Alternating Simulation and IOCO*

Margus Veanes and Nikolaj Bjørner

Microsoft Research, Redmond, WA, USA
{margus,nbjorner}@microsoft.com

Abstract. We propose a symbolic framework called *guarded labeled assignment systems* or GLASs and show how GLASs can be used as a foundation for symbolic analysis of various aspects of formal specification languages. We define a notion of i/o-refinement over GLASs as an alternating simulation relation and provide formal proofs that relate i/o-refinement to ioco. We show that non-i/o-refinement reduces to a reachability problem and provide a translation from bounded non-i/o-refinement or bounded non-ioco to checking first-order assertions.

1 Introduction

The view of a system behavior as a labeled transition system (LTS) provides the semantical foundation for many behavioral aspects of systems in the context of formal verification and testing. The central problem in testing is to determine if an implementation LTS *conforms* to a given specification LTS and to find a counterexample if this is not the case. In the case of open systems, or in the presence of input (controllable) and output (observable) behavior, the conformance relation is commonly described as input-output conformance or *ioco* [39]. A closely related notion of *alternating simulation* [4] is used in the context of open system verification, in particular for interface automata refinement [19, 18]. In this paper we propose a theory of *guarded labeled assignment systems* or *GLASs* that formally relates these two notions and provides a foundation for their symbolic analysis.

The main characteristic of *symbolic* analysis techniques is that it makes use of *implicit* representations of (parts of) program behavior, typically as constraints in an appropriate logic, avoiding state-space explosion that would otherwise arise if the analysis would be performed using *explicit* state exploration or concrete execution. For scalability, symbolic techniques are often considered as a necessary ingredient in modern testing techniques, including, but not limited to, fuzz testing [26], unit testing [38] and model-based testing [17]. Some key enabling factors behind the use of symbolic techniques can be attributed to recent advances in satisfiability modulo theories solving [21].

GLASs are a generalization of *non-deterministic model programs* [45] to a purely symbolic setting, by abstracting from the particular background universe and the particular (action) label domain. The semantics of GLASs uses classical

* Microsoft Research Technical Report MSR-TR-2010-38, Updated September 2011

model theory. A GLAS is a symbolic representation of behavior whose trace semantics is given by an LTS that corresponds to the least fix-point of the strongest post-condition induced by the assignment system of the GLAS. We define the notion of *i/o-refinement* over GLASs that is based on alternating simulation and show that it is a generalization of *ioco* for all GLASs, generalizing an earlier result [42] for the deterministic case. The notion of i/o-refinement is essentially a *compositional* version of *ioco*. We provide a rigorous account for formally dealing with *quiescence* in GLASs in a way that supports symbolic analysis with or without the presence of quiescence. We also define the notion of a symbolic composition of GLASs that respects the standard parallel synchronous composition of LTSs [32, 34] with the interleaving semantics of unshared labels. Composition of GLASs is used to show that the i/o-refinement relation between two GLASs can be formulated as an condition of the composite GLAS. This leads to a mapping of the non-i/o-refinement checking problem into a reachability checking problem for a pair of GLASs. For a class of GLASs that we call *robust* we can furthermore use established methods developed for verifying safety properties of reactive systems. We show that the non-i/o-refinement checking problem can be reduced to first-order assertion checking by using proof-rules similar to those that have been formulated for checking invariants of reactive systems. It can also be approximated as a *bounded model program checking problem* [45].

Although the focus of the paper is theoretical, GLASs provide a foundation of applying state-of-the-art *satisfiability modulo theories* [7, 21] (SMT) technology to a wide range of problems that are difficult to tackle using other techniques. Modern SMT solvers combine a hybrid set of technologies. Most state-of-the-art solvers include efficient SAT solvers for handling finite domains. They also support operations over unbounded universes, such as integers, and integrate reasoning with quantifiers. Compared to many automated theorem proving techniques, they provide *solutions* as witness of satisfiability. The following three are sample applications: 1) symbolic model-checking of a given specification GLAS [45] with respect to a given property automaton; 2) symbolic refinement checking between two symbolic LTSs represented as GLASs; 3) incremental model-based parameter generation during on-the-fly testing for increased specification GLAS coverage. In all cases, the use of GLAS composition is central, e.g., for symbolic i/o-refinement or *ioco*, composition is used in Theorem 9. All examples used in the paper are tailored to such analyses and illustrate the use of background theories that are supported by state-of-the-art SMT solvers such as Z3 [20].

The paper is an extension of the conference papers [44, 42]. Besides minor corrections and changes in the structure of the paper, it includes full details of all proofs, more examples and explanations, an experimental evaluation section, and a more comprehensive related work section.

2 Preliminaries

We use classical logic and work in a fixed multi-sorted universe $\mathcal{U}$ of values. For each sort σ, $\mathcal{U}^\sigma$ is a sub-universe of $\mathcal{U}$. The basic sorts needed in this paper

are the Boolean sort $\mathbb{B}$, ($\mathcal{U}^{\mathbb{B}} = \{true, false\}$), and the integer sort $\mathbb{Z}$. There is a collection of functions with a fixed meaning associated with the universe, e.g., arithmetical operations over $\mathcal{U}^{\mathbb{Z}}$. These functions (and the corresponding function symbols) are called *background* functions. For example, the background function $<: \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$ denotes the standard order on integers. There is also a generic background function $Ite{:}\mathbb{B} \times \sigma \times \sigma \to \sigma$ where σ is a given sort.

Terms are defined by induction as usual and are assumed to be well-sorted. The sort σ of a term t is denoted by $sort(t)$ or by $t{:}\sigma$. We write $FV(t)$ for the set of free variables in t. Boolean terms are also called *formulas* or *predicates*. For example if P is the formula $x < y \wedge \exists x'(x < x' \wedge x' < y)$ then $FV(P) = \{x, y\}$. A term t *over* Σ has $FV(t) \subseteq \Sigma$. A term with no free variables is *closed*.

We use x' as an *injective renaming* operation on variables x, and lift the renaming to sets of variables, $\Sigma' \stackrel{\text{def}}{=} \{x' \mid x \in \Sigma\}$. We also introduce the following, technically very convenient, convention. If t is a term, then t' is a term where each variable x in t, including each bound variable, has been renamed by x', e.g., given P as above, P' is the formula $x' < y' \wedge \exists x''(x' < x'' \wedge x'' < y')$.

A *Σ-model* M is a mapping from Σ to $\mathcal{U}$.[1] The interpretation of a term t over Σ in a Σ-model M, is denoted by t^M and is defined by induction as usual. In particular, for *Ite*-terms:

$$Ite(\varphi, t, f)^M = \begin{cases} t^M, \text{ if } \varphi^M = true; \\ f^M, \text{ otherwise.} \end{cases}$$

Let φ be a formula over Σ. A Σ-model M *satisfies* φ or φ *is true in* M, denoted by $M \models \varphi$, if $\varphi^M = true$; φ is *satisfiable* if it has a model; φ is *true* if $\varphi^M = true$ for all Σ-models M. We use elements in $\mathcal{U}$ also as terms and define the *predicate of a Σ-model* M as the predicate $P_M \stackrel{\text{def}}{=} \bigwedge_{x \in \Sigma} x = x^M$ over Σ. Note that for any predicate P over Σ, $\exists \Sigma P$ and $\exists \Sigma' P'$ are equivalent closed formulas.

3 Guarded Labeled Assignment Systems

This section introduces Guarded Labeled Assignment Systems, GLAS for short. The definition of GLAS combines labels, guarded updates, and internal choice. They capture the semantics of model programs. Model programs are high-level operational specifications [31]. Model programs are being used to model complex application level network protocols in the industrial protocol testing tool SpecExplorer [17]. A sample model program is illustrated below in Example 2.

We start by providing the formal definition, which is followed by examples illustrating the definition. An *assignment* is a pair $x := u$ where x is a variable, u is a term, and $sort(x) = sort(u)$.

[1] More precisely, variables are viewed as fresh constants expanding the background signature. Note that the background function symbols have the same interpretation in all models (and are thus implicit).

Definition 1. A *Guarded Labeled Assignment System* or *GLAS* G is a tuple $(\Sigma, X, \ell, \imath, \alpha, \gamma, \Delta)$ where

- Σ is a finite set of variables called the *model signature*;
- X is a finite set of variables disjoint from Σ called the *choice signature*;
- ℓ is a variable not in Σ or X, called the *label variable*;
- $\imath$ is a satisfiable formula over Σ called the *initial condition*;
- α is a formula over $\{\ell\}$ called the *label predicate*;
- γ is a formula over $\Sigma \cup X \cup \{\ell\}$ called the *guard*;
- Δ is a set of assignments $\{z := u_z\}_{z \in \Sigma}$ where, for all $z \in \Sigma$, u_z is a term over $\Sigma \cup X \cup \{\ell\}$; Δ is called the *assignment system*.

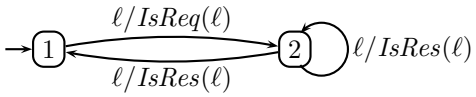
The set $\Sigma \cup X$ is called the *internal signature* of G.

We first illustrate a simple two-state GLAS.

Example 1. Consider the GLAS

$$A = (\{z{:}\mathbb{Z}\}, \{x{:}\mathbb{B}\}, \ell{:}\mathbb{L}, z = 1, \mathit{IsReq}(\ell) \vee \mathit{IsRes}(\ell), \\ \mathit{Ite}(\mathit{IsReq}(\ell), z = 1, z = 2), \{z := \mathit{Ite}(z = 1, 2, \mathit{Ite}(x, 1, 2))\}).$$

A has one integer valued model variable z; one Boolean choice variable x; the labels have sort $\mathbb{L}$ (suppose $\mathbb{L}$ is associated with predicates $\mathit{IsReq}, \mathit{IsRes}{:}\mathbb{L} \to \mathbb{B}$); the initial condition is that $z = 1$; the label predicate specifies that ℓ is either a "request" (satisfies the predicate *IsReq*) or a "response" (satisfies the predicate *IsRes*). We will enforce that the predicates *IsReq* and *IsRes* are mutually exclusive by treating $\mathbb{L}$ as an algebraic data-type with two constructors, one for requests, the other for responses. The predicates *IsReq* and *IsRes* recognize the respective data-type constructors. The guard of A states that requests are only enabled when $z = 1$ and responses are only enabled when $z = 2$; the assignment system contains the assignment that either changes the value of z from 1 to 2, or nondeterministically changes the value of z from 2 to 1 or 2. The nondeterministic choice is given by the value of x, that can be either true or false. A can be visualized as the following FSM where each state is labeled by the value of the model variable z.



Intuitively, A specifies a sequence of request and response labels where a single request is followed by one or more responses. ⊠

The following example illustrates how an AsmL [5, 27] program can be represented as a GLAS. Other encodings are possible using different techniques. The example makes use of several background sorts. Such sorts are derived from the given program. An important point regarding practical applications is that all sorts and associated axioms that are used, are either directly supported, or user definable without any significant overhead, in state-of-the-art SMT solvers.

Example 2. We consider the following model program called *Credits* that describes the message-id-usage facet of a client-server sliding window protocol [31].

```
var ranges as Set of (Integer,Integer) = {(0,0)}
var used as Set of Integer = {}
var max as Integer = 0
var msgs as Map of Integer to Integer = {->}

IsValidUnusedMessageId(m as Integer) as Boolean
  return not (m in used) and  Exists r in ranges where First(r)<=m and m<=Second(r)

[Action] Req(m as Integer, c as Integer)
  require IsValidUnusedMessageId(m) and c > 0
  msgs(m) := c
  add m to used

[Action] Res(m as Integer, c as Integer)
   require m in msgs and 0<=c and c<=msgs(m)
   remove m from msgs
   if c>0 add (max, max+c) to ranges
   max := max+c
```

Let us assume a sort $\mathbb{L}$ derived from the method signatures of the program; $\mathcal{U}^{\mathbb{L}}$ is an *algebraic data type*. In addition to the predicates *IsReq* and *IsRes* introduced in Example 1, $\mathbb{L}$ is associated with the constructors: $\mathit{Req}, \mathit{Res}{:}\mathbb{Z} \times \mathbb{Z} \to \mathbb{L}$ and accessors: $\mathit{Req\_m}, \mathit{Res\_m}, \mathit{Req\_c}, \mathit{Res\_c}{:}\mathbb{L} \to \mathbb{Z}$. For example, $\mathit{IsReq}(\mathit{Res}(6,7))$ is false and $\mathit{Req\_c}(\mathit{Req}(3,4))$ is equal to 4.

The example uses *tuples*. There is a generic n-tuple sort $\mathbb{T}(\sigma_0, \ldots, \sigma_{n-1})$ of given element sorts σ_i for $i < n$. An n-tuple constructor is denoted by $\langle t_0, \ldots, t_{n-1} \rangle$ and the projection functions are denoted by π_i for $i < n$. For example $\pi_1(\langle t_0, t_1 \rangle) = t_1$.

The example also uses *arrays*, the sort $\mathbb{A}(\sigma, \rho)$ is a generic sort for extensional arrays (mathematical maps) with domain sort σ and range sort ρ. The functions on arrays are reading and storing elements in the array:

$$\mathit{Read}{:}\mathbb{A}(\sigma,\rho) \times \sigma \to \rho, \quad \mathit{Store}{:}\mathbb{A}(\sigma,\rho) \times \sigma \times \rho \to \mathbb{A}(\sigma,\rho).$$

The *empty array* ε maps every domain element to a *default* value of the range sort. For $\mathbb{Z}$ the default is 0 and for $\mathbb{B}$ the default is *false*. We map *Credits* to the GLAS G_{Credits}: $(\Sigma, \emptyset, \ell, \imath, \mathit{IsReq}(\ell) \vee \mathit{IsRes}(\ell), \gamma, \Delta)$ where

$$\Sigma = \{\mathit{ranges}{:}\mathbb{A}(\mathbb{T}(\mathbb{Z},\mathbb{Z}),\mathbb{B}), \mathit{used}{:}\mathbb{A}(\mathbb{Z},\mathbb{B}), \mathit{max}{:}\mathbb{Z}, \mathit{msgs}{:}\mathbb{A}(\mathbb{Z},\mathbb{Z})\}.$$

The axioms assumed for arrays are the usual ones for propagating reads over store and the extensionality axiom:

$$\forall a\, i\, v\, j\, (\mathit{Read}(\mathit{Store}(a,i,v),j) = \mathit{Ite}(i = j, v, \mathit{Read}(a,j)))$$

$$\forall a\, b\, (\forall i\, (\mathit{Read}(a,i) = \mathit{Read}(b,i)) \;\Rightarrow\; a = b)$$

For example, $\mathit{Read}(\mathit{Store}(\varepsilon,1,2),1) = 2$, $\mathit{Read}(\mathit{Store}(\varepsilon,1,2),3) = \mathit{default}$, and $\mathit{Store}(\mathit{array},\mathit{key},\mathit{default}) = \mathit{array}$. We write $\mathit{key} \in \mathit{array}$ for $\mathit{Read}(\mathit{array},\mathit{key}) \neq \mathit{default}$. The initial condition $\imath$ is

$$\mathit{ranges} = \mathit{Store}(\varepsilon, \langle 0,0 \rangle, \mathit{true}) \wedge \mathit{used} = \varepsilon \wedge \mathit{max} = 0 \wedge \mathit{msgs} = \varepsilon.$$

Given by the require-statements, the guard γ is:

$$\begin{array}{l}(\mathit{IsReq}(\ell) \wedge \mathit{Req\_m}(\ell) \notin \mathit{used} \\ \qquad \wedge \exists r\,(r \in \mathit{ranges} \wedge \pi_0(r) \leq \mathit{Req\_m}(\ell) \wedge \mathit{Req\_m}(\ell) \leq \pi_1(r)) \\ \qquad \wedge\ \mathit{Req\_c}(\ell) > 0) \vee \\ (\mathit{IsRes}(\ell) \wedge \mathit{Res\_m}(\ell) \in \mathit{msgs} \wedge 0 \leq \mathit{Res\_c}(\ell) \\ \qquad \wedge\ \mathit{Res\_c}(\ell) \leq \mathit{Read}(\mathit{msgs}, \mathit{Res\_m}(\ell)))\end{array}$$

