SEMANTICS AND REFINEMENT OF BEHAVIOR STATE MACHINES

Kevin Lano, David Clark
Department of Computer Science, King's College London, Strand, London, UK
kevin.lano@kcl.ac.uk, david.j.clark@kcl.ac.uk

Keywords: UML; state machines; refinement; model transformations

Abstract: In this paper we present an axiomatic semantics for UML 2 behavior state machines, and give transformation rules for establishing refinements of behavior state machines, together with proofs of the semantic validity of these rules, based on a unified semantics of UML 2.

1 INTRODUCTION

The state machine notation of UML is widely used and supports dynamic modelling of applications. It is perhaps the most complex of the UML notations, and can be used to express most forms of UML activity diagrams (Chapter 12 of (OMG, 2007)), in addition to providing a semantic basis for verification of interaction diagrams.

In previous papers we have introduced an axiomatic semantics for UML class diagrams, OCL and flat state machines (Lano, 2008a), and extended this to structured protocol state machines (Lano and Clark, 2007) and sequence diagrams (Lano, 2007).

In this paper we complete the semantics of UML 2 state machines by considering structured behavior state machines, with communication between state machines. We apply the semantics to prove the validity of refinement transformations on behavior state machines.

In Section 2 we define the syntax of UML 2 state machines, Sections 3 and 4 define their semantics. Section 5 gives a definition of refinement. Section 6 gives several refinement transformations and proves these correct using the semantics.

2 UML 2 STATE MACHINES

Figure 1 shows the version of the UML 2 behavior metamodel which we consider here. State invariants will be allowed for both protocol and behavior state machines.

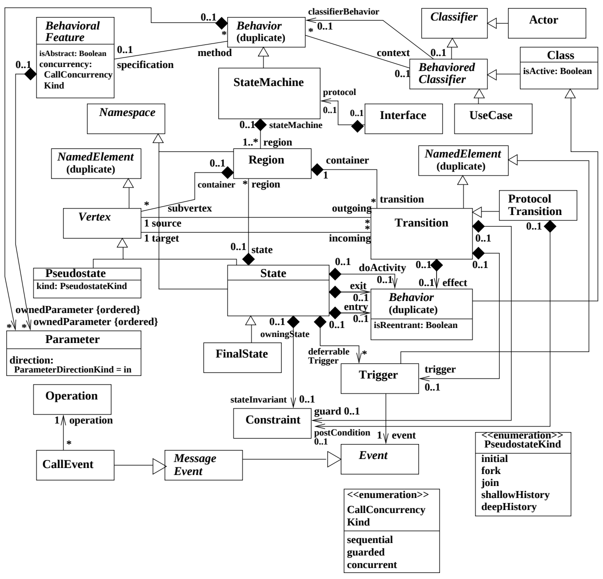

Figure 1: UML behaviour metamodel.

A *basic* state is a state with **region**.**size** $= 0$, other states are *composite* states. A composite state with one region is termed an OR state, and a composite state with more than one region is termed an AND state. Each OR state/region **s** contains a unique initial pseudo-state, and a unique transition from this to a normal state of the OR state/region, termed the *default initial state* of the OR state or region, and denoted $\mathbf{initial_s}$.

The notation $\mathbf{s} \sqsubseteq \mathbf{s}'$ means that $\mathbf{s} = \mathbf{s}'$ or **s** is a (recursive) substate of $\mathbf{s}'$.

3 SEMANTICS FOR UML STATE MACHINES

We give first the semantics for simple state machines as used in (Lano, 2008a), and then extend this to the full metamodel of Figure 1. The semantics of protocol and behavior state machines for a class **C** are incorporated into theories representing the semantics of **C**. This enables semantic checks of the consistency of the state machine models compared to the class diagram model.

The semantics is expressed in terms of temporal logic theories using the notation of Real-time Logic (RTL) and Real-time Action Logic (RAL) (Lano, 1998). The reason for using this general framework is that related notations of UML, such as interactions, require explicit treatment of the times of events.

Each UML class and model is represented as a temporal logic theory, which has semantic elements (attributes and actions) representing structural and behavioral features of the class or model, and axioms defining their properties. A generic instance of **C** is represented as a theory $I_{\mathbf{C}}$, the class itself by a theory $\Gamma_{\mathbf{C}}$, and models **M** by a theory $\Gamma_{\mathbf{M}}$ composed from the theories of the classes of **M**.

Refinement of model **M**1 by model **M**2 means that the theory $\Gamma_{\mathbf{M}2}$ proves each axiom of **M**1, under some interpretation of the elements of **M**1 in **M**2. This corresponds to state-based concepts of refinement, such as the weakening of preconditions and strengthening of postconditions (Morgan, 1990), and to concepts based on behavioural compatibility (Simons, 2005).

