Symbolic Model based Testing for Component oriented Systems

Alain Faivre[1], Christophe Gaston[1] and Pascale Le Gall[2⋆]

[1] CEA LIST Saclay F-91191 Gif sur Yvette
email : {alain.faivre,christophe.gaston}@cea.fr
[2] Université d'Évry, IBISC - FRE CNRS 2873,
523 pl. des Terrasses F-91000 Évry
email : pascale.legall@ibisc.univ-evry.fr

Abstract. In a component oriented approach, components are designed, developed and validated in order to be widely used. However one cannot always foresee which specific uses will be made of components depending on the system they will constitute. In this paper we propose an approach to test each component of a system by extracting accurate behaviours using information given by the system specification. System specifications are defined as input/output symbolic transition systems structured by a communication operator (synchronized product) and an encapsulation operator (hiding communication channels). By projecting symbolic execution of a system on its components, we derive unitary symbolic behaviours to be used as test purposes at the component level. In practice, those behaviours can be seen as typical behaviours of the component in the context of the system. We will illustrate on an example that those behaviours could not have been extracted by reasoning uniquely at the component level.
Keywords: component based system, ioco-based conformance testing, input/output symbolic transition system, symbolic execution.

1 Introduction

In the framework of reactive systems, a component oriented system is constituted of components continuously interacting together and with their environment by means of communication mechanisms. In a first step, basic components are usually specified, implemented and tested: this is called *unitary testing*. Then, the complete system is specified, implemented and tested taking into account the component based structure: this is called *integration testing*. Concerning integration testing, two main approaches can be followed depending on the targeted fault model. In the first approach, the global system is tested according to behaviors involving communication mechanisms, focusing on cases for which those mechanisms are not observable (*i.e* internal communications). Obviously, this

⋆ This work was partially supported by the RNRT French project STACS and the RNTL French project EDEN2 .

approach is used when the targeted fault model mainly deals with communication mechanisms as in [9,5,1]. In the second approach, the global system is tested by selecting behaviors of basic components that are typically activated in the system. It amounts to re-enforce unitary testing with respect to those behaviors. In terms of fault model, the counterpart of this approach is that communication mechanisms are supposed to be correctly implemented and correctly used by programmers. Thus, in this case, a non conformance of the system should only result of uncorrect implementations of components. [13] has proposed a theoretical framework based on these assumptions and has stated results concerning preservation of conformance through component composition. In this contribution, our objective is to re-enforce testing of components and intermediate sub-systems. Now, the question is: how to choose behaviors to re-enforce component and sub-system testing in order to make them more reliable in the context of the system? In fact, when a sub-system is involved in a more complex one, it is very probable that all the sub-system behaviors are not activated. In this paper, the models that we use to denote specifications of communicating systems are made of simple input/output symbolic transition systems (IOSTS) ([4,6,3]) for denoting basic components, and of two structuring operators, namely *composition* and *hiding* (as in [13]). Those models based on input/output symbolic transition systems are equipped with naming mechanisms that allow us to easily retrieve all relevant information concerning sub-systems. Those naming mechanisms together with symbolic execution technics [7] are used to define relevant behaviors of sub-systems. Moreover, we show how to use those behaviors as test purposes in an *ioco*-based [11,12,3,4] conformance testing framework. From a technical point of view, this contribution is an extension of the one presented in [4] for component oriented system testing. As we do not make any assumption concerning the communication mechanisms, a system (implementation) is considered as conformant with respect to a structured specification if it has the same structure, if for each intermediate subspecification, there exists a subsystem corresponding to it, and if each subsystem is conformant according to the ioco conformance relation with respect to the corresponding subspecification.

The paper is organized as follows. In Section 2, we present the IOSTS formalism, the notion of basic component based system and the notion of (sub-)system. In Section 3, we show how to define test purposes from symbolic execution of such systems and how to project them on any sub-system. In Section 4, we define our symbolic test purposes. Section 5 is a conclusion.

2 Structured Input/Output Symbolic Transition Systems

IOSTS are used to represent behaviors of reactive systems. Those behaviors are composed of internal actions and communication actions which are emissions or receptions of values through *channels*. Internal states are modeled by assignments of particular variables called *attributes*.

2.1 Basic definitions of IOSTS

We use the following set theory notations. The set of functions of domain A and codomain B is denoted B^A. $\coprod$ stands for the disjoint union.

For any set X, $Ident_X$ denotes the identity function on X. For any two functions $f : A \to B$ and $g : C \to D$ such that $A \cap C = \emptyset$, $f|g : A \cup C \to B \cup D$ is the function such that $f|g(x) = f(x)$ if $x \in A$ and $f|g(x) = g(x)$ otherwise. Moreover, for any $E \subseteq A$, $f|_E$ is the restriction of f to E. A data type signature is a couple $\Omega = (S, Op)$ where S is a set of type names, Op is a set of operation names, each of them being provided with a profile $s_1 \cdots s_{n-1} \to s_n$ (for $i \leq n$, $s_i \in S$). Let $V = \bigcup_{s \in S} V_s$ be a set of typed variable names. The set of *Ω-terms* with variables in V is denoted $T_\Omega(V) = \bigcup_{s \in S} T_\Omega(V)_s$ and is inductively defined as usual over Op and V. $Type : T_\Omega(V) \to S$ is the function such that for each $t \in T_\Omega(V)_s$, $Type(t) = s$. In the following, we overload the notation $Type$ by defining $Type(X) = s$ for any set $X \subseteq V_s$. $T_\Omega(\emptyset)$ is simply denoted T_Ω. An *Ω-substitution* is a function of $T_\Omega(V)^V$ preserving types. Any substitution may be canonically extended to terms. The set $Sen_\Omega(V)$ of all typed equational *Ω-formulae* contains the truth values $true$, $false$ and all formulae built using the equality predicates $t = t'$ for $t, t' \in T_\Omega(V)_s$, and the usual connectives $\neg, \vee, \wedge$. A *Ω-model* is a family $M = \{M_s\}_{s \in S}$ with, for each $f : s_1 \cdots s_n \to s \in Op$, a function $f_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$. We define *$\Omega$-interpretations over V* as applications of M^V preserving types, that are also extended to terms of $T_\Omega(V)$. A model M satisfies a formula φ, denoted by $M \models \varphi$, if and only if, for all interpretations ν, $M \models_\nu \varphi$, where $M \models_\nu t = t'$ iff $\nu(t) = \nu(t')$, and where the truth values and the connectives are handled as usual. Given a model M and a formula φ, φ is said *satisfiable* in M, if there exists an interpretation ν such that $M \models_\nu \varphi$. In the sequel, we suppose that data types of our IOSTS correspond to the generic signature $\Omega = (S, Op)$ and are interpreted in a fixed model M.

IOSTS-signatures are composed of a set of particular variables called *Attributes* and of a set of *Channel names*.

