Conditional Transition Systems with Upgrades

Harsh Beohar[a], Barbara König[a], Sebastian Küpper[b], Alexandra Silva[c]

[a] *Universität Duisburg-Essen, Germany*
[b] *FernUniversität in Hagen, Germany*
[c] *University College London, UK*

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.

This is an extended version of the TASE 2017 paper [1], including all proofs, additional examples, an extension of the formalism to account for deactivation of updates and detailed runtime results.

Keywords: conditional transition systems, software product lines, BDDs, features, bisimulation

1. Introduction

Conditional transition systems (CTSs)[1] have been introduced in [2] as a compact model for classes of systems with similar behaviour, wherein the transitions are guarded by 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

Email addresses: harsh.beohar@uni-due.de (Harsh Beohar), barbara_koenig@uni-due.de (Barbara König), sebastian.kuepper@fernuni-hagen.de (Sebastian Küpper), alexandra.silva@ucl.ac.uk (Alexandra Silva)

[1] Note that, prior to [2], the term "conditional transition system" has been used earlier in [3], but for a different type of transition system.

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, the definition of a CTS using posets, instead of the dual view using lattices, is natural to model the behaviour of a software product line; however, LaTSs are more compact in nature than their CTS counterparts fruitful for symbolic manipulation. 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 [4–12]).

Featured transition systems (FTS) – a recent extension of conventional transition systems proposed by Classen et al.[6] – 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 [13], 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 standard FTSs. Furthermore, we also consider deactivation of transitions 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. In particular, we explain 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. Here, we also review a variation of the system model where the requirement of monotone upgrading is relaxed. We then focus on applications to conventional and adaptive SPLs. With this application in mind, we present our implementation based on binary decision diagrams (BDDs), which provides a compact encoding of a propositional formula and also show how BDDs can be employed in a lattice-based setting.

This paper is organised as follows. Section 2 recalls the fundamentals of lattice theory relevant to this paper. Then, in Section 3 we formally introduce CTSs and conditional bisimilarity. In Section 4, using the Birkhoff duality, it is shown that CTSs can be represented as lattice transition systems (LaTSs) whose transitions are labelled with the elements from a distributive lattice. Moreover, the bisimilarity introduced on LaTSs is shown to coincide with the conditional bisimilarity on the corresponding CTSs. In Section 5, we show how bisimilarity can be computed using a form of matrix multiplication. Then in Section 6 we introduce the bisimulation game and explain how to extend the formalism by a deactivation mechanism for transitions in Section 7. Section 8 focusses on the translation between an FTS and a CTS, and moreover, a BDD-based implementation of checking bisimilarity is laid out. Here we also discuss car control as a more complex example. Section 9 provides a detailed comparison of related work. Lastly, we conclude with a discussion future work in Section 10.

2. Preliminaries

We recall some basic definitions concerning lattices, including the well-known Birkhoff duality from [14].

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* $\ell \sqcap \ell' \sqsubseteq m$*, called the residuum. The residuum can also be defined as* $\ell \to m = \bigsqcup\{\ell' \mid \ell \sqcap \ell' \sqsubseteq m\}$*, furthermore we define negation as* $\neg\ell = \ell \to 0$*. A* Boolean algebra $\mathbb{L}$ *is a Heyting algebra satisfying* $\neg\neg\ell = \ell$ *for all* $\ell \in \mathbb{L}$*.*

Example 2. *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 [14]. In the sequel we will focus on finite distributive lattices (which are Heyting algebras). We first need the following concepts.

Definition 3. *Let* $\mathbb{L}$ *be a bounded lattice. An element* $n \in \mathbb{L} \setminus \{0\}$ *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* $\mathbb{L}$*.*

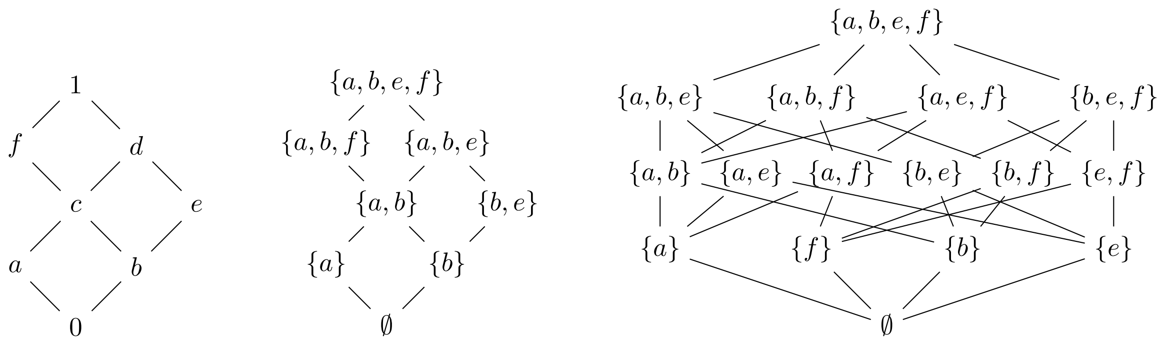


Figure 1: An example motivating Birkhoff's representation theorem.

Let $(S, \leq)$ be a partially ordered set. A subset $S' \subseteq S$ is downward-closed, *whenever $s' \in S'$ and $s \leq s'$ imply $s \in S'$. 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 $s \in S$.*

Example 4. *For our example of a Boolean algebra $\mathbb{B}(N)$, quotiented by equivalence, the irreducibles are the complete conjunctions of literals, or, alternatively, all the singletons of the form $\{C\}$ (for $C \subseteq N$).*

We can now state Birkhoff's representation theorem for finite distributive lattices [14].

Theorem 5. *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 6. *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, and f, i.e. exactly those elements that have a unique direct predecessor and they are ordered as follows: $a \sqsubseteq f$, $b \sqsubseteq f$, $b \sqsubseteq e$. In the middle we depict the dual representation of $\mathbb{L}$ 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 $\mathcal{J}(\mathbb{L})$. On the right we display the Boolean algebra $\mathcal{P}(\mathcal{J}(\mathbb{L}))$. The embedding of $\mathbb{L}$ into $\mathcal{P}(\mathcal{J}(\mathbb{L}))$ can be easily determined via these pictures: any lattice element in the lattice on the left maps to its dual element in the middle drawn in the same relative position of the diagram and each set from the dual representation is mapped to itself in the Boolean algebra on the right (whose elements are ordered by set inclusion).*

Proposition 7 (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 $\eta(\ell) = \{\ell' \in \mathcal{J}(\mathbb{L}) \mid \ell' \sqsubseteq \ell\}$, in particular η is a lattice homomorphism.*

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 8. *Let $\mathbb{L}$ be a complete distributive lattice that embeds into a Boolean algebra $\mathbb{B}$. 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 Example 6, the set of irreducibles $\{a, e, f\}$, which is not downward-closed, is approximated by $\lfloor \{a, e, f\} \rfloor = \{a\}$. In general, the approximation of each element ℓ can be determined in the example by finding the highest-level predecessor that is also in $\mathbb{L}$, i.e. taking the maximal subset of ℓ, that is contained in $\mathbb{L}$. Note, that this set is indeed unique, because if $\mathbb{L} \ni \ell_1 \subseteq \ell$ and $\mathbb{L} \ni \ell_2 \subseteq \ell$, then also $\ell_1 \sqcup \ell_2 = \ell_1 \cup \ell_2 \in \mathbb{L}$ and $\ell_1 \cup \ell_2 \subseteq \ell$.

Lemma 9. *Let $\mathbb{L}$ be a complete distributive lattice that embeds into a Boolean algebra $\mathbb{B}$. For ℓ, $m \in \mathbb{B}$, we have $\lfloor \ell \sqcap m \rfloor = \lfloor \ell \rfloor \sqcap \lfloor m \rfloor$ and furthermore $\ell \sqsubseteq m$ implies $\lfloor \ell \rfloor \sqsubseteq \lfloor m \rfloor$. If $\ell, m \in \mathbb{L}$, then $\lfloor \ell \sqcup \neg m \rfloor = m \rightarrow_{\mathbb{L}} \ell$.*

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

Infimum: 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 $\lfloor \ell \rfloor \sqcap \lfloor m \rfloor \sqsupseteq \lfloor \ell \sqcap m \rfloor$.

Residuum: 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 $m \rightarrow_{\mathbb{L}} \ell = \bigsqcup\{x \in \mathbb{L} \mid m \sqcap x \sqsubseteq \ell\}$. 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 $\ell \sqcup \neg m \sqsupseteq (m \sqcap x) \sqcup \neg m = (m \sqcup \neg m) \sqcap (x \sqcup \neg m) = 1 \sqcap (x \sqcup \neg m) = x \sqcup \neg m \sqsupseteq x$. □

Note that in general it does not hold that $\lfloor \ell \sqcup m \rfloor = \lfloor \ell \rfloor \sqcup \lfloor m \rfloor$ and $\lfloor \ell \sqcup \neg m \rfloor = \lfloor m \rfloor \rightarrow_{\mathbb{L}} \lfloor \ell \rfloor$ for arbitrary $\ell, m \in \mathbb{B}$. To witness why these equations fail to hold, take $\ell = \{a, e\}$ and $m = \{b, f\}$ in Example 6 as counterexample.

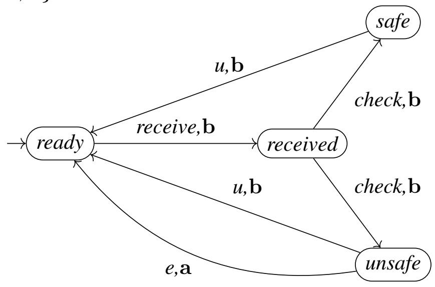
3. Conditional Transition Systems

In this section, we introduce conditional transition systems together with a notion of behavioural equivalence based on bisimulation. In [2], such transition systems were already investigated in a coalgebraic setting without an order on the conditions. Naturally, the CTSs from [2] arise as a special case of our notion of CTS, when $\leq$ is chosen as the discrete order $=$.

Definition 10. *Let $(\Phi, \leq)$ be a finite poset. Then, a* conditional transition system *(CTS) is a triple (X, A, f) consisting of a set of states X, a finite set A called the label alphabet, and a function $f \colon X \times A \to (\Phi \to \mathcal{P}(X))$ mapping every pair in $X \times A$ to a monotone function of type $(\Phi, \leq) \to (\mathcal{P}(X), \supseteq)$. We call the elements of Φ the conditions of the CTS. 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 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 (e.g. a product line). 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 the 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.[2] Notice that LTSs arise as a special case when $|\Phi| = 1$.

Example 11. *Consider a simplified example (originally from [13]) of an adaptive routing protocol modelled as a CTS over the alphabet $A = \{receive, check, u, e\}$:*



The system consists of two products: the basic *system with no encryption feature written as* $\mathbf{b}$ *and the* advanced *system with encryption feature written as* $\mathbf{a}$*. The ordering on the sets of valid products is defined as the smallest poset containing the relation* $\mathbf{a} < \mathbf{b}$*. 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 be easily encoded by a transition function. E.g., $f(received, check)(\mathbf{b}) = \{safe, unsafe\}$ *and* $f(received, a)(\varphi) = \emptyset$ *(for* $\varphi \in \{\mathbf{a}, \mathbf{b}\}$*,* $a \in A \setminus \{check\}$*) 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 products in their transfer properties.