The following temporal logic notations are used to define the semantics:

1. The times $\leftarrow(\mathbf{op}(\mathbf{p}),\mathbf{i})$, $\rightarrow(\mathbf{op}(\mathbf{p}),\mathbf{i})$, $\uparrow(\mathbf{op}(\mathbf{p}),\mathbf{i})$, $\downarrow(\mathbf{op}(\mathbf{p}),\mathbf{i})$ of sending, request arrival, activation and termination of an operation execution $(\mathbf{op}(\mathbf{p}),\mathbf{i})$. These have values in a set TIME (normally $\mathbb{N}$) and are enumerated by the index $\mathbf{i} : \mathbb{N}_1$ in order of the reception times $\rightarrow(\mathbf{op}(\mathbf{p}),\mathbf{i})$.
2. Formulae $\mathbf{P} \circledcirc \mathbf{t}$, denoting that formula **P** holds at time **t**, and expressions $\mathbf{e} \circledast \mathbf{t}$ denoting the value of expression **e** at time **t**.

From these, other notations such as the RTL event-occurrence operators $\clubsuit(\psi := \mathbf{true}, \mathbf{i})$ "the **i**-th time that ψ becomes true", and $\#\mathbf{active}(\mathbf{op}(\mathbf{x}))$, the number of currently executing occurrences of $\mathbf{op}(\mathbf{x})$, can also be defined.

To define transition actions and other actions within a state machine, we use a generic procedural language with assignment, conditionals, loops, etc. Composite statements in this notation correspond to (structured) activities in UML 2.

The semantics of such statements is given by corresponding semantic actions $\mathbf{v} := \mathbf{e}$, $\alpha;\ \beta$, etc, in RAL. Each action has a *write frame*, which is the set of attributes it may change.

We can express that one action always calls another when it executes:

$$\alpha \supset \beta \ \equiv \ \forall \mathbf{i} : \mathbb{N}_1 \cdot \exists \mathbf{j} : \mathbb{N}_1 \cdot \uparrow(\alpha, \mathbf{i}) = \uparrow(\beta, \mathbf{j}) \ \wedge \ \downarrow(\alpha, \mathbf{i}) = \downarrow(\beta, \mathbf{j})$$

"α calls β". This is also used to express that α is defined by a (composite) action β.

Assignment $\mathbf{t}_1 := \mathbf{t}_2$ can be defined as the action $\alpha_{\mathbf{t}_1 := \mathbf{t}_2}$ where $\mathbf{t}_1$ is an attribute symbol, the write frame of this action is $\{\mathbf{t}_1\}$, and

$$\forall \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{t}_1 \circledast \downarrow(\alpha_{\mathbf{t}_1 := \mathbf{t}_2}, \mathbf{i}) = \mathbf{t}_2 \circledast \uparrow(\alpha_{\mathbf{t}_1 := \mathbf{t}_2}, \mathbf{i})$$

Similarly, sequential composition ; and parallel composition $||$ of actions can be expressed as derived combinators.

The ; and $||$ composite actions have write frames the union of the write frames of their component actions.

Occurrences of **if E then** $\mathbf{S}_1$ **else** $\mathbf{S}_2$ are either occurrences of $\mathbf{S}_1$ if **E** holds at commencement of this action, or occurrences of $\mathbf{S}_2$, if $\neg$ **E** holds. This action has write frame the union of those of $\mathbf{S}_1$ and $\mathbf{S}_2$.

Occurrences of **while E do S** are a sequence of occurrences $(\mathbf{S}, \mathbf{i}_1), \ldots, (\mathbf{S}, \mathbf{i_n})$ of **S**, where **E** holds at the commencement of each of these actions, and where **E** fails to hold at termination of $(\mathbf{S}, \mathbf{i_n})$. The **while** action has the same write frame as **S**.

Some important properties of $\supset$ are that it is transitive:

$$(\alpha \supset \beta) \wedge (\beta \supset \gamma) \ \Rightarrow \ (\alpha \supset \gamma)$$

and that statement constructs such as ; and **if then else** are monotonic with respect to it:

$$(\alpha_1 \supset \alpha_2) \wedge (\beta_1 \supset \beta_2) \ \Rightarrow \ (\alpha_1;\ \beta_1 \ \supset \ \alpha_2;\ \beta_2)$$

and

$$(\alpha_1 \supset \alpha_2) \wedge (\beta_1 \supset \beta_2) \ \Rightarrow \ \mathbf{if\ E\ then}\ \alpha_1\ \mathbf{else}\ \beta_1 \ \supset \ \mathbf{if\ E\ then}\ \alpha_2\ \mathbf{else}\ \beta_2$$

3.1 Unstructured Behavior State Machines

The semantics of a flat behavior state machine **SC** can be defined in the instance theory $\mathcal{I}_{\mathbf{C}}$ of its associated class, **C**, using composite actions (Lano, 2008a).

