Programming Research Group

A UNIFYING APPROACH TO DATA-INDEPENDENCE

Ranko Lazić
David Nowak

PRG-TR-4-00

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

A Unifying Approach to Data-independence[⋆] [⋆⋆]

Ranko Lazić[⋆⋆⋆] and David Nowak

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

{Ranko.Lazic, David.Nowak}@comlab.ox.ac.uk

Abstract. A concurrent system is data-independent with respect to a data type when the only operation it can perform on values of that type is equality testing. The system can also assign, input, nondeterministically choose, and output such values. Based on this intuitive definition, syntactic restrictions which ensure data-independence have been formulated for a variety of different formalisms. However, it is difficult to see how these are related.

We present the first semantic definition of data-independence which allows equality testing, and its extension which allows constant symbols and predicate symbols. Both are special cases of a definition of when a family of labelled transition systems is parametric. This provides a unified approach to data-independence and its extensions.

The paper also contains two theorems which, given a system and a specification which are data-independent, enable the verification for all instantiations of the data types (and of the constant symbols and the predicate symbols, in the case of the extension) to be reduced to the verification for a finite number of finite instantiations.

We illustrate the applicability of the approach to particular formalisms by a programming language similar to UNITY, and by linear-time temporal logic.

1 Introduction

Model checking. During the past 20 years, model checking [3] has developed into a key approach to verifying finite-state concurrent systems. It has a number of advantages over traditional approaches that are based on simulation, testing, and deduction. In particular, model checking is automatic, and provides exact

⋆ A preliminary version of this paper was presented at the 11th International Conference on Concurrency Theory (CONCUR 2000), PSU, State College, Pennsylvania, USA, 22–25 August 2000 [13].

⋆⋆ The research in this paper was supported in part by the EPSRC Standard Research Grant GR/M32900.

⋆⋆⋆ Also affiliated to the Mathematical Institute, Belgrade. Supported in part by a Junior Research Fellowship at Christ Church, Oxford. Previously supported by a Domus and Harmsworth Senior Scholarship at Merton College, Oxford, and by a scholarship from Hajrija & Boris Vukobrat and Copechim France SA.

counter-examples in case of error. Model checking has been applied successfully to industrial systems.

The three main challenges for model checking are: the state-explosion problem, caused by the fact that the state space of a system may grow exponentially with the number of its parallel components; model checking of parameterised systems, where one or more parameters of a system have large or infinite ranges; and model checking of infinite-state systems. Over the past 10 to 15 years, considerable progress has been made on each of these challenges.

Data-independence. A system is data-independent with respect to a data type when the only operation it can perform on values of that type is equality testing. The system can also assign, input, nondeterministically choose, and output such values. Many practically important systems are data-independent. For example: communication protocols are usually data-independent with respect to the types of data which they communicate; memory or database systems may be data-independent with respect to the type of values which they store, as well as the type of addresses.

A data type with respect to which a system is data-independent can be treated as a parameter of the system, i.e. as a type variable that can be instantiated by any concrete type. If correctness of the system is important for all instantiations, we get a special case of the problem of model checking parameterised systems. A simpler case is when correctness is important only for an infinite instantiation (for example, the type of natural numbers), which gives us a special case of the problem of model checking infinite-state systems.

Techniques for model checking data-independent systems have been developed for a number of different frameworks [24, 10, 9, 7, 8, 12]. They enable verification for all instantiations of the data type to be reduced, either statically or dynamically, to the verification for a finite number of finite instantiations (or only one). Consequently, in the special case of data-independent systems, the challenges of model checking parameterised and infinite-state systems have been overcome.

Recently, it has been shown that data-independence techniques can be combined with induction to yield powerful methods for model checking of systems which are parameterised by their network topology [5], and that they can be applied to enable the verification of security protocols (which involves infinite-state systems) by model checking [20].

This paper. Based on the intuitive definition of data-independence stated above, it is straightforward to formulate syntactic restrictions which ensure that a system or a specification is data-independent with respect to a data type. In [10, 9, 7, 8, 12], such syntactic restrictions are given for each of the formalisms used. However, it is hard to see how the restrictions and the results for different formalisms are related.

Data-independence therefore needs to be defined semantically. This has turned out to be difficult, since a satisfactory definition must not be stronger than the syntactic restrictions, but it has to be strong enough to prove results which enable reductions to finite instantiations. The only semantic definition in the literature is in [24], and it does not allow equality testing.

In this paper, we present a semantic definition of data-independence, in which we can choose whether to allow equality testing or not. The definition is a special case of a definition of what it means for a family of labelled transition systems to be parametric. Intuitively, systems that use type variables and operations on them (which can be thought of as abstract data types) in a polymorphic manner give rise to parametric families of LTSs, where each LTS is the semantics of the system with respect to an instantiation of the type variables and the operations. In the special case where the operations are only equality tests, the family of LTSs is data-independent.

The fact that the definition of data-independence is a special case of the definition of parametricity makes it easy to extend. We demonstrate this by extending it to allow constant symbols whose types are type variables, and predicate symbols which are functions from type variables to fixed non-empty finite types.

The definitions also apply to Rabin automata [21], by regarding them as a subclass of labelled transition systems.

Using the definition of data-independence and its extension, we prove two theorems which enable reductions to finite numbers of finite instantiations of the type variables (and the constants and predicates, in the case of the extension).

We may consider this to be a unifying approach to data-independence (and its extensions). Firstly, the definitions and theorems are not in terms of any particular formalism, but are semantic. Secondly, a wide range of specification frameworks, such as linear-time temporal logic [23] and CSP refinement [19], can be translated into labelled transition systems and Rabin automata. In particular, this is possible for each of the frameworks in [24, 9, 7, 8, 12].

The approach is therefore a solid basis for future work. Moreover, in comparison with the formalism-specific approaches, it allows one to draw and to exploit connections between data-independence and parametricity [18], abstract interpretation [4], and relational data refinement [11] more clearly and more deeply.

To illustrate applying the approach to particular formalisms, we consider a programming language similar to UNITY [1], and linear-time temporal logic.

Comparisons with related work. As we stated, the only semantic definition of data-independence in the literature [24] does not allow equality testing. More precisely, it excludes most systems which are data-independent according to the intuitive definition given earlier in this paper and which use equality testing. Moreover, if the definition in [24] is weakened by restricting the functions $f : D \to D'$ to be injective, so that it does allow equality testing, it becomes too weak to imply results which enable reductions to finite instantiations.

In [17], data-insensitive programs are defined semantically as those that have finite simulation quotients which preserve the values of all control variables. However, most such programs are not data-independent, and the definition applies only to programs in which the relevant data types are infinite.

Theorem 2 in this paper is at least as powerful as the data-independence results in [9], the main results in [7], the general result in [8], and the main results in [12, 15]. Moreover, nondeterministic sequential Rabin automata that may be infinite-state and that may be defined using state variables, which provide the specification framework in this paper, are more expressive than each of the specification frameworks in [9, 7, 8, 12, 15].

Because of complex technical details in differences among frameworks, the comparisons in the previous paragraph are not formal. They are based on the facts that the labelled transition systems and the Rabin automata used in this paper can express safety specifications and liveness specifications under fairness assumptions, and that the non-data-independent constructs in the results in the literature can be reduced to constants and predicates. In any case, one may regard the main novelties in this paper to lie in the definitions rather than the theorems.

Organisation. Section 2 is devoted to preliminaries. In Section 3, we define what it means for a family of labelled transition systems to be parametric (which also applies to families of Rabin automata), and then define data-independence and its extension by constants and predicates as special cases. The two reduction theorems are presented in Section 4. Section 5 considers a UNITY-like programming language and linear-time temporal logic. Section 6 contains some conclusions and directions for future work.

In order not to interrupt the flow of the presentation, proofs are given in the appendix.

2 Preliminaries

Tuples and maps. For any tuple $\underline{a} = (a_1, \ldots, a_n)$ and $i \in \{1, \ldots, n\}$, let $\pi_i(\underline{a}) = a_i$.

For any tuple $\underline{a} = (a_1, \ldots, a_n)$, any mutually distinct $i_1, \ldots, i_k \in \{1, \ldots, n\}$, and any $b_1, \ldots, b_k$, let $\underline{a}[i_1 \mapsto b_1, \ldots, i_k \mapsto b_k]$ be the tuple obtained from $\underline{a}$ by replacing each a_{i_j} by b_j.

For any map f, any mutually distinct $a_1, \ldots, a_n$, and any $b_1, \ldots, b_n$, let $f[a_1 \mapsto b_1, \ldots, a_n \mapsto b_n]$ be the map which agrees with f except that it maps each a_i to b_i.

We also use the notation $[a_1 \mapsto b_1, \ldots, a_n \mapsto b_n]$ to mean the map which maps each a_i to b_i.

Syntax of types. In addition to type variables, we consider sum types, product types, inductive types (which can express types of finite lists or finite trees), and function types. These will act as types of operations (for example, $(X \times X) \rightarrow$ *Bool* for equality testing on a type variable X) in signatures of families of labelled transition systems or Rabin automata, and as types of states and transition labels of such families.

We assume *TypeVars* is an infinite set of names for type variables.

The syntax of types is defined by

$$T ::= X \in \mathit{TypeVars} \mid T_1 + \cdots + T_n \mid T_1 \times \cdots \times T_n \mid \mu X.T \mid T_1 \to T_2$$

where n ranges over nonnegative integers, and where in any $\mu X.T$, any free occurence of X in T must not be within a $T_1 \to T_2$ construct.

For any type T, we write $\mathit{Free}(T)$ for the set of all type variables which occur free in T.

For any types T, T'_1, ..., T'_n, and mutually distinct type variables X_1, ..., X_n, we write $T[T'_1/X_1, \ldots, T'_n/X_n]$ for a type obtained from T by substituting each T'_i for every free occurence of X_i in T.

Semantics of types. A *set map* is a partial map from *TypeVars* to sets, whose domain is finite.

For any type T, and any set map δ such that $\mathit{Free}(T) \subseteq \mathit{Domain}(\delta)$, we write $[\![T]\!]_\delta$ for the denotational semantics of T with respect to δ.

This is defined by

$$\begin{aligned}
[\![X]\!]_\delta &= \delta[\![X]\!] \\
[\![T_1 + \cdots + T_n]\!]_\delta &= \{1\} \times [\![T_1]\!]_\delta \cup \cdots \cup \{n\} \times [\![T_n]\!]_\delta \\
[\![T_1 \times \cdots \times T_n]\!]_\delta &= [\![T_1]\!]_\delta \times \cdots \times [\![T_n]\!]_\delta \\
[\![\mu X.T]\!]_\delta &= \bigcup_{k \in \mathbb{N}} \mathcal{F}^k(\{\}), \ \mathit{where}\ \mathcal{F}(A) = [\![T]\!]_{\delta[X \mapsto A]} \\
[\![T_1 \to T_2]\!]_\delta &= [\![T_1]\!]_\delta \to [\![T_2]\!]_\delta
\end{aligned}$$

Some standard types. We define some standard types by abbreviations:

$$\begin{aligned}
\mathit{Empty} &= \mathit{(the\ empty\ sum\ type)} \\
\mathit{Single} &= \mathit{(the\ empty\ product\ type)} \\
\mathit{Bool} &= \mathit{Single} + \mathit{Single} \\
\mathit{Enum}_l &= \overbrace{\mathit{Single} + \cdots + \mathit{Single}}^{l}, \ \mathit{for\ any}\ l \in \mathbb{N} \\
\mathit{Nat} &= \mu X.\mathit{Single} + X
\end{aligned}$$

We also define:

$$\begin{array}{lll}
\overline{\mathit{false}} = (1, ()) & \overline{\mathit{true}} = (2, ()) & \overline{\mathit{enum}_i} = (i, ()) \\
\overline{0} = (1, ()) & \overline{i+1} = (2, \overline{i}) &
\end{array}$$

so that we have:

$$\begin{array}{ll}
[\![\mathit{Bool}]\!]_{\{\}} = \{\overline{\mathit{false}}, \overline{\mathit{true}}\} & [\![\mathit{Enum}_l]\!]_{\{\}} = \{\overline{\mathit{enum}_1}, \ldots, \overline{\mathit{enum}_l}\} \\
[\![\mathit{Nat}]\!]_{\{\}} = \{\overline{0}, \overline{1}, \ldots\} &
\end{array}$$

Logical relations. Intuitively, given a system (or specification) which uses type variables and operations on them, it gives rise to a separate labelled transition system (or Rabin automaton) for each instantiation of the type variables and the operations. Supposing we have two such instantiations and, for each type variable, a relation between the two sets by which it is instantiated, logical relations [18] enable us to lift those relations to more complex types. In particular, they will enable us to relate the two instantiations of each of the operations, and to relate states and transition labels in the two labelled transition systems (or Rabin automata).

A *relation map* is a triple (ρ, δ, δ') such that

- ρ is a partial map from *TypeVars* to relations, whose domain is finite,
- δ and δ' are set maps with the same domain as ρ, and
- $\rho[\![X]\!]$ is a relation between $\delta[\![X]\!]$ and $\delta'[\![X]\!]$, for each $X \in \mathit{Domain}(\rho)$.

