Behavior Composition in the Presence of Failure

Sebastian Sardina  
RMIT University, Melbourne, Australia
Fabio Patrizi & Giuseppe De Giacomo  
Sapienza Univ. Roma, Italy

KR’08, Sept. 2008, Sydney Australia

Introduction

There are at least two kinds of games. One could be called finite, the other infinite.

A finite game is played for the purpose of winning ...

... an infinite game for the purpose of continuing the play.

Finite and Infinite Games  
J. P. Carse
Behavior composition vs Planning

Planning

- Operators: atomic
- Goal: desired state of affair
- Finite game: compose operator sequentially so as to reach the goal
- Playing strategy: plan

Behavior composition

- “Operators”: available transition systems
- “Goal”: target transition system
- Infinite game: compose available transition systems concurrently so as to play the target transition systems
- Playing strategy: composition controller

Behavior composition

Given:
- a set of available behaviors $B_1, \ldots, B_n$
- a target behavior $T$

we want to realize $T$ by delegating actions to $B_1, \ldots, B_n$

i.e.: control the concurrent execution of $B_1, \ldots, B_n$ so as to mimic $T$ over time

Behavior composition: synthesis of the controller
Example

Example
Example

Synthesizing a composition

Techniques for computing compositions:

- Reduction to PDL SAT [IJCAI07, AAAI07, VLDB05, ICSOC03]
- Simulation-based
- LTL synthesis as model checking of game structure [ICAPS08]

All techniques are for finite state behaviors
Simulation-based technique

Directly based on

“... control the concurrent execution of $B_1, \ldots, B_n$ so as to mimic $T$ ”

Note this is possible ...

.... if the concurrent execution of $B_1, \ldots, B_n$ can mimic $T$

Thm: this is possible iff

... the asynchronous (Cartesian) product $C$ of $B_1, \ldots, B_n$ can (ND-)simulate $T$

Simulation relation

- Given two transition systems $T = < A, S_T, t^0, \delta_T >$ and $C = < A, S_C, s^0, \delta_C >$ a (ND-)simulation is a relation $R$ between the states $t \in T$ an $(s_1, \ldots, s_n)$ of $C$ such that:
  - $(t, s_1, \ldots, s_n) \in R$ implies that
    - for all $t \rightarrow_a t'$ exists a $B_i \in C$ s.t.
      - $\exists s_i \rightarrow_a s_i'$ in $B_i$
      - $\forall s_i \rightarrow_a s_i'$ in $B_i \Rightarrow (t', s_1, \ldots, s_i', \ldots, s_n) \in R$
  - If exists a simulation relation $R$ such that $(t^0, s^0) \in R$, then we say that $T$ is simulated by $C$.
  - Simulated-by is (i) a simulation; (ii) the largest simulation.

Simulated-by is a coinductive definition
**Algorithm** Compute (ND-)simulation

**Input:** target behavior $T = \langle A, S_T, t_0, \delta_T, F_T \rangle$ and
(Cart. prod. of) available behaviors $C = \langle A, S_C, s_0, \delta_C, F_C \rangle$

**Output:** the simulated-by relation (the largest simulation)

**Body**

1. $R = \emptyset$
2. $R' = S_T \times S_C$
3. while $(R \neq R')$
   - $R := R'$
   - $R' := R' - \{ (t, s_1, \ldots, s_n) \mid \exists t \rightarrow t' \text{ in } T \land \forall B_i \cdot \neg \exists s \rightarrow_a s' \text{ in } B_i \lor \exists s_i \rightarrow_a s_i' \text{ in } B_i \land (t', s_1, s_1', \ldots, s_n) \notin R' \}$
4. return $R'$

**End**
Using simulation for composition

Given the largest simulation $R$ of $T$ by $C$, we can build every composition through the **controller generator** ($CG$).

$CG = < A, [1,...,n], S_r, s_r^0, \delta, \omega >$ with

- $A$ : the actions shared by the behaviors
- $[1,...,n]$ : the identifiers of the available behaviors
- $S_r = S_T \times S_1 \times ... \times S_n$ : the states of the controller generator
- $s_r^0 = (t^0, s_1^0, ..., s_n^0)$ : the initial state of the controller generator
- $\omega$: $S_r \times A \rightarrow 2^{[1,...,n]}$ : the output function, defined as follows:

$$
\omega(t, s_1,..,s_n, a) = \{ i | B_i \text{ can do } a \text{ and remain in } R \}
$$

- $\delta \subseteq S_r \times A \times [1,...,n] \rightarrow S_r$ : the state transition function, defined as follows

$$(t, s_1,..,s_i,..,s_n) \rightarrow_{\delta, i} (t', s_1',..,s'_i,..,s_n) \text{ iff } i \in \omega(t, s_1,..,s_i,..,s_n, a)$$

**Example**

![Diagram of T, C, and B1, B2, B3](image-url)
Results for simulation

**Thm:** Choosing at each point any value in $\omega$ gives us a correct controller for the composition.

**Thm:** Every controller that is a composition can be obtained by choosing, at each point, a suitable value in $\omega$.