The transitions of such state machines have an action which executes when the transition is taken, instead of a postcondition. The transition actions **acts** are sequences

$$\mathbf{obj}_1.\mathbf{op}_1(\mathbf{e}_1);\ ...;\ \mathbf{obj_n}.\mathbf{op_n}(\mathbf{e_n})$$

of operation calls on supplier objects, sets of supplier objects, or on the **self** object. Such statements have a direct interpretation as composite actions $\mathbf{acts}'$ in RAL:

$$\mathbf{obj}'_1.\mathbf{op}_1(\mathbf{e}'_1);\ ...;\ \mathbf{obj}'_\mathbf{n}.\mathbf{op_n}(\mathbf{e}'_\mathbf{n})$$

where the $\mathbf{obj}'_\mathbf{i}$ and $\mathbf{e}'_\mathbf{j}$ are the interpretations of these expressions in RAL.

In addition to state invariants, there may be entry and exit actions of states, $\mathbf{entry_s}$, $\mathbf{exit_s}$, and do activities $\mathbf{do_s}$ of state **s**. Entry actions of a state should establish the state invariant, and do actions preserve the invariant.

The axiomatic representation of a flat behavior state machine is then:

1. The set of states is represented as a new enumerated type $\mathbf{State_{SC}}$.
2. A new attribute **c_state** of this type is added to $\mathcal{I}_{\mathbf{C}}$, together with the initialisation $\mathbf{c\_state} := \mathbf{initial_{SC}}$ of this attribute to the initial state of **SC**. An entry action $\mathbf{entry_{initial_{SC}}}$ executes prior to this update, if present. Local attributes of the state machine are represented as attributes of $\mathcal{I}_{\mathbf{C}}$.
3. If the transitions triggered by an operation $\mathbf{op}(\mathbf{x})$ of **C** in **SC** are $\mathbf{tr_i}$, $\mathbf{i} : 1..\mathbf{k}$, from states $\mathbf{src_i}$ to states $\mathbf{trg_i}$, with guard $\mathbf{G_i}$ and actions $\mathbf{acts_i}$, then the behavior of $\mathbf{op}(\mathbf{x})$ is defined as a composite action $\mathbf{Code_{op}}$:

```
$\alpha(\mathbf{x}) \supset$
    $\mathbf{if}\ (\mathbf{c\_state} = \mathbf{src}_1\ \wedge\ \mathbf{G}'_1)$
    $\mathbf{then\ exit}'_{\mathbf{src}_1};\ \mathbf{acts}'_1;\ \mathbf{entry}'_{\mathbf{trg}_1};$
        $\mathbf{c\_state} := \mathbf{trg}_1$
    $\mathbf{else\ if}....$
    $\mathbf{else\ if}\ (\mathbf{c\_state} = \mathbf{src_k}\ \wedge\ \mathbf{G}'_\mathbf{k})$
    $\mathbf{then\ exit}'_{\mathbf{src_k}};\ \mathbf{acts}'_\mathbf{k};\ \mathbf{entry}'_{\mathbf{trg_k}};$
        $\mathbf{c\_state} := \mathbf{trg_k}$
```

   where α represents **op**.

   Entry actions of a state must complete before the state machine is considered to properly enter the state ("before commencing a run-to-completion step, a state machine is in a stable state configuration with all entry ... activities completed", page 561 of (OMG, 2007)). An entry action will often be used to ensure that the state invariant holds.

   If there is already an existing procedural definition $\mathbf{D_{op}}$ of **op** in the class **C**, the complete definition of **op** is $\mathbf{D}'_{\mathbf{op}};\ \mathbf{Code_{op}}$ (page 436 of (OMG, 2007); we assume that an existing pre/post specification should however always refer to the entire span of execution of **op**).

   We also need to define the effect of do-actions. These can only execute while their state is occupied:

$$\#\mathbf{active}(\mathbf{do}'_\mathbf{s}) > 0\ \Rightarrow\ \mathbf{c\_state} = \mathbf{s}$$

   and they initiate execution at the point where their state is entered (Page 548 of (OMG, 2007)):

$$\forall \mathbf{i} : \mathbb{N}_1 \cdot \uparrow(\mathbf{do}'_\mathbf{s}, \mathbf{i}) = \clubsuit((\mathbf{c\_state} = \mathbf{s}) := \mathbf{true}, \mathbf{i})$$

4. The axioms $(\mathbf{StateInv})$:

$$\mathbf{c\_state} = \mathbf{s}\ \Rightarrow\ \mathbf{Inv}'_\mathbf{s}$$

The semantics defined here corresponds to the usual 'run to completion' semantics of UML state machines: a transition only completes execution when all of its generated actions do so (page 546 of (OMG, 2007)).

A flat behavior state machine **SC** attached to an operation **op** defines an explicit algorithm for **op**. It can be formalised as a while loop action (Lano, 2008a).