Definition 12 (Conditional Bisimulation)**.** *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* $f_\varphi \colon X \times A \to \mathcal{P}(X)$ *with* $f_\varphi(x, a) = f(x, a)(\varphi)$ *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* $R_{\varphi'}$ *(for every* $\varphi' \leq \varphi$*) such that*

[2]Monotonicity is important to ensure the duality between CTSs and lattice transition systems (see Section 4). In Section 7 we will weaken the requirement that transitions remain active after an upgrade by discussing a mechanism for deactivating transitions via priorities.

(i) each relation $R_{\varphi'}$ is a traditional bisimulation relation[3] *between $f_{\varphi'}$ and $g_{\varphi'}$,*

(ii) whenever $\varphi' \leq \varphi''$, it holds that $R_{\varphi'} \supseteq R_{\varphi''}$, and

(iii) R_φ relates x and y, i.e. $(x, y) \in R_\varphi$.

To illustrate conditional bisimulation, we discuss our previous example of a CTS.

Example 13. *As an example of conditional bisimulation, consider the CTS illustrated in Example 11 where the condition $\mathbf{b}$ of the transition 'received $\xrightarrow{\textit{check},\mathbf{b}}$ unsafe' is replaced by $\mathbf{a}$. Let ready$_1$ and ready$_2$ denote the initial states of the system before and after the above modification, as well as f the original transition relation and g the modified transition relation, respectively. Then, we find ready$_1 \sim_\mathbf{a}$ ready$_2$; however, ready$_1 \not\sim_\mathbf{b}$ ready$_2$. To see why the latter fails to hold, consider the smallest bisimulation relation $R_\mathbf{b}$ in the traditional sense between f and g under the condition $\mathbf{b}$ that contains the pair $(\textit{ready}_1, \textit{ready}_2)$. The relation $R_\mathbf{b}$ has the following form:*

$$R_\mathbf{b} = \{(\textit{ready}_1, \textit{ready}_2), (\textit{received}_1, \textit{received}_2), (\textit{unsafe}_1, \textit{safe}_2), (\textit{safe}_1, \textit{safe}_2)\}.$$

Similarly, we can give the bisimilarity *$R_\mathbf{a}$ between the instantiations of f and g to the condition $\mathbf{a}$:*

$$R_\mathbf{a} = \{(\textit{ready}_1, \textit{ready}_2), (\textit{received}_1, \textit{received}_2), (\textit{unsafe}_1, \textit{unsafe}_2), (\textit{safe}_1, \textit{safe}_2)\}.$$

Observe, that the states unsafe$_1$, safe$_2$ are bisimilar in the traditional sense, under the condition $\mathbf{b}$, i.e. $(\textit{unsafe}_1, \textit{safe}_2) \in R_\mathbf{b}$. This pair is required in any bisimulation relation that contains $(\textit{ready}_1, \textit{ready}_2)$, because otherwise, in the bisimulation game, after the necessary receive-steps in round 1, Player 1 can make a select-move from received$_1$ to unsafe$_1$ which cannot be answered by Player 2. However, the two states cannot be related by any traditional bisimulation relation under the condition $\mathbf{a}$, because from unsafe$_1$ an encrypted-transition is possible, which is not possible from the state safe$_2$. Therefore, Condition (ii) of Definition 12 is violated and the states ready$_1$, ready$_2$ can be seen not to be conditionally bisimilar, even though there are bisimilar for all possible instantiations.
Indeed, the two systems behave differently. In the first, it is possible to receive a message, check (arrive in state unsafe), perform 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 state 'safe' before doing the upgrade. However, without upgrades (i.e. choosing the discrete order on the products instead), the above systems would be conditionally bisimilar for both products.

4. Lattice Transition Systems

4.1. Lattice Transition Systems and Lattice Bisimilarity

Recall from Theorem 5, that there is a duality between partial orders and distributive lattices. In fact, 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 (cf. Theorem 15).

[3]Since φ' is fixed, the traditional bisimulation relation only takes labels from A into account.

Definition 14 (Lattice Transition Systems)**.** *A* lattice transition system *(LaTS) over an alphabet A and a finite distributive lattice $\mathbb{L}$ is a triple (X, A, α) consisting of a set of states X and a transition function $\alpha\colon X \times A \times X \to \mathbb{L}$.*

Note that superficially, lattice transition systems resemble weighted automata [15]. 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 also 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 15. *The set of all CTSs over a set of ordered conditions Φ is isomorphic to the set of all LaTSs over the lattice whose elements are the downward-closed subsets of Φ.*

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

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

We need to show that these functions actually map into their proclaimed domain, so we will show that $\eta(f)(x, a, x')$ is downward-closed and that $\eta'(\alpha)(x, a)$ is a monotone function.

- *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 definition of η' 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 to each other, because by the uniqueness of inverses we then have $\eta' = \eta^{-1}$. First, we show that $\eta' \circ \eta = \mathsf{id}$:

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

Analogously we can show that $\eta \circ \eta' = \mathsf{id}$:

$$\begin{aligned}&\varphi \in \alpha(x, a, x') \iff x' \in \{x'' \mid \varphi \in \alpha(x, a, x'')\} \iff x' \in \eta'(\alpha)(x, a)(\varphi)\\ &\quad\iff x' \in \eta'(\alpha)_\varphi(x, a) \iff \varphi \in \{\varphi' \in \Phi \mid x' \in \eta'(\alpha)_{\varphi'}(x, a)\} \iff \varphi \in \eta(\eta'(\alpha))(x, a, x').\end{aligned}$$ □

So every LaTS over a finite distributive lattice gives rise to a CTS in our sense (cf. Definition 10) and, since finite Boolean algebras are finite distributive lattices, conditional transition systems in the sense of [2] 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 to model the behaviour of a software product line, though when computing (symbolically) with CTSs we often choose the lattice view. More importantly, the point is, by adopting the lattice view, conditional bisimulations can be computed more elegantly than the following brute-force approach: (1) Instantiate a given CTS for each condition and apply the well-known algorithm for labelled transition systems to each element. (2) For each condition φ that is not minimal, take the bisimulations of all smaller conditions than φ, intersect their respective bisimulations to obtain a new bisimulation for φ. (3) Repeat until the relations R_φ do not change anymore.

Definition 16 (Lattice bisimulation)**.** *Let $(X, A, \alpha), (Y, A, \beta)$ be two LaTSs over a lattice $\mathbb{L}$. A conditional relation R, i.e. a function of type $R\colon X \times Y \to \mathbb{L}$ is a* lattice bisimulation *if and only if the following 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 a condition $\varphi \in \Phi$, we have a transition $x \xrightarrow{a,\varphi} x'$ in the CTS if and only if there exists a transition $x \xrightarrow{a,\downarrow\varphi} x'$ in the corresponding LaTS. Consequently, they are denoted by the same symbol.

Theorem 17. *Let $(X, A, f), (Y, A, g)$ be any two CTSs over Φ. Two states $x \in X, y \in Y$ are conditionally bisimilar under $\varphi \in \Phi$ if and only if there is a lattice bisimulation R between the corresponding LaTSs such that $\varphi \in R(x, y)$.*

Proof. $\boxed{\Leftarrow}$ Let $\varphi \in \Phi$ be a condition and let R be a lattice bisimulation relation such that $\varphi \in R(x,y)$ for $x \in X, y \in Y$. Then, we can construct a family of relations $R_{\varphi'}$ (for $\varphi' \leq \varphi$) as follows: $xR_{\varphi'}y \Leftrightarrow \varphi' \in R(x,y)$. 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 $xR_{\varphi'}y$ 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 proven.

$\boxed{\Rightarrow}$ 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 12(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 $\downarrow\varphi' \subseteq R(x,y)$. Thus, $x' \in f_{\varphi'}(x, a)$ 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, $y \xrightarrow{a,\downarrow\varphi'} y'$ and $\varphi' \in R(x', y')$, which implies $\downarrow\varphi' \subseteq R(x', y')$. Likewise, the symmetric condition of lattice bisimulation can be proven. □

The order in $\mathbb{L}$ also gives rise to a natural order on lattice bisimulations. Let $R_1, R_2 \colon X \times Y \to \mathbb{L}$ be any two lattice bisimulations. 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 18. *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.*

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 $\ell \in \mathcal{J}(\mathbb{L})$. 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 proven. ☐

4.2. Correspondence to Fitting's Bisimulation

Fitting [16] has conducted work on characterising conditional relations as matrices, which is strongly related to lattice bisimulation when restricted to the Boolean case. Remember that Boolean algebras are semirings for the operators $\sqcap$, $\sqcup$, therefore we can use matrix operations such as matrix multiplication $\cdot$. Additionally, for any matrix M, M^T denotes its transposed matrix. For a Boolean algebra $\mathbb{B}$, Fitting calls a matrix $R \colon X \times X \to \mathbb{B}$ a bisimulation for a transition matrix $\alpha \colon X \times X \to \mathbb{B}$ if and only if $R \cdot \alpha \sqsubseteq \alpha \cdot R$ and $R^T \cdot \alpha \sqsubseteq \alpha \cdot R^T$. 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 notion of bisimulation and lattice bisimulation.

Recall, that an atom in a Boolean algebra is a minimal element among non-bottom elements and that we call a Boolean algebra $\mathbb{B}$ atomic if each element $b \in \mathbb{B}$ can be written as the supremum of a suitable set of atoms.

Lemma 19. *Let (X, α) be a LaTS over an atomic Boolean algebra $\mathbb{B}$. Then, a conditional relation $R \colon X \times X \to \mathbb{B}$ 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 removing the action labels.

Proof. $\boxed{\Leftarrow}$ Suppose $R \cdot \alpha \sqsubseteq \alpha \cdot R$ and $R^T \cdot \alpha \sqsubseteq \alpha \cdot R^T$, for some $R \colon X \times X \to \mathbb{B}$. 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,

$$\ell \sqsubseteq R(x,x') \sqcap \alpha(x,y) = R^T(x',x) \sqcap \alpha(x,y) \sqsubseteq (R^T \cdot \alpha)(x',y) \sqsubseteq (\alpha \cdot R^T)(x',y).$$

By expanding the last term from above, we find that $\ell \sqsubseteq \alpha(x',y') \sqcap R^T(y',y)$, for some y', because ℓ is irreducible. Thus, $\ell \sqsubseteq \alpha(x',y')$ (i.e. $x' \xrightarrow{\ell} y'$) and $\ell \sqsubseteq R(y,y')$. Similarly, the transition emanating from x' can be simulated using $R \cdot \alpha \sqsubseteq \alpha \cdot R$.

$\boxed{\Rightarrow}$ Let $R \colon X \times X \to \mathbb{B}$ 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. Suppose, for some index set I, we have $(R \cdot \alpha)(x,y') = \bigsqcup_{i\in I} \ell_i$ such that each 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, $(R \cdot \alpha)(x, y') \sqsubseteq (\alpha \cdot R)(x, y')$. $\square$

5. Computation of Lattice Bisimilarity

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 (pointwise) 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 based on the Kleene fixpoint theorem 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 non-standard form of matrix multiplication found in the literature of residuated lattices [17] and database design [18].

