Test Purpose Concretization through Symbolic Action Refinement

Alain Faivre[1], Christophe Gaston[1], Pascale Le Gall[2], and Assia Touil[3,⋆]

[1] CEA LIST Saclay
F-91191 Gif sur Yvette Cedex
[2] Ecole Centrale Paris - Laboratoire MAS
F-92295 Chatenay Malabry
[3] Supelec - Computer Science Department
F-91192 Gif sur Yvette Cedex

Abstract. In a Model Driven Design process, model refinement methodologies allow one to denote system behaviors at several levels of abstraction. In the frame of a model-based testing process, benefits can be taken from such refinement processes by extracting test cases from the different intermediate models. As a consequence, test cases extracted from abstract models often have to be concretized in order to be executable on the System Under Test. In order to properly define a test concretization process, a notion of conformance relating SUTs and abstract models has to be defined. We define such a relation for models described in a symbolic manner as so-called IOSTSs (Input Output Symbolic Transition Systems) and for a particular kind of refinement, namely action refinement, which consists in replacing communication actions of abstract models with sets of sequences of more concrete communication actions. Our relation is defined as an extension of the *ioco*-conformance relation which relates SUTs and models whose communication actions are defined at the same level of abstraction. Finally we show from an example how a test purpose resulting from an abstract IOSTS-model can be concretized in a test purpose defined at the abstraction level of the SUT.

Keywords: Model Based Testing, Action Refinement, Symbolic Conformance Testing, Test Purpose, Test Purpose Concretisation.

1 Introduction

Model Driven Design approaches allow one to describe systems at several abstraction levels. In an usual software development top-down approach, requirements are expressed at a very high abstraction level in a given model and refined into several more concrete models which successively detail more an more the implementation choices: this process is also called the *refinement process*. When dealing with reactive systems, automata based languages including input and

⋆ This work was partially supported by the RNTL French project EDEN2 and by the Action Marie Curie TAROT.

K. Suzuki et al. (Eds.): TestCom/FATES 2008, LNCS 5047, pp. 184–199, 2008.

output mechanisms can be used as modeling languages: models denote behaviors in term of so-called *traces* which are sequences of inputs and outputs exchanged with the environment through interfaces. In the context of automata-based languages, such interfaces are often denoted by symbols representing communication channels. The ability to describe interfaces at different levels of abstraction is important in a refinement process: it allows one to denote behaviors while abstracting from implementation details concerning interfaces. Let us illustrate on a simple example of an ATM application: in an abstract model, expressing that a PIN-code is entered and compared to the actual PIN-code does not require to detail the concrete mechanisms to enter it: the corresponding interface can be denoted by an abstract channel PIN through which the number representing the PIN-code transits. At a more concrete level, one may specify that the abstract input sent through the channel PIN is in fact refined into a sequence of four inputs through a channel $DIGIT$ corresponding to the four digits composing the PIN-code. Traces from the abstract model are concretized into new ones, built over concrete interface elements which are directly compatible with the real system. Defining how an abstract interface is mapped to a concrete one and how abstract inputs and outputs are refined is a key point in a refinement process and is usually called *action refinement* [3].

The paper presents an approach to take benefits from action refinement in a model-based testing process. In a previous contribution [2], we have proposed a method to extract test cases from models given in the form of Input-Output Symbolic Transition Systems (IOSTS). IOSTSs are automata based models involving data and communication actions (input and output actions) denoted in a symbolic manner: those input (resp. output) actions denote sets of actual input (resp. output) values received (resp. emitted) through channels. Test cases are traces of an IOSTS-model, characterizing sequences of stimulations (*i.e.* input values) to be sent to the *System Under Test* (SUT) and of intended reactions (*i.e.* outputs values) of the SUT. In [2], we have built test cases from *test purposes* described as tree-like structures obtained by *symbolically executing* IOSTS-models. Defining a test purpose amounts to choosing a finite number of finite paths in the symbolic execution tree as behaviors to be tested. Symbolic execution has been first defined for programs ([4]) and mainly consists in replacing concrete input 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 follows the same intuition considering guards of transitions as conditions and assignment together with communication actions as instructions. Symbolic execution of an IOSTS results in a so-called *symbolic execution tree* in which each path characterizes a set of behaviors constituted by all traces obtained by solving the constraints associated to the path. A test case associated to the test purpose will interact with the SUT in order to make it perform at least one trace per chosen finite path. Moreover, such test purposes may be automatically defined [2]: this is useful to generate test purposes with no human intervention while ensuring a coverage of the IOSTS-model behaviors. Applying such coverage criteria on the different models of a refinement process strengthens even

more the coverage of all specified behaviors. However, when dealing with test purposes extracted from abstract models, due to the action refinement steps, it is not possible to directly define test cases interacting with the SUT: they have to be concretized at the description level of the SUT interface.

In this paper we extend the symbolic model-based testing framework defined in [2] to deal with action refinement, following the approach proposed in [9]. We adapt the notion of *refinement pairs* as defined in [9] in a non-symbolic context. Intuitively, refinement pairs associate abstract communication actions to some concrete communication action sequences. Concrete IOSTS can be derived from an abstract one by replacing abstract communication actions by their associated concrete communication action sequences. As in [9] we focus on input action refinement, but since in our framework, a symbolic input action is an abstraction of a set of input values, a symbolic refinement pair denotes a (possibly infinite) set of refinement pairs as defined in [9]. Moreover, in [9], the authors consider linear atomic input-inputs refinement: an abstract input is refined as one sequence of concrete inputs (regardless of particular intermediate action sequences denoting the quiescence of the SUT). We also consider atomic input-inputs refinement, but we do not require linearity: an abstract symbolic input can be refined by a set of possible input sequences. This is useful to define complex refinements in which an abstract input is refined in a non deterministic manner (*e.g.* an amount of money to be sent to an ATM may be concretely entered in different ways depending on the chosen coins). As in [9], conformance between an abstract model and a SUT is defined by means of a dedicated conformance relation derived from the ioco-conformance relation [7]. The usual ioco-conformance is adapted to cope with sets of symbolic input action refinements. We then establish a result which can be seen as an extension of the one given in [9], stating that *ioco*-conformance to a concrete model is equivalent to the extended notion of ioco-conformance to an abstract model provided that concrete and abstract considered models are related by the refinement pair set used to define the extended ioco-conformance relation. Finally, we illustrate with the help of an example how an abstract test purpose can be concretized through refinement pairs.

