Model-Checking Process Equivalences

Martin Lange Etienne Lozes Manuel Vargas Guzmán
School of Electrical Engineering and Computer Science
University of Kassel, Germany
martin.lange@uni-kassel.de lozes@lsv.ens-cachan.fr manuel.vargas@uni-kassel.de

Process equivalences are formal methods that relate programs and system which, informally, behave in the same way. Since there is no unique notion of what it means for two dynamic systems to display the same behaviour there are a multitude of formal process equivalences, ranging from bisimulation to trace equivalence, categorised in the linear-time branching-time spectrum.

We present a logical framework based on an expressive modal fixpoint logic which is capable of defining many process equivalence relations: for each such equivalence there is a fixed formula which is satisfied by a pair of processes if and only if they are equivalent with respect to this relation. We explain how to do model checking, even symbolically, for a significant fragment of this logic that captures many process equivalences. This allows model checking technology to be used for process equivalence checking. We show how partial evaluation can be used to obtain decision procedures for process equivalences from the generic model checking scheme.

1 Introduction

In concurrency theory, a process equivalence is an equivalence relation between processes — represented as states of a labeled transition system (LTS) — that aims at capturing the informal notion of "having the same behaviour". A theory of behavioural equivalence obviously has applications in formal systems design because it explains which programs or modules can be replaced by others without changing the system's behaviour.

There is no single mathematical notion of process equivalence as an equivalence relation on LTS. Instead a multitude of different relations has been studied with respect to their pragmatics, axiomatisability, computational complexity, etc. These form a hierarchy with respect to containment, known as the *linear-time branching-time spectrum* [6]. We refer to the literature for a comprehensive overview over all these equivalence relations at this point.

There are a few techniques which have proved to yield decision procedures for certain process equivalences, for example *approximations* [9], *characteristic formulas* [1, 5] and *characteristic games* [14, 13]. Often, for each equivalence notion, the same questions are being considered independently of each other, like "can the algorithm be made to work with symbolic (BDD-based) representations of LTS?", and the answer may depend on the technique being used to obtain the algorithm.

In this paper we introduce a further and generic, thus powerful technique, using the notion of *defining formulas*. We present a modal fixpoint logic which is expressive enough to define these equivalences in the sense that, for an equivalence relation R, there is a fixed formula Φ_R which evaluates to true in a pair of processes if and only if they are related by R. We also give a model checking algorithm for this logic. This can then be instantiated with such formulas Φ_R in order to obtain an equivalence checking algorithm for R. Furthermore, the model checking algorithm can easily deal with symbolic representations. Thus, this yields BDD-based equivalence checking algorithms for all the process equivalences mentioned in this paper. Moreover, with this generic framework, the task of *designing* an equivalence checking algorithm

M. Faella, A. Murano (Eds.): Games, Automata, Logics and Formal Verification (GandALF 2012)
EPTCS 96, 2012, pp. 43–56, doi:10.4204/EPTCS.96.4

for any new equivalence notion boils down to simply *defining* this relation in the modal fixpoint logic presented here.

This is related to work on *characteristic formulas*, yet it is different. There, in order to check two processes P and Q for, say, bisimilarity, one builds the characteristic formula $\Phi^P_\sim$ describing all processes that are bisimilar to P and checks whether or not $Q \models \Phi^P_\sim$ holds. Here, we take a fixed formula $\Phi_\sim$ and check whether or not $(P,Q) \models \Phi_\sim$ holds. Note that the former cannot be made to work with a symbolic representation of P whereas the latter can. In general, using defining instead of characteristic formulas has the advantage of lifting more model checking technology to process equivalence checking.

The use of fixed formulas expressing process equivalences is being made possible by the design of a new modal fixpoint logic. It is obtained as the merger between two extensions of the modal μ-calculus, namely the *higher-dimensional μ-calculus* $\mathscr{L}^\omega_\mu$ [12] and the *higher-order μ-calculus* HFL [16]. The former allows formulas to make assertions about tuples of states rather than states alone. This is clearly useful in this setup given that process equivalences are binary relations. Not surprisingly, it is known for instance that there is a formula in $\mathscr{L}^2_\mu$ — the fragment speaking about tuples of length 2 — that expresses bisimilarity. On the other hand, HFL's higher-order features allow the logic to express properties that are more difficult than being polynomial-time decidable. It is known for instance that it can make assertions of the kind "for every finite word w there is a path labeled with w" which is very useful for describing variants of trace equivalence.

The rest of the paper is organised as follows. Sect. 2 recalls the linear-time branching-time hierarchy. For the sake of completeness, the exact definitions of these relations are presented in an appendix. Sect. 3 defines the aforementioned modal fixpoint logic. Sect. 4 realises the reduction from process equivalence checking to model checking fixed formulas by simply spelling out the definition of those equivalence notions in this modal fixpoint logic. Sect. 5 shows how to do model checking for the fragment of this logic which is most significant to process equivalence checking, and how the naïve model checking algorithm can be optimised using need-driven function evaluation and partial evaluation. Sect. 6 concludes with ideas on further work in this direction.

2 Process Equivalences

In this section we present the hierarchy of the linear-time branching-time spectrum, as it can be seen from Fig. 1, the greatest equivalence is *finite trace equivalence*, and the finest one is *bisimulation*. First we introduce some preliminaries and notation. We use letters $a,b,\ldots$ to denote actions, and letter t to denote a trace. Letters $P,Q,\ldots$ denote processes.

A *labeled transition system* (LTS) over a set of actions[1] $\mathsf{Act} = \{a,b,...\}$ is a triple $(\mathsf{Pr},\mathsf{Act},\rightarrow)$, where Pr is a set of states representing processes, Act is the set of actions, and $\rightarrow \subseteq \mathsf{Pr}\times\mathsf{Act}\times\mathsf{Pr}$ is a transition relation. We write $P \xrightarrow{a} Q$ for $(P,a,Q) \in \rightarrow$. $I(P) := \{a \in \mathsf{Act} \mid \exists Q.P \xrightarrow{a} Q\}$ denotes the set of *initial actions* of a process P.

A *finite trace* $t \in \mathsf{Act}^*$ of P_0, is a finite sequence of actions $a_1...a_n$ s.t. there are $P_0...P_n$ with $P_{i-1} \xrightarrow{a_i} P_n$ for all $i = 1,...,n$. We write $P \xrightarrow{t} Q$ if there is a trace t of P that ends in Q.

Since the main purpose of this paper is not to focus deeply on the semantics of process equivalences, we do not address the definitions in this section. For further details, the reader can find the exact definitions of all process equivalences in Appendix A.

[1] For simplicity, we do not consider state labels.

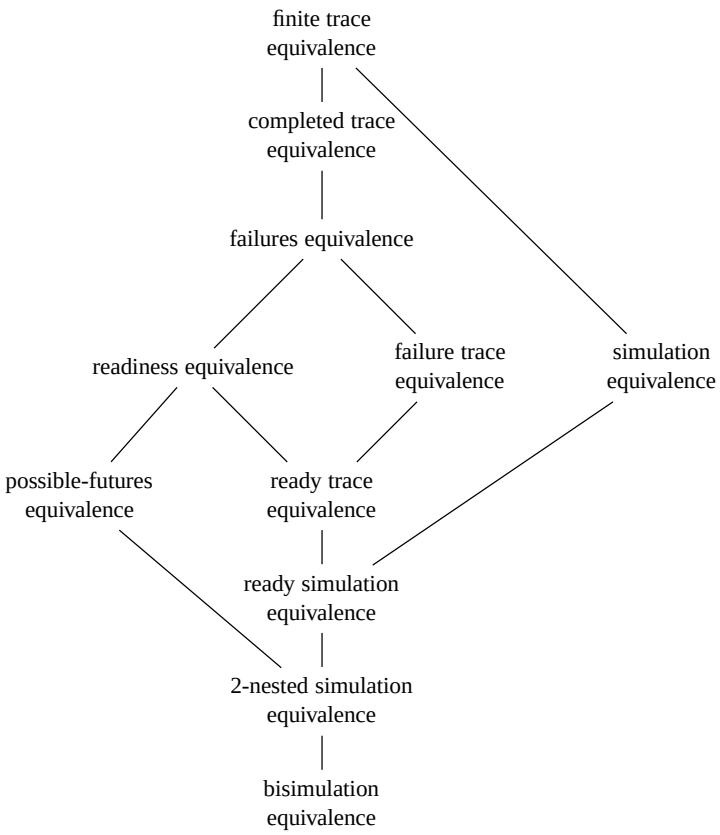


Figure 1: The linear-time branching-time hierarchy.