The assignment system Δ consists of the assignments:

$$\begin{array}{rl}\mathit{ranges}:= & \mathit{Ite}(\mathit{IsReq}(\ell), \mathit{ranges}, \mathit{Ite}(\mathit{Res\_c}(\ell) > 0, \\ & \qquad \mathit{Store}(\mathit{ranges}, \langle \mathit{max}, \mathit{max} + \mathit{Res\_c}(\ell)\rangle, \mathit{true}), \mathit{ranges})) \\ \mathit{used}:= & \mathit{Ite}(\mathit{IsReq}(\ell), \mathit{Store}(\mathit{used}, \mathit{Req\_m}(\ell), \mathit{Req\_c}(\ell)), \mathit{used}) \\ \mathit{max}:= & \mathit{Ite}(\mathit{IsReq}(\ell), \mathit{max}, \mathit{max} + \mathit{Res\_c}(\ell)) \\ \mathit{msgs}:= & \mathit{Ite}(\mathit{IsReq}(\ell), \mathit{Store}(\mathit{msgs}, \mathit{Req\_m}(\ell), \mathit{Req\_c}(\ell)), \\ & \qquad \mathit{Store}(\mathit{msgs}, \mathit{Res\_m}(\ell), \mathit{default}_{\mathbb{Z}}))\end{array}$$

The right-hand-sides of the assignments are easy to automatically generate from the program, but much harder to comprehend than the original assignments in the program, since they combine all the assignments from the separate actions by doing a case split based on the action label. They also add trivial assignments that take care of the implicit *frame condition* in AsmL that states that all variables not updated retain their previous values. ⊠

A GLAS is a symbolic representation of a labeled transition system (LTS). Before defining the formal semantics, let us briefly revisit the GLAS A in Example 1. Assume also that $\mathbb{L}$ and $\mathcal{U}^{\mathbb{L}}$ are as defined in Example 2. Then the concrete LTS $\lfloor A \rfloor$ (formally defined in Definition 7) represented by A has two states S_1 and S_2 (corresponding to the states of the displayed FSM in Example 1) and $\lfloor A \rfloor$ has infinitely many transitions; namely, for each $a \in \mathcal{U}^{\mathbb{L}}$, if $\mathit{IsReq}(a)$ holds then there is a transition $S_1 \xrightarrow{a} S_2$ and if $\mathit{IsRes}(a)$ holds then there is a transition $S_2 \xrightarrow{a} S_2$ and a transition $S_2 \xrightarrow{a} S_1$. Thus, in this case A is a finite (symbolic) representation of the infinite LTS $\lfloor A \rfloor$.

In order to keep the paper self-contained and to fix notation we include the standard definitions of LTSs and traces.

Definition 2. An *LTS* is a tuple $\mathcal{L} = (\mathbf{S}, \mathbf{S}^0, L, T)$, where $\mathbf{S}$ is a set of *states*; $\mathbf{S}^0 \subseteq \mathbf{S}$ is a nonempty set of *initial states*; L is a set of *labels*; $T \subseteq \mathbf{S} \times L \times \mathbf{S}$ is a *transition relation*. A label $a \in L$ is *enabled in a state* S if $(S, a, S') \in T$ for some $S' \in \mathbf{S}$.

We use $\mathcal{L}$ as a subscript to identify its components. If $(S, a, S') \in T_{\mathcal{L}}$ we write $S \xrightarrow{a}_{\mathcal{L}} S'$ or $S \xrightarrow{a} S'$ if $\mathcal{L}$ is clear from the context. If $a \in L_{\mathcal{L}}$ is enabled in $S \in \mathbf{S}_{\mathcal{L}}$ write $S \xrightarrow{a}_{\mathcal{L}}$. If $a \in L_{\mathcal{L}}$ is not enabled in $S \in \mathbf{S}_{\mathcal{L}}$, we write $S \not\xrightarrow{a}_{\mathcal{L}}$. In this paper we are only concerned with *finite* traces.

Definition 3. A label sequence $\mathbf{a} = (a_i)_{i<k}$ such that $S_i \xrightarrow{a_i}_{\mathcal{L}} S_{i+1}$, $i < k$, is a *trace of* $\mathcal{L}$ *from* S_0 or a *trace of* $\mathcal{L}$ if $S_0 \in \mathbf{S}^0_{\mathcal{L}}$; we write $S_0 \xrightarrow{\mathbf{a}} S_k$ and $S \xrightarrow{\epsilon} S$ where ϵ is the empty sequence. The set of all traces of $\mathcal{L}$ is denoted by $\mathit{Tr}(\mathcal{L})$.

Definition 4. LTS $\mathcal{L}$ is *deterministic* if $\mathcal{L}$ has a single initial state and for all $a \in L$ and $S \in \mathbf{S}_\mathcal{L}$ there is at most one $S_1 \in \mathbf{S}$ such that $S \xrightarrow{a}_\mathcal{L} S_1$. We view a deterministic LTS $\mathcal{L}$ as a function from $L^*_\mathcal{L}$ to $\mathbf{S}_\mathcal{L} \cup \{\bot_\mathcal{L}\}$:

$$\mathcal{L}(\mathbf{a}) \stackrel{\text{def}}{=} \begin{cases} \bot_\mathcal{L}, \text{ if } \mathbf{a} \notin \mathit{Tr}(\mathcal{L}); \\ S, \quad \text{otherwise, where } \mathbf{S}^0_\mathcal{L} = \{S^0\} \text{ and } S^0 \xrightarrow{\mathbf{a}}_\mathcal{L} S. \end{cases} \tag{1}$$

Note that $\mathcal{L}(\epsilon)$ is the unique initial state of a deterministic LTS $\mathcal{L}$.

A GLAS is associated with a transition relation formula that describes a single application of its assignments and a predicate transformer that maps a given predicate to a new predicate. The predicate transformer is used below to define semantics of GLASs in terms of LTSs. Recall the variable renaming convention on terms that is used below.

Definition 5. Let $G = (\Sigma, X, \ell, \imath, \alpha, \gamma, \{z := u_z\}_{z \in \Sigma})$ be a GLAS. We define the *transition relation* TR_G, and the *strongest post-condition predicate transformer* SP_G, for G, where P is a predicate over Σ and $a \in \mathcal{U}^{sort(\ell)}$:

$$TR_G(\Sigma', \ell', \Sigma) \stackrel{\text{def}}{=} \alpha' \wedge \exists X' \, (\gamma' \wedge \bigwedge_{z \in \Sigma} z = u'_z)$$

$$SP_G(P, a) \stackrel{\text{def}}{=} \exists \Sigma' \, (P' \wedge TR_G(\Sigma', a, \Sigma))$$

Note that $SP_G(P, a)$ is a predicate over Σ. The intuition behind TR_G is that it is the symbolic transition relation corresponding to *one step* of G, where choice variables have a *local scope* within one step. The intuition behind $SP_G(P, a)$ is that it describes the strongest post-condition [23] of one step of G with respect to the symbolic state P and the concrete label a. The following example illustrates the definition of GLAS on a simple case.

Example 3. Let G be the GLAS

$$(\{z{:}\mathbb{Z}\}, \{x{:}\mathbb{B}\}, \ell{:}\mathbb{Z}, z \geq 0, \ell \neq 0, \mathit{Ite}(x, \ell > 0, \ell < 0), \{z := \mathit{Ite}(x, z + \ell, z - \ell)\}).$$

Then TR_G is the following predicate, where we apply equivalence preserving simplifications with respect to the assumed background of integer arithmetic:

$$\begin{aligned} TR_G(z', \ell', z) = \; & (\ell' \neq 0 \wedge \exists \, x'(\mathit{Ite}(x', \ell' > 0, \ell' < 0) \wedge z = \mathit{Ite}(x', z' + \ell', z' - \ell'))) \\ & \Leftrightarrow (\ell' \neq 0 \wedge ((\ell' > 0 \wedge z = z' + \ell') \vee (\ell' < 0 \wedge z = z' - \ell'))) \\ & \Leftrightarrow (\ell' \neq 0 \wedge z = z' + \ell') \end{aligned}$$

Let P be the initial condition $z \geq 0$ and consider the concrete label 3. Then $SP_G(P, 3)$ is the following predicate, where we apply equivalence preserving simplifications:

$$\begin{aligned} SP_G(P, 3) = \; & \exists \, z'(z' \geq 0 \wedge TR_G(z', 3, z)) \\ & \Leftrightarrow \exists \, z'(z' \geq 0 \wedge (3 \neq 0 \wedge z = z' + 3)) \\ & \Leftrightarrow z \geq 3 \end{aligned}$$

Note that in this particular case the existential quantifiers could be simplified away. This is clearly not the case in general, but typical for GLASs that correspond to model programs, such as the one in Example 2. ⊠

Next, we define two related semantics of a GLAS G in terms of LTSs. One is the *concrete semantics* $\lfloor G \rfloor$ and the other one is the *symbolic semantics* $\lceil G \rceil$. In the concrete semantics, states are Σ_G-models. In the symbolic semantics, states are predicates over Σ_G.

Definition 6. The set of *labels of* G is $L_G \stackrel{\text{def}}{=} \{\ell_G^M \mid M \models \alpha_G\}$.

Definition 7. $\lfloor G \rfloor \stackrel{\text{def}}{=} (\mathbf{S}, \{M \mid M \models \imath_G\}, L_G, T)$ where $\mathbf{S}$, T are the least sets such that $\mathbf{S}^0_{\lfloor G \rfloor} \subseteq \mathbf{S}$ and $(M, a, N) \in T$ for $a \in L_G$, $M \in \mathbf{S}$, and $N \models SP_G(P_M, a)$, then $N \in \mathbf{S}$.

Thus, in $\lfloor G \rfloor$ states are Σ_G-models and there is a transition $M \stackrel{a}{\rightarrow}_{\lfloor G \rfloor} N$ if the assignments of G, when evaluated in a state M where a is enabled yield the state N (for some choice of X_G).

Definition 8. $\lceil G \rceil \stackrel{\text{def}}{=} (\mathbf{S}, \{\imath_G\}, L_G, T)$ where $\mathbf{S}$, T are the least sets such that $\imath_G \in \mathbf{S}$, $(P, a, SP_G(P, a)) \in T$ for $a \in L_G$ and $P \in \mathbf{S}$ if $SP_G(P, a)$ is satisfiable.

Note that $\lceil G \rceil$ is a deterministic LTS by definition. In $\lceil G \rceil$, states are predicates over Σ_G, so several models may correspond to a single state in $\lceil G \rceil$. Also, $\lceil G \rceil$ may have several distinct but logically equivalent states. It is useful to consider the quotient of $\lceil G \rceil$ under logical equivalence, where logically equivalent states of $\lceil G \rceil$ form a single equivalence class. For a predicate P, define $[P]$ as *the class of all formulas logically equivalent to* P and lift the definition to sets S of formulas in the usual way: $[S] \stackrel{\text{def}}{=} \{[P] \mid P \in S\}$.

Definition 9. $[G] \stackrel{\text{def}}{=} ([\mathbf{S}_{\lceil G \rceil}], [\mathbf{S}^0_{\lceil G \rceil}], L_G, \{([P], a, [Q]) \mid (P, a, Q) \in T_{\lceil G \rceil}\})$.

We know from the definition of SP_G that if $P \Leftrightarrow Q$ then $SP_G(P, a) \Leftrightarrow SP_G(Q, a)$. Thus, if $[P] \stackrel{a}{\rightarrow}_{[G]}$ then $[P] \stackrel{a}{\rightarrow}_{[G]} [SP_G(P, a)]$, i.e., $[G]$ is also a *deterministic* LTS. $[G]$ provides a more intuitive way of understanding the symbolic semantics. In many cases, infinitely many equivalent predicates that arise through repeated applications of SP_G in $\lceil G \rceil$ are collapsed into a single state of $[G]$. Let $\bot_{[G]} \stackrel{\text{def}}{=} [\mathit{false}]$ and let $\bot_{\lceil G \rceil} \stackrel{\text{def}}{=} \mathit{false}$ (recall (1)).

Lemma 1. *For* $\mathbf{a} \in L_G^*$, $[G](\mathbf{a}) = [\lceil G \rceil(\mathbf{a})]$.

Proof. Use definitions 5, 8, and 9. ⊠

Formally, the notion of traces of G is based on the symbolic semantics of G, reflecting its use in symbolic analysis. While the standard definition of LTS semantics of programs is based on their concrete execution semantics (that corresponds to the concrete semantics of a GLAS), the intended correctness of Definition 10 follows from Theorem 1 proved next.

Definition 10. $Tr(G) \stackrel{\text{def}}{=} Tr(\lceil G \rceil)$.

We show that both the concrete and the symbolic semantics of G yield the same traces, i.e., $\lceil G \rceil$ does not introduce new traces, although several models of $\lfloor G \rfloor$ may collapse into a single state in $\lceil G \rceil$. We use the following technical lemma. Given a sequence $\mathbf{a}$ and an element a, we write $\mathbf{a} \cdot a$ for the extended sequence. The empty sequence is denoted by ϵ.

Lemma 2. *For $\mathbf{a} \in L^*_G$ and Σ_G-models M,*

$$M \models \lceil G \rceil(\mathbf{a}) \Leftrightarrow \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\mathbf{a}}_{\lfloor G \rfloor} M).$$

Proof. By induction over the length of $\mathbf{a}$. The base case, $\mathbf{a} = \epsilon$, holds trivially by $\{M \mid M \models \lceil G \rceil(\epsilon)\} = \mathbf{S}^0_{\lfloor G \rfloor} = \{M \mid \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\epsilon}_{\lfloor G \rfloor} M)\}$. Assume as IH (induction hypothesis) that the statement holds for $\mathbf{a}$, we prove it for $\mathbf{a} \cdot a$.

$$\begin{aligned}
M \models \lceil G \rceil(\mathbf{a} \cdot a) &\overset{(\text{def } 8)}{\Leftrightarrow} M \models SP_G(\lceil G \rceil(\mathbf{a}), a) \\
&\overset{(\text{def } 5)}{\Leftrightarrow} M \models \exists \Sigma'(\lceil G \rceil(\mathbf{a})' \wedge TR_G(\Sigma', a, \Sigma)) \\
&\Leftrightarrow \exists N(N \models \lceil G \rceil(\mathbf{a}), M \models \exists \Sigma'(P'_N \wedge TR_G(\Sigma', a, \Sigma))) \\
&\overset{(\text{def } 5)}{\Leftrightarrow} \exists N(N \models \lceil G \rceil(\mathbf{a}), M \models SP_G(P_N, a)) \\
&\overset{(\text{IH})}{\Leftrightarrow} \exists N\, \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\mathbf{a}}_{\lfloor G \rfloor} N, M \models SP_G(P_N, a)) \\
&\overset{(\text{def } 7)}{\Leftrightarrow} \exists N\, \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\mathbf{a}}_{\lfloor G \rfloor} N,\ N \xrightarrow{a}_{\lfloor G \rfloor} M) \\
&\Leftrightarrow \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\mathbf{a} \cdot a}_{\lfloor G \rfloor} M)
\end{aligned}$$

The statement follows by the induction principle. ⊠

The lemma implies the following theorem that is a fundamental property of the symbolic semantics. It justifies the whole approach presented in the paper and provides a symbolic generalization of the classical LTS determinization.

Theorem 1. $Tr(\lfloor G \rfloor) = Tr(\lceil G \rceil) = Tr([G])$.

Proof. $Tr(\lceil G \rceil)$ equals $\{\mathbf{a} \mid \{M \mid M \models \lceil G \rceil(\mathbf{a})\} \neq \emptyset\}$ that, by Lemma 2, equals $\{\mathbf{a} \mid \{M \mid \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}(M_0 \xrightarrow{\mathbf{a}}_{\lfloor G \rfloor} M)\} \neq \emptyset\}$ that is the definition of $Tr(\lfloor G \rfloor)$. The second equality follows from Lemma 1. ⊠

There is an important point about this choice of trace-style semantics. It is tailored for the case where internal choices of GLASs are *opaque*. Symbolic semantics plays an important role when we later define alternating simulation and conformance, where G may be nondeterministic, i.e., $\lfloor G \rfloor$ is nondeterministic, but where $\lceil G \rceil$ is used, which, by Theorem 1, does not change the intended trace semantics of G. Moreover, $\lceil G \rceil$ directly reflects the symbolic unfolding of the transition relation of a GLAS, that is fundamental in the construction of first-order assertions for reduction to symbolic analysis. Note that $\lceil G \rceil$ is, by definition, a deterministic LTS although $\lfloor G \rfloor$ may be nondeterministic, since the predicate transformer operates on the level of symbolic states. Symbolic states may be satisfied by more than one model in the case when the choice signature is nonempty or the initial condition is satisfied by more than one model.