Paper organization. In Section 2, we introduce IOSTS and define symbolic action refinement pairs. In Section 3, we extend symbolic model based testing to take into account refinement of actions. Section 4 contains our example of test purpose concretization.

2 Symbolic Action Refinement for IOSTS

2.1 Input/Output Symbolic Transition Systems(IOSTS)

We assume the reader familiar with basic notions of many sorted first-order equational logic [5]. We recall notations about IOSTS as given in [2], [6] and [1]. Let us first introduce the data part specified with a many sorted first-order equational logic. A *data signature* is a couple $\Omega = (S, Op)$ where S is a set of types, Op is a set of operations, each one being provided with a profile $s_1 \cdots s_{n-1} \rightarrow s_n$ (for $i \leq n$,

$s_i \in S$). The set $T_\Omega(V) = \bigcup_{s \in S} T_\Omega(V)_s$ of *terms* with typed variables in $V = \bigcup_{s \in S} V_s$ is inductively defined as usual over Op and V. For any term t in $T_\Omega(V)$, $Var(t)$ denotes the set of all variables of V occurring in t. A *variable renaming* is any injective mapping $\mu_V : V \to V'$ and can be canonically extended to the set of terms $T_\Omega(V)$. A *substitution* is a function $\sigma : V \to T_\Omega(V)$ preserving types which can also be canonically extended to $T_\Omega(V)$. $T_\Omega(V)^V$ denotes the set of all substitutions defined on V. The definition domain $Dom(\sigma)$ of a substitution σ is the set $\{x | x \in V, x \neq \sigma(x)\}$. The set $Sen_\Omega(V)$ of all typed equational *formulas* contains the truth values *true* and *false* and all formulas built using the equality predicates $t = t'$ for $t, t' \in T_\Omega(V)_s$, and the usual connectives $\neg, \vee$ and $\wedge$.

A *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 $f_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$. *Interpretations* are applications ν from V to M preserving types, extended to terms in $T_\Omega(V)$. A model M satisfies a formula φ, denoted by $M \models \varphi$, iff, for all interpretations ν, $M \models_\nu \varphi$, where $M \models_\nu t = t'$ is defined by $\nu(t) = \nu(t')$, and where the truth values and the connectives are handled as usual. M^V is the set of all interpretations from V to M. In the sequel, we only use integers, booleans, enumerated types and character strings as data types. Thus, data types are interpreted in a fixed model denoted M and defined for a given data signature $\Omega = (S, Op)$ dedicated to specify those data types.

IOSTS-signatures are couples (A, C) where $A = \bigcup_{s \in S} A_s$ is a set of *attribute variables* and where C is a set of *communication channels*. Sig is the set of all IOSTS-signatures. For two signatures $\Sigma_1 = (A_1, C_1)$ and $\Sigma_2 = (A_2, C_2)$, usual set operators can be extended: $\Sigma_1 \subseteq \Sigma_2$, iff $C_1 \subseteq C_2$ and $A_1 \subseteq A_2$; $\Sigma_1 \cup \Sigma_2 = (A_1 \cup A_2, C_1 \cup C_2)$; $\Sigma_1 \cap \Sigma_2 = (A_1 \cap A_2, C_1 \cap C_2)$; finally $\Sigma_1 \backslash \Sigma_2 = (A_1 \backslash A_2, C_1 \backslash C_2)$.

The *set* $Act(\Sigma)$ *of communication actions* over an IOSTS-signature Σ contains the unobservable action τ, the set $Input(\Sigma) = \{c?y \mid c \in C, y \in A\}$ whose elements are called *receptions* or *input actions* and the set $Output(\Sigma) = \{c!t \mid c \in C, t \in T_\Sigma(A)\}$ whose elements are called *emissions* or *output actions*.

Definition 1 (IOSTS). *An* IOSTS *over a signature* $\Sigma = (A, C)$ *is a tuple* $G = (Q, q_0, Trans)$ *where* Q *is a set of* states, $q_0 \in Q$ *is the* initial state *and* $Trans \subseteq Q \times Act(\Sigma) \times Sen_\Omega(A) \times T_\Omega(A)^A \times Q$ *is a set of* transitions.

A transition $tr = (q, act, \varphi, \rho, q')$ *of* $Trans$ *is composed of a source state* $source(tr) = q$, *an action* $act(tr) = act$, *a guard* $guard(tr) = \varphi$, *a substitution of variables* $subs(tr) = \rho$ *and a target state* $target(tr) = q'$.

$STS(\Sigma)$ *denotes the set of all IOSTS over the signature* Σ.

For a transition tr with $act(tr) = c?x$, $rec(tr)$ denotes the variable x and for an IOSTS $G = (Q, q_0, Trans)$, $\mathcal{T}_{c?}(G)$ is the subset of $Trans$ of all transitions tr such that $act(tr)$ is of the form of $c?x$. We also denote $Att(G)$ for A, $init(G)$ for q_0, $Trans(G)$ for $Trans$, $Interface(G)$ for C and $Sig(G)$ for Σ.

Example 1. Fig. 1 depicts a model of a very simple drink vending machine, called Abstract Vending Machine, or AVM for short. This machine allows the user to

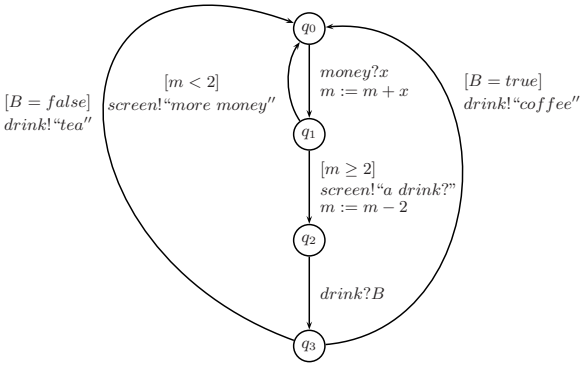

Fig. 1. AVM: an example of IOSTS

order a coffee or a tea. First, the user introduces some money. If the amount is greater than or equal to the cost of drinks (here 2) the user can choose a tea or a coffee, else he has to introduce more money. At last, the machine serves the user the asked drink.