5.1. A Fixpoint Approach

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

Definition 20. *Recall the residuum operator ($\to$) on a finite distributive lattice (cf. Definition 1) and define three operators $F, F_1, F_2\colon (X \times Y \to \mathbb{L}) \to (X \times Y \to \mathbb{L})$ in the following way:*

$$\begin{aligned}
F_1(R)(x,y) &= \bigsqcap_{a\in A, x'\in X} \Big(\alpha(x,a,x') \to \big(\bigsqcup_{y'\in Y} (\beta(y,a,y') \sqcap R(x',y'))\big)\Big),\\
F_2(R)(x,y) &= \bigsqcap_{a\in A, y'\in Y} \Big(\beta(y,a,y') \to \big(\bigsqcup_{x'\in X} (\alpha(x,a,x') \sqcap R(x',y'))\big)\Big),\\
F(R)(x,y) &= F_1(R)(x,y) \sqcap F_2(R)(x,y).
\end{aligned}$$

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 21. *A conditional relation R is a lattice bisimulation if and only if R is a post-fixpoint of F, i.e. $R \sqsubseteq F(R)$.*

Proof. $\boxed{\Leftarrow}$ Let $R\colon X \times Y \to \mathbb{L}$ be a conditional relation over a pair of LaTS (X, A, α), (Y, A, β) 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 $x \xrightarrow{a,\ell} x'$ (that is, $\ell \sqsubseteq \alpha(x, a, x')$) 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 proven using F_2.

$\boxed{\Rightarrow}$ Let $R\colon X \times Y \to \mathbb{L}$ 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:

$$\begin{aligned}
F_1(R)(x,y) &= \bigsqcap_{a,x'}(\alpha(x,a,x') \to \bigsqcup_{y'\in Y}(\beta(y,a,y') \sqcap R(x',y'))) \\
&= \bigsqcap_{a,x'} \left\lfloor \bigsqcup_{y'\in Y}(\beta(y,a,y') \sqcap R(x',y')) \sqcup \neg\alpha(x,a,x') \right\rfloor \quad \text{(Lemma 9)} \\
&= \bigsqcap_{a,x'} \bigsqcup\{m \in \mathbb{L} \mid m \sqsubseteq \bigsqcup_{y'\in Y}(\beta(y,a,y') \sqcap R(x',y')) \sqcup \neg\alpha(x,a,x')\}.
\end{aligned}$$

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

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 monotone, and also, the residuum operation is monotone in the second component. Moreover, since we are only considering finite $X, Y, \mathbb{L}$, from the fact that F is monotone it directly follows that F is co-continuous. Thus, we can use the following fixpoint iteration to compute the greatest bisimulation while working with the above assumptions.

Algorithm 1 Partition refinement algorithm for computing the bisimilarity between two CTSs (X,A,α) and (Y,A,β).

Input: Two CTSs (X,A,α) and (Y,A,β), where the sets X, Y, and A are all finite.

Output: A matrix $R \subseteq \mathbb{L}^{X\times Y}$ that is the greatest bisimulation between α and β.

```
set $R_0$ as $R_0(x,y) = 1$ for all $x \in X, y \in Y$;
for $i \in \mathbb{N}$ loop
  compute $R_{i+1} = F(R_i)$
  if $R_i \sqsubseteq R_{i+1}$ return $R_i$
end loop;
```

If we assume $\alpha = \beta$, then it is not hard to see that the fixpoint iteration must stabilise after at most $|X|$ steps, since the R_i always induce equivalence relations for all conditions φ and refinements regarding φ are immediately propagated to every $\varphi' \geq \varphi$. We will now formalise and prove this result. The crux is to prove that the intermediate results $R_i : X \times X \to \mathbb{L}$ induce equivalence relations on X, for which we need the following lemma.

Lemma 22. *Let $x, y \in X$. Then, $\varphi \in F_1(R)(x,y)$ if and only if for all $\varphi' \leq \varphi$, and $a \in A, x' \in X$ it holds that whenever $\varphi' \in \alpha(x,a,x')$ there exists a y' such that $\varphi' \in \alpha(y,a,y')$ and $\varphi' \in R(x',y')$.*

Analogously, $\varphi \in F_2(R)(x,y)$ if and only if for all $\varphi' \leq \varphi$, and $a \in A, y' \in X$ it holds that whenever $\varphi' \in \alpha(y,a,y')$ there exists an x' such that $\varphi' \in \alpha(x,a,x')$ and $\varphi' \in R(x',y')$.

Proof. We will only prove the claim for F_1, it can be proven analogously for F_2. Using Lemma 9, the approximation operation $\lfloor\cdot\rfloor$, and Definition 8, we can observe that

$$\begin{aligned}F_1(R)(x,y) &= \bigsqcap_{a\in A, x'\in X}\left(\alpha(x,a,x') \rightarrow_{\mathbb{L}} \Big(\bigsqcup_{y'\in X}(\alpha(y,a,y') \sqcap R(x',y'))\Big)\right)\\ &= \bigwedge_{a\in A, x'\in X}\left(\left\lfloor \alpha(x,a,x') \rightarrow_{\mathbb{B}} \Big(\bigvee_{y'\in X}(\alpha(y,a,y') \wedge R(x',y'))\Big)\right\rfloor\right).\end{aligned}$$

Now, by the definition of approximation, $\varphi \in \lfloor l \rfloor$ if and only if for all $\varphi' \leq \varphi$ it holds that $\varphi' \in l$. Thus, we can rewrite this as

$$\varphi' \in \left(\alpha(x,a,x') \rightarrow_{\mathbb{B}} \Big(\bigvee_{y'\in Y}(\alpha(y,a,y') \wedge R(x',y'))\Big)\right) \qquad \text{for all } \varphi' \leq \varphi, a \in A, x' \in X.$$

Hence, the well-known characterisation of $\rightarrow_{\mathbb{B}}$ via negation and disjunction yields the expected result. □

Definition 23. *For a given condition relation $R\colon X \times X \to \mathbb{L}$ and $\varphi \in \Phi$, we define the relation $R[\varphi] \subseteq X \times X$ as:*

$$R[\varphi] = \{(x,y) \mid \varphi \in R(x,y)\}.$$

Lemma 24. *Let $R_0, R_1, \ldots, R_n$ be the sequence of conditional relations obtained via Algorithm 1 applied to (X, α) over $\mathbb{L}$, then for all $\varphi \in \Phi$, $R_i[\varphi]$ is an equivalence relation.*

Proof. Let $\varphi \in \Phi$ be chosen arbitrarily, we will now show that $R_i[\varphi]$ is an equivalence relation for all iterations i. We prove this via induction over the index i. For $R_0[\varphi]$ the claim is trivially true, since R_0 is the 1-matrix and therefore $R_0[\varphi] = X \times X$ for all conditions $\varphi \in \Phi$.

Assume that for all $\varphi \in \Phi$, $R_i[\varphi]$ is an equivalence relation and we show that then $R_{i+1}[\varphi]$ is an equivalence relation for all $\varphi \in \Phi$ as well. Let $\varphi \in \Phi$. We can now check the three conditions of equivalence relations.

- *Reflexivity:* Let $x \in X$. Then, via the induction hypothesis we obtain $(x,x) \in R_i[\varphi]$ for all φ. Our aim is to show $(x,x) \in R_{i+1}[\varphi]$ for all φ, where $R_{i+1}[\varphi] = F_1(R_i)[\varphi] \cap F_2(R_i)[\varphi]$.

  We prove that $(x,x) \in F_1(R_i)[\varphi]$ using Lemma 22: let $\varphi' \leq \varphi$, $a \in A$, $x' \in X$. Whenever $\varphi' \in \alpha(x,a,x')$ we can choose $y' = x'$, where $\varphi' \in R_i(x',x')$ (since $R_i[\varphi']$ is reflexive by induction hypothesis). This implies $(x,x) \in F_1(R_i)[\varphi]$. The case of $F_2(R_i)$ is analogous.

- *Symmetry:* The proof of symmetry follows directly from the construction of $F_1(R)$ and $F_2(R)$. In particular,

$$F_2(R_i)(x,y) = \bigsqcap_{a\in A, y'\in Y}\left(\alpha(y,a,y') \rightarrow_{\mathbb{L}} \bigsqcup_{x'\in X}(\alpha(x,a,x') \wedge R_i(x',y'))\right) \qquad \text{(Definition 20)}$$

$$= \prod_{a\in A, y'\in Y} \left(\alpha(y,a,y') \to_{\mathbb{L}} \bigsqcup_{x'\in X} (\alpha(x,a,x') \wedge R_i(y',x'))\right) \qquad \text{(Induction hypothesis)}$$
$$= F_1(R_i)(y,x).$$

- *Transitivity:* Let $(x,y) \in R_{i+1}[\varphi]$ and $(y,z) \in R_{i+1}[\varphi]$, where $R_{i+1}[\varphi] = F_1(R_i)[\varphi] \cap F_2(R_i)[\varphi]$. We prove that $(x,z) \in F_1(R_i)[\varphi]$ using Lemma 22: let $\varphi' \le \varphi$, $a \in A$, $x' \in X$. Whenever $\varphi' \in \alpha(x,a,x')$, we know from the characterization of $F_1(R_i)$ in Lemma 22 that there exists a y' such that $\varphi' \in \alpha(y,a,y')$ and $\varphi' \in R_i(x',y')$. Using the same argument again, we obtain a z' such that $\varphi' \in \alpha(z,a,z')$ and $\varphi' \in R_i(y',z')$. Hence $(x',y'),(y',z') \in R_i[\varphi']$ and by transitivity we obtain $(x',z') \in R_i[\varphi']$, hence $\varphi' \in R_i(x',z')$. Now, applying Lemma 22 again (using the other direction of the implication this time), we get $\varphi \in F_1(R_i)(x,z)$, which is equivalent to $(x,z) \in F_1(R_i)[\varphi]$, as desired. The case of $F_2(R_i)$ is analogous. □

Lemma 25. *If, for all $\varphi' \le \varphi$, $F(R_i)[\varphi'] = R_i[\varphi']$, then $F(F(R_i))[\varphi'] = R_i[\varphi']$ for all $\varphi' \le \varphi$, i.e. if the fixpoint iteration does not modify the relation for φ and all smaller conditions φ', it becomes stationary.*

Proof. It is sufficient to show that $\varphi \in F(F(R_i))(x,y) \iff \varphi \in F(R_i)(x,y)$, for any $x,y \in X$ and $\varphi \in \Phi$.

$$\begin{aligned}
\varphi \in{} & F(F(R_i))(x,y) \\
&\iff \forall \varphi' \le \varphi, a \in A, x' \in X : \varphi' \in \alpha(x,a,x') \implies \exists y' \in X : \varphi' \in \alpha(y,a,y') \wedge \varphi' \in F(R_i)(x',y') \\
&\iff \forall \varphi' \le \varphi, a \in A, x' \in X : \varphi' \in \alpha(x,a,x') \implies \exists y' \in X : \varphi' \in \alpha(y,a,y') \wedge \varphi' \in R_i(x',y') \\
&\iff \varphi \in F(R_i)(x,y).
\end{aligned}$$
□

Lemma 26. *$R[\varphi] \subseteq R[\varphi']$ for all $\varphi' \le \varphi$.*

Proof. Since $R(x,y)$ is downward closed, we obtain $\varphi \in R(x,y) \implies \varphi' \in R(x,y) \implies (x,y) \in R[\varphi']$. □

Theorem 27. *The fixpoint iteration of Algorithm 1, applied to the system (X,α) terminates after at most $|X|$ steps.*

Proof. Fix $\varphi \in \Phi$. We show that the fixpoint iteration becomes stationary for φ after at most $|X|$ iterations, then, from Lemma 25 it follows that the whole fixpoint iteration must become stationary after at most $|X|$ steps, because the argument can be applied to all conditions independently of each other. Consider the set

$$\Phi_i = \{\varphi' \le \varphi \mid \exists \varphi'' \le \varphi' : R_i[\varphi''] \ne R_{i+1}[\varphi'']\}$$

and we show inductively that for all $\psi \in \Phi_i$, it holds that $R_{i+1}[\psi]$ has at least $i+1$ equivalence classes. For $i=0$ this is trivially true, because every equivalence relation contains at least one equivalence class. Now, assume the claim is true for i, and show that it also holds for $i+1$. If $\psi \in \Phi_{i+1}$, then there exists a $\psi' \le \psi$ such that $R_{i+1}[\psi'] \ne R_{i+2}[\psi']$. The induction hypothesis yields that $R_{i+1}[\psi']$ has at least $i+1$ equivalence classes, therefore the equivalence relation $R_{i+2}[\psi'] \subset R_{i+1}[\psi']$ must have at least $i+2$ equivalence classes. Since $R_{i+2}[\psi] \subseteq R_{i+2}[\psi']$, $R_{i+2}[\psi]$ must also contain at least $i+2$ equivalence classes.