Example 4. The Credits program in Example 2 is deterministic. The following is a trace of $G_{Credits}$:

$$(Req(0,3), Res(0,2), Req(2,1), Req(1,1), Res(2,0), Res(1,0)).$$

Intuitively, the trace describes a valid communication scenario between the client and the server (based on a sliding window protocol), where the client is able to use message ids based on credits granted earlier by the server.

The client sends a request with message id 0 and asks for 3 credits. The server responds to that message, granting the client 2 credits. Then the client sends two more requests to the server using the message ids 2 and 1, respectively. The server first responds to the first request (with id 2) and then to the second request (with id 1) not granting more credits in either response. Note that the client has now run out of message ids and cannot send more requests. ⊠

Definition of a GLAS G is motivated by separation of concerns. By providing the label predicate, the guard, the assignments, and the choice variables separately, several operations over GLASs, such as composition and quiescence extension discussed below, have simple and intuitive definitions. Note also that if $X_G = \emptyset$ and if $\mathbf{S}_{\lfloor G \rfloor}$ is a singleton set then G (i.e. $\lfloor G \rfloor$) is deterministic and there is a one-to-one mapping between the concrete and the symbolic semantics. A natural question that arises is if the definition of GLASs is general enough to describe arbitrary symbolic transition systems or if the assignment form limits the expressivity. The following example implies that the results of the paper carry over to arbitrary symbolic labeled transition systems that can be described by a predicate.

Example 5. Let $P(x, \ell, y)$ be an arbitrary predicate over $\{x{:}\sigma, \ell{:}\rho, y{:}\sigma\}$. Assume P specifies the LTS $\mathcal{L} = (\mathcal{U}^\sigma, \mathcal{U}^\sigma, \mathcal{U}^\rho, \{(x^M, \ell^M, y^M) \mid M \models P\})$. Let

$$G = (\{x\}, \{y\}, \ell, \mathit{true}, \mathit{true}, P, \{x := y\})$$

Then $TR_G \Leftrightarrow \exists y' \, (P' \wedge x = y') \Leftrightarrow P(x', \ell', x)$ and $\mathcal{L} \equiv \lfloor G \rfloor$. ⊠

3.1 GLAS Composition

The use of composition of model programs for numerous analysis tasks is the core theme of the textbook [31] and is also discussed at a more technical level in [47]. Composition of GLASs is a symbolic generalization of model program composition. The composition of two GLASs has practical benefits. The size of the composed GLAS is *linear* in the sizes of the individual GLASs, while preserving the intended semantics, as shown in Theorem 2. Moreover, standard DFS algorithms can be developed for special cases of GLASs that use an external solver to incrementally eliminate unsatisfiable predicates that arise when forming a conjunction of guards.

Definition 11. Let $G_i = (\Sigma_i, X_i, \ell, \iota_i, \alpha_i, \gamma_i, \{z := u_z\}_{z\in\Sigma_i})$, for $i \in I$, be GLASs with disjoint internal signatures. We also assume that I is non-empty. The *composition of G_i for $i \in I$*, is the GLAS

$$\prod_{i\in I} G_i \stackrel{\text{def}}{=} (\bigcup_{i\in I}\Sigma_i, \bigcup_{i\in I} X_i, \ell, \bigwedge_{i\in I}\iota_i, \bigvee_{i\in I}\alpha_i, \bigwedge_{i\in I}(\alpha_i \Rightarrow \gamma_i), \bigcup_{i\in I}\{z := Ite(\alpha_i, u_z, z)\}_{z\in\Sigma_i})$$

We abbreviate $\prod_{i\in I} G_i$ by $\prod_I G_i$ and for $\prod_{\{1,2\}} G_i$ we write $G_1 \times G_2$. Note that $\prod_I G_i$ is indeed well-defined as a GLAS. In particular, $\iota_{\prod_I G_i}$ is satisfiable because all the individual initial conditions are satisfiable and do not share free variables. The other side conditions in Definition 1 hold similarly. The following technical lemma is used below. Let G_i, for $i \in I$, be as above.

Lemma 3. *Let $G = \prod_I G_i$. Assume $\alpha_i \Leftrightarrow \alpha_j$ for $i, j \in I$. Let P_i be a predicate over Σ_i for $i \in I$ and let $a \in L_G$. Then $SP_G(\bigwedge_{i\in I} P_i, a) \Leftrightarrow \bigwedge_{i\in I} SP_{G_i}(P_i, a)$.*

Proof. Let $\Sigma = \Sigma_G$ and $X = X_G$. We first show

$$TR_G(\Sigma', \ell', \Sigma) \Leftrightarrow \bigwedge_{i\in I} TR_{G_i}(\Sigma'_i, \ell', \Sigma_i) \tag{2}$$

as follows:

$$\begin{aligned}
TR_G(\Sigma', \ell', \Sigma) &\stackrel{(\text{def } 5)}{\Leftrightarrow} \alpha'_G \wedge \exists X' \, (\gamma'_G \bigwedge_{z\in\Sigma} z = u'_z) \\
&\stackrel{(\text{def } 11)}{\Leftrightarrow} (\bigvee_{i\in I} \alpha'_i) \wedge \exists X' \, (\bigwedge_{i\in I}(\alpha'_i \Rightarrow \gamma'_i) \bigwedge_{i\in I, z\in\Sigma_i} z = Ite(\alpha'_i, u'_z, z')) \\
&\stackrel{(\alpha_i \Leftrightarrow \alpha_j)}{\Leftrightarrow} (\bigwedge_{i\in I} \alpha'_i) \wedge \exists X' \, (\bigwedge_{i\in I}(\alpha'_i \Rightarrow \gamma'_i) \bigwedge_{i\in I, z\in\Sigma_i} z = Ite(\alpha'_i, u'_z, z')) \\
&\Leftrightarrow (\bigwedge_{i\in I} \alpha'_i) \wedge \exists X' \, (\bigwedge_{i\in I}(\gamma'_i \bigwedge_{z\in\Sigma_i} z = u'_z)) \\
&\stackrel{(X'_i\text{s disjoint})}{\Leftrightarrow} \bigwedge_{i\in I}(\alpha'_i \wedge \exists X'_i \, (\gamma'_i \wedge \bigwedge_{z\in\Sigma_i} z = u'_z))) \\
&\stackrel{(\text{def } 5)}{\Leftrightarrow} \bigwedge_{i\in I} TR_{G_i}(\Sigma'_i, \ell', \Sigma_i)
\end{aligned}$$

Now:

$$\begin{aligned}
SP_G(\bigwedge_{i\in I} P_i, a) &\stackrel{(\text{def } 5)}{\Leftrightarrow} \exists \Sigma' \, (\bigwedge_{i\in I} P'_i \wedge TR_G(\Sigma', a, \Sigma)) \\
&\stackrel{(\text{by } (2))}{\Leftrightarrow} \exists \Sigma' \, (\bigwedge_{i\in I}(P'_i \wedge TR_{G_i}(\Sigma'_i, a, \Sigma_i))) \\
&\stackrel{(\Sigma'_i\text{s disjoint})}{\Leftrightarrow} \bigwedge_{i\in I} \exists \Sigma'_i (P'_i \wedge TR_{G_i}(\Sigma'_i, a, \Sigma_i)) \\
&\stackrel{(\text{def } 5)}{\Leftrightarrow} \bigwedge_{i\in I} SP_{G_i}(P_i, a)
\end{aligned}$$

that proves the lemma. ⊠

One can show that composition of GLASs respects the standard parallel synchronous composition of LTSs with the interleaving semantics of unshared labels. Here we prove the special case of all labels being shared, i.e. $L_{G_i} = L_{G_j}$ for $i, j \in I$, that is used below.

Theorem 2. *Let $G = \prod_I G_i$. Assume $\alpha_i \Leftrightarrow \alpha_j$ for $i, j \in I$.*
(i) *For all* $\mathbf{a}$, $\lceil G \rceil(\mathbf{a}) \Leftrightarrow \bigwedge_{i \in I} \lceil G_i \rceil(\mathbf{a})$.
(ii) $Tr(G) = \bigcap_{i \in I} Tr(G_i)$.

Proof. We prove (i) by induction over $\mathbf{a}$. The base case holds trivially since $\lceil G \rceil(\epsilon) = \bigwedge_{i \in I} \imath_i = \bigwedge_{i \in I} \lceil G_i \rceil(\epsilon)$. Assume (i) holds for $\mathbf{a}$; we prove (i) for $\mathbf{a} \cdot a$:

$$\begin{array}{rl} \lceil G \rceil(\mathbf{a} \cdot a) \overset{(\text{def } 8)}{\Leftrightarrow} SP_G(\lceil G \rceil(\mathbf{a}), a) & \overset{(\text{IH})}{\Leftrightarrow} \; SP_G(\bigwedge_I \lceil G_i \rceil(\mathbf{a}), a) \\ & \overset{(\text{lemma 3})}{\Leftrightarrow} \; \bigwedge_I SP_{G_i}(\lceil G_i \rceil(\mathbf{a}), a) \\ & \overset{(\text{def } 8)}{\Leftrightarrow} \; \bigwedge_I \lceil G_i \rceil(\mathbf{a} \cdot a) \end{array}$$

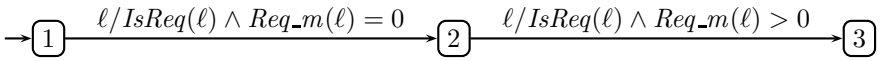
Statement (i) follows by the induction principle. Now:

$$\begin{array}{rcl} Tr(G) = \{\mathbf{a} \mid \lceil G \rceil(\mathbf{a}) \neq \mathit{false}\} & \overset{(\text{by (i)})}{=} & \{\mathbf{a} \mid \bigwedge_I \lceil G_i \rceil(\mathbf{a}) \text{ is satisfiable}\} \\ & \overset{(\Sigma_i\text{'s disjoint})}{=} & \{\mathbf{a} \mid \forall i \in I(\lceil G_i \rceil(\mathbf{a}) \neq \mathit{false})\} \\ & = & \{\mathbf{a} \mid \forall i \in I(\mathbf{a} \in Tr(G_i))\} \end{array}$$

that proves (ii). ⊠

Example 6. Consider the composition $G = G_{Credits} \times G_A$ with $G_{Credits}$ and G_A from examples 2 and 1, respectively. The traces of G are the traces of both $G_{Credits}$ and G_A, i.e., the traces that conform to the *Credits* specification while restricted to the scenarios described by A. For example, the trace illustrated in Example 4 is therefore *not* a trace of G. ⊠

Example 7. Consider the FSM B:



B describes a scenario where only two requests occur, the first request uses message id 0 and the second request uses a message id > 0. Suppose that the responses are irrelevant rather than disabled, i.e., $\alpha_{G_B} = \mathit{IsReq}(\ell_{G_B})$. In this case, the composition $G_{Credits} \times G_B$ represents the intended slice of $G_{Credits}$ where responses are considered as internal actions of $G_{Credits}$ from the point of view of B and are unconstrained by B (i.e., viewed as "self-loops" in B). ⊠

We will use the following *closed* composition of GLASs (in Theorem 7) that *disables* non-shared labels.

Definition 12. Given a GLAS G and a predicate β over $\{\ell_G\}$, let $G \lfloor \beta$ denote the modification of G such that $\alpha_{G \lfloor \beta} \overset{\text{def}}{=} \alpha_G \vee \beta$ and $\gamma_{G \lfloor \beta} \overset{\text{def}}{=} \gamma_G \wedge (\beta \Rightarrow \alpha_G)$. The *closed composition* of GLASs G and H with disjoint internal signatures and $\ell_G = \ell_H$ is the GLAS $G \otimes H \overset{\text{def}}{=} G \lfloor \alpha_H \times H \lfloor \alpha_G$.

Observe that if $\alpha \Rightarrow \alpha_G$ (e.g. when $\alpha = \mathit{false}$) then $G\lfloor\alpha$ is a logical noop. When $\alpha = \mathit{true}$ then $L_{G\lfloor\alpha} = \mathcal{U}^{sort(\ell_G)}$ and $G\lfloor\alpha$ strengthens the guard of G with α_G. In the general case, $L_{G\lfloor\alpha}$ extends L_G with labels accepted by α, while the guard $\gamma_{G\lfloor\alpha}$ is disabled with respect to any new labels (if any). We use the following corollary of Theorem 2.

Corollary 1. *Let G and H have disjoint internal signatures and $\ell = \ell_G = \ell_H$.*
(i) *For all* $\mathbf{a}$, $\lceil G{\otimes}H\rceil(\mathbf{a}) \Leftrightarrow \lceil G\rceil(\mathbf{a}) \wedge \lceil H\rceil(\mathbf{a})$.
(ii) $Tr(G{\otimes}H) = Tr(G) \cap Tr(H)$.

Proof. We use (3), i.e., for all ℓ, ℓ is enabled in $G\lfloor\alpha_H$ and $H\lfloor\alpha_G$ iff ℓ is enabled in G and H:

$$((\alpha_{G\lfloor\alpha_H} \wedge \gamma_{G\lfloor\alpha_H}) \wedge (\alpha_{H\lfloor\alpha_G} \wedge \gamma_{H\lfloor\alpha_G})) \Leftrightarrow ((\alpha_G \wedge \gamma_G) \wedge (\alpha_H \wedge \gamma_H)) \qquad (3)$$

that follows by expaning the l.h.s. using Definition 12 and using standard logic:

$$\left(\begin{array}{l}((\alpha_G \vee \alpha_H) \wedge \gamma_G \wedge (\alpha_H \Rightarrow \alpha_G))\wedge\\ ((\alpha_H \vee \alpha_G) \wedge \gamma_H \wedge (\alpha_G \Rightarrow \alpha_H))\end{array}\right) \Leftrightarrow ((\alpha_G \vee \alpha_H) \wedge (\alpha_H \Leftrightarrow \alpha_G) \wedge \gamma_G \wedge \gamma_H)$$

To prove (i) let $\mathbf{a}$ be any finite sequence over $\mathcal{U}^{sort(\ell)}$:

$$\begin{array}{rl}\lceil G{\otimes}H\rceil(\mathbf{a}) & \overset{(\text{def } 12)}{\Leftrightarrow} \lceil G\lfloor\alpha_H\rceil{\times}H\lfloor\alpha_G(\mathbf{a})\\ & \overset{(\text{thm } 2(\text{i}))}{\Leftrightarrow} \lceil G\lfloor\alpha_H\rceil(\mathbf{a}) \wedge \lceil H\lfloor\alpha_G\rceil(\mathbf{a})\\ & \overset{(\text{by } (3))}{\Leftrightarrow} \lceil G\rceil(\mathbf{a}) \wedge \lceil H\rceil(\mathbf{a})\end{array}$$

Statement (ii) follows from Theorem 2(ii) and that for any GLAS G and any predicate α over ℓ_G, $Tr(G) = Tr(G\lfloor\alpha)$. ⊠

The label variable provides a way to *abstract* behaviors. In order to expose the values of all model variables $\{x_i\}_{i<k}$, the label variable can be declared as a tuple of the sort of the model variables and the guard can be defined as $\bigwedge_{i<n} \pi_i(\ell) = x_i$.

4 I/O GLAS

Here we consider GLASs where the labels are divided into input and output labels that describe reactive or open system behavior.

Definition 13. An *i/o-GLAS* G is an extension (H, α^{out}) of a GLAS H where α^{out} is a formula, called the *output label predicate*, such that $\alpha^{\text{out}} \Rightarrow \alpha_H$.

In the corresponding i/o LTS the labels are separated so that L_G^{out} is the set of all labels that satisfy α_G^{out} and L_G^{in} is the set of all labels that satisfy $\alpha_G^{\text{in}} \overset{\text{def}}{=} \alpha_G \wedge \neg\alpha_G^{\text{out}}$. We say GLAS (LTS) to also mean i/o-GLAS (i/o LTS) and let the context determine whether the labels are separated into input and output labels.