4 SEMANTICS FOR STRUCTURED STATE MACHINES

We extend the semantics of flat state machines to state machines with OR and AND composite states, compound transitions and history and final states.

For each OR state **s** in the state machine, we define a state attribute $\mathbf{s\_state} : \mathbf{State_s}$ where $\mathbf{State_s}$ represents the set of normal states (including final states) directly contained in **s**. Regions of an AND state are also represented by a type and an attribute in the same manner (and so must be named). Each such OR state/region has a default initial state $\mathbf{initial_s}$ and each **s_state** is initialised to this value. If a final state is present, it is denoted by $\mathbf{final_s}$.

The top level states of a state machine **SC** are also represented by an attribute $\mathbf{SC\_state} : \mathbf{State}$.

For each state $\mathbf{s}$ in the state machine diagram, a predicate $\varphi_{\mathbf{s}}$ can be defined, which expresses that $\mathbf{s}$ is part of the current state configuration of the state machine (Table 1).

Table 1: State predicate.

State/region **s**	*State predicate* $\varphi_{\mathbf{s}}$
Top-level state	$\mathbf{SC\_state} = \mathbf{s}$
Region of AND state **p**	$\varphi_{\mathbf{p}}$
Immediate substate of OR state/region **r**	$\varphi_{\mathbf{r}} \wedge \mathbf{r\_state} = \mathbf{s}$

A predicate $\mathbf{InitialState}_{\mathbf{s}}$ similarly expresses that the (recursive) initial state of **s** is occupied.

Using these predicates, the state-changing behaviour of transitions can be expressed as pre and post conditions. The *enabling condition* $\mathbf{enc}(\mathbf{tr}, \mathbf{s})$ of a transition **tr**

$$\mathbf{s} \rightarrow_{\mathbf{op}[\mathbf{G}]/\mathbf{Post}} \mathbf{t}$$

from a state **s** is $\varphi_{\mathbf{s}} \wedge \mathbf{G}'$, conjoined with $\neg\,(\varphi_{\mathbf{ss}} \wedge \mathbf{G}1')$ for each different transition

$$\mathbf{ss} \rightarrow_{\mathbf{op}[\mathbf{G}1]/\mathbf{Post}1} \mathbf{tt}$$

triggered by **op** on a state **ss**, $\mathbf{ss} \neq \mathbf{s}$, $\mathbf{ss} \sqsubseteq \mathbf{s}$.

This expresses that **tr** is only enabled on **s** if higher-priority transitions for the same trigger operation/event are not enabled.

The complete enabling condition of **tr** is the conjunction of the enabling conditions from each explicit source of **tr** (**tr** may be a compound transition with multiple source states $\mathbf{sources}(\mathbf{tr})$):

$$\mathbf{enc}(\mathbf{tr}) \;=\; \bigwedge\nolimits_{\mathbf{s} \in \mathbf{sources}(\mathbf{tr})} \mathbf{enc}(\mathbf{tr}, \mathbf{s})$$

The precondition derived from a transition **tr** triggered by **op** is then $\mathbf{enc}(\mathbf{tr})$.

The enabling condition is a critical semantic aspect which can be defined in different ways to produce different semantic profiles for state machines. We could use instead an alternative definition $\mathbf{encs}(\mathbf{tr})$, which is the conjunction of $\mathbf{enc}(\mathbf{tr}, \mathbf{s})$ for each *explicit and implicit* source **s** of **tr**. Implicit sources are those AND state regions which contain no explicit source of **tr** but will be exited when it takes place.

For the postcondition, there are several cases. A predicate $\mathbf{Target}_{\mathbf{tr}}$ expresses what state(s) are directly entered because of the transition **tr** (Lano and Clark, 2007).

For transitions with multiple targets, the conjunction of the **Target** predicate for each target is taken.

In addition to the postcondition describing the direct target, the transition may also cause other states to be reinitialised. After taking account of the effect of history and final states, for each AND composite state **x**, if transition **tr** causes **x** to be entered, then all the regions of **x** which do not contain an explicit target of **tr** must be reinitialised. This additional effect (which may apply to several AND compositions) is expressed by a predicate $\mathbf{ReInit}_{\mathbf{tr}}$.

The complete postcondition $\Pi_{\mathbf{tr}}$ of **tr** is the conjunction of its explicit postcondition, its target state predicate(s), and $\mathbf{ReInit}_{\mathbf{tr}}$.

4.1 Structured Behavior State Machines

The semantics for structured behavioral state machines defines the sequence of actions caused by the firing of a transition, in addition to the target state predicate. In general these actions are all the actions caused by exiting the source state(s), followed by the explicit actions on the transition, followed by the actions caused by entering the target state.

A transition **tr** causes a state **s** to be (explicitly) entered if **tr** has **s** as an explicit target, or it has a target contained in **s**, and some source not contained in or equal to **s**. It causes a region **r** of an AND state **p** to be implicitly entered (at its initial state) if **p** or another region of **p** is explicitly entered because of **tr** and there is no explicit target of **tr** in **r**.