3 A Higher-Order Higher-Dimensional μ-Calculus

3.1 Combining Higher-Order and Higher-Dimensionality

In this section, we introduce a logical formalism, called $\mu\mathbf{HL}^{\omega}_{\omega}$ that extends the standard modal μ-calculus. It can be seen as the combination of two extensions of the μ-calculus that were previously defined: the higher-order fixpoint logic HFL [16], and the higher-dimensional modal mu-calculus $\mathscr{L}^{\omega}_{\mu}$ [12]. First we build some intuition about the use of higher-order and higher-dimensional features in modal logics.

In HFL, formulas may denote not only sets of processes, but also *predicate transformers*, *i.e.* functions from sets of processes to sets of processes, and more generally any higher-order functions of some functional type built on top of the basic type **Pr** of set of processes. For instance, the formula

$$\lambda x : \mathbf{Pr}.\ \langle a\rangle x \wedge [b]\bot$$

denotes the function that takes a predicate Φ of type **Pr**, *i.e.* a set of processes, and returns the predicate $\langle a\rangle\Phi \wedge [b]\bot$, *i.e* the set of processes P for which $P \overset{b}{\not\rightarrow}$ and $P \xrightarrow{a} P'$ for some $P' \models \Phi$. Similarly, the formula

$$\lambda f : \mathbf{Pr} \to \mathbf{Pr}.\ \lambda x : \mathbf{Pr}.\ f\ (f\ x)$$

denotes the function that maps any predicate transformer f to the predicate transformer f^2.

Like in the standard μ-calculus, to every monotone function of type $\mathbf{Pr} \to \mathbf{Pr}$ denoted by a formula $\lambda x : \mathbf{Pr}.\ \Phi$, HFL associates a least fixed point $\mu x : \mathbf{Pr}.\ \Phi$. In HFL, this construction generalises well to any monotone function of type $\tau \to \tau$, thanks to a construction based on the pointwise ordering of functions we recall below. For instance, the formula $\mu f : \mathbf{Pr} \to \mathbf{Pr}.\ f$ denotes the constant function $\lambda x.\ \bot$, since it is the smallest predicate transformer, according to the pointwise ordering, that is fixed by the identity function. A bit more elaborated, the formula

$$\mu f : \mathbf{Pr} \to \mathbf{Pr}.\ \lambda x : \mathbf{Pr}.\lambda y : \mathbf{Pr}.\ \ (x \wedge y) \ \vee \ \ f\ \langle a\rangle x\ \langle b\rangle y$$

can be unfolded as

$$f\,x\,y \quad = \quad (x \wedge y) \ \vee \ f\ \langle a\rangle x\ \langle b\rangle y \quad = \quad (x \wedge y) \ \vee \ (\langle a\rangle x \wedge \langle b\rangle y) \ \vee \ f\ \langle aa\rangle x\ \langle bb\rangle y \quad = \quad \ldots$$

and thus denotes the function $\lambda x,y.\ \bigvee_{n\geq 0} \langle a\rangle^n x \wedge \langle b\rangle^n y$.

The higher-dimensional μ-calculus extends the μ-calculus in a different way. In $\mathscr{L}_\mu^\omega$, logical formulas do not denote sets of processes, but sets of tuples of processes. The i-th component of a tuple can be changed by the i-th modality $\langle a\rangle_i$. For instance, the 2-dimensional formula $\langle a\rangle_1 \top \wedge \langle b\rangle_2 \top$ denotes the set of pairs (P,Q) such that $P \xrightarrow{a} P'$ and $Q \xrightarrow{b} Q'$ for some P', Q'. The modality $\langle a\rangle_i$ only modifies the i-th component of the tuple, and leaves all other components unchanged, which validates some rules like

$$\begin{array}{lrcll}
\text{(commutation)} & \langle a\rangle_1\langle b\rangle_2\,\Phi & \Leftrightarrow & \langle b\rangle_2\langle a\rangle_1\,\Phi & \\
\text{(scope extrusion)} & \langle a\rangle_d\,(\Phi \wedge \Psi) & \Leftrightarrow & \langle a\rangle_d\,\Phi \ \wedge \ \Psi & \big(\mathsf{dim}(\Psi) < d\big).
\end{array}$$

We associate a type $\mathbf{Pr}_d$ to the formulas of the d-dimensional μ-calculus. Note that there is a significant difference between *e.g.* $\mathbf{Pr}_2$ and the product type $\mathbf{Pr} \times \mathbf{Pr}$: the former is the type of binary predicates over processes, whereas the latter is the type of pairs of unary predicates. There is indeed no obvious way of representing $\mathscr{L}_\mu^\omega$ in HFL, although HFL may encode some of product types using standard techniques.

3.2 Syntax and Semantics

Let Act be as above. Fix $d \in \mathbb{N}$. We assume an infinite set $\mathsf{Var} = \{x,y,z,\ldots\}$ of variables. A formula is a Φ that can be derived from by

$$\begin{array}{rcll}
\Phi,\Psi & ::= & \top \mid \langle a\rangle_i\Phi \mid \neg\Phi \mid \Phi\wedge\Psi \mid x \mid \lambda x^\nu : \tau.\ \Phi \mid \mu x : \tau.\Phi \mid \Phi\ \Psi & \text{(formulas)} \\
\nu & ::= & + \mid - \mid \pm & \text{(variances)} \\
\tau,\sigma & ::= & \mathbf{Pr}_d \mid \tau^\nu \to \sigma & \text{(types)}
\end{array}$$

where $1 \leq i \leq d$, $a \in \mathsf{Act}$ and $x \in \mathsf{Var}$.

The typing arrow $\to$ is — as usual — right-associative. Thus, every type is of the form $\tau = \tau_1^{\nu_1} \to \ldots \to \tau_m^{\nu_m} \to \mathbf{Pr}_d$ for some $m \geq 0$. For such normalised types we can define their *order* simply as $ord(\tau) := \max\{1 + ord(\tau_i) : i = 1,\ldots,m\}$ with the convention of $\max\emptyset = 0$.