We define $EN_G \stackrel{\text{def}}{=} \alpha_G \wedge \exists X_G(\gamma_G)$ as the *enabling condition of* G, $O_G \stackrel{\text{def}}{=} \alpha_G^{\text{out}} \wedge EN_G$ as the *output enabling condition of* G, and $I_G \stackrel{\text{def}}{=} \alpha_G^{\text{in}} \wedge EN_G$ as the *input enabling condition of* G. Note that the input and output enabling conditions are mutually exclusive predicates over $\Sigma_G \cup \{\ell\}$. They characterize the states where G has successor states for an input, respectively output, label. The definition is consistent with the usual enabling condition on transition relations. In particular, $EN_G(\ell, \Sigma)$ is derived from $\alpha_G \wedge \exists \Sigma'\ TR_G(\Sigma, \ell, \Sigma')$, which expands to $\alpha_G \wedge \exists \Sigma' \exists X_G\,(\gamma_G \wedge \bigwedge_{z' \in \Sigma'} z' = u_z)$, and simplifies to $\alpha_G \wedge \exists X_G\,(\gamma_G)$.

Example 8. Consider the *Credits* program and assume that *Req* is marked as an input-action and *Res* is marked as an output-action. The output label predicate α^{out} is a disjunction over all cases of action labels in the AsmL program that are marked as output-actions, i.e, in this case α^{out} is $IsRes(\ell)$. ⊠

4.1 Quiescence

When dealing with formal notions of conformance, in particular *ioco* [39], an important aspect is how to deal with *quiescence*, that is a special output label, usually denoted by δ, indicating absence of other enabled output labels in a given state. An LTS can be extended to include δ as a new output label [39]:

Definition 14. Let $\mathcal{L}$ be an LTS and $\delta \notin L_{\mathcal{L}}$. Then $\mathcal{L}^\delta$ is the extension of $\mathcal{L}$ where $L^{\text{out}}_{\mathcal{L}^\delta} = L^{\text{out}}_{\mathcal{L}} \cup \{\delta\}$ and $S \stackrel{\delta}{\rightarrow}_{\mathcal{L}^\delta} S$ iff for all $a \in L^{\text{out}}_{\mathcal{L}}$, $S \stackrel{a}{\not\rightarrow}_{\mathcal{L}}$.

We define a corresponding symbolic extension for GLASs.

Definition 15. For $G = (\Sigma, X, \ell, \imath, \alpha, \gamma, \{z := u_z\}_{z \in \Sigma}, \alpha^{\text{out}})$, $\delta \in \mathcal{U}^{sort(\ell)} \setminus L_G$:

$$G^\delta \stackrel{\text{def}}{=} (\ \Sigma,\ X,\ \ell,\ \imath,\ \alpha \vee \ell = \delta,\ Ite(\ell = \delta, \neg\exists \ell\, X\,(\alpha^{\text{out}} \wedge \gamma), \gamma),\\ \{z := Ite(\ell = \delta, z, u_z)\}_{z \in \Sigma},\ \alpha^{\text{out}} \vee \ell = \delta)$$

Thus, in G^δ there is a new output label δ and $M \stackrel{\delta}{\rightarrow}_{\lfloor G^\delta \rfloor}$ if and only if for all $a \in L^{\text{out}}_G$, $M \stackrel{a}{\not\rightarrow}_{\lfloor G \rfloor}$. For example, in the case of the $\mathbb{L}$ sort used in Example 2, assume that $\delta{:}\mathbb{L}$ is an additional constructor, i.e., $\delta \in \mathcal{U}^{\mathbb{L}}$. The intended meaning of G^δ is made precise by the following theorem that says that the symbolic extension precisely captures the intended suspension trace semantics [39] of $\lfloor G \rfloor$.

Theorem 3. (i) $\lfloor G^\delta \rfloor = \lfloor G \rfloor^\delta$. (ii) $Tr([G^\delta]) = Tr(\lfloor G \rfloor^\delta)$.

Proof. (i) follows from definitions 14,15. (ii) uses (i) and Theorem 1. ⊠

One can also show that $Tr([G]^\delta) \subseteq Tr(G^\delta)$. However, $Tr(G^\delta) \neq Tr([G]^\delta)$ as illustrated by the following example.

Example 9. The example is derived from a standard example that is used to illustrate properties of quiescence during determinization of non-deterministic LTSs [39, Figure 6]. The GLAS G is represented below by an FSM where there

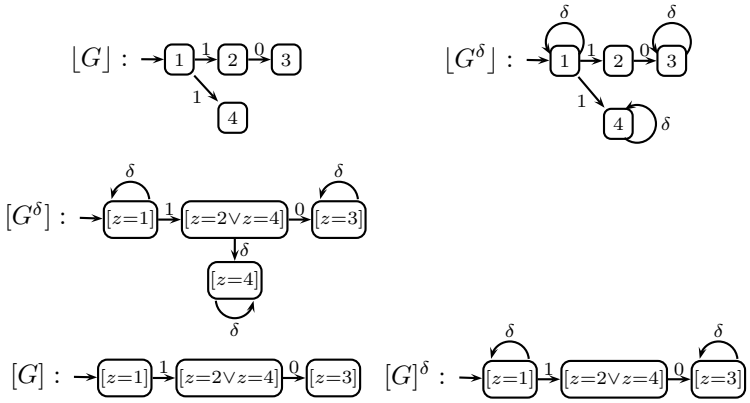
is a single input label 1 and a single output label 0. We assume the following representation for G:

$$G = (\ \{z{:}\mathbb{Z}\},\ \{x{:}\mathbb{B}\},\ \ell{:}\mathbb{Z},\ z = 1,\ 0 \le \ell \le 1,\ \ell = 0 \Leftrightarrow z = 2,\\ \{z := \mathit{Ite}(z = 1, \mathit{Ite}(x, 2, 4), 3)\},\ \ell = 0\ )$$

G^δ is the following GLAS where we have simplified γ_{G^δ} by using that the formula $\neg\exists \ell\, x\, (\ell = 0 \wedge (\ell = 0 \Leftrightarrow z = 2))$ is equivalent to $z \neq 2$. (Below let, e.g. $\delta = 42$),

$$G^\delta = (\ \{z{:}\mathbb{Z}\},\ \{x{:}\mathbb{B}\},\ \ell{:}\mathbb{Z},\ z = 1,\ 0 \le \ell \le 1 \vee \ell = \delta,\\ \mathit{Ite}(\ell = \delta, z \neq 2, \ell = 0 \Leftrightarrow z = 2),\\ \{z := \mathit{Ite}(\ell = \delta, z, \mathit{Ite}(z = 1, \mathit{Ite}(x, 2, 4), 3))\},\ \ell = 0 \vee \ell = \delta\ )$$

We can illustrate the GLASs as follows:



Thus $\mathit{Tr}([G^\delta]) \neq \mathit{Tr}([G]^\delta)$. In order to see how we obtained the equivalent state predicates and the transitions of $[G^\delta]$, we illustrate the use of the definitions explicitly for the case $[\imath_{G^\delta}] \xrightarrow{1}_{[G^\delta]} [SP_{G^\delta}(\imath_{G^\delta}, 1)]$:

$$\begin{aligned}
SP_{G^\delta}(\imath_{G^\delta}, 1) &\Leftrightarrow \exists z'\, (\imath'_{G^\delta} \wedge TR_{G^\delta}(z', 1, z))\\
&\Leftrightarrow \exists z'\, (z' = 1 \wedge (0 \le 1 \le 1 \vee 1 = \delta) \wedge\\
&\qquad \exists x'\, (\mathit{Ite}(1 = \delta, z' \neq 2, 1 = 0 \Leftrightarrow z' = 3) \wedge\\
&\qquad\quad z = \mathit{Ite}(1 = \delta, z', \mathit{Ite}(z' = 1, \mathit{Ite}(x', 2, 4), 3))))\\
&\Leftrightarrow \exists z'\, (z' = 1 \wedge (1 = 0 \Leftrightarrow z' = 3) \wedge\\
&\qquad \exists x'\, (z = \mathit{Ite}(z' = 1, \mathit{Ite}(x', 2, 4), 3)))\\
&\Leftrightarrow \exists x'\, (z = \mathit{Ite}(x', 2, 4))\\
&\Leftrightarrow z = 2 \vee z = 4
\end{aligned}$$

and the case $[z = 2 \vee z = 4] \xrightarrow{\delta}_{[G^\delta]} [SP_{G^\delta}(z = 2 \vee z = 4, \delta)]$:

$$\begin{aligned}
SP_{G^\delta}(z = 2 \vee z = 4, \delta) &\Leftrightarrow \exists z'\, ((z' = 2 \vee z' = 4) \wedge TR_{G^\delta}(z', \delta, z))\\
&\Leftrightarrow \exists z'\, ((z' = 2 \vee z' = 4) \wedge (0 \le \delta \le 1 \vee \delta = \delta) \wedge
\end{aligned}$$

$$\begin{aligned}
&\exists x'\,(\mathit{Ite}(\delta = \delta, z' \neq 2, \delta = 0 \Leftrightarrow z' = 3) \wedge \\
&\qquad z = \mathit{Ite}(\delta = \delta, z', \mathit{Ite}(z' = 1, \mathit{Ite}(x', 2, 4), 3))))\\
\Leftrightarrow\ &\exists z'\,((z' = 2 \vee z' = 4) \wedge z' \neq 2 \wedge z = z')\\
\Leftrightarrow\ &z = 4
\end{aligned}$$

The remaining state predicates and transitions are obtained similarly. ⊠

Example 10. Consider $G = G_{Credits}$ from Example 2. The formula that defines absence of outputs in G, $\neg\exists\ell\, X_G(\alpha_G^{\text{out}} \wedge \gamma_G)$, is, after simplifications, equivalent to the formula $msgs = \varepsilon$. Intuitively, there should not be a response from the server, i.e. the server must be quiescent, if there is no pending request from the client, i.e., δ is enabled in any model of $\lfloor G^\delta \rfloor$ where *msgs* is empty. ⊠

4.2 I/O refinement

We define a notion of conformance between two GLASs that is based on *alternating simulation* [4] between two LTSs and show below that this notion of conformance coincides with *ioco* for GLASs.

Let $\mathcal{L}_i = (\mathbf{S}_i, \{S_i^0\}, L_i, L_i^{\text{in}}, L_i^{\text{out}}, T_i)$, for $i = 1, 2$, be deterministic LTSs.[2] The intuition behind the following definition is that $\mathcal{L}_1$ can only make outputs that $\mathcal{L}_2$ can make, and $\mathcal{L}_2$ can only make inputs that $\mathcal{L}_1$ can make.

Definition 16. $\mathcal{L}_1$ *i/o-refines* $\mathcal{L}_2$, $\mathcal{L}_1 \preceq \mathcal{L}_2$, iff there exists an alternating simulation ρ from $\mathcal{L}_1$ to $\mathcal{L}_2$ such that $(S_1^0, S_2^0) \in \rho$, where an *alternating simulation from* $\mathcal{L}_1$ *to* $\mathcal{L}_2$ is a relation $\rho \subseteq \mathbf{S}_1 \times \mathbf{S}_2$ such that, for all $(S_1, S_2) \in \rho$

$$\begin{aligned}
&\forall o \in L_1^{\text{out}}(S_1 \xrightarrow{o}_{\mathcal{L}_1} S_1' \Rightarrow \exists S_2'(S_2 \xrightarrow{o}_{\mathcal{L}_2} S_2' \wedge (S_1', S_2') \in \rho))\\
&\forall i \in L_2^{\text{in}}(S_2 \xrightarrow{i}_{\mathcal{L}_2} S_2' \Rightarrow \exists S_1'(S_1 \xrightarrow{i}_{\mathcal{L}_1} S_1' \wedge (S_1', S_2') \in \rho))
\end{aligned}$$

Note that, since $\mathcal{L}_1$ and $\mathcal{L}_2$ are deterministic, the choice of S_1' and S_2' above is unique, i.e., $S \rightarrow_{\mathcal{L}} S'$ behaves as a function. When considering i/o-refinement it is natural to assume that the LTSs agree on what is an input label and what is an output label.

Definition 17. $\mathcal{L}_1$ is *i/o-compatible* with $\mathcal{L}_2$ if $L_1^{\text{out}} \subseteq L_2^{\text{out}}$ and $L_2^{\text{in}} \subseteq L_1^{\text{in}}$.

We will impose the i/o-compatibility requirement, both, to simplify the technical presentation as well as the intuition. We lift the definitions to GLASs: G is *i/o-compatible* with H if $\lceil G \rceil$ is *i/o-compatible* with $\lceil H \rceil$; and $G \preceq H$ if $\lceil G \rceil \preceq \lceil H \rceil$. Note that G ($\lfloor G \rfloor$) and H ($\lfloor H \rfloor$) are *not* assumed be deterministic.

It follows from definitions that $\lceil G \rceil \preceq \lceil H \rceil$ iff $[G] \preceq [H]$. The latter view is often more natural, as it avoids duplicate equivalent states but the former view maps more directly to symbolic analysis that works with concrete formulas.

Definition 16 is consistent with [18]. In particular, several foundational properties of $\preceq$ (like reflexivity and transitivity, i.e., $\preceq$ is a preorder) are established in [18] that show that $\preceq$ is a suitable refinement relation.

[2] Deterministic LTSs are called *interface automata* in [19].

Example 11. Consider two GLASs *Spec* and *Impl* where ℓ:$\mathbb{B}$ and α^{out} is $\neg\ell$.

$$\lceil \mathit{Spec} \rceil = (\{S_1\}, \{S_1\}, \mathcal{U}^{\mathbb{B}}, \{\mathit{true}\}, \{\mathit{false}\}, \{(S_1, \mathit{false}, S_1)\})$$
$$\lceil \mathit{Impl} \rceil = (\{S_2\}, \{S_2\}, \mathcal{U}^{\mathbb{B}}, \{\mathit{true}\}, \{\mathit{false}\}, \{(S_2, \mathit{true}, S_2)\})$$

It is easy to see that $\mathit{Impl} \preceq \mathit{Spec}$ and $\mathit{Spec} \not\preceq \mathit{Impl}$. ⊠

A useful characterization of i/o-refinement uses counter-examples.

Definition 18. A sequence $\mathbf{a} \cdot a$ is a *witness of* $\mathcal{L}_1 \not\preceq \mathcal{L}_2$ or a *counterexample of* $\mathcal{L}_1 \preceq \mathcal{L}_2$ if $\mathbf{a} \in \mathit{Tr}(\mathcal{L}_1) \cap \mathit{Tr}(\mathcal{L}_2)$ and either

- $a \in L_2^{\text{in}}$ and $\mathbf{a} \cdot a \in \mathit{Tr}(\mathcal{L}_2) \setminus \mathit{Tr}(\mathcal{L}_1)$, or
- $a \in L_1^{\text{out}}$ and $\mathbf{a} \cdot a \in \mathit{Tr}(\mathcal{L}_1) \setminus \mathit{Tr}(\mathcal{L}_2)$.

The intuition in the definition of $\mathcal{L}_1 \preceq \mathcal{L}_2$ is that $\mathcal{L}_1$ is the implementation and $\mathcal{L}_2$ is the specification. The first case means that $\mathcal{L}_2$ (specification) produces an input that is not allowed in $\mathcal{L}_1$ (implementation). Converesely, the second case means that $\mathcal{L}_1$ produces an output that is not allowed according to $\mathcal{L}_2$, e.g., the implementation produces an output that does not conform to the specification.

For example, the (singleton) sequence *true* is a counterexample of $\lceil \mathit{Spec} \rceil \preceq \lceil \mathit{Impl} \rceil$ in Example 11. The following lemma justifies Definition 18.

