, G. Perrouin, P.-Y. Schobbens, P. Heymans, and A. Legay. Simulation-based abstractions for software product-line model checking. In *Proc. of ICSE '12*, pages 672–682. IEEE, 2012.

[13] B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 2002.

[14] M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata*. Springer, 2009.

[15] C. Dubslaff, S. Klüppelholz, and C. Baier. Probabilistic model checking for energy analysis in software product lines. In *Proc. of MODULARITY '14*, pages 169–180. ACM, 2014.

[16] M. Fitting. Bisimulations and boolean vectors. In *Advances in Modal Logic*, volume 4, pages 97–126. World Scientific Publishing, 2002.

[17] A. Gruler, M. Leucker, and K. Scheidemann. Modeling and model checking software product lines. In *Proc. of FMOODS'08*, pages 113–131. Springer, 2008. LNCS 5051.

[18] L. J. Kohout and W. Bandler. Relational-product architectures for information processing. *Inf. Sci.*, 37(1-3):25–37, December 1985.

[19] O. Kupferman and Y. Lustig. Latticed simulation relations and games. *Int. Journal of Found. of Computer Science*, 21(02):167–189, 2010.
[20] M. Lochau, J. Bürdek, S. Hölzle, and A. Schürr. Specification and automated validation of staged reconfiguration processes for dynamic software product lines. *Software & Sys. Modeling*, 16(1):125–152, 2017.
[21] A. Metzger and K. Pohl. Software product line engineering and variability management: Achievements and challenges. In *Proc. of FOSE '14*, pages 70–84, New York, NY, USA, 2014. ACM.
[22] T. K. Nguyen, J. Sun, Y. Liu, J. S. Dong, and Y. Liu. Improved BDD-based discrete analysis of timed systems. In *Proc. of FM '12*, volume 7436 of *LNCS*, pages 326–340. Springer, 2012.
[23] M. H. ter Beek, A. Legay, A. Lluch Lafuente, and A. Vandin. Statistical analysis of probabilistic models of software product lines with quantitative constraints. In *Proc. of SPLC '15*, pages 11–15. ACM, 2015.
[24] M.H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints. *JLAMP*, 85(2):287 – 315, 2016.

APPENDIX A
PROOFS

Here we give proofs for all lemmas and propositions for which we have omitted the proofs in the article.

A. Proofs concerning Lattices and Lattice Transition Systems

Proof of Lemma 1

Proof: Let $\ell, m \in \mathbb{B}$. Monotonicity of the approximation is immediate from the definition.

We next show $\lfloor \ell \sqcap m \rfloor \sqsupseteq \lfloor \ell \rfloor \sqcap \lfloor m \rfloor$: by definition we have $\lfloor \ell \rfloor \sqsubseteq \ell$, $\lfloor m \rfloor \sqsubseteq m$ and hence $\lfloor \ell \rfloor \sqcap \lfloor m \rfloor \sqsubseteq \ell \sqcap m$. Since $\lfloor \ell \sqcap m \rfloor$ is the best approximation of $\ell \sqcap m$ and $\lfloor \ell \rfloor \sqcap \lfloor m \rfloor$ is one approximation, the inequality follows.

In order to prove $\lfloor \ell \sqcap m \rfloor \sqsubseteq \lfloor \ell \rfloor \sqcap \lfloor m \rfloor$ observe that $\lfloor \ell \rfloor \sqsupseteq \lfloor \ell \sqcap m \rfloor$ and $\lfloor m \rfloor \sqsupseteq \lfloor \ell \sqcap m \rfloor$ by monotonicity of the approximation. Hence $\lfloor \ell \sqcap m \rfloor$ is a lower bound of $\lfloor \ell \rfloor, \lfloor m \rfloor$, which implies $\lfloor \ell \rfloor \sqcap \lfloor m \rfloor \sqsupseteq \lfloor \ell \rfloor \sqcap \lfloor m \rfloor$.

Now let $\ell, m \in \mathbb{L}$. Recall the definitions $\lfloor \ell \sqcup \neg m \rfloor = \bigsqcup \{x \in \mathbb{L} \mid x \sqsubseteq \ell \sqcup \neg m\}$ and $m \to_{\mathbb{L}} \ell = \bigsqcup \{x \in \mathbb{L} \mid m \sqcap x \sqsubseteq \ell\}$. We will prove that both sets are equal.

Assume $x \in \mathbb{L}$ with $x \sqsubseteq \ell \sqcup \neg m$, then $m \sqcap x \sqsubseteq m \sqcap (\ell \sqcup \neg m) = (m \sqcap \ell) \sqcup (m \sqcap \neg m) = (m \sqcap \ell) \sqcup 0 = m \sqcap \ell \sqsubseteq \ell$. For the other direction assume $m \sqcap x \sqsubseteq \ell$, then $\ell \sqcup \neg m \sqsupseteq (m \sqcap x) \sqcup \neg m = (m \sqcup \neg m) \sqcap (x \sqcup \neg m) = 1 \sqcap (x \sqcup \neg m) = x \sqcup \neg m \sqsupseteq x$. ∎

Proof of Theorem 2

Proof: Given a set X, a partially ordered set $(\Phi, \leq)$, and $\mathbb{L} = \mathcal{O}(\Phi)$, we define an isomorphism between the sets $(\Phi \xrightarrow{\text{mon.}} \mathcal{P}(X))^{X \times A}$ and $\mathcal{O}(\Phi)^{X \times A \times X}$. Consider the following function mappings $\eta : (\Phi \xrightarrow{\text{mon.}} \mathcal{P}(X))^{X \times A} \to \mathcal{O}(\Phi)^{X \times A \times X}$, $f \mapsto \eta(f)$ and $\eta' : \mathcal{O}(\Phi)^{X \times A \times X} \to (\Phi \xrightarrow{\text{mon.}} \mathcal{P}(X))^{X \times A}$, $\alpha \mapsto \eta'(\alpha)$ defined as:

$$\eta(f)(x, a, x') = \{\varphi \in \Phi \mid x' \in f_\varphi(x, a)\},$$
$$\eta'(\alpha)(x, a)(\varphi) = \{x' \mid \varphi \in \alpha(x, a, x')\}.$$

Downward-closed Let $\varphi \in \eta(f)(x, a, x')$ and $\varphi' \leq \varphi$. By using these facts in the definition of $f_{\varphi'}$ we find $x' \in f_{\varphi'}(x, a)$, i.e., $\varphi' \in \eta(f)(x, a, x')$.

Anti-monotonicity Let $\varphi \leq \varphi'$ and $x' \in \eta'(\alpha)(x, a)(\varphi')$. Then by the above construction we find $\varphi' \in \alpha(x, a, x')$. And by downward-closedness of $\alpha(x, a, x')$ we get $\varphi \in \alpha(x, a, x')$, i.e., $x' \in \eta'(\alpha)(x, a)(\varphi)$.