Formulas are ruled by the type system depicted on Fig. 2. Intuitively, the aim of the type system is to prevent applications of non-functions to formulas, as well as fixpoint definitions of non-monotone functions, like $\mu x.\lambda y.\neg x\ y$. In order to exclude the latter, *variances* are introduced for each function parameter.

$$\Gamma \vdash \top : \mathbf{Pr}_d \qquad \frac{\Gamma \vdash \Phi : \mathbf{Pr}_d \qquad i \leq d}{\Gamma \vdash \langle a \rangle_i \Phi : \mathbf{Pr}_d} \qquad \frac{\neg(\Gamma) \vdash \Phi : \mathbf{Pr}_d}{\Gamma \vdash \neg\Phi : \mathbf{Pr}_d} \qquad \frac{\Gamma \vdash \Phi : \mathbf{Pr}_d \qquad \Gamma \vdash \Psi : \mathbf{Pr}_d}{\Gamma \vdash \Phi \wedge \Psi : \mathbf{Pr}_d}$$

$$\frac{v \in \{+, \pm\}}{\Gamma\,,\, x^v : \tau \vdash x : \tau} \qquad \frac{\Gamma, x^v : \sigma \vdash \Phi : \tau}{\Gamma \vdash \lambda x^v : \sigma.\ \Phi : \sigma^v \to \tau} \qquad \frac{\Gamma, x^+ : \tau \vdash \Phi : \tau}{\Gamma \vdash \mu x(y_1, \ldots, y_m) : \tau.\ \Phi : \tau}$$

$$\frac{\Gamma \vdash \Phi : \sigma^+ \to \tau \qquad \Gamma \vdash \Psi : \sigma}{\Gamma \vdash \Phi\ \Psi : \tau} \qquad \frac{\Gamma \vdash \Phi : \sigma^- \to \tau \qquad \neg(\Gamma) \vdash \Psi : \sigma}{\Gamma \vdash \Phi\ \Psi : \tau}$$

$$\frac{\Gamma \vdash \Phi : \sigma^\pm \to \tau \qquad \Gamma \vdash \Psi : \sigma \qquad \neg(\Gamma) \vdash \Psi : \sigma}{\Gamma \vdash \Phi\ \Psi : \tau}$$

Figure 2: The type system of $\mu\mathbf{HL}^\omega_\omega$.

For $d \geq 1$ and $o \geq 0$, let $\mu\mathbf{HL}^o_d$ consist of all closed formulas Φ such that the statement $\emptyset \vdash \Phi : \mathbf{Pr}_d$ is typable, and each type annotation in Φ has order at most o. In general, a statement of the form $\Gamma \vdash \Phi : \tau$ asserts that the formula Φ has type τ under the assumptions Γ, which is a list of the form $x_1^{v_1} : \tau_1, \ldots, x_m^{v_m} : \tau_m$. For such a list of assumptions, $\neg\Gamma$ is obtained from Γ by swapping the variance of each variable: $+$ becomes $-$ and vice-versa, and $\pm$ remain the same. Thus, $\mu\mathbf{HL}^o_d$ consists of all well-typed and closed formulas of type that should denote a set of i-tuples in an LTS and use at most higher-order features of order o. Let

$$\mu\mathbf{HL}^o_\omega \;:=\; \bigcup_{d \geq 1} \mu\mathbf{HL}^o_d \quad , \qquad \mu\mathbf{HL}^\omega_d \;:=\; \bigcup_{o \geq 0} \mu\mathbf{HL}^o_d \quad , \qquad \mu\mathbf{HL}^\omega_\omega \;:=\; \bigcup_{o \geq 0} \bigcup_{d \geq 1} \mu\mathbf{HL}^o_d$$

Before we can explain the semantics of a formula we need to give the types a semantics too. Let $\mathscr{T}_i = (\mathsf{Pr}_i, \mathsf{Act}, \to_i)$ for $i = 1, \ldots, d$ be LTS. We take them to be fixed and simply write $[\![\tau]\!]$ instead of $[\![\tau]\!]^{\mathscr{T}_1, \ldots, \mathscr{T}_d}$. The semantics of a type is inductively defined as follows.

- $[\![\mathbf{Pr}_d]\!]$ is the set of all sets of d-ary predicates of processes, ordered by inclusion, i.e. $[\![\mathbf{Pr}_d]\!] = (\mathscr{P}(\mathsf{Pr}_1 \times \ldots \times \mathsf{Pr}_d), \leq_{\mathbf{Pr}_d})$, with $\mathscr{S} \leq_{\mathbf{Pr}_d} \mathscr{S}'$ if $\mathscr{S} \subseteq \mathscr{S}'$.

- $[\![\tau^+ \to \sigma]\!]$ is the set of monotone functions from $[\![\tau]\!]$ to $[\![\sigma]\!]$, ordered by pointwise ordering, i.e. $[\![\tau^+ \to \sigma]\!] = \{ f \in [\![\sigma]\!]^{[\![\tau]\!]} \ : \ \forall x, y.\ x \leq_\tau y \Rightarrow f(x) \leq_\sigma f(y) \}$ and $f \leq_{\tau^+ \to \sigma} g$ if for all $x \in [\![\tau]\!]$, $f(x) \leq_\sigma g(x)$.

- similarly $[\![\tau^- \to \sigma]\!]$ is the set of co-monotone functions, and $[\![\tau^\pm \to \sigma]\!]$ is the set of all functions from $[\![\tau]\!]$ to $[\![\sigma]\!]$, ordered by pointwise ordering.

All these domains are complete lattice. As a consequence, any function $f \in [\![\tau^+ \to \tau]\!]$ has a least fixpoint according to the Knaster-Tarski Theorem [10, 15]; we write $\mathsf{LFP}_\tau\ f$ to denote it.

The semantics of a formula Φ of type τ with respect to an environment Γ, the underlying LTS $\mathscr{T}_1, \ldots, \mathscr{T}_d$ and an interpretation η of its free variable is an element of $[\![\tau]\!]$, defined as follows. Let

$\mathsf{Pr}^d := \mathsf{Pr}_1 \times \ldots \times \mathsf{Pr}_d$.

$$\begin{array}{rcl}
[\![\Gamma \vdash \top : \mathbf{Pr}_d]\!]_\eta & = & \mathsf{Pr}^d \\
[\![\Gamma \vdash \langle a\rangle_i \Phi : \mathbf{Pr}_d]\!]_\eta & = & \{(P_1,\ldots,P_d) \in \mathsf{Pr}^d \ : \ \exists P_i' \in \mathsf{Pr}_i.\ P_i \xrightarrow{a} P_i' \text{ and} \\
 & & \qquad (P_1,\ldots,P_i',\ldots P_d) \in [\![\Gamma \vdash \Phi : \mathbf{Pr}_d]\!]_\eta\} \\
[\![\Gamma \vdash \neg\Phi : \mathbf{Pr}_d]\!]_\eta & = & \mathsf{Pr}^d \setminus [\![\neg(\Gamma) \vdash \Phi : \mathbf{Pr}_d]\!]_\eta \\
[\![\Gamma \vdash \Phi \wedge \Psi : \mathbf{Pr}_d]\!]_\eta & = & [\![\Gamma \vdash \Phi : \mathbf{Pr}_d]\!]_\eta \ \cap \ [\![\Gamma \vdash \Psi : \mathbf{Pr}_d]\!]_\eta \\
[\![\Gamma, x^\nu \vdash x^\nu]\!]_\eta & = & \eta(x) \\
[\![\Gamma \vdash \lambda x^\nu : \sigma.\ \Phi \ : \ \tau]\!]_\eta & = & f \text{ such that for all } e \in [\![\sigma]\!], \quad f(e) \ = \ [\![\Gamma, x^\nu : \sigma \vdash \Phi : \tau]\!]_{\eta[x \mapsto e]} \\
[\![\Gamma \vdash \mu x : \tau.\ \Phi \ : \ \tau]\!]_\eta & = & \mathsf{LFP}_\tau \ [\![\Gamma \vdash \lambda x^+ : \tau.\ \Phi \ : \ \tau]\!]_\eta \\
[\![\Gamma \vdash \Phi\ \Psi : \tau]\!]_\eta & = & f(e), \text{ where } f = [\![\Gamma \vdash \Phi : \sigma^\nu \to \tau]\!]_\eta \text{ and } e = [\![\Gamma' \vdash \Psi : \sigma]\!]_\eta
\end{array}$$

If $\vdash \Phi : \tau$ and $m \in [\![\tau]\!]$, we write $m \models \Phi$ to denote that $m \in [\![\vdash \Phi : \tau]\!]$.