Given a relation map (ρ, δ, δ'), it determines a logical relation indexed by the types T such that $\mathit{Free}(T) \subseteq \mathit{Domain}(\rho)$. For any such type T, we write $[\![T]\!]_{(\rho,\delta,\delta')}$ for the component at T of the logical relation. This is a relation between the sets $[\![T]\!]_\delta$ and $[\![T]\!]_{\delta'}$, and is defined by

$$\begin{array}{l}
[\![X]\!]_{(\rho,\delta,\delta')} = \rho[\![X]\!] \\
[\![T_1 + \cdots + T_n]\!]_{(\rho,\delta,\delta')} = \{((i,a),(i',a')) \mid i = i' \ \wedge \ a[\![T_i]\!]_{(\rho,\delta,\delta')}a'\} \\
[\![T_1 \times \cdots \times T_n]\!]_{(\rho,\delta,\delta')} = \{(\underline{a},\underline{a'}) \mid \forall i \in \{1,\ldots,n\}.\pi_i(\underline{a})[\![T_i]\!]_{(\rho,\delta,\delta')}\pi_i(\underline{a'})\} \\
[\![\mu X.T]\!]_{(\rho,\delta,\delta')} = \bigcup_{k \in \mathbb{N}} R_k, \ \ \mathit{where} \\
\quad \mathcal{H}(R,A,A') = ([\![T]\!]_{\rho[X\mapsto R],\delta[X\mapsto A],\delta'[X\mapsto A']}, [\![T]\!]_{\delta[X\mapsto A]}, [\![T]\!]_{\delta'[X\mapsto A']}) \\
\quad (R_0, A_0, A'_0) = (\{\},\{\},\{\}) \\
\quad (R_{k+1}, A_{k+1}, A'_{k+1}) = \mathcal{H}(R_k, A_k, A'_k) \\
[\![T_1 \to T_2]\!]_{(\rho,\delta,\delta')} = \{(f,f') \mid a[\![T_1]\!]_{(\rho,\delta,\delta')}a' \ \Rightarrow \ f(a)[\![T_2]\!]_{(\rho,\delta,\delta')}f'(a')\}
\end{array}$$

The following is a standard auxiliary result [18]. It states that the definition above preserves the property of being an identity relation. In particular, for any type T in which no type variable occurs free, and any relation map (ρ, δ, δ'), $[\![T]\!]_{(\rho,\delta,\delta')}$ is the identity relation on $[\![T]\!]_{\{\}}$.

Proposition 1. *Suppose*

- (ρ, δ, δ) *is a relation map such that, for each* X, $\rho[\![X]\!]$ *is the identity relation on* $\delta[\![X]\!]$*, and*
- T *is a type such that* $\mathit{Free}(T) \subseteq \mathit{Domain}(\rho)$.

Then $[\![T]\!]_{(\rho,\delta,\delta)}$ *is the identity relation on* $[\![T]\!]_\delta$. □

Labelled transition systems. A *labelled transition system (LTS)* is a tuple $\mathcal{S} = (A, B, I, \longrightarrow)$ such that A and B are sets, $I \subseteq A$ and $\longrightarrow \subseteq A \times B \times A$.

We say that A is the set of states, B is the set of transition labels, I is the set of initial states, and $\longrightarrow$ is the transition relation.

$a \in A$ is *reachable* iff it can be reached from some $a_0 \in I$ by finitely many transitions.

$\mathcal{S}$ is *deterministic* iff, for each reachable $a \in A$, $\forall b \in B.|\{a' \mid a \xrightarrow{b} a'\}| \leq 1$.

$\mathcal{S}$ is *complete* iff, for each reachable $a \in A$, $\forall b \in B.|\{a' \mid a \xrightarrow{b} a'\}| \geq 1$.

A *run* of $\mathcal{S}$ is an infinite sequence $\underline{a} = (a_0, a_1, \ldots)$ such that $a_0 \in I$ and $\forall i \in \mathbb{N}.\exists b_i.a_i \xrightarrow{b_i} a_{i+1}$. For any infinite sequence $\underline{b} = (b_0, b_1, \ldots)$ which satisfies the previous formula, we say that $\underline{a}$ is a run of $\mathcal{S}$ on $\underline{b}$ from a_0.

Generalised Rabin automata. The kind of automata we work with in this paper is a generalisation of (sequential) Rabin automata [21]. The generalisation consists of allowing infinitely many states, infinitely many transition labels (i.e. infinite alphabets), sets of initial states rather than only one, and nondeterministic transition relations. These are sometimes convenient, both in theory and in applications, and they do not complicate the main developments in the paper.

We regard the generalised Rabin automata to be the subclass of LTSs in which states are $(1 + 2q)$-tuples, where the final $2q$ components are booleans. The booleans are thought of as q pairs of red and green lights, so that the jth red (respectively, green) light is considered to be on in a state iff the $(2j - 1)$th (respectively, $2j$th) boolean is $\overline{true}$. By adopting this presentation, we can state a number of developments only for LTSs, and have the versions for generalised Rabin automata as special cases. In particular, this is the case with the definitions of parametricity, data-independence, and data-independence with constants and predicates.

The main motivation for working with Rabin automata in this paper is that for any linear-time temporal logic formula, a corresponding deterministic Rabin automaton can be constructed [23, 21], which is significant for applicability of the theorems in Section 4. Although Muller automata [21] have the same property, they are less convenient for regarding as a subclass of LTSs.

Formally, a *generalised Rabin automaton (GRA)* is an LTS $\mathcal{R} = (D, A, J, \longrightarrow)$ such that $D = C \times ([\![Bool]\!]_{\{\}})^{2q}$ for some set C and some natural number q.

A run $\underline{d} = (d_0, d_1, \ldots)$ of $\mathcal{R}$ is *successful* iff there exists $j \in \{1, \ldots, q\}$ such that $\pi_{2j-1}(h_i) = \overline{true}$ for only finitely many i and $\pi_{2j}(h_i) = \overline{true}$ for infinitely many i, where $d_i = (c_i, h_i)$.

An infinite sequence $\underline{a}$ of elements of A is *accepted* by $\mathcal{R}$ iff, for each $d_0 \in J$, there exists a run of $\mathcal{R}$ on $\underline{a}$ from d_0 which is successful.

The reason why there should be a successful run from each initial state is that it should not matter which of the possible initial states the automaton begins from for the sequence to be accepted. For example, there may be more than one initial state because of arbitrary initial values of state variables, which should be irrelevant.

Satisfaction of a GRA by an LTS. Given an LTS $\mathcal{S}$ and a GRA $\mathcal{R}$ such that the set of states of $\mathcal{S}$ equals the set of transition labels of $\mathcal{R}$, we say that $\mathcal{S}$ *satisfies* $\mathcal{R}$ iff each run of $\mathcal{S}$ is accepted by $\mathcal{R}$.

3 Defining data-independence

We will define data-independence as a special case of parametricity. The idea is that, given a system (or specification) which uses a number of type variables and operations on them (which can be thought of as abstract data types), its semantics is a family of LTSs (or GRAs), which consists of an LTS (or GRA) for each instantiation of the type variables by sets and of the operation symbols by operations on those sets. If the system (or specification) uses the type variables and the operations in a polymorphic fashion, the family will be parametric. Data-independence will be the special case where the operations are only equality tests.

We will use the term 'signature' to mean a finite set Υ of type variables together with a finite set Γ of typed operation symbols, where the types in Γ do not contain free occurences of any type variables outside Υ.

Definition 1 (signature, instantiation).

(a) *We assume TermVars is an infinite set of names for term variables.*
A type map *is a partial map from TermVars to types, whose domain is finite.*
A signature *is an ordered pair* (Υ, Γ)*, where*
- Υ *is a finite subset of TypeVars, and*
- Γ *is a type map such that* $\mathit{Free}(\Gamma(x)) \subseteq \Upsilon$ *for each* $x \in \mathit{Domain}(\Gamma)$*.*

(b) *A* value map *is a map whose domain is a finite subset of TermVars.*
Given a type map Γ *and a set map* δ *such that* $\mathit{Free}(\Gamma(x)) \subseteq \mathit{Domain}(\delta)$ *for each* $x \in \mathit{Domain}(\Gamma)$*, we say that a value map* η *is* with respect to Γ and δ *iff*
- $\mathit{Domain}(\eta) = \mathit{Domain}(\Gamma)$*, and*
- $\eta[\![x]\!] \in [\![\Gamma(x)]\!]_\delta$ *for each* $x \in \mathit{Domain}(\Gamma)$*.*

An instantiation *of a signature* (Υ, Γ) *is an ordered pair* (δ, η) *such that*
- δ *is a set map whose domain is* Υ*, and*
- η *is a value map with respect to* Γ *and* δ*.* □

We can now define families of LTSs and GRAs. It is not necessary that they are indexed by all instantiations of a signature — for example, a system whose signature is $(\{X\}, [\mathit{leq} \mapsto ((X \times X) \to \mathit{Bool})])$ may make sense only for instantiations (δ, η) such that $\eta[\![\mathit{leq}]\!]$ is a partial ordering.

In any family of LTSs (or GRAs), there is a type of states and a type of transition labels which are shared by all members of the family. Intuitively, these are the types of states and transition labels of the program whose semantics is the family. In a family of GRAs which is not generated by a program but by a linear-time temporal logic formula [23, 21], the type of states may be a closed type, in which case its semantics is the same for each instantiation of the signature.

Although transition labels in an LTS and states (other than their lights) in a GRA are irrelevant for whether the former satisfies the latter, types of transition labels in families of LTSs and types of states in families of GRAs will play a role in relating members which correspond to different signature instantiations, and thus in defining parametric and data-independent families.

Definition 2 (families of LTSs and GRAs).

(a) A family of LTSs *is a tuple* $(\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ *such that*
- (Υ, Γ) *is a signature,*
- T *and* U *are types with* $\mathit{Free}(T) \subseteq \Upsilon$ *and* $\mathit{Free}(U) \subseteq \Upsilon$,
- $\mathcal{I}$ *is a class*[1] *of instantiations of* (Υ, Γ), *and*
- $\underline{\mathcal{S}}$ *consists of an LTS* $\mathcal{S}_{(\delta,\eta)}$ *whose set of states is* $[\![T]\!]_\delta$ *and whose set of transition labels is* $[\![U]\!]_\delta$, *for each* $(\delta, \eta) \in \mathcal{I}$.

(b) A family of GRAs *is a family of LTSs* $(\Upsilon, \Gamma, W, T, \mathcal{I}, \underline{\mathcal{R}})$ *such that* W *is of the form* $V \times \mathit{Bool}^{2q}$. □

Example 1. The following UNITY system RD_X removes adjacent duplicates from an input stream. It is parameterised by the type X of the values of the stream.

```
declare
  state      : Single+X;
  to_output  : Bool;
initially
  state=(1,()) /\ to_output=false;
assign
   [label:X+X] : to_output=false /\ fst(label)=1 /\
                 not (fst(state)=2 /\
                      eq(snd(state),snd(label))) ::
     state,to_output:=(2,snd(label)),true;
[] [label:X+X] : to_output=false /\ fst(label)=1 /\
                 fst(state)=2 /\
                 eq(snd(state),snd(label)) ::
     skip
[] [label:X+X] : to_output=true  /\ fst(label)=2 /\
                 eq(snd(label),snd(state)) ::
     to_output:=false;
end
```

In this example, for the sake of simplicity only, outputs are treated as inputs. Sum types are encoded as pairs. The function `fst` (respectively, `snd`) accesses to the first (respectively, second) component of a pair.

The semantics of RD_X is a family of LTSs $(\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ where:

- $\Upsilon = \{X\}$;
- $\Gamma = \{\mathsf{eq} \mapsto X \times X \to \mathit{Bool}\}$;
- $T = (\mathit{Single} + X) \times \mathit{Bool}$;
- $U = X + X$;

[1] We allow $\mathcal{I}$ to be a class rather than only a set because Definitions 7 and 8 will require $\mathcal{I}$ to be as large as the class of all sets. However, this foundational issue is not important in this paper.

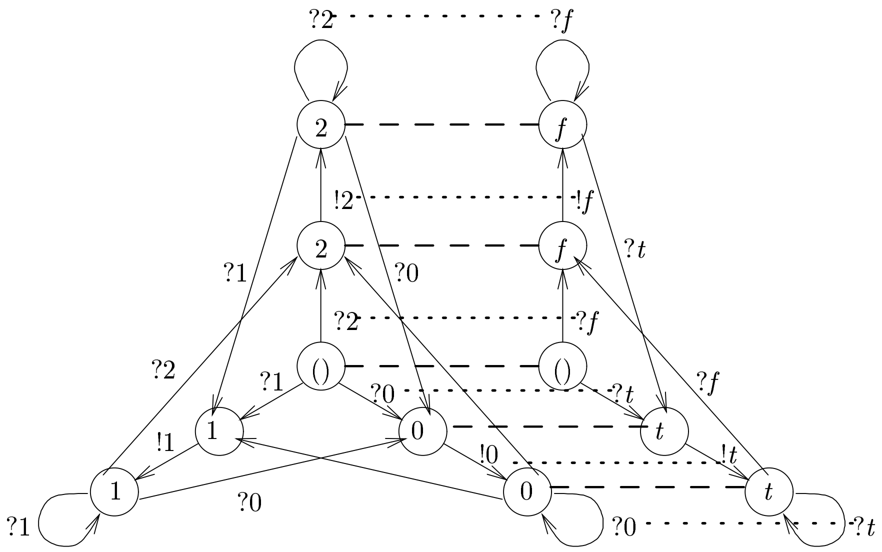


Fig. 1. Related LTSs

- $\mathcal{I}$ is the set of instantiations (δ, η) where δ is a set map whose domain is Υ, and η is a value map whose domain is $\{\texttt{eq}\}$ and such that $\eta[\![\texttt{eq}]\!]$ is the equality on $\delta[\![X]\!]$;
- $\underline{\mathcal{S}}$ consists of an LTS $\mathcal{S}_{(\delta,\eta)}$ whose set of states is $[\![T]\!]_\delta$ and whose set of transition labels is $[\![U]\!]_\delta$, for each $(\delta, \eta) \in \mathcal{I}$, and such that it is the operational semantics of the process RD_X where X is instantiated by the set $\delta[\![X]\!]$.

Let δ and δ' be set maps which map X to $\{0, 1, 2\}$ and $\{t, f\}$ respectively, and let η and η' be value maps which map e_X to equalities on $\{0, 1, 2\}$ and $\{t, f\}$ respectively. Figure 1 shows LTSs $\mathcal{S}_{(\delta,\eta)}$ and $\mathcal{S}_{(\delta',\eta')}$. □

A family of LTSs will be parametric iff, for any two instantiations (δ, η) and (δ', η') of its signature (Υ, Γ), and any relation map (ρ, δ, δ') on Υ which relates (δ, η) and (δ', η'), $[\![T]\!]_{(\rho,\delta,\delta')}$ is a certain kind of bisimulation with respect to $[\![U]\!]_{(\rho,\delta,\delta')}$ between the members of the family corresponding to (δ, η) and (δ', η'), where T is the type of states and U the type of transition labels.