Internal transitions of a state do not cause any state entry or exit.

Table 2 shows the definition of the complete action executed when a state **s** is entered. $\mathbf{init}_{\mathbf{s}}$ is the action

Table 2: Entry actions.

State **s**	*Entry actions* $\mathbf{Entry}_{\mathbf{s}}$
basic state	$\mathbf{entry}_{\mathbf{s}}$
OR state/region, one direct substate **trg** is explicitly entered	$\mathbf{entry}_{\mathbf{s}};\ \mathbf{Entry}_{\mathbf{trg}}$
OR state/region implicitly entered or explicit target	$\mathbf{entry}_{\mathbf{s}};\ \mathbf{init}_{\mathbf{s}};\ \mathbf{Entry}_{\mathbf{initial}_{\mathbf{s}}}$
AND state with regions **r**1, ..., **rn**	$\mathbf{entry}_{\mathbf{s}};\ (\mathbf{Entry}_{\mathbf{r}1} \mid\mid ... \mid\mid \mathbf{Entry}_{\mathbf{rn}})$

on the default initial transition of **s**, in the 3rd case (cf., page 551 of (OMG, 2007)).

The parallel combinator $||$ is used in this definition because UML 2 does not prescribe any relative ordering of the combined actions (page 551 of (OMG, 2007)).

A transition **tr** causes a state **s** to be (explicitly) exited if **tr** has **s** or a substate of **s** as an explicit source, and some target which is not contained in **s**. It causes a region **r** of an AND state **p** to be implicitly exited (at its current state) if **p** or another region of **p** is explicitly exited because of **tr** and there is no explicit source of **tr** in **r**. If an OR state **p** is exited because of **tr**, the currently occupied substate of **p** will be exited.

Table 3 shows the definition of the complete action executed when a state **s** is exited.

Table 3: Exit actions.

State **s**	*Exit actions* $\mathbf{Exit_s}$
basic state	$\mathbf{exit_s}$
OR state/region, one direct substate **src** is explicitly exited	$\mathbf{Exit_{src}}; \mathbf{exit_s}$
OR state/region implicitly exited or explicit source	$\mathbf{Exit_{s\_state}}; \mathbf{exit_s}$
AND state with regions **r**1, ..., **rn**	$(\mathbf{Exit_{r1}} \mid\mid \ldots \mid\mid \mathbf{Exit_{rn}}); \mathbf{exit_s}$

The following axiom, axiom 3, defines the behaviour of an operation resulting from all the transitions for it.

If the transitions triggered by $\mathbf{op}(\mathbf{x})$ are $\mathbf{tr_i}$, $\mathbf{i} : 1..\mathbf{k}$, with actions $\mathbf{acts_i}$, then the behavior of $\mathbf{op}(\mathbf{x})$ is defined as a composite action $\mathbf{Code_{op}}$:

$$||_{\mathbf{j}:1..\mathbf{k}}\ (\mathbf{if\ enc}(\mathbf{tr_j})\ \mathbf{then\ Exit'_j};\ \mathbf{acts'_j};\ \mathbf{Entry'_j})$$

where each $\mathbf{Exit_j}$ are the exit actions caused by $\mathbf{tr_j}$ (Table 3), and $\mathbf{Entry_j}$ the entry actions (Table 2).

This definition chooses a maximal set of enabled transitions to execute at each step (page 563 of (OMG, 2007)). If no transition is enabled, a skip is performed (in accordance with the UML semantics of behavior state machines, page 561 of (OMG, 2007)).

If we know that the $\mathbf{enc}(\mathbf{tr_j})$ are mutually exclusive, this can be simplified to:

```
if $\mathbf{enc}(\mathbf{tr}_1)$
then $\mathbf{Exit'_1};\ \mathbf{acts'_1};\ \mathbf{Entry'_1}$
else if....
else if $\mathbf{enc}(\mathbf{tr_k})$
then $\mathbf{Exit'_k};\ \mathbf{acts'_k};\ \mathbf{Entry'_k}$
```

because $(\mathbf{if\ E}_1\ \mathbf{then\ C}_1)\ ||\ (\mathbf{if\ E}_2\ \mathbf{then\ C}_2)$ is equivalent to

```
if $\mathbf{E}_1 \wedge \mathbf{E}_2$ then $\mathbf{C}_1 \mid\mid \mathbf{C}_2$
else if $\mathbf{E}_1$ then $\mathbf{C}_1$
else if $\mathbf{E}_2$ then $\mathbf{C}_2$
```

For each transition **tr** triggered by an operation $\mathbf{op}(\mathbf{x})$, the pre-post behaviour due to **tr** is:

$$\forall \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{enc}(\mathbf{tr}) \circledcirc \uparrow(\mathbf{op}(\mathbf{x}), \mathbf{i})\ \Rightarrow\ \Pi_{\mathbf{tr}} \circledcirc \downarrow(\mathbf{op}(\mathbf{x}), \mathbf{i})$$

The axiom for do-actions holds with $\varphi_\mathbf{s}$ in place of $\mathbf{c\_state} = \mathbf{s}$.