So for all $\psi \in \Phi_i$, $R_{i+1}[\psi]$ has at least $i+1$ equivalence classes. Now assume the algorithm made $|X|$ steps, then this would yield that for all $\psi \in \Phi_{|X|}$, $R_{|X|+1}[\psi]$ has at least $|X|+1$ equivalence classes. Since there are only $|X|$ states, such an equivalence relation cannot exist, so $\Phi_{|X|}$ is necessarily empty, i.e. the algorithm terminates. □

5.2. *Lattice Bisimilarity is Finer than Boolean Bisimilarity*

We now show the close relation of the notions of bisimilarity for an 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 monotone operators as defined in Definition 20 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 28.

(i) *If $R\colon X \times Y \to \mathbb{B}$ where $R(X,Y) \subseteq \mathbb{L}$ (i.e. all the entries of R are in $\mathbb{L}$), 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 $R\colon X \times Y \to \mathbb{B}$ is an $\mathbb{L}$-bisimulation, whenever $R(X,Y) \subseteq \mathbb{L}$.*

Proof.

(i) This follows directly from Lemma 9, which allows us to move the approximations to the inside towards the implications and to approximate the implication in $\mathbb{B}$ via the residuum 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 $F_{\mathbb{L}}(R) = \lfloor F_{\mathbb{B}}(R) \rfloor$ (using (i)), 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. □

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 11 where we encounter a $\mathbb{B}$-bisimulation, which is not an $\mathbb{L}$-bisimulation).

5.3. *Matrix Multiplication*

An alternative way to represent a LaTS (X, A, α) is to view the transition function α as a family of matrices $\alpha_a\colon X \times X \to \mathbb{L}$ (one for each action $a \in A$), where the function α_a is defined as follows: $\alpha_a(x, x') = \alpha(x, a, x')$, for every $x, x' \in X$. We use standard matrix multiplication denoted by $\cdot$ (where $\sqcup$ is used for addition and $\sqcap$ for multiplication), as well as a special form of matrix multiplication [17, 18].

Definition 29 (Matrix Residuum)**.** *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 matrix residuum $\otimes$ of U and V as follows:*

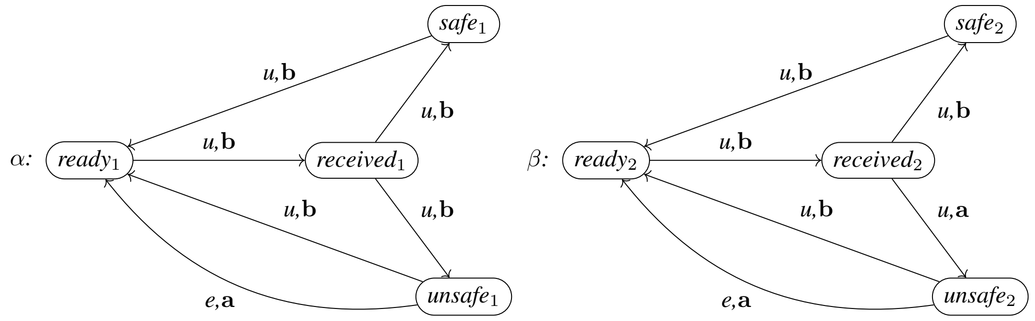
$$(U \otimes V)(x,z) = \bigsqcap_{y \in Y} \big(U(x,y) \to_{\mathbb{L}} V(y,z)\big) \ .$$

We obtain the following alternative characterisation of the fixpoint operator F.

Theorem 30. *Let $R\colon X \times Y \to L$ be a conditional relation between a pair of LaTSs (X, A, α) and (Y, A, β). Then, $F(R) = \bigsqcap_{a \in A}((\alpha_a \otimes (R \cdot {\beta_a}^T)) \sqcap (\beta_a \otimes (\alpha_a \cdot R)^T)^T)$, where R^T denotes the transpose of a matrix R.*

To illustrate the fixpoint iteration and its representation via matrix multiplication, we consider a slightly simplified version of the routing protocol from previous examples.

Example 31. *Consider the following pair of systems α, β, a slightly modified version of the systems in Example 11. (The purpose of the modification is just to have fewer labels and hence fewer matrices, because otherwise the computations would get unwieldy.)*



We present these systems by the matrices $\alpha_u, \alpha_e, \beta_u, \beta_e$, where the columns and rows refer to the states $ready_i$, $received_i$, $safe_i$, $unsafe_i$ (in that order). For e.g., the entry $\{\mathbf{a}, \mathbf{b}\}$ in the third line, first column of α_u represents the information that it is possible to make a u step from state $safe_1$ to $ready_1$ under both conditions, $\mathbf{a}$ and $\mathbf{b}$.

$$\alpha_u = \begin{pmatrix} \emptyset & \{\mathbf{a},\mathbf{b}\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} \\ \{\mathbf{a},\mathbf{b}\} & \emptyset & \emptyset & \emptyset \\ \{\mathbf{a},\mathbf{b}\} & \emptyset & \emptyset & \emptyset \end{pmatrix} \qquad \alpha_e = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \{\mathbf{a}\} & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

$$\beta_u = \begin{pmatrix} \emptyset & \{\mathbf{a},\mathbf{b}\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a}\} \\ \{\mathbf{a},\mathbf{b}\} & \emptyset & \emptyset & \emptyset \\ \{\mathbf{a},\mathbf{b}\} & \emptyset & \emptyset & \emptyset \end{pmatrix} \qquad \beta_e = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \{\mathbf{a}\} & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

Now we can execute the matrix multiplication algorithm as follows: we initiate the algorithm with the constant 1*-matrix.*

$$R_0 = \begin{pmatrix} \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} \\ \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} \\ \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} \\ \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} \end{pmatrix}$$

In the next steps we apply F *using the following formula*

$$F(R) = (\alpha_u \otimes (R \cdot \beta_u^T)) \sqcap (\beta_u \otimes (\alpha_u \cdot R)^T)^T \sqcap (\alpha_e \otimes (R \cdot \beta_e^T)) \sqcap (\beta_e \otimes (\alpha_e \cdot R)^T)^T.$$

$$R_1 = F(R_0) = \begin{pmatrix} \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \emptyset \\ \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \emptyset \\ \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \{\mathbf{a},\mathbf{b}\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{\mathbf{a},\mathbf{b}\} \end{pmatrix} \quad R_2 = F(R_1) = \begin{pmatrix} \{\mathbf{a},\mathbf{b}\} & \emptyset & \{\mathbf{a},\mathbf{b}\} & \emptyset \\ \emptyset & \{\mathbf{a}\} & \emptyset & \emptyset \\ \{\mathbf{a},\mathbf{b}\} & \emptyset & \{\mathbf{a},\mathbf{b}\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{\mathbf{a},\mathbf{b}\} \end{pmatrix}$$

$$R_3 = F(R_2) = \begin{pmatrix} \{\mathbf{a}\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \{\mathbf{a}\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{\mathbf{a},\mathbf{b}\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{\mathbf{a},\mathbf{b}\} \end{pmatrix} \quad R_4 = F(R_3) = \begin{pmatrix} \{\mathbf{a}\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \{\mathbf{a}\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{\mathbf{a}\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{\mathbf{a}\} \end{pmatrix} = F(R_4)$$

The algorithm stops after computing $R_5 = F(R_4) = R_4$. *We conclude that* $x_1 \sim_{\mathbf{a}} x_2$ *for all* $x \in \{\mathit{ready}, \mathit{received}, \mathit{safe}, \mathit{unsafe}\}$ *but no other pairs of states are bisimilar under* $\mathbf{a}$ *and no pair of states is bisimilar under* $\mathbf{b}$.

In a Boolean algebra, it is well known that the residuum operator can be eliminated by the negation and join operators. Thus, in this case, using only the standard matrix multiplication and (component wise) negation we get $U \otimes V = \neg(U \cdot (\neg V))$. Hence, a conditional relation $F(R)$ can be rewritten as:

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

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

6. Bisimulation Game

The characterisation of bisimulation equivalence via a two-player game, so-called bisimulation game, is useful in two ways. Firstly, it provides an intuitive understanding of the transfer properties of a bisimulation relation useful medium for didactic purpose in showing two non-bisimilar states. Secondly, the existence of game strategies is a useful proof technique for further research on bisimulation games. This is demonstrated, for instance, in the work of Colin Stirling on Hennessy-Milner logics [19], where the correspondence between the logics and bisimilarity is established using the bisimulation game. Similarly, Katja Poltermann [20] has demonstrated in a recent bachelor thesis, that a related logics for conditional transition systems can be found, making use of the bisimulation game for conditional bisimilarity introduced in this section.

We now show that conditional bisimulation gives rise to a bisimulation game that exactly characterises conditional bisimulation. This game is an adaptation of the bisimulation game for traditional labelled transition systems.

Definition 32 (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 the CTSs as game boards. Let (x, y, φ) be a game instance indicating that x, y are marked and that the current condition is φ. The game progresses to the next game instance as follows:*

- *Player 1 is the first 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 then choose either the marked state $x \in X$ or $y \in Y$ and must perform a transition $x \xrightarrow{a,\varphi'} x'$ or $y \xrightarrow{a,\varphi'} y'$ for some $a \in A$.*
- *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, as expected, bisimulation is characterised as follows: Player 2 has a winning strategy for a game instance (x, y, φ) if and only if $x \sim_\varphi y$. This fact is established over the next two lemmata.

Lemma 33. *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 34. *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. Consider the LaTSs corresponding to the CTSs A, B and compute the fixpoint by using the matrix multiplication algorithm, obtaining a sequence $R_0, R_1, \dots, R_n = R_{n+1} = \dots$ of lattice-valued relations $R_i \colon X \times Y \to \mathcal{O}(\Phi, \leq)$. Instead of using exactly the matrix multiplication method, we can also use the characterisation of Definition 16: whenever there exists a transition $x \xrightarrow{a,\varphi} x'$, for which there is no matching transition with $y \xrightarrow{a,\varphi} y'$ 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 $R_i(x, y)$.

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 consider a 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 $\varphi' \leq \varphi$, define $\omega(\varphi')$ to be the following term:

$$\min\left\{ \min_{a,x'} \left\{ \max_{y'} \left\{ M^{\varphi'}(x', y') \mid y \xrightarrow{a,\varphi'} y' \right\} \mid x \xrightarrow{a,\varphi'} x' \right\}, \min_{a,y'} \left\{ \max_{x'} \left\{ M^{\varphi'}(x', y') \mid x \xrightarrow{a,\varphi'} x' \right\} \mid y \xrightarrow{a,\varphi'} y' \right\} \right\}.$$

The formula can be interpreted as follows: the outer $\min$ corresponds to the choice of making a step in the 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 among all $\varphi' \leq \varphi$ (so a minimal choice for $\arg\min \omega$). 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 $M^{\overline{\varphi}}(x', y') \leq \omega(\overline{\varphi})$.