The required kind of bisimulation is 'universal partial $[\![U]\!]_{(\rho,\delta,\delta')}$-bisimulation', and it is defined below. $[\![U]\!]_{(\rho,\delta,\delta')}$ acts as the relation between the sets of transition labels of the two LTSs. The bisimulation is called 'partial' because $[\![U]\!]_{(\rho,\delta,\delta')}$ may be a partial relation, and it is called 'universal' because it treats initial states and transition labels in a universal-quantifier manner.

As we remarked above, a family of LTSs is thought of as the semantics of a program. In order for the definition of parametricity (and, consequently, for the definition of data-independence and its extension) to make sense, a few further implicit assumptions need to be satisfied. Firstly, initial states need to be all those which satisfy a condition in the program, such as $pc = 0$ where pc is a program-counter variable. Secondly, transition labels need to consist of values of transition parameters. In the program, a transition is enabled iff its parameters satisfy a condition which may involve state variables. After a transition, the new values of state variables are given by expressions which may involve state variables (with their old values) and the transition parameters. There may be transitions in the program which may have the same source state and the same parameter values, but distinct target states.

The approach in this paper is therefore most readily applicable to languages similar to UNITY [1]. Such an example is given in Section 5. However, the applicability of the approach is considerably wider. For example, to apply it to a process algebra such as CSP [19], outputs can be modelled by special state variables, and inputs and nondeterministic choices of variables can both be modelled by transition parameters, where other special transition parameters may be used for distinguishing between visible and invisible transitions.

Definition 3 (related instantiations). *Suppose*

- (δ, η) *and* (δ', η') *are two instantiations of a signature* (Υ, Γ), *and*
- (ρ, δ, δ') *is a relation map.*

We say that (ρ, δ, δ') relates (δ, η) *and* (δ', η') *iff*

$$\forall x \in Domain(\Gamma).\eta[\![x]\!][\![\Gamma(x)]\!]_{(\rho,\delta,\delta')}\eta'[\![x]\!] \quad \square$$

Definition 4 (universal partial R-bisimulation). *Suppose*

- $\mathcal{S} = (A, B, I, \longrightarrow)$ *is an LTS,*
- $\mathcal{S}' = (A', B', I', \longrightarrow')$ *is an LTS,*
- P *is a relation between* A *and* A', *and*
- R *is a relation between* B *and* B'.

We say that P *is a* universal partial R-bisimulation between $\mathcal{S}$ and $\mathcal{S}'$ *iff*

(i) $\forall a, a'.aPa' \Rightarrow (a \in I \Leftrightarrow a' \in I')$,

(ii) $\forall a_1, a_1', b, b', a_2.(a_1 P a_1' \wedge a_1 \xrightarrow{b} a_2 \wedge bRb') \Rightarrow (\exists a_2'.a_1' \xrightarrow{b'}{}' a_2' \wedge a_2 P a_2')$, *and*

(iii) $\forall a_1, a_1', b, b', a_2'.(a_1 P a_1' \wedge a_1' \xrightarrow{b'}{}' a_2' \wedge bRb') \Rightarrow (\exists a_2.a_1 \xrightarrow{b} a_2 \wedge a_2 P a_2')$. $\square$

Definition 5 (related LTSs). *Suppose*

- T *and* U *are types,*
- (ρ, δ, δ') *is a relation map such that* $Free(T), Free(U) \subseteq Domain(\rho)$,
- $\mathcal{S}$ *is an LTS with set of states* $[\![T]\!]_\delta$ *and set of transition labels* $[\![U]\!]_\delta$, *and*
- $\mathcal{S}'$ *is an LTS with set of states* $[\![T]\!]_{\delta'}$ *and set of transition labels* $[\![U]\!]_{\delta'}$.

We say that (ρ, δ, δ') relates $\mathcal{S}$ *and* $\mathcal{S}'$ *iff* $[\![T]\!]_{(\rho,\delta,\delta')}$ *is a universal partial* $[\![U]\!]_{(\rho,\delta,\delta')}$*-bisimulation between* $\mathcal{S}$ *and* $\mathcal{S}'$. $\square$

Definition 6 (parametric family of LTSs). *A family of LTSs* $(\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ *is* parametric *iff, for any* (δ, η) *and* (δ', η') *from* $\mathcal{I}$*, and any relation map* (ρ, δ, δ')*, we have*

$$(\rho, \delta, \delta') \text{ relates } (\delta, \eta) \text{ and } (\delta', \eta') \;\Rightarrow\; (\rho, \delta, \delta') \text{ relates } \mathcal{S}_{(\delta,\eta)} \text{ and } \mathcal{S}_{(\delta',\eta')} \quad \square$$

A family of LTSs will be data-independent iff it is parametric and the operations in its signature (Υ, Γ) are only equality tests. More precisely, we define data-independence with respect to a subset Υ' of Υ, so that equality testing is allowed on type variables in Υ', but not allowed on those in $\Upsilon \setminus \Upsilon'$. The relevant instantiations of (Υ, Γ) are all those in which the equality-test symbols are instantiated by equalities.

Definition 7 (data-independence). *Suppose* $(\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ *is a family of LTSs, and* $\Upsilon' \subseteq \Upsilon$*.*

$(\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ *is* data-independent with Υ'-equalities *iff*

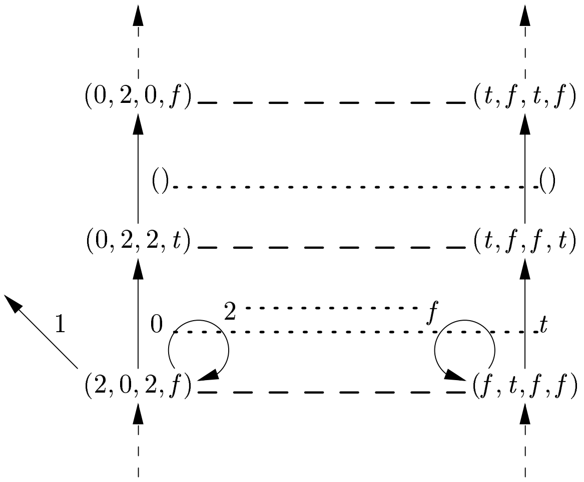
(i) $(\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ *is parametric,*
(ii) $\mathit{Domain}(\Gamma) = \{e_X \mid X \in \Upsilon'\}$*,*
(iii) $\Gamma(e_X) = ((X \times X) \rightarrow \mathit{Bool})$*, and*
(iv) $\mathcal{I}$ *consists of all instantiations* (δ, η) *of* (Υ, Γ) *such that* $\eta[\![e_X]\!]$ *is the equality on* $\delta[\![X]\!]$ *for each* $X \in \Upsilon'$*.* $\square$

Example 2. Consider the following UNITY system *Sys* which removes adjacent duplicates from an input stream, where the type of data is a type variable X.

```
declare
  state, prev_state, output : X;
  output_avl                : Bool;
initially
  output_avl=false;
assign
   [in:X] : output_avl=false /\ not eq(in,state) ::
     state,prev_state,output_avl:=in,state,true;
[] [in:X] : output_avl=false /\ eq(in,state) ::
     skip;
[] [in:Single] : output_avl=true ::
     output,output_avl:=state,false;
end
```

This system can be represented by a family of LTSs $(\{X\}, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ which is data-independent with $\{X\}$-equalities. Γ and $\mathcal{I}$ are given by Definition 7, $T = X \times X \times X \times \mathit{Bool}$, and $U = \mathit{Single} + X$. The first two components (`state` and `prev_state`) of a state store the two most recent outputs (partly to make possible a specification in linear-time temporal logic), whereas the remaining two components (`output` and `output_avl`) model an output channel. Transition labels of type X represent inputs.

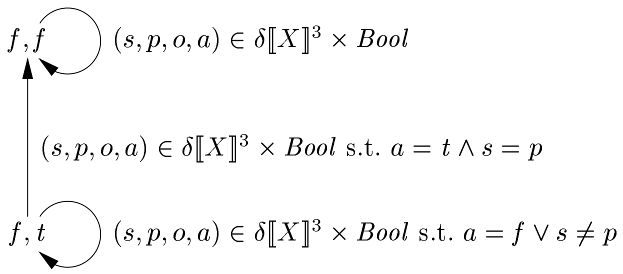
Let δ and δ' be set maps which map X to $\{0,1,2\}$ and $\{t,f\}$ respectively, and let η and η' be value maps which map e_X to equalities on $\{0,1,2\}$ and $\{t,f\}$ respectively. Let ρ be a relation map which maps X to $\{(0,t),(2,f)\}$. Since both $\rho[\![X]\!]$ and its inverse are injective, (ρ,δ,δ') relates (δ,η) and (δ',η'). Therefore, by Definitions 7, 6 and 5, $[\![T]\!]_{(\rho,\delta,\delta')}$ is a universal partial $[\![U]\!]_{(\rho,\delta,\delta')}$-bisimulation between $\mathcal{S}_{(\delta,\eta)}$ and $\mathcal{S}_{(\delta',\eta')}$. This is illustrated on parts of $\mathcal{S}_{(\delta,\eta)}$ and $\mathcal{S}_{(\delta',\eta')}$ in the following figure.



The following LTL specification *Spec* states that two consecutive outputs will be different.

$$\mathbf{G}(\texttt{output\_avl} = t \Rightarrow \neg(\texttt{state} = \texttt{prev\_state}))$$

Its semantics as a family of Rabin automata is given in the following figure.



$\square$

The following is an extension (i.e. a weakening) of data-independence. In addition to equality-test symbols, it allows constant symbols whose types are type variables, and predicate symbols which are functions from type variables to fixed non-empty finite types. This extension has been shown important in the recent work on model checking systems which are parameterised by their network structure [5].

Definition 8 (data-independence with constants and predicates). *Suppose* $(\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ *is a family of LTSs,* $\Upsilon' \subseteq \Upsilon$, $k : \Upsilon \to \mathbb{N}$ *and* $l : \Upsilon \to (\mathbb{N} \setminus \{0\})$.

$(\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ *is* data-independent with Υ'-equalities, k-constants and l-predicates *iff*

(i) $(\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ *is parametric,*
(ii) $Domain(\Gamma) = \{e_X \mid X \in \Upsilon'\} \cup \{s_{X,i} \mid i \in \{1, \dots, k(X)\}\} \cup \{p_X \mid X \in \Upsilon\}$,
(iii) $\Gamma(e_X) = ((X \times X) \to \mathit{Bool})$, $\Gamma(s_{X,i}) = X$ *and* $\Gamma(p_X) = (X \to \mathit{Enum}_{l(X)})$, *and*
(iv) $\mathcal{I}$ *consists of all instantiations* (δ, η) *of* (Υ, Γ) *such that* $\eta[\![e_X]\!]$ *is the equality on* $\delta[\![X]\!]$ *for each* $X \in \Upsilon'$. □

4 Reduction theorems

For any family of LTSs $\mathcal{F}$ and any type variable X from its signature, $\kappa_X(\mathcal{F})$ (respectively, $\lambda_X(\mathcal{F})$) will denote the supremum of the number of distinct values of type X in any reachable state (respectively, transition label). For these to make sense, it will be assumed that the type of states (respectively, transition labels) contains no function-type constructs.[2] If a supremum is over an empty set, its value is 0. A supremum may also be ∞.

Notation (*Comps*). For any type variable X, any type T which contains no function-type constructs, and any a which is an element of some $[\![T]\!]_\delta$, we write $\mathit{Comps}_{X,T}(a)$ for the set of all components of a which correspond to free occurences of X in T. This is defined by

$$\begin{aligned}
&\mathit{Comps}_{X,Y}(a) = \begin{cases} \{a\}, & \textit{if } Y = X \\ \{\}, & \textit{if } Y \neq X \end{cases} \\
&\mathit{Comps}_{X,T_1+\dots+T_n}((i,a)) = \mathit{Comps}_{X,T_i}(a) \\
&\mathit{Comps}_{X,T_1\times\dots\times T_n}((a_1,\dots,a_n)) = \mathit{Comps}_{X,T_1}(a_1) \cup \dots \cup \mathit{Comps}_{X,T_n}(a_n) \\
&\mathit{Comps}_{X,\mu Y.T}(a) = \begin{cases} \{\}, & \textit{if } Y = X \\ \mathit{Fold}(\mathit{Comps}_{X,T}(\cdot) \cup \bigcup \mathit{Comps}_{Y,T}(\cdot), a), & \textit{if } Y \neq X \end{cases}
\end{aligned}$$

where the definition of $\mathit{Comps}_{X,\mu Y.T}(a)$ in the case $Y \neq X$ is the recursion of the operator $\mathit{Comps}_{X,T}(\cdot) \cup \bigcup \mathit{Comps}_{Y,T}(\cdot)$ on the structure of a, recalling that any such a was built by finitely many iterations, by the definition of the denotational semantics of inductive types given in Section 2. □

Notation (κ). For any family of LTSs $\mathcal{F} = (\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ such that T contains no function-type constructs, and any $X \in \Upsilon$, let

$$\kappa_X(\mathcal{F}) = \sup\{|\mathit{Comps}_{X,T}(a)| \mid a \textit{ is a reachable state in some } \mathcal{S}_{(\delta,\eta)}\} \quad \square$$

[2] This could be relaxed, in particular to allow subtypes $T_1 \to T_2$ such that T_1 and T_2 do not contain any of the free occurences of X in the overall type.

