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* [18]. A closely related notion of *alternating simulation* [3] is used in the context of open system verification, in particular for interface automata refinement [9,8]. 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.

GLASs are a generalization of *non-deterministic model programs* [23] to a purely symbolic setting, by abstracting from the particular background universe and the particular (action) label domain. The semantics of GLASs uses classical 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 [21] 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 [15,16] with the interleaving semantics of unshared labels. Composition

A. Petrenko, A. Simão, and J.C. Maldonado (Eds.): ICTSS 2010, LNCS 6435, pp. 47–62, 2010.

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* or BMPC [23]. Detailed proofs of all statements omitted here can be found in the technical report [22].

Although the focus of the paper is theoretical, GLASs provide a foundation of applying state-of-the-art *satisfiability modulo theories* [5] (SMT) technology to a wide range of problems that are difficult to tackle using other techniques. SMT solving is a hybrid technology that has a flavor of model checking, SAT solving, and theorem proving. An advantage over model checking is avoidance of *state-space explosion*. Compared to SAT solving, *bit blasting* can be avoided by encoding operations over unbounded universes, such as integers, more succinctly. Compared to many automated theorem proving techniques, a *solution* is provided as a witness of satisfiability. The following three are sample applications: 1) symbolic model-checking of a given specification GLAS [23] 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 5. 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 [10].

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*. We use x' as an injective *renaming* operation on variables x, and lift the renaming to sets of variables, $\Sigma' \stackrel{\text{def}}{=} \{x' \mid x \in \Sigma\}$. A term t *over* Σ has $FV(t) \subseteq \Sigma$.

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, $Ite(\varphi, t_1, t_2)^M$ equals t_1^M, if φ^M is true; it equals t_2^M, otherwise.

M satisfies φ or *φ is true in M*, denoted by $M \models \varphi$, if φ^M is *true*. A formula φ is *satisfiable* if it has a model and *valid*, denoted by $\models \varphi$, if φ is true in all models. For two formulas φ and ψ, $\varphi \models \psi$ means that any model of φ is also a model of ψ. 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.

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. 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 $sort(x) = sort(u)$.

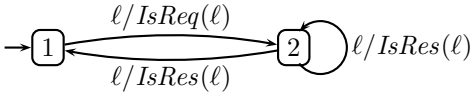
Definition 1. A *Guarded Labeled Assignment System* or *GLAS* G is a tuple $(\Sigma, X, \ell, \imath, \alpha, \gamma, \Delta)$ where

- 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 $\{z := u_z\}_{z \in \Sigma}$ where each u_z is a term over $\Sigma \cup X \cup \{\ell\}$, 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 FSM A:



Intuitively, A specifies a sequence of request and response labels where a single request is followed by one or more respones. Suppose that the labels have sort $\mathbb{L}$ and that $\mathbb{L}$ is associated with predicates $IsReq, IsRes\colon \mathbb{L} \to \mathbb{B}$. A can be represented by the GLAS $G_A = (\{z\colon \mathbb{Z}\}, \{x\colon \mathbb{B}\}, \ell\colon \mathbb{L}, z = 1, IsReq(\ell) \vee IsRes(\ell), Ite(IsReq(\ell), z = 1, z = 2), \{z := Ite(z = 1, 2, Ite(x, 1, 2))\})$. Note that x represents a nondeterministic choice of the target state of a response transition. ⊠

[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).