Lemma 4. $\mathcal{L}_1 \preceq \mathcal{L}_2 \iff \mathcal{L}_1 \preceq \mathcal{L}_2$ *has no counterexamples.*

Proof. Let $\mathit{Tr} = \mathit{Tr}(\mathcal{L}_1) \cap \mathit{Tr}(\mathcal{L}_2)$. We need to show

$$\mathcal{L}_1 \preceq \mathcal{L}_2 \iff \mathit{RHS}$$

where *RHS* is: for all $\mathbf{a} \in \mathit{Tr}$: for all $b \in L_1^{\text{out}}$ if b is enabled in $\mathcal{L}_1(\mathbf{a})$ then b is enabled in $\mathcal{L}_2(\mathbf{a})$; and for all $b \in L_2^{\text{in}}$ if b is enabled in $\mathcal{L}_2(\mathbf{a})$ then b is enabled in $\mathcal{L}_1(\mathbf{a})$.

($\Longleftarrow$): Assume *RHS*. Let

$$\rho = \{(\mathcal{L}_1(\mathbf{a}), \mathcal{L}_2(\mathbf{a})) \mid \mathbf{a} \in \mathit{Tr}\}.$$

Clearly $(S_1^0, S_2^0) \in \rho$ since the empty trace is in Tr. We show that ρ is an alternating simulation from $\mathcal{L}_1$ to $\mathcal{L}_2$. Fix $(S_1, S_2) \in \rho$. By definition of ρ there is fixed $\mathbf{a} \in \mathit{Tr}$ such that $S_1 = \mathcal{L}_1(\mathbf{a})$ and $S_2 = \mathcal{L}_2(\mathbf{a})$. Consider any $b \in L_1^{\text{out}}$ such that b is enabled in S_1. It follows from *RHS* that b is enabled in S_2. Thus, $\mathbf{a} \cdot b \in \mathit{Tr}$ and, by definition of ρ, $(\mathcal{L}_1(\mathbf{a} \cdot b), \mathcal{L}_2(\mathbf{a} \cdot b)) \in \rho$. Symmetrical argument is used for $b \in L_2^{\text{in}}$. Thus $\mathcal{L}_1 \preceq \mathcal{L}_2$.

($\Longrightarrow$): Assume that $\mathcal{L}_1 \preceq \mathcal{L}_2$. Consider a fixed $\mathbf{a} \in \mathit{Tr}$. We show *RHS*. Let ρ be an alternating simulation from $\mathcal{L}_1$ to $\mathcal{L}_2$ such that $(S_1^0, S_2^0) \in \rho$. It follows from the definition of alternating simulation and $\mathbf{a} \in \mathit{Tr}$ that $(\mathcal{L}_1(\mathbf{a}), \mathcal{L}_2(\mathbf{a})) \in \rho$. Consider $b \in L_1^{\text{out}}$ such that b is enabled in $\mathcal{L}_1(\mathbf{a})$. Then b is enabled in $\mathcal{L}_2(\mathbf{a})$ since ρ is an alternating simulation. Symmetrical argument is used for $b \in L_2^{\text{in}}$. Thus, *RHS* holds. ⊠

For symbolic analysis, we are interested in the approximations of i/o-refinement that hold for a given upper length bound on traces.

Definition 19. $\mathcal{L}_1 \preceq_n \mathcal{L}_2 \stackrel{\text{def}}{=} \mathcal{L}_1 \preceq \mathcal{L}_2$ has no conterexamples of length $\leq n$.

It follows directly from Lemma 4 that $\mathcal{L}_1 \preceq \mathcal{L}_2$ iff $\mathcal{L}_1 \preceq_n \mathcal{L}_2$ for all $n > 0$. For example, *Spec* $\not\preceq_1$ *Impl* in Example 11.

4.3 Relation to ioco

The conformance relation *ioco* [39] is used for testing reactive systems, it stands for **i**nput-**o**utput **co**nformance. There are several variations of *ioco*, here we consider basic *ioco*. An LTS $\mathcal{L}$ is *input-enabled* if in all states in $\mathcal{L}$ that are reachable from the initial state, all input-labels are enabled.[3] The following definition of *ioco* is consistent with the definition in [39].

Definition 20. Let $\mathcal{L}$ be an LTS and $\mathcal{M}$ an input-enabled LTS. $\mathcal{M}$ *ioco* $\mathcal{L}$ iff, for all $\mathbf{a} \in Tr(\mathcal{L})$ and output-labels a, if $\mathbf{a} \cdot a \in Tr(\mathcal{M})$ then $\mathbf{a} \cdot a \in Tr(\mathcal{L})$.

Note that G and H are *not* required to be deterministic in Theorem 4.

Theorem 4. *If* $\lfloor G \rfloor$ *is input-enabled then* $\lfloor G \rfloor$ *ioco* $\lfloor H \rfloor \Longleftrightarrow G \preceq H$.

Proof. Assume $\lfloor G \rfloor$ is input-enabled. Thus $\lceil G \rceil$ is input-enabled by Lemma 2.

($\Longrightarrow$): Assume $G \not\preceq H$. We show that $\lfloor G \rfloor$ *ioco* $\lfloor H \rfloor$ does not hold. From Definition 16 follows that there exists a trace $\mathbf{a}$ of $\lceil G \rceil$ and $\lceil H \rceil$ and a label a such that either

1. a is a output-label that is enabled in $\lceil G \rceil(\mathbf{a})$ but not enabled in $\lceil H \rceil(\mathbf{a})$, or
2. a is a input-label that is enabled in $\lceil H \rceil(\mathbf{a})$ but not enabled in $\lceil G \rceil(\mathbf{a})$.

The second case cannot be true since $\lceil G \rceil$ is input-enabled. So there exists a trace $\mathbf{a} \in Tr(\lceil H \rceil)$ and an output-label a such that $\mathbf{a} \cdot a \in Tr(\lceil G \rceil)$ but $\mathbf{a} \cdot a \notin Tr(\lceil H \rceil)$. By Theorem 1, $Tr(\lceil G \rceil) = Tr(\lfloor G \rfloor)$ and $Tr(\lceil H \rceil) = Tr(\lfloor H \rfloor)$. Thus $\lfloor G \rfloor$ *ioco* $\lfloor H \rfloor$ is not true.

($\Longleftarrow$): Assume $\lfloor G \rfloor$ *ioco* $\lfloor H \rfloor$ is not true. We show that $G \not\preceq H$. From Definition 20 follows that there exists a trace $\mathbf{a} \in Tr(\lfloor H \rfloor)$ and an output-label a such that $\mathbf{a} \cdot a \in Tr(\lfloor G \rfloor)$ but $\mathbf{a} \cdot a \notin Tr(\lfloor H \rfloor)$. Now use Theorem 1 and Lemma 4. ⊠

We also get the following corollary from Theorem 4 and Theorem 3 that considers the suspension traces of LTSs.

Corollary 2. *If* $\lfloor G \rfloor$ *is input-enabled then* $\lfloor G \rfloor^\delta$ *ioco* $\lfloor H \rfloor^\delta \Longleftrightarrow G^\delta \preceq H^\delta$.

[3] Such LTSs are called *input-output transition systems* in [39].

4.4 Bounded Non-Conformance

We are interested in the following decision problem. For GLASs G and H, a *counterexample of* $G \preceq H$ is a counterexample of $\lceil G \rceil \preceq \lceil H \rceil$ and we let $G \preceq_n H \stackrel{\text{def}}{=} \lceil G \rceil \preceq_n \lceil H \rceil$.

Definition 21. *Bounded Non-Conformance* or *BNC* is the problem of deciding if $G \not\preceq_n H$, for given G, H and $n > 0$, and finding a witness of $G \not\preceq_n H$.

The general case of BNC can be mapped to deciding unsatisfiability of a bounded alternating simulation formula $ASIM(\imath_G, \imath_H, n)$ that is defined as follows by induction over n. The correctness of the definition, i.e., that $ASIM(\imath_G, \imath_H, n)$ is true if and only if $G \preceq_n H$ holds, follows by induction over n from the definition of i/o-refinement (Definition 16). Note that $ASIM(\imath_G, \imath_H, n)$ is a closed formula that, in general, contains n alternations of universal and existential quantifiers over model variables.

$$
\begin{aligned}
ASIM(P, Q, 0) &\stackrel{\text{def}}{=} \mathit{true};\\
ASIM(P, Q, n+1) &\stackrel{\text{def}}{=} \forall \ell_n (\exists \Sigma_G (P \wedge O_G) \Rightarrow (\exists \Sigma_H (Q \wedge O_H) \ \wedge \\
&\qquad ASIM(SP_G(P, \ell_n), SP_H(Q, \ell_n), n))) \ \wedge \\
&\quad \forall \ell_n (\exists \Sigma_H (Q \wedge I_H) \Rightarrow (\exists \Sigma_G (P \wedge I_G) \ \wedge \\
&\qquad ASIM(SP_G(P, \ell_n), SP_H(Q, \ell_n), n)))
\end{aligned}
$$

We can simplify $ASIM(P, Q, n+1)$ to an equivalent and more pleasing form. To see this, first observe by using de-Morgan distribution laws it is equivalent to:

$$\forall \ell_n (\exists \Sigma_G (P \wedge O_G) \Rightarrow \exists \Sigma_H (Q \wedge O_H)) \ \wedge \tag{4}$$

$$\forall \ell_n (\exists \Sigma_H (Q \wedge I_H) \Rightarrow \exists \Sigma_G (P \wedge I_G)) \ \wedge \tag{5}$$

$$\forall \ell_n \left(\begin{array}{r} (\exists \Sigma_G (P \wedge O_G)) \ \vee \ (\exists \Sigma_H (Q \wedge I_H)) \ \Rightarrow \\ ASIM(SP_G(P, \ell_n), SP_H(Q, \ell_n), n) \end{array} \right) \tag{6}$$

The antecedents of the implications encode that G and H are enabled on states P and Q. On states where G and H are not enabled, the strongest post-conditions are false, so we have

$$\forall \ell_n (\alpha^{\text{out}}(\ell_n) \ \wedge \ \neg (\exists \Sigma_G (P \wedge O_G)) \ \Rightarrow \ \neg \exists \Sigma_G SP_G(P, \ell_n))) \tag{7}$$

and the same for H, Q:

$$\forall \ell_n (\alpha^{\text{in}}(\ell_n) \ \wedge \ \neg (\exists \Sigma_H (Q \wedge I_H)) \ \Rightarrow \ \neg \exists \Sigma_H SP_H(Q, \ell_n))) \tag{8}$$

Notice that $\alpha^{\text{out}}(\ell_n)$ and $\alpha^{\text{in}}(\ell_n)$ are complementary, so by resolving (7) and (8) with (6), we obtain a formula that after further simplifications is

$$\forall \ell_n (\exists \Sigma_G \Sigma_H (EN_G \wedge EN_H \wedge P \wedge Q) \Rightarrow ASIM(SP_G(P, \ell_n), SP_H(Q, \ell_n), n)) \tag{9}$$

So $ASIM(P, Q, n+1)$ is equivalent to the conjunction of (4), (5) and (9).

There is a dual symbolic view of bounded non-conformance that builds on Lemma 4 and Definition 18. For a glas G and $n \geq 0$ let $SP_G^{(n)}$ be the n-fold strongest postcondition transformation of G for a *fixed* sequence of distinct unique label variables $(\ell_0, \ldots, \ell_{n-1})$:

$$SP_G^{(n)} \stackrel{\text{def}}{=} \begin{cases} \imath_G, & \text{if } n = 0; \\ SP_G(SP_G^{(n-1)}, \ell_{n-1}), & \text{otherwise.} \end{cases}$$

In other words, there exists $M \models SP_G^{(n)}$ and $\mathbf{a} = (\ell_0^M, \ldots, \ell_{n-1}^M)$ iff $\mathbf{a}$ is a trace of G of length n. Now, satisfiability of the following formula $W_{G \not\preceq H}^{=n}$ for $n > 0$ is intended to mean that there exists a witness of $G \not\preceq H$ of length n (note that the empty trace cannot be such a witness), and thus, satisfiability of the formula $W_{G \not\preceq H}^{\leq n}$ is intended to capture $G \not\preceq_n H$:

$$W_{G \not\preceq H}^{=n} \stackrel{\text{def}}{=} SP_G^{(n-1)} \wedge SP_H^{(n-1)} \wedge \begin{pmatrix} (O_G \wedge \forall \Sigma_H \, (SP_H^{(n-1)} \Rightarrow \neg O_H)) \vee \\ (I_H \wedge \forall \Sigma_G \, (SP_G^{(n-1)} \Rightarrow \neg I_G)) \end{pmatrix}$$

$$W_{G \not\preceq H}^{\leq n} \stackrel{\text{def}}{=} \bigvee_{m=1}^{n} W_{G \not\preceq H}^{=m}$$

The witness formulas $W_{G \not\preceq H}^{=n}$ and $W_{G \not\preceq H}^{\leq n}$ are formulas over the free variables $\Sigma_G \cup \Sigma_H \cup \{\ell_i\}_{i<n} \cup \{\ell\}$, where, besides i/o-compatibility, it is assumed that the internal signatures are disjoint (in particular $\Sigma_G \cap \Sigma_H = \emptyset$) and the label variables are the same ($\ell = \ell_G = \ell_H$). The intended meaning of the witness formula is made precise by the following theorem.

Theorem 5. *Assume G is i/o-compatible with H, the internal signatures are disjoint, and $\ell = \ell_G = \ell_H$. Then $W_{G \not\preceq H}^{\leq n}$ is satisfiable iff $G \not\preceq_n H$.*

Proof. Let G and H be as stated. Note that i/o-compatibility means that $\alpha_G^{\text{out}} \Rightarrow \alpha_H^{\text{out}}$ and $\alpha_H^{\text{in}} \Rightarrow \alpha_G^{\text{in}}$. By using Lemma 4 and definition of $W_{G \not\preceq H}^{\leq n}$ it suffices to show that $W_{G \not\preceq H}^{=n}$ is satisfiable $\Longleftrightarrow$ there exists a witness of $G \not\preceq H$ of length n. Let $n \geq 1$ be fixed.

($\Longrightarrow$): Assume $M \models W_{G \not\preceq H}^{=n}$. Let $\mathbf{a} = (\ell_0^M, \ldots, \ell_{n-1}^M)$ and $b = \ell^M$. So $M_G \models SP_G^{(n-1)}$ and $M_H \models SP_H^{(n-1)}$ where M_G omits Σ_H (resp. M_H omits Σ_G).

Now assume that M satisfies the output refinement violation condition (first disjunct of the second conjunct) of $W_{G \not\preceq H}^{=n}$, so

$$M_G \models O_G \wedge \neg \exists \Sigma_H \, (SP_H^{(n-1)} \wedge O_H).$$

because the variables Σ_H are not free (and thus unrelated to the interpretation given by M_H). Since $M_G \models O_G$ we have $b \in L_G^{\text{out}}$ and $\lceil G \rceil(\mathbf{a}) \stackrel{b}{\rightarrow}_{\lceil G \rceil}$, and thus $\mathbf{a} \cdot b \in Tr(G)$. We show that $\mathbf{a} \cdot b \notin Tr(H)$ by way of contradiction:

- Suppose that $\mathbf{a} \cdot b \in Tr(H)$. Then there exists a model $N \models SP_H^{(n-1)} \wedge EN_H$ such that $\mathbf{a} = (\ell_0^N, \ldots, \ell_{n-1}^N)$ and $\ell^N = b$. Since $\alpha_G^{\text{out}} \Rightarrow \alpha_H^{\text{out}}$, it follows from $N \models EN_H$ and $N \models \alpha_G^{\text{out}}$ that $N \models O_H$. Let $M' = M_G \cup N$; M' is

well-defined because Σ_G and Σ_H are disjoint and the interpretations of the label variables are identical. So $M' \models SP_H^{(n-1)} \wedge O_H$ that *contradicts* that $M_G \models \neg\exists\Sigma_H\,(SP_H^{(n-1)} \wedge O_H)$.

It follows that $\mathbf{a} \in Tr(G) \cap Tr(H)$, $b \in L_G^{\text{out}}$ and $\mathbf{a}\cdot b \in Tr(G) \setminus Tr(H)$.

The case when M satisfies the input refinement violation condition of $W^{=n}_{G \not\preceq H}$ is symmetrical and shows that there exists $\mathbf{a} \in Tr(G) \cap Tr(H)$ of length $n-1$, $b \in L_H^{\text{in}}$ and $\mathbf{a}\cdot b \in Tr(H) \setminus Tr(G)$.