Notation (λ). For any family of LTSs $\mathcal{F} = (\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ such that U contains no function-type constructs, and any $X \in \Upsilon$, let

$$\lambda_X(\mathcal{F}) = \sup\{|Comps_{X,U}(b)| \mid a_1 \xrightarrow{b} a_2 \textit{ and } a_1 \textit{ is a reachable state in some } \mathcal{S}_{(\delta,\eta)}\} \quad \square$$

Given a family of LTSs $\mathcal{F}$ (representing a system) and a family of GRAs $\mathcal{G}$ (representing a specification) which have the same signature (Υ, Γ) and are data-independent, and given two instantiations (δ, η) and (δ', η') of (Υ, Γ), the first reduction theorem states that satisfaction for (δ, η) and satisfaction for (δ', η') are equivalent provided, for any $X \in \Upsilon$, the sizes of the sets $\delta[\![X]\!]$ and $\delta'[\![X]\!]$ are either equal or no less than $\kappa_X(\mathcal{F}) + \lambda_X(\mathcal{F}) + \kappa_X(\mathcal{G})$. The types of states and transition labels are assumed to contain no function-type constructs, and the GRAs are assumed to be deterministic and complete. The theorem applies to the weakest kind of data-independence, where equality testing is allowed on all type variables.

If $\kappa_X(\mathcal{F}) + \lambda_X(\mathcal{F}) + \kappa_X(\mathcal{G})$ is finite, the theorem reduces the verification for all instantiations to the verification for a finite number of finite instantiations.

Theorem 1. *Suppose Υ is a finite subset of TypeVars.*

Suppose T, U and V are types such that Free$(T) \subseteq \Upsilon$, Free$(U) \subseteq \Upsilon$ and Free$(V) \subseteq \Upsilon$, and T, U and V contain no function-type constructs.

Suppose $\mathcal{F} = (\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ is a family of LTSs which is data-independent with Υ-equalities.

Suppose $\mathcal{G} = (\Upsilon, \Gamma, V \times Bool^{2q}, T, \mathcal{I}, \underline{\mathcal{R}})$ is a family of GRAs which is data-independent with Υ-equalities, and such that, for any $(\delta, \eta) \in \mathcal{I}$, $\mathcal{R}_{(\delta,\eta)}$ is deterministic and complete.

If $(\delta, \eta), (\delta', \eta') \in \mathcal{I}$ are such that, for any $X \in \Upsilon$, either $|\delta[\![X]\!]| = |\delta'[\![X]\!]|$ or

$$|\delta[\![X]\!]|, |\delta'[\![X]\!]| \geq \kappa_X(\mathcal{F}) + \lambda_X(\mathcal{F}) + \kappa_X(\mathcal{G})$$

then $\mathcal{S}_{(\delta,\eta)}$ satisfies $\mathcal{R}_{(\delta,\eta)}$ iff $\mathcal{S}_{(\delta',\eta')}$ satisfies $\mathcal{R}_{(\delta',\eta')}$. $\square$

The following theorem generalises the first one by being applicable to data-independence with constants and predicates. For simplicity, it assumes that the GRAs have unique initial states, and considers only instantiations which assign mutually distinct values to the constant symbols.

Theorem 2. *Suppose Υ is a finite subset of TypeVars, $k : \Upsilon \to \mathbb{N}$ and $l : \Upsilon \to (\mathbb{N} \setminus \{0\})$.*

Suppose T, U and V are types such that Free$(T) \subseteq \Upsilon$, Free$(U) \subseteq \Upsilon$ and Free$(V) \subseteq \Upsilon$, and T, U and V contain no function-type constructs.

Suppose $\mathcal{F} = (\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ is a family of LTSs which is data-independent with Υ-equalities, k-constants and l-predicates.

Suppose $\mathcal{G} = (\Upsilon, \Gamma, V \times Bool^{2q}, T, \mathcal{I}, \underline{\mathcal{R}})$ is a family of GRAs which is data-independent with Υ-equalities, k-constants and l-predicates, and such that, for any $(\delta, \eta) \in \mathcal{I}$, $\mathcal{R}_{(\delta,\eta)}$ has a unique initial state and is deterministic and complete.

If $(\delta, \eta), (\delta', \eta') \in \mathcal{I}$ are such that *(i)* *for any* $X \in \Upsilon$, $\eta[\![s_{X,1}]\!]$, ..., $\eta[\![s_{X,k(X)}]\!]$ *are mutually distinct, and* $\eta'[\![s_{X,1}]\!]$, ..., $\eta'[\![s_{X,k(X)}]\!]$ *are mutually distinct,*

(ii) *for any* $X \in \Upsilon$ *and any* $i \in \{1, \ldots, k(X)\}$, $\eta[\![p_X]\!](\eta[\![s_{X,i}]\!]) = \eta'[\![p_X]\!](\eta'[\![s_{X,i}]\!])$, *and*

(iii) *for any* $X \in \Upsilon$ *and any* $j \in \{1, \ldots, l(X)\}$, *either* $\nu_{\eta,X,j} = \nu_{\eta',X,j}$ *or*

$$\nu_{\eta,X,j}, \nu_{\eta',X,j} \geq \kappa_X(\mathcal{F}) + \lambda_X(\mathcal{F}) + \kappa_X(\mathcal{G})$$

where $\nu_{\theta,X,j} = |\theta[\![p_X]\!]^{-1}(\overline{enum_j}) \setminus \{\theta[\![s_{X,i}]\!] \mid i \in \{1, \ldots, k(X)\}\}|$,

then $\mathcal{S}_{(\delta,\eta)}$ *satisfies* $\mathcal{R}_{(\delta,\eta)}$ *iff* $\mathcal{S}_{(\delta',\eta')}$ *satisfies* $\mathcal{R}_{(\delta',\eta')}$. □

5 Applications of the approach

In this section, we show how the approach we presented can be applied to a programming language similar to UNITY [1] and to LTL [23].

5.1 Data independence in a UNITY-like language

A program in this language consists of three sections: a declare section which declares state variables, an initially section which specifies the initial condition satisfied by state variables, and a section assign which is a finite collection of guarded assignments. For the sake of simplicity, the language does not contain inductive or function type constructs. We assume the operators f range over a finite set.

$$\begin{aligned}
T &::= X \in \mathit{TypeVars} \mid T_1 + \cdots + T_n \mid T_1 \times \cdots \times T_n \\
\mathit{decl} &::= x_1{:}T; \ldots; x_n{:}T \\
\mathit{expr} &::= x \in \mathit{TermVars} \mid f(\mathit{expr}_1, \ldots, \mathit{expr}_n) \mid e_X(\mathit{expr}_1, \mathit{expr}_2) \\
\mathit{assign} &::= [y{:}T] : \mathit{expr} :: x_1, \ldots, x_n := \mathit{expr}_1, \ldots, \mathit{expr}_n \\
\mathit{assigns} &::= \mathit{assign}_1 \;[\!]\; \cdots \;[\!]\; \mathit{assign}_p \\
\mathit{progr} &::= \textsf{declare}\ \mathit{decl}\ \textsf{initially}\ \mathit{expr}\ \textsf{assign}\ \mathit{assigns}\ \textsf{end}
\end{aligned}$$

where e_X is the equality-test operator on the type X, i.e. its semantics is the equality on the semantics of X. In the guarded assignments, $[y{:}T]$ is a typed parameter, *expr* is a guard (a boolean expression), and $x_1, \ldots, x_n := \mathit{expr}_1, \ldots, \mathit{expr}_n$ is a simultaneous assignment.

Notation (signature, state type, action type). Suppose P is a program. Its signature (cf. Definition 1) is $(\Upsilon(P), \Gamma(P))$ where

- $\Upsilon(P)$ is the set of all type variables which occur free in P and
- $\Gamma(P)$ is the set of operators (f and e_X in the syntax) such that their types involve free type variables.

$\mathit{StateType}(P)$ is the product of the types of the state variables of P. $\mathit{ActType}(P)$ is the sum of the types of the parameters of P. □

We give an operational semantics to programs which are well typed.

Suppose P is a program which declares n state variables $x_1, \ldots, x_n$ respectively of type $T_1, \ldots, T_n$ (i.e. $\mathit{StateType}(P) = T_1 \times \cdots \times T_n$). Suppose (δ, η) is an instantiation of $(\Upsilon(P), \Gamma(P))$. The semantics with respect to (δ, η) and a tuple a of the expressions of P is defined by

- $[\![x_i]\!]_a = \pi_i(a)$ where x_i is one of the n state variables,
- $[\![y]\!]_a = \pi_{n+1}(a)$ where y is a parameter,
- $[\![f(\mathit{expr}_1, \ldots, \mathit{expr}_q)]\!]_a = \eta[\![f]\!]([\![\mathit{expr}_1]\!]_a, \ldots, [\![\mathit{expr}_q]\!]_a)$ if $f \in \mathit{Domain}(\Gamma(P))$, and
- $[\![f(\mathit{expr}_1, \ldots, \mathit{expr}_q)]\!]_a = [\![f]\!]([\![\mathit{expr}_1]\!]_a, \ldots, [\![\mathit{expr}_q]\!]_a)$ if $f \notin \mathit{Domain}(\Gamma(P))$, where $[\![f]\!]$ is the semantics of f.

We write $[\![P]\!]_{(\delta,\eta)}$ for the semantics of the program P with respect to (δ, η). It is an LTS $([\![T]\!]_\delta, [\![U]\!]_\delta, I_{(\delta,\eta)}, \longrightarrow_{(\delta,\eta)})$ where:

- $I_{(\delta,\eta)}$ is the set of initial states, i.e. states such that the initial condition of P is true.
$$I_{(\delta,\eta)} = \{(v_1, \ldots, v_n) \in [\![T_1]\!]_\delta \times \cdots \times [\![T_n]\!]_\delta \mid [\![\mathit{init}]\!]_{(v_1,\ldots,v_n)} = \overline{\mathit{true}}\}$$
where the expression *init* is the initial condition of P.
- For each guarded assignment assign_i of P such that its guard is true and the value of its parameter is u, there is a transition labelled by (i, u). More formally, the transition relation $\longrightarrow_{(\delta,\eta)}$ is defined by the following inference rule: for any guarded assignment of P, such that its parameter is of type U_i, its guard is the expression g, and its assignment is $x_{i_1}, \ldots, x_{i_q} := \mathit{expr}_1, \ldots, \mathit{expr}_q$,
$$\frac{[\![g]\!]_{(v_1,\ldots,v_n,u)} = \overline{\mathit{true}} \qquad \forall i \in \{i_1, \ldots, i_q\},\ [\![\mathit{expr}_i]\!]_{(v_1,\ldots,v_n,u)} = v'_i}{(v_1, \ldots, v_n) \xrightarrow{(i,u)}_{(\delta,\eta)} (v_1, \ldots, v_n)[i_1 \mapsto v'_{i_1}, \ldots, i_q \mapsto v'_{i_q}]}$$

We write $[\![P]\!]$ for the operational semantics of a program P. It is the family of LTSs $(\Upsilon(P), \Gamma(P), \mathit{StateType}(P), \mathit{ActType}(P), \mathcal{I}, \underline{\mathcal{S}})$ (cf. Definition 2) where $\mathcal{I}$ is the class of all instantiations (δ, η) of $(\Upsilon(P), \Gamma(P))$ such that each $\eta[\![e_X]\!]$ is the equality on $\delta[\![X]\!]$ (cf. Definition 1) and $\underline{\mathcal{S}}$ consists of an LTS $[\![P]\!]_{(\delta,\eta)}$ for each $(\delta, \eta) \in \mathcal{I}$.

We show that the semantics of any program is parametric (cf. Definition 6).

Proposition 2. *For any program P, the family of LTSs $[\![P]\!]$ is parametric.* □

We define a syntactical subclass of programs which are data-independent (cf. Definition 7).

Definition 9. *Suppose P is a program and Υ' is a subset of $\Upsilon(P)$. P is said to be* syntactically data-independent with Υ'-equalities *if and only if $\Gamma(P)$ only contains equality-test operators on type variables from Υ', i.e. $\Gamma(P)$ is equal to $\{e_X \mid X \in \Upsilon'\}$.* □

It is clear that syntactic data-independence is decidable.

Proposition 3. *Suppose P is a program and Υ' is a subset of $\Upsilon(P)$. If P is syntactically data-independent with Υ'-equalities, then the family of LTSs $[\![P]\!]$ is data-independent with Υ'-equalities.* □

We show how to compute an upper approximation of the suprema used by Theorem 1.

Definition 10. *For any type variable X and any type T which contains no inductive or function type constructs, $k_{X,T}$ is defined by*

$$\begin{aligned} k_{X,Y} &= \begin{cases} 1, & \textit{if } Y = X \\ 0, & \textit{if } Y \neq X \end{cases} \\ k_{X,T_1+\cdots+T_n} &= max\{k_{X,T_i} \mid 1 \leq i \leq n\} \\ k_{X,T_1\times\cdots\times T_n} &= \Sigma_{i=1}^{n} k_{X,T_i} \quad \square \end{aligned}$$

Proposition 4. *Suppose P is a program and $X \in \Upsilon(P)$.*

$$k_{X,StateType(P)} \geq \kappa_X([\![P]\!]) \textit{ and } k_{X,ActType(P)} \geq \lambda_X([\![P]\!]) \quad \square$$

5.2 Data-independence for LTL

In this subsection, we show how Theorem 1 of the approach we presented can be applied to Linear-time Temporal Logic.

We assume the operators p range over a finite set.

$$\begin{aligned} decl &::= x{:}T;\ decl \mid x{:}T; \\ prop &::= p(x_1, \ldots, x_q) \mid e_X(x_1, x_2) \\ spec &::= prop \mid \neg spec \mid spec_1 \wedge spec_2 \mid \mathbf{X} spec \mid spec_1 \mathbf{U} spec_2 \\ \varphi &::= decl\ spec \end{aligned}$$

where e_X is the equality-test operator on the type X, i.e. its semantics is the equality on the semantics of X.

Notation (signature, state type, action type). Suppose φ is an LTL formula. Its signature (cf. Definition 1) is $(\Upsilon(\varphi), \Gamma(\varphi))$ where

- $\Upsilon(\varphi)$ is the set of all type variables which occur free in φ and
- $\Gamma(\varphi)$ is the set of operators (p and e_X in the syntax) such that their types involve free type variables.