We assume standard notations for derived boolean and modal operators, and write $\Phi \vee \Psi$ for $\neg(\neg\Phi \wedge \neg\Psi)$, or $[a]_i\Phi$ for $\neg\langle a\rangle_i\neg\Phi$, or $\Phi \Leftrightarrow \Psi$ for $(\Phi \wedge \neg\Psi) \vee (\neg\Phi \wedge \Psi)$, etc. If $\Gamma \vdash \Phi : \tau_1^+ \to \tau_2$ and $\Gamma \vdash \Psi : \tau_2^+ \to \tau_3$ are two monotone functions, we write $\Psi \circ \Phi$ as a shorthand for the monotone function $\lambda x^+ : \tau_1.\ \Psi\ (\Phi\ x)$. We will also write $\mu x(y_1,\ldots,y_m) : \sigma_1^{\nu_1} \to \ldots \to \sigma_m^{\nu_m} \to \tau.\Phi$ instead of $\mu x : \tau.\lambda y_1 : \sigma_1^{\nu_1} \ldots \lambda y_m : \sigma_m^{\nu_m}.\Phi$. Finally, $\Phi[\Psi/x]$ is obtained from Φ by replacing every free occurrence of the variable x with the formula Ψ.

4 Process Equivalences as Formulas

In this section, we show how all process equivalences of the linear-time branching-time hierarchy can be characterised by $\mu\mathbf{HL}_\omega^\omega$ in a certain sense. To improve readability, we will often keep the type system implicit, and use different variable symbols in order to suggest the type. For instance, we write X, Y to range over sets of tuples of processes, F, G to range over first-order functions of type $\mathbf{Pr}_2^{\nu_1} \to \cdots \to \mathbf{Pr}_2^{\nu_m} \to \mathbf{Pr}_2$, whereas $\mathscr{F}$ ranges over second-order functions. We write $\Phi[1 \leftrightarrow 2]$ for the formula Φ in which $\langle a\rangle_1$ and $\langle a\rangle_2$ are swapped for any $a \in \mathsf{Act}$, equally for $[a]_1$ and $[a]_2$. For any $t = a_1 \ldots a_n \in \mathsf{Act}^*$ we write $\langle t\rangle_i\Phi$ to abbreviate $\langle a_1\rangle_i \ldots \langle a_n\rangle_i\Phi$, and similarly for $[t]_i$.

We say that an equivalence relation $\mathscr{R}$ over processes is *characterised* by a closed formula Φ of type $\mathbf{Pr}_2$ if for all processes P, Q

$$P\ \mathscr{R}\ Q \qquad \Leftrightarrow \qquad (P,Q) \models \Phi.$$

We will say that a formula Φ *tests* for $\mathscr{R}$ if $\neg\Phi \wedge \neg\Phi[1 \leftrightarrow 2]$ characterises $\mathscr{R}$. Intuitively, Φ tests for $P\ \mathscr{R}\ Q$ if it is true when P presents a behavior that Q cannot reproduce. For readability, we only present formulas that test process equivalences, but it is straightforward to derive formulas that characterise process equivalence. We later write $\Phi_{\mathscr{R}}$ for a formula that tests $\mathscr{R}$.

Let us first consider trace equivalence. If we were to consider a logic with infinite disjunctions, a formula testing finite trace equivalence would be $\bigvee_{t \in \mathsf{Act}^*} \langle t\rangle_1 \top \wedge [t]_2 \bot$. Encoding such an infinite disjunction is not easy in general, and it is indeed impossible in the ordinary μ-calculus. But the $\mu\mathbf{HL}_\omega^\omega$ formula

$$\Phi_{\mathsf{t}} \qquad \triangleq \qquad \big(\mu F(X,Y).\quad (X \wedge Y) \quad \vee \bigvee_{a \in \mathsf{Act}} F \quad \langle a\rangle_1 X \quad [a]_2 Y\big) \qquad \top \qquad \bot$$

equivalence	Mod	Pred
trace	$\{\langle a\rangle_1 X : a \in \mathsf{Act}\}$	$\{\top\}$
completed trace	$\{\langle a\rangle_1 X : a \in \mathsf{Act}\}$	$\{\bigwedge_{a\in\mathsf{Act}}[a]_1\bot\}$
failure	$\{\langle a\rangle_1 X : a \in \mathsf{Act}\}$	$\{\mathsf{fail}(A) : A \subseteq \mathsf{Act}\}$
failure trace	$\{\langle a\rangle_1 X : a \in \mathsf{Act}\} \cup \{X \wedge \mathsf{fail}(A) : A \subseteq \mathsf{Act}\}$	$\{\top\}$
readiness	$\{\langle a\rangle_1 X : a \in \mathsf{Act}\}$	$\{\mathsf{ready}(A) : A \subseteq \mathsf{Act}\}$
ready trace	$\{\langle a\rangle_1 X : a \in \mathsf{Act}\} \cup \{X \wedge \mathsf{ready}(A) : A \subseteq \mathsf{Act}\}$	$\{\top\}$

equivalence	Mod	Test
simulation	$\{\langle a\rangle_1[a]_2 X : a \in \mathsf{Act}\}$	$\bot$
completed simulation	$\{\langle a\rangle_1[a]_2 X : a \in \mathsf{Act}\}$	$\mathsf{deadlock}_1 \not\Leftrightarrow \mathsf{deadlock}_2$
ready simulation	$\{\langle a\rangle_1[a]_2 X : a \in \mathsf{Act}\}$	$\bigvee_{A\subseteq\mathsf{Act}} \mathsf{ready}_1(A) \not\Leftrightarrow \mathsf{ready}_2(A)$
2-nested simulation	$\{\langle a\rangle_1[a]_2 X : a \in \mathsf{Act}\}$	$\mathbf{\Phi}_\mathsf{s}[1 \leftrightarrow 2]$
bisimulation	$\{\langle a\rangle_1[a]_2 X, \langle a\rangle_2[a]_1 X : a \in \mathsf{Act}\}$	$\bot$

Figure 3: Instantiations of the parameters for the template formulas.

is equivalent to the one with the infinite disjunction, and thus tests trace equivalence.

Let us consider now all other equivalences of the lower part of the hierarchy. As all these equivalences are variations around finite trace equivalence, it can be expected that the formulas testing them are very similar. We introduce the template formula $\mathsf{TemplateTrace}(\mathsf{Mod},\mathsf{Pred}) \triangleq$

$$\bigvee_{\Phi\in\mathsf{Pred}} \Big(\mu F(X,Y).\quad (X \wedge Y) \quad \vee \bigvee_{\Psi\in\mathsf{Mod}} F \quad \Psi \quad \neg\Psi[1 \leftrightarrow 2][\neg Y/X]\Big) \qquad \Phi \qquad \neg\Phi[1 \leftrightarrow 2]$$

for some finite sets Pred and Mod of 0-order formulas. For instance, the above formula testing trace equivalence is obtained for $\mathsf{Pred} = \{\top\}$ and $\mathsf{Mod} = \{\langle a\rangle_1 X : a \in \mathsf{Act}\}$. Other instantiations of these two parameters provide all equivalences above simulations, c.f. the upper table in Fig. 3. Let $\mathsf{fail}(\mathsf{A}) \triangleq \bigwedge_{a\in A}[a]_1\bot$ and $\mathsf{ready}(A) \triangleq \bigwedge_{a\in A}\langle a\rangle_1\bot \wedge \bigwedge_{a\notin A}[a]_1\bot$.

Formulas testing the relations below simulation equivalence can also be derived from a common, but simpler template. In these case, no higher-order features are needed. Let $\mathsf{TemplateSim}(\mathsf{Mod},\mathsf{Test}) \triangleq$

$$\mu X.\ \mathsf{Test} \ \vee \bigvee_{\Psi\in\mathsf{Mod}} \Psi$$

where Test stands for an $\mu\mathbf{HL}^0_2$ formula, and Mod is a finite set of $\mu\mathbf{HL}^0_2$ formulas. The instantiations for the respective equivalence relations are presented in the lower table of Fig. 3. In the case of 2-nested simulation equivalence, $\mathbf{\Phi}_\mathsf{s}$ stands for the formula that is obtained from this template for simulation equivalence. We define $\mathsf{deadlock}_i \triangleq \bigwedge_{a\in\mathsf{Act}}[a]_i\bot$.