The following example illustrates how an AsmL [4] 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 [14].

```
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 m notin 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: $Req, Res\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{L}$ and accessors: $Req\_m, Res\_m, Req\_c, Res\_c\colon \mathbb{L} \to \mathbb{Z}$. For example, $IsReq(Res(6,7))$ is false and $Req\_c(Req(3,4))$ 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:

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

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

We map *Credits* to the GLAS $G_{Credits}$: $(\Sigma, \emptyset, \ell, \imath, IsReq(\ell) \vee IsRes(\ell), \gamma, \Delta)$ where $\Sigma = \{ranges\colon \mathbb{A}(\mathbb{T}(\mathbb{Z},\mathbb{Z}),\mathbb{B}), used\colon \mathbb{A}(\mathbb{Z},\mathbb{B}), max\colon \mathbb{Z}, msgs\colon \mathbb{A}(\mathbb{Z},\mathbb{Z})\}$. The initial condition $\imath$ is

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

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

$$\begin{aligned}
&(\mathit{IsReq}(\ell) \wedge \mathit{Req_m}(\ell) \notin \mathit{used} \\
&\qquad \wedge \exists r\,(r \in \mathit{ranges} \wedge \pi_0(r) \leq \mathit{Req_m}(\ell) \wedge \mathit{Req_m}(\ell) \leq \pi_1(r)) \\
&\qquad \wedge \mathit{Req_c}(\ell) > 0) \vee \\
&(\mathit{IsRes}(\ell) \wedge \mathit{Res_m}(\ell) \in \mathit{msgs} \wedge 0 \leq \mathit{Res_c}(\ell) \\
&\qquad \wedge \mathit{Res_c}(\ell) \leq \mathit{Read}(\mathit{msgs}, \mathit{Res_m}(\ell)))
\end{aligned}$$

The assignment system Δ consists of the assignments:

$$\begin{aligned}
\mathit{ranges} := &\mathit{Ite}(\mathit{IsReq}(\ell), \mathit{ranges}, \mathit{Ite}(\mathit{Res_c}(\ell) > 0, \\
&\qquad \mathit{Store}(\mathit{ranges}, \langle \mathit{max}, \mathit{max} + \mathit{Res_c}(\ell)\rangle, \mathit{true}), \mathit{ranges})) \\
\mathit{used} := &\mathit{Ite}(\mathit{IsReq}(\ell), \mathit{Store}(\mathit{used}, \mathit{Req_m}(\ell), \mathit{Req_c}(\ell)), \mathit{used}) \\
\mathit{max} := &\mathit{Ite}(\mathit{IsReq}(\ell), \mathit{max}, \mathit{max} + \mathit{Res_c}(\ell)) \\
\mathit{msgs} := &\mathit{Ite}(\mathit{IsReq}(\ell), \mathit{Store}(\mathit{msgs}, \mathit{Req_m}(\ell), \mathit{Req_c}(\ell)), \\
&\qquad \mathit{Store}(\mathit{msgs}, \mathit{Res_m}(\ell), \mathit{default}_{\mathbb{Z}}))
\end{aligned}$$

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). In order to keep the paper self-contained and to fix the notations 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 $S' \in \mathbf{S}$. $\mathcal{L}$ is *deterministic* if $\mathcal{L}$ has a single initial state and for all $a \in L$ and $S \in \mathbf{S}$ there is at most one $S' \in \mathbf{S}$ such that $(S, a, S') \in T$.

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 $S \xrightarrow{a}\!\!\!\!\!\!/\;\;_{\mathcal{L}}$. 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 $\mathit{Tr}(\mathcal{L})$.

When $\mathcal{L}$ is deterministic, we view $\mathcal{L}$ as a function from all label sequences $\mathbf{a}$ to states or the value $\perp_{\mathcal{L}}$ when $\mathbf{a}$ is not a trace of $\mathcal{L}$. Thus,

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

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.

Definition 4. Let $G = (\Sigma, X, \ell, \imath(\Sigma), \alpha, \gamma(\Sigma), \{z := u_z(\Sigma)\}_{z\in\Sigma})$ be a GLAS. We define the *transition relation* TR_G, and the *strongest post-condition predicate transformer* SP_G, for G, where $P(\Sigma)$ is a predicate over Σ:

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

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

Note that, for $a \in \mathcal{U}^{sort(\ell)}$, $SP_G(P, a)$ is a predicate over Σ. 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 in the SP_G-closure of $\{\imath_G\}$. We define the set of *labels of* G as

$$L_G \stackrel{\text{def}}{=} \{\ell_G^M \mid M \models \alpha_G\}.$$

Definition 5. $\lfloor G \rfloor = (\mathbf{S}, \{M \mid M \models \imath_G\}, L_G, T)$ 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 $N \in \mathbf{S}$.

Definition 6. $\lceil G \rceil = (\mathbf{S}, \{\imath_G\}, L_G, T)$ where $\mathbf{S}$, T are the least sets such that $\imath_G \in \mathbf{S}$, $(P, a, SP_G(P, a)) \in T$ for $a \in L_G$, $P \in \mathbf{S}$ where $SP_G(P, a)$ is satisfiable.

The notion of traces of G is based on the symbolic semantics of G.

Definition 7. $Tr(G) \stackrel{\text{def}}{=} Tr(\lceil G \rceil)$.

We show that both semantics 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. Note that $\lceil G \rceil$ is deterministic and recall (1); let $\perp_{\lceil G \rceil} \stackrel{\text{def}}{=} \mathit{false}$. 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 1. *For all* $\mathbf{a}$, $\{M \mid M \models \lceil G \rceil(\mathbf{a})\} = \{M \mid \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\mathbf{a}}_{\lfloor G \rfloor} M)\}$.