$ActType(\varphi)$ is the product of the types of the variables declared in φ. □

Suppose φ is an well typed LTL formula which declares n variables $x_1, \ldots, x_n$ respectively of type $T_1, \ldots, T_n$ (i.e. $ActType(\varphi) = T_1 \times \cdots \times T_n$). Suppose (δ, η) is an instantiation of $(\Upsilon(\varphi), \Gamma(\varphi))$. The semantics of atomic propositions with respect to (δ, η) and a tuple $b \in [\![ActType(\varphi)]\!]_\delta$ is defined by

– $[\![x_i]\!]_{(\delta,\eta),b} = \pi_i(b)$
– $[\![p(x_{i_1},\ldots,x_{i_q})]\!]_{(\delta,\eta),b} = \eta[\![p]\!]([\![x_{i_1}]\!]_{(\delta,\eta),b},\ldots,[\![x_{i_q}]\!]_{(\delta,\eta),b})$, if $p \in \mathit{Domain}(\Gamma(\varphi))$, and
– $[\![p(x_{i_1},\ldots,x_{i_q})]\!]_{(\delta,\eta),b} = [\![p]\!]([\![x_{i_1}]\!]_{(\delta,\eta),b},\ldots,[\![x_{i_q}]\!]_{(\delta,\eta),b})$, if $p \notin \mathit{Domain}(\Gamma(\varphi))$ where $[\![p]\!]$ is the semantics of p.

We write $[\![\varphi]\!]_{(\delta,\eta)}$ for the semantics of the LTL formula φ with respect to (δ,η). It is an LTS (Rabin automaton) $([\![\mathit{StateType}]\!]_{\{\}} \times ([\![\mathit{Bool}]\!]_{\{\}})^{2q}, [\![\mathit{ActType}(\varphi)]\!]_\delta, I_{(\delta,\eta)}, \longrightarrow_{(\delta,\eta)})$ which recognizes the same language than φ, where *StateType* is the type of the states of the automaton (Note that *StateType* does not contain any free type variable). To construct this LTS, we apply the method described in [23]. Such constructed LTS accepts words over $2^{\mathit{Prop}_\varphi}$ where Prop_φ is the set of atomic propositions in the LTL formula φ. But, to each $b \in [\![\mathit{ActType}(\varphi)]\!]_\delta$, we can associate the set $F_{(\delta,\eta)}(b)$ of atomic propositions it satisfies, where $F_{(\delta,\eta)} : [\![\mathit{ActType}(\varphi)]\!]_\delta \to 2^{\mathit{Prop}_\varphi}$ is the function such that: $p(x_{i_1},\ldots,x_{i_q}) \in F_{(\delta,\eta)}(b)$ if and only if

$$[\![p(x_{i_1},\ldots,x_{i_q})]\!]_{(\delta,\eta),b} = \overline{\mathit{true}}.$$

Moreover, the obtained automaton is a Büchi automaton that is generally nondeterministic. But we can thus transform it into a deterministic Rabin automaton.

We write $[\![\varphi]\!]$ for the semantics of the parameterised LTL formula φ. It is the family of LTSs $(\Upsilon(\varphi), \Gamma(\varphi), \mathit{StateType} \times \mathit{Bool}^{2q}, \mathit{ActType}(\varphi), \mathcal{I}, \underline{\mathcal{R}})$ where $\mathcal{I}$ is the class of all instantiations (δ,η) of $(\Upsilon(\varphi),\Gamma(\varphi))$ such that each $\eta[\![e_X]\!]$ is the equality on $\delta[\![X]\!]$ (cf. Definition 1) and $\underline{\mathcal{R}}$ consists of the LTS $[\![\varphi]\!]_{(\delta,\eta)}$ for each $(\delta,\eta) \in \mathcal{I}$.

We show that the semantics of any LTL formula is parametric (cf. Definition 6).

Proposition 5. *For any LTL formula φ, the family of LTSs $[\![\varphi]\!]$ is parametric.* □

We define a syntactical subclass of LTL formulae which are data-independent (cf. Definition 7).

Definition 11. *Suppose φ is an LTL formula and Υ' is a subset of $\Upsilon(\varphi)$. φ is said to be* syntactically data-independent with Υ'-equalities *if and only if $\Gamma(\varphi)$ only contains equality-test operators on type variables from Υ'.* □

It is clear that syntactic data-independence is decidable.

Proposition 6. *Suppose φ is an LTL formula and Υ' is a subset of $\Upsilon(\varphi)$. If φ is syntactically data-independent with Υ'-equalities, then the family of LTSs $[\![\varphi]\!]$ is data-independent with Υ'-equalities.* □

We show that the supremum used by Theorem 1 is always equal to 0 for an LTL formula.

Proposition 7. *Suppose φ is an LTL formula and $X \in \Upsilon(\varphi)$.*

$$\kappa_X([\![\varphi]\!]) = 0 \quad \square$$

5.3 A data-independent system and its specification

Consider the UNITY system *Sys* and its specification *Spec* described in example 2. Definition 10 and Proposition 4 gives:

$$\begin{aligned} k_{X,X\times X\times X\times Bool} &= 3 \geq \kappa_X([\![Sys]\!]) \\ k_{X,X+X+Single} &= 1 \geq \lambda_X([\![Sys]\!]) \end{aligned}$$

And Proposition 7 gives $\kappa_X([\![Spec]\!]) = 0$.

By applying Theorem 1, if the system *Sys* satisfies the specification *Spec* for an instantiation of the type variable X by a type of size greater than or equal to 4, then *Sys* satisfies *Spec* for any instantiation of X by a type of size greater than or equal to 4.

6 Conclusions and future work

We have presented the first semantic definition of data-independence which allows equality testing, and its extension which allows constant symbols and predicate symbols. Both are special cases of a definition of when a family of labelled transition systems (or generalised Rabin automata) is parametric. This provides a unified approach to data-independence and its extensions.

The paper also contains two theorems which, given a system and a specification which are data-independent, enable the verification for all instantiations of the type variables (and of the constant symbols and the predicate symbols, in the case of the extension) to be reduced to the verification for a finite number of finite instantiations.

The applicability of the approach to particular formalisms has been illustrated by a programming language similar to UNITY [1] and by linear-time temporal logic [23].

One direction for future work is considering other extensions (i.e. weakenings) of data-independence. Another is considering cases where the types of states and transition labels may contain function-type constructs — this will be addressed in [16], which is based on earlier work presented in [14]. A third direction is tree automata [21], in order to allow specifications given by formulae in branching-time temporal logics.

Acknowledgements

We are grateful to Antti Valmari for pointing out a shortcoming in a previous version of the paper, to Michael Goldsmith, Bengt Jonsson, David Naumann and Uday Reddy for useful discussions, to Philippa Broadfoot and Bill Roscoe for proof-reading, and to anonymous reviewers for their comments.

References

1. K.M. Chandy and J. Misra. *Parallel Program Design, A Foundation*. Addison-Wesley, 1988.
2. E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting Symmetry in Temporal Logic Model Checking. In [6], pages 77–104, 1996. Kluwer.
3. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
4. P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction of Approximation of Fixed Points. In *Proceedings of the 4th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, 1977. ACM.
5. S.J. Creese and A.W. Roscoe. Formal Verification of Arbitrary Network Topologies. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Volume II, pages 1033–1039, 1999. CSREA Press.
6. E.A. Emerson (editor). Formal Methods in System Design 9 (1–2) (Special Issue on Symmetry in Automatic Verification). Kluwer, 1996.
7. R. Hojati and R.K. Brayton. Automatic Datapath Abstraction In Hardware Systems. In P. Wolper, editor, *Proceedings of the 7th International Conference On Computer Aided Verification*, volume 939 of *Lecture Notes in Computer Science*, pages 98–113, 1995. Springer Verlag.
8. R. Hojati, D.L. Dill, and R.K. Brayton. Verifying linear temporal properties of data insensitive controllers using finite instantiations. In *Proceedings of the 15th International Conference on Computer Hardware Description Languages and their Applications*, 1997.
9. C.N. Ip and D.L. Dill. Better verification through symmetry. In [6], pages 41–75, 1996.
10. B. Jonsson and J. Parrow. Deciding Bisimulation Equivalences for a Class of Non-Finite-State Programs. *Information and Computation*, 107(2):272–302, 1993.
11. Y. Kinoshita, P.W. O'Hearn, A.J. Power, and M. Takeyama. An Axiomatic Approach to Binary Logical Relations with Applications to Data Refinement. In *Proceedings of the 3rd International Symposium on Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 191–212, 1997. Springer Verlag.
12. R.S. Lazić. *A Semantic Study of Data Independence with Applications to Model Checking*. D.Phil. thesis, Oxford University Computing Laboratory, 1999.
13. R.S. Lazić and D. Nowak. A Unifying Approach to Data-independence. In *Proceedings of the 11th International Conference on Concurrency Theory, Lecture Notes in Computer Science*, 2000. Springer Verlag.
14. R.S. Lazić and A.W. Roscoe. Verifying Determinism of Concurrent Systems Which Use Unbounded Arrays. In *Proceedings of the 3rd International Workshop on Verification of Infinite State Systems*, Report TUM-I9825, Technical University of Munich, 1998. A fuller version is the Technical Report PRG-TR-2-98, Oxford University Computing Laboratory.
15. R.S. Lazić and A.W. Roscoe. Data Independence with Generalised Predicate Symbols. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Volume I, pages 319–325, 1999. CSREA Press.
16. R.S. Lazić and A.W. Roscoe. Model Checking of Systems Which Use Arrays Data-Independently. In preparation, 2000.

17. K.S. Namjoshi and R.P. Kurshan. Syntactic Program Transformations for Automatic Abstraction. In *Proceedings of the 12th International Conference On Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, 2000. Springer Verlag.
18. J.C. Reynolds. Types, Abstraction and Parametric Polymorphism. In *Proceedings of the IFIP 9th World Congress*, pages 513–523, 1983. Elsevier Science Publishers B. V. (Norht-Holland).
19. A.W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.
20. A.W. Roscoe and P. J. Broadfoot. Proving security protocols with model checkers by data independence techniques. In *Journal of Computer Security, Special Issue CSFW11*, pages 147–190, 1999. IOS Press.
21. W. Thomas. Automata on Infinite Objects. In [22], chapter 4, pages 133–191. Elsevier Science Publishers B. V., 1990.
22. J. van Leeuwen. *Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics.* Elsevier Science Publishers B. V., 1990.
23. M.Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science*, pages 332–345, 1986. IEEE Computer Society Press.
24. P. Wolper. Expressing Interesting Properties of Programs in Propositional Temporal Logic. In *Proceedings of the 13th Annual ACM Symposium on Principles of Programming Languages*, pages 184–193, 1986. ACM.

A Proofs

Notation (*Compose***).** For any family of LTSs $\mathcal{F} = (\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ and family of GRAs $\mathcal{G} = (\Upsilon, \Gamma, V \times Bool^{2q}, T, \mathcal{I}, \underline{\mathcal{R}})$, let $Compose(\mathcal{F}, \mathcal{G})$ be the family of LTSs $(\Upsilon, \Gamma, T \times V \times Bool^{2q}, U, \mathcal{I}, \underline{\mathcal{P}})$ such that, for each $(\delta, \eta) \in \mathcal{I}$,

- (a, c, h) is an initial state of $\mathcal{P}_{(\delta,\eta)}$ iff a is an initial state of $\mathcal{S}_{(\delta,\eta)}$ and (c, h) is an initial state of $\mathcal{R}_{(\delta,\eta)}$;
- $(a_1, c_1, h_1) \xrightarrow{b} (a_2, c_2, h_2)$ in $\mathcal{P}_{(\delta,\eta)}$ iff $a_1 \xrightarrow{b} a_2$ in $\mathcal{S}_{(\delta,\eta)}$ and $(c_1, h_1) \xrightarrow{a_1} (c_2, h_2)$ in $\mathcal{R}_{(\delta,\eta)}$. □

Lemma 1. *Suppose $\mathcal{F} = (\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ and $\mathcal{G} = (\Upsilon, \Gamma, V \times Bool^{2q}, T, \mathcal{I}, \underline{\mathcal{R}})$ are a family of LTSs and a family of GRAs (respectively).*
Let $(\Upsilon, \Gamma, T \times V \times Bool^{2q}, U, \mathcal{I}, \underline{\mathcal{P}})$ be the family of LTSs $Compose(\mathcal{F}, \mathcal{G})$.

(i) For each $(\delta, \eta) \in \mathcal{I}$ such that $\mathcal{R}_{(\delta,\eta)}$ is deterministic and complete, $\mathcal{S}_{(\delta,\eta)}$ satisfies $\mathcal{R}_{(\delta,\eta)}$ iff, for each run $((a_0, c_0, h_0), (a_1, c_1, h_1), \ldots)$ of $\mathcal{P}_{(\delta,\eta)}$, there exists $j \in \{1, \ldots, q\}$ such that $\pi_{2j-1}(h_i) = \overline{true}$ for only finitely many i and $\pi_{2j}(h_i) = \overline{true}$ for infinitely many i.
(ii) If $\Upsilon' \subseteq \Upsilon$, and $\mathcal{F}$ and $\mathcal{G}$ are data-independent with Υ'-equalities, then $Compose(\mathcal{F}, \mathcal{G})$ is data-independent with Υ'-equalities.
(iii) If T, U and V contain no function-type constructs, then for any $X \in \Upsilon$,

$$\kappa_X(\mathcal{F}) + \lambda_X(\mathcal{F}) + \kappa_X(\mathcal{G}) \geq$$
$$\kappa_X(Compose(\mathcal{F}, \mathcal{G})) + \lambda_X(Compose(\mathcal{F}, \mathcal{G}))$$

Proof. Straightforward. $\square$

Notation (*Unlabel*). For any family of LTSs $\mathcal{F} = (\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$, let *Unlabel*$(\mathcal{F})$ be the family of LTSs $(\Upsilon, \Gamma, T \times U, \mathit{Single}, \mathcal{I}, \underline{\mathcal{U}})$ such that, for each $(\delta, \eta) \in \mathcal{I}$,

- (a_1, b) is an initial state of $\mathcal{U}_{(\delta,\eta)}$ iff a_1 is an initial state of $\mathcal{S}_{(\delta,\eta)}$ and $a_1 \xrightarrow{b} a_2$ in $\mathcal{S}_{(\delta,\eta)}$ for some a_2;
- $(a_1, b_1) \xrightarrow{()} (a_2, b_2)$ in $\mathcal{U}_{(\delta,\eta)}$ iff $a_1 \xrightarrow{b_1} a_2$ in $\mathcal{S}_{(\delta,\eta)}$ and $a_2 \xrightarrow{b_2} a_3$ in $\mathcal{S}_{(\delta,\eta)}$ for some a_3. $\square$

Notation (*Bij*). For any type T and any set map δ such that $\mathit{Free}(T) \subseteq \mathit{Domain}(\delta)$, we define

$$\mathit{Bij}_{T,\delta} = \{[\![T]\!]_{(\rho,\delta,\delta)} \mid \forall X \in \mathit{Domain}(\delta).\rho[\![X]\!] \textit{ is a bijection on } \delta[\![X]\!]\} \quad \square$$

Lemma 2. *Suppose $\mathcal{F} = (\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ is a family of LTSs which is data-independent with Υ-equalities.*

Let $(\Upsilon, \Gamma, T \times U, \mathit{Single}, \mathcal{I}, \underline{\mathcal{U}})$ be the family of LTSs Unlabel$(\mathcal{F})$.