($\Longleftarrow$): Assume there exists a witness $\mathbf{a}\cdot b$ of $G \not\preceq H$ of length n. Thus $\mathbf{a} \in Tr(G) \cap Tr(H)$, $|\mathbf{a}| = n-1$, and, assume $b \in L_G^{\text{out}}$ and $\mathbf{a}\cdot b \in Tr(G) \setminus Tr(H)$ (the input refinement violation case is symmetrical). So there exists $N \models \lceil G \rceil(\mathbf{a}) \wedge \lceil H \rceil(\mathbf{a})$. Let $M = N \cup \{\ell \mapsto b\} \cup \{\ell_i \mapsto \mathbf{a}[i]\}_{0 \le i < |\mathbf{a}|}$. It follows by similar reasoning that is used above that $M \models W^{=n}_{G \not\preceq H}$. ⊠

The dual formulation follows the lines of the *bounded model program checking* (BMPC) problem [45], here adapted for GLASs. In terms of GLASs, BMPC [45] is the following decision problem: given a glas G, bound n, and a *reachability condition* φ that is a formula such that $FV(\varphi) \subseteq \Sigma_G$, decide if there exists a trace $\mathbf{a}$ of G of length $\le n$ such that $M \models \varphi$ for *some* $M \models \lceil G \rceil(\mathbf{a})$. For n-bounded approximation of i/o-refinement the reachability condition φ for $G \otimes H$ corresponds to the *second conjunct* of $W^{=n}_{G \not\preceq H}$. In the following we examine a subclass of GLASs when the reachability condition can be simplified by eliminating the components $SP_G^{(n-1)}$ and $SP_H^{(n-1)}$ and in this way reducing the reachability condition to $(O_G \wedge \neg O_H) \vee (I_H \wedge \neg I_G)$.

4.5 Robust Bounded Non-Conformance

The above formulation requires several alternations of quantifiers. This poses a challenge even to modern SMT solvers and theorem provers, especially when there are many quantified variables or they range over large or infinite domains. We will therefore show in the following how to reduce BNC to a simpler BMPC problem. For this reduction we consider GLASs that are *robust* in the following sense.

Definition 22. For $a \in L_G$ and $P \in \mathbf{S}_{\lceil G \rceil}$, a is *universal in* P if $P \xrightarrow{a}_{\lceil G \rceil}$ implies $M \xrightarrow{a}_{\lfloor G \rfloor}$ for all $M \models P$.

Intuitively, if a is universal and enabled in a symbolic state, then a is enabled in *all* of the corresponding concrete states.

Definition 23. G is *output-robust* (*input-robust*) if all output (input) labels are universal in all states of $\mathbf{S}_{\lceil G \rceil}$. G is *robust* if it is both input-robust and output-robust.

The following is a logical formulation of the robustness conditions of a GLAS G where we omit the index G, (observe that the conditions hold trivially when G is deterministic):

G *is output-robust* : $\forall P \in \mathbf{S}_{\lceil G \rceil}\, \forall \ell[\, \forall \Sigma(P \Rightarrow \neg O) \vee \forall \Sigma(P \Rightarrow O)\,]$
G *is input-robust* : $\forall P \in \mathbf{S}_{\lceil G \rceil}\, \forall \ell[\, \forall \Sigma(P \Rightarrow \neg I) \vee \forall \Sigma(P \Rightarrow I)\,]$

We make use of the following simple observations that follow directly from the robustness conditions but are useful to emphasize.

Lemma 5. *Let G be a GLAS and $P \in \mathbf{S}_{\lceil G \rceil}$.*
(a) If G is output-robust then $(P \wedge \forall \Sigma\,(P \Rightarrow \neg O)) \Leftrightarrow P \wedge \neg O$.
(b) If G is input-robust then $(P \wedge \forall \Sigma\,(P \Rightarrow \neg I)) \Leftrightarrow P \wedge \neg I$.

Proof. ($\Rightarrow$) is a logical fact; ($\Leftarrow$) follows by robustness as follows for the case (a): $P \wedge \neg O$ implies $\exists \Sigma\,(P \wedge \neg O)$, i.e., $\neg \forall \Sigma\,(P \Rightarrow O)$, and therefore, by output-robustness, $\forall \Sigma\,(P \Rightarrow \neg O)$ must hold. ⊠

The robustness criterion is a sufficient condition under which BNC reduces to the simpler BMPC problem by avoiding alternation of existential and universal quantification of model variables of G and H that arises in the general case in the $W^{=n}_{G \not\preceq H}$ formula. Robustness is therefore not a limitation of the approach per se, but a criterion that enables direct use of theories that are supported in SMT solvers today. We define the following counterparts of the witness formulas for the robust case.

$$
\begin{aligned}
INV_{G \preceq H} &\stackrel{\text{def}}{=} (O_G \Rightarrow O_H) \wedge (I_H \Rightarrow I_G) \\
RobustW^{=n}_{G \not\preceq H} &\stackrel{\text{def}}{=} SP^{(n-1)}_G \wedge SP^{(n-1)}_H \wedge \neg INV_{G \preceq H} \\
RobustW^{\leq n}_{G \not\preceq H} &\stackrel{\text{def}}{=} \bigvee_{m=1}^{n} RobustW^{=m}_{G \not\preceq H}
\end{aligned}
$$

The corretness of the formulas with respect to the intended use, is implied by the following theorem and Theorem 5.

Theorem 6. *If G is input-robust and H is output-robust then, for all $n > 0$, $W^{=n}_{G \not\preceq H} \Leftrightarrow RobustW^{=n}_{G \not\preceq H}$.*

Proof. Assume that G is input-robust and H is output-robust. We have the following equivalences.

$$
\begin{aligned}
W^{=n}_{G \not\preceq H} \quad &\Leftrightarrow \quad SP^{(n-1)}_G \wedge SP^{(n-1)}_H \wedge \left(\begin{array}{l} (O_G \wedge \forall \Sigma_H\,(SP^{(n-1)}_H \Rightarrow \neg O_H)) \vee \\ (I_H \wedge \forall \Sigma_G\,(SP^{(n-1)}_G \Rightarrow \neg I_G)) \end{array} \right) \\
&\Leftrightarrow \quad (SP^{(n-1)}_G \wedge SP^{(n-1)}_H \wedge O_G \wedge \forall \Sigma_H\,(SP^{(n-1)}_H \Rightarrow \neg O_H)) \vee \\
&\qquad\quad (SP^{(n-1)}_G \wedge SP^{(n-1)}_H \wedge I_H \wedge \forall \Sigma_G\,(SP^{(n-1)}_G \Rightarrow \neg I_G)) \\
&\stackrel{(\text{Lma 5})}{\Leftrightarrow} (SP^{(n-1)}_G \wedge SP^{(n-1)}_H \wedge O_G \wedge \neg O_H) \vee \\
&\qquad\quad (SP^{(n-1)}_G \wedge SP^{(n-1)}_H \wedge I_H \wedge \neg I_G) \\
&\Leftrightarrow \quad SP^{(n-1)}_G \wedge SP^{(n-1)}_H \wedge ((O_G \wedge \neg O_H) \vee (I_H \wedge \neg I_G)) \\
&\Leftrightarrow \quad SP^{(n-1)}_G \wedge SP^{(n-1)}_H \wedge \neg((O_G \Rightarrow O_H) \wedge (I_H \Rightarrow I_G)) \\
&\Leftrightarrow \quad RobustW^{=n}_{G \not\preceq H}
\end{aligned}
$$

$\boxtimes$

The core idea of the reduction of BNC to BMPC is to reduce existence of witness of $G \not\preceq_n H$ to reachability of $\neg INV_{G \preceq H}$ of bounded explorations of the composition $G \otimes H$, where $INV_{G \preceq H}$ is an invariant $P_0 \wedge P_1$ (as defined above) for expressing the i/o refinement relation in a given symbolic state. P_0 has the following basic form for stating the output label conformance of the alternating simulation relation (definition of P_1 is symmetrical):

$$\forall \ell \, (\exists X_G(\varphi(\ell, X_G, \Sigma_G)) \Rightarrow \exists X_H(\psi(\ell, X_H, \Sigma_H)))$$

where $\exists X_G \varphi$ (resp. $\exists X_H \psi$) corresponds to O_G (resp. O_H). When the choice variables X_H have a finite sort σ, i.e., $\mathcal{U}^\sigma$ is finite, then P_0 is equivalent to the formula

$$\forall \ell \left(\exists X_G(\varphi(\ell, X_G, \Sigma_G)) \Rightarrow \bigvee_{u_H \in \mathcal{U}^\sigma} \psi(\ell, u_H, \Sigma_H) \right)$$

Note that the sorts of ℓ and model variables do not need to be finite. In this case $\neg P_0$ is equivalent to the existential formula

$$\exists \ell \, X_G \left(\varphi(\ell, X_G, \Sigma_G) \wedge \bigwedge_{u_H \in \mathcal{U}^\sigma} \neg \psi(\ell, u_H, \Sigma_H) \right)$$

that is handled as a quantifier-free formula (over the expanded signature) for the purposes of satisfiability checking, assuming of course that the guards themselves are quantifier free.

The intuition behind robustness is that internal choices should behave *uniformly* in terms of external behavior. For example, deterministic GLASs (such as $G_{Credits}$) are trivially robust, since there are no internal choices. The following example illustrates a nontrivial example of a robust GLAS that is nondeterministic and where internal choices arise naturally as a way of abstracting externally visible behavior. Nondeterministic but robust model-programs arise naturally in object-oriented modeling when objects are used internally and lead to isomorphic but behaviorally indistinguishable models.

Example 12. We consider the *Credits* program and modify it by abstracting the message ids from the labels. The constructors of the $\mathbb{L}$ sort are also modified so that $Req, Res : \mathbb{Z} \to \mathbb{L}$ and the accessors Req_m and Res_m are removed. We call the resulting program *Credits2*:

```
[Action] Req(c as Integer)
  require exists m where IsValidUnusedMessageId(m) and c > 0
  choose m where IsValidUnusedMessageId(m)
    msgs(m) := c
    add m to used

[Action] Res(c as Integer)
   require  exists m where m in msgs and 0<=c and c<=msgs(m)
   choose m where m in msgs and 0<=c and c<=msgs(m)
     remove m from msgs
     if c>0 add (max, max+c) to ranges
     max := max+c
```

We write *Credits* and *Credits2* also for the corresponding GLASs. *Credits2* has two choice variables, say m_{Req} and m_{Res}, the guard and the assignment system of *Credits2* is obtained from the guard and the assignment system of *Credits* by replacing each occurrence of $Req\_m(\ell)$ ($Res\_m(\ell)$) with m_{Req} (m_{Res}). The resulting assignment system is thus given by the require-statements, the guard γ is:

$$\begin{array}{l}(IsReq(\ell) \wedge m_{Req} \notin used \\ \qquad \wedge \exists r\,(r \in ranges \wedge \pi_0(r) \leq m_{Req} \leq \pi_1(r)) \wedge\ Req\_c(\ell) > 0) \vee \\ (IsRes(\ell) \wedge m_{Res} \in msgs \wedge 0 \leq Res\_c(\ell) \leq Read(msgs, m_{Res}))\end{array}$$

The assignment system Δ consists of the assignments:

$$\begin{array}{rl} ranges := & Ite(IsReq(\ell), ranges, Ite(Res\_c(\ell) > 0, \\ & \qquad Store(ranges, \langle max, max + Res\_c(\ell)\rangle, true), ranges)) \\ used := & Ite(IsReq(\ell), Store(used, m_{Req}, Req\_c(\ell)), used) \\ max := & Ite(IsReq(\ell), max, max + Res\_c(\ell)) \\ msgs := & Ite(IsReq(\ell), Store(msgs, m_{Req}, Req\_c(\ell)), \\ & \qquad Store(msgs, m_{Res}, 0)) \end{array}$$

It is easy to see (at least it is easy by considering the more readable AsmL program) that *Credits2* is non-deterministic. For example, $\mathbf{a} = (Req(3), Res(3), Req(1))$ is a trace of *Credits2*. Initially, *ranges* contains the singleton $\langle 0, 0\rangle$ range. So after $Req(3)$ there is a pending request with message id (identifier) 0 (because 0 is the only possible number in the range $\langle 0, 0\rangle$. The effect of $Req(3)$ also updates the map *msgs* with to $[0 \mapsto 3]$, and finally the set *used* becomes $\{0\}$. After $Res(3)$ the range of possible message ids contains the pair $\langle 1, 3\rangle$ and the set of used messages is still $\{0\}$. After $Req(1)$, i.e., in the state $S = \lceil Credits2 \rceil(\mathbf{a})$, there are 3 possible models because there are three different valid unused message ids. The three models are different on *msgs*. One sets $msgs = [1 \mapsto 1]$, the other sets $msgs = [2 \mapsto 1]$, and the last sets $msgs = [3 \mapsto 1]$. One can show that *Credits2* is both input-robust and output-robust. The key property that determines enabledness of a request is the *number* of available message ids, but not their identity. So all models of a reachable symbolic state P are isomorphic modulo the values of the message ids. The value of the message ids do not influence whether a guard is enabled. This property is also used for enabledness of a response. ⊠

The following example illustrates a GLAS that is not output-robust.

Example 13. We consider the *Credits* program again and this time we modify only the *Req* action as in Example 12. We call it *Credits3*. The AsmL formulation is listed below.

```
[Action] Req(c as Integer)
  require exists m where IsValidUnusedMessageId(m) and c > 0
  choose m where IsValidUnusedMessageId(m)
    msgs(m) := c
    add m to used

[Action] Res(m as Integer, c as Integer)
```

```
require m in msgs and 0<=c and c<=msgs(m)
remove m from msgs
if c>0 add (max, max+c) to ranges
max := max+c
```

For example, consider the trace $\mathbf{a} = (\mathit{Req}(3), \mathit{Res}(0,3), \mathit{Req}(1))$ of *Credits3*. The state $S = \lceil \mathit{Credits3} \rceil(\mathbf{a})$ contains 3 models, where, for example, the output label $\mathit{Res}(1,1)$ is only enabled in the model in S where request 1 is pending but not in the model where request 2 is pending. So *Credits3* is not output-robust. ⊠

Theorem 7. *Assume G is input-robust, H is output-robust, G is i/o-compatible with H, $\Sigma_G \cap \Sigma_H = \emptyset$ and $\ell_G = \ell_H$. Then $G \not\preceq_n H$ iff $(\bigvee_{m=1}^{n} SP_{G\otimes H}^{(m-1)}) \wedge \neg INV_{G\preceq H}$ is satisfiable.*

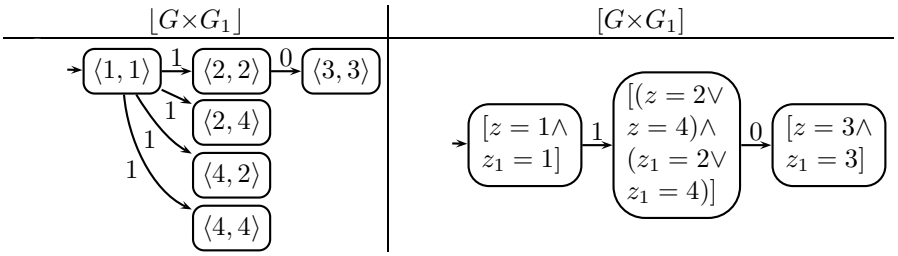
Proof. By Theorem 5, Theorem 6, and Corollary 1. ⊠

The robustness assumptions cannot be omitted in general. This is illustrated by the following example.