The only equivalence that is shown in Fig. 1 but not dealt with so far is possible-futures equivalence. It is definable in $\mu\mathbf{HL}^2_2$ through

$$\Big(\mu\mathscr{F}.\quad \lambda G_1, G_2.\lambda X.\quad G_1\ (G_2\ X) \vee \bigvee_{a\in\mathsf{Act}} \big(\mathscr{F}\ (\langle a\rangle_1 \circ G_1)\ ([a]_2 \circ G_2)\ X\big)\Big) \quad \lambda X.X \quad \lambda X.X \quad \Psi_\mathsf{t}$$

where $\Psi_t = \Phi_t \vee \Phi_t[1 \leftrightarrow 2]$ is the negation of the characteristic formula for trace equivalence. It remains to be seen whether or not it is also definable in $\mu\mathbf{HL}_2^1$ like the other equivalences are.

5 Model-Checking $\mu\mathbf{HL}_2^1$

5.1 From Model Checking to Process Equivalence Checking

The characterisations of process equivalences by modal fixpoint formulas give a uniform treatment of the descriptive complexity of such equivalence relations. However, they do not (yet) provide an algorithmic treatment. The aim of this section is to do so. To this end, we explain how to do model checking for $\mu\mathbf{HL}_\omega^\omega$. In fact, much less suffices already. Remember that the input to a model checking procedure is a pair consisting of — typically — an LTS and a formula. Higher-dimensionality of the underlying logic means that the input is a pair consisting of a tuple of LTS on one side and a formula on the other. Now any algorithm that does model checking for a pair of LTS and any formula $\Phi_{\mathscr{R}}$ given in the previous section is in fact an algorithm that decides the process equivalence $\mathscr{R}$. Thus, for these purposes it suffices to explain how to do model checking for any fragment that encompasses the formulas given there.

Here we restrict our attention to the fragment $\mu\mathbf{HL}_2^1$. This captures all process equivalences considered here apart from possible-futures equivalence, because all their characteristic formulas are naturally of dimension 2 — they describe a binary relation — and are of order 1. The extension to higher dimensionality is straight-forward. The extension to higher orders is also possible but not done here for ease of presentation.

5.2 A Symbolic Model-Checking Algorithm

We give a model checking algorithm for $\mu\mathbf{HL}_2^1$ that can be seen as a suitable extension of the usual fixpoint iteration algorithm for the modal μ-calculus. It merges the ideas used in model checking for the higher-dimension μ-calculus [11] and for higher-order fixpoint logic [3, 2].

Let Φ be a well-typed formula of $\mu\mathbf{HL}_2^1$. Then each of its subformulas has a type of the form $\mathbf{Pr}_2^{\nu_1} \to \ldots \to \mathbf{Pr}_2^{\nu_m} \to \mathbf{Pr}_2$ for some $m \geq 0$. Algorithm 1 takes as input two LTS $\mathscr{T}_i = (\mathsf{Pr}_i, \mathsf{Act}, \to_i)$ for $i \in \{1,2\}$ and an $\mu\mathbf{HL}_2^1$ formula Φ, and returns the set of all pairs of processes from these two LTS that satisfy Φ. Model checking is done by simply computing the semantics of each such subformula on the two underlying LTS.

The difference to model checking the modal μ-calculus is the handling of higher-order subformulas. Note that the semantics of a function of type $\mathbf{Pr}_2^{\nu_1} \to \ldots \to \mathbf{Pr}_2^{\nu_m} \to \mathbf{Pr}_2$ over a pair of LTS with n_1, respectively n_2 many processes can be represented as a table with $(2^{n_1 \cdot n_2})^m$ many entries — one for each possible combination of argument values to this function. Algorithm MC is designed to compute such a table for the corresponding subformulas.

Theorem 1. *Let Φ be a closed $\mu\mathbf{HL}_2^1$ formula of size k, and $\mathscr{T}_1, \mathscr{T}_2$ be two finite LTS, each of size n at most. The call of* MC$(\Phi, [])$ *correctly computes $[\![\emptyset \vdash \Phi : \mathbf{Pr}_2]\!]$ with respect to $\mathscr{T}_1, \mathscr{T}_2$ in time $\mathscr{O}(n^2 \cdot 2^{n^2 k^2})$.*

Proof. (Sketch) Correctness is established through a straight-forward induction on the structure of Φ. Note that the theorem is too weak to be used as an inductive invariant. Instead, one can easily prove the following stronger assertion: for any provable statement $\Gamma \vdash \Psi : \tau$ and any interpretation η, MC(Ψ, η) computes $[\![\Gamma \vdash \Psi : \tau]\!]_\eta$. For most cases this follows immediately from the definition of the semantics and the induction hypothesis. For fixpoint formulas it also uses the well-known characterisation of least

Algorithm 1 Model Checking $\mu\mathbf{HL}_2^1$

procedure MC(Φ,ρ) ▷ assume $\mathscr{T}_i = (\mathsf{Pr}_i, \mathsf{Act}, \to_i)$ to be fixed for $i = 1,2$
  case Φ **of**
    $\top$: **return** $\mathsf{Pr}_1 \times \mathsf{Pr}_2$
    x: **return** $\rho(x)$ ▷ some variable of type $\mathbf{Pr}_2^{\nu_1} \to \ldots \to \mathbf{Pr}_2^{\nu_m} \to \mathbf{Pr}_2$
    $\neg\Psi$: **return** $(\mathsf{Pr}_1 \times \mathsf{Pr}_2) \setminus \mathrm{MC}(\Psi,\rho)$
    $\Psi_1 \wedge \Psi_2$: **return** $\mathrm{MC}(\Psi_1,\rho) \cap \mathrm{MC}(\Psi_2,\rho)$
    $\langle a\rangle_1\Psi$: **return** $\{(P_1,P_2) \mid \exists P' \in \mathsf{Pr}_1 \text{ s.t. } P_1 \xrightarrow{a}_1 P' \text{ and } (P',P_2) \in \mathrm{MC}(\Psi,\rho)\}$
    $\langle a\rangle_2\Psi$: **return** $\{(P_1,P_2) \mid \exists P' \in \mathsf{Pr}_2 \text{ s.t. } P_2 \xrightarrow{a}_2 P' \text{ and } (P_1,P') \in \mathrm{MC}(\Psi,\rho)\}$
    $\lambda x_1,\ldots,x_m : \mathbf{Pr}_2^{\nu_1} \to \ldots \to \mathbf{Pr}_2^{\nu_m} \to \mathbf{Pr}_2$:
      for all $(T_1,\ldots,T_m) \in (2^{\mathsf{Pr}_1\times\mathsf{Pr}_2})^m$ **do**
        $F(T_1,\ldots,T_m) \leftarrow \mathrm{MC}(\Psi,\eta[x_1 \mapsto T_1,\ldots,x_m \mapsto T_m])$
      end for
      return F
    $\Psi\ \Psi_1\ \ldots\ \Psi_m$:
      return $\mathrm{MC}(\Psi,\rho)(\mathrm{MC}(\Psi_1,\rho),\ldots,\mathrm{MC}(\Psi_m,\rho))$
    $\mu x : \mathbf{Pr}_2^{\nu_1} \to \ldots \to \mathbf{Pr}_2^{\nu_m} \to \mathbf{Pr}_2.\Psi$:
      for all $(T_1,\ldots,T_m) \in (2^{\mathsf{Pr}_1\times\mathsf{Pr}_2})^m$ **do**
        $F(T_1,\ldots,T_m) \leftarrow \emptyset$
      end for
      repeat
        $F' \leftarrow F$
        for all $(T_1,\ldots,T_m) \in (2^{\mathsf{Pr}_1\times\mathsf{Pr}_2})^m$ **do**
          $F(T_1,\ldots,T_m) \leftarrow \mathrm{MC}(\Psi,\rho[x \mapsto F'])$
        end for
      until $F = F'$
      return F
  end case
end procedure

fixpoints by their chain of approximants. Note that the underlying power lattice is finite, even for higher-order types. Thus, fixpoint iteration from below — as done in algorithm MC — converges to the least fixpoint of the corresponding function in a finite number of steps.