Definition 1. (*IOSTS*-signature) *An IOSTS-*signature *is a couple $(Att, Chan)$ such that $Att = \bigcup_{s \in S} Att_s$. For any two IOSTS-signatures $\Sigma_i = (Att_i, Chan_i)$ with $i \in \{1, 2\}$, the* union *of Σ_1 and Σ_2, denoted $\Sigma_1 \cup \Sigma_2$ is the IOSTS-signature $(Att_1 \coprod Att_2, Chan_1 \cup Chan_2)$.*

Union of signatures does not collapse attributes. Even though Att_1 and Att_2 contain a common variable name x, the union $\Sigma_1 \cup \Sigma_2$ distinguishes the two occurrences of x. On the contrary, channel names are used to synchronize communication actions and thus, are shared by a simple identification in the union.

Definition 2. (Actions) *The* set of communication actions *over $\Sigma = (Att, Chan)$, denoted $Act(\Sigma)$, is the set $Input(\Sigma) \cup Output(\Sigma) \cup \{\tau\}$, where:*

$Input(\Sigma) = \{c?Y \mid c \in Chan, \exists s \in S, Y \subset Att_s\}$

$Output(\Sigma) = \{c!t \mid c \in Chan, t \in T_\Omega(Att)\}$

$c?Y$ denotes the awaiting of a value to be received through the channel c and to be stored on all variables of Y. In the sequel, when Y is a singleton $\{y\}$, we can note $c?y$ instead of $c?\{y\}$. $c!t$ denotes the emission of the value t through the channel c and τ is an internal action without any communication action.

We enrich basic-IOSTS of [4] with a naming mechanism associating to each transition a name chosen in a set TN of *transition names*.

Definition 3. (*IOSTS*) *An IOSTS* over a signature $\Sigma = (Att, Chan)$ *is a triple* $G = (State, init, Trans)$ *defined by a set* $State$ *of* state names*, an* initial state $init \in State$*, and a set of* transitions $Trans \subseteq TN \times (State \times Act(\Sigma) \times Sen_\Omega(Att) \times T_\Omega(Att)^{Att} \times State)$. $\mathcal{STS}$ *denotes the set of all IOSTS.*

In the sequel, for any transition tr of the form $(n, (q, act, \varphi, \rho, q'))$, $name(tr)$ stands for n and is called the *name of* tr, $source(tr)$ (resp. $target(tr)$) stands for q (resp. q') and is called the *source state of* tr (resp. *target state of* tr), $act(tr)$ stands for act and is called the *communication action of* tr, $guard(tr)$ stands for φ and is called the *guard of* tr, $subst(tr)$ stands for ρ and defines how the attributes are modified when the transition is fired. Finally, $body(tr)$ stands for $(q, act, \varphi, \rho, q')$. For an IOSTS G, $Sig(G)$, $Att(G)$, $Chan(G)$, $State(G)$, $init(G)$ and $Trans(G)$ resp. stand for Σ, Att, $Chan$, $State$, $init$ and $Trans$.

Definition 4. (Runs of a transition) *With notations of Def. 3, let* $tr \in Trans$*. Let us note* $Act(M) = (Chan \times \{?, !\} \times M) \cup \{\tau\}$*. The set* $Run(tr) \subseteq M^{Att} \times Act(M) \times M^{Att}$ *of* execution runs *of* tr *is s. t.* $(\nu^i, act_M, \nu^f) \in Run(tr)$ *iff:*

- *if* $act(tr)$ *is of the form* $c!t$ *(resp.* τ*) then* $M \models_{\nu^i} guard(tr)$*,* $\nu^f = \nu^i \circ subst(tr)$ *and* $act_M = c!\nu^i(t)$ *(resp.* $act_M = \tau$*),*
- *if* $act(tr)$ *is of the form* $c?Y$ *then* $M \models_{\nu^i} guard(tr)$*, there exists* ν^a *such that* $\nu^a(z) = \nu^i(z)$ *for all* $z \notin Y$ *and for any* $x, y \in Y$ $\nu^a(x) = \nu^a(y)$*,* $\nu^f = \nu^a \circ subst(tr)$ *and* $act_M = c?\nu^a(y)$ *for an arbitrary* $y \in Y$*.*

For $r = (\nu^i, act_M, \nu^f)$*, we note* $source(r)$*,* $act(r)$*,* $target(r)$ *resp.* ν^i*,* act_M*,* ν^f*.*

As in [3], we will use $\delta!$ to denote under which semantic conditions an IOSTS is *quiescent*: quiescence refers to situations for which it is not possible to fire an output transition but only possibly input transitions or τ transitions.

Definition 5. (Suspension traces and *IOSTS* semantics) *The set of finite paths in* G*, denoted* $FP(G)$ *contains all finite sequence* $p = tr_1 \dots tr_n$ *of transitions in* $Trans(G)$ *such that* $source(tr_1) = init(G)$ *and for all* $i < n$*,* $target(tr_i) = source(tr_{i+1})$*. The set of* runs *of* p *denoted* $Run(p)$ *is the set of sequences* $r = r_1 \dots r_n$ *such that for all* $i \leq n$*,* r_i *is a run of* tr_i *and for all* $i < n$*,* $target(r_i) = source(r_{i+1})$*. We note* $Tr(r) = act(r_1) \dots act(r_n)$*. The set of* suspension traces of a run r of a finite path p*, with* $r \in Run(p)$*, denoted* $STr(p, r)$ *is the least set s. t.:*

- *If* p *can be decomposed as* $p'.tr$ *with* $tr \in Trans(G)$ *and with* r *of the form* $r'.r_{tr}$ *with* $r_{tr} \in Run(tr)$*, then* $\{m.act(r_{tr}) | m \in STr(p', r')\} \subseteq STr(p, r)$*.*

- *If there exists no finite path* $p.p'$ *for which there exists* $r.r_1 \cdots r_k \in Run(p.p')$ *with for all* $i \leq k-1$*,* $act(r_i) = \tau$ *and* $act(r_k) = c!m$ *for some* c *and* m*, then for any*[3] $\delta_m \in \{\delta!\}^*$*,* $Tr(r).\delta_m \in STr(p, r)$*.*

The set of suspension traces of a path p *is* $STr(p) = \bigcup_{r \in Run(p)} STr(p, r)$ *and* semantics *of* G *are* $STr(G) = \bigcup_{p \in FP(G)} STr(p)$*.*

[3] A^* denotes the set of finite sequences of elements of A

2.2 Systems

We introduce the concept of *library* which intuitively allows us to characterize a set of IOSTS denoting basic components from which systems can be built. Formally a library is a set of couples, each of them being constituted of an IOSTS name and an IOSTS definition. IOSTS names are chosen in a given set BN whose elements are called *basic-IOSTS names*.

Definition 6. (Library) *A* library *is a set $\mathcal{B}$ whose elements are of the form (n, G) where $n \in BN$ and $G \in \mathcal{STS}$, s. t. for any two $(n_1, G_1), (n_2, G_2)$ in $\mathcal{B}$, $n_1 = n_2$ iff $G_1 = G_2$. If $G_1 \neq G_2$, for any $t_1 \in Trans(G_1)$ and $t_2 \in Trans(G_2)$, $name(t_1) \neq name(t_2)$. Elements of a library are called* basic-$IOSTS$.