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 by IH that the statement holds for $\mathbf{a}$, we prove it for $\mathbf{a} \cdot a$.

$$\{M \mid M \models \lceil G \rceil(\mathbf{a} \cdot a)\} \stackrel{(\text{def } 6)}{=} \{M \mid M \models SP_G(\lceil G \rceil(\mathbf{a}), a)\}$$
$$\stackrel{(\text{def } 4)}{=} \{M \mid M \models \exists \Sigma'(\lceil G \rceil(\mathbf{a})(\Sigma') \wedge TR_G(\Sigma', a, \Sigma))\}$$

$$
\begin{aligned}
&= \{M \mid \exists N(N \models \lceil G \rceil(\mathbf{a}), \\
&\qquad M \models \exists \Sigma'(P_N(\Sigma') \wedge TR_G(\Sigma', a, \Sigma)))\} \\
&\overset{\text{(IH)}}{=} \{M \mid \exists N\, \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\mathbf{a}}_{\lfloor G \rfloor} N, \\
&\qquad M \models \exists \Sigma'(P_N(\Sigma') \wedge TR_G(\Sigma', a, \Sigma)))\} \\
&\overset{\text{(def 4)}}{=} \{M \mid \exists N\, \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\mathbf{a}}_{\lfloor G \rfloor} N, \\
&\qquad M \models SP_G(P_N, a))\} \\
&\overset{\text{(def 5)}}{=} \{M \mid \exists N\, \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\mathbf{a}}_{\lfloor G \rfloor} N, \\
&\qquad N \xrightarrow{a}_{\lfloor G \rfloor} M)\} \\
&= \{M \mid \exists M_0 \in \mathbf{S}^0_{\lfloor G \rfloor}\, (M_0 \xrightarrow{\mathbf{a} \cdot a}_{\lfloor G \rfloor} M)\}
\end{aligned}
$$

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. $Tr(\lceil G \rceil) = Tr(\lfloor G \rfloor)$.

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

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.

Example 3. 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. ⊠

3.1 GLAS Composition

Composition of GLASs is a purely symbolic construction.

Definition 8. Let $G_i = (\Sigma_i, X_i, \ell, \imath_i, \alpha_i, \gamma_i, \{z := u_z\}_{z \in \Sigma_i})$, for $i \in I$, be GLASs with disjoint internal signatures. The *composition of* G_i *for* $i \in I$, is the GLAS

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

We abbreviate $\bigotimes_{i\in I} G_i$ by $\bigotimes_I G_i$ and for $\bigotimes_{\{1,2\}} G_i$ we write $G_1 \otimes G_2$. Note that $\bigotimes_I G_i$ is indeed well-defined as a GLAS. In particular, $\imath_{\bigotimes_I G_i}$ 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 2. *Let $G = \bigotimes_I G_i$. Assume $L_{G_i} = L_{G_j}$ for $i, j \in I$. Let P_i be a predicate over Σ_i for $i \in I$. Then $\models SP_G(\bigwedge_{i\in I} P_i, a) \Leftrightarrow \bigwedge_{i\in I} SP_{G_i}(P_i, a)$.*

Proof. We first show (*): $\models TR_G(\Sigma'_G, \ell, \Sigma_G) \Leftrightarrow \bigwedge_{i\in I} TR_{G_i}(\Sigma'_{G_i}, \ell, \Sigma_{G_i})$ by using the assumption, definition of TR_G, Definition 8, and standard logical transformations (that use disjointness of the internal signatures of G_i for $i \in I$). The lemma follows by using (*), Definition 4, and further logical transformations. ⊠

One can show that composition of GLASs respects the standard parallel synchronous composition of LTSs with the interleaving semantics of unshared labels. Here we assume the special case of all labels being shared, i.e. $L_{G_i} = L_{G_j}$ for $i, j \in I$. A general statement can be formulated that describes the interleaving of unshared labels, but the special case is sufficient for this paper.

Theorem 2. *Let $G = \bigotimes_I G_i$. Assume $L_{G_i} = L_{G_j}$ for $i, j \in I$.*
(i) *For all* $\mathbf{a}$, $\models \lceil G \rceil(\mathbf{a}) \Leftrightarrow \bigwedge_{i\in I} \lceil G_i \rceil(\mathbf{a})$.
(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 $\mathbf{a} \cdot a$:

$$\begin{array}{lll} \lceil G \rceil(\mathbf{a} \cdot a) \overset{(\text{def } 6)}{\Leftrightarrow} SP_G(\lceil G \rceil(\mathbf{a}), a) & \overset{(\text{IH})}{\Leftrightarrow} & SP_G(\bigwedge_I \lceil G_i \rceil(\mathbf{a}), a) \\ & \overset{(\text{lemma } 2)}{\Leftrightarrow} & \bigwedge_I SP_{G_i}(\lceil G_i \rceil(\mathbf{a}), a) \\ & \overset{(\text{def } 6)}{\Leftrightarrow} & \bigwedge_I \lceil G_i \rceil(\mathbf{a} \cdot a) \end{array}$$

Statement (i) follows by the induction principle. We now prove (ii):

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

The third equality assumes disjointness of Σ_{G_i} for $i \in I$. ⊠

Example 4. Consider the composition $G = G_{Credits} \otimes G_A$ 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 3 is therefore *not* a trace of G. ⊠

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 9. An *i/o-GLAS* G is an extension $(G', \alpha^{\text{out}})$ of a GLAS G' where α^{out} is a formula such that $\alpha^{\text{out}} \models \alpha_G$ called the *output label predicate*.