The upper bound on the worst-case running time is established as follows. Note that k is an upper bound on the arity of each subformulas first-order type, i.e. in $\mathbf{Pr}_2^{\nu_1} \to \ldots \to \mathbf{Pr}_2^{\nu_m} \to \mathbf{Pr}_2$ we have $m \leq k$. Clearly, the running time for each case-clause is dominated by the one for fixpoint formulas which — disregarding recursive calls — can be done in time $\mathscr{O}(n^2 \cdot 2^{n^2 k})$. Note that it needs to fill a table with $2^{n^2 k}$ many entries using fixpoint iteration. Each table entry can change at most n^2 many times due to monotonicity. Furthermore, note that it is not the case that the semantics of each subformula is only computed once. Because of nested fixpoint formulas, we obtain an additional exponent which is bounded by the number of fixpoint formulas, i.e. also bounded by k, resulting in an upper bound of $\mathscr{O}(n^2 \cdot 2^{n^2 k^2})$. □

This establishes exponential-time upper bounds for all the process equivalence relations which can be defined in $\mu\mathbf{HL}_2^1$.

Corollary 2. *Trace, completed trace, failure, failure trace, readiness and ready trace equivalence can be checked in time* $2^{\mathcal{O}(n^2)}$.

It is easily checked that for $\mu\mathbf{HL}_2^0$ formulas, algorithm MC runs in time $\mathcal{O}((kn^2)^k)$. By instantiation we obtain polynomial-time algorithms for further process equivalences.

Corollary 3. *Completed, ready, 2-nested, bi- and simulation equivalence can be checked in polynomial time.*

We point out that algorithm MC can be made to work symbolically on BDDs just like the algorithm for the μ-calculus can [4]. A function is then represented as a table of BDDs. Furthermore, it can straight-forwardly be extended to higher orders which increases the complexity by one exponential per order. As a result, we obtain the following.

Proposition 4. *Possible-futures equivalence can be checked in doubly exponential time.*

5.3 Need-Driven Function Evaluation

Algorithm MC computes values for functions in a very naïve and brute-force way: it tabulates all possible arguments to the function and computes all their values. This results in far too many value computations than are needed in order to compute $[\![\emptyset \vdash \Phi : \mathbf{Pr}_2]\!]$ for any closed formula Φ. Consider for example $(\lambda X^+ : \mathbf{Pr}_2.[a]_2 X)\ \bot$. Its semantics is the set of all pairs (P,Q) such that Q has no a-successors. However, algorithm MC would compute the set of all pairs (P,Q) such that all a-successors of Q belong to the second components of any set of pairs (P,Q').

Need-driven function evaluation avoids these unnecessary computations. For formulas without fixpoint quantifiers it could easily be realised by evaluating arguments first, and then passing these values to the computation of the function, comparable to lazy evaluation in functional programming. Need-driven function evaluation in the presence of fixpoint quantifiers is more complicated, though [8]. For recursively defined functions it is not sufficient to simply compute their value on a given argument using fixpoint iteration for instance, but the computation of the value on some argument may need the value on some other argument. Need-driven function evaluation intertwines the computation of these values with the exploration of the function's domain [2]. The following example shows the optimising potential of this technique.

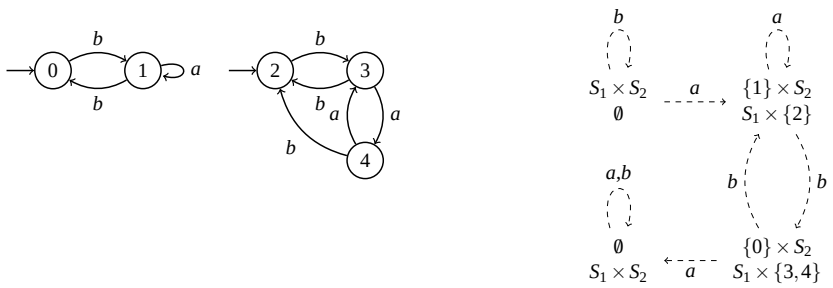
Example 1. Consider the two LTS presented in Fig. 4. Let $S_1 \triangleq \{0,1\}$ and $S_2 \triangleq \{2,3,4\}$ be their state spaces. We will show how need-driven function evaluation works on algorithm MC, these two LTS and the formula that tests for trace equivalence over $\mathsf{Act} = \{a,b\}$, namely

$$\Phi_{\mathsf{t}} \;=\; \big(\mu F(X,Y).\ (X \wedge Y)\ \vee\ (F\ \langle a\rangle_1 X\ [a]_2 Y)\ \vee\ (F\ \langle b\rangle_1 X\ [b]_2 Y)\big)\ \top\ \bot\ .$$

Note that it should be true on a pair (P,Q) of processes iff P has a trace that Q does not.

Φ_{t} defines a function $\mathscr{F}$ via least-fixpoint recursion. It takes two arguments X and Y and returns the union of their intersection with the value of $\mathscr{F}$ applied to two other sets of arguments, defined by $\langle a\rangle_1 X$ and $[a]_2 Y$ in one case and equally with b in the other. Moreover, we are interested in the value of $\mathscr{F}$ on the argument pair $(S_1 \times S_2, \emptyset)$.

Need-driven function evaluation builds the table for $\mathscr{F}$ via fixpoint iteration, i.e. by building its approximants $\mathscr{F}^0$, $\mathscr{F}^1$, ... with $\mathscr{F}^0(X,Y) = \emptyset$ for any $X,Y \subseteq S_1 \times S_2$, starting with the argument on which we need the function's value. Since $\mathscr{F}$ is recursively defined, the value on this argument may need the value on other arguments. Fig. 4 shows the part of the dependency graph that is reachable from this initial argument, where an arrow $(X,Y) \overset{a}{\dashrightarrow} (X',Y')$ states that the computation of the value on (X,Y)



X	$S_1 \times S_2$	$\{1\} \times S_2$	$\{0\} \times S_2$	$\emptyset$
Y	$\emptyset$	$S_1 \times \{2\}$	$S_1 \times \{3,4\}$	$S_1 \times S_2$
$\mathscr{F}^0$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\mathscr{F}^1$	$\emptyset$	$\{(1,2)\}$	$\{(0,3),(0,4)\}$	$\emptyset$
$\mathscr{F}^2$	$\{(1,2)\}$	$\{(1,2),(0,3),(0,4)\}$	$\{(1,2),(0,3),(0,4)\}$	$\emptyset$
$\mathscr{F}^3$	$\{(1,2),(0,3),(0,4)\}$	$\{(1,2),(0,3),(0,4)\}$	$\{(1,2),(0,3),(0,4)\}$	$\emptyset$
$\mathscr{F}^4$	$\{(1,2),(0,3),(0,4)\}$	$\{(1,2),(0,3),(0,4)\}$	$\{(1,2),(0,3),(0,4)\}$	$\emptyset$

Figure 4: Example of need-driven function evaluation for trace equivalence checking.

triggers the first recursive call on (X',Y'). Similarly, an arrow $\overset{b}{\dashrightarrow}$ shows the dependency via the second recursive call.

Finally, Fig. 4 shows the table of values computed by fixpoint iteration restricted to those arguments that occur in the dependency graph, i.e. the part of the function's domain which is necessary to iterate on until stability in order to determine the fixpoint's value on the initial argument. The optimising potential of need-driven function evaluation is justified by the table's width: note that the naïve version of algorithm MC would fill that table for all possible arguments of which there are $(2^{2\cdot 3})^2 = 4096$ while it suffices to reach stability on these 4 arguments alone.