Now it suffices to show that η, η' are inverse of each other because by the uniqueness of inverses we then have $\eta' = \eta^{-1}$. We only give the proof of $\eta' \circ \eta = \mathsf{id}$, the proof of the other case ($\eta \circ \eta' = \mathsf{id}$) is similar. The former follows from the following observation:

$$x' \in f(x, a)(\varphi) \Leftrightarrow x' \in f_\varphi(x, a) \Leftrightarrow \varphi \in \eta(f)(x, a, x')$$
$$\Leftrightarrow x' \in \eta'(\eta(f))(x, a)(\varphi) \ .$$

∎

Proof of Theorem 3

Proof: Let $x \in X, y \in Y$ be any two states in CTSs (LaTSs) $(X, A, f), (Y, A, g)$ over the conditions Φ ($(X, A, \alpha), (Y, A, \beta)$ over the lattice $\mathcal{O}(\Phi \leq)$), respectively.
($\Leftarrow$) Let $\varphi \in \Phi$ be a condition and let R be a lattice bisimulation relation such that $\varphi \in R(x, y)$. Then, we can construct a family of relations $R_{\varphi'}$ (for $\varphi' \leq \varphi$) as follows: $x R_{\varphi'} y \Leftrightarrow \varphi' \in R(x, y)$. For all other φ' we set $R_{\varphi'} = \emptyset$. The downward-closure of $R(x, y)$ ensures that $R_{\varphi''} \subseteq R_{\varphi'}$ (for $\varphi', \varphi'' \leq \varphi$), whenever $\varphi' \leq \varphi''$.

Thus, it remains to show that every relation $R_{\varphi'}$ is a bisimulation. Let $x R_{\varphi'} y$ and $x' \in f_{\varphi'}(x, a)$. Then, $x \xrightarrow{a, \downarrow \varphi'} x'$. Since $\downarrow \varphi'$ is an irreducible in the lattice, $\downarrow \varphi' \subseteq R(x, y)$ and R is a lattice bisimulation, we find $y \xrightarrow{a, \downarrow \varphi'} y'$ and $\downarrow \varphi' \subseteq R(x', y')$, which implies $\varphi' \in R(x', y')$. That is, $y' \in g_{\varphi'}(y, a)$ and $x' R_{\varphi'} y'$. Likewise, the remaining symmetric condition of bisimulation can be proved.

($\Rightarrow$) Let $\sim_\varphi$ be a conditional bisimulation between the CTSs $(X, A, f), (Y, A, g)$, for some $\varphi \in \Phi$. Then, construct a conditional relation: $R(x, y) = \{\varphi \mid x \sim_\varphi y\}$. Clearly, the set $R(x, y)$ is a downward-closed subset of Φ due to Definition 5(ii); i.e., an element in the lattice $\mathcal{O}(\Phi)$. Next, we show that R is a lattice bisimulation.

Let $x \xrightarrow{a, \downarrow \varphi'} x'$ and $\downarrow \varphi' \subseteq R(x, y)$. This implies $x' \in f_{\varphi'}(x, a)$ and $\varphi' \in R(x, y)$, hence $x \sim_{\varphi'} y$. So using the transfer property of traditional bisimulation, we obtain $y' \in g_{\varphi'}(y, a)$ and $x' \sim_{\varphi'} y'$. That is, $y \xrightarrow{b, \downarrow \varphi'} y'$ and $\varphi' \in R(x', y')$, which implies $\downarrow \varphi' \subseteq R(x', y')$. Likewise the symmetric condition of lattice bisimulation can be proved. ∎

Proof of Lemma 2

Proof: Let $x, x' \in X, a \in A, y \in Y$ and $\ell \in \mathcal{J}(\mathbb{L})$ such that $\ell \sqsubseteq \bigsqcup_{i \in I} R_i(x, y)$ and $x \xrightarrow{a, \ell} x'$. Then, there is an index $i \in I$ such that $\ell \sqsubseteq R_i(x, y)$, since ℓ is an irreducible. Thus, there is a y' such that $y \xrightarrow{a, \ell} y'$ and $\ell \sqsubseteq R_i(x', y') \sqsubseteq \bigsqcup_{i \in I} R_i(x', y')$. Likewise, the symmetric condition when a transition emanates from y can be proved. ∎

Proof of Theorem 4

Proof:
($\Leftarrow$) Let $R : X \times Y \to \mathbb{L}$ be a conditional relation over a pair of LaTSs $(X, A, \alpha), (Y, A, \beta)$ such that $R \sqsubseteq F(R)$. Next, we show that R is a lattice bisimulation. For this purpose, let $\ell \in \mathcal{J}(\mathbb{L})$, $a \in A$. Furthermore, let $x \xrightarrow{a, \ell} x'$ (which implies $\ell \sqsubseteq$

$\alpha(x,a,x'))$ and $\ell \sqsubseteq R(x,y)$. From $R(x,y) \sqsubseteq F_1(R)(x,y)$ we infer $\ell \sqsubseteq F_1(R)(x,y)$. This means that $\ell \sqsubseteq \alpha(x,a,x') \to \big(\bigsqcup_{y' \in Y}(\beta(y,a,y') \sqcap R(x',y'))\big)$. Since $\ell_1 \sqcap (\ell_1 \to \ell_2) \sqsubseteq \ell_2$ we can take the infimum with $\alpha(x,a,x')$ on both sides and obtain $\ell \sqsubseteq \ell \sqcap \alpha(x,a,x') \sqsubseteq \bigsqcup_{y' \in Y}(\beta(y,a,y') \sqcap R(x',y'))$ (the first inequality holds since $\ell \sqsubseteq \alpha(x,a,x')$). Since ℓ is irreducible there exists a y' such that $\ell \sqsubseteq \beta(y,a,y')$, i.e., $y \xrightarrow{a,\ell} y'$, and $\ell \sqsubseteq R(x',y')$.

Likewise, the remaining condition when a transition emanates from y can be proved.

($\Rightarrow$) Let $R : X \times Y \to \mathbb{L}$ be a lattice bisimulation on $(X,A,\alpha),(Y,A,\beta)$. Then, we need to show that $R \sqsubseteq F(R)$, i.e., $R \sqsubseteq F_1(R)$ and $R \sqsubseteq F_2(R)$. We will only give the proof of the former inequality, the proof of the latter is analogous. To show $R \sqsubseteq F_1(R)$, it is sufficient to prove $\ell \sqsubseteq R(x,y) \Rightarrow \ell \sqsubseteq F_1(R)(x,y)$, for all $x \in X, y \in Y$ and all irreducibles ℓ. So let $\ell \sqsubseteq R(x,y)$, for some x,y. Next, simplify $F_1(R)$ as follows:

$$
\begin{aligned}
&F_1(R)(x,y)\\
&= \prod_{a,x'}(\alpha(x,a,x') \to \bigsqcup_{y' \in Y}(\beta(y,a,y') \sqcap R(x',y')))\\
&= \prod_{a,x'} \Big\lfloor \bigsqcup_{y' \in Y}(\beta(y,a,y') \sqcap R(x',y')) \sqcup \neg\alpha(x,a,x') \Big\rfloor \quad \text{(L. 1)}\\
&= \prod_{a,x'} \bigsqcup \{m \in \mathbb{L} \mid m \sqsubseteq \bigsqcup_{y' \in Y}(\beta(y,a,y') \sqcap R(x',y'))\\
&\quad \sqcup \neg\alpha(x,a,x')\}.
\end{aligned}
$$