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 $\alpha_G \wedge \neg\alpha_G^{\text{out}}$. 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.

Example 5. 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 $IsRes(\ell)$. ⊠

When dealing with formal notions of conformance, in particular *ioco* [18], 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 [18]:

Definition 10. Let $\mathcal{L}$ be an LTS and $\delta \notin L_{\mathcal{L}}$. Then $\mathcal{L}^\delta$ is the extension of $\mathcal{L}$ where $L_{\mathcal{L}^\delta}^{\text{out}} = L_{\mathcal{L}}^{\text{out}} \cup \{\delta\}$ and $S \xrightarrow{\delta}_{\mathcal{L}^\delta} S$ iff for all $a \in L_{\mathcal{L}}^{\text{out}}$, $S \overset{a}{\not\to}_{\mathcal{L}}$.

We define a corresponding symbolic extension for GLASs.

Definition 11. For $G = (\Sigma, X, \ell, \imath, \alpha, \gamma, \{z := u_z\}_{z\in\Sigma}, \alpha^{\text{out}})$, $\delta \in \mathcal{U}^{sort(\ell)} \setminus L_G$:

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

Thus, in G^δ there is a new output label δ and $M \xrightarrow{\delta}_{\lfloor G^\delta \rfloor}$ if and only if for all $a \in L_G^{\text{out}}$, $M \overset{a}{\not\to}_{\lfloor G \rfloor}$. 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 [18] 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. (ii) uses (i), Definition 7 and Theorem 1. ⊠

Note however that $Tr(G^\delta) \neq Tr(\lceil G \rceil^\delta)$ as illustrated by the following example which also illustrates the use of choice variables in a GLAS.

Example 6. The example is derived from a standard example that is used to illustrate properties of quiescence during determinization of non-deterministic LTSs [18, Figure 6]. The GLAS G is represented below by an FSM where there

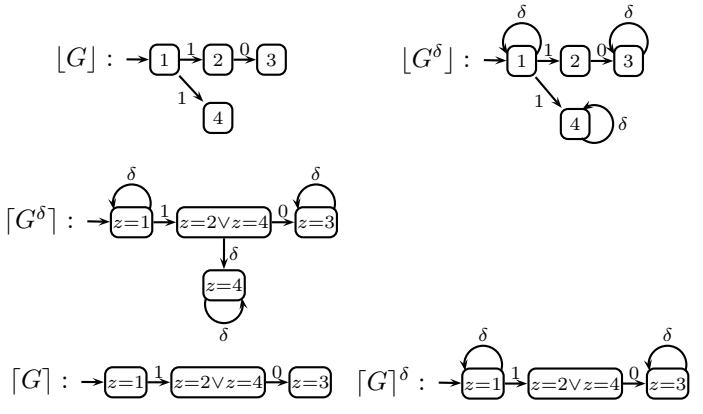
is a single input label 1 and a single output label 0. We assume the following representation for G:

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

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 $z \neq 2$. (Let, e.g. $\delta = 2$),

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

We can illustrate the GLASs as follows:



Thus $\lceil G^\delta \rceil \neq \lceil G \rceil^\delta$. Moreover, $Tr(\lceil G^\delta \rceil) \neq Tr(\lceil G \rceil^\delta)$. ⊠

Example 7. 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. ⊠

We define a notion of conformance between two GLASs that is based on *alternating simulation* [3] between two LTSs and show below that this notion of conformance coincides with *ioco* for GLASs.

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

Definition 12. $\mathcal{M}_1$ *i/o-refines* $\mathcal{M}_2$, $\mathcal{M}_1 \preceq \mathcal{M}_2$, iff there exists an alternating simulation ρ from $\mathcal{M}_1$ to $\mathcal{M}_2$ such that $(S_1^0, S_2^0) \in \rho$, where an *alternating simulation from* $\mathcal{M}_1$ *to* $\mathcal{M}_2$ is a relation $\rho \subseteq \mathbf{S}_1 \times \mathbf{S}_2$ such that, for all $(S_1, S_2) \in \rho$

[2] Deterministic LTSs are called *interface automata* in [9].

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

Given GLASs G and H then $G \preceq H \stackrel{\text{def}}{=} \lceil G \rceil \preceq \lceil H \rceil$.

Definition 12 is consistent with [8]. In particular, several foundational properties of $\preceq$ (like reflexivity and transitivity) are established in [8] that show that $\preceq$ is a suitable refinement relation.

Example 8. Consider two GLASs *Spec* and *Impl* where $\ell : \mathbb{B}$ and α^{out} is $\neg\ell$.

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

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

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

Definition 13. A sequence $\mathbf{a} \cdot a$ is a *witness of* $\mathcal{M}_1 \not\preceq \mathcal{M}_2$ if $\mathbf{a} \in Tr(\mathcal{M}_1) \cap Tr(\mathcal{M}_2)$ and either $a \in L_1^{\text{in}}$ and $\mathbf{a} \cdot a \in Tr(\mathcal{M}_1) \setminus Tr(\mathcal{M}_2)$, or $a \in L_1^{\text{out}}$ and $\mathbf{a} \cdot a \in Tr(\mathcal{M}_2) \setminus Tr(\mathcal{M}_1)$.

For example, the (singleton) sequence *true* is a witness of $\lceil Spec \rceil \not\preceq \lceil Impl \rceil$ in Example 8. The following lemma justifies Definition 13.

Lemma 3. $\mathcal{M}_1 \preceq \mathcal{M}_2 \iff \mathcal{M}_1 \not\preceq \mathcal{M}_2$ *has no witnesses.*

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

Definition 14. $\mathcal{M}_1 \preceq_n \mathcal{M}_2 \stackrel{\text{def}}{=} \mathcal{M}_1 \not\preceq \mathcal{M}_2$ has no witness of length $\leq n$.