Axiom 4 holds in the form

$$\varphi_\mathbf{s}\ \Rightarrow\ \mathbf{Inv'_s}$$

for each state **s**.

The semantics of behavior state machines attached to operations is generalised to structured state machines in the same way.

The above semantics can be used to give a meaning to models which extend the UML 2 standard, eg, where there are transitions which cross from one region of an AND state to another (page 572 of (OMG, 2007)).

5 REFINEMENT OF STATE MACHINES

Refinements of state machines are relationships between state machine models which establish that one state machine (the refined model) correctly implements all the behaviour of the other, abstract, model.

Often the relationship between a refined state machine **C** of a class **CC**, and an abstract state machine **A** of a class **AC** can be expressed by a morphism **f** from **C** to **A**, with the properties of *refinement* and *adequacy* (Lano et al., 2000; Lano et al., 2002; Simons, 2005).

The condition for a model **D** to refine a model **C** is that the theory of **D** should prove each axiom of the theory of **C**, possibly under some interpretation σ of the symbols of **C** in terms of those of **D**:

$$\Gamma_\mathbf{D} \vdash \sigma(\psi)$$

for each axiom ψ of $\Gamma_\mathbf{C}$. Typically σ is defined in terms of some abstraction morphism **f**.

6 REFINEMENT TRANSFORMATIONS

Using the above definition of refinement, we can verify that many refinement transformations on state machines are semantically correct:

1. A protocol statemachine can be refined into a behavior statemachine by replacing transition postconditions by actions which establish the postcondition. This is termed *protocol conformance* in (OMG, 2007).
2. Exit and entry and transition actions can be extended to invoke operations on new orthogonal regions/OR states (page 550 of (OMG, 2007)).

3. A supplier state machine can be refined, this will produce a refinement of the complete model, if the calling relationship between state machine regions/OR-states is non-cyclic.

4. A do-action of a state can be refined into a composite activity.

Other transformations, such as moving entry and exit actions from states onto transitions, can also be directly verified using our semantics.

Once a transformation has been proved correct, it can be used as required to produce or verify correct refinements, without requiring repetition of the proof.

6.1 Refinement of Supplier State Machines

This transformation involves the refinement of one composite state/region within a state machine, whilst leaving unchanged other states/regions.

Figure 2 shows an example of this transformation.

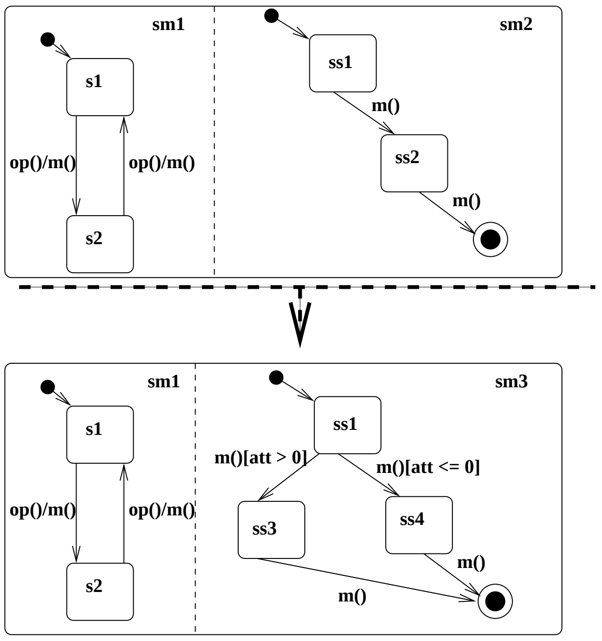

Figure 2: Refining a supplier state machine.

The axioms of the original model are:

$$
\begin{aligned}
&\mathbf{op}() \supset \ \mathbf{if}\ \mathbf{sm1\_state} = \mathrm{s1}\ \mathbf{then}\ \mathbf{m}()\ \mathbf{else}\\
&\qquad \mathbf{if}\ \mathbf{sm1\_state} = \mathrm{s2}\ \mathbf{then}\ \mathbf{m}()\\
&\forall \mathbf{i} : \mathbb{N}_1 \cdot (\mathbf{sm1\_state} = \mathbf{s1}) \circledcirc \uparrow(\mathbf{op}, \mathbf{i}) \Rightarrow\\
&\qquad (\mathbf{sm1\_state} = \mathbf{s2}) \circledcirc \downarrow(\mathbf{op}, \mathbf{i})\\
&\forall \mathbf{i} : \mathbb{N}_1 \cdot (\mathbf{sm1\_state} = \mathbf{s2}) \circledcirc \uparrow(\mathbf{op}, \mathbf{i}) \Rightarrow\\
&\qquad (\mathbf{sm1\_state} = \mathbf{s1}) \circledcirc \downarrow(\mathbf{op}, \mathbf{i})\\
&\forall \mathbf{i} : \mathbb{N}_1 \cdot (\mathbf{sm2\_state} = \mathbf{ss1}) \circledcirc \uparrow(\mathbf{m}, \mathbf{i}) \Rightarrow\\
&\qquad (\mathbf{sm2\_state} = \mathbf{ss2}) \circledcirc \downarrow(\mathbf{m}, \mathbf{i})\\
&\forall \mathbf{i} : \mathbb{N}_1 \cdot (\mathbf{sm2\_state} = \mathbf{ss2}) \circledcirc \uparrow(\mathbf{m}, \mathbf{i}) \Rightarrow\\
&\qquad (\mathbf{sm2\_state} = \mathbf{final_{sm2}}) \circledcirc \downarrow(\mathbf{m}, \mathbf{i})
\end{aligned}
$$

When **sm**2 is replaced by its refinement **sm**3, the axioms of **sm**2 are replaced by:

$$
\begin{aligned}
&\forall \mathbf{i} : \mathbb{N}_1 \cdot (\mathbf{sm3\_state} = \mathbf{ss1} \land \mathbf{att} > 0) \circledcirc \uparrow(\mathbf{m}, \mathbf{i}) \Rightarrow\\
&\qquad (\mathbf{sm3\_state} = \mathbf{ss3}) \circledcirc \downarrow(\mathbf{m}, \mathbf{i})\\
&\forall \mathbf{i} : \mathbb{N}_1 \cdot (\mathbf{sm3\_state} = \mathbf{ss1} \land \mathbf{att} \le 0) \circledcirc \uparrow(\mathbf{m}, \mathbf{i}) \Rightarrow\\
&\qquad (\mathbf{sm3\_state} = \mathbf{ss4}) \circledcirc \downarrow(\mathbf{m}, \mathbf{i})\\
&\forall \mathbf{i} : \mathbb{N}_1 \cdot (\mathbf{sm3\_state} = \mathbf{ss3}) \circledcirc \uparrow(\mathbf{m}, \mathbf{i}) \Rightarrow\\
&\qquad (\mathbf{sm3\_state} = \mathbf{final_{sm3}}) \circledcirc \downarrow(\mathbf{m}, \mathbf{i})\\
&\forall \mathbf{i} : \mathbb{N}_1 \cdot (\mathbf{sm3\_state} = \mathbf{ss4}) \circledcirc \uparrow(\mathbf{m}, \mathbf{i}) \Rightarrow\\
&\qquad (\mathbf{sm3\_state} = \mathbf{final_{sm3}}) \circledcirc \downarrow(\mathbf{m}, \mathbf{i})
\end{aligned}
$$

to form the semantics of the complete refined model. But these axioms establish the axioms of the original model, under the interpretation σ of **sm2_state** as **f**(**sm**3**_state**), where **f** is the abstraction mapping:

$$
\begin{aligned}
&\mathbf{ss1} \longmapsto \mathbf{ss1}\\
&\mathbf{ss4} \longmapsto \mathbf{ss2}\\
&\mathbf{ss3} \longmapsto \mathbf{ss2}\\
&\mathbf{final_{sm3}} \longmapsto \mathbf{final_{sm2}}
\end{aligned}
$$

Therefore the new model is a refinement of the original model. The same reasoning can be used in other cases of this transformation. If the state of the region which is being refined is referred to in the region which invokes its operations, then these references must be replaced by the equivalent expressions in terms of the refined state. Eg, a guard **in ss**2 for a transition of **sm**1 must be replaced by **in ss**3 **or in ss**4 in the refined model, in the above example.

6.2 Adding New Regions and Concurrent generations

A new region can be added to an existing AND state, provided this region does not invoke operations of other states (it is a leaf module of the calling hierarchy). The actions of existing transitions can be extended to invoke operations of the new module, in parallel with their existing actions. Likewise, the entry and exit actions of existing states can be extended in this way.

This transformation yields a refinement since

$$\alpha \,||\, \beta \supset \alpha$$

so that axiom 3 in the new system implies axiom 3 of the unrefined system, due to the monotonicity of the action constructions wrt $\supset$.

6.3 Refining do-actions into Composite Activities

This transformation replaces a do-action α of a state by an activity expressed as an internal state machine of the state.

Figure 3 shows an example of the transformation.