The set $Obs(\Sigma)$ of *observations over* Σ is $(C \times \{?, !\} \times M)$. An observation of the form $c!m$ (resp. $c?m$) is called an output (resp. input) value. The set $Run(tr) \subseteq M^A \times (Obs(\Sigma) \cup \{\tau\}) \times M^A$ of *runs* of $tr = (q, act, \varphi, \rho, q') \in Trans$ is s.t. $(\nu^i, act_M, \nu^f) \in Run(tr)$ iff: (1) if act is of the form $c!t$ (resp. τ) then $M \models_{\nu^i} \varphi$, $\nu^f = \nu^i \circ \rho$ and $act_M = c!\nu^i(t)$ (resp. $act_M = \tau$), (2) if act is of the form $c?y$ then $M \models_{\nu^i} \varphi$, there exists ν^a such that $\nu^a(z) = \nu^i(z)$ for all $z \neq y$, $\nu^f = \nu^a \circ \rho$ and $act_M = c?\nu^a(y)$.

For a run $r = (\nu^i, act_M, \nu^f)$, we denote $source(r)$, $act(r)$ and $target(r)$ respectively ν^i, act_M and ν^f.

As in [7,8], we will use $\delta!$ to denote *quiescence*: quiescence refers to situations for which it is not possible to execute an output action.

For an IOSTS G, the set of its finite paths, denoted $FP(G)$ contains all finite sequences $p = tr_1 \dots tr_n$ 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 $i \leq n$, $r_i \in Run(tr_i)$ and for all $i < n$, $target(r_i) = source(r_{i+1})$. Following the approach of [8] and with the notation $Tr(r) = act(r_1) \dots act(r_n)$ for $r \in Run(p)$, the set $STr(p, r)$ of *suspension traces of a run* r *of* p 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 $\{m.act(r_{tr}) | m \in STr(p', r')\} \subseteq STr(p, r)$ with the convention that τ is the neutral element for action concatenation.
- 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[1] $\delta_m \in \{\delta!\}^*$, $Tr(r).\delta_m \in STr(p, r)$.

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

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 $STr(G) = \bigcup_{p \in FP(G)} STr(p)$.

We note $STr(\Sigma) = \bigcup_{G \in STS(\Sigma)} STr(G)$ and $STr = \bigcup_{\Sigma \in Sig} STr(\Sigma)$. Moreover, for any observation trace $st \in STr$, $Interface(st)$ is the set of channel names occurring in at least one observation of st.

For a finite path p, we define the set $Def(p)$ of its defined variables. It contains all attribute variables which are defined along p. Intuitively, a variable z is defined either if there is a reception on the variable z for one of the transitions occuring in p or if there exists a variable substitution ρ associated to a transition of p such that $z \in Dom(\rho)$ and all variables in $\rho(z)$ are already defined in the sub-path preceding the considered transition. On the contrary, when a variable z is assigned by a transition substitution to a term containing undefined variables, then z becomes undefined and should be removed from the set of defined variables. More formally, the set of defined variables for a path may be characterized inductively as follows:

Definition 2 (Set of defined variables). *Let $G = (Q, q_0, Trans)$ be an IOSTS over Σ. Let p be a finite path of $FP(G)$. The* set $Def(p)$ of defined variables *of p is defined as follows:*

(1) if p is of the form ε, $Def(p) = \emptyset$
(2) if p is of the form $p'.tr$ with $tr = (q, act, \varphi, \rho, q')$

$$Def(p) = (Def^{tr}(p') \cup \{z | z \in Dom(\rho) \wedge Var(\rho(z)) \subseteq Def^{tr}(p')\}) \setminus \{z | z \in Dom(\rho) \wedge Var(\rho(z)) \not\subseteq Def^{tr}(p')\}$$

$$\text{with } Def^{tr}(p') = \begin{cases} Def(p') \cup \{x\} \text{ if } act \text{ is of the form } c?x \\ Def(p') \text{ otherwise} \end{cases}$$

Finally, a *System Under Test* SUT is defined by a set $STr(SUT) \subseteq STr$. We note $Interface(SUT) = \bigcup_{st \in STr(SUT)} Interface(st)$. $STr(SUT)$ is required to be stable by prefix[2], and to be *input-complete*, that is: $\forall st \in STr(SUT), \forall c \in Interface(SUT), \forall m \in M, st.c?m \in STr(SUT)$.

2.2 Symbolic Action Refinement for IOSTS

An action refinement indicates which concrete action sequences implement an abstract action and possibly concretize the data handled in the abstract action by decomposing it into several concrete data. For instance, one can refine the abstract input action $money?x$ as many successive concrete input actions $coin?y_i$ as necessary to cope with the required money amount. Intuitively the abstract variable x is related to the concrete variables y_i by the equality $x = y_1 + \ldots + y_n$ with n denoting the number of successive input actions $coin?y_i$. In the sequel, we focus on the refinement of symbolic input actions as in [9]. In fact, the case of output actions is simpler than the one of input actions[3] and it can be treated in

[2] $STr(SUT)$ is stable by prefix if any prefix of traces in $STr(SUT)$ belongs to $STr(SUT)$.

[3] Indeed, contrarily to input actions, output actions do not impact values assigned to attribute variables.

a similar way. We simply do not treat it because we want to limit the complexity of definitions.