It follows directly from Lemma 3 that $\mathcal{M}_1 \preceq \mathcal{M}_2$ iff $\mathcal{M}_1 \preceq_n \mathcal{M}_2$ for all $n > 0$. For example, $Spec \not\preceq_1 Impl$ in Example 8. We are interested in the following decision problem. For GLASs G and H, a *witness of* $G \not\preceq H$ is a witness of $\lceil G \rceil \not\preceq \lceil H \rceil$ and we let $G \preceq_n H \stackrel{\text{def}}{=} \lceil G \rceil \preceq_n \lceil H \rceil$.

Definition 15. *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 $G \not\preceq_n H$.

We show how to reduce BNC to the BMPC problem [23] for a class of GLASs. There is a mapping of the *BMPC* problem over AsmL model programs and the encoding described in [23] to GLASs: given G, 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 $\leq n$ such that $M \models \varphi$ for *some* $M \models \lceil G \rceil(\mathbf{a})$. For this reduction we need to consider GLASs that are *robust* in the following sense.

Definition 16. For $a \in L_G$ and $P \in \mathbf{S}_{\lceil G \rceil}$, a is *robust 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 robust and enabled in a symbolic state, then a is enabled in *all* of the corresponding concrete states.

Definition 17. G is *output-robust* (*input-robust*) if for all $P \in \mathbf{S}_{\lceil G \rceil}$ and all $a \in L_G^{\text{out}}$ ($a \in L_G^{\text{in}}$), a is robust in P. G is *robust* if it is both input-robust and output-robust.

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.

Example 9. 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}). It is easy to see that *Credits2* is non-deterministic. For example, $\mathbf{a} = (Req(3), Res(3), Req(1))$ is a trace of *Credits2*. After $Req(3)$ there is a pending request with id 0. After $Res(3)$ the range of possible message ids contains the pair $\langle 1, 3\rangle$ and the used set of messages in $\{0\}$. After $Req(1)$, i.e., in the state $S = \lceil Credits2 \rceil(\mathbf{a})$, there are 3 possible models. 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, similarly for responses. ⊠

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

Example 10. We consider the *Credits* program again and this time we modify only the *Req* action as in Example 9. We call it *Credits3*. For example, consider the trace $\mathbf{a} = (Req(3), Res(0, 3), Req(1))$ of *Credits3*. The state $S = \lceil Credits2 \rceil(\mathbf{a})$ contains 3 models, where, for example, the output label $Res(1, 1)$

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. ⊠

The key insight of reducing BNC to BMPC comes from Lemma 3 and the use of composition. Let α^{in} stand for the formula $\alpha \wedge \neg\alpha^{\text{out}}$. We assume that G and H below have disjoint internal signatures.

$$P_{\preceq}(G,H) \stackrel{\text{def}}{=} \forall \ell\, (((\alpha_G^{\text{out}} \wedge \exists X_G\, \gamma_G) \Rightarrow (\alpha_H^{\text{out}} \wedge \exists X_H\, \gamma_H)) \wedge ((\alpha_H^{\text{in}} \wedge \exists X_H\, \gamma_H) \Rightarrow (\alpha_G^{\text{in}} \wedge \exists X_G\, \gamma_G)))$$

The intuition behind $P_{\preceq}$ is that if an output label ℓ is possible in G then ℓ must be possible in H, and vice versa for input labels. We get the following corollary by using the definition $P_{\preceq}$, Lemma 3, and Theorem 2.

Corollary 1. $G \not\preceq H \iff \exists \mathbf{a} \in Tr(G \otimes H)(\lceil G \otimes H \rceil(\mathbf{a}) \models \neg P_{\preceq}(G,H))$.

In the following theorem we assume, without loss of generality, that $L_{\lceil G \rceil} = L_{\lceil H \rceil}$. The proof uses Lemma 3.

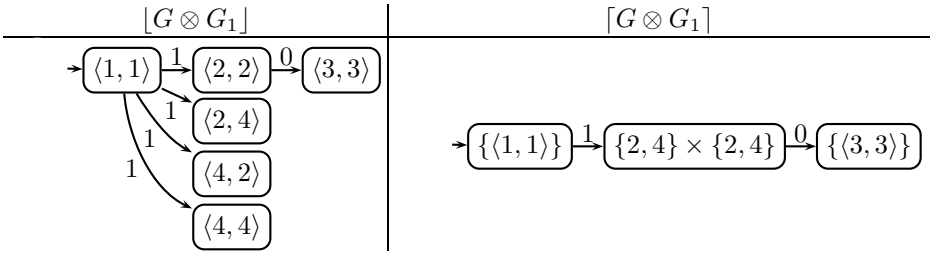
Theorem 4. *Assume G is input-robust and H is output-robust. $G \preceq_n H \iff$ forall $\mathbf{a} \in Tr(G \otimes H)$, if length$(\mathbf{a}) < n$ then $\lceil G \otimes H \rceil(\mathbf{a}) \models P_{\preceq}(G,H)$.*

The robustness assumptions are not needed for the direction $\Longleftarrow$ of the theorem. It is easy to show that the direction $\Longrightarrow$ does not hold without the assumption.

Example 11. Consider the GLAS G illustrated by the FSM in Example 6. 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 $\lceil G \rceil \preceq \lceil G_1 \rceil$ (since $\lceil G \rceil \preceq \lceil G \rceil$ by reflexivity of $\preceq$). Now consider $G \otimes G_1$, where

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

The LTSs $\lfloor G \otimes G_1 \rfloor$ and $\lceil G \otimes G_1 \rceil$ can be illustrated as follows where a pair $\langle z, z_1 \rangle$ shows the values of the respective model variables:



Consider the singleton trace 1. Fix $M = \{z \mapsto 2, z_1 \mapsto 4\} \models \lceil G \otimes G_1 \rceil(1)$. It is easy to see that $M \not\models P_{\preceq}(G, G_1)$ because $M \cup \{\ell \mapsto 0\} \not\models z_1 = 2$. ⊠

The following example illustrates a case when H in Theorem 4 is nondeterministic but robust.

Example 12. 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 c <> [] and c <= Head(cs) and c >= 0
   cs := Tail(cs)
```