5.4 Partial Evaluation

The example above shows another potential for optimisation. Remember that the formals defining process equivalences do not depend on the actual LTS on which they are being evaluated. Thus, we can devise a simpler algorithm for trace equivalence for instance by analysing the behaviour of MC on an arbitrary pair of LTS and the fixed formula Φ_{t}. We note that the filling of the table values follows a simple scheme: the value in row i at position (X,Y) is the union of three values, namely the one in row 1 of this position and the values in row $i-1$ of the two successors of (X,Y) in the dependency graph. This leads to the simple Algorithm 2 for trace equivalence checking.

6 Conclusion and Further Work

We have presented a highly expressive modal fixpoint logic which can define many process equivalence relations. We have presented a model checking algorithm which can be instantiated in order to yield decision procedures for the relations on finite systems. This re-establishes already known decidability results [7]. Its main contribution, though, is the — to the best of our knowledge — first framework that provides a generic and uniform algorithmic approach to process equivalence checking via defining

Algorithm 2 Trace Equivalence Checking

procedure TREQ($\mathscr{T}_1, \mathscr{T}_2$) ▷ let $\mathscr{T}_i = (\mathsf{Pr}_i, \mathsf{Act}, \to_i)$
  $X_0 \leftarrow \mathsf{Pr}_1 \times \mathsf{Pr}_2$
  $Y_0 \leftarrow \emptyset$
  $\mathscr{W} = \{(X_0, Y_0)\}$ ▷ work list
  $\mathscr{D} = \emptyset$ ▷ domain of the dependency graph
  while $\mathscr{W} \neq \emptyset$ **do** ▷ build dependency graph
    remove some (X,Y) from $\mathscr{W}$
    for all $a \in \mathsf{Act}$ **do**
      $(X',Y') \leftarrow (\langle a \rangle_1 X, [a]_2 Y)$
      $d_a(X,Y) \leftarrow (X',Y')$ ▷ record arrows in dependency graph
      $\mathscr{D} \leftarrow \mathscr{D} \cup \{(X,Y)\}$
      if $(X',Y') \notin \mathscr{D}$ **then**
        $\mathscr{W} \leftarrow \mathscr{W} \cup \{(X',Y')\}$
      end if
    end for
  end while
  for all $(X,Y) \in \mathscr{D}$ **do**
    $I(X,Y) \leftarrow X \cap Y$
    $F(X,Y) \leftarrow \emptyset$
  end for
  repeat
    for all $(X,Y) \in \mathscr{D}$ **do**
      $F(X,Y) \leftarrow I(X,Y) \cup \bigcup_{a \in \mathsf{Act}} F(d_a(X,Y))$
    end for
  until F does not change anymore
  return $F(X_0, Y_0)$
end procedure

formulas. In particular, it allows technology from the well-developed field of model checking to be transferred to process equivalence checking.

There is a lot of potential further work into this direction. The exponential-time bound for the trace-like equivalences is not optimal since they are generally PSPACE-complete [7]. It remains to be seen whether the formulas defining them have a particular structure that would allow a PSPACE model checking algorithm for instance. This would make a real improvement since model checking $\mu\mathbf{HL}^1_2$ is EXPTIME-hard in general which follows from such a bound for the first-order fragment of HFL [3]. Also, it remains to be seen whether or not possible-futures equivalence can be defined $\mu\mathbf{HL}^1_2$.

We leave the exact formulation of a model checking procedure for the entire logic $\mu\mathbf{HL}^\omega_\omega$ for future work. Such an algorithm may be interesting for other fields as well, not just process equivalence checking.

There are more equivalence relations which we have not considered here for lack of space, e.g. possible-worlds equivalence, tree equivalence, 2-bounded trace bisimulation, etc. We believe that creating defining formulas for them in $\mu\mathbf{HL}^\omega_\omega$ is of no particular difficulty.

We intend to also investigate the practicability of this approach. To this end, we aim to extend an existing prototypical implementation of a symbolic model checking tool for the higher-dimension μ-

calculus to $\mu\mathbf{HL}^1_2$, and possible $\mu\mathbf{HL}^\omega_\omega$ in general. We believe that using need-driven function evaluation and partial evaluation techniques will have a major influence on the applicability of the algorithms obtained by instantiating the generic model checking procedure with a fixed formula.

References

[1] H. R. Andersen (1993): *Verification of Temporal Properties of Concurrent Systems*. Ph.D. thesis, Dept. of Computer Science, University of Aarhus, DK.

[2] R. Axelsson & M. Lange (2007): *Model Checking the First-Order Fragment of Higher-Order Fixpoint Logic*. In: *Proc. 14th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'07*, *LNCS* 4790, Springer, pp. 62–76, doi:10.1007/978-3-540-75560-9_7.

[3] R. Axelsson, M. Lange & R. Somla (2007): *The Complexity of Model Checking Higher-Order Fixpoint Logic*. *Logical Methods in Computer Science* 3, pp. 1–33, doi:10.2168/LMCS-3(2:7)2007.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill & L. J. Hwang (1992): *Symbolic Model Checking:* 10^{20} *States and Beyond*. *Information and Computation* 98(2), pp. 142–170, doi:10.1016/0890-5401(92)90017-A.

[5] R. Cleaveland & B. Steffen (1991): *Computing Behavioural Relations, Logically*. In: *Proc. 18th Int. Coll. on Automata, Languages and Programming, ICALP'91*, *LNCS* 510, Springer, pp. 127–138, doi:10.1007/3-540-54233-7_129.

[6] R. J. van Glabbeek (2001): *The Linear Time – Branching Time Spectrum I; The Semantics of Concrete, Sequential Processes*. In: *Handbook of Process Algebra*, chapter 1, Elsevier, pp. 3–99, doi:10.1016/B978-044482830-9/50019-9.

[7] H. Hüttel & S. Shukla (1996): *On the Complexity of Deciding Behavioural Equivalences and Preorders*. Technical Report SUNYA-CS-96-03, State University of New York at Albany.

[8] N. Jørgensen (1994): *Finding Fixpoints in Finite Function Spaces using Neededness Analysis and Chaotic Iteration*. In: *Proc. 1st Int. Static Analysis Symposium, SAS'94*, *LNCS* 864, Springer, pp. 329–345, doi:10.1007/3-540-58485-4_50.

[9] P. C. Kanellakis & S. A. Smolka (1990): *CCS Expressions, Finite State Processes, and Three Problems of Equivalence*. *Information and Computation* 86(1), pp. 43–68, doi:10.1016/0890-5401(90)90025-D.

[10] B. Knaster (1928): *Un théorèm sur les fonctions d'ensembles*. *Annals Soc. Pol. Math* 6, pp. 133–134.

[11] M. Lange & E. Lozes (2012): *Model Checking the Higher-Dimensional Modal μ-Calculus*. In: *Proc. 8th Workshop on Fixpoints in Computer Science, FICS'12*, *Electr. Proc. in Theor. Comp. Sc.* 77, pp. 39–46, doi:10.4204/EPTCS.77.6.

[12] M. Otto (1999): *Bisimulation-invariant PTIME and higher-dimensional μ-calculus*. *Theor. Comput. Sci.* 224(1-2), pp. 237–265, doi:10.1016/S0304-3975(98)00314-4.

[13] S. K. Shukla, H. B. Hunt III & D. J. Rosenkrantz (1996): *HORNSAT, Model Checking, Verification and games (Extended Abstract)*. In: *8th Int. Conf. on Computer Aided Verification, CAV'96*, *LNCS* 1102, Springer, pp. 99–110, doi:10.1007/3-540-61474-5_61.