For each $(\delta, \eta) \in \mathcal{I}$, $\mathit{Bij}_{T\times U,\delta}$ is a group of automorphisms of $\mathcal{U}_{(\delta,\eta)}$.

Proof. It suffices to prove that, for each $(\delta, \eta) \in \mathcal{I}$ and each relation map (ρ, δ, δ) such that $\rho[\![X]\!]$ is a bijection on $\delta[\![X]\!]$ for all X,

(i) if (a_1, b) is an initial state of $\mathcal{U}_{(\delta,\eta)}$ and $(a_1, b)[\![T \times U]\!]_{(\rho,\delta,\delta)}(a_1', b')$, then (a_1', b') is an initial state of $\mathcal{U}_{(\delta,\eta)}$;

(ii) if $(a_1, b_1) \xrightarrow{()} (a_2, b_2)$ in $\mathcal{U}_{(\delta,\eta)}$, $(a_1, b_1)[\![T \times U]\!]_{(\rho,\delta,\delta)}(a_1', b_1')$ and $(a_2, b_2)[\![T \times U]\!]_{(\rho,\delta,\delta)}(a_2', b_2')$, then $(a_1', b_1') \xrightarrow{()} (a_2', b_2')$ in $\mathcal{U}_{(\delta,\eta)}$.

Since $\mathcal{F}$ is data-independent with Υ-equalities and (ρ, δ, δ) relates (δ, η) and (δ, η), we have that $[\![T]\!]_{(\rho,\delta,\delta)}$ is a universal partial $[\![U]\!]_{(\rho,\delta,\delta)}$-bisimulation between $\mathcal{S}_{(\delta,\eta)}$ and $\mathcal{S}_{(\delta,\eta)}$.

For (i), a_1 is an initial state of $\mathcal{S}_{(\delta,\eta)}$ and $a_1[\![T]\!]_{(\rho,\delta,\delta)}a_1'$, so a_1' is an initial state of $\mathcal{S}_{(\delta,\eta)}$. Let a_2 be such that $a_1 \xrightarrow{b} a_2$. Since $b[\![U]\!]_{(\rho,\delta,\delta)}b'$, there exists a_2' such that $a_1' \xrightarrow{b'} a_2'$ and $a_2[\![T]\!]_{(\rho,\delta,\delta)}a_2'$. Therefore, (a_1', b') is an initial state of $\mathcal{U}_{(\delta,\eta)}$.

For (ii), $a_1[\![T]\!]_{(\rho,\delta,\delta)}a_1'$, $a_1 \xrightarrow{b_1} a_2$ and $b_1[\![U]\!]_{(\rho,\delta,\delta)}b_1'$, so $a_1' \xrightarrow{b_1'} a_2'$, since $a_2[\![T]\!]_{(\rho,\delta,\delta)}a_2'$ and $[\![T]\!]_{(\rho,\delta,\delta)}$ is a bijection on $[\![T]\!]_\delta$. Let a_3 be such that $a_2 \xrightarrow{b_2} a_3$. Since $b_2[\![U]\!]_{(\rho,\delta,\delta)}b_2'$, there exists a_3' such that $a_2' \xrightarrow{b_2'} a_3'$ and $a_3[\![T]\!]_{(\rho,\delta,\delta)}a_3'$. Therefore, $(a_1', b_1') \xrightarrow{()} (a_2', b_2')$ in $\mathcal{U}_{(\delta,\eta)}$. $\square$

Notation (*Reach*). For any LTS $\mathcal{S} = (A, B, I, \longrightarrow)$, we write *Reach*$(\mathcal{S})$ for the LTS $(A', B, I, \longrightarrow')$ which is the restriction of $\mathcal{S}$ to the set A' of all its reachable states. $\square$

Lemma 3. *Suppose $\mathcal{F} = (\Upsilon, \Gamma, T, U, \mathcal{I}, \underline{\mathcal{S}})$ is a family of LTSs which is data-independent with Υ-equalities, and T and U contain no function-type constructs.*

Let $(\Upsilon, \Gamma, T \times U, Single, \mathcal{I}, \underline{\mathcal{U}})$ be the family of LTSs Unlabel$(\mathcal{F})$.

For any $(\delta, \eta) \in \mathcal{I}$, let $\mathcal{Q}_\delta$ be the LTS which is the quotient of $\mathcal{U}_{(\delta,\eta)}$ by $Bij_{T\times U,\delta}$.

For any $(\delta, \eta), (\delta', \eta') \in \mathcal{I}$ and any $C \subseteq [\![T \times U]\!]_\delta$, let

$$\phi_{\delta,\delta'}(C) = \left\{ (a', b') \;\middle|\; \begin{array}{l} (a, b) \in C \;\wedge\; (a, b)[\![T \times U]\!]_{(\rho,\delta,\delta')}(a', b') \;\wedge \\ \forall X \in \Upsilon . \rho[\![X]\!] \textit{ is an injective partial function} \end{array} \right\}$$

If $(\delta, \eta), (\delta', \eta') \in \mathcal{I}$ are such that, for any $X \in \Upsilon$, either $|\delta[\![X]\!]| = |\delta'[\![X]\!]|$ or

$$|\delta[\![X]\!]|, |\delta'[\![X]\!]| \geq \kappa_X(\mathcal{F}) + \lambda_X(\mathcal{F})$$

then $\phi_{\delta,\delta'}$ is an isomorphism between Reach$(\mathcal{Q}_\delta)$ and Reach$(\mathcal{Q}_{\delta'})$.

Proof. It suffices to prove that, for any (δ, η) and (δ', η') as in the conclusion of the lemma,

(i) for any reachable state C of $\mathcal{Q}_\delta$, $\phi_{\delta,\delta'}(C)$ is a state of $\mathcal{Q}_{\delta'}$;
(ii) for any initial state C of $\mathcal{Q}_\delta$, $\phi_{\delta,\delta'}(C)$ is an initial state of $\mathcal{Q}_{\delta'}$;
(iii) whenever $C_1 \xrightarrow{()} C_2$ and C_1 is reachable in $\mathcal{Q}_\delta$, then $\phi_{\delta,\delta'}(C_1) \xrightarrow{()} \phi_{\delta,\delta'}(C_2)$ in $\mathcal{Q}_{\delta'}$;
(iv) for any reachable state C of $\mathcal{Q}_\delta$, $\phi_{\delta',\delta}(\phi_{\delta,\delta'}(C)) = C$.

For any $(\delta, \eta) \in \mathcal{I}$, we have that C is a state of $\mathcal{Q}_\delta$ iff

$$\begin{array}{l} \exists (a, b) \in [\![T \times U]\!]_\delta . \\ C = \left\{ (a^\dagger, b^\dagger) \;\middle|\; \begin{array}{l} (a, b)[\![T \times U]\!]_{(\rho,\delta,\delta)}(a^\dagger, b^\dagger) \;\wedge \\ \forall X \in \Upsilon . \rho[\![X]\!] \textit{ is a bijection on } \delta[\![X]\!] \end{array} \right\} \end{array} \tag{1}$$

For (i), any $(a, b) \in C$ is a reachable state of $\mathcal{U}_{(\delta,\eta)}$, and so $|Comps_{X,T}(a)| \leq \kappa_X(\mathcal{F})$ and $|Comps_{X,U}(b)| \leq \lambda_X(\mathcal{F})$ for all $X \in \Upsilon$. It follows that

$$|Comps_{X,T\times U}(a, b)| \leq |\delta'[\![X]\!]|$$

for all $X \in \Upsilon$. Consequently, by (1), $\phi_{\delta,\delta'}(C)$ is a state of $\mathcal{Q}_{\delta'}$.

Since $\mathcal{F}$ is data-independent with Υ-equalities, for any $(\delta, \eta), (\delta', \eta') \in \mathcal{I}$ and any relation map (ρ, δ, δ') such that $\rho[\![X]\!]$ is an injective partial function for all $X \in \Upsilon$, $[\![T]\!]_{(\rho,\delta,\delta')}$ is a universal partial $[\![U]\!]_{(\rho,\delta,\delta')}$-bisimulation between $\mathcal{S}_{(\delta,\eta)}$ and $\mathcal{S}_{(\delta',\eta')}$.

For (ii), let $(a_1, b) \in C$. Then a_1 is an initial state of $\mathcal{S}_{(\delta,\eta)}$ and there exists a_2 such that $a_1 \xrightarrow{b} a_2$. Let (ρ, δ, δ') be a relation map such that, for all $X \in \Upsilon$, $\rho[\![X]\!]$ is an injective partial function and $Comps_{X,T\times U}(a_1, b) \subseteq Domain(\rho[\![X]\!])$. Let (a'_1, b') be such that $(a_1, b)[\![T \times U]\!]_{(\rho,\delta,\delta')}(a'_1, b')$. Since $[\![T]\!]_{(\rho,\delta,\delta')}$ is a universal partial $[\![U]\!]_{(\rho,\delta,\delta')}$-bisimulation between $\mathcal{S}_{(\delta,\eta)}$ and $\mathcal{S}_{(\delta',\eta')}$, a'_1 is an initial state of $\mathcal{S}_{(\delta',\eta')}$ and there exists a'_2 such that $a'_1 \xrightarrow{b'} a'_2$ and $a_2[\![T]\!]_{(\rho,\delta,\delta')}a'_2$. Therefore, (a'_1, b') is an initial state of $\mathcal{U}_{(\delta',\eta')}$, so $\phi_{\delta,\delta'}(C)$ is an initial state of $\mathcal{Q}_{\delta'}$.

For (iii), let $(a_1, b_1) \in C_1$ and $(a_2, b_2) \in C_2$ be such that $(a_1, b_1) \xrightarrow{()} (a_2, b_2)$ in $\mathcal{U}_{(\delta,\eta)}$. We also have that (a_1, b_1) is reachable in $\mathcal{U}_{(\delta,\eta)}$.

Let $(\rho_1, \delta, \delta')$ be a relation map such that, for all $X \in \Upsilon$, $\rho_1[\![X]\!]$ is an injective partial function and $\mathit{Comps}_{X,T\times U}(a_1, b_1) \subseteq \mathit{Domain}(\rho_1[\![X]\!])$. Let (a'_1, b'_1) be such that $(a_1, b_1)[\![T \times U]\!]_{(\rho_1,\delta,\delta')}(a'_1, b'_1)$. Since $[\![T]\!]_{(\rho_1,\delta,\delta')}$ is a universal partial $[\![U]\!]_{(\rho_1,\delta,\delta')}$-bisimulation between $\mathcal{S}_{(\delta,\eta)}$ and $\mathcal{S}_{(\delta',\eta')}$, there exists a'_2 such that $a'_1 \xrightarrow{b'_1} a'_2$ in $\mathcal{S}_{(\delta',\eta')}$ and $a_2[\![T]\!]_{(\rho_1,\delta,\delta')}a'_2$.

Now let $(\rho_2, \delta, \delta')$ be a relation map such that, for all $X \in \Upsilon$, $\rho_2[\![X]\!]$ is an injective partial function, $\mathit{Comps}_{X,T\times U}(a_2, b_2) \subseteq \mathit{Domain}(\rho_2[\![X]\!])$, and $\rho_1[\![X]\!]$ and $\rho_2[\![X]\!]$ agree on $\mathit{Comps}_{X,T}(a_2)$. We then have that $a_2[\![T]\!]_{(\rho_2,\delta,\delta')}a'_2$. Let a_3 be such that $a_2 \xrightarrow{b_2} a_3$ in $\mathcal{S}_{(\delta,\eta)}$, and let b'_2 be such that $b_2[\![U]\!]_{(\rho_2,\delta,\delta')}b'_2$. Since $[\![T]\!]_{(\rho_2,\delta,\delta')}$ is a universal partial $[\![U]\!]_{(\rho_2,\delta,\delta')}$-bisimulation between $\mathcal{S}_{(\delta,\eta)}$ and $\mathcal{S}_{(\delta',\eta')}$, there exists a'_3 such that $a'_2 \xrightarrow{b'_2} a'_3$ in $\mathcal{S}_{(\delta',\eta')}$ and $a_3[\![T]\!]_{(\rho_2,\delta,\delta')}a'_3$.

Therefore, $(a'_1, b'_1) \xrightarrow{()} (a'_2, b'_2)$ in $\mathcal{U}_{(\delta',eta')}$, so $\phi_{\delta,\delta'}(C_1) \xrightarrow{()} \phi_{\delta,\delta'}(C_2)$ in $\mathcal{Q}_{\delta'}$.