The GLASs *Credits2* (from Example 9) and *CreditsImpl* are robust. On can show that *CreditsImpl* $\preceq_n$ *Credits2* for any n by using the product encoding and Theorem 4. ⊠

Theorem 4 identifies conditions where we can use standard techniques for verification of safety formulas. We use this in Theorem 5 to formulate checking for $P_{\preceq}(G,H)$ as a symbolic bounded model checking problem.

Theorem 5. *Assume that G is input-robust and H is output-robust. There is an effective procedure that given G, H and a bound $n > 0$, creates a formula $BNC(G,H,n)$ of size $O(n(|G| + |H|))$ with free variables ℓ_i for $i < n$, such that $BNC(G,H,n)$ is satisfiable iff $G \not\preceq_n H$, and if $M \models BNC(G,H,n)$ then for some a, and $m < n$, $(\ell_0^M, \ldots, \ell_m^M, a)$ is a witness of $G \not\preceq_n H$.*

Proof. Given a GLAS G, we can characterize the set of states reachable after n steps by unfolding of the transition relation of G n times. The corresponding formula is $Reach(G,n) \stackrel{\text{def}}{=} \imath_G \wedge \bigwedge_{i=0}^{n-1} TR_G(\Sigma_G^i, \ell_i, \Sigma_G^{i+1})$ where $\Sigma_G^0 = \Sigma_G$ and $\Sigma_G^{i+1} = (\Sigma_G^i)'$. The bounded non-conformance checking formula $BNC(G,H,n)$ is now $Reach(G \otimes H, n) \wedge \bigvee_{i=0}^{n} \neg P_{\preceq}(G,H)(\Sigma_{G\otimes H}^i)$. The formula is satisfiable if and only if $P_{\preceq}(G,H)$ is violated within n steps. The size of the formula is $O(n(|G \otimes H| + |\neg P_{\preceq}(G,H)|))$. Theorem 4 ensures that it suffices to check $P_{\preceq}$ as a state invariant. ⊠