Next, by contradiction, we show that $\omega(\overline{\varphi}) < i$. 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 $\varphi' \geq \varphi$. That is, 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. $\square$

To illustrate this conditional bisimulation game, we will show how Player 1 may act to prove that the states ready_1 and ready_2 from Example 13 are not bisimilar.

Example 35. *Consider the two systems from Example 13. A winning strategy for Player 1 starting in the states* ready_1 *and* ready_2 *and initial condition* $\mathbf{b}$ *could work as follows:*

- *Choose either* ready_1 *or* ready_2*, do not perform an upgrade and take the only possible transition labelled* $\mathit{receive}$*, leading to the state* $\mathit{received}_1$*, respectively* $\mathit{received}_2$*. Player 2 can only answer this step by doing the* $\mathit{receive}$ *step in the other state. The new game situation is* $(\mathit{received}_1, \mathit{received}_2, \mathbf{b})$*.*

- *Player 1 now must choose state* $\mathit{received}_1$ *and perform no upgrade. Using the* check *transition, Player 1 does a step to the state* unsafe_1*. Player 2 again only has a single choice for his answering step, since no upgrade was performed and therefore, the transition to* unsafe_2 *is not available. Thus, Player 2 has to take the* check *transition to* safe_2*. So the new game situation is* $(\mathit{unsafe}_1, \mathit{safe}_2, \mathbf{b})$*.*

- *Now Player 1 can win the game by performing an upgrade to* $\mathbf{a}$*, enabling Player 1 to do an* e *step from* unsafe_1 *to* ready_1*. Player 2 cannot answer this step, because in* safe_2*, no* e *transition is available.*

7. Deactivating Transitions

We have introduced conditional transition systems (CTS) as a modelling formalism that allows to model a family of systems that are all based on a common design, but with different actions available for different products. Products 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 might not always be suitable, 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 functionality right away.

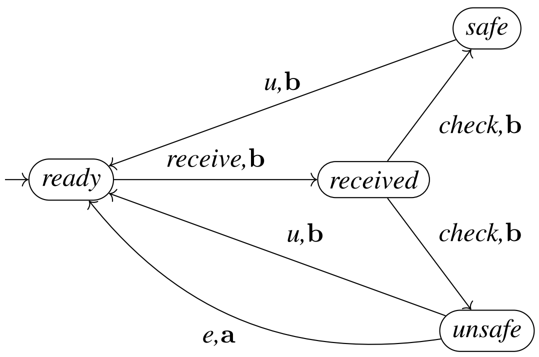
A practical motivation may be derived from our Example 11. In this CTS one may want to model that the advanced version can only send an encrypted message in the unsafe state, since we assume that the user is always interested in a secure communication, ensured either by a safe channel or by encryption. 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. Thus, to model such situations we add priorities that allow to deactivate the unencrypted transition in the presence of an encrypted transition. To this end, we propose a slight variation of the definition of CTS/LaTS.

Definition 36. *A* conditional transition system with action precedence *is a triple* $(X, (A, \leq_A), f)$*, where* (X, A, f) *is a CTS and* $\leq_A$ *is a partial order on* A*. As usual, we will write* $a <_A a'$ *for* $a, a' \in A$ *whenever* $a \leq_A a'$ *and* $a \neq a'$*.*

Intuitively, a CTS with action precedence evolves in a way similar 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. Now all the transitions that have a condition greater than or equal to φ are activated, while the remaining transitions remain 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 $a' >_A a$, i.e. a' takes precedence over a, then $x \xrightarrow{a,\varphi} x'$ is considered to be inactive as well.* 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.*

For the remainder of this section, we will just write CTS for CTS with action precedence.

Example 37. *Consider a refinement of the adaptive routing protocol modelled as a CTS (cf. Example 11) over the alphabet $A = \{receive, check, u, e\}$ by adding an action precedence $<_A$ where $u <_A e$. The transition function remains unchanged and so does its visual representation.*


$$\begin{tikzcd}
 & & \mathit{safe} \arrow[dll, "{u,\mathbf{b}}"'] \\
\arrow[r] & \mathit{ready} \arrow[r, "{receive,\mathbf{b}}"] & \mathit{received} \arrow[u, "{check,\mathbf{b}}"'] \arrow[d, "{check,\mathbf{b}}"] \\
 & & \mathit{unsafe} \arrow[ull, "{u,\mathbf{b}}"'] \arrow[ull, bend left, "{e,\mathbf{a}}"]
\end{tikzcd}$$

The system retains two products: the basic *system with no encryption feature written as* $\mathbf{b}$ *and the* advanced *system with encryption feature written as* $\mathbf{a}$. *However, the action precedence changes the behaviour in the advanced version of the system. Under* $\mathbf{a}$, *in state $unsafe$, it is not anymore possible to make a u (unencrypted) transition, since the alternative e (encrypted) transition has precedence over u and is available in state $unsafe$ as well. Therefore, no non-deterministic choice may occur anymore in presence of u and e and the system model enforces that in unsafe environments, only encrypted messages can be sent. However, under product* $\mathbf{b}$, *in state $unsafe$, the u transition remains active, since no e transition can take precedence over it. Similarly, also in state $safe$, for product* $\mathbf{a}$, *the u transition remains active.*

This changed interpretation of the behaviour of a CTS of course also has an effect on the bisimulation, which is influenced by the labelled transition systems associated with each condition $\varphi \in \Phi$.

Definition 38. *Let $(X, (A, \leq_A), f)$ and $(Y, (A, \leq_A), g)$ be two CTSs with action precedence 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 $(X, (A, \leq_A), f)$*, where*

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

Two states $x \in X, y \in Y$ *are* conditionally bisimilar (w.r.t. 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 action precedence, 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 39. *Let* $(X, (A, \leq_A), \alpha)$ *and* $(Y, (A, \leq_A), \beta)$ *be LaTSs (with actions precedence). Recall the residuum operator (*$\rightarrow$*) on a finite distributive lattice and define three operators* $G, G_1, G_2 \colon (X \times Y \to \mathbb{L}) \to (X \times Y \to \mathbb{L})$ *in the following way:*

$$\begin{aligned}
G_1(R)(x, y) &= \bigsqcap_{a \in A, x' \in X} \Big(\alpha(x, a, x') \to \Big(\bigsqcup_{y' \in Y} \Big(\beta(y, a, y') \sqcap R(x', y')\Big) \sqcup \bigsqcup_{a' >_A a, x'' \in X} \alpha(x, a', x'')\Big)\Big)\\
G_2(R)(x, y) &= \bigsqcap_{a \in A, y' \in Y} \Big(\beta(y, a, y') \to \Big(\bigsqcup_{x' \in X} \Big(\alpha(x, a, x') \sqcap R(x', y')\Big) \sqcup \bigsqcup_{a' >_A a, y'' \in Y} \beta(y, a', y'')\Big)\Big)\\
G(R)(x, y) &= G_1(R)(x, y) \sqcap G_2(R)(x, y).
\end{aligned}$$

Now, we need to show that we can characterise the new notion of bisimulations as post-fixpoints of this operator G.

Theorem 40. *Let* $(X, (A, \leq_A), f)$*,* $(Y, (A, \leq_A), g)$ *be two CTSs with action precedence over the partially ordered set* $(\Phi, \leq)$ *and let* $(X, (A, \leq_A), \alpha)$*,* $(Y, (A, \leq_A), \beta)$ *over* $\mathcal{O}(\Phi)$ *be the corresponding LaTSs. For 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 *(i.e.* $R \sqsubseteq G(R)$*) such that* $\varphi \in R(x, y)$*.*

Proof. Let $\ell_1, \ell_2 \in \mathbb{L}$. From Lemma 9 we know, that $\varphi \in (\ell_1 \to \ell_2)$ is equivalent to showing that for all $\varphi' \leq \varphi$, $\varphi' \notin \ell_1$ or $\varphi' \in \ell_2$. (*)

$\boxed{\Leftarrow}$ Assume $R \sqsubseteq G(R)$ and let $\varphi \in R(x, y)$, for some $x \in X, y \in Y$. Define a family of relations:

$$(x', y') \in R_{\varphi'} \iff \varphi' \in R(x', y') \qquad \text{(for each } \varphi' \leq \varphi\text{)}.$$

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, remember that we assume $\varphi \in R(x, y)$, thus $(x, y) \in R_{\varphi'}$ must hold for all $\varphi' \leq \varphi$. So we only need to show that each $R_{\varphi'}$ is a traditional bisimulation 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'' exist, then the first bisimulation condition is trivially true. For G_1, it must be true that $\varphi' \in G_1(R)(x, y)$. Thus,

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

Using (*) above and the fact $\varphi' \in \alpha(x', a, x'')$ (because $x'' \in \bar{f}_{\varphi'}(x', a)$), we obtain that

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

Per definition of $\bar{f}_{\varphi'}$, there is no $a' >_A a$ such that $\bar{f}_{\varphi'}(x'', a') \neq \emptyset$, i.e. $\varphi' \notin \bigsqcup_{a' >_A a, x''' \in X} \alpha(x', a', x''')$. Thus, $\varphi' \in \bigsqcup_{y'' \in Y} \beta(y', a, y'') \sqcap R(x'', y'')$, i.e. $\varphi' \in \beta(y', a, y'') \sqcap R(x'', y'')$, for some $y'' \in Y$. Clearly, $(x'', y'') \in R_{\varphi'}$.

Now we show that $y'' \in \bar{g}_{\varphi'}(y', a)$ holds. 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 $a' >_A a, y''' \in Y$ such that $\varphi' \in \beta(y', a', y''')$. Without loss of generality, choose a maximal element $a' >_A a$. Since $(x', y') \in R_{\varphi'}$ and $R \sqsubseteq G(R)$, it is clear that $\varphi' \in G_2(R)(x', y')$. Thus,

$$\varphi' \in \Big(\beta(y', a', y''') \to \Big(\bigsqcup_{x''' \in X} \Big(\alpha(x', a', x''') \sqcap R(x''', y''') \Big) \sqcup \bigsqcup_{a'' >_A a', y'''' \in Y} \beta(y', a'', y'''') \Big) \Big). \tag{1}$$

Since we chose a' maximal, we know that $\varphi' \notin \bigsqcup_{a'' >_A a', y'''' \in Y} \beta(y', a'', y'''')$. Moreover, since $a' >_A a, x'' \in \bar{f}_{\varphi'}(x', a)$, there is no x''' such that $\varphi' \in \alpha(x', a', x''')$. Thus, φ' is not in the right side of the residuum occurring in (1), yet it is in the left side of the residuum; therefore, it is not in the residuum. Thus, $\varphi' \notin G_2(R)(x', y')$, which is a contradiction.

Likewise the symmetric condition can be proven by reversing the roles of G_2 and G_1 to find the answer step in $\bar{f}_{\varphi'}$.

$\boxed{\Rightarrow}$ Let $(R_\varphi)_\varphi$ be a family of bisimulations between $\bar{f}_\varphi, \bar{g}_\varphi$ such that $\varphi_1 \leq \varphi_2$ implies $R_{\varphi_1} \supseteq R_{\varphi_2}$, for all $\varphi_1, \varphi_2 \in \Phi$. Define $R\colon X \times Y \to \mathbb{L}$ as: $R(x, y) = \{\varphi \mid (x, y) \in R_\varphi\}$. Due to anti-monotonicity of the family $(R_\varphi)_\varphi$ all entries in R are indeed lattice elements from $\mathcal{O}(\Phi, \leq)$. So it only remains to be shown that R is a post-fixpoint.