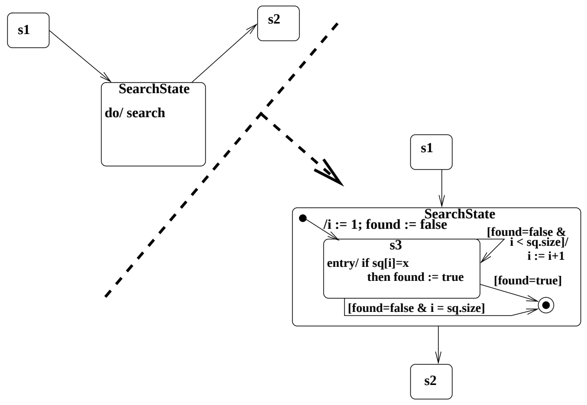

Figure 3: Refining a do-action.

In this transformation the original do action α is interpreted by the new composite action **Act**. Any axiom

$$\mathbf{op}(\mathbf{x}) \supset \mathbf{Ac}$$

defining the behavior of an operation **op** in terms of a composite action **Ac** derived from the original model is still valid in the new model, because in interpreted form it is

$$\mathbf{op}(\mathbf{x}) \supset \mathbf{Ac}[\mathbf{Act}/\alpha]$$

which is the corresponding axiom of the new model. Likewise for the other axioms of do actions.

7 CONCLUSION

We have defined an axiomatic semantics for a large part of UML 2 state machine notation, using the informal OMG superstructure definition of the semantics as the basis. Our semantics supports verification of internal consistency of state machines, of refinement between state machines, and verification of the consistency of sequence diagrams with state machines.

Other elements of UML 2 state machine notation, such as deferred states, can also be given a semantics in this formalism (Lano and Clark, 2007).

The axiomatic semantics approach has the advantage of expressing UML semantics in a high-level manner, in a formalism which is similar to, but independent of, UML. The semantics presented here is used as the basis for the code generation process of the UML-RSDS tools (Lano, 2008b), so ensuring the correctness of the generated code with respect to the specification.

In contrast to the approach of (Damm et al., 2005; Merseguer et al., 2002), we do not flatten UML state machines, but retain the structure of the machines. This enables analysis results and generated code structure to be directly related to the specifications.

As far as possible, our semantics represents the meaning of state machines in notations which are close to UML class diagram and OCL notations. The semantics may therefore be more accessible to UML users than semantics which use external formalisms such as Petri Nets (Merseguer et al., 2002) or term algebras (Lilius and Paltor, 1999). An axiomatic semantics is also well-suited for use with logic-based semantic analysis tools such as B. Compared to (Lilius and Paltor, 1999) we do not represent sync states, however we can express the semantics of time-triggered transitions using the RAL formalism (Lano, 2008a), extending (Lilius and Paltor, 1999).

The approach of (Le et al., 2006) is close to ours, but translates directly into B from statecharts, instead of utilising an underlying axiomatic semantics. Elements of UML state machine notation such as time triggers, which require a temporal logic semantics, are not handled by this approach.

Although this paper has been concerned with the semantic problems of UML, the fact that a relatively simple and coherent semantics can be assigned to UML state machines does show that the notation is suitable for developments where semantic correctness is important.

REFERENCES

Damm, W., Josko, B., Pnueli, A., and Votintseva, A. (2005). A discrete-time UML semantics for concurrency and communication in safety-critical applications. *Science of Computer Programming*, 55:81–115.

Lano, K. (1998). Logical specification of reactive and real-time systems. *Journal of Logic and Computation*, 8(5):679–711.

Lano, K. (2007). Formal specification using interaction diagrams. In *SEFM '07*.

Lano, K. (2008a). A compositional semantics of UML-RSDS. *SoSyM*.

Lano, K. (2008b). Constraint-driven development. *Information and Software Technology*.

Lano, K. and Clark, D. (2007). Direct semantics of extended state machines. *Journal of Object Technology*.

Lano, K., Clark, D., and Androutsopolous, K. (2002). From implicit specifications to explicit designs in reactive system development. In *IFM '02*.

Lano, K., Clark, D., Androutsopolous, K., and Kan, P. (2000). Invariant-based synthesis of fault-tolerant systems. In *FTRTFT*. Springer-Verlag.

Le, D., Sekerinski, E., and West, S. (2006). Statechart verification with istate. In *FM 06*.

Lilius, J. and Paltor, I. (1999). The semantics of UML state machines. Turku Centre for Computer Science, TUCS technical report 273.

Merseguer, J., Campos, J., Bernardi, S., and Donatelli, S. (2002). A compositional semantics for UML state machines aimed at performance evaluation. In Silva, M., Giua, A., and Colom, J., editors, *6 Int. Workshop on Discrete Event Systems (WODES 2002)*.

Morgan, C. (1990). *Programming from Specifications: The Refinement Calculus*. Prentice Hall.

OMG (2007). UML superstructure, version 2.1.1. OMG document formal/2007-02-03.

Simons, A. (2005). A theory of regression testing for behaviourally compatible object types. In *3rd Conf. UK Software Testing Research (5-6 September)*, pages 103–121.