We characterize a refinement pair R as a couple associating a channel name c which defines the class of abstract input actions to be refined to an input refinement describing all the associated intended behaviors: an input refinement is a tuple composed of an IOSTS $G = (Q, q_0, Trans)$, an exit (or final) state s belonging to Q and a variable χ. Intuitively, an input action $c?x$ can be refined in any of the finite paths starting from the entry state q_0 and ending at the exit state s. We now discuss about the usefulness of χ. As we are in a symbolic framework, the input action to be refined can be any input action through the channel c on any attribute variable of the abstract IOSTS. For generality sake, our refinement mechanism is independent of the attribute variable introduced in the input action to be refined. The variable χ allows us to link the symbolic input action and the refining IOSTS G by applying some renaming mechanisms on G substituting χ by the targeted abstract attribute variable (this is done in Definition 4).

Definition 3 (Input Refinement/Refinement pair). *An* input refinement *is a triple* (G, s, χ) *with* $G = (Q, q_0, Trans)$ *an IOSTS,* $s \in Q$ *with* $s \neq q_0$ *called the* final state *and denoted by* $final(G)$*, and* χ *a variable satisfying* $\chi \notin Att(G)$ *such that:*

- $\forall p \in FP(G)$*, there exists* p' *with* $p.p' \in FP(G)$ *and* $target(p.p') = s$*.*
- $\forall tr \in Trans$*,* $act(tr) \notin Output(\Sigma)$*.*
- $\forall p \in FP(G), target(p) = s \Rightarrow \chi \in Def(p)$*.*

For any $c \in C$*, the couple* $(c, (G, s, \chi))$ *is called* a refinement pair*.*

We note $RP(\Sigma)$ *the set of all refinement pairs* $(c, (G, s, \chi))$ *with* $Sig(G) = \Sigma$*.*

By abuse, G will be called the input refinement or the refining IOSTS.

Intuitively, a refinement pair $(c, (G, s, \chi))$ denotes the capacity of refining any action of the form $c?x$ by all the paths of G from q_0 to s, provided that within the IOSTS G, the variable χ has been first substituted by x.

Let us point out that actions used in refining IOSTS are required to be either input actions or internal actions. This restriction has already been made in [9] and corresponds to a simplification motivated by testing issues: imposing that restriction ensures the ability to define the variable χ only in relation to the refining input actions. If refinement of input actions would authorize output actions, the SUT might have different ways to answer and this would make the testing process more difficult. By requiring that refinement actions of abstract input actions are only made of input actions, we can compute *a priori* a sequence of refining input actions matching the targeted abstract one. Such a restriction, also known as input-inputs refinement, is thus of particular interest for testing since the level of controllability is maintained by action refinement.

Example 2. In Fig. 2, we show an IOSTS, denoted G in the sequel, which is used to build an input refinement $R = (G, s, \chi)$. A refinement pair $(money, (G, s, \chi))$

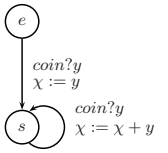

Fig. 2. Refining IOSTS G

is then constituted of an abstract channel *money* and the input refinement $R = (G, s, \chi)$ built over G. In order to put money in the drink vending machine (the AVM of Example 1), we have to put coins whose sum corresponds to the total amount sent on the channel *money*. The refinement consists in decomposing that amount into repetitive receptions of coins whose accumulated sum corresponds to the abstract amount. For example, if the price of a drink is 2, the customer can put either a coin of 2 or two coins of 1. Let us remark that all paths in G clearly define the variable χ, whose value exactly corresponds to the sum of introduced coins. Indeed, let us point out that for any path p of G starting from e, $Def(p) = \{\chi, y\}$.

We now define the function refining an abstract model G_a into a concrete model w.r.t. a refinement pair $(c, (G, s, \chi))$. Intuitively, all input actions of the form $c?x$ of G_a are replaced by the IOSTS G in which χ is renamed by x. All possible paths of G starting at q_0 and ending at s concretize $c?x$.

Definition 4 (Refinement Function). *Let $\Sigma = (A, C)$ and $\Sigma_R = (A_R, C_R)$ be two IOSTS-signatures, such that $A \cap A_R = \emptyset$.*

We define the refinement function

$$\begin{array}{lll} ref : RP(\Sigma_R) & \times\ STS(\Sigma) & \rightarrow STS(\Sigma \cup \Sigma_R) \\ \quad R = (c, ((Q, q_0, Trans), s, \chi)) & \quad G_a{=}(Q_a, q_{a0}, Trans_a) & \mapsto (Q', q_{a0}, Trans') \end{array}$$

and the family of attribute renaming $\mu_{A,rec(tr)}$ indexed by the variable $rec(tr)$ such that $\mu_{A,rec(tr)}(\chi) = rec(tr)$ and for all variables y of Σ_R, verifying $y \neq \chi$, $\mu_{A,rec(tr)}(y) = y$.

$ref(R, G_a) = (Q', q_{a0}, Trans')$ is the IOSTS s.t.[4]*:*

$$Q' = Q_a \cup \{(q, tr) \mid q \in Q, tr \in \mathcal{T}_{c?}(G_a)\}$$

$$\begin{array}{rl} Trans' = & (Trans_a \setminus \mathcal{T}_{c?}(G_a)) \\ \bigcup_{tr \in \mathcal{T}_{c?}(G_a)} & (\{((q, tr), \mu_{A,rec(tr)}(act), \mu_{A,rec(tr)}(\varphi), \mu_{A,rec(tr)}(\rho), (q', tr)) \\ & \quad \mid (q, act, \varphi, \rho, q') \in Trans\} \\ & \cup \{(source(tr), \tau, guard(tr), id_A, (q_0, tr)), \\ & \quad ((s, tr), \tau, true, subs(tr), target(tr))\}) \end{array}$$

$ref(R, G_a)$ is the concretization *of G_a w.r.t. the refinement pair R.*

[4] id_A is the identity function on A and $\mu_{A,rec(tr)}$ is extended to communication channels and formulae in a canonical way.