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 [18] provided that δ is part of the output labels.

Definition 18. 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 $\mathbf{a} \cdot a \in Tr(\mathcal{L})$.

Theorem 6. *If $\lceil G \rceil$ is input-enabled then $\lceil G \rceil$ ioco $\lceil H \rceil \iff G \preceq H$.*

Proof. Assume $\lceil G \rceil$ is input-enabled. ($\Longrightarrow$): Assume $G \not\preceq H$. We show that $\lceil G \rceil$ *ioco* $\lceil H \rceil$ does not hold. From Definition 12 follows that there exists a trace $\mathbf{a}$ such that $S_G^0 \stackrel{\mathbf{a}}{\rightarrow} S'_G$ and $S_H^0 \stackrel{\mathbf{a}}{\rightarrow} S'_H$, and there is a label a such that either

[3] Such LTSs are called *input-output transition systems* in [18].

1) a is a output-label that is enabled in S'_G but not enabled in S'_H, or 2) a is a input-label that is enabled in S'_H but not enabled in S'_G. The second case cannot be true since $\lceil G \rceil$ is input-enabled. Thus, there is 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 $\mathbf{a} \cdot a \notin Tr(\lceil H \rceil)$. ($\Longleftarrow$): Assume $\lceil G \rceil$ *ioco* $\lceil H \rceil$ does not hold. We show that $G \not\preceq H$. From Definition 18 follows that 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 $\mathbf{a} \cdot a \notin Tr(\lceil H \rceil)$. Now use Lemma 3. ⊠

5 Related work

The current paper generalizes the notion of model programs to GLASs and generalizes the results in [21] related to deterministic input-output model programs to GLASs. We introduced the notion of robustness as a nontrivial extension of deterministic GLASs by supporting "safe" internal nondeterminism, while retaining the property that non i/o-refinement checking reduces to safety analysis.

The literature on ioco [6,19,20] and various extension of ioco is extensive. A recent overview and the formal foundations are described in [18]. An extension of ioco theory to symbolic transition systems is proposed in [13]. Composition of GLASs is related to composition of symbolic transition systems [12]. 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 [13,12].