[14] C. Stirling (2001): *Modal and Temporal Properties of Processes*. Texts in Computer Science, Springer.

[15] A. Tarski (1955): *A Lattice-theoretical Fixpoint Theorem and its Application*. *Pacific Journal of Mathematics* 5, pp. 285–309.

[16] M. Viswanathan & R. Viswanathan (2004): *A Higher Order Modal Fixed Point Logic*. In Ph. Gardner & N. Yoshida, editors: *CONCUR*, *Lecture Notes in Computer Science* 3170, Springer, pp. 512–528, doi:10.1007/978-3-540-28644-8_33.

A Definitions of Process Equivalences

Finite Trace Equivalence. Let $T(P) := \{t \mid \exists Q.P \xrightarrow{t} Q\}$ be the set of all finite traces of P. Two processes P and Q are *finite trace equivalent*, $P \sim_{\mathsf{t}} Q$, if $T(P) = T(Q)$.

Completed Trace Equivalence. A sequence $t \in \mathsf{Act}^*$ of a process P is a *completed trace* if there is a Q s.t. $P \xrightarrow{t} Q$ and $I(Q) = \emptyset$. Let $CT(P)$ be the set of all completed traces of P. Two processes P and Q are *completed trace equivalent*, $P \sim_{\mathsf{ct}} Q$, if $T(P) = T(Q)$ and $CT(P) = CT(Q)$.

Failures Equivalence. A pair $\langle t, A\rangle$ is a *failure pair* of P if there is a process Q s.t. $P \xrightarrow{t} Q$ and $I(Q) \cap A = \emptyset$. Let $F(P)$ denote the set of all failure pairs of P. Two processes P and Q are *failures equivalent*, $P \sim_{\mathsf{f}} Q$, if $F(P) = F(Q)$.

Failure Trace Equivalence. A *failure trace* is a $u \in (\mathsf{Act} \cup 2^{\mathsf{Act}})^*$. We extend the reachability relation of processes to failure traces by including $P \xrightarrow{\varepsilon}_{\mathsf{ft}} P$ for any P and the triples $P \xrightarrow{A}_{\mathsf{ft}} Q$ whenever $I(P) \cap A = \emptyset$, and then closing it off under compositions: if $P \xrightarrow{u}_{\mathsf{ft}} R$ and $R \xrightarrow{u'}_{\mathsf{ft}} Q$ then $P \xrightarrow{uu'}_{\mathsf{ft}} Q$. Let $FT(P) := \{u \mid \exists Q.P \xrightarrow{u}_{\mathsf{ft}} Q\}$ be the set of all failure traces of P. Two processes P and Q are *failure trace equivalent*, $P \sim_{\mathsf{ft}} Q$, if $FT(P) = FT(Q)$.

Readiness Equivalence. A pair $\langle t, A\rangle$ is a *ready pair* of P if there is a process Q s.t. $P \xrightarrow{t} Q$ and $A = I(Q)$. Let $R(P)$ denote the set of all ready pairs of P. Two processes P and Q are *ready equivalent*, $P \sim_{\mathsf{r}} Q$, if $R(P) = R(Q)$.

Ready Trace Equivalence. A *ready trace* is a $u \in (\mathsf{Act} \cup 2^{\mathsf{Act}})^*$. We extend the reachability relation of processes to ready traces by including $P \xrightarrow{\varepsilon}_{\mathsf{rt}} P$, and $P \xrightarrow{A}_{\mathsf{rt}} Q$ whenever $I(P) = A$, and closing it off under compositions as in the case of failure trace equivalence. Let $RT(P) := \{u \mid \exists Q.P \xrightarrow{u}_{\mathsf{rt}} Q\}$ be the set of all ready traces of P. Two processes P and Q are *ready trace equivalent*, $P \sim_{\mathsf{rt}} Q$, if $RT(P) = RT(Q)$.

Possible-Futures Equivalence. A pair $\langle t, L\rangle$ is a *possible future* of P if there is a process Q s.t. $P \xrightarrow{t} Q$ and $L = T(Q)$. Let $PF(P)$ be the set of all possible futures of P. Two processes P and Q are *possible-futures equivalent*, $P \sim_{\mathsf{pf}} Q$, if $PF(P) = PF(Q)$.

Simulation Equivalence. A binary relation $\mathscr{R}$ is a *simulation* on processes if it satisfies for any $a \in \mathsf{Act}$: if $(P,Q) \in \mathscr{R}$ and $P \xrightarrow{a} P'$, then $\exists Q'.Q \xrightarrow{a} Q'$ and $(P',Q') \in \mathscr{R}$. P and Q are *similar*, $P \sim_{\mathsf{s}} Q$, if there are simulations $\mathscr{R}$ and $\mathscr{R}'$ s.t. $(P,Q) \in \mathscr{R}$ and $(Q,P) \in \mathscr{R}'$.

Completed Simulation Equivalence. A binary relation $\mathscr{R}$ is a *completed simulation* on processes if it satisfies for any $a \in \mathsf{Act}$: if $(P,Q) \in \mathscr{R}$ and $P \xrightarrow{a} P'$, then $\exists Q'.Q \xrightarrow{a} Q'$ and $(P',Q') \in \mathscr{R}$. And if $(P,Q) \in \mathscr{R}$ then $I(P) = \emptyset \Leftrightarrow I(Q) = \emptyset$. Two processes P and Q are *completed simulation equivalent*, $P \sim_{\mathsf{cs}} Q$, if there are completed simulations $\mathscr{R}$ and $\mathscr{R}'$ s.t. $(P,Q) \in \mathscr{R}$ and $(Q,P) \in \mathscr{R}'$.

Ready Simulation Equivalence. A binary relation $\mathscr{R}$ is a *ready simulation* on processes if it satisfies for any $a \in \mathsf{Act}$: if $(P,Q) \in \mathscr{R}$ and $P \xrightarrow{a} P'$, then $\exists Q'.Q \xrightarrow{a} Q'$ and $(P',Q') \in \mathscr{R}$. And if $(P,Q) \in \mathscr{R}$ then $I(P) = I(Q)$. Two processes P and Q are *ready simulation equivalent*, $P \sim_{\mathsf{rs}} Q$, if there are ready simulations $\mathscr{R}$ and $\mathscr{R}'$ s.t. $(P,Q) \in \mathscr{R}$ and $(Q,P) \in \mathscr{R}'$.

2-Nested Simulation Equivalence. A binary relation $\mathscr{R}$ is a *2-nested simulation* on processes if it satisfies for any $a \in \mathsf{Act}$: if $(P,Q) \in \mathscr{R}$ and $P \xrightarrow{a} P'$, then $\exists Q'.Q \xrightarrow{a} Q'$ and $(P',Q') \in \mathscr{R}$. And if $(P,Q) \in \mathscr{R}$ then $Q \sim_{\mathsf{s}} P$. Two processes P and Q are *2-nested simulation equivalent*, $P \sim_{\mathsf{2s}} Q$, if there are 2-nested simulations $\mathscr{R}$ and $\mathscr{R}'$ s.t. $(P,Q) \in \mathscr{R}$ and $(Q,P) \in \mathscr{R}'$.

Bisimulation. A binary relation $\mathscr{R}$ is a *bisimulation* on processes if it satisfies for any $a \in \mathsf{Act}$: if $(P,Q) \in \mathscr{R}$ and $P \xrightarrow{a} P'$, then $\exists Q'.Q \xrightarrow{a} Q'$ and $(P',Q') \in \mathscr{R}$. And if $(P,Q) \in \mathscr{R}$ and $Q \xrightarrow{a} Q'$, then $\exists P'.P \xrightarrow{a} P'$ and $(P',Q') \in \mathscr{R}$. Two processes P and Q are *bisimilar*, $P \sim_{\mathsf{b}} Q$, if there is a bisimulation $\mathscr{R}$ s.t. $(P,Q) \in \mathscr{R}$.