Thus, it is sufficient to show that $\ell \sqsubseteq \bigsqcup_{y' \in Y}(\beta(y,a,y') \sqcap R(x',y')) \sqcup \neg\alpha(x,a,x')$, for any $a \in A, x' \in X$. We do this by distinguishing the following cases: either $\ell \sqsubseteq \neg\alpha(x,a,x')$ or $\ell \sqsubseteq \alpha(x,a,x')$. If the former holds (which corresponds to the case where there is no a-labelled transition under ℓ), then the result holds trivially. So assume $\ell \sqsubseteq \alpha(x,a,x')$. Recall, from above, that $\ell \sqsubseteq R(x,y)$ and R is a lattice bisimulation. Thus, there is a $y' \in Y$ such that $\ell \sqsubseteq \beta(y,a,y')$ and $\ell \sqsubseteq R(x',y')$; hence,

$$\ell \sqsubseteq \bigsqcup_{y' \in Y}(\beta(y,a,y') \sqcap R(x',y')) \sqcup \neg\alpha(x,a,x') \ .$$

■

Proof of Proposition 2

Proof:

(i) This follows directly from Lemma 1, allowing to move the approximations to the inside towards the implications and Lemma 1, allowing to approximate the implication in $\mathbb{B}$ via the implication in $\mathbb{L}$.

(ii) If R is a bisimulation in $\mathbb{L}$, then $F_{\mathbb{L}}(R) \sqsupseteq R$. Since by definition, $\lfloor Q \rfloor \sqsubseteq Q$ for all conditional relations Q and we have shown in (i) that $F_{\mathbb{L}}(R) = \lfloor F_{\mathbb{B}}(R) \rfloor$, we can conclude $F_{\mathbb{B}}(R) \sqsupseteq \lfloor F_{\mathbb{B}}(R) \rfloor = F_{\mathbb{L}}(R) \sqsupseteq R$. Thus, R is a $\mathbb{B}$-bisimulation.

(iii) Clearly $R \sqsubseteq F_{\mathbb{B}}(R)$ because R is a $\mathbb{B}$-bisimulation. Since R has exclusively entries from $\mathbb{L}$, $F_{\mathbb{B}}(R) = \lfloor F_{\mathbb{B}}(R) \rfloor$, and finally (i) yields that $\lfloor F_{\mathbb{B}}(R) \rfloor = F_{\mathbb{L}}(R)$; thus, R is an $\mathbb{L}$-bisimulation.

■

Proof of Theorem 6

Proof:

($\Leftarrow$) Let $R : X \times X \to \mathbb{B}$ be a conditional relation satisfying $R \cdot \alpha \sqsubseteq \alpha \cdot R$ and $R^T \cdot \alpha \sqsubseteq \alpha \cdot R^T$. Then, we need to show that R is a lattice bisimulation. Let $x \xrightarrow{\ell} y$ such that $\ell \in \mathcal{J}(\mathbb{B})$ and $\ell \sqsubseteq R(x,x')$. Then, we find $\ell \sqsubseteq \alpha(x,y)$. That is,

$$
\begin{aligned}
\ell &\sqsubseteq R(x,x') \sqcap \alpha(x,y) = R^T(x',x) \sqcap \alpha(x,y)\\
&\sqsubseteq (R^T \cdot \alpha)(x',y) \sqsubseteq (\alpha \cdot R^T)(x',y).
\end{aligned}
$$

By expanding the last term from above, we find that $\ell \sqsubseteq \alpha(x',y') \sqcap R^T(y',y)$, for some y'. Thus, $\ell \sqsubseteq \alpha(x',y')$ (which implies $x' \xrightarrow{\ell} y'$) and $\ell \sqsubseteq R(y,y')$. Similarly, the remaining condition when the transition emanates from x' can be verified using $R \cdot \alpha \sqsubseteq \alpha \cdot R$.

($\Rightarrow$) Let $R : X \times X \to \mathbb{B}$ be a lattice bisimulation. Then, we only prove $R \cdot \alpha \sqsubseteq \alpha \cdot R$; the proof of $R^T \cdot \alpha \sqsubseteq \alpha \cdot R^T$ is similar. Note that, for any $x, y' \in X$, we know that the element $(R \cdot \alpha)(x,y')$ can be decomposed into a set of atoms, since $\mathbb{B}$ is an atomic Boolean algebra. Let $(R \cdot \alpha)(x,y') = \bigsqcup_i \ell_i$ for some index set I such that the ℓ_i are atoms or irreducibles in $\mathbb{B}$.

Furthermore, expanding the above inequality we get, for every $i \in I$ there is a state $y \in X$ such that $\ell_i \sqsubseteq R(x,y) \sqcap \alpha(y,y')$, since the ℓ_i are irreducibles. That is, for every $i \in I$ we have some state y such that $\ell_i \sqsubseteq R(x,y)$ and $\ell_i \sqsubseteq \alpha(y,y')$. Now using the transfer property of R we find some state x' such that $\ell_i \sqsubseteq \alpha(x,x')$ and $\ell_i \sqsubseteq R(x',y')$. Thus, for every $i \in I$ we find that $\ell_i \sqsubseteq (\alpha \cdot R)(x,y')$; hence, since $(\alpha \cdot R)(x,y')$ is an upper bound of all ℓ_i, $(R \cdot \alpha)(x,y') \sqsubseteq (\alpha \cdot R)(x,y')$. ■

B. Strategies for the Bisimulation Game

In the main text we claimed that there exists a winning strategy for Player 2 in the conditional bisimulation game if and only if the start states are conditionally bisimilar. In this section we will describe the strategy and prove that it is correct.

Lemma 5. *Given two CTSs (X,A,f), (Y,A,g) and an instance (x,y,φ) of a bisimulation game, then whenever $x \sim_\varphi y$, Player 2 has a winning strategy for (x,y,φ).*

Proof: The strategy of Player 2 can be directly derived from the family of CTS bisimulation relations $\{R_{\varphi'} \mid \varphi' \in \Phi\}$ where $(x,y) \in R_\varphi$. The strategy works inductively. Assume at any given point of time in the game, we have that the currently investigated condition is φ and $(x,y) \in R_\varphi$, where x and y are the currently marked states in X respectively Y. Then Player 1 upgrades to $\varphi' \leq \varphi$. Due to the condition on CTS bisimulations of reverse inclusion, we have $R_{\varphi'} \supseteq R_\varphi$, therefore $(x,y) \in R_{\varphi'}$. Then, when Player 1 makes a step $x \xrightarrow{a,\varphi'} x'$ in f, there must exist a transition $y \xrightarrow{a,\varphi'} y'$ in g such that $(x',y') \in R_{\varphi'}$ due to $R_{\varphi'}$ being a (traditional) bisimulation. Analogously if Player 1 chooses a transition

$y \xrightarrow{a,\varphi'} y'$ in g, there exists a transition $x \xrightarrow{a,\varphi'} x'$ in f for Player 2 such that $(x', y') \in R_{\varphi'}$. Hence, Player 2 will be able to react and establish the inductive condition again. In the beginning, the condition holds per definition. Thus, Player 2 has a winning strategy.