Let $x \in X, y \in Y, \varphi \in \Phi$ be such that $\varphi \in R(x, y)$. We now show $\varphi \in G_1(R)(x, y)$ since the symmetric case $\varphi \in G_2(R)(x, y)$ can be shown similarly. Thus, for any $a \in A$, we need to show that

$$\varphi \in \Big(\alpha(x, a, x') \to \Big(\bigsqcup_{y' \in Y} \Big(\beta(y, a, y') \sqcap R(x', y') \Big) \sqcup \bigsqcup_{a' >_A a, x'' \in X} \alpha(x, a', x'') \Big) \Big). \tag{2}$$

So let $\varphi' \leq \varphi$. By using (*) above it suffices to show that

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

whenever $\varphi' \in \alpha(x, a, x')$. In order to prove this let $a \in A$. We distinguish according to whether a is maximal such that $\varphi' \in \alpha(x, a, x'')$:

- Suppose there is no $a' >_A a$ such that $\varphi' \in \alpha(x, a, x'')$, for any $x'' \in X$. Then there is 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.
- Suppose there is an $a' >_A a$ such that $\varphi' \in \alpha(x, a, x'')$, for some $x'' \in X$. Then $\varphi' \in \bigsqcup_{a' >_A a, x'' \in X} \alpha(x, a', x'')$.

From this we can conclude that Equation (2) holds. □

Hence we can compute the bisimulation via a fixpoint iteration, as with LaTS without an ordering on the 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, $\ell_1 \to (\ell_2 \sqcup \ell_3) \neq (\ell_1 \to \ell_2) \sqcup \ell_3$, for any $\ell_1, \ell_2, \ell_3 \in \mathbb{L}$. In fact, moving the supremum $\bigsqcup_{a' >_A a, x'' \in X} \alpha(x, a', x'')$ outside of the residuum in G 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 property since the Birkhoff duality (cf. Theorem 5) holds only for (finite) posets (not preorders in general). As a result, the fixpoint algorithm that allows to compute the bisimilarity in parallel for all products would be rendered incorrect.

8. Application to Software Product Lines

In this section, we will consider various relevant aspects of the application of CTSs to software product lines (SPL). We begin by discussing the relation to featured transition systems (FTS), which are the most commonly used model for software product lines. Then we show how FTSs are subsumed by CTSs and how to model CTSs for SPL using BDDs, in order to obtain compact representations. We will in particular define a new variant of BDDs which allows to model upgrading in a specialised form for SPL.

Turning our attention to a real-world example, we show how a car control system can be modelled by a CTS. When compared to the original FTS model, we not only gain a more compact representation, but we can also model additional behaviour, in this case, the option to upgrade the car control system with an additional security systems. Such on-the-fly upgrades are gaining importance in a more software-driven product model for transportation.

Finally, to put the algorithmic performance and the advantages of using BDDs into context, we have built a prototypical implementation, which is based on the tool PAWS for the analysis of weighted systems [21] and ran tests with increasing numbers of features – which is crucial, because the run time can in principle grow doubly-exponential in the number of features.

8.1. 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" [22]. 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. [23]). In this section, we show that featured transition system (FTS) – a well-known formal model that is expressive enough to specify an SPL – is a special instance of a CTS. We begin by giving the definition of an FTS (taken from [7]).

Definition 41. *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,* $\gamma\colon T \to \mathbb{B}(N)$ *assigns a Boolean expression over* N *to each transition.*

FTSs are often accompanied by a so-called *feature diagram* [6, 8, 13], 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 can also be seen as a special case of a LaTS due to Theorem 15 and $\mathcal{O}([\![d]\!], =) = \mathcal{P}([\![d]\!])$. Given an FTS $\mathcal{F} = (X, A, T, \gamma)$ and a feature diagram d, then the corresponding LaTS is (X, A, α), where the function α is defined as follows: $\alpha(x, a, y) = [\![\gamma(x, a, y) \wedge d]\!]$, 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 $C, C' \in [\![d]\!]$:

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

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 sequel we will consider the lattice $\mathcal{O}([\![d]\!], \leq)$ of all downward-closed subsets of $[\![d]\!]$.

8.2. BDDs as Models for Boolean Formulae

In this section, we discuss our implementation of the lattice bisimulation check 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; non-adaptive SPLs based on Boolean algebras are a special case. BDD-based implementations of FTSs without upgrades have already been discussed in [4].

A *binary decision diagram* (BDD) is a rooted, directed, and acyclic graph that represents a Boolean expression $b \in \mathbb{B}(N)$. Every BDD has two distinguished terminal nodes 1 and 0, representing the logical constants *true* and *false*. The inner nodes of the BDD are labelled by the atomic propositions in N 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, where we choose the high-successor whenever the label of a node is in C and the low-successor otherwise. If we arrive at the terminal node labelled 1 we have established that $C \models b$, otherwise $C \not\models b$.

We will use a special class of BDDs, called *reduced and ordered BDDs* (ROBDDs) – full details in [24] – 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 removed 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 simple polynomial-time implementations for the basic operations – negation, conjunction, and disjunction. These are however sensitive to the ordering of atomic propositions and an exponential blow-up cannot be ruled out, but often it can be avoided.

Consider a Boolean formula b with $[\![b]\!] = \{\emptyset, \{f_2, f_3\}, \{f_0, f_1\}, \{f_0, f_1, f_2, f_3\}\}$ and the ordering on the atomic propositions as f_0, f_1, f_2, f_3. Figure 2 shows the ROBDD for b, where the inner nodes, terminal nodes, and high (low) edges are depicted as circles, rectangles, and solid (dashed) lines, respectively.

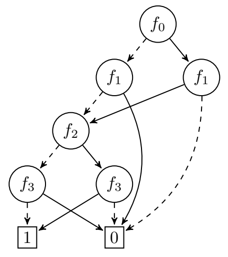


Figure 2: BDD for b.

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 $low(b) = b_0$. As a result, the elements of the Boolean algebra $\mathcal{P}(\mathcal{P}(N))$ *correspond exactly* to ROBDDs over N. Hence we will use b to stand both for a boolean expression as well as for an ROBDD.

8.3. BDDs for Lattices

We now discuss how ROBDDs can also be used to specify and manipulate elements of the lattice $\mathcal{O}([\![d]\!], \leq)$. In particular, computing the infimum and the supremum in the lattice $\mathcal{O}([\![d]\!], \leq)$ is standard, since this lattice can be embedded into $\mathcal{P}(\mathcal{P}(N))$ and the infimum and supremum operations coincide in both the structures. Therefore, it remains to represent the lattice elements and the residuum.

We say that an ROBDD b is *downward-closed* with respect to $\leq$ (or simply, downward-closed) if the set of configurations $[\![b]\!]$ is downward-closed with respect to $\leq$. The following lemma characterises when an ROBDD b is downward closed. Note that $F \in \mathcal{P}(\mathcal{P}(N))$ is downward-closed if and only if $\forall C \in F, f \in U$ $(C \cup \{f\} \in F)$.

Lemma 42. *Let $U \subseteq N$ be a set of upgrade features. An ROBDD b over N is downward-closed if and only if for each node labelled with an element of U, the* low*-successor implies the* high*-successor.*

Proof. $\boxed{\Rightarrow}$ Assume that $low(n) \models high(n)$ for all nodes n of b that are associated with upgrade features (i.e. $root(n) \in U$). Let $C' \in [\![b]\!]$ and $C \leq C'$. Without loss of generality we can assume that $C = C' \cup \{f\}$ for some $f \in U$ (for other choice of C, this then 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.

$\boxed{\Leftarrow}$ 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 $C \notin [\![b]\!]$. Thus a contradiction since $[\![b]\!]$ is downward-closed. □

The next step is to compute a residuum in $\mathcal{O}([\![d]\!], \leq)$ 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 (equivalently represented as an ROBDD) in the lattice $\mathcal{O}(\mathcal{P}(N), \leq)$.

Algorithm 2 Approximation $\llfloor b \rrfloor$ of an ROBDD b in $\mathcal{O}(\mathcal{P}(N), \leq)$

Input: An ROBDD b over a set of features N and a set of upgrade features $U \subseteq N$.

Output: An ROBDD $\llfloor b \rrfloor$, which is the best approximation of b in the lattice.

```
if $b$ is a leaf then return $b$
if $root(b) \in U$ then
    return build($root(b)$, $\llfloor high(b) \rrfloor$, $\llfloor high(b) \rrfloor \wedge \llfloor low(b) \rrfloor$)
return build($root(b)$, $\llfloor high(b) \rrfloor$, $\llfloor low(b) \rrfloor$)
```

In the above algorithm, for each non-terminal node that carries a label in U, we replace the *low*-successor with the conjunction of the *low* and the *high*-successor using the above procedure. 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 $\llfloor b \rrfloor$ coincides with the approximation $\lfloor [\![b]\!] \rfloor$ of the ROBDD b.

Lemma 43. *Let b be an ROBDD. Then $\llfloor b \rrfloor$ is downward-closed. Furthermore, $\llfloor b \rrfloor \models b$ and there is no other downward-closed ROBDD b' such that $\llfloor b \rrfloor \models b' \models b$. Hence, $\llfloor b \rrfloor = \lfloor [\![b]\!] \rfloor$.*

Proof. We first show that $\llfloor b \rrfloor$ as obtained by Algorithm 2 is downward-closed via induction over the number of different features occurring in b. If b only consists of a leaf node, then $\llfloor b \rrfloor$ is certainly downward-closed. Otherwise, we know from the induction hypothesis that $\llfloor high(b) \rrfloor$, $\llfloor low(b) \rrfloor$ are downward-closed. If $root(b) \notin U$, then $\llfloor b \rrfloor$ is downward-closed due to Lemma 42. If, however, $root(b) \in U$, then $\llfloor high(b) \rrfloor \wedge \llfloor low(b) \rrfloor$ is downward-closed (since downward-closed sets are closed under intersection). Furthermore $\llfloor high(b) \rrfloor \wedge \llfloor low(b) \rrfloor \models \llfloor high(b) \rrfloor$, i.e. the new *low*-successor implies the *high*-successor. Thus, the characterising condition of Lemma 42 is satisfied at the root and elsewhere in the BDD; hence, $\llfloor b \rrfloor$ is downward-closed.