The refinement function has the following two interesting properties. (1) Since the refining IOSTS are such that the variable under refinement, denoted χ in Definition 3, is defined, the refinement of a symbolic input action $c?x$ ensures that in $ref(R, G_a)$, the attribute variable x receives a value controlled by the input refinement (this value is the result of a function only depending on the concrete sequence of input values). Such variables are sometimes referred as *controllable*. (2) The refinement of a symbolic input action can restrict the set of reachable values for the targeted attribute variable x since concrete input actions can be specialized as much as wished according to the designer choices.

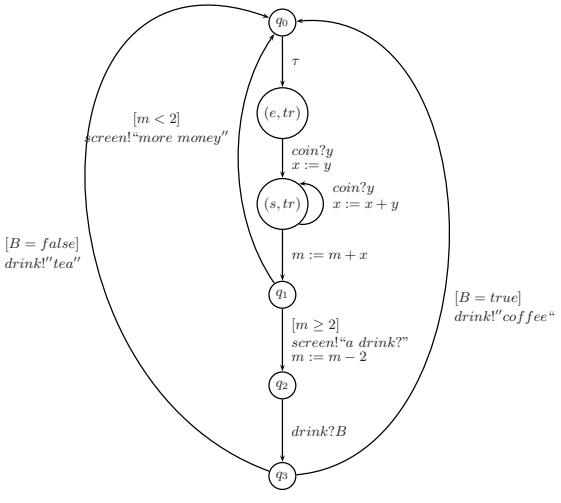

Fig. 3. CVM IOSTS

Example 3. We refine the AVM of Example 1 using the refinement pair ($money$, (G, s, χ)) of Example 2. The transition tr of source q_0 and target q_1, carrying the abstract action $money?x$ is replaced by concrete behaviors. For this purpose, we rename each state st of G by (st, tr). Here we obtain two new states (e, tr) and (s, tr). The variable χ is replaced by the variable x that corresponds to the reception of the amount in the abstract specification. The refinement function connects then the states (e, tr) and (s, tr) respectively to q_0 and q_1 with τ transitions and adds the substitution $m := m + x$ on an exit transition. Finally, we obtain a concrete model *Concrete Vending Machine*, or CVM for short, that specifies how to introduce coins to order some drink.

We may use several refinement pairs to refine a given abstract model. It suffices to iteratively apply the refinement function ref with the considered refinement pairs. In order to ensure that refining an abstract IOSTS G_a according to a refinement pair RP_1 defined for a channel $c1$ then refining according to another refinement pair RP_2 for a channel $c2$ (with $c1 \neq c2$) leads to the same model than refining G_a according to RP_2 and then to RP_1, it suffices to require that for

any considered refinement pair $(c, (G, s, \chi))$, c is an abstract channel of G_a and the refining IOSTS G does not share a channel with the abstract specification G_a under consideration. Thus, the channels occurring in the refining IOSTS which appear in the intermediate refined specifications from G_a cannot be refined later in the next refinement steps. Under this condition, the order of refinement steps does not matter. Indeed, let us recall that a refinement pair ensures that the abstract reception variable occurring in the abstract action to be refined is defined within the input refinement (up to the variable renaming). This means that for each path of the input refinement, the resulting value associated to the abstract reception variable is uniquely defined in function of the concrete input actions. In other words, the value of the abstract reception variable cannot depend either on the attribute variables of the abstract IOSTS to be refined or on the attribute variables of the input refinements. So, it does not depend on the context of use of the refinement pairs. Let us also remark that different input refinements may share some concrete channels or attributes.

In the sequel, for any abstract specification G_a and for any family of refinement pairs $\mathcal{R}$ verifying the above hypotheses (not the same abstract channel for two refinement pairs, no abstract channel in refining IOSTS), then we note $ref(\mathcal{R}, G_a)$ the resulting specification obtained by applying the refinement function on G_a for all refinement pairs in $\mathcal{R}$. We note $\Sigma_{\mathcal{R}}$ (resp. $C_{\mathcal{R}}$) the union of all signatures Σ_R (interfaces $Interface(G_R)$) for the refinement pairs $(c_R, (G_R, s_R, \chi_R))$ belonging to $\mathcal{R}$ with $Sig(G_R) = \Sigma_R$.

3 IOCO Conformance up to Refinement

Testing a system w.r.t. a specification requires the definition of a conformance relation. Our approach is based on the *ioco*-conformance relation [7].

Definition 5 (*ioco*). *Let G be an IOSTS and SUT be a system under test*[5] *such that $Interface(G) = Interface(SUT)$. SUT is* ioco-conform *to G, iff for any $str \in STr(G) \cap STr(SUT)$, if there exists act of the form $c!v$ or $\delta!$ such that $str.act \in STr(SUT)$, then $str.act \in STr(G)$. In such a case, we also say that str is* ioco-conform *to G.*

We extend Definition 5 in order to reason about conformance of a SUT to an IOSTS-model G_a up to a set of refinement pairs. This extension requires to define abstractions of traces of the SUT. Those abstractions are traces only involving channels of $Interface(G_a)$. For a refinement pair RP and a trace str, we define the set of inputs that can be concretized in the form of str through RP.

Definition 6. *Let $RP = (c, (G, s, \chi))$ be a refinement pair. Let str be a suspension trace of STr. Let p be a finite path of G with $target(p) = s$ such that $str \in STr(p)$. Such a path p is called a* complete path of *str.*

[5] Let us recall that by hypothesis, a system under test is such that its set of traces is stable by prefix and input-complete.

We note $Run(str, p) \subseteq \{r \mid r \in Run(p), str \in STr(p, r)\}$*, and* $CP(str)$ *the set of all complete paths of* str*.*

For any run $r = r_1 \cdots r_n$ *of* $Run(str, p)$*,* $target(r_n)(\chi)$ *is called the* value assigned by str to χ through p and r*. One notes* $(str, p)(\chi)$ *the set of all values assigned by* str *to* χ *through* p *and* r *for any* $r \in Run(str, p)$*.*

The set $Abs(str, RP)$ of abstractions of str associated to RP *is the set:*

$$\{c?v \mid \exists p \in CP(str), v \in (str, p)(\chi)\}$$

Note that $Abs(str, RP)$ is necessarily empty if str is not a trace only made up of observations which are either inputs whose associated channels are in $Interface(G)$ or $\delta!$.