In the sequel we consider a library $\mathcal{B}$ and we note $BN(\mathcal{B}) = \{n \mid (n, G) \in \mathcal{B}\}$ and $Chan(\mathcal{B}) = \{c \mid \exists (n, G) \in \mathcal{B}, c \in Chan(G)\}$. A *system over a library $\mathcal{B}$* is built from IOSTS of $\mathcal{B}$ using two structuring mechanisms: *composition* which is used to aggregate two systems by connecting common channels and *hiding* is used to internalize some channels inside the system (they are no more visible from the environment). As for basic-IOSTS, we denote any system by a name and an IOSTS. The name associated to a system reflects the structure of the system. The set $SN(\mathcal{B})$ of *system names over $\mathcal{B}$* is defined as follows:

- for any $n \in BN(\mathcal{B})$, $n \in SN(\mathcal{B})$, (a basic-IOSTS is also a system),
- for any $n_1, n_2 \in SN(\mathcal{B})$, $(n_1 \otimes n_2) \in SN(\mathcal{B})$ (corresponding to the system obtained by composing two systems named resp. n_1 and n_2),
- for any $n \in SN(\mathcal{B})$ and $C \subseteq Chan(\mathcal{B})$, $Hide(C, n) \in SN(\mathcal{B})$ (corresponding to the system obtained by hiding channels of C in the system named n).

Intuitively, for any system, transitions introduced in its associated IOSTS are defined over transitions of basic-IOSTSs composing the system, mainly by synchronization mechanisms. In order to be able to identify basic transitions involved in system transitions, the name associated to system transitions will explicit the underlying synchronization mechanism. Therefore, those names are of the form $(o, \{i_1, \cdots, i_n\})$ where o is a name of basic output-transition or a τ-transition and $i_1, \cdots, i_n$ are names of basic input-transitions (with possibly $n = 0$). Roughly speaking, the name $(o, \{i_1, \cdots, i_n\})$ generally refers to the synchronization of a basic output-transition named o with basic input-transitions named $i_1, \cdots, i_n$. Let us point out some particular cases. Any transition obtained by synchronizing input-transitions named $i_1, \cdots, i_n$ with an emission of the environment is denoted $(\varepsilon, \{i_1, \cdots, i_n\})$ where ε denotes the absence of output-transition. Any τ-transition in a system has a name of the form $(n, \emptyset)$ where n is the name of some underlying basic τ-transition. The set of *system transition names*, denoted STN, is then the set $(TN \cup \{\varepsilon\}) \times 2^{TN}$ where $\varepsilon \notin TN$.

We now define systems over a library by means of three constructions: renaming to convert a basic-IOSTS into a system, composition and hiding.

Definition 7. (Systems over $\mathcal{B}$) *The set $Sys(\mathcal{B})$ of* systems over $\mathcal{B}$ *is the subset of $SN(\mathcal{B}) \times \mathcal{IOSTS}$ defined as follows:*

Renaming: *For any* $(n, G) \in \mathcal{B}$ *and* $t \in Trans(G)$*, let us define* $sn(t) = (name(t), \emptyset)$ *if* $act(t) = \tau$ *or* $act(t) \in Output(\Sigma)$ *and* $sn(t) = (\varepsilon, \{name(t)\})$ *otherwise. Let us define* $R(Trans(G)) = \bigcup_{t \in Trans(G)} \{(sn(t), body(t))\}$*.*
$(n, (State(G), Init(G), R(Trans(G))))$ *is in* $Sys(\mathcal{B})$*.*

Composition: *For any two systems* (n_1, G_1) *and* (n_2, G_2) *of* $Sys(\mathcal{B})$*, let us note* $G = (State(G_1) \times State(G_2), (init(G_1), init(G_2)), Trans)$ *the* $IOSTS$ *over* $Sig(G_1) \cup Sig(G_2)$ *where* $Trans$ *is defined as follows:*

- *If* $((o_1, i_1), (q_1, c!t, \varphi_1, \rho_1, q_1')) \in Trans(G_1)$ *and* $((\varepsilon, i_2), (q_2, c?Y, \varphi_2, \rho_2, q_2')) \in Trans(G_2)$ *and such that* $type(t) = type(Y)$*, then*
  $t = ((o_1, i_1 \cup i_2), ((q_1, q_2), c!t, \varphi_1 \wedge \varphi_2, \rho_1 \mid \rho_2[Y \leftarrow t], (q_1', q_2'))) \in Trans$*.*
- *If* $((o_1, i_1), (q_1, c!t, \varphi, \rho, q_1')) \in Trans(G_1)$*, for all* $q_2 \in State(G_2)$ *let us note* $tr_1, \cdots, tr_n$ *all transitions of the form* $tr_i = (n_i, (q_2, c?Y_i, \varphi_i, \rho_i, q_i'')) \in Trans(G_2)$ *for which* $type(Y_i) = type(t)$*. Let us note* $guard = \wedge_{i \leq n} \neg \varphi_i$ *if* $n > 0$ *and* $guard = true$ *otherwise. Then*
  $t = ((o_1, i_1), ((q_1, q_2), c!t, \varphi \wedge guard, \rho \mid Ident_{Att(G_2)}, (q_1', q_2))) \in Trans$*.*
- *For any two transitions of the form* $((\varepsilon, i_1), (q_1, c?Y_1, \varphi_1, \rho_1, q_1')) \in Trans(G_1)$*, and* $((\varepsilon, i_2), ((q_2, c?Y_2, \varphi_2, \rho_2, q_2')) \in Trans(G_2)$ *such that* $type(Y_1) = type(Y_2)$*, then*
  $t = ((\varepsilon, i_1 \cup i_2), ((q_1, q_2), c?(Y_1 \cup Y_2), \varphi_1 \wedge \varphi_2, \rho_1 \mid \rho_2, (q_1', q_2'))) \in Trans$*.*
- *If* $((\varepsilon, i_1), (q_1, c?Y, \varphi, \rho, q_1')) \in Trans(G_1)$*, for all* $q_2 \in State(G_2)$*, let us note* $tr_1, \cdots, tr_n$ *all transitions of the form* $tr_i = (n_i, (q_2, c?Y_i, \varphi_i, \rho_i, q_i'')) \in Trans(G_2)$ *for which* $type(Y_i) = type(Y)$*. Let us note* $guard = \wedge_{i \leq n} \neg \varphi_i$ *if* $n > 0$ *and* $guard = true$ *otherwise. Then*
  $t = ((\varepsilon, i_1), ((q_1, q_2), c?Y, \varphi \wedge guard, \rho \mid Ident_{Att(G_2)}, (q_1', q_2))) \in Trans$*.*
- *If* $((o_1, \emptyset), (q_1, \tau, \varphi_1, \rho, q_1')) \in Trans_1$*, for all* $q_2 \in State(G_2)$*, then*
  $((o_1, \emptyset), ((q_1, q_2), \tau, \varphi_1, \rho \mid Ident_{Att(G_2)}, (q_1', q_2))) \in Trans$*.*
- *The role of* G_1 *and* G_2 *can be permuted in all rules described above.*

$((n_1 \otimes n_2), G)$ *is in* $Sys(\mathcal{B})$*.*

Hiding: *For any* $(n, G) \in Sys(\mathcal{B})$*, for any* $C \subseteq Chan(G)$*, let us note* $G' = (State(G), init(G), Trans')$ *where* $Trans'$ *is defined as follows:*