■

We will now prove the converse by explicitly constructing a winning strategy for Player 1 whenever two states are not in a bisimulation relation.

Lemma 6. *Given two CTSs A, B and an instance (x, y, φ) of a bisimulation game, then whenever $x \not\sim_\varphi y$, Player 1 has a winning strategy for (x, y, φ).*

Proof: We consider the LaTSs which correspond to the CTSs A, B and compute the fixpoint by using the matrix multiplication algorithm, obtaining a sequence $R_0, R_1, \ldots, R_n = R_{n+1} = \ldots$ of lattice-valued relations $R_i \colon X \times Y \to \mathcal{O}(\Phi, \leq)$. Note that instead of using exactly the matrix multiplication method, we can also use the characterisation of Definition 8: whenever there exists a transition $x \xrightarrow{a,\varphi} x'$, for which there is no matching transition with $y \xrightarrow{y,\varphi} y'$ with $\varphi \in R_{i-1}(x', y')$, the condition φ and all larger conditions $\varphi' \geq \varphi$ have to be removed from $R_{i-1}(x, y)$ in the construction of $R_i(x, y)$.

We will now define $M^{\varphi'}(x, y) = \max\{i \in \mathbb{N}_0 \mid \varphi' \in R_i(x, y)\}$, where $\max \mathbb{N}_0 = \infty$. An entry $M^{\varphi'}(x, y) = \infty$ signifies that $x \sim_{\varphi'} y$, whereas any other entry $i < \infty$ means that x, y were separated under condition φ' at step i and hence $x \not\sim_{\varphi'} y$.

Now assume we are in a game situation with game instance (x, y, φ) where Player 1 has to make a step. We will show that if $M^\varphi(x, y) = i < \infty$, Player 1 can choose an upgrade $\overline{\varphi} \leq \varphi$, an action $a \in A$ and a step $x \xrightarrow{a,\overline{\varphi}} x'$ (or $y \xrightarrow{a,\overline{\varphi}} y'$) such that independently of the choice of the corresponding state y', respectively x', which Player 2 makes, $M^{\overline{\varphi}}(x', y') < i$.

For each $\varphi' \leq \varphi$ compute

$$\begin{aligned}&\omega(\varphi')\\ =\min\quad &\{\min_{a,x'}\{\max_{y'}\{M_n^{\varphi'}(x', y') \mid y \xrightarrow{a,\varphi'} y'\} \mid x \xrightarrow{a,\varphi'} x'\},\\ &\{\min_{a,y'}\{\max_{x'}\{M_n^{\varphi'}(x', y') \mid x \xrightarrow{a,\varphi'} x'\} \mid y \xrightarrow{a,\varphi'} y'\}\}\end{aligned}$$

The formula can be interpreted as follows: The outer $\min$ corresponds to the choice of making a step in transition system A or B. The inner $\min$ corresponds to choosing the step that yields the best, i.e. lowest, guaranteed separation value and the $\max$ corresponds to the choice of Player 2 that yields the best, i.e. greatest, separation value for him.

Now choose a minimal condition $\overline{\varphi}$ such that $\omega(\overline{\varphi})$ is minimal for all $\varphi' \leq \varphi$. Player 1 now makes an upgrade from φ to $\overline{\varphi}$ and chooses a transition $x \xrightarrow{a,\overline{\varphi}} x'$ or $y \xrightarrow{a,\overline{\varphi}} y'$ such that the minimum in $\omega(\overline{\varphi})$ is reached. This means that Player 2 can only choose a corresponding successor state y' respectively x' such that $M^{\overline{\varphi}}(x', y') \leq \omega(\overline{\varphi})$.

Now it remains to be shown that $\omega(\overline{\varphi}) < i$, via contradiction: assume that $\omega(\overline{\varphi}) \geq i$. Since $\omega(\overline{\varphi})$ is minimal for all $\varphi' \leq \varphi$, we obtain $\omega(\varphi') \geq i$ for all $\varphi' \geq \varphi$. This implies that for each step $x \xrightarrow{a,\varphi'} x'$ there exists an answering step $y \xrightarrow{a,\varphi'} y'$ such that $M^{\varphi'}(x', y') \geq i$ (analogously for every step of y). The condition $M^{\varphi'}(x', y') \geq i$ is equivalent to $\varphi' \in R_i(x', y')$ and hence we can infer that $\varphi' \in R_{i+1}(x, y)$. This also holds for $\varphi' = \varphi$, which is a contradiction to $M^\varphi(x, y) = i$.

In order to conclude, take two states x, y and a condition φ such that $x \not\sim_\varphi y$. Then $M^\varphi(x, y) = i < \infty$ and the above strategy allows Player 1 to force Player 2 into a game instance $(x', y', \overline{\varphi})$ where $M^{\overline{\varphi}}(x', y') < M^\varphi(x, y)$. Whenever $M^\varphi(x, y) = 1$ Player 1 wins immediately, because then x allows a transition that y can not mimic or vice-versa, and Player 1 simply takes this transition. Therefore we have found a winning strategy for Player 1.

■

C. Proofs concerning ROBDDs

Proof of Lemma 3

Proof:

($\Rightarrow$) Assume that $low(n) \models high(n)$ for all nodes n of b.

Let $C' \in [\![b]\!]$ and $C \leq C'$. Without loss of generality we can assume that $C = C' \cup \{f\}$ for some $f \in U$. (The rest follows from transitivity.) For the configuration C' there exists a path in b that leads to 1. We distinguish the following two cases:

- There is no f-labelled node on the path. Then the path for C also leads to 1 and we have $C \in [\![b]\!]$.
- If there is an f-labelled node n on the path, then C' takes the *low*-successor, C the *high*-successor of this node. Since $low(n) \models high(n)$ we obtain $[\![low(n)]\!] \subseteq [\![high(n)]\!]$. Hence the remaining path for C, which contains the same features as the path for C', will also reach 1.

($\Leftarrow$) Assume by contradiction that $[\![b]\!]$ is downward-closed, but there exists a node n with $low(n) \not\models high(n)$ and $f = root(n) \in U$. Hence there must be a path from the *low*-successor that reaches 1, but does not reach 1 from the *high*-successor. Prefix this with the path that reaches n from the root of b.