Example 14. Consider the GLAS G illustrated by the FSM in Example 9. Note that G is not output-robust. Let G_1 be a copy of G where z is replaced by z_1 and x is replaced by x_1. Clearly $[G] \preceq [G_1]$ (since $[G] \preceq [G]$ by reflexivity of $\preceq$). Now consider $G{\times}G_1$ (that is the same as $G{\otimes}G_1$ here), where

$$\begin{aligned}
\imath_{G\times G_1} &= (z = 1) \wedge (z_1 = 1);\\
\gamma_{G\times G_1} &= (\ell = 1 \wedge z = 1 \wedge z_1 = 1) \vee (\ell = 0 \wedge z = 2 \wedge z_1 = 2);\\
\Delta_{G\times G_1} &= \{z := \mathit{Ite}(z = 1, \mathit{Ite}(x, 2, 4), 3),\\
&\qquad z_1 := \mathit{Ite}(z_1 = 1, \mathit{Ite}(x_1, 2, 4), 3)\}.
\end{aligned}$$

The LTSs $\lfloor G{\times}G_1 \rfloor$ and $[G{\times}G_1]$ can be illustrated as follows where a pair $\langle z, z_1 \rangle$ shows the values of the respective model variables in $\lfloor G{\times}G_1 \rfloor$:



Consider the singleton trace (1). Fix $M = \{z \mapsto 2, z_1 \mapsto 4\} \models \lceil G{\times}G_1 \rceil(1)$. We show that $M \models \exists \ell\, \neg \mathit{INV}_{G\preceq G_1}$, thus $SP_{G\otimes G_1}^{(1)} \wedge \neg \mathit{INV}_{G\preceq G_1}$ is satisfiable but $G \preceq G_1$. The formula $\exists \ell\, \neg \mathit{INV}_{G\preceq G_1}$ is

$$\begin{aligned}
\exists \ell\, \neg(\ &((\ell = 0 \wedge \exists x(\ell = 0 \Leftrightarrow z = 2)) \Rightarrow (\ell = 0 \wedge \exists x_1(\ell = 0 \Leftrightarrow z_1 = 2))) \wedge\\
&((\ell = 1 \wedge \exists x_1(\ell = 0 \Leftrightarrow z_1 = 2)) \Rightarrow (\ell = 1 \wedge \exists x(\ell = 0 \Leftrightarrow z = 2))))
\end{aligned}$$

that is equivalent to

$$\exists \ell \, (\, ((\ell = 0 \wedge (\ell = 0 \Leftrightarrow z = 2)) \wedge \neg(\ell = 0 \wedge (\ell = 0 \Leftrightarrow z_1 = 2))) \vee \\ ((\ell = 1 \wedge (\ell = 0 \Leftrightarrow z_1 = 2)) \wedge \neg(\ell = 1 \wedge (\ell = 0 \Leftrightarrow z = 2))))$$

and after further simplifications is equivalent to

$$(z = 2 \wedge \neg(z_1 = 2)) \vee (\neg(z_1 = 2) \wedge \neg(\neg(z = 2))),$$

which simplifies to

$$z = 2 \ \wedge \ \neg(z_1 = 2)$$

Hence, $M \models \exists \ell \, \neg INV_{G \preceq G_1}$. ⊠

The following example illustrates an application of Theorem 7 when the specification GLAS H is nondeterministic but robust.

Example 15. We consider a model program *CreditsImpl* that describes the abstracted behavior of a protocol implementation.

```
var cs as Seq of Integer = []
[Action] Req(c as Integer)
  require true
  cs := cs + [c]
[Action] Res(c as Integer)
   require cs <> [] and c <= Head(cs) and c >= 0
   cs := Tail(cs)
```

The GLASs *Credits2* (from Example 12) and *CreditsImpl* are both robust. One can show that *CreditsImpl* $\preceq_n$ *Credits2* for any n by using Theorem 7. The following is an AsmL formulation of the composition *CreditsImpl*$\otimes$*Credits2*:

```
var cs as Seq of Integer = []

var ranges as Set of (Integer,Integer) = {(0,0)}
var used as Set of Integer = {}
var max as Integer = 0
var msgs as Map of Integer to Integer = {->}

[Action] Req(c as Integer)
  require exists m where IsValidUnusedMessageId(m) and c > 0
  cs := cs + [c]
  choose m where IsValidUnusedMessageId(m)
    msgs(m) := c
    add m to used

[Action] Res(c as Integer)
   require cs <> [] and c <= Head(cs) and c >= 0
           and exists m where m in msgs and 0<=c and c<=msgs(m)
   cs := Tail(cs)
   choose m where m in msgs and 0<=c and c<=msgs(m)
     remove m from msgs
     if c>0 add (max, max+c) to ranges
     max := max+c
```

The output label (response) conformance part P_0 of the invariant $INV_{\mathit{CreditsImpl} \preceq \mathit{Credits2}}$ can be formulated roughly as the following condition in AsmL (the check that we are dealing with response messages is left implicit here):

```
forall c ((cs <> [] and c <= Head(cs) and c >= 0)
          implies
          exists m where m in msgs and 0<=c and c<=msgs(m))
```

The negation $\neg P_0$ corresponds to the following condition where c is the credits parameter of a response message:

```
exists c ((cs <> [] and c <= Head(cs) and c >= 0)
          and
          forall m in msgs (0>c or c>msgs(m))
```

Within the context of a response message, the condition $\neg P_0$ states that a response from the implementation provides c credits, but according to the specification there exists no justification for receiving those credits. ⊠

Theorem 7 can be seen as a bounded exploration of a symbolic generalization of the *ioco-product* of LTSs [12] that uses special fail states that detect violations of *ioco*. The following definition is such a generalization.

Definition 24. Assume that G is input-robust, H is output-robust, G is i/o-compatible with H, $\Sigma_G \cap \Sigma_H = \emptyset$, and $\ell = \ell_G = \ell_H$. The *i/o refinement composition* of G with H is the following GLAS:

$$G \diamond H \stackrel{\text{def}}{=} (\Sigma_{G\otimes H} \cup \{safe\}, \quad X_{G\otimes H}, \quad \ell, \quad \imath_{G\otimes H} \wedge safe, \quad \alpha_{G\otimes H}, \\ safe \wedge (INV_{G\preceq H} \Rightarrow \gamma_{G\otimes H}), \quad \Delta_{G\otimes H} \cup \{safe := INV_{G\preceq H}\})$$

where *safe* is a fresh Boolean variable called the *safety flag* of $G \diamond H$.

We say that a GLAS G *can reach* a predicate P over Σ_G, if there exists $n \geq 0$ such that $SP_G^{(n)} \wedge P$ is satisfiable, and we say that a trace $\mathbf{a}$ of G *reaches* P if $\lceil G \rceil(\mathbf{a}) \wedge P$ is satisfiable.

Theorem 8. *Assume that G is input-robust, H is output-robust, G is i/o-compatible with H, $\Sigma_G \cap \Sigma_H = \emptyset$, and $\ell = \ell_G = \ell_H$. Then $G \not\preceq H \iff G \diamond H$ can reach $\neg safe$, and if $\mathbf{a} \in Tr(G \diamond H)$ reaches $\neg safe$ then $\mathbf{a}$ is a witness of $G \not\preceq H$.*

Proof. Let G and H be as stated. Let $C = G \otimes H$. We show first that

$$EN_C \Rightarrow INV_{G\preceq H}. \tag{10}$$

From Corollary 1(i) follows that $EN_C \Leftrightarrow EN_G \wedge EN_H$. By using i/o-compatibility, we have that

$$(EN_G \wedge EN_H) \Leftrightarrow ((I_G \wedge I_H) \vee (O_G \wedge O_H) \vee (I_G \wedge O_H))$$

where $I_H \wedge O_G$ has been eliminated because $I_H \Rightarrow \neg\alpha_H^{\text{out}}$, $\neg\alpha_H^{\text{out}} \Rightarrow \neg\alpha_G^{\text{out}}$ (by i/o-compatibility), and $\neg\alpha_G^{\text{out}} \Rightarrow \neg O_G$, i.e., $I_H \Rightarrow \neg O_G$. We now get, by case analysis (recall that $INV_{G\preceq H}$ is $(O_G \Rightarrow O_H) \wedge (I_H \Rightarrow I_G)$):

- $(I_G \wedge I_H) \Rightarrow (I_G \wedge I_H \wedge \neg O_G) \Rightarrow INV_{G\preceq H}$,
- $(O_G \wedge O_H) \Rightarrow (O_G \wedge O_H \wedge \neg I_H) \Rightarrow INV_{G\preceq H}$,
- $(I_G \wedge O_H) \Rightarrow (\neg O_G \wedge \neg I_H) \Rightarrow INV_{G\preceq H}$.

That proves (10). We have that

$$\begin{aligned} EN_{G\diamond H} &\Leftrightarrow \alpha_C \wedge \exists X_C(\mathit{safe} \wedge (\mathit{INV}_{G\preceq H} \Rightarrow \gamma_C)) \\ &\Leftrightarrow \mathit{safe} \wedge ((\alpha_C \wedge \neg \mathit{INV}_{G\preceq H}) \vee EN_C) \end{aligned}$$

Where, by (10), the predicates $\alpha_C \wedge \neg \mathit{INV}_{G\preceq H}$ and EN_C are mutually exclusive. The initial state is safe and $G\diamond H$ behaves exactly as C on safe states, i.e., has exactly the same traces as C up to the point when the i/o-refinement invariant $\mathit{INV}_{G\preceq H}$ is violated at which point it transitions to an unsafe state where no further labels are enabled. Thus,

$$Tr(G\diamond H) = Tr(C) \cup \bigcup_{\mathbf{a}\in Tr(C)} \{\mathbf{a}\cdot \ell^M \mid M \models \lceil C \rceil(\mathbf{a}) \wedge \neg \mathit{INV}_{G\preceq H}\}$$

The main statement follows now by Theorem 7. ⊠

Theorem 7 (or Theorem 8) also identifies conditions where we can use standard techniques for verification of safety formulas. We use this in Theorem 9 to formulate checking for $\mathit{INV}_{G\preceq H}$ as a symbolic bounded model checking problem.

It can be derived from a general safety rule given in Figure 1. It is also using the result from Theorem 7. It modifies the standard proof rule for checking invariants to instead check for $\mathit{INV}_{G\preceq H}$. It uses an auxiliary invariant I for the composed system $G\otimes H$. The premises *Ini* and *Ind* ensure that I is inductive, thus encode (super-sets) of the reachable states in $G\otimes H$.

The last premise checks that $\mathit{INV}_{G\preceq H}$ holds under the assumption of I, and therefore holds in all reachable states. The invariant I summarizes states reachable in $G\otimes H$. The internal choices are opaque in the strongest post-condition, so I cannot distinguish states based on internal choices.

$$\frac{\begin{array}{rl} \mathit{Ini}: & \varphi_G^0 \ \wedge \varphi_H^0 \ \Rightarrow \ I \\ \mathit{Ind}: & SP_{G\otimes H}(I,a) \ \Rightarrow \ I \text{ for each } a \\ \mathit{INV}_{\preceq}: & I \ \Rightarrow \ \mathit{INV}_{G\preceq H} \end{array}}{G\otimes H \vdash \mathit{INV}_{G\preceq H}}$$

Fig. 1. Invariant rule for checking $\mathit{INV}_{G\preceq H}$ when G is input- and H is output robust.

For bounded i/o refinement checking we have the following result.

Theorem 9. *Assume that G is input-robust, H is output-robust, and G is i/o-compatible with H. There is an effective procedure that given G, H and a bound $n > 0$, creates a formula $BNC(G,H,n)$ of size $O(n(|G| + |H|))$ containing free variables ℓ^i, safe^i for $1 \le i \le n$, and is such that $BNC(G,H,n)$ is satisfiable iff $G \not\preceq_n H$. Morover, if $M \models BNC(G,H,n)$ then $((\ell^1)^M, \ldots, (\ell^m)^M)$ is a witness of $G \not\preceq_n H$ where $m \le n$ is such that for all $i < m$, $M \models \mathit{safe}^i$, and $M \models \neg \mathit{safe}^m$.*

Proof. Let G, H, and n be as stated and assume (wlog) that the internal signatures of G and H are disjoint and $\ell = \ell_G = \ell_H$. Let $C = G \diamond H$. Let $\Sigma^0 = \Sigma_C$ and $\Sigma^{i+1} = (\Sigma^i)'$. Define $BNC(G, H, n)$ as follows, illustrated for $n = 3$:

$$\begin{array}{l} \imath_C \wedge (\ TR_C(\Sigma^0, \ell^1, \Sigma^1) \wedge (safe^1 \ \Rightarrow \\ \qquad (\ TR_C(\Sigma^1, \ell^2, \Sigma^2) \wedge (safe^2 \ \Rightarrow \\ \qquad\quad (\ TR_C(\Sigma^2, \ell^3, \Sigma^3) \wedge \neg safe^3))))) \end{array}$$

Theorem 8 implies that $BNC(G, H, n)$ is satisfiable iff $G \not\preceq_n H$. The size of the formula is $O(n|C|)$. ⊠

For the *bounded* version of *ioco* we restrict the length of the traces by a given bound n so that all traces in Definition 20 have a length that is at most n; denoted here by $ioco_n$. We get the following corollary of Theorem 9 and Theorem 4. Note that input-enabledness of $\lfloor G \rfloor$ implies input-robustness of G because if all input labels are enabled in all rechable states in $\lfloor G \rfloor$ then, by Lemma 2, all input labels are universal in all symbolic states in $\lceil G \rceil$.

Corollary 3. *Let H be output-robust and let $\lfloor G \rfloor$ be input-enabled. Then $BNC(G, H, n)$ is satisfiable iff $\lfloor G \rfloor$ $ioco_n$ $\lfloor H \rfloor$ does not hold.*

5 Experiments

We created a prototype implementation for checking bounded i/o-refinement formulas. The prototype uses the F# programmatic interface to the state-of-the art SMT solver Z3 [20] to represent GLASs as a collection of transition pairs. Each pair consists of a specification and an implementation transition and is tagged as either $_{\text{in}}$ or $_{\text{out}}$ to indicate which direction to check the i/o-refinement. The sample AsmL model programs used in the experiment are shown in Figure 2. The data-types used in the model programs are mapped directly to native Z3 theories. For example, the `Mode` enumeration type is mapped into a special case of algebraic data-types where enumerations are encoded as nullary constructors.

The finite map M is represented as an array, and the theory of extensional arrays is used to handle the operations on M. Similarly, the set R is represented as an array that maps integers to Booleans. The operations, element-wise addition and removal required by *Req* and *Res*) are simply array updates. Z3 supports richer set operations as an extension to the theory of arrays, but this example does not make use of these. The prototype uses the fact that terms in Z3 are internally represented as shared directed acyclic graphs. Each distinct formula is built only once, and reused in different occurrences. The size of the resulting path formula is therefore proportional to the number of unfoldings n and to the size of the input model programs.

On the other hand, the size of the input does not depend on the size of the state space. The potentially unbounded size of the state space is also not a factor when checking for bounded i/o-refinement, but our techniques are sensitive to

Model program *Spec* (H)

```
enum Mode
  Undef     = 0
  Sent      = 1
  Canceled = 2

var M as Map of Integer to Mode = {->}

[in,Action]
Req(m as Integer)
  require not (m in M)
  M(m) := Sent

[in,Action]
Cancel(m as Integer)
  require true
  if M(m) = Sent
    M(m) := Canceled

[out,Action]
Res(m as Integer, b as Boolean)
  require m in M and
          (b or M(m) = Canceled)
  remove m from M
```

Model program *Impl* (G)

```
var R as Set of Integer = {}

[in,Action]
Req(m as Integer)
  require true
  add m to R

[in,Action]
Cancel(m as Integer)
  require true
  skip

[out,Action]
Res(m as Integer, b as Boolean)
  require (m in R) and b
  remove m from R
```

Fig. 2. Here *Req* and *Cancel* are in-actions and *Res* is an out-action. The model program *Spec* specifies a request cancellation protocol. A request, identified by a message id m, can be Canceled at any time. A response must be associated to some pending request, where if b is false then the request must have been Canceled. The model program *Impl* describes a particular implementation that never cancels any requests, and responds to all requests in some arbitrary order. *Spec* is an abstracted version of the cancellation feature in the SMB2 protocol [37] (successor of the Windows SMB protocol used for file sharing by Windows machines and machines running third party implementations, such as Samba).

the number of paths in the unfoldings. Figure 3 shows the timings required for checking the property $BNC(G, H, n)$. We observed that the time overhead grows exponentially with the number of unfoldings n (that is linear in the number of paths that are checked). Not shown is the space overhead, which was very modest: space consumption during solving grew linearly with n, from 12 MB to around 20 MB.