- *For any* $tr \in Trans(G)$ *where* $act(tr)$ *is either of the form* τ*,* $c!t$ *or* $c?X$ *for some* $c \notin C$*, then* $tr \in Trans'$*.*
- *For any* $tr \in Trans(G)$ *where* $act(tr)$ *is of the form* $c!t$ *with* $c \in C$*, then* $(name(tr), (source(tr), \tau, guard(tr), subst(tr), target(tr))) \in Trans'$*.*

$(Hide(C, n), G')$ *is in* $Sys(\mathcal{B})$*.*

Systems inherit all notations from the underlying IOSTS framework: for any system $sys = (n, G)$, $Sig(sys)$ stands for $Sig(G)$, $Att(Sys)$ stands for $Att(G)$... In the same way, semantics of sys are the set of suspension traces of G: $STr(sys) = STr(G)$. Note that for composition, emissions and receptions are not blocking: if no transition can be synchronized with an input (resp. output)-transition tr, then tr is synchronized with the environment. A synchronization involves at most one output-transition: when several output transitions sharing the same source state could be considered at the same time to define a synchronization, this leads to non-determinism. The hiding operation make unobservable actions $c!t$ when c is in C but this operation is non blocking (the output-transition introducing

$c!t$ is kept by replacing the communication action by τ). The hiding operation is blocking for inputs $c?X$ for c in C: corresponding transitions are simply removed in $Hiding(C, n)$. We now define sub-systems involved in a given system.

Definition 8. (Sub-systems) *Let* $(n, G) \in Sys(\mathcal{B})$. *The* set of sub-systems *of* (n, G) *denoted* $SubS((n, G)) \subseteq Sys(\mathcal{B})$ *is inductively defined as follows:*

- *If* $n \in BN$ *then* $SubS((n, G)) = \{(n, G)\}$,
- *If* n *is of the form* $n_1 \otimes n_2$ *then* $SubS((n, G)) = \{(n, G)\} \cup SubS((n_1, G_1)) \cup SubS((n_2, G_2))$ *where* (n_1, G_1) *and* (n_2, G_2) *belongs to* $Sys(\mathcal{B})$,
- *If* n *is of the form* $Hide(C, n')$, *then* $SubS((n, G)) = \{(n, G)\} \cup SubS((n', G'))$ *where* (n', G') *belongs to* $Sys(\mathcal{B})$.

For any sub-system sys' of a system sys, we can identify for any transition tr of sys the underlying transition of sys' involved in the definition of tr. This transition when it exists is called the projection of tr on sys'.

Definition 9. (Projection of a transition) *Let* $sys \in Sys(\mathcal{B})$, $sys' \in SubS(sys)$ *and* $tr = ((o, i), b) \in Trans(sys)$. *The* projection of tr on sys' *is the transition, when it is defined,* $tr_{sys'} = ((o', i'), b') \in Trans(G')$ *s. t.* $o' = o$ *or* $o' = \varepsilon$ *and* $i' \subseteq i$.

The naming mechanism for system transitions in Definition 7 makes $((o', i'), b')$ unique when it exists. Intuitively, the name (o, i) captures all the subparts of the system whose state is modified by firing the transition tr. In particular, if (o, i) does not include names of transitions issued from the sub-system sys', it simply means that there is no modification of the state concerning the sub-system sys', and thus that there does not exist a corresponding transition $tr_{sys'}$.

2.3 An example of a slot machine

We consider a simple slot machine, named S and presented in Figure 1. The player can enter a bet into the slot machine and if he/she wins, he/she gets back the amount of his/her bet multiplied by 10. The system S is built from two basic-IOSTS, named resp. Int and SM for Interface and SlotMachine. Those two basic-IOSTS are composed and some channels, used for internal communications, are hidden. Thus the name of S is of the form $Hiding(C, Int \otimes SM)$ where:

- Int corresponds to the basic interface IOSTS between the environment (player) and the slot machine SM. When the system is reset (reception on int_start), the interface IOSTS waits for a bet from the player. The bet is refused when its amount is greater than 100. Otherwise, the IOSTS transmits to SM the amount of the bet and then, waits for a result, win or not, from the SM. Depending of the result, Int transmits to SM which gain should be given to the player.
- SM corresponds to the internal mechanism of the slot machine. It manages the different functionalities as appropriately updating the bank amount, deciding whether the player wins or not, and in the relevant cases, delivering cash to the player. For simplicity sake, the algorithm used to decide whether the player wins or not, is abstracted by a boolean non initialized variable w.

– C corresponds to all the channels used by Int and SM to communicate. That is, $C = \{int\_start, int\_bet, int\_wim, int\_amount, int\_cash\}$.

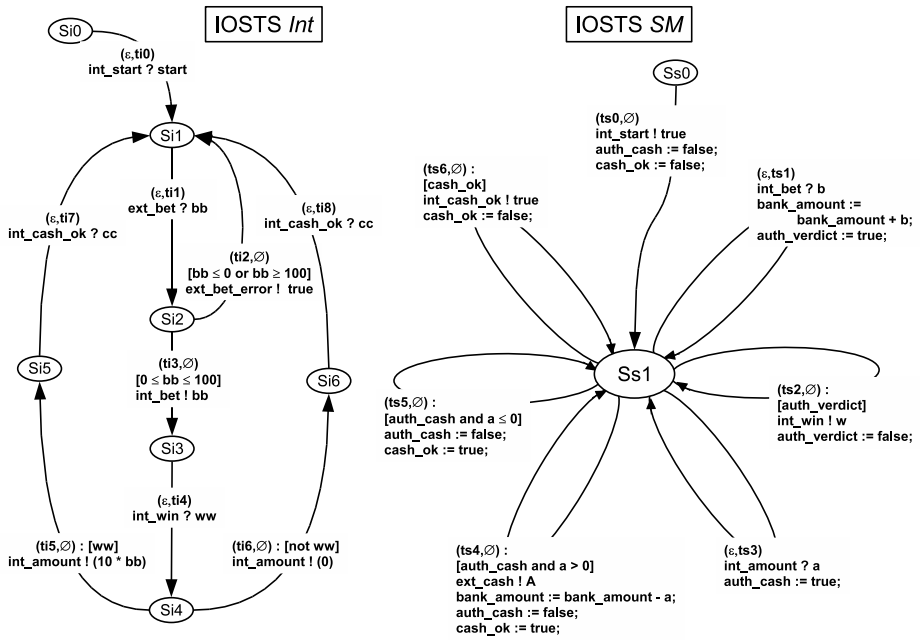

Fig. 1. An example of a slot machine

3 System based test Purposes for sub-systems

We show how we define for any system, some test purposes dedicated to test its sub-systems. Those test purposes will capture behaviors of sub-systems that typically occur in the whole system. This is done by combining symbolic execution technics and projection mechanisms.