For the uniqueness of $\llfloor b \rrfloor$, first observe that $\llfloor b \rrfloor \models b$ since from the construction a *low*-successor is always replaced by a stronger *low*-successor. Thus, it remains to show that there is no other downward-closed ROBDD b' such that $\llfloor b \rrfloor \models b' \models b$. Assume to the contrary, that there exists such a downward-closed BDD b'. Hence, there exists a configuration $C \subseteq N$, such that $C \not\models \llfloor b \rrfloor$, $C \models b'$, $C \models b$. Choose C maximal w.r.t. 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 $b = b_0, \ldots, b_m = \llfloor b \rrfloor$ of BDDs that is constructed by the approximation algorithm (Algorithm 2), where the BDD structure is upgraded bottom-up. We have $\llfloor b \rrfloor = b_m \models b_{m-1} \models \ldots \models b_0 = b$, since in each newly constructed BDD $low(n)$ is replaced by $high(n) \wedge low(n)$, for some node n with $root(n) \in U$.
Since $C \models b$ and $C \not\models \llfloor b \rrfloor$, there must be an index k such that $C \models b_k$ and $C \not\models b_{k+1}$. Let n be the node that is modified in step k with $root(n) = f \in U$. Then $f \notin C$, 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 letting $C' = C \cup \{f\} \models b$ contradicts the maximality of C; hence, $C \cup \{f\} \not\models b$, as required. □

For each node in the BDD we compute at most one supremum, which is quadratic. Hence the entire run-time of the approximation procedure is at most cubic. Finally, we discuss how to compute the residuum in $\mathcal{O}(\llbracket d \rrbracket, \leq)$. Notice that we identify a Boolean expression and its ROBDD as syntactically the same; thus, below we use the Boolean operations $\vee, \wedge$ to compose ROBDDs when viewed as elements of either the lattice $\mathcal{O}(\llbracket d \rrbracket)$ or the Boolean algebra $\mathcal{P}(\mathcal{P}(N))$. This notational convenience is not problematic since the two operations of meet and join coincide in both structures. Furthermore, below, by abuse of notation we write $\llbracket b \rrbracket$ simply as b for a Boolean expression b.

Now we are ready to show how to compute a residuum on ROBDDs.

Proposition 44. *Let b_1, b_2 be two ROBDDs representing elements of $\mathcal{O}(\llbracket d \rrbracket, \leq)$, i.e. both b_1, b_2 are downward-closed and $b_1 \models d$, $b_2 \models d$. Then, the following statements are valid.*

(i) The residuum $b_1 \to b_2$ in the lattice $\mathcal{O}(\llbracket d \rrbracket, \leq)$ is given by $\llfloor \neg b_1 \vee b_2 \vee \neg d \rrfloor \wedge d$.

(ii) If d is downward-closed, then the above residuum simplifies to $b_1 \to b_2 = \llfloor \neg b_1 \vee b_2 \rrfloor \wedge d$.

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

Proof. 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 $\mathbb{B}_d = \mathcal{P}(\llbracket d \rrbracket)$ and one on the lattice $\mathbb{L}_d = (\mathcal{O}(\llbracket d \rrbracket), \leq_d)$ where $\leq_d = \leq|_{\llbracket d \rrbracket \times \llbracket d \rrbracket}$. 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 $\leq_d$ is written $\lfloor \_ \rfloor_d$. Next, we claim that

$$\lfloor b \rfloor_d \equiv \lfloor b \vee \neg d \rfloor \wedge d \qquad \text{(for any } b \in \mathbb{B}_d\text{).} \tag{3}$$

($\Leftarrow$) We show $\lfloor b \vee \neg d \rfloor \wedge d \models \lfloor b \rfloor_d$. Clearly, we have

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

Since $\lfloor b \vee \neg d \rfloor \wedge d$ implies d, it certainly is in $\mathbb{B}_d$. We now show that it is downward-closed w.r.t. $\leq_d$: we use an auxiliary relation $\leq''$, which is the smallest partial order on $\mathbb{B}$ that contains $\leq_d$, i.e. $\leq_d$ extended to $\mathbb{B}$. We have $\leq'' \subseteq \leq$. Since $\lfloor b \vee \neg d \rfloor$ is an approximation, it is downward-closed w.r.t. $\leq$ and hence downward-closed w.r.t. $\leq''$. Moreover, d is downward-closed relative to $\leq''$ (obvious by definition). Since the intersection of two downward-closed sets is again downward-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 $\leq_d$, provided we discuss an element from $\mathbb{B}_d$, we can conclude that $\lfloor b \vee \neg d \rfloor \wedge d$ belongs to $\mathbb{L}_d$. From $\lfloor b \vee \neg d \rfloor \wedge d \in \mathbb{L}_d$ and $\lfloor b \vee \neg d \rfloor \wedge d \models b$ it follows that $\lfloor b \vee \neg d \rfloor \wedge d \models \lfloor b \rfloor_d$ by definition of the approximation.

($\Rightarrow$) We show $\lfloor b \rfloor_d \models \lfloor b \vee \neg d \rfloor \wedge d$. Let $C \in \mathcal{P}(N)$ be such that $C \in [\![\lfloor b \rfloor_d]\!]$. We show that in this case $C \in [\![\lfloor b \vee \neg d \rfloor \wedge d]\!]$, which proves $\lfloor b \rfloor_d \models \lfloor b \vee \neg d \rfloor \wedge d$. Let $\downarrow C$ be the downwards-closure of C w.r.t. $\leq$. Since $\lfloor b \rfloor_d$ must be downward-closed relative to $\leq_d$, it holds that $\downarrow C \cap [\![d]\!] \subseteq [\![\lfloor b \rfloor_d]\!]$. Disjunction with $\neg d$ on both sides yields $\downarrow C \subseteq [\![\lfloor b \rfloor_d \vee \neg d]\!] \subseteq [\![b \vee \neg d]\!]$, since $c \models c \vee \neg d \equiv (c \wedge d) \vee \neg d$. The set $\downarrow C$ is downwards-closed w.r.t. $\leq$, so it is contained in the approximation relative to $\leq$ of this set, i.e $\downarrow C \subseteq [\![\lfloor b \vee \neg d \rfloor]\!]$. Thus, in particular, $C \in [\![\lfloor b \vee \neg d \rfloor]\!]$. Since $C \in [\![\lfloor b \rfloor_d]\!]$, it follows that $C \in [\![d]\!]$, therefore we can conclude $C \in [\![\lfloor b \vee \neg d \rfloor \wedge d]\!]$.

Now we can prove the statement in (i) as follows:

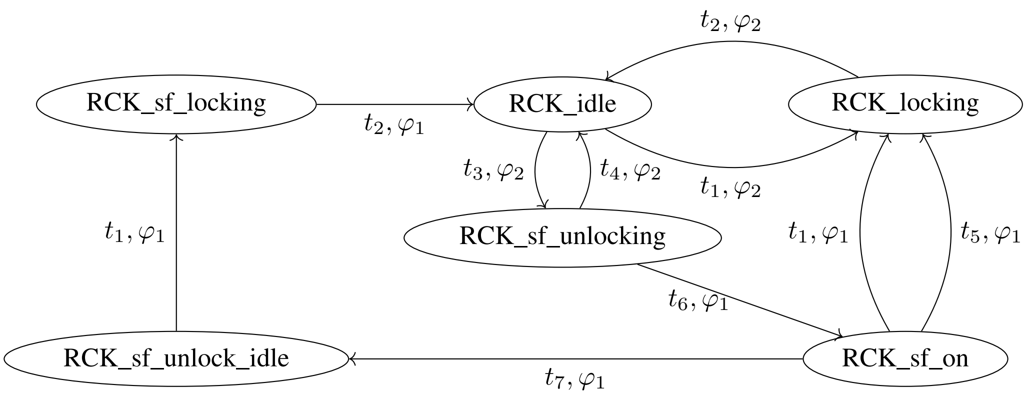
$$b_1 \rightarrow_{\mathbb{L}_d} b_2 \equiv \lfloor \neg_{\mathbb{B}_d} b_1 \vee b_2 \rfloor_d \equiv \lfloor (\neg_{\mathbb{B}} b_1 \wedge d) \vee b_2 \rfloor_d \equiv \lfloor (\neg_{\mathbb{B}} b_1 \wedge d) \vee b_2 \vee \neg d \rfloor \wedge d \equiv \lfloor \neg_{\mathbb{B}} b_1 \vee b_2 \vee \neg d \rfloor \wedge d.$$

For (ii), assume d is downward-closed w.r.t. $\leq$, $d = \lfloor d \rfloor$. By using Lemma 9, we get $\lfloor \neg b_1 \vee b_2 \vee \neg d \rfloor \wedge d \equiv \lfloor \neg b_1 \vee b_2 \vee \neg d \rfloor \wedge \lfloor d \rfloor \equiv \lfloor (\neg b_1 \vee b_2 \vee \neg d) \wedge d \rfloor \equiv \lfloor (\neg b_1 \vee b_2) \wedge d \vee \neg d \wedge d \rfloor \equiv \lfloor (\neg b_1 \vee b_2) \wedge d \rfloor \equiv \lfloor \neg b_1 \vee b_2 \rfloor \wedge \lfloor d \rfloor \equiv \lfloor \neg b_1 \vee b_2 \rfloor \wedge d$. $\square$

8.4. Car Control: A More Complex Example for CTS

So far, we have considered only a small example to highlight the features of conditional transition systems. However, to demonstrate the model's practical usefulness, we will now consider a real-world example, based on a case study by Lity et al. [25]. In this case study, the authors model a product line of cars with numerous optional features, that a car from said product line might have. Note that the original study does not consider the option to add features at a later point (i.e. after sale), but this appears to be a legitimate desired property of the product line.

While the overall study involves various optional features and a very large-scale model, it is a modular model. We will focus on just one aspect detailed in the original work, namely the remote control key of the car. Having a remote control key is an optional feature, named RCK and if present, it allows the user to lock or unlock the car using the remote control key. The second feature is one that only makes sense if RCK is already present: The safety function SF, which ensures that upon unlocking the car, a timer starts running that automatically locks the door after the timer is up to ensure that the car is not accidentally left unlocked. Note that this feature is a software feature that may reasonably be added to a previously sold car. In an effort to remain as close to the model of [25] as possible, we propose the remote control key functionality may be modelled as follows, using a CTS with action precedence. On can think of RCK_idle as the start state where the car is waiting for input and it does not matter whether it is locked or unlocked.



The products under consideration are $\varphi_1 = \{\text{RCK}, \text{SF}\}$ and $\varphi_2 = \{\text{RCK}\}$, where $\varphi_1 \leq \varphi_2$. Additionally, we have the precedence relation $t_6 > t_4$, which enforces that in product φ_1, the safety function can operate. Transitions that are present due to monotonicity are omitted. The action labels are the following:

t_1	RCK Lock-Button	Lock button is pressed
t_2	RCK Lock	Car is locked
t_3	RCK Unlock-Button	Unlock button is pressed
t_4	RCK Unlock	Car is unlocked
t_5	RCK SF Safety Time Elapsed	Timer for safety time runs out
t_6	RCK Safe Unlock	Car is unlocked safely
t_7	RCK Open Door	Door is opened

In a similar way, the various other components of the extensive car control case study can be modelled. However, it is important to make reasonable decisions, whether certain features are sensibly considered as upgrade features. Our modelling method thus leads to a more compact model, where we do not need to duplicate states to model upgrades.

8.5. Implementation and Run-Time Results

We have implemented an algorithm that computes the lattice bisimulation relation based on matrix multiplication (see Theorem 30) 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 run-time 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

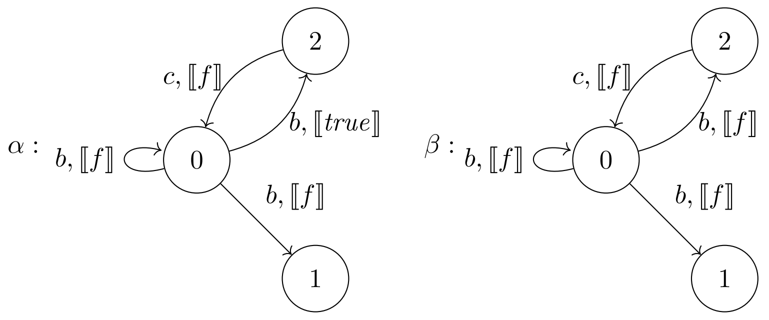


Figure 3: Components for α and β, where f is viewed as a Boolean expression indicating its presence.

via the interface PAT.BDD [26]. The core of this implementation is publicly available in the tool PAWS, which was presented in [21]. Note however, that the tool itself does not offer an interface to run these tests and is meant for analysis of single CTSs, rather than runtime analysis.