In this way we obtain two configurations $C = C' \cup \{f\}$, i.e., $C \leq C'$, where $C' \in [\![b]\!]$, but $C \notin [\![b]\!]$. This is a contradiction to the fact that $[\![b]\!]$ is downward-closed.

■

Proof of Lemma 4

Proof:

- We show that $\lfloor\!\lfloor b \rfloor\!\rfloor$ as obtained by Algorithm 1 is downward-closed. This can be seen via induction over the number of different features occurring in the BDD b. If b only consists of a leaf node, then $\lfloor\!\lfloor b \rfloor\!\rfloor$ is certainly downward-closed. Otherwise, we know from the induction hypothesis that $\lfloor\!\lfloor high(b) \rfloor\!\rfloor$, $\lfloor\!\lfloor low(b) \rfloor\!\rfloor$ are downward-closed. If $root(b) \notin U$, then $\lfloor\!\lfloor b \rfloor\!\rfloor$ is downward-closed due to Lemma 3. If however $root(b) \in U$, then $\lfloor\!\lfloor high(b) \rfloor\!\rfloor \wedge \lfloor\!\lfloor low(b) \rfloor\!\rfloor$ is downward-closed (since

downward-closed sets are closed under intersection). Furthermore $\llbracket high(b) \rrbracket \wedge \llbracket low(b) \rrbracket \models \llbracket high(b) \rrbracket$, i.e., the new *low*-successor implies the *high*-successor. That means that the condition of Lemma 3 is satisfied at the root and elsewhere in the BDD and hence the resulting BDD $\llbracket b \rrbracket$ is downward-closed.

- First, from the construction where a *low*-successor is always replaced by a stronger *low*-successor, it is easy to see that $\llbracket b \rrbracket \models b$.
  We now show that there is no other downward-closed ROBDD b' such that $\llbracket b \rrbracket \models b' \models b$: Assume to the contrary that there exists such a downward-closed BDD b'. Hence there exists a configuration $C \subseteq N$ such that $C \not\models \llbracket b \rrbracket$, $C \models b'$, $C \models b$. Choose C maximal wrt. inclusion.
  Now we show that there exists a feature $f \in U$ such that $f \notin C$ and $C \cup \{f\} = C' \not\models b$. If this is the case, then $C' \leq C$ and $C' \not\models b'$, which is a contradiction to the fact that b' is downward-closed.
  Consider the sequence $b = b_0, \ldots, b_m = \llbracket b \rrbracket$ of BDDs that is constructed by the approximation algorithm (Algorithm 1), where the BDD structure is upgraded bottom-up. We have $\llbracket b \rrbracket = b_m \models b_{m-1} \models \ldots \models b_0 = b$, since in each newly constructed BDD for some node n $low(n)$ with $root(n) \in U$ is replaced by $high(n) \wedge low(n)$.
  Since $C \models b$ and $C \not\models \llbracket b \rrbracket$, there must be an index k such that $C \models b_k$, $C \not\models b_{k+1}$. Let n be the node that is modified in step k, where $root(n) = f \in U$. We must have $f \notin C$, since the changes concern only the *low*-successor and if $f \in C$, the corresponding path would take the *high*-successor and nothing would change concerning acceptance of C from b_k to b_{k+1}.
  Now assume that $C' = C \cup \{f\} \models b$. This would be a contradiction to the maximality of C and hence $C \cup \{f\} \not\models b$, as required.

■

Proof of Proposition 3

Proof:

(i) For this proof, we work with the set-based interpretation, which allows for four views, one on the Boolean algebra $\mathbb{B} = \mathcal{P}(\mathcal{P}(N))$, one on the lattice $\mathbb{L} = \mathcal{O}(\mathcal{P}(N), \leq)$, one of the Boolean algebra $\mathbb{B}' = \mathcal{P}(\llbracket d \rrbracket)$ and one on the lattice $\mathbb{L}' = (\mathcal{O}(\llbracket d \rrbracket), \leq')$ where $\leq' = \leq|_{\llbracket d \rrbracket \times \llbracket d \rrbracket}$. We will mostly argue in the Boolean algebra $\mathbb{B}$. When talking about downward-closed sets, we will usually indicate with respect to which order. Similarly, the approximation relative to $\leq$ is written $\lfloor \_ \rfloor$, whereas the approximation relative to $\leq'$ is written $\lfloor \_ \rfloor'$.
We can compute:

$$b_1 \rightarrow_{\mathbb{L}'} b_2 \equiv \lfloor \neg_{\mathbb{B}'} b_1 \vee b_2 \rfloor' \equiv \lfloor (\neg_{\mathbb{B}} b_1 \wedge d) \vee b_2 \rfloor'$$

To conclude the proof, we will now show that $\lfloor b \rfloor' \equiv \lfloor b \vee \neg d \rfloor \wedge d$ for any $b \in \mathbb{B}'$. We prove this via mutual implication.

- We show $\lfloor b \vee \neg d \rfloor \wedge d \models \lfloor b \rfloor'$:

  $$\lfloor b \vee \neg d \rfloor \wedge d \models (b \vee \neg d) \wedge d \equiv (b \wedge d) \vee (\neg d \wedge d) \equiv b \wedge d \models b$$

  Since $\lfloor b \vee \neg d \rfloor \wedge d$ implies d, it certainly is in $\mathbb{B}'$. We now show that it is downward-closed wrt. $\leq'$: we use an auxiliary relation $\leq''$, which is the smallest partial order on $\mathbb{B}$ that contains $\leq'$, i.e., $\leq'$ extended to $\mathbb{B}$. We have $\leq'' \subseteq \leq$. Since $\lfloor b \vee \neg d \rfloor$ is an approximation it is downward-closed wrt. $\leq$ and hence downward-closed wrt. $\leq'$. Moreover, d is downward-closed relative to $\leq''$ (obvious by definition). Since the intersection of two downward-closed sets is again downwards closed, $\lfloor b \vee \neg d \rfloor \wedge d$ is downward-closed relative to $\leq''$ and since finally, downward-closure relative to $\leq''$ is the same as downward-closure relative to $\leq'$ provided we discuss an element from $\mathbb{B}'$, we can conclude that $\lfloor b \vee \neg d \rfloor \wedge d$ belongs to $\mathbb{L}'$.
  From $\lfloor b \vee \neg d \rfloor \wedge d \in \mathbb{L}'$ and $\lfloor b \vee \neg d \rfloor \wedge d \models b$ it follows that $\lfloor b \vee \neg d \rfloor \wedge d \models \lfloor b \rfloor'$ by definition of the approximation.
- We show $\lfloor b \rfloor' \models \lfloor b \vee \neg d \rfloor \wedge d$:
  Let any $C \in \mathcal{P}(N)$ be given, such that $C \in \llbracket \lfloor b \rfloor' \rrbracket$. We show that in this case $C \in \llbracket \lfloor b \vee \neg d \rfloor \wedge d \rrbracket$, which proves $\lfloor b \rfloor' \models \lfloor b \vee \neg d \rfloor \wedge d$. Let $\downarrow C$ be the downwards-closure of C wrt. $\leq$.
  Since $\lfloor b \rfloor'$ must de downward-closed relative to $\leq'$, it holds that $\downarrow C \cap \llbracket d \rrbracket \subseteq \llbracket \lfloor b \rfloor' \rrbracket$. Disjunction with $\neg d$ on both sides yields $\downarrow C \subseteq \llbracket \lfloor b \rfloor' \vee \neg d \rrbracket \subseteq \llbracket b \vee \neg d \rrbracket$, since $c \models c \vee \neg d \equiv (c \wedge d) \vee \neg d$. The set $\downarrow C$ is downwards-closed wrt. $\leq$, so it is contained in the approximation relative to $\leq$ of this set, i.e $\downarrow C \subseteq \llbracket \lfloor b \vee \neg d \rfloor \rrbracket$. Thus, in particular, $C \in \llbracket \lfloor b \vee \neg d \rfloor \rrbracket$. Since $C \in \llbracket \lfloor b \rfloor' \rrbracket$, it follows that $C \in \llbracket d \rrbracket$, therefore we can conclude $C \in \llbracket \lfloor b \vee \neg d \rfloor \wedge d \rrbracket$.