A refinement process generally involves a family of refinement pairs. Therefore we generalize Definition 6 to define the set of inputs that can be concretized in the form of str through at least one refinement pair of the family.

Definition 7. *Let* $\mathcal{R}$ *be a family of refinement pairs and* $str \in STr$*. The* set of abstractions of str associated to $\mathcal{R}$ *is the set* $Abs(str, \mathcal{R}) = \bigcup_{RP \in \mathcal{R}} Abs(str, RP)$*.*

Note that we do not require that several IOSTSs defined in different refinement pairs of $\mathcal{R}$ do not share channels (*i.e.* have disjoint interfaces). Therefore, there may exist $RP_1 \neq RP_2 \in \mathcal{R}$ s.t. $Abs(str, RP_1) \neq \emptyset$ and $Abs(str, RP_2) \neq \emptyset$: a concrete trace may be abstracted through several refinement pairs.

Suspension traces of $STr(SUT)$ can only be composed of: inputs and outputs defined over channels of $Interface(G_a)$, the action $\delta!$, and inputs defined over channels introduced by refinement pairs of $\mathcal{R}$. Thus we generalize Definition 7 to define abstractions of any trace of $STr(SUT)$.

Definition 8. *With notations of Definition 7, the* set of abstract suspension traces of str associated to $\mathcal{R}$ *is the set* $AbsC(str, \mathcal{R})$ *defined as follows:*

- *if* str *is of the form* ε *then*

$$AbsC(str, \mathcal{R}) = \{\varepsilon\}$$

- *if* str *is of the form* $a.str'$ *where* a *is an observation s.t.* $a \notin Obs(\Sigma_{\mathcal{R}})$ *then*

$$AbsC(str, \mathcal{R}) = \{a.str'' \mid str'' \in AbsC(str', \mathcal{R})\}$$

- *if* str *is of the form* $str_r.str'$ *where* $str_r \in STr(\Sigma_{\mathcal{R}})$*,* $str_r \neq \emptyset$*, and* str' *is the empty trace or a trace beginning by an observation not in* $Obs(\Sigma_{\mathcal{R}})$*, let us consider* $Dec(str_r)$ *the set of all decompositions* (pr, sf) *of* str_r *(i.e* $str_r = pr.sf$ *and* $pr \neq \varepsilon$*), then* $AbsC(str, \mathcal{R})$ *is the set*[6]*:*

$$\bigcup_{(pr,sf) \in Dec(str_r)} Abs(pr, \mathcal{R}).AbsC(sf.str', \mathcal{R})$$

[6] For two sets E and F, $E.F = \{e.f \mid e \in E, f \in F\}$. If $E = \emptyset$ or $F = \emptyset$ then $E.F = \emptyset$.

Roughly speaking, $AbsC(str, \mathcal{R})$ denotes the set of all suspension traces composed of abstract observations that are abstractions of str w.r.t. $\mathcal{R}$. Let us observe that if the trace str cannot be correctly abstracted then its associated set of abstract suspension traces is empty.

Let us point out that for any G_c obtained by applying iteratively Definition 4 for all refinement pairs of $\mathcal{R}$ on an IOSTS G_a, the two following lemmas hold:

Lemma 1. *With $G_c = ref(\mathcal{R}, G_a)$, for all $str_a \in STr(G_a)$ s.t. there exists str_c with $str_a \in AbsC(str_c, \mathcal{R})$ then $str_c \in STr(G_c)$.*

Intuitively, Lemma 1 holds because by construction, Definition 4 ensures that for all $str_a \in STr(G_a)$, $STr(G_c)$ contains all suspension traces str_c such that $str_a \in AbsC(str_c, \mathcal{R})$.

Lemma 2. *With $G_c = ref(\mathcal{R}, G_a)$, for any $str_c \in STr(G_c)$ such that $AbsC(str_c, \mathcal{R}) \neq \emptyset$, there exists $str_a \in STr(G_a)$ s.t. $str_a \in AbsC(str_c, \mathcal{R})$.*

By Definition 4 any suspension trace str_c of G_c such that $AbsC(str_c, \mathcal{R}) \neq \emptyset$ is either also a suspension trace of G_a, and in that case $str_c \in AbsC(str_c, \mathcal{R})$, or a suspension trace of a path p_c of G_c which can be built from a path p_a of G_a by replacing in p_a transitions involving input actions to be refined by a complete path (of some suspension trace) of the IOSTS defined in the corresponding refinement pairs RP. For each such complete path p appearing in the definition of p_c and associated suspension trace σ, one can build a suspension trace str_a of p_a by replacing σ by any input action $c?v$ where c is the refined channel of RP and v belongs to $(\sigma, p)(\chi)$. This ensures that $str_a \in AbsC(str_c, \mathcal{R})$. Lemmas 1 and 2 are useful to prove Theorem 1.

We now define conformance up to a family of refinement pairs.

Definition 9 (Conformance up to refinement). *Let SUT be a system under test s. t. $Interface(SUT) \subseteq Interface(G_a) \cup C_{\mathcal{R}}$. SUT is $ioco_{\mathcal{R}}$-conform to G_a iff for any $str \in STr(SUT)$:*

- *if $AbsC(str, \mathcal{R}) \cap STr(G_a) \neq \emptyset$ then there exists $str_a \in AbsC(str, \mathcal{R}) \cap STr(G_a)$ s.t. if there exists act of the form $c!v$ or $\delta!$ with $str.act \in STr(SUT)$, then $str_a.act \in STr(G_a)$,*
- *if $AbsC(str, \mathcal{R}) = \emptyset$ and there exists $str.str' \in STr(SUT)$ s.t. $AbsC(str.str', \mathcal{R}) \cap STr(G_a) \neq \emptyset$ then for any suspension trace $str.str'' \in STr(SUT)$, str'' has no prefix of the form $\delta_m.act$ where $\delta_m \in \{\delta!\}^*$ and act is of the form $c!v$.*

The first item corresponds to situations in which str can be abstracted in the form of some suspension traces of G_a. In such a case at least one of them can be extended in G_a by any output that extends str in SUT. The second item corresponds intuitively to situations in which str ends by a sequence of concrete inputs which does not concretize an abstract one but it is possible to extend str in SUT by concrete inputs in order to form a trace that can be abstracted. In such a case, it is required that no output can occur until str is completed.