A more interesting use of bounded conformance checking is to detect bugs in the models used for either the specifications or implementations. We can *plant* a bug in our example from Figure 2 by changing the *Impl* transition *Res* to forget removing m from R. The bogus transition is therefore:

```
[out,Action]
Res(m as Integer, b as Boolean)
  require (m in R) and b
  skip
```

It takes Z3 well below a second to create a counter-example of length 3. Since the $BNC(G, H, n)$ formula contains equalities that track which actions are taken

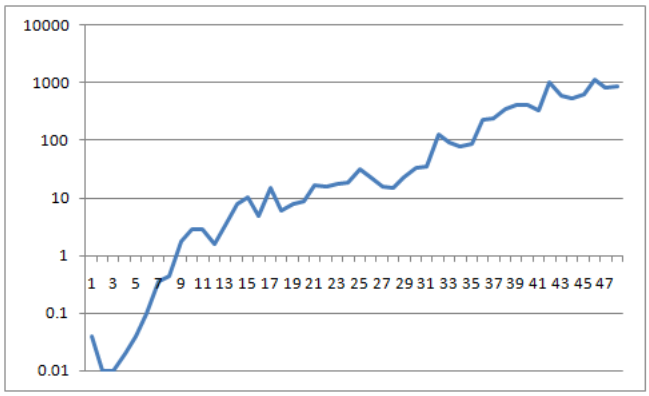


Fig. 3. Timing $BNC(G, H, n)$, $n = 1..47$.

together with their parameters, it is easy to use Z3's model-producing facilities to extract the counter-example:

```
actions0 -> (req 1)
actions1 -> (res 1 true)
actions2 -> (res 1 true)
```

The counter-example says that the client request (*Req*) action is applied with input 1, followed by two server responses (*Res*) using the same parameter 1. The *Spec* model program is not enabled in response to this second action.

6 Related work

The current paper is an extension of [44] that generalizes the notion of model programs to GLASs and generalizes the results in [42] related to deterministic input-output model programs to GLASs. For the reduction from non i/o-refinement checking to bounded model checking over GLASs the paper extends the corresponding result in [42] to a class of *robust* GLASs that provide a nontrivial extension over deterministic model programs. The experiments in Section 5 are based on the implementation originally discussed in [42]. Several computational complexity results about i/o-refinement are also studied in [42] (where i/o-refinement is called game conformance) that carry over to the general case of GLASs. There is a related notion of conformance of nondeterministic model programs [43] (that checks trace inclusion, rather than i/o-refinement). An interesting topic is to formally investigate its relation to i/o-refinement.

Theorem 1 is a fundamental result. It provides a link between the symbolic and the concrete semantics of GLASs, or symbolic labeled transition systems in general, that is instrumental for example in Theorem 4 relating i/o-refinement to *ioco*. Theorem 1 may have other uses. For example, it is easy to see that it implies the classical theorem that for each nondeterministic automaton there is a deterministic automaton accepting the same language (set of finite traces) by

using a special label or symbol identifying final states and considering only traces ending with this symbol. Theorem 1 is also related to fundamental results that relate trace semantics of CSP and CCS [28, 29], where the former correponds to a declarative (symbolic) semantics and the latter to an operational (concrete) semantics.

The literature on ioco [13, 40, 41] and various extension of ioco is extensive. A recent overview and the formal foundations are described in [39]. An extension of ioco theory to symbolic transition systems is proposed in [25]. Composition of GLASs is a generalization of composition of model programs [47] and is also related to composition of symbolic transition systems [24]. The application of composition for symbolic analysis and formal relation to open system verification has not been studied in those contexts as far as we know. We believe that the results presented here can be used and complement the work on symbolic transition systems in [25, 24].

The definition of SP_G can be seen as a generalization the strongest postcondition (sp) transformer for discrete event dynamic systems [33], by incorporating a (possibly infinite) alphabet theory of event labels and allowing nondeterminism. In control theory, a (symbolic) *plant* is a tuple $\mathbf{G} = (\mathcal{P}, \Sigma, sp, I)$ where $\mathcal{P}$ is a set of predicates, Σ is a finite set of events, sp is the strongest post-condition transformer over $\mathcal{P}$, and $I \in \mathcal{P}$ is a predicate (called the initial condition). The plant $\mathbf{G}$ corresponds to the symbolic semantics $\lceil G \rceil$ of a GLAS G such that $\mathcal{P} = \mathbf{S}_{\lceil G \rceil}$, $I = \mathbf{S}^0_{\lceil G \rceil}$, $\Sigma = L_{\lceil G \rceil}$, and, for $S \in \mathcal{P}$, $a \in \Sigma$, $T_{\lceil G \rceil} = \{(S, a, SP_G(S, a)) \mid S \in \mathcal{P}\}$, and where $sp(S)$ for $S \in \mathcal{P}$ in $\mathbf{G}$ is defined as $\bigvee_{a \in \Sigma} SP_G(S, a)$. Similar to GLASs, the motivation for symbolic plants is to generalize the state space evolution of a discrete event dynamic system to an infinite state space by using predicates.

Recent work on verification of hybrid systems uses *qualitative action systems* [3] that build on *hybrid action systems* [36] that provide a discretized view of hybrid systems through *action systems* [6]. The discrete event view makes it possible to verify the *ioco* relation between two hybrid systems [12]. Definition 24 is a symbolic generalization of the *ioco-product* of LTSs [12] that uses special fail states that detect violations of *ioco*. For ioco, the input-conformance part of $INV_{G \preceq H}$ can be omitted, and while G is assumed input-enabled and thus input-robust, the construct $G \diamond H$ assumes H to be output-robust and G to be i/o-compatible with H.

Action systems can be viewed as GLASs, where the semantics is given through *strongest postcondition transformers* rather than through the dual *weakest precondition transformers* [23]. We believe that the results here can be used as a mathematical foundation for a fully symbolic analysis approach of qualitative action systems through GLASs and SMT solving, using i/o-refinement as a generalization of ioco.

We believe that GLASs can also be used as a foundation for symbolic analysis of Event-B models [2] that is an extension of the B-method [1] with events (corresponding to labels of a GLAS) that describe atomic behaviors, where each event is associated with a guard and an assignment, that causes a state transition

when the guard is true in a given state. Composition of Event-B models is discussed in [35, 16].

Symbolic analysis of refinement relations through theorem proving are used in hardware [15, 14]. Various refinement problems between specifications are also the topic of many analysis tools, where sets and maps are used as foundational data structures, such as ASMs, RAISE, Z, TLA+, B, see [8], where the techniques introduced here could be applied. In some cases, like in RAISE, the underlying logic is three-valued. In many of the formalisms, frame conditions need to be specified explicitly, and are not implicit as in the case of model programs or ASMs. In Alloy [30], the analysis is reduced to SAT, by finitizing the data types. In our case we bound the search depth rather than the size of the data types.

For implementation, we use the state of the art SMT solver Z3 [20], discussed in Section 5. Our experiment indicated that Z3 could be used for modest bounded exploration. More interestingly, it posed an intriguing challenge for solvers like Z3 to better handle *diamond* structured formulas. One technique for handling *diamond* style formulas is explored in [9]. It uses a combination of abstract interpretation and constraint propagation to speed up the underlying constraint solving engine.

BMPC [45], that is used in Section 4, is a generalization of SMT based bounded model checking [22] first to deterministic model programs [46] and then to GLASs. The notion of *i/o-refinement* of GLASs uses the game view of open systems [18]. The game view can also be used to formulate other problems related to input-output GLASs, such as finding winning strategies to reach certain goal states. Game based testing approaches with finite model programs are discussed in [10] using reachability games. There is a different notion of IO-refinement, introduced in [11], that is used in Z as a generalization of data refinement by allowing refinement of input and output parameters. This is quite different from i/o-refinement of GLASs, that is based on refinement of interface automata.

7 Conclusions

GLASs provide a framework for symbolic analysis of labeled transition systems and close the, almost decade-long, discussion regarding the precise relationship between *alternating simulation* and *ioco*, where both notions are being used in state-of-the-art model-based testing tools both in industry as well as academia. In a more general setting, the results of the paper relate the areas of open system verification and testing of reactive systems by providing a common mathematical foundation for the basic underlying notions. We believe that recent advances in symbolic analysis techniques, in particular SMT solving, make the approach also practical, expose new challenges, and will lead to new algorithmic insights in software testing and verification.

Acknowledgement

The first author thanks Bernhard Aichernig for several valueable comments and pointers to related work during discussions at the Dagstuhl seminar *Model-Based Testing in Practice* in October 2010.

References

1. J.-R. Abrial. *The B-Book: Assigning programs to meanings*. Cambridge University Press, 1996.
2. J.-R. Abrial and S. Hallerstede. Refinement, decomposition and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, 77(1-2):1–28, 2007.
3. B. Aichernig, H. Brandl, and W. Krenn. Qualitative action systems. In K. Breitman and A. Cavalcanti, editors, *ICFEM'09*, volume 5885 of *LNCS*, pages 206–225. Springer Berlin / Heidelberg, 2009.
4. R. Alur, T. A. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *CONCUR'98*, volume 1466 of *LNCS*, pages 163–178. Springer, 1998.
5. AsmL. http://research.microsoft.com/fse/AsmL/.
6. R. J. R. Back and K. Sere. Stepwise refinement of parallel algorithms. *Sci. Comput. Program.*, 13:133–180, April 1990.
7. C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. *Satisfiability Modulo Theories*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, February 2009.
8. D. Bjørner and M. Henson, editors. *Logics of Specification Languages*. Springer, 2008.
9. N. Bjørner, B. Dutertre, and L. de Moura. Accelerating Lemma Learning using Joins - DPPL(Join). In *Proceedings of short papers at LPAR'08*, 2008.
10. A. Blass, Y. Gurevich, L. Nachmanson, and M. Veanes. Play to test. Technical Report MSR-TR-2005-04, Microsoft Research, January 2005. Short version appears in FATES 2005.
11. E. Boiten and J. Derrick. IO-refinement in Z. In *3rd Northern Formal Methods Workshop*, 1998.
12. H. Brandl, M. Weiglhofer, and B. K. Aichernig. Automated conformance verification of hybrid systems. In *QSIC 2010: the 10th International Conference on Quality Software*, pages 3–12. IEEE Computer Society, 2010.
13. E. Brinksma and J. Tretmans. Testing Transition Systems: An Annotated Bibliography. In *MOVEP'2k*, volume 2067 of *LNCS*, pages 187–193. Springer, 2001.
14. R. E. Bryant, S. M. German, and M. N. Velev. Exploiting positive equality in a logic of equality with uninterpreted functions. In *Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 470–482. Springer, 1999.
15. J. R. Burch and D. L. Dill. Automatic verification of pipelined microprocessor control. In D. L. Dill, editor, *Conference on Computer-Aided Verification (CAV'94)*, volume 818 of *LNCS*, pages 68–80. Springer, 1994.
16. M. Butler. Decomposition structures for Event-B. In M. Leuschel and H. Wehrheim, editors, *IFM'09*, volume 5423 of *LNCS*, pages 20–38. Springer, 2009.
17. CACM Staff. Microsoft's protocol documentation program: Interoperability testing at scale, a discussion with Nico Kicillof, Wolfgang Grieskamp, and Bob Binder. *Communications of the ACM*, 54(7):51–57, 2011.

18. L. de Alfaro. Game models for open systems. In N. Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 269–289. Springer, 2004.
19. L. de Alfaro and T. A. Henzinger. Interface automata. In *ESEC/FSE*, pages 109–120. ACM, 2001.
20. L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'08)*, LNCS. Springer, 2008.
21. L. de Moura and N. Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54:69–77, 2011.
22. L. de Moura, H. Rueß, and M. Sorea. Lazy Theorem Proving for Bounded Model Checking over Infinite Domains. In *Proceedings of the 18th International Conference on Automated Deduction (CADE'02)*, volume 2392 of *LNCS*, pages 438–455. Springer, 2002.
23. E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Springer, New York, 1990.
24. L. Frantzen, J. Tretmans, and T. Willemse. A symbolic framework for model-based testing. In K. Havelund, M. Núñez, G. Rosu, and B. Wolff, editors, *FATES/RV 2006*, number 4262 in LNCS, pages 40–54. Springer, 2006.
25. L. Franzen, J. Tretmans, and T. Willemse. Test generation based on symbolic specifications. In J. Grabowski and B. Nielsen, editors, *Proceedings of the Workshop on Formal Approaches to Software Testing (FATES 2004)*, pages 3–17, Linz, Austria, September 2004. To appear in *LNCS*.
26. P. Godefroid, M. Levin, and D. Molnar. Automated whitebox fuzz testing. In *Network and Distributed System Security Symposium*, 2008.
27. Y. Gurevich, B. Rossman, and W. Schulte. Semantic essence of AsmL. *Theor. Comput. Sci.*, 343(3):370–412, 2005.
28. J. He and T. Hoare. CSP is a retract of CCS. In S. Dunne and W. Stoddart, editors, *UTP'2006*, volume 4010 of *LNCS*, pages 38–62. Springer, 2006.
29. J. He and T. Hoare. CSP is a retract of CCS. *Theor. Comput. Sci.*, 411:1311–1337, March 2010.
30. D. Jackson. *Software Abstractions*. MIT Press, 2006.
31. J. Jacky, M. Veanes, C. Campbell, and W. Schulte. *Model-based Software Testing and Analysis with C#*. Cambridge University Press, 2008.
32. R. Keller. Formal verification of parallel programs. *Communications of the ACM*, pages 371–384, July 1976.
33. R. Kumar, V. K. Garg, and S. I. Marcus. A predicate transformer approach to control of discrete event systems. *IEEE Transactions on Automatic Control*, 38(2):232–247, February 1993.
34. N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 137–151. ACM Press, 1987.
35. M. Poppleton. The composition of Event-B models. In E. Börger, M. J. Butler, J. P. Bowen, and P. Boca, editors, *Int. Conference on ASM, B and Z (ABZ'08)*, volume 5238 of *LNCS*, pages 209–222. Springer, 2008.
36. M. Rönkkö, A. P. Ravn, and K. Sere. Hybrid action systems. *Theoretical Computer Science*, 290(1):937–973, 2003.
37. SMB2. http://msdn2.microsoft.com/en-us/library/cc246482.aspx, 2008.
38. N. Tillmann and W. Schulte. Parameterized unit tests. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 253–262. ACM, 2005.

39. J. Tretmans. Model based testing with labelled transition systems. In R. Hierons, J. Bowen, and M. Harman, editors, *Formal Methods and Testing*, volume 4949 of *LNCS*, pages 1–38. Springer, 2008.
40. J. Tretmans and A. Belinfante. Automatic testing with formal methods. In *EuroSTAR'99: 7th European Int. Conference on Software Testing, Analysis & Review*, Barcelona, Spain, November 8–12, 1999. EuroStar Conferences, Galway, Ireland.
41. M. van der Bij, A. Rensink, and J. Tretmans. Compositional testing with ioco. In A. Petrenko and A. Ulrich, editors, *Formal Approaches to Software Testing: Third International Workshop, FATES 2003*, volume 2931 of *LNCS*, pages 86–100. Springer, 2004.
42. M. Veanes and N. Bjørner. Input-Output Model Programs. In *ICTAC'09*, volume 5684 of *LNCS*, pages 322–335. Springer, 2009.
43. M. Veanes and N. Bjørner. Symbolic bounded conformance checking of model programs. In A. Pnueli, I. Virbitskaite, and A. Voronkov, editors, *Perspectives of System Informatics (PSI'09)*, volume 5947 of *LNCS*, pages 388–400. Springer, 2009.
44. M. Veanes and N. Bjørner. Alternating Simulation and IOCO. In A. Petrenko, A. Simão, and J. Maldonado, editors, *ICTSS'2010*, volume 6435 of *LNCS*, pages 47–62. Springer, 2010.
45. M. Veanes, N. Bjørner, Y. Gurevich, and W. Schulte. Symbolic Bounded Model Checking of Abstract State Machines. *Int J Software Informatics*, 3(2–3):149–170, June/September 2009.
46. M. Veanes, N. Bjørner, and A. Raschke. An SMT approach to bounded reachability analysis of model programs. In *FORTE'08*, volume 5048 of *LNCS*, pages 53–68. Springer, 2008.
47. M. Veanes and J. Jacky. Composing model programs for analysis. *Journal of Logic and Algebraic Programming*, 79(7):467–482, 2010.