Thoughts on Message-based Behavioral Signatures vis-a-vis PSL

Michael Gruninger and Richard Hull

Draft: December 3, 2003

(Not for broad distribution)

This document describes one approach for "embedding" automata-based signature descriptions of e-service compositions into PSL. In particular, the embedding is into the core PSL theory T_{disc_state}.

1 Review of model for message-based behavioral signatures

We assume a model of e-compositions as described in Hull's PODS 2003 paper. In particular, an *ec-schema* is a triple (P, C, M) of (abstract) peers, channels, and messages (or message classes), respectively. It is assumed that each message is associated with exactly one channel. Channels have a *source* and a *target*.

A *peer* is an e-service, along with a finite state automata (fsa) that serves as a (*message-based*) *behavioral signature*, and describes the peer's message-passing behavior, in terms of both incoming messages (denoted $?\alpha$) and outgoing messages (denoted $!\alpha$). The fsa might be deterministic or non-deterministic. (The nondeterminism can be used to capture the case of conditionals based on parameter values associated with incoming messages in the underlying e-service.)

Given an ec-schema $S = (P, C, M)$, an *implementation* of S is an assignment of concrete peers (or at least, behavioral signatures of concrete peers) to each abstract peer in P. In normal execution of an implementation all peers start in their start states. A move involves one peer transmitting a message to another peer (in which case the message is placed on the queue for the receiving peer), or it involves one peer "eating" a letter from its queue.

An *instantaneous description* is a complete description of a stage of processing in an execution of an implementation of an ec-schema. More formally, an instantaneous description is an n-tuple (here n is the number of peers in the ec-schema) of form $(< \sigma_1, Q_1 >, \ldots, < \sigma_n, Q_n >)$ where σ_i denotes the "current" state of the i^{th} peer and Q_i denotes the "current" contents of the queue of the i^{th} peer. One can discuss executions as sequences $I_1 \vdash I_2 \vdash \ldots \vdash I_n$, where each I_j is an instantaneous description and $\vdash$ is the "moves-to" relation. An execution is *successful* if it ends with each queue empty and each peer in a final state.

In this development we assume that the queue for each peer can have maximum length 2. It is clear how to generalize this for any finite bound; it is not clear at this point how to support unbounded queue lengths.

2 Mapping of behavioral signature model into PSL

The basic idea here is to create a family of fluents that can be used to describe instantaneous descriptions of an implementation of an ec-schema, establish some logical expressions that capture the properties of an instantaneous description, and then to encode the family of valid moves of the implementation into logical expressions.

2.1 A family of actions

Associated with each peer Γ we have the following actions

1. Γ_eats_α if there is some transition in Γ labeled by $?\alpha$.
2. Γ_sends_α if there is some transition in Γ labeled by $!\alpha$.

Note: One could also represent these as parameterized actions, e.g., to use $\Gamma\_eats(.)$, where the possible parameter values are the input alphabet for Γ. The same comment applies to the fluents described below.

2.2 A family of fluents

We assume a family of fluents for each (concrete) peer Γ. Specifically, these are

1. Γ_σ, for each state σ of Γ. This is intended to mean that Γ is in state σ.
2. Γ_i_α, for each valid input message α of Γ, and for $i \in \{1, 2\}$. This is intended to mean that the i^{th} element of $\Gamma's$ queue holds message α.
3. Γ_i_empty, for $i \in \{1, 2\}$. This is intended to mean that the i^{th} element of Γ's queue is empty. This can be viewed as short-hand for $\bigvee_{\beta \in R(\Gamma)} \neg\Gamma\_i\_\beta$, where $R(\Gamma)$ denotes the set of messages received by Γ. To this end we add the logical expressions, for $i \in \{1, 2\}$,
$$\Gamma\_i\_empty \equiv \bigvee_{\beta \in R(\Gamma)} \neg\Gamma\_i\_\beta$$
4. Γ_start, which is intended to mean that Γ is in the start state. To this end, we include the logical expression
$$\Gamma\_start \equiv \Gamma\_\sigma$$
where σ is the start state of Γ.
5. Γ_final, which is intended to mean that Γ is in a final state. To this end, we include the logical expression
$$\Gamma\_final \equiv \bigvee_{\sigma \in F(\Gamma)} \Gamma\_\sigma$$
where $F(\Gamma)$ denotes the set of final states of Γ

2.3 Restrictions of sets of fluents

We now include some "simple state constraints". These will be logical expressions that restrict valid collections of fluents to be only those corresponding to possible instantaneous descriptions. These expressions include, for each peer Γ,

For each of the following expressions ϕ the actual state constraint is $(\forall o), prior(\phi, o)$.

1. $\bigvee_{\sigma \in S(\Gamma)} \Gamma_\sigma$, where $S(\Gamma)$ is the set of states of Γ.

   I.e., Γ is in at least one state

2. $\bigwedge_{\sigma,\tau \in S(\Gamma)} [\Gamma\_\sigma \wedge \sigma \neq \tau \supset \neg \Gamma_\tau]$.

   I.e., Γ cannot be in two states at the same time.