We now state the following theorem which relates *ioco*-conformance and $ioco_{\mathcal{R}}$-conformance.

Theorem 1. *SUT is ioco-conform to* $ref(\mathcal{R}, G_a)$ *iff SUT is* $ioco_{\mathcal{R}}$*-conform to* G_a.

4 Test Purpose Concretization

We illustrate by means of an example how to adapt our testing framework ([2], [1]) to take into account concretization through action refinement. In [2], test cases are extracted from so-called *test purposes* which are tree-like structures from the so-called *symbolic execution tree* of the IOSTS-model. The symbolic execution tree is composed of *symbolic extended states* and transitions between symbolic extended states. Paths in the symbolic execution tree denote executions of the model. A symbolic extended state is a triple (q, π, σ) structuring three pieces of information relatively to the path (*i.e.* execution) leading to it: (1) the reached state q of the model; (2) values assigned to attributes at this step of the execution (in the form of a substitution $\sigma : A \rightarrow T_\Omega(V_{froz})$ [7]); (3) constraints over symbolic terms assigned to attributes in the form of a formula π called the *path condition* [8]. Each transition st of the symbolic execution tree (*symbolic transition* for short) corresponds to the execution of a transition tr of the model whose source (*resp.* target) is the state introduced in the symbolic extended state at the source (*resp.* target) of the symbolic transition. Moreover, st is labeled by a symbolic action obtained by replacing terms occurring in $act(tr)$ by their symbolic values. Characterizing a test purpose simply consists in choosing behaviors to be tested (*i.e.* paths) in the symbolic execution tree by labeling their final symbolic extended states with the '*accept*' flag.

Example 4. Fig. 4 contains a test purpose TP_a extracted from the AVM of Example 1. Symbolic extended states labeled by $\odot$ are targets of paths which are outside of the behavior to be tested. The path to be tested denotes the following behavior:

pay at least 2 *units*[9]*; the AVM requires a choice for the drink to be served; ask for a coffee; receive the coffee.*

In [2], we have proposed an algorithm to extract test cases from such test purposes. The appliance of that algorithm requires that the IOSTS-model and the SUT share a common interface. If we consider as SUT an actual vending machine whose associated Interface is the one of the CVM of Example 2, the abstract test purpose TP_a cannot obviously be used directly: it has to be refined.

Intuitively, in order to define a test purpose at the good level of abstraction from TP_a, our goal is to characterize a test purpose in which paths to be tested concretize those characterized in TP_a. In TP_a there is only one path to be tested. We extract from TP_a the sequence of consecutive transitions of AVM of Fig. 1

[7] Values assigned to variables by σ are denoted by terms over frozen variables (chosen in a set of frozen variables V_{froz}) whose assignments result either from inputs or from execution of substitutions introduced in transitions of the IOSTS-model.

[8] Those constraints are deduced from guards of transitions executed to reach the symbolic extended state.

[9] Provided that m is initialized to 0.

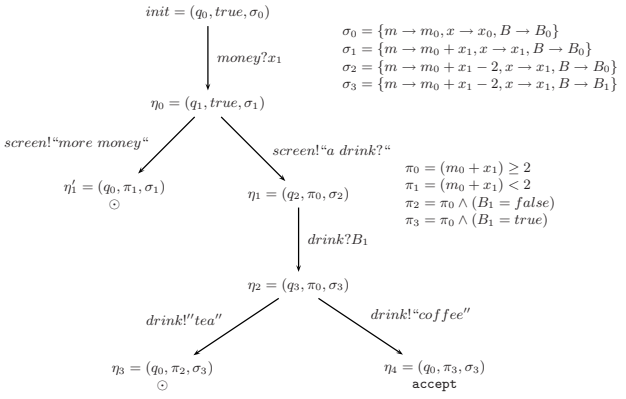

Fig. 4. Abstract test purpose TP_a

which have to be executed in order to reach the symbolic extended state labeled by *accept*. This sequence is a path p of AVM which relates states q_0, q_1, q_2, q_3, q_0 and whose transition from q_3 to q_0 is the one associated to the output action $drink!\text{“}coffee\text{”}$. In the following we need to differentiate several occurrences of a state or of a transition in p. To reach this purpose we construct a set of transitions T_p containing possibly several copies of transitions of AVM used to define p (one copy per occurrence). Practically, the source state and target state of a transition tr of p are named in a different manner than the one used in AVM of Fig. 1: this is done by using symbolic extended states as source and target states. For example to represent the occurrence of the transition $(q_0, money?x, true, [m := m+x], q_1)$ in p, the transition $(init, money?x, true, [m := m + x], \eta_0)$ belongs to T_p.

The following phase consists in defining an IOSTS whose associated set of transitions is T_p. We define the IOSTS $G_{TP_a} = (Q_p, q_0, T_p)$ where Q_p is the set of all source and target states of all transitions of T_p.

Example 5. The IOSTS G_{TP_a} associated to TP_a is depicted in Fig. 5.