We believe that GLASs can 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 is a given state. Composition of Event-B models is discussed in [17,7].

BMPC [23], that is used in Section 4, is a generalization of SMT based bounded model checking [11] to GLASs. The notion of *i/o-refinement* of GLASs builds on the game view of systems [8], that can also be used to formulate other problems related to input-output GLASs, such as finding winning strategies to reach certain goal states.

References

1. Abrial, J.-R.: The B-Book: Assigning programs to meanings. Cambridge University Press, Cambridge (1996)
2. Abrial, J.-R., Hallerstede, S.: Refinement, decomposition and instantiation of discrete models: Application to Event-B. Fundamenta Informaticae 77(1-2), 1–28 (2007)
3. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.: Alternating refinement relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 163–178. Springer, Heidelberg (1998)
4. AsmL, `http://research.microsoft.com/fse/AsmL/`

5. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories, ch. 26. Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 825–885. IOS Press, Amsterdam (2009)
6. Brinksma, E., Tretmans, J.: Testing Transition Systems: An Annotated Bibliography. In: Cassez, F., Jard, C., Rozoy, B., Dermot, M. (eds.) MOVEP 2000. LNCS, vol. 2067, pp. 187–193. Springer, Heidelberg (2001)
7. Butler, M.: Decomposition structures for Event-B. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, pp. 20–38. Springer, Heidelberg (2009)
8. de Alfaro, L.: Game models for open systems. In: Dershowitz, N. (ed.) Verification: Theory and Practice. LNCS, vol. 2772, pp. 269–289. Springer, Heidelberg (2004)
9. Alfaro, L.d., Henzinger, T.A.: Interface automata. In: ESEC/FSE, pp. 109–120. ACM, New York (2001)
10. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
11. de Moura, L., Rueß, H., Sorea, M.: Lazy theorem proving for bounded model checking over infinite domains. In: Voronkov, A. (ed.) CADE 2002. LNCS (LNAI), vol. 2392, pp. 438–455. Springer, Heidelberg (2002)
12. Frantzen, L., Tretmans, J., Willemse, T.: A symbolic framework for model-based testing. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES 2006 and RV 2006. LNCS, vol. 4262, pp. 40–54. Springer, Heidelberg (2006)
13. Franzen, L., Tretmans, J., Willemse, T.: Test generation based on symbolic specifications. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. LNCS, vol. 3395, pp. 1–15. Springer, Heidelberg (2005) (to appear)
14. Jacky, J., Veanes, M., Campbell, C., Schulte, W.: Model-based Software Testing and Analysis with C#. Cambridge University Press, Cambridge (2008)
15. Keller, R.: Formal verification of parallel programs. Communications of the ACM, 371–384 (July 1976)
16. Lynch, N., Tuttle, M.: Hierarchical correctness proofs for distributed algorithms. In: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, pp. 137–151. ACM Press, New York (1987)
17. Poppleton, M.: The composition of Event-B models. In: Börger, E., Butler, M., Bowen, J.P., Boca, P. (eds.) ABZ 2008. LNCS, vol. 5238, pp. 209–222. Springer, Heidelberg (2008)
18. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS, vol. 4949, pp. 1–38. Springer, Heidelberg (2008)
19. Tretmans, J., Belinfante, A.: Automatic testing with formal methods. In: EuroSTAR 1999: 7th European Int. Conference on Software Testing, Analysis & Review, Barcelona, Spain, November 8–12, EuroStar Conferences, Galway, Ireland (1999)
20. van der Bij, M., Rensink, A., Tretmans, J.: Compositional testing with ioco. In: Petrenko, A., Ulrich, A. (eds.) FATES 2003. LNCS, vol. 2931, pp. 86–100. Springer, Heidelberg (2004)
21. Veanes, M., Bjørner, N.: Input-Output Model Programs. In: Leucker, M., Morgan, C. (eds.) Theoretical Aspects of Computing - ICTAC 2009. LNCS, vol. 5684, pp. 322–335. Springer, Heidelberg (2009)
22. Veanes, M., Bjørner, N.: Alternating Simulation and IOCO. Technical Report MSR-TR-2010-38, Microsoft Research (April 2010)
23. Veanes, M., Bjørner, N., Gurevich, Y., Schulte, W.: Symbolic bounded model checking of abstract state machines. Int. J. Software Informatics 33(2-3), 1–22 (2009)