  Hence

  $$\lfloor (\neg_{\mathbb{B}} b_1 \wedge d) \vee b_2 \rfloor' \equiv \lfloor (\neg_{\mathbb{B}} b_1 \wedge d) \vee b_2 \vee \neg d \rfloor \wedge d$$
  $$\equiv \lfloor \neg_{\mathbb{B}} b_1 \vee b_2 \vee \neg d \rfloor \wedge d$$

(ii) Since d is downward-closed wrt. $\leq$, $d = \lfloor d \rfloor$, therefore, using Lemma 1, we obtain $\lfloor \neg b_1 \vee b_2 \vee \neg d \rfloor \wedge d \equiv \lfloor \neg b_1 \vee b_2 \vee \neg d \rfloor \wedge \lfloor d \rfloor \equiv \lfloor (\neg b_1 \vee b_2 \vee \neg d) \wedge d \rfloor \equiv \lfloor (\neg b_1 \vee b_2) \wedge d \vee \neg d \wedge d \rfloor \equiv \lfloor (\neg b_1 \vee b_2) \wedge d \rfloor \equiv \lfloor \neg b_1 \vee b_2 \rfloor \wedge \lfloor d \rfloor \equiv \lfloor \neg b_1 \vee b_2 \rfloor \wedge d$.

■

APPENDIX B
RUN-TIME RESULTS

In this section we present the detailed run-time results for our BDD-based implementation versus the non-BDD-based implementation for a sequence of CTSs.

Table 5 shows the runtime results (in milliseconds) for the computation of the largest bisimulation for our implementation on the family of CTSs described in Section VI-C. Despite some fluctuations, the quotient of the time taken when not using BDDs and when using BDDs increases exponentially by factor of about 2.

# features	time(BDD)	time(without BDD)	time(without BDD)/time(BDD)
1	42	13	0.3
2	64	32	0.5
3	143	90	0.6
4	311	312	1.0
5	552	1128	2.0
6	1140	3242	2.8
7	1894	8792	4.6
8	1513	13256	8.8
9	1872	39784	21
10	3208	168178	52
11	5501	513356	93
12	7535	1383752	184
13	5637	3329418	591
14	6955	8208349	1180
15	11719	23700878	2022
16	15601	57959962	3715
17	18226	150677674	8267
18	17001	347281057	20427
19	22145	out of memory	—

Fig. 5. Run-time results (in milliseconds)

APPENDIX C
DEACTIVATING TRANSITIONS

We will now work on an extension that allows transitions to deactivate when upgrading.

We have introduced conditional transition systems (CTS) as a modelling technique that can be used for modelling software product lines (SPLs). CTSs are a strictly stronger model than (standard) FTSs, allowing for upgrades. Products derived from a software product line may be upgraded to advanced versions, activating additional transitions in the system. A change in the transition function can only be realised in one direction: by adding transitions which were previously not available, while all previously active transitions remain active.

However, this choice may not be the optimal choice in all cases, because sometimes an advanced version of a system may offer improved transitions over the base product. For instance, a free version of a system may display a commercial when choosing a certain transition, whereas a premium model may forego the commercial and offer the base functionality right away.

A practical motivation may be derived from our Example 4. In this transition system one may want to be able to model that in the unsafe state, the advanced version can only send an encrypted message, since we assume that the user is always interested in a secure communication, ensured either by a safe channel or by encryption. However, it is not an option to simply drop the unencrpyted transition from the unsafe state with respect to the base version, because then, whenever the system encounters an unsafe state in the base version, the system will remain in a deadlock unless the user decides to perform an upgrade. We will solve such a situation as follows: we will add priorities that allow to deactivate the unencrypted transition in the presence of an encrypted transition.

In order to allow for deactivation of transitions when upgrading, we propose a slight variation of of the definition of CTS/LaTS and the corresponding bisimulation relation. A *conditional transition system with action precedence* is a triple $(X, (A, <_A), f)$, where (X, A, f) is a CTS and $<_A$ is a strict order on A.

Intuitively, a CTS with action precedence evolves in a very similar way to standard CTS. Before the system starts acting, it is assumed that all the conditions are fixed and a condition $\varphi \in \Phi$ is chosen arbitrarily which represents a selection of a valid product of the system (product line). Now all the transitions that have a condition greater than or equal to φ are activated, while the remaining transitions are inactive. This is unchanged from standard CTS, however, *if from a state x there exist two transitions $x \xrightarrow{a,\varphi} x'$ and $x \xrightarrow{a',\varphi} x''$ where $a' > a$, i.e. a' takes precedence over a, then additionally $x \xrightarrow{a,\varphi} x'$ remains inactive*. Henceforth, the system behaves like a standard transition system; until at any point in the computation, the condition is changed to a smaller one (say, φ') signifying a selection of a valid, upgraded product. Now (de)activation of transitions depends on the new condition φ', rather than on the old condition φ. As before, active transitions remain active during an upgrade, *unless new active transitions appear that are exiting the same state and are labelled with an action of higher priority*.

In the sequel we will just write CTS for CTS with action precedence, since for the remainder of this section we will solely investigate this variation of CTS. This changed interpretation of the behaviour of a CTS of course also has an

effect on the bisimulation.