3.1 Symbolic Execution

We call a *symbolic behavior* of a system sys any finite path p of sys for which $STr(p) \neq \emptyset$. In order to characterize the set of suspension traces of a symbolic behavior we propose to use a *symbolic execution* mechanism. Symbolic execution has been first defined for programs [7] and mainly consists in replacing concrete input values and initialization values of variables by symbolic ones in order to compute constraints induced on these variables by the execution of the program. Symbolic execution applied to IOSTS-based systems follows the same intuition considering guards of transitions as conditions and assignments together with communication actions as instructions. Herein, symbolic execution is presented as an adaptation of [4]. In the sequel, we assume that a set of fresh variables $F = \bigcup_{s \in S} F_s$ disjoint from the set of attribute variables $\coprod_{(n,G) \in \mathcal{B}} Att(G)$ is given. We now give the intermediate definition of *symbolic extended state* which is

a structure allowing to store information about a symbolic behavior: the system current state (target state of the last transition of the symbolic behavior), the path condition which characterizes a constraint on symbolic variables to reach this state, and the symbolic values associated to attribute variables. As compared to [4], we also add a fourth stored information: it is given in the form of a constraint on symbolic variables which is not computed during the symbolic execution of the system. It is called an *external constraint* and in practice it will be inherited from a projection mechanism.

Definition 10. (Symbolic extended state) *A* symbolic extended state of sys *is a quadruple* $\eta = (q, \pi, f, \sigma)$ *where* $q \in State(sys)$, $\pi \in Sen_\Omega(F)$ *is called a* path condition, $f \in Sen_\Omega(F)$ *is called an* external constraint *and* $\sigma \in T_\Omega(F)^{Att(sys)}$ *is called a* symbolic assignment of variables. $\eta = (q, \pi, f, \sigma)$ *is said to be satisfiable if* $\pi \wedge f$ *is satisfiable*[4]. *One notes* $\mathcal{S}(sys)$ *(resp.* $\mathcal{S}_{sat}(sys)$*) the set of all the (resp. satisfiable) symbolic extended states over* F.

For any symbolic extended state η of the form (q, π, f, σ), q is denoted $state(\eta)$, π is denoted $pc(\eta)$, σ is denoted $sav(\eta)$ (for symbolic assignment of variables) and f is denoted $ec(\eta)$. Now, we show how to give symbolic counterparts to transitions of a system. The idea is to consider any symbolic extended state defined over the source state of the transition, and to construct a new target symbolic extended state defined over the target state of the transition. The external constraint of the target symbolic extended state is a conjunction formed with the external constraint of the source symbolic extended state and a new external constraint (denoted ct in the following Definition). In the sequel, for any system sys, $Sig(sys, F)$ stands for the signature $(F, Chan(sys))$.

Definition 11. (Symbolic execution of a transition) *With notations of Definition 10, for any* $\eta \in \mathcal{S}(sys)$*, for any* $tr \in Trans(sys)$ *such that* $source(tr) = state(\eta)$*, a* symbolic execution of tr from η *is a triple* $st = (\eta, sa, \eta') \in \mathcal{S}(sys) \times Act(Sig(sys, F)) \times \mathcal{S}(sys)$ *such that there exists* $ct \in Sen_\Omega(F)$ *for which:*

- *if* $act(tr) = c!t$ *then* sa *is of the form* $c!z$ *for some* $z \in F$ *and* $\eta' = (target(tr), pc(\eta) \wedge sav(\eta)(guard(tr)) \wedge z = sav(\eta)(t), ec(\eta) \wedge ct, sav(\eta) \circ subst(tr))$,
- *if* $act(tr) = c?Y$ *then* sa *is of the form* $c?z$ *for some* $z \in F$ *and* $\eta' = (target(tr), pc(\eta) \wedge sav(\eta)(guard(tr)), ec(\eta) \wedge ct, sav(\eta) \circ (y \mapsto z)_{y \in Y} \circ subst(tr))$,
- *if* $act(tr) = \tau$ *then* $sa = \tau$ *and* $\eta' = (target(tr), pc(\eta) \wedge sav(\eta)(guard(tr)), ec(\eta) \wedge ct, sav(\eta) \circ subst(tr))$.

The definition of st only depends on tr, η, ct and the chosen variable z. Therefore, it is conveniently denoted $SE(tr, \eta, ct, z)$ (if $act(tr) = \tau$, z is useless). For any $st = (\eta, sa, \eta')$, $source(st)$ stands for η, $target(st)$ stands for η' and $act(st)$ stands for sa.

We now define symbolic execution of systems. Intuitively, a symbolic execution of a system sys is seen as a rooted tree whose paths are composed of

[4] Here $\pi \wedge f$ is *satisfiable* if and only if there exists $\nu \in M^F$ such that $M \models_\nu \pi \wedge f$.

sequences of symbolic executions of transitions which are consecutive in sys. The root is a symbolic extended state made of the initial state $init(sys)$, the path condition $true$, an arbitrary initialization σ_0 of variables of $Att(sys)$ in F, and an external constraint reduced to $true$ (no constraint at the beginning of the execution). Moreover, if a transition is symbolically executed with an external constraint ct, then it is also executed with the external constraint $\neg ct$.

Definition 12. (Symbolic execution of a system) *A* full symbolic execution of sys over F *is a triple* $sys_{symb} = (\mathcal{S}(sys), init, R)$ *with* $init = (init(sys), true, true, \sigma_0)$ *where* σ_0 *is an injective substitution in* $F^{Att(sys)}$ *and* $R \subseteq \mathcal{S}(sys) \times Act(Sig(sys, F)) \times \mathcal{S}(sys)$ *satisfies the following properties:*

- *for any* $\eta \in \mathcal{S}(sys)$*, for all* $tr \in Trans(sys)$ *such that* $source(tr) = state(\eta)$*, there exists exactly two constrained symbolic executions of* tr *in* R *respectively of the form* $SE(tr, \eta, ct, z)$ *and* $SE(tr, \eta, \neg ct, z)$*. Those two transitions are said to be* complementary*.*
- *for any* $(\eta^i, c\natural x, \eta^f) \in R$ *with* $\natural \in \{!, ?\}$*,* $\forall a \in Att(sys)$*, then* $\sigma_0(a) \neq x$*,*
- *for any* $(\eta^i, c\natural x, \eta^f) \in R$ *and* $(\eta'^i, d\sharp x, \eta'^f) \in R$ *with* $\natural, \sharp \in \{!, ?\}$ *which are not complementary, then* $x \neq y$*.*

The symbolic execution of sys over F associated to sys_{symb} *is the triple* $SE(sys) = (\mathcal{S}_{sat}(sys), init, R_{sat})$ *where* R_{sat} *is the restriction of* R *to* $\mathcal{S}_{sat}(sys) \times Act(Sig(sys, F)) \times \mathcal{S}_{sat}(sys)$*.*

We use the notation $FP(SE(sys))$ to denote the set of finite paths of $SE(sys)$. To define a run of a finite path p, we proceed as follows. We choose an interpretation $\nu : F \rightarrow M$ such that $M \models_\nu pc(\eta_f) \wedge ec(\eta_f)$ where η_f is the last symbolic extended state of p. Then for each (η, act, η') of p we associate a run $(\nu(sav(\eta)), act_M, \nu(sav(\eta')))$ where $act_M = \tau$ if $act = \tau$ and $act_M = c\natural\nu(z)$ if act is of the form $c\natural z$ with $\natural \in \{!, ?\}$. The sequence of such formed triples constitute a run of p. Note that the set of all runs of all finite paths of $FP(SE(sys))$ is exactly the set of all runs of all finite paths of sys in the sense of Definition 5 and this set is independent of the external constraints chosen to execute transitions. Those external constraints are simply used to partition symbolic behaviors. A trivial partitioning can be characterized by choosing $true$ as external constraints for executing any transition from any symbolic state. In this case the obtained symbolic execution is isomorphic to the one described in [4] which does not contain any external constraint. Besides note that any finite path p of a symbolic execution of sys characterizes a set of suspension traces obviously determined by its set of runs and the finite path corresponding to p in sys (See Definition 5). Therefore any symbolic execution of sys characterizes a set of suspension traces which can be easily proven to be this associated to sys in the sense of Definition 5. Now, since internal actions are not observable in black box testing, we propose to eliminate them as follows.