**Thm:** Computing the controller generator is EXPTIME (composition is EXPTIME-complete [IJCAI07]) where the exponential depends only on the number (not the size) of the available behaviors.

Behavior failures

Components may become unexpectedly unavailable for various reasons.

We consider four kinds of behavior failures:

- A behavior **temporarily freezes**; it will eventually resume in the same state it was in;
- A behavior (or the environment) unexpectedly and arbitrarily (i.e., without respecting its transition relation) **changes its current state**;
- A behavior **dies** - it becomes permanently unavailable.
- A dead behavior unexpectedly comes **alive again** (this is an opportunity more than a failure).
Just-in-time composition

Once we have the controller generator ...

... we can avoid choosing any particular composition apriori ...

... and use directly $\omega$ to choose the available behavior to which delegate the next action.

We can be lazy and make such choice just-in-time, possibly adapting reactively to runtime feedback.

Reactive failure recovery with CG

CG already solves:

- **Temporary freezing** of an available behavior $B_i$
  - In principle: wait for $B_i$
  - But with CG: stop selecting $B_i$ until it comes back!

- **Unexpected behavior (environment) state change**
  - In principle: recompute CG / simulated-by from new initial state ...
  - ... but CG / simulated-by independent from initial state!
  - Hence: simply use old CG / simulated-by from the new state!!
Parsimonious failure recovery

**Algorithm Computing (ND-)simulation - parametrized version**

**Input:** transition system \( T = \langle A, T, t^0_T, F_T \rangle \) and transition system \( C = \langle A, S, s^0_C, \delta_C, F_C \rangle \)

relation \( R_{\text{raw}} \) including the simulated-by relation
relation \( R_{\text{sure}} \) included the simulated-by relation

**Output:** the simulated-by relation (the largest simulation)

**Body**

\[
Q = \emptyset
\]

\[
Q' = R_{\text{raw}} - R_{\text{sure}} \quad // \text{Note} \quad R' = (Q' \cup R_{\text{sure}})
\]

while \( (Q \neq Q') \) {

\[
Q := Q'
\]

\[
Q' := Q' - \{(t, s_1, \ldots, s_n) \mid t \rightarrow_a t' \in T \land \\
\forall B_i . \exists s \rightarrow_a s' \in B_i \lor \exists s_i \rightarrow_a s'_i \in B_i \land (t', s_1, \ldots, s_i, \ldots, s_n) \not\in Q' \cup R_{\text{sure}}\}
\]

return \( Q' \cup R_{\text{sure}} \)

End

---

Parsimonious failure recovery (cont.)

Let \([1, \ldots, n] = W \cup F\) be the available behaviors.

Let \( R = R_{WUF} \) be the simulated-by relation of target by behaviors \( W \cup F \).

Then the following hold:

- \( R_W \subseteq \pi_W(R_{WUF}) \)
  - \( \pi_W(R_{WUF}) \) is not a simulation in general
  - Behaviors \( F \) die: compute \( R_W \) with \( R_{\text{raw}} = \pi_W(R_{WUF}) \)

- \( R_W \times F \subseteq R_{WUF} \)
  - \( R_W \times F \) is a simulation of target by behaviors \( W \cup F \)
  - Dead behaviors \( F \) come back: compute \( R_{WUF} \) with \( R_{\text{sure}} = R_W \times F \)
Tools for computing composition based on simulation

- Computing simulation is a well-studied problem (related to bisimulation, a key notion in process algebra). Tools, like the Edinburgh Concurrency Workbench and its clones, can be adapted to compute composition via simulation.

- Also LTL-based syntesis tools, like TLV, can be used for (indirectly) computing composition via simulation [Patrizi PhD08]

  \[ We \ are \ currently \ focussing \ on \ the \ second \ approach. \]

Conclusion

- Behavior composition: \textit{an infinite game}.

- Simulation based composition techniques allow for \textit{failure tolerance}!

- It relies on a \textit{controller generator}: kind of stateful universal plan generator for composition.

- \textit{Full observability} of available behavior’ states is crucial for CG to work properly. But ... \textit{Partial observability} addressable by manipulating knowledge states! [work in progress]

- All techniques are for finite states. What about dealing with infinite states? Very difficult, but also crucial when \textit{mixing processes and data}!