Definition 12. *Let $(X,(A,<_A),f)$, $(Y,(A,<_A),g)$ be two CTSs over the same set of conditions $(\Phi,\leq)$. For a condition $\varphi\in\Phi$, we define $\bar{f}_\varphi(x,a)$ to denote the* labelled transition system *induced by a CTS $(X,(A,<_A),f)$ with action precedence, where*

$$\bar{f}_\varphi(x,a) =$$
$$\{x' \mid x \xrightarrow{a,\varphi} x' \wedge \neg(\exists a'\in A, x''\in X : a' > a \wedge x \xrightarrow{a',\varphi} x'')\}.$$

Two states $x\in X, y\in Y$ are conditionally bisimilar (wrt. action precedence) *under a condition $\varphi\in\Phi$, denoted $x\sim^p_\varphi y$, if there is a family of relations $R_{\varphi'}$ (for every $\varphi'\leq\varphi$) such that*

(i) each relation $R_{\varphi'}$ is a traditional bisimulation relation between $\bar{f}_{\varphi'}$ and $\bar{g}_{\varphi'}$,
(ii) whenever $\varphi'\leq\varphi''$, we have $R_{\varphi'}\supseteq R_{\varphi''}$, and
(iii) R_φ relates x and y, i.e. $(x,y)\in R_\varphi$.

The definition of bisimilarity is analogous to traditional CTS but refers to the new transition system given by $\bar{f}$, which contains only the maximal transitions.

Lattice transition systems (LaTSs) can be extended in the same way, by adding an order on the set of actions and leaving the remaining definition unchanged. Disregarding deactivation, there still is a duality between CTSs and LaTSs. Now, in order to characterise bisimulation using a fixpoint operator, we can modify the operators F_1, F_2 and F to obtain G_1, G_2 and G, respecting the deactivation of transitions as follows.

Definition 13. *Let $(X,(A,<_A),\alpha)$ and $(Y,(A,<_A),\beta)$ be LaTSs (with ordered actions). Recall the residuum operator $(\to)$ on a lattice and define three operators $G_1,G_2,G:(X\times Y\to\mathbb{L})\to(X\times Y\to\mathbb{L})$ in the following way:*

$$G_1(R)(x,y) = \bigsqcap_{a\in A,x'\in X}\Big(\alpha(x,a,x')\to(\bigsqcup_{y'\in Y}(\beta(y,a,y')\sqcap R(x',y'))\sqcup\bigsqcup_{a'>a,x''\in X}\alpha(x,a',x''))\Big),$$

$$G_2(R)(x,y) = \bigsqcap_{a\in A,y'\in Y}\Big(\beta(y,a,y')\to(\bigsqcup_{x'\in X}(\alpha(x,a,x')\sqcap R(x',y'))\sqcup\bigsqcup_{a'>a,y''\in Y}\beta(y,a',y''))\Big),$$

$$G(R)(x,y) = G_1(R)(x,y)\sqcap G_2(R)(x,y).$$

Now, we need to show that we can characterise the new notion of bisimulations as post-fixpoints of this operator G. For the corresponding proof we will make use of the following observation:

Lemma 7. *Let $\mathbb{L}=\mathcal{O}(\Phi)$ for any finite partially ordered set $(\Phi,\leq)$ be a lattice that embeds into $\mathbb{B}=\mathcal{P}(\Phi)$. Take $\varphi\in\Phi$. Then, in order to show that $\varphi\in(l_1\to l_2)$, for any given $l_1,l_2\in\mathbb{L}$, it suffices to show that for all $\varphi'\leq\varphi$, $\varphi'\notin l_1$ or $\varphi'\in l_2$.*

Proof: We have already shown that $l_1\to l_2 = \lfloor l_1\to l_2\rfloor = \lfloor\neg l_1\sqcup l_2\rfloor$. Now, if all $\varphi'\leq\varphi$ are not in l_1, i.e. in $\neg l_1$, or in l_2, then all $\varphi'\leq\varphi$ are in $\neg l_1\sqcup l_2$. Therefore, $\downarrow\varphi\subseteq\neg l_1\sqcup l_2$, and thus $\varphi\in l_1\to l_2$. ■

Theorem 7. *Let $(X,(A,<_A),f)$, $(Y,(A,<_A),g)$ be two CTSs over $(\Phi,\leq)$ and $(X,(A,<_A),\alpha)$, $(Y,(A,<_A),\beta)$ over $\mathcal{O}(\Phi)$ be the corresponding LaTS. For any two states $x\in X, y\in Y$ it holds that $x\sim^p_\varphi y$ if and only if there exists a post-fixpoint $R:X\times Y\to\mathbb{L}$ of G ($R\sqsubseteq G(R)$) such that $\varphi\in R(x,y)$.*

Proof:

- Assume R is a post-fixpoint of G, i.e. $R\sqsubseteq G(R)$, let $x\in X$ and $y\in Y$ be given arbitrarily and $\varphi\in R(x,y)$. We define for each $\varphi'\leq\varphi$ a relation $R_{\varphi'}$ according to
$$(x',y')\in R_{\varphi'}\Leftrightarrow\varphi'\in R(x',y').$$
Since each set $R(x',y')$ is downward-closed for all $x'\in X, y'\in Y$, it holds that $R_{\varphi_1}\subseteq R_{\varphi_2}$ whenever $\varphi_1\geq\varphi_2$. Moreover, since we assume $\varphi\in R(x,y)$, $(x,y)\in R_{\varphi'}$ must hold for all $\varphi'\leq\varphi$.
So we only need to show that all $R_{\varphi'}$ are traditional bisimulations for $\bar{f}_{\varphi'}$. For this purpose let x',y',φ' be given, such that $(x',y')\in R_{\varphi'}$. Moreover, let $a\in A$ and $x''\in X$ be given such that $x''\in\bar{f}_{\varphi'}(x',a)$ – if no such a and x'' exists then the first bisimulation condition is trivially true. For G_1, it must be true that $\varphi'\in G_1(R)(x,y)$. Thus,
$$\varphi'\in\Big(\alpha(x',a,x'')\to(\bigsqcup_{y''\in Y}(\beta(y',a,y'')\sqcap R(x'',y''))\sqcup\bigsqcup_{a'>a,x'''\in X}\alpha(x',a',x'''))\Big)$$
Since we also know that $\varphi'\in\alpha(x',a,x'')$ because $x''\in\bar{f}_{\varphi'}(x',a)$, it must be true that
$$\varphi'\in\Big(\bigsqcup_{y''\in Y}(\beta(y',a,y'')\sqcap R(x'',y''))\sqcup\bigsqcup_{a'>a,x'''\in X}\alpha(x',a',x''')\Big).$$
This is true because $\psi\in l_1\to l_2\Leftrightarrow\psi\in\lfloor\neg l_1\vee l_2\rfloor\Rightarrow\psi\in\neg l_1\vee l_2$ (Lemma 7) and, if $\psi\in l_1$, hence $\psi\notin\neg l_1$, it follows that $\psi\in l_2$.
Per definition of $\bar{f}_{\varphi'}$, there exists no $a'>a$ such that $\bar{f}_{\varphi'}(x'',a')\neq\emptyset$. Therefore,
$$\varphi'\notin\bigsqcup_{a'>a,x'''\in X}\alpha(x',a',x''').$$
It follows that
$$\varphi'\in\bigsqcup_{y''\in Y}\beta(y',a,y'')\sqcap R(x'',y'').$$
Then, there must exist at least one $y''\in Y$ such that $\varphi'\in\beta(y',a,y'')\sqcap R(x'',y'')$. It follows that $\varphi'\in R(x'',y'')$, i.e. $(x'',y'')\in R_{\varphi'}$.