(iv) is straightforward to prove using (1). □

Proof (Theorem 1). By Lemma 1, by applying Lemmas 2 and 3 to $\mathit{Compose}(\mathcal{F}, \mathcal{G})$, and by applying Theorem 4.1 from [2] with the formula

$$A\,(((FG\neg\pi_1(h)) \wedge (GF\pi_2(h))) \;\vee\; \cdots \;\vee\; ((FG\neg\pi_{2q-1}(h)) \wedge (GF\pi_{2q}(h))))$$

(The differences between the frameworks in this paper and [2] are not difficult to overcome.) □

Proof (Theorem 2). As the first alternative, suppose $X \in \Upsilon$ is such that $l(X) > 1$, and let:

- $Y_1, \ldots, Y_{l(X)}$ be fresh members of *TypeVars*;
- $\Upsilon^\dagger = (\Upsilon \setminus \{X\}) \cup \{Y_1, \ldots, Y_{l(X)}\}$;
- $k^\dagger : \Upsilon^\dagger \to \mathbb{N}$ be defined by $k^\dagger(X') = k(X')$ for $X' \in \Upsilon \setminus \{X\}$, and
$$k^\dagger(Y_j) = |\{i \mid \eta[\![p_X]\!](\eta[\![s_{X,i}]\!]) = \overline{\mathit{enum}_j}\}|$$
- $l^\dagger : \Upsilon^\dagger \to (\mathbb{N} \setminus \{0\})$ be defined by $l^\dagger(X') = l(X')$ for $X' \in \Upsilon \setminus \{X\}$, and $l^\dagger(Y_j) = 1$;
- $T^\dagger = T[(\sum Y_j)/X]$, $U^\dagger = U[(\sum Y_j)/X]$, and $V^\dagger = V[(\sum Y_j)/X]$;
- $\Gamma^\dagger$ be defined as in Definition 8 with $\Upsilon^\dagger$-equalities, $k^\dagger$-constants and $l^\dagger$-predicates, where e_{Y_j} and p_{Y_j} are fresh members of *TermVars*, and
$$\{s_{Y_j,1}, \ldots, s_{Y_j,k^\dagger(Y_j)}\} = \{s_{X,i} \mid \eta[\![p_X]\!](\eta[\![s_{X,i}]\!]) = \overline{\mathit{enum}_j}\}$$
- $\mathcal{I}^\dagger$ be defined as in Definition 8 with $\Upsilon^\dagger$-equalities, $k^\dagger$-constants and $l^\dagger$-predicates;
- for any $(\epsilon, \zeta) \in \mathcal{I}$, $(\pi(\epsilon), \pi(\zeta)) \in \mathcal{I}^\dagger$ be defined by:
  - $\pi(\epsilon)[\![X']\!] = \epsilon[\![X']\!]$ for $X' \in \Upsilon \setminus \{X\}$, and $\pi(\epsilon)[\![Y_j]\!] = \zeta[\![p_X]\!]^{-1}(\overline{\mathit{enum}_j})$;

- $\pi(\zeta)[\![s_{X',i}]\!] = \zeta[\![s_{X',i}]\!]$ for $X' \in \Upsilon^\dagger$, and $\pi(\zeta)[\![p_{X'}]\!] = \zeta[\![p_{X'}]\!]$ for $X' \in \Upsilon \setminus \{X\}$;

– for any $(\epsilon^\dagger, \zeta^\dagger) \in \mathcal{I}^\dagger$, $(\iota(\epsilon^\dagger), \iota(\zeta^\dagger)) \in \mathcal{I}$ be defined by:
  - $\iota(\epsilon^\dagger)[\![X']\!] = \epsilon^\dagger[\![X']\!]$ for $X' \in \Upsilon \setminus \{X\}$, and

$$\iota(\epsilon^\dagger)[\![X]\!] = \{1\} \times \epsilon^\dagger[\![Y_1]\!] \cup \cdots \cup \{l(X)\} \times \epsilon^\dagger[\![Y_{l(X)}]\!]$$

  - $\iota(\zeta^\dagger)[\![s_{X',i}]\!] = \zeta^\dagger[\![s_{X',i}]\!]$ for $X' \in \Upsilon \setminus \{X\}$, $\iota(\zeta^\dagger)[\![s_{Y_j,i}]\!] = (j, \zeta^\dagger[\![s_{Y_j,i}]\!])$, $\iota(\zeta^\dagger)[\![p_{X'}]\!] = \zeta^\dagger[\![p_{X'}]\!]$ for $X' \in \Upsilon \setminus \{X\}$, and $\iota(\zeta^\dagger)[\![p_X]\!](j,a) = \overline{\mathit{enum}_j}$;

– for any $(\epsilon^\dagger, \zeta^\dagger) \in \mathcal{I}^\dagger$, $\mathcal{S}^\dagger_{(\epsilon^\dagger,\zeta^\dagger)} = \mathcal{S}_{(\iota(\epsilon^\dagger),\iota(\zeta^\dagger))}$, and $\mathcal{R}^\dagger_{(\epsilon^\dagger,\zeta^\dagger)} = \mathcal{R}_{(\iota(\epsilon^\dagger),\iota(\zeta^\dagger))}$.

With these definitions, it is straightforward to check that:

– $\Upsilon^\dagger$, $k^\dagger$, $l^\dagger$, $T^\dagger$, $U^\dagger$, $V^\dagger$, $\Gamma^\dagger$, $\mathcal{I}^\dagger$, $\underline{\mathcal{S}^\dagger}$, $\underline{\mathcal{R}^\dagger}$, $(\pi(\delta), \pi(\eta))$ and $(\pi(\delta'), \pi(\eta'))$ satisfy the assumptions of the theorem;

– for any $(\epsilon, \zeta) \in \mathcal{I}$, $\mathcal{S}_{(\epsilon,\zeta)}$ satisfies $\mathcal{R}_{(\epsilon,\zeta)}$ iff $\mathcal{S}^\dagger_{(\pi(\epsilon),\pi(\zeta))}$ satisfies $\mathcal{R}^\dagger_{(\pi(\epsilon),\pi(\zeta))}$.

As the second alternative, suppose $X \in \Upsilon$ is such that $k(X) > 0$ and $l(X) = 1$, and let:

– $k^\dagger = k[X \mapsto (k(X) - 1)]$;

– $T^\dagger = X \times T$, and $V^\dagger = \mathit{Single} + V$;

– $\Gamma^\dagger = \Gamma \setminus [s_{X,k(X)} \mapsto X]$;

– $\mathcal{I}^\dagger$ be defined as in Definition 8 with Υ-equalities, $k^\dagger$-constants and l-predicates;

– for any $(\epsilon, \zeta) \in \mathcal{I}$, $\pi(\zeta) = \zeta \setminus [s_{X,k(X)} \mapsto \zeta[\![s_{X,k(X)}]\!]]$;

– for any $(\epsilon, \zeta^\dagger) \in \mathcal{I}^\dagger$, $\mathcal{S}^\dagger_{(\epsilon,\zeta^\dagger)}$ be the LTS with set of states $[\![T^\dagger]\!]_\epsilon$, with set of transition labels $[\![U]\!]_\epsilon$, and such that:
  - (a_1, a_2) is an initial state of $\mathcal{S}^\dagger_{(\epsilon,\zeta^\dagger)}$ iff

$$\forall i \in \{1, \ldots, k(X) - 1\}. a_1 \neq \zeta^\dagger[\![s_{X,i}]\!]$$

    and a_2 is an initial state of $\mathcal{S}_{(\epsilon,\zeta^\dagger[s_{X,k(X)} \mapsto a_1])}$;
  - $(a_1, a_2) \xrightarrow{b} (a_3, a_4)$ in $\mathcal{S}^\dagger_{(\epsilon,\zeta^\dagger)}$ iff $a_1 = a_3$ and $a_2 \xrightarrow{b} a_4$ in $\mathcal{S}_{(\epsilon,\zeta^\dagger[s_{X,k(X)} \mapsto a_1])}$;

– for any $(\epsilon, \zeta^\dagger) \in \mathcal{I}^\dagger$, $\mathcal{R}^\dagger_{(\epsilon,\zeta^\dagger)}$ be the GRA with set of states $[\![V^\dagger \times \mathit{Bool}^{2q}]\!]_\epsilon$, with set of transition labels $[\![T^\dagger]\!]_\epsilon$, and such that:
  - $(c^\dagger, h)$ is an initial state of $\mathcal{R}^\dagger_{(\epsilon,\zeta^\dagger)}$ iff $c^\dagger = (1, ())$ and $h = (\overline{\mathit{false}})^{2q}$;
  - $((1, ()), h_0) \xrightarrow{(a_1,a_2)} (c^\dagger, h_2)$ in $\mathcal{R}^\dagger_{(\epsilon,\zeta^\dagger)}$ iff $c^\dagger$ is of the form $(2, c_2)$ and $(c_1, h_1) \xrightarrow{a_2} (c_2, h_2)$ in $\mathcal{R}_{(\epsilon,\zeta^\dagger[s_{X,k(X)} \mapsto a_1])}$, where (c_1, h_1) is the unique initial state of $\mathcal{R}_{(\epsilon,\zeta^\dagger[s_{X,k(X)} \mapsto a_1])}$;
  - $((2, c_1), h_1) \xrightarrow{(a_1,a_2)} (c^\dagger, h_2)$ in $\mathcal{R}^\dagger_{(\epsilon,\zeta^\dagger)}$ iff $c^\dagger$ is of the form $(2, c_2)$ and $(c_1, h_1) \xrightarrow{a_2} (c_2, h_2)$ in $\mathcal{R}_{(\epsilon,\zeta^\dagger[s_{X,k(X)} \mapsto a_1])}$.

With these definitions, it is again straightforward to check that:

- Υ, $k^\dagger$, l, $T^\dagger$, U, $V^\dagger$, $\Gamma^\dagger$, $\mathcal{I}^\dagger$, $\underline{\mathcal{S}}^\dagger$, $\underline{\mathcal{R}}^\dagger$, $(\delta, \pi(\eta))$ and $(\delta', \pi(\eta'))$ satisfy the assumptions of the theorem;
- for any $(\epsilon, \zeta) \in \mathcal{I}$ such that

$$\forall i \in \{1, \ldots, k(X) - 1\}. \zeta[\![s_{X,k(X)}]\!] \neq \zeta[\![s_{X,i}]\!]$$

$\mathcal{S}_{(\epsilon,\zeta)}$ satisfies $\mathcal{R}_{(\epsilon,\zeta)}$ iff $\mathcal{S}^\dagger_{(\epsilon,\pi(\zeta))}$ satisfies $\mathcal{R}^\dagger_{(\epsilon,\pi(\zeta))}$.

By applying the two transformations above finitely many times, we can assume that $k(X) = 0$ and $l(X) = 1$ for all $X \in \Upsilon$. The conclusion of the theorem then follows by Theorem 1. $\square$

Lemma 4. *Suppose:*

- *P is a program which declares n state variables $x_1, \ldots, x_n$ respectively of type $T_1, \ldots, T_n$ (i.e. $StateType(P) = T_1 \times \cdots \times T_n$);*
- *$[\![P]\!] = (\Upsilon(P), \Gamma(P), T, U, \mathcal{I}, \underline{\mathcal{S}})$ where $T = StateType(P)$ and $U = ActType(P)$;*
- *(δ, η) and (δ', η') are two instantiations of the signature $(\Upsilon(P), \Gamma(P))$;*
- *(ρ, δ, δ') is a relation map which relates (δ, η) and (δ', η');*
- *V and W are types;*
- *e is an expression from P of type V; and*
- *$a \in [\![W]\!]_\delta$ and $a' \in [\![W]\!]_{\delta'}$.*

If $a[\![W]\!]_{(\rho,\delta,\delta')}a'$, then $[\![e]\!]_a[\![V]\!]_{(\rho,\delta,\delta')}[\![e]\!]_{a'}$.

Proof. By induction on e.

- If $e = x_i$, then $V = T_i$, $[\![e]\!]_a = \pi_i(a)$, and $[\![e]\!]_{a'} = \pi_i(a')$. Thus $[\![e]\!]_a[\![V]\!]_{(\rho,\delta,\delta')}[\![e]\!]_{a'}$.
- If $e = y$, then there exists an i such that $V = U_i$ (i.e. the ith component of the sum type U), $[\![e]\!]_a = \pi_{n+1}(a)$, and $[\![e]\!]_{a'} = \pi_{n+1}(a')$. Thus $[\![e]\!]_a[\![V]\!]_{(\rho,\delta,\delta')}[\![e]\!]_{a'}$.
- If $e = f(e_1, \ldots, e_q)$ and f is of type $V_1 \times \cdots \times V_q \to V$, then, by induction hypothesis, for each $i \in \{1, \ldots, q\}$, $[\![e_i]\!]_a[\![V_i]\!]_{(\rho,\delta,\delta')}[\![e_i]\!]_{a'}$.
  - If $f \in Domain(\Gamma(P))$, then $[\![e]\!]_a = [\![f(e_1, \ldots, e_q)]\!]_a = \eta[\![f]\!]([\![e_1]\!]_a, \ldots, [\![e_q]\!]_a)$, and (ρ, δ, δ') relates (δ, η) and (δ', η') implies

$$\eta[\![f]\!][\![V_1 \times \cdots \times V_q \to V]\!]\eta'[\![f]\!].$$

    Therefore $[\![e]\!]_a[\![V]\!]_{(\rho,\delta,\delta')}[\![e]\!]_{a'}$.
  - If $f \notin Domain(\Gamma(P))$, then $[\![e]\!]_a = [\![f(e_1, \ldots, e_q)]\!]_a = [\![f]\!]([\![e_1]\!]_a, \ldots, [\![e_q]\!]_a)$, and the expressions $e_1, \ldots, e_q$ do not contain any sub-expression of any type $X \in \Upsilon(P)$. Therefore, by identity preservation, $[\![e]\!]_a[\![V]\!]_{(\rho,\delta,\delta')}$ $[\![e]\!]_{a'}$. $\square$

Notation ($\oplus$). Suppose, for any natural number n, a is a tuple $(u_1, \ldots, u_n)$ and u is a value. We define $a \oplus u$ to be the tuple $(u_1, \ldots, u_n, u)$. $\square$

Proof (Proposition 2). Suppose P is a program and $[\![P]\!] = (\Upsilon(P), \Gamma(P), T, U, \mathcal{I}, \underline{\mathcal{S}})$ where $T = \mathit{StateType}(P)$ and $U = \mathit{ActType}(P)$. Suppose $(\delta, \eta), (\delta', \eta') \in \mathcal{I}$ and (ρ, δ, δ') is a relation map such that (ρ, δ, δ') relates (δ, η) and (δ', η'). We have to prove (ρ, δ, δ') relates $[\![P]\!]_{(\delta,\eta)}$ and $[\![P]\!]_{(\delta',\eta')}$. In other words, we have to prove $[\![T]\!]_{(\rho,\delta,\delta')}$ is a universal partial $[\![U]\!]_{(\rho,\delta,\delta')}$-bisimulation between $[\![P]\!]_{(\delta,\eta)}$ and $[\![P]\!]_{(\delta',\eta')}$.