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. Hence, let F be a set of features. The example we studied contains, for each feature $f \in F$, one disconnected component in both LaTSs that is depicted in Figure 3: 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 Table 1 depicted on the next page). 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.

Table 1 shows the run-time results (in milliseconds) for the computation of the largest bisimulation for our implementation on the family of CTSs. Admittedly, this example is somewhat artificial, but we were unable to find a realistic case study where the number of features can be arbitrarily increased.

In general we believe that as long as the boolean expression d characterizing the admissible feature combinations is relatively compact or can be specified by a small ROBDD, one can expect that many feature combinations are possible without causing any efficiency problems (since everything is computed on the level of ROBDDs).

9. Related Work

Software product lines

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 [10, 11, 13, 27].

features	time(BDD) (in ms)	time(without BDD) (in ms)	$\frac{\text{time(without BDD)}}{\text{time(BDD)}}$
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	—

Table 1: Runtime results on a family of CTS.

Cordy et al. [13] 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 [13], our focus was on bisimilarity between adaptive SPLs.

In [10, 11, 28], alternative ways to model adaptive SPLs by using the synchronous parallel composition of two separate computational models is presented. Intuitively, one approach 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. [10] 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 monotone 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 [27], 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 [10], their aim is to obtain statistical evidence by performing simulations.

Behavioural equivalences

Behavioural equivalences such as (bi)simulation relations have already been studied in the literature of traditional SPLs. In [4], a simulation relation between any two FTSs (without upgrades) is proposed to combat the state explosion problem by establishing a simulation relation between a system and its refined version. In contrast, the authors in [7] 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.

In [29], Hennessy and Lin describe symbolic bisimulations in the setting of value-passing CCS process terms, where Boolean expressions restrict the interpretations for which one shows bisimilarity. The aim was to develop algorithms to compute symbolic bisimulation between the states of a symbolic transition system (which can be seen as an abstraction of a concrete transition system having an infinite number of states and transitions).

(Bi)simulation relations on lattice Kripke structures were also studied in [30], 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 [30] is quite similar to our Definition 20 (another similar formula occurs in [4]). However, in [30] 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 [4, 30].

Lastly, as we have already shown, there is a strong correspondence of conditional bisimulation in the Boolean case to Fitting's [16] bisimulation relations of unlabelled transition systems (Lemma 19). A notion of bisimulation for Heyting algebras similar to ours is studied in [31] under the name of weak bisimulation, together with a game characterization. However, this paper does not consider duality, the matrix multiplication algorithm and in particular it does not work out the notion of upgrades and the connection to featured transition systems.

Open terms

The name conditional transition systems is not uniquely used as we do in this work, various other concepts that share the name exist. One such example is the work of Rensink [3], in which the term is used to deal with process algebras with open terms. As Rensink observed, in open terms, often variables stand in the way of arguing about possible transitions. One way to deal with this issue is to assert some e.g. a-transition. So if given the CCS term $x + y$, where x and y are variables, one may assume that x enables an a-transition to x' and then be able to derive

$x + y \xrightarrow{a} x'$ from that. Such assertions are called conditional transitions. Related to this work, Baldan et al. [32] proposed bisimilarity on open systems that are specified by terms with a hole or place-holder. The concept of modelling behaviour depending on a context can be even traced back to Larsen's PhD thesis [33], where a process (modelled as an LTS) is embedded into an environment (modelled as a transducer) consuming the transitions of the process. In contrast to CTS as defined in this paper, the goal in this line of work is not to analyse a class of systems with guarding products, but to reason about specifications with holes, that may be implemented by a number of different behaviours.

Dynamic modal logic

Reactive automata as investigated by Crochemore and Gabbay [34] allow for a finer granularity of activation and deactivation of transition when compared to CTSs. In reactive automata, leaving certain states or taking certain transitions can have a side effect of (de-)activating other transitions. Consequently, the user has very limited control over (de-)activation of transitions and in fact such a system can be flattened to a labelled transition system. On the other hand, in terms of modelling, it allows for very fine control of available actions and can lead to significantly smaller system models than a classical labelled transition system. In contrast to CTSs, (de-)activation comes with no monotonicity constraint.

In a similar spirit to reactive automata, there are also interesting studies [35–37] carried out in modal logic – initiated by van Benthem's swap logic – to model dynamic changes in transition relations (or accessibility relations as they are known in the modal logic literature) of the underlying Kripke structure. The expressivity of such Kripke structures extended with *model update functions* (cf. [36]) and CTSs is incomparable. On the one hand, thanks to the model update functions, one may change the accessibility relation without any restriction of the underlying Kripke structure with every state transition. On the other hand, unlike CTSs, dynamic changes in the structure of the underlying Kripke structures are not guarded by any condition. Thus, these extended models are not suitable to model an adaptive SPL.

10. Conclusion 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. 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 computes 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.

One possible extension of CTSs as a specification language, which is still lacking, is how to incorporate downgrades. In this regard, one could 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.

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. To open up CTS for model checking, we have also started work on modal logics for CTS, which has led to first results in [20].

On the more theoretical side of things, we have worked out the coalgebraic concepts for CTSs [38], where it was non-trivial to find an functor describing the branching type and to integrate the notion of upgrades. We compared the matrix multiplication algorithm to the final chain algorithm presented in [2], when applied to CTSs. It was shown that the final chain algorithm, when instantiated to CTSs, takes precisely as many steps to determine the conditional bisimilarity as the algorithm presented in this paper, though the representation of each individual step is significantly larger. In terms of the duality between CTS and LaTS, it is interesting to note that the duality observed in the concrete case extends to an isomorphism of categories in the coalgebraic setting.

The concept of having environmental variables to allow for different versions of a transition system can also be extended to graph transformation systems, or, more generally, reactive systems. We are currently working on conditional transition systems in the context of reactive systems, extending previous work on conditional reactive systems [39].

Acknowledgements: We thank Filippo Bonchi and Mathias Hülsbusch for interesting discussions on earlier drafts. This research is partially supported by DFG project BEMEGA and ERC Starting Grant ProFoundNet (grant agreement 679127).

References

[1] H. Beohar, B. König, S. Küpper, A. Silva, Conditional transition systems with upgrades, in: Proc. of TASE '17 (Theoretical Aspects of Software Engineering), IEEE Xplore, 2017.

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

[3] A. Rensink, Bisimilarity of open terms, Information and Computation 156 (1) (2000) 345 – 385.

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

[5] M. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints, JLAMP 85 (2) (2016) 287 – 315.

[6] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, 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) (2013) 1069–1089.

[7] J. M. Atlee, U. Fahrenberg, A. Legay, Measuring behaviour interactions between product-line features, in: Proc. of Formalise '15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 20–25.

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

[9] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, Symbolic model checking of software product lines, in: Proc. of ICSE '11, ACM, 2011.

[10] C. Dubslaff, S. Klüppelholz, C. Baier, Probabilistic model checking for energy analysis in software product lines, in: Proc. of MODULARITY '14, ACM, 2014, pp. 169–180.

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

[12] A. Gruler, M. Leucker, K. Scheidemann, Modeling and model checking software product lines, in: Proc. of FMOODS'08, Springer, 2008, pp. 113–131, LNCS 5051.

[13] M. Cordy, A. Classen, P. Heymans, A. Legay, P.-Y. Schobbens, Model checking adaptive software with featured transition systems, in: Assurances for Self-Adaptive Systems, Springer, 2013, pp. 1–29.

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

[15] M. Droste, W. Kuich, H. Vogler (Eds.), Handbook of Weighted Automata, Springer, 2009.

[16] M. Fitting, Bisimulations and boolean vectors, in: Advances in Modal Logic, Vol. 4, World Scientific Publishing, 2002, pp. 97–126.

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

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

[19] C. Stirling, Bisimulation, model checking and other games, notes for Mathfit instructional meeting on games and computation, Edinburgh (June 1997).

[20] K. Poltermann, Eine modallogik für conditional transition systems, bachelor Thesis (October 2017).

[21] B. König, C. Mika, S. Küpper, Paws: A tool for the analysis of weighted systems, in: Proc. of QAPL '17 (International Workshop on Quantitative Aspects of Programming Languages and Systems), Vol. 250 of EPTCS, 2017.

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

[23] A. Metzger, K. Pohl, Software product line engineering and variability management: Achievements and challenges, in: Proc. of FOSE '14, ACM, New York, NY, USA, 2014, pp. 70–84.

[24] H. R. Andersen, An introduction to binary decision diagrams, course notes (1997).

[25] S. Lity, R. Lachmann, M. Lochau, I. Schaefer, Delta-oriented software product line test models – the body comfort system case study, Report TU Darmstadt (2013).
URL `http://tubiblio.ulb.tu-darmstadt.de/73869/`

[26] T. K. Nguyen, J. Sun, Y. Liu, J. S. Dong, Y. Liu, Improved BDD-based discrete analysis of timed systems, in: Proc. of FM '12, Vol. 7436 of LNCS, Springer, 2012, pp. 326–340.

[27] M. H. ter Beek, A. Legay, A. L. Lafuente, A. Vandin, Statistical analysis of probabilistic models of software product lines with quantitative constraints, in: Proc. of SPLC '15, ACM, 2015, pp. 11–15.

[28] M. Lochau, J. Bürdek, S. Hölzle, A. Schürr, Specification and automated validation of staged reconfiguration processes for dynamic software product lines, Software & Sys. Modeling 16 (1) (2017) 125–152.

[29] M. Hennessy, H. Lin, Symbolic bisimulations, Theoretical Computer Science 138 (2) (1995) 353–389.

[30] O. Kupferman, Y. Lustig, Latticed simulation relations and games, Int. Journal of Found. of Computer Science 21 (02) (2010) 167–189.

[31] P. Eleftheriou, C. Koutras, C. Nomikos, Notions of bisimulation for Heyting-valued modal languages, J. Logic Comput. 22 (2) (2012) 213–235.

[32] P. Baldan, A. Bracciali, R. Bruni, Bisimulation by unification, in: Proc. of AMAST '02, Springer, 2002, pp. 254–270, LNCS 2422.

[33] K. G. Larsen, Context-dependent bisimulation between processes, Ph.D. thesis, University of Edinburgh (1986).

[34] M. Crochemore, D. M. Gabbay, Reactive automata, Information and Computation 209 (4) (2011) 692 – 704.

[35] D. M. Gabbay, Reactive Kripke Semantics, 1st Edition, Cognitive Technologies, Springer-Verlag Berlin Heidelberg, 2013.

[36] C. Areces, R. Fervari, G. Hoffmann, Relation-changing modal operators, Logic Journal of the IGPL 23 (4) (2015) 601–627.

[37] C. Areces, R. Fervari, G. Hoffmann, M. Martel, Satisfiability for relation-changing logics, Journal of Logic and Computation 28 (7) (2018) 1443–1470.

[38] H. Beohar, B. König, S. Küpper, A. Silva, T. Wißmann, A coalgebraic treatment of conditional transition systems with upgrades, Logical Methods in Computer Science 14 (1), Special Festschrift Issue in Honor of Jiří Adámek.

[39] M. Hülsbusch, B. König, Deriving bisimulation congruences for conditional reactive systems, in: Proc. of FOSSACS '12, Springer, 2012, pp. 361–375, LNCS/ARCoSS 7213.