Definition 13. (τ-reduction of a constrained symbolic execution) *The* τ-reduction of $SE(sys)$ *is the triple* $SE(sys)_\tau = (\mathcal{S}_{sat}(sys), init, R^\tau_{sat})$*, where* $R^\tau_{sat} \subseteq \mathcal{S}_{sat}(sys) \times Act(Sig(sys, F)) \times \mathcal{S}_{sat}(sys)$ *is such that for any sequence* $st_1 \cdots st_n$ *where for all* $i \leq n$ $st_i \in R_{sat}$*:*

– *for all* $i \leq n-1$ $act(st_i) = \tau$, $source(st_{i+1}) = target(st_i)$ *and* $act(st_n) \neq \tau$,
– *either* $source(st_1) = init$ *or there exists* $st \in R_{sat}$ *such that* $target(st) = source(st_1)$ *and* $act(st) \neq \tau$,
then $(source(st_1), act(st_n), target(st_n)) \in R^{\tau}_{sat}$.

Note that $SE(sys)$ and $SE(sys)_\tau$ characterize the same suspension traces. However, we need in the sequel to be able to symbolically identify situations in which quiescence is allowed. This is done by adding symbolic transitions labeled by $\delta!$ in the $SE(sys)_\tau$.

Definition 14. (Quiescence enrichment) Quiescence enrichment of $SE(sys)$ *is the triple* $SE(sys)_\delta = (\mathcal{S}_{sat}(sys), init, R_\delta)$ *where* $R_\delta = R^{\tau}_{sat} \cup \Delta R_\delta$ *with* $\Delta R_\delta \subseteq \mathcal{S}_{sat}(sys) \times \{\delta!\} \times \mathcal{S}_{sat}(sys)$ *is such that for any* $\eta \in \mathcal{S}_{sat}(sys)$, *if we note* $out_\eta = \{tr_1, \cdots, tr_n\}$ *the set of all transitions* $tr_i \in R^{\tau}_{sat}$ *such that* $act(tr_i) \in output(Sig(sys, F))$, *if we note* $f \in Sen_\Omega(F)$ *the formula of the form true if* out_η *is empty and of the form* $\bigwedge_{i \leq n} \neg(pc(target(tr_i)) \wedge ec(target(tr_i)))$ *otherwise, if we note* $\eta' = (state(\eta), pc(\eta) \wedge f, ec(\eta), sav(\eta))$ *then* $(\eta, \delta!, \eta') \in \Delta R_\delta$.

An example of a slot machine: symbolic execution Figure 2 shows a sub-tree of the symbolic execution of the slot machine system presented in Figure 1, as carried out by the AGATHA tool ([8,2]).

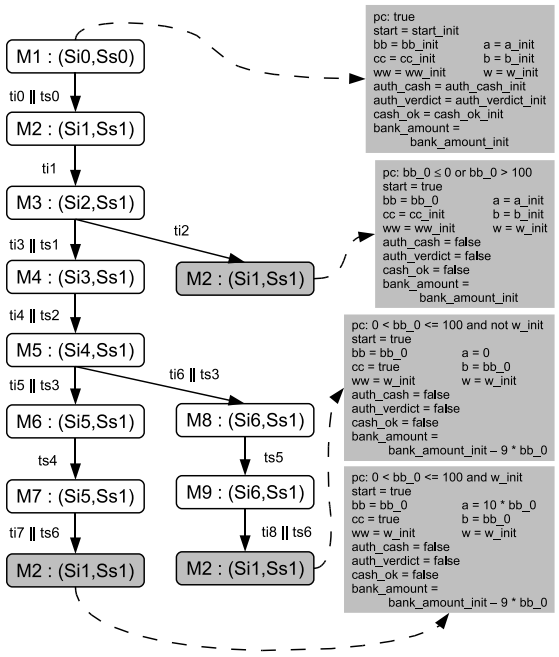

Fig. 2. Symbolic execution of the slot machine

External constraints for any two complementary transitions are resp. *true* and $false$ in the corresponding full symbolic execution. They never appear in the figure. We use the so-called *inclusion criteria* to end this execution. This criteria allows to stop symbolic execution when it detects that an encountered symbolic extended state is *included* in another already computed one. Intuitively, (q, π, f, σ) is included in (q', π', f', σ') if $q' = q$ and the constraints induced on $Att(sys)$ by σ and $\pi \wedge f$ are stronger than those induced by σ' and $\pi' \wedge f'$. The interested readers can refer to [10,4] for more formal definitions. Let us point out that the symbolic sub-tree of S computes three characteristic symbolic behaviors. The left path corresponds to a winning bet,

the middle path corresponds to a lost bet, and finally the right path corresponds to a forbidden bet. The initial and ending states are annotated with symbolic values of all attribute variables.

3.2 Symbolic behavior projections

For any finite path p of a symbolic execution of sys and a sub-system sys' of sys, we characterize the symbolic behavior $p_{sys'}$ of sys' involved in p. For this purpose, we begin by defining the *projection of a symbolic transition.*

Definition 15. (Projection of a symbolic transition) *Let sys be a system of $Sys(\mathcal{B})$. Let $sys' \in SubS(sys)$. Let $tr \in Trans(sys)$ such that $tr_{sys'}$ is defined. Let us note $st = SE(tr, \eta, ct, z)$ a symbolic execution of tr and $\eta_{sys'} \in \mathcal{S}_{sat}(sys')$ such that $state(\eta_{sys'}) = source(tr_{sys'})$. The* projection of st on sys' of source $\eta_{sys'}$ *is $SE(tr_{sys'}, \eta_{sys'}, pc(target(st)) \wedge ec(target(st)), z)$.*

The external constraint of the target state of the projection represents the constraints induced by the nature of the interactions of the sub-system with the other parts of the whole system. Now we generalize to symbolic behaviors.