- Suppose $a \in [\![T]\!]_{(\delta,\eta)}$ and $a' \in [\![T]\!]_{(\delta',\eta')}$ are such that $a[\![T]\!]_{(\rho,\delta,\delta')}a'$. Suppose $a \in I_{(\delta,\eta)}$ and the expression *init* is the initial condition of P. Thus $[\![\mathit{init}]\!]_a$ is equal to $\overline{\mathit{true}}$. From $a[\![T]\!]_{(\rho,\delta,\delta')}a'$, we deduce, by applying Lemma 4, $[\![\mathit{init}]\!]_a[\![\mathit{Bool}]\!]_{(\rho,\delta,\delta')}[\![\mathit{init}]\!]_{a'}$. Thus, by identity preservation, $[\![\mathit{init}]\!]_{a'}$ is also equal to $\overline{\mathit{true}}$. Therefore $a' \in I_{(\delta',\eta')}$. Symmetrically, we can prove $a' \in I_{(\delta',\eta')}$ implies $a \in I_{(\delta,\eta)}$.
- Suppose $a_1, a_2 \in [\![T]\!]_{(\delta,\eta)}$, $a_1' \in [\![T]\!]_{(\delta',\eta')}$, $b \in [\![U]\!]_{(\delta,\eta)}$, and $b' \in [\![U]\!]_{(\delta',\eta')}$ are such that $a_1[\![T]\!]_{(\rho,\delta,\delta')}a_1'$, $b[\![U]\!]_{(\rho,\delta,\delta')}b'$, and $a_1 \xrightarrow{b}_{(\delta,\eta)} a_2$. Suppose the guarded assignments of P are $\mathit{assign}_1,\ldots,\mathit{assign}_p$. For each $i \in \{1, \ldots, p\}$, the guarded assignment assign_i is such that its input variable is of type U_i and its assignment is $x_{i_1}, \ldots, x_{i_q} := e_1, \ldots, e_q$. Thus there exists $u \in [\![U_i]\!]_\delta$, such that $b = (i, u)$. $b[\![U]\!]_{(\rho,\delta,\delta')}b'$ implies there exists u' such that $b' = (i, u')$ and $u[\![U_i]\!]_{(\rho,\delta,\delta')}u'$. Thus, by application of the inference rule, $a_1' \xrightarrow{b'}_{(\delta',\eta')} a_2'$ where $a_2' = a_1'[i_1 \mapsto [\![e_1]\!]_{a_1' \oplus u'}, \ldots i_q \mapsto [\![e_q]\!]_{a_1' \oplus u'}]$. $a_2[\![T]\!]_{(\rho,\delta,\delta')}a_2'$ follows by applying Lemma 4.
- Symmetrically, we can prove the last condition. □

Proof (Proposition 3). Suppose P is a program. From the proposition 2, we know $[\![P]\!]$ is parametric. It is clear that other conditions for data-independence are satisfied. Therefore $[\![P]\!]$ is data-independent with equalities on Υ'. □

Proof (Proposition 4). Suppose P is a program, and $[\![P]\!] = (\Upsilon(P), \Gamma(P), T, U, \mathcal{I}, \underline{\mathcal{S}})$.

- We assume there exists an instantiation (δ, η) of $(\Upsilon(P), \Gamma(P))$ and a reachable state $a \in [\![T]\!]_\delta$ such that $\kappa_X([\![P]\!]) = \mathit{Comps}_{X,T}(a)$. Note that if there is no reachable state, then $\kappa_X([\![P]\!]) = 0$, and thus $k_{X,T} \geq \kappa_X([\![P]\!])$.
  We prove by induction on T that $k_{X,T} \geq |\mathit{Comps}_{X,T}(a)|$.
  - Suppose $T = X$. Thus $|\mathit{Comps}_{X,X}(a)| = |\{a\}| = 1$ and $k_{X,X} = 1 \geq |\mathit{Comps}_{X,X}(a)|$.
  - Suppose $T = Y$. Thus $|\mathit{Comps}_{X,Y}(a)| = |\{\}| = 0$ and $k_{X,Y} = 0 \geq |\mathit{Comps}_{X,Y}(a)|$.
  - Suppose $T = T_1 + \cdots + T_n$. Thus there exists i and a' such that $a = (i, a')$. By induction hypothesis, $k_{X,T_i} \geq |\mathit{Comps}_{X,T_i}(a')|$. Thus $k_{X,T_1+\cdots+T_n} \geq k_{X,T_i} \geq |\mathit{Comps}_{X,T_i}(a')| = |\mathit{Comps}_{X,T_1+\cdots+T_n}(a)|$.
  - Suppose $T = T_1 \times \cdots \times T_n$. Thus there exists $a_1, \ldots, a_n$ such that $a = (a_1, \ldots, a_n)$. By induction hypothesis, for any i such that $1 \leq i \leq n$, $k_{X,T_i} \geq |\mathit{Comps}_{X,T_i}(a_i)|$. Thus $k_{X,T_1\times\cdots\times T_n} = \Sigma_{i=1}^n k_{X,T_i} \geq \Sigma_{i=1}^n |\mathit{Comps}_{X,T_i}(a_i)| \geq |\mathit{Comps}_{X,T_1}(a_1) \cup \cdots \cup \mathit{Comps}_{X,T_n}(a_n)| = |\mathit{Comps}_{X,T_1\times\cdots\times T_n}(a)|$.

Thus $k_{X,T} \geq |Comps_{X,T}(a)| = \kappa_X([\![P]\!])$.

– We assume there exists an instantiation (δ, η) of $(\Upsilon(P), \Gamma(P))$, a reachable state $a_1 \in [\![T]\!]_\delta$, a state $a_2 \in [\![T]\!]_\delta$, and a label $b \in [\![U]\!]_\delta$ such that $a_1 \xrightarrow{b}_{(\delta,\eta)} a_2$ and $\lambda_X([\![P]\!]) = Comps_{X,U}(b)$. Note that if there is no reachable state, then $\lambda_X([\![P]\!]) = 0$, and thus $k_{X,U} \geq \lambda_X([\![P]\!])$.
We have $U = U_1 + \cdots + U_p$. Thus there exists an i, such that $1 \leq i \leq p$, and a b' such that $b = (i, b')$. According to the previous item, we have $k_{X,U_i} \geq |Comps_{X,U_i}(b')|$. Thus $k_{X,U} \geq k_{X,U_i} \geq |Comps_{X,U_i}(b')| = |Comps_{X,U}(b)| = \lambda_X([\![P]\!])$. □

Proof (Proposition 5).
Suppose φ is an LTL formula and $[\![\varphi]\!] = (\Upsilon(\varphi), \Gamma(\varphi), \mathit{StateType} \times \mathit{Bool}^{2q}, \mathit{ActType}(\varphi), \mathcal{I}, \underline{\mathcal{R}})$. Suppose $(\delta, \eta), (\delta', \eta') \in \mathcal{I}$ and (ρ, δ, δ') is a relation map which relates (δ, η) and (δ', η'). We have to prove (ρ, δ, δ') relates $[\![\varphi]\!]_{(\delta,\eta)}$ and $[\![\varphi]\!]_{(\delta',\eta')}$. That is to say, we have to prove $[\![\mathit{StateType} \times \mathit{Bool}^{2q}]\!]_{(\rho,\delta,\delta')}$ is a universal partial $[\![\mathit{ActType}(\varphi)]\!]_{(\rho,\delta,\delta')}$-bisimulation between $[\![\varphi]\!]_{(\delta,\eta)}$ and $[\![\varphi]\!]_{(\delta',\eta')}$.

– Suppose $a \in [\![\mathit{StateType} \times \mathit{Bool}^{2q}]\!]_{(\delta,\eta)}$ and $a' \in [\![\mathit{StateType} \times \mathit{Bool}^{2q}]\!]_{(\delta',\eta')}$ are such that $a[\![\mathit{StateType} \times \mathit{Bool}^{2q}]\!]_{(\rho,\delta,\delta')}a'$. Suppose $a \in I_{(\delta,\eta)}$. The type $\mathit{StateType}$ does not contain any free type variable from $\Upsilon(\varphi)$. Therefore, by identity preservation, $a[\![\mathit{StateType} \times \mathit{Bool}^{2q}]\!]_{(\rho,\delta,\delta')}a'$ implies $a = a'$. And $I_{(\delta,\eta)} = I_{(\delta',\eta')}$, by construction of the LTS. Therefore $a' \in I_{(\delta',\eta')}$. Symmetrically, we can prove $a' \in I_{(\delta',\eta')}$ implies $a \in I_{(\delta,\eta)}$.
– Suppose $a_1, a_2 \in [\![\mathit{StateType} \times \mathit{Bool}^{2q}]\!]_{(\delta,\eta)}$, $a_1' \in [\![\mathit{StateType} \times \mathit{Bool}^{2q}]\!]_{(\delta',\eta')}$, $b \in [\![\mathit{ActType}(\varphi)]\!]_{(\delta,\eta)}$, and $b' \in [\![\mathit{ActType}(\varphi)]\!]_{(\delta',\eta')}$ are such that
$a_1[\![\mathit{StateType} \times \mathit{Bool}^{2q}]\!]_{(\rho,\delta,\delta')}a_1'$, $b[\![\mathit{ActType}(\varphi)]\!]_{(\rho,\delta,\delta')}b'$, and $a_1 \xrightarrow{b}_{(\delta,\eta)} a_2$.
We have to prove there exists an a_2' such that $a_1' \xrightarrow{b'}_{(\delta',\eta')} a_2'$. The type $\mathit{StateType} \times \mathit{Bool}^{2q}$ does not contain any free type variable from $\Upsilon(\varphi)$. Therefore $a_1' = a_1$, by identity preservation. Let a_2' be equal to a_2. We now have to prove $a_1 \xrightarrow{b'}_{(\delta',\eta')} a_2$. Actually, by construction of the automaton, it is sufficient to prove that $F_{(\delta,\eta)}(b) = F_{(\delta',\eta')}(b')$. Suppose $p(x_{i_1}, \ldots, x_{i_q}) \in F_{(\delta,\eta)}(b)$. For any $i \in \{i_1, \ldots, i_q\}$, $[\![x_i]\!]_{(\delta,\eta),b} = \pi_i(b)$ and $[\![x_i]\!]_{(\delta,\eta),b'} = \pi_i(b')$. $b[\![\mathit{ActType}(\varphi)]\!]_{(\rho,\delta,\delta')}b'$, thus $[\![x_i]\!]_{(\delta,\eta),b}[\![T_i]\!]_{(\rho,\delta,\delta')}[\![x_i]\!]_{(\delta,\eta),b'}$ where T_i is the type of the variable x_i.
  - If $p \in \Gamma(\varphi)$, we then deduce $\eta[\![p]\!](\pi_{i_1}(b), \ldots, \pi_{i_q}(b))$ is true, by definition of $F_{(\delta,\eta)}$. (ρ, δ, δ') relates (δ, η) and (δ', η') implies

$$\eta[\![p]\!][\![T_1 \times \cdots \times T_n \to \mathit{Bool}]\!]_{(\rho,\delta,\delta')}\eta'[\![p]\!].$$

    Thus $[\![p(x_{i_1}, \ldots, x_{i_q})]\!]_{(\delta,\eta),b}[\![\mathit{Bool}]\!]_{(\rho,\delta,\delta')}[\![p(x_{i_1}, \ldots, x_{i_q})]\!]_{(\delta',\eta'),b'}$. Thus, by identity preservation, $[\![p(x_{i_1}, \ldots, x_{i_q})]\!]_{(\delta,\eta),b'} = \mathit{true}$. Therefore $p(x_{i_1}, \ldots, x_{i_q}) \in F_{(\delta',\eta')}(b')$.
  - If $p \notin \Gamma(\varphi)$, we then deduce, by definition of $F_{(\delta,\eta)}$, that $[\![p]\!](\pi_{i_1}(b), \ldots, \pi_{i_q}(b))$, where $[\![p]\!]$ is the semantics of p, is true. And the types of $x_{i_1}, \ldots,$

x_{i_q} do not contain any type variable from $\Upsilon(\varphi)$. Thus, by identity preservation, $[\![p(x_{i_1},\ldots,x_{i_q})]\!]_{(\delta,\eta),b}[\![Bool]\!]_{(\rho,\delta,\delta')}[\![p(x_{i_1},\ldots,x_{i_q})]\!]_{(\delta',\eta'),b'}$. Therefore $p(x_{i_1},\ldots,x_{i_q}) \in F_{(\delta',\eta')}(b')$.

Therefore $F_{(\delta,\eta)}(b) \subseteq F_{(\delta',\eta')}(b')$. Symmetrically, we can prove $F_{(\delta',\eta')}(b') \subseteq F_{(\delta,\eta)}(b)$. □

Proof (Proposition 6). From the proposition 5, we know $[\![\varphi]\!]$ is parametric. It is clear that other conditions for data-independence are satisfied. Therefore $[\![\varphi]\!]$ is data-independent with equalities on Υ'. □

Proof (Proposition 7). By definition of $\kappa_X([\![\varphi]\!])$. □