3. $\bigwedge_{\sigma,\tau \in S(\Gamma)} [\Gamma\_i\_\sigma \wedge \sigma \neq \tau \supset \neg \Gamma\_i\_\tau]$, for $i \in \{1, 2\}$.

   I.e., Γ cannot have 2 messages in position i of its queue

4. $\bigwedge_{\sigma \in S(\Gamma)} [\Gamma\_2\_\sigma \supset \bigvee_{\tau \in R(\Gamma)} \Gamma\_1\_\tau]$.

   I.e., if there is a message in position 2 of Γ's queue, then there is a message in position 1 of that queue.

2.4 Capturing valid moves

We start by assuming that the peers are deterministic, and then indicate modifications needed in case they are non-deterministic.

We first describe some logical expressions that will hold if for some peer Γ we have $\delta_\Gamma(\sigma, ?\alpha) = \tau$.

1. (A precondition)
   $(\forall s)\ holds(\Gamma\_\sigma, s) \wedge holds(\Gamma\_1\_\alpha, s) \supset poss(\Gamma\_eats\_\alpha, s)$.

   I.e., if Γ is in state σ and α is at the head of the queue, then it is legal for the next occurrence to be of the action Γ_eats_α.

2. (An effect)
   $(\forall o) occurrence\_of(o, \Gamma\_eats\_\alpha) \wedge prior(\Gamma\_\sigma, o) \supset holds(\Gamma\_\tau, o)$.

   I.e., by the action Γ_eats_α, Γ will move to state τ.

3. (An effect)
   $(\forall o) occurrence\_of(o, \Gamma\_eats\_\alpha) \wedge prior(\Gamma\_2\_empty, o) \supset holds(\Gamma\_1\_empty, o) \wedge holds(\Gamma\_2\_empty, o)$

   I.e., if Γ eats the first member of queue, and the second slot of the queue is empty, then the both slots of the queue becomes empty.

4. (An effect)
   $(\forall o) occurrence\_of(o, \Gamma\_eats\_\alpha) \wedge prior(\Gamma\_2\_\beta, o) \supset holds(\Gamma\_1\_\beta, o) \wedge holds(\Gamma\_2\_empty, o)$

   I.e., if Γ eats the first member of queue and the second slot of the queue is non-empty, then the second member of queue moves to the first place, and the second slot of the queue becomse empty.

Note: To support non-deterministic automata, item 2 above might hold a disjunction in its consequence.

Note: The effects given above are context-sensetive. We could also create an embedding in which the effects are context-free. To do this, we would increase the number of activities. Instead of corresponding simply to a transition in a peer, each activity would correspond to a transition in a peer along with a specific manipulation of that peer's queue. The preconditions would now become more intricate, but the effects would not have any *prior* predicates in their antecedants.

We now turn to the logical expressions that will hold if for some peer Γ we have $\delta_\Gamma(\sigma, !\alpha) = \tau$. We assume that the target of message α is Φ.

1. (A precondition)
   $(\forall s) holds(\Gamma\_\sigma, s) \wedge \Phi\_2\_empty \supset poss(\Gamma\_send\_\alpha, s)$

   I.e., if Γ is in state σ and if Φ has room on its queue, then Γ can send α to Φ.

2. (An effect)
   $(\forall o) occurrence\_of(o, \Gamma\_send\_\alpha) \wedge prior(\Gamma\_\sigma, o) \wedge prior(\Phi\_1_empty, o) \supset holds(\Gamma\_\tau, o) \wedge holds(\Phi\_1\_\alpha, o) \wedge holds(\Phi\_2\_empty, o)$

   I.e., if Γ is in σ before the action Γ_send_α, and if Φ's queue is empty, then Γ moves to τ and α is stuck onto the queue of Φ.

3. (An effect)
   $(\forall o) occurrence\_of(o, \Gamma\_send\_\alpha) \wedge prior(\Gamma\_\sigma, o) \wedge prior(\Phi\_1\_\beta, o) \wedge prior(\Phi\_2\_empty, o) \supset holds(\Gamma\_\tau, o) \wedge holds(\Phi\_1\_\beta, o) \wedge holds(\Phi\_2\_\alpha, o)$

   I.e., similar to previous case except starting with β in first position of Φ's queue and second position is empty.

Note: For non-deterministic automata things are not significantly more difficult. The main thing is that a disjunction must be introduced in each of the above rules, analogous to the case of Γ moves that "eat" a message.

2.5 Capturing valid start and halting states

The following ensures that we look only at occurrence trees whose root corresponds to the start state of the composition. We include the following for each peer Γ

1. $initial(s) \supset prior(\Gamma\_start, s)$

For a computation to be in a halting state, then each peer must be in a final state and all queues must be empty. This can be captured by taking the conjunction of the following expression for each peer Γ.

1. $holds(\Gamma\_final, s) \wedge holds(\Gamma\_1\_empty, s)$

3 Some immediate questions

1. Does this PSL representation make certain kinds of reasoning easier?
2. Can we characterize the family of occurrance trees that correspond to embeddings of behavioral signature descriptions of composite services?
3. How can we blend the embedding given here with the embedding of DAML-S services into PSL?