Definition 16. (Projection of a path) *Let $p \in FP(SE(Sys))$. The* projection of p on sys' *denoted $p_{sys'} \in (\mathcal{S}_{sat}(Sys') \times Act(Sig(sys', F)) \times \mathcal{S}_{sat}(Sys'))^*$ together with its associated* target state *denoted $target(p_{sys'})$ are inductively mutually defined as follows:*

- *if p is of the form $st = SE(tr, init, ct, z) \in R_{sat}$ then let us note $\eta_{sys'} = (init(sys'), true, true, sav(init)|_{Att(sys')})$ then $p_{sys'}$ is the projection $st_{sys'}$ of st on sys' of source $\eta_{sys'}$ when it is defined, and in this case $target(p_{sys'}) = target(st_{sys'})$. Otherwise $p_{sys'}$ is the empty path and $target(p_{sys'}) = \eta_{sys'}$.*
- *if p is of the form $p'.st$ with $st = SE(tr, \eta, ct, z)$ then either the projection $st_{sys'}$ of st on sys' of source $target(p'_{sys'})$ is defined and: $p_{sys'} = p'_{sys'}.st_{sys'}$ and $target(p_{sys'}) = target(st_{sys'})$. Otherwise, $p_{sys'} = p'_{sys'}$ and $target(p_{sys'}) = target(p'_{sys'})$.*

Thus from any symbolic behavior of a system we can identify by projection symbolic behaviors of any sub-system whose external constraints reflect a usage of the sub-system in the whole system. Those projected behaviors are then good candidates to become behaviors to be tested on sub-systems: thus they will be chosen to construct test purposes.

4 Symbolic execution based conformance testing

4.1 Conformance testing and system-based test purposes

Model-based testing supposes that a conformance relation formally defines how are linked the specification G and the system under test SUT. Our work is based on the widely used *ioco* relation, initially designed for labeled transition systems

[11] and afterwards adapted for symbolic transition systems [6,3,4]. All the ioco-based testing settings consider that the SUT is a black-box system which can be observed only by its behavior given as input/output sequences. These sequences of observations may include the special output $\delta!$ indicating that the SUT is in a quescient state. The set of all traces, possibly including suspension transitions, which can be observed from SUT is denoted $STr(SUT)$. When dealing with IOSTS, data handled in these sequences are concrete values denoted by ground terms of T_Ω. By test hypothesis, the SUT is modeled as a labeled transition system S for which transitions are emissions (outputs), receptions (inputs) carrying concrete values and such that the set of suspension traces of S coincide with $STr(SUT)$. Moreover, as usual, the SUT is supposed to accept all inputs in all states (hypothesis of input-enabled system). Intuitively a SUT conforms to its specification G with respect to *ioco* if any SUT output (including $\delta!$) is specified in G provided that the sequence of input/output preceding the considered observation is also specified in G.

Definition 17. **(*ioco*)** *An input-enabled system SUT conforms to G iff for any* $tra \in STr(G) \cap STr(SUT)$*, if there exists* $act \in Act(M) \cup \{\delta!\}$ *of the form* $c!t$ *or* $\delta!$ *such that* $tra.act \in STr(SUT)$*, then* $tra.act \in STr(G)$*.*

A test execution consists in executing a transition system, called a test case, on the SUT in order to produce test verdicts. The test case and the SUT are synchronized by coupling emissions and receptions. Test purposes are used to select some behaviors to be tested. In a previous work [4], we have proposed to model test purposes as finite trees extracted from symbolic executions of G. Such a symbolic execution describes all the possible behaviors of G. Therefore it is equivalent to test the SUT by selecting paths in G or in a symbolic execution of G. Indeed, we have demonstrated the following completeness result : if an SUT does not conform to a specification G, then there exists a test purpose such that our corresponding testing algorithm can emit a verdict $FAIL$. The main advantage of characterizing test purposes from a symbolic execution of G is that the testing process can be expressed as a simultaneous traversal of both the symbolic execution and the test purpose. Verdicts are emitted according to the fact that the observed behavior, in the form of a sequence of inputs (stimulations) and outputs (observations), does or does not belong to the test purpose and to the symbolic execution. We have defined 4 verdicts: $WeakPASS$ when the behavior belongs to the test purpose and to at least one path of the symbolic execution which is not in the test purpose, $PASS$ when the behavior belongs to the test purpose and not to any path of the symbolic execution which does not belong to the test purpose, $INCONC$ (for inconclusive) when the behavior belongs to the symbolic execution and not to the test purpose, and finally $FAIL$ when the behavior belongs neither to the test purpose nor to the symbolic execution. In the sequel, we slightly adapt the framework described in [4] to our purpose. Behaviors of any sub-system sys' to be tested are obtained by projecting behaviors of a symbolic execution of the whole system. It remains to define test purposes dedicated to test such projected behaviors. As basic-IOSTS and hiding mechanism introduce τ-transitions, then such a projected

behavior $p_{sys'}$ may contain τ-transitions. Since such internal transitions cannot be observed during testing, we construct test purposes from a τ-reduced symbolic execution enriched by quiescence. We identify all its finite paths whose last transitions are output-transitions (including δ-transitions) and which result of the τ-reduction of a path whose $p_{sys'}$ is a prefix. Those τ-reduced finite paths become behaviors to be tested.

Definition 18. (Test purpose) *Let $SE(sys')$ be a symbolic execution of sys' such that $p_{sys'} \in FP(SE(sys'))$. Let us note $ext_o(p_{sys'})$ the set $\{p_{sys'}\}$ if $p_{sys'}$ is of the form $p.(\eta, act, \eta')$ with $act \in Output(Sig(sys', F))$ and whose elements are all paths of the form $p_{sys'}.(\eta_1, act_1, \eta'_1) \cdots (\eta_n, act_n, \eta'_n)$ with $act_i = \tau$ for $i < n$ and $act_n \in Output(Sig(sys', F))$ otherwise. Let us note $T \subseteq \mathcal{S}_{sat}(sys')$ the set of all the target states of all the finite paths of $ext_o(p_{sys'})$. A* symbolic test purpose *for $p_{sys'}$ and $SE(sys')$ is an application $TP : \mathcal{S}_{sat} \rightarrow \{skip, accept, \odot\}$ such that:*

- *for all $\eta \in T$, $TP(\eta) = accept$,*
- *for all finite path $st_1 \cdots st_n$ such that for all $i \leq n$, $st_i \in R_\tau$ and $TP(target(st_n)) = accept$, then $TP(source(st_i)) = skip$,*
- *If $ext_o(p_{sys'}) = \{p_{sys'}\}$ then all other states η verify $TP(\eta) = \odot$,*
- *if $ext_o(p_{sys'}) \neq \{p_{sys'}\}$ and the last transition of $p_{sys'}$ is an input-transition st then if there exists a transition $st_\delta \in \Delta R_\delta$ s. t. $source(st_\delta) = target(st)$ then $TP(target(st_\delta)) = accept$ and all other states $\eta \in R_\delta$ verify $TP(\eta) = \odot$,*
- *if $ext_o(p_{sys'}) \neq \{p_{sys'}\}$ and the last transition of $p_{sys'}$ is a τ-transition then all other states $\eta \in R_\delta$ verify $TP(\eta) = \odot$.*

Definition 18 introduces the notion of symbolic test purpose, which extends the notion of test purposes as defined in [4] by considering a symbolic execution of a system which incorporates constraints issued from a surrounding system. Let us remark that constraint symbolic executions allow us to characterize test purposes in the same way: a test purpose is a finite sub-tree of a δ-enriched symbolic execution whose leaves are target states of output transitions (identified by means of the labeling function which associates *accept* to those states). The algorithm of test case generation given in [4] can directly be applied.