We will now show that $y'' \in \bar{g}_{\varphi'}(y', a)$, holds as well. Assume, to the contrary, that $y'' \notin \bar{g}_{\varphi'}(y', a)$, then, due to $\varphi' \in \beta(y', a, y'')$, there must exist an $a' > a$ and a $y''' \in Y$ such that $\varphi' \in \beta(y', a', y''')$. W.l.o.g. choose a' maximal. Since we required $(x', y') \in R_{\varphi'}$, it has to hold that $\varphi' \in G_2(R)(x', y')$. So in particular,

$$\varphi' \in \Big(\beta(y', a', y''') \to \big(\bigsqcup_{x''' \in X} (\alpha(x', a', x''') \sqcap R(x''', y''')) \sqcup \bigsqcup_{a'' > a', y'''' \in Y} \beta(y', a'', y'''')\big)\Big)$$

Since we chose a' maximal, we know that $\varphi' \notin \bigsqcup_{a'' > a', y'''' \in Y} \beta(y', a'', y'''')$. Moreover, since $a' > a$ and $x'' \in \bar{f}_{\varphi'}(x', a)$, there exists no x''' such that $\varphi' \in \alpha(x', a', x''')$. Thus, φ' is not in the right side of the residuum, yet it is in the left side of the residuum, therefore, it is not in the residuum. Thus, we can conclude $\varphi' \notin G_2(R)(x', y')$, which is a contradiction.
Thus, the first bisimulation condition is true. The second condition can be proven analogously, reversing the roles of G_2 and G_1 to find the answer step in $\bar{f}_{\varphi'}$.

- Now, assume the other way around, that a family R_φ of bisimulations from $\bar{f}_\varphi$ to $\bar{g}_\varphi$ exists such that for all states $x \in X$, $y \in Y$ and for all pairs of conditions $\varphi_1, \varphi_2 \in \Phi$ the expression $\varphi_1 \leq \varphi_2$ implies $R_{\varphi_1} \supseteq R_{\varphi_2}$. Moreover, let φ, $x \in X$ and $y \in Y$ be given such that $(x, y) \in R_\varphi$. We define $R : X \times Y \to \mathbb{L}$ according to

$$R(x, y) = \{\varphi' \mid (x, y) \in R_{\varphi'}\}.$$

Due to anti-monotonicity of the family of $R_{\varphi'}$ all entries in R are indeed lattice elements from $\mathcal{O}(\Phi, \leq)$. Moreover, by definition, $\varphi \in R(x, y)$. So it only remains to be shown that R is a post-fixpoint.
For this purpose, let $x' \in X$, $y' \in Y$ and $\varphi' \in \Phi$ be given, such that $\varphi' \in R(x', y')$. (If no such x', y', φ' exist, then R is the zero matrix (where all entries are $\emptyset$) and $R \sqsubseteq G(R)$ holds trivially.) We will now show that $\varphi' \in G_1(R)(x', y')$. The fact that $\varphi' \in G_2(R)(x', y')$ can be shown analogously. We need to show that

$$\varphi' \in \Big(\alpha(x, a, x') \to \big(\bigsqcup_{y' \in Y} (\beta(y, a, y') \sqcap R(x', y')) \sqcup \bigsqcup_{a' > a, x'' \in X} \alpha(x, a', x'')\big)\Big)$$

for all $x'' \in X$ and $a \in A$.
We recall that $l_1 \to_{\mathbb{L}} l_2 = \lfloor l_1 \to_{\mathbb{B}} l_2 \rfloor = \lfloor \neg l_1 \vee l_2 \rfloor$ (Lemma 1) and show that whenever $\varphi' \in \alpha(x, a, x')$, it holds that $\varphi' \in \big(\bigsqcup_{y' \in Y} (\beta(y, a, y') \sqcap R(x', y')) \sqcup \bigsqcup_{a' > a, x'' \in X} \alpha(x, a', x'')\big)$. We distinguish according to whether a is maximal such that $\varphi' \in \alpha(x, a, x'')$:

- There is no $a' > a$ such that $\varphi' \in \alpha(x, a, x'')$ for any $x'' \in X$:
  Then there must exist a $y' \in Y$ such that $\varphi' \in \beta(y, a, y')$ and $(x', y') \in R_{\varphi'}$, i.e. $\varphi' \in R(x', y')$, because $R_{\varphi'}$ is a bisimulation and for all $\varphi'' \leq \varphi'$ we have $R_{\varphi'} \subseteq R_{\varphi''}$.
- There is an $a' > a$ such that $\varphi' \in \alpha(x, a, x'')$ for some $x'' \in X$:
  Then $\varphi' \in \bigsqcup_{a' > a, x'' \in X} \alpha(x, a', x'')$.

So we have shown for all $\varphi' \in R(x', y')$ that $\varphi' \in \alpha(x, a, x')$ implies

$$\varphi' \in \big(\bigsqcup_{y' \in Y} (\beta(y, a, y') \sqcap R(x', y')) \sqcup \bigsqcup_{a' > a, x'' \in X} \alpha(x, a', x'')\big),$$

i.e. we have

$$\varphi' \in \neg\alpha(x, a, x') \sqcup \big(\bigsqcup_{y' \in Y} (\beta(y, a, y') \sqcap R(x', y')) \sqcup \bigsqcup_{a' > a, x'' \in X} \alpha(x, a', x'')\big)$$

in the Boolean algebra. Since $R(x', y')$ is a lattice element and therefore downward-closed, we can apply Lemma 7 and conclude that

$$\varphi' \in \Big(\alpha(x, a, x') \to \big(\bigsqcup_{y' \in Y} (\beta(y, a, y') \sqcap R(x', y')) \sqcup \bigsqcup_{a' > a, x'' \in X} \alpha(x, a', x'')\big)\Big)$$

in the lattice, concluding the proof. ∎

Hence we can compute the bisimulation via a fixpoint iteration, as with LaTSs without an ordering on the action labels. Due to the additional supremum in the fixpoint operator, the matrix notation cannot be used anymore. However, since the additional supremum term can be precomputed for each pair of states $x \in X$ or $y \in Y$ and action $a \in A$, the performance of the algorithm should not be affected in a significant way.

Note that, different from the Boolean case, $l_1 \to (l_2 \sqcup l_3) \not\equiv (l_1 \to l_2) \sqcup l_3$, which is relevant for the definition of G. In fact, moving the supremum $\bigsqcup_{a' > a, x'' \in X} \alpha(x, a', x'')$ outside of the residuum would yield an incorrect notion of bisimilarity.

In addition, it may appear more convenient to drop the monotonicity requirement for transitions and to allow arbitrary deactivation of transitions, independently of their label. However, this would result in a loss of the duality result and as a result, the fixpoint algorithm that allows to compute the bisimilarity in parallel for all products would be rendered incorrect.