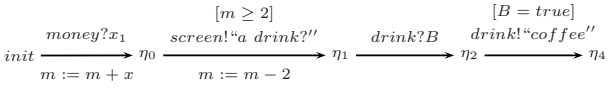

Fig. 5. G_{TP_a} for the test purpose of Fig. 4

A set of distinguished symbolic extended states are those labeled by *accept* in the test purpose. In Fig. 4 there is only one such state: η_4. We note $final(TP_a) = \{\eta_4\}$ this set. Now let us note RP the refinement pair described in Example 2 and used to define the $CVM = (AVM, \{RP\})$. We note G_{SUT} the IOSTS $(G_{TP_a}, \{RP\})$. G_{SUT} characterizes all finite paths that concretize p through refinement pairs of RPs. The interface of G_{SUT} is the one of the actual vending

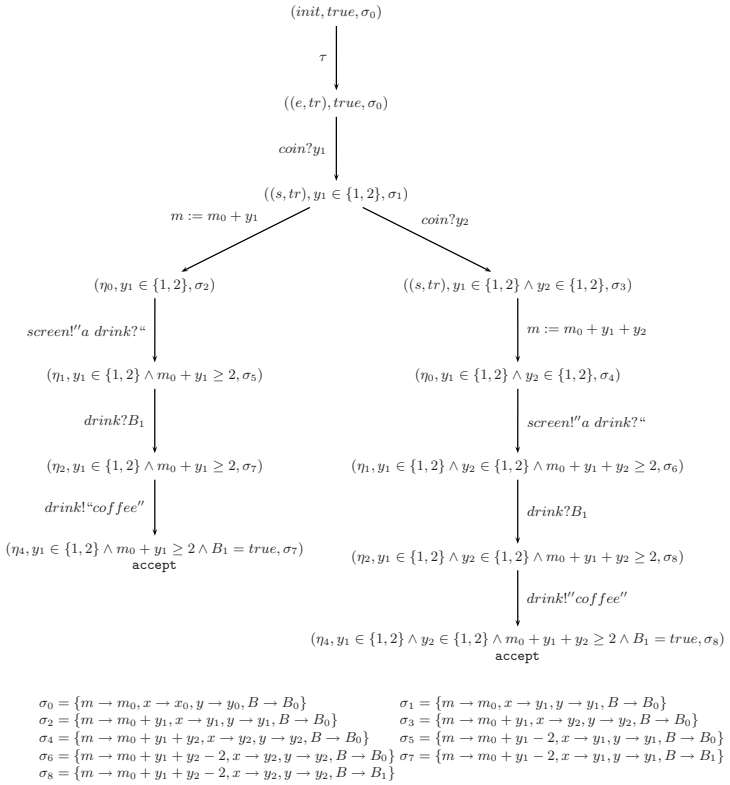

Fig. 6. Concrete test purpose

machine. Therefore it can be used to define test purposes for testing the actual vending machine by means of our algorithm. The set of concrete test purposes associated to TP_a contains all test purposes defined by symbolically executing G_c and labeling only symbolic extended states whose associated state is in $final(TP_a)$.

Example 6. In Fig. 6 we show a concrete test purpose associated to the abstract test purpose of Example 4. The variable y can take two values to denote the two possible coins that can be entered in the CVM: 1 for coins of 1 unit and 2 for coins of 2 units. It refines the behavior characterized in the test purpose of Fig. 4 by two concrete behaviors which correspond both to enter 2 units in the form of either one coin of two units (left) or two coins of one unit (right).

5 Conclusion

We have defined a model-based testing framework incorporating symbolic action refinement. Our framework is built as an extension of the symbolic conformance testing theory given in [2]. A refinement pair is made of a channel name c and

an IOSTS with a final state and a frozen variable χ to be substituted by the targeted attribute variable. This denotes the capacity of refining any reception on a variable x through c by any behavior of the refining IOSTS leading to the final state provided that the variable χ has been first substituted by the variable x. These concrete behaviors are composed of input actions on concrete channels or of internal actions and must define the variable χ. Under theses hypotheses, we have defined the concretization of an abstract IOSTS w.r.t. a family of refinement pairs. The classical *ioco*-relation underlying most conformance testing frameworks has been relaxed by associating to each concrete observable trace of an implementation all possible abstract observable traces which are compatible with the given family of refinement pairs. Thus, the conformance relation has been parameterized by the considered family of refinement pairs. We have then explained on an example how we can derive from test purposes extracted from the abstract specification some concrete test purposes sharing with the implementation the same interface and unfolding the abstract behaviors selected in the abstract test purpose w.r.t. refining IOSTS.

We are currently implementing this approach in the *AGATHA* tool developed at CEA LIST. This integration will be used in the frame of the RNTL French project EDEN2.

References

1. Faivre, A., Gaston, C., Le Gall, P.: Symbolic model based testing for component oriented systems. In: Petrenko, A., Veanes, M., Tretmans, J., Grieskamp, W. (eds.) TestCom/FATES 2007. LNCS, vol. 4581, pp. 90–106. Springer, Heidelberg (2007)
2. Gaston, C., Le Gall, P., Rapin, N., Touil, A.: Symbolic execution techniques for test purpose definition. In: Uyar, M.Ü., Duale, A.Y., Fecko, M.A. (eds.) TestCom 2006. LNCS, vol. 3964, Springer, Heidelberg (2006)
3. Gorrieri, R., Rensink, A.: Handbook of process algebra. In: Ch. Action refinement, pp. 1047–1147. Elsevier, Amsterdam (2001)
4. King, J.-C.: A new approach to program testing. In: Proceedings of the international conference on Reliable software, Los Angeles, California, vol. 21-23, pp. 228–233 (1975)
5. Loeckx, J., Ehrich, H.-D., Wolf, M.: Specification of abstract data types. John Wiley & Sons, Inc., New York (1997)
6. Rapin, N., Le Gall, P., Touil, A.: Symbolic execution techniques for refinement testing. In: Gurevich, Y., Meyer, B. (eds.) TAP 2007. LNCS, vol. 4454, pp. 131–148. Springer, Heidelberg (2007)
7. Tretmans, J.: Test generation with inputs, outputs, and quiescence. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 127–146. Springer, Heidelberg (1996)
8. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software - Concepts and Tools 17(3), 103–120 (1996)
9. van der Bijl, H.M., Rensink, A., Tretmans, G.J.: Action refinement in conformance testing. In: Khendek, F., Dssouli, R. (eds.) TestCom 2005. LNCS, vol. 3502, pp. 81–96. Springer, Heidelberg (2005)