An example of a slot machine: projection Let us consider p the left path of Figure 2, corresponding to the winning case. In Figure 3, the left path represents p. The right path is the projection p_{SM} of p on SM. Nearby each symbolic extended state name Ss_i we indicate in the grey box the content of the symbolic state, up to simplifications in path conditions and external constraints for sake of readability. The behavior denoted by p_{SM} corresponds intuitively to the following scenario: after the initialization, a bet is received for amount greater to 0 and less or equal to 100 (this is a constraint induced by the interface). Then SM sends a verdict stating that the player has won, the value to be removed of the bank account is received and correspond to 10 times the bet. The amount is sent to the interface and effectively removed from the bank account. Finally, SM sends an ending operation message to the Int. A test purpose L for this

behavior would label N_6 by *accept* and $N0$ to $N5$ by *skip*. On the right part of the figure, $N'2$ and $N'4$ are target states of the complementary transitions of respectively $(N\_1, int\_bet?bb\_0, N\_2)$ and $(N\_3, int\_amount?a\_0, N4)$. $N'2$ characterizes cases for which the received bet is out of the range allowed by the interface. N'_4 characterizes situation for which the gain does not correspond to 10 times the bet contrarily to the information sent by the interface.

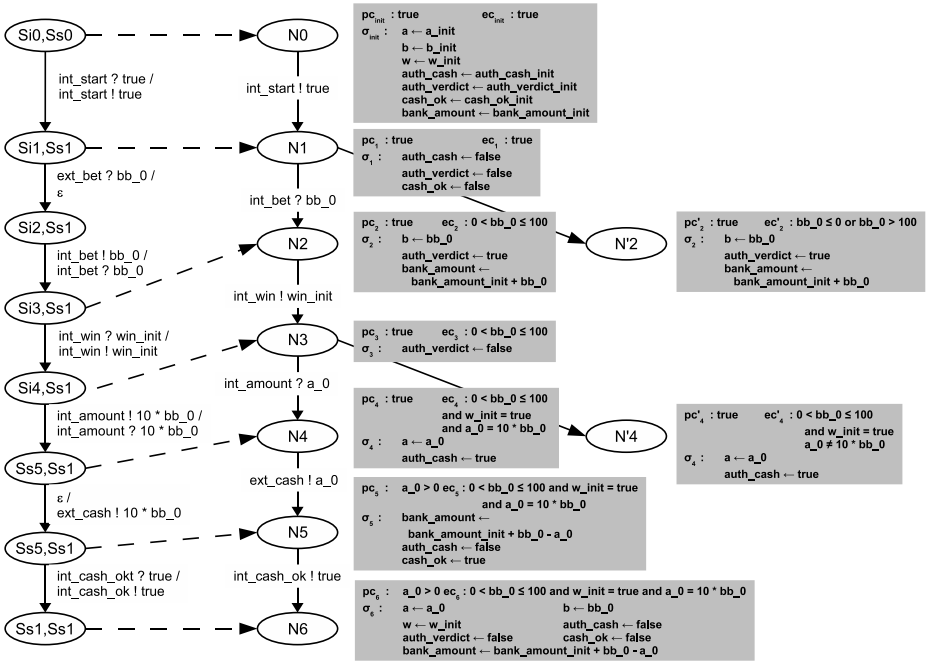

Fig. 3. Projection in the example of the slot machine

Those two situations are possible for SM but note relevant in the frame of the whole system. Therefore L would label $N'2$ and $N'4$ with $\odot$. To conclude, let us point out that such a test purpose cannot be deduced only from the knowledge of SM: it clearly depends on the way SM is used in the whole system S. This exemplifies our initial goal of eliciting from a system dedicated test purposes for each subsystem.

5 Conclusion and future works

We have extended the framework of IOSTS introduced in [4], in order to deal with component-based system specifications and we have used symbolic execution mechanisms in order to compute behaviors of sub-systems constrained by systems in which they are involved. Then, we have defined test purposes from those constrained behaviors. The definition of dedicated methodologies for component based systems should clearly rely on the targeted fault models. We plan to study fault models that mainly deal with communication mechanisms as in

[5]. For such fault models, a testing methodology would probably preconize to construct test purposes for behaviors involving a lot of internal communication synchronizations. Besides, we also plan to target fault models that mainly deal with basic components. As in [13], we could consider that composition and hiding mechanisms are well implemented such that an appropriate testing methodology would only consider test purposes directly defined at the component level. More generally, our next goal is to provide testing methodologies for component based systems which take advantage of the fact that some components or subsystems have been previously intensively tested such that a large class of tests becomes useless in the context of the whole system.

References

1. I. Berrada, R. Castanet, and P. Félix. Testing Communicating Systems : a Model, a Methodology, and a Tool. In *Proc. of the 17th Int. Conference TestCom 2005*, volume 3502 of *LNCS*, pages 111–128. Springer-Verlag, 2005.
2. C. Bigot, A. Faivre, J.-P. Gallois, A. Lapitre, D. Lugato, J.-Y. Pierron, and N. Rapin. Automatic test generation with AGATHA. In *Tool session of TACAS 2003*, pages 591–596, 2003.
3. L. Frantzen, J. Tretmans, and T.A.C. Willemse. A Symbolic Framework for Model-Based Testing. In *Proc. of the Int. Workshops FATES/RV 2006*, volume 4262 of *LNCS*, pages 40–54. Springer-Verlag, 2006.
4. C. Gaston, P. Le Gall, N. Rapin, and A. Touil. Symbolic Execution Techniques for Test Purpose Definition. In *Proc. of the 18th Int. Conference TestCom 2006*, volume 3964 of *LNCS*, pages 1–18. Springer-Verlag, 2006.
5. R. Gotzhein and F. Khendek. Compositional Testing of Communication Systems. In *Proc. of the 18th Int. Conference TestCom 2006*, volume 3964 of *LNCS*, pages 227–244. Springer-Verlag, 2006.
6. B. Jeannet, T. Jéron, V. Rusu, and E. Zinovieva. Symbolic test selection based on approximate analysis. In *Proc. of the 11th Int. Conference TACAS 2005*, volume 3440 of *LNCS*, pages 349–364. Springer-Verlag, 2005.
7. J.-C. King. A new approach to program testing. *Proc. of the Int. Conference on Reliable software*, 21-23:228–233, 1975.
8. D. Lugato, N. Rapin, and J.-P. Gallois. Verification and tests generation for SDL industrial specifications with the AGATHA toolset. In *Proc. of the Workshop on Real-Time Tools affiliated to CONCUR 2001*, 2001. ISSN 1404-3203.
9. P. Pelliccione, H. Muccini, A. Bucchiarone, and F. Facchini. TeStor: Deriving Test Sequences from Model-based Specification. In *Proc. of the 8th Int. Symp. CBSE 2005*, volume 3489 of *LNCS*, pages 267–282. Springer-Verlag, 2005.
10. N. Rapin, C. Gaston, A. Lapitre, and J.-P. Gallois. Behavioural unfolding of formal specifications based on communicating automata. In *Proc. of the 1th Int. Workshop ATVA 2003*, 2003.
11. J. Tretmans. Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation. *Computer Networks and ISDN Systems*, 29:49–79, 1996.
12. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3):103–120, 1996.
13. M. van der Bijl, A. Rensink, and J. Tretmans. Compositional Testing with IOCO. In *Proc. of the 3rd Int. Workshop FATES 2003*, volume 2931 of *LNCS*, pages 86–100. Springer-Verlag, 2003.