Realising Deterministic Behavior from Multiple Non-Deterministic Behaviors

Thomas Ströder
Department of Computer Science
Aachen University of Technology
Ahorn Str. 55
D-52056, Aachen, Germany
thomas.stroeder@rwth-aachen.de

Maurice Pagnucco
National ICT Australia and ARC Centre
of Excellence in Autonomous Systems
School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW, 2052, Australia
morri@cse.unsw.edu.au

Abstract

This paper considers the problem of composing or scheduling several (non-deterministic) behaviors so as to conform to a specified target behavior as well as satisfying constraints imposed by the environment in which the behaviors are to be performed. This problem has already been considered by several works in the literature and applied to areas such as web service composition, the composition of robot behaviors and co-ordination of distributed devices. We develop a sound and complete algorithm for determining such a composition which has a number of significant advantages over previous proposals: a) our algorithm is different from previous proposals which resort to dynamic logic or simulation relations, b) we realized an implementation in Java as opposed to other approaches for which there are no known implementations, c) our algorithm determines all possible schedulers at once, and d) we can use our framework to define a notion of approximation when the target behavior cannot be realized.

Building and developing re-usable modules is one of the cornerstones of computer science. Furthermore, building on previously established infrastructure has allowed us to construct elaborate and sophisticated structures from skyscrapers and aeroplanes through to the world-wide web. Developing the components that go into these structures is only part of the problem however. Once we have them in place, we need to develop methods for piecing them together so as to achieve the desired outcome.

In this paper we consider the problem of composing behaviors. This problem has already attracted some attention in the recent literature [Berardi *et al.*, 2008; Calvanese *et al.*, 2008; de Giacomo and Sardina, 2007; Sardina and de Giacomo, 2007; Sardina *et al.*, 2008; Sardina and de Giacomo, 2008; Berardi *et al.*, 2006b; 2006a] with several proposals being put forward. More precisely, we consider the problem of composing or scheduling several (non-deterministic) behaviors so as to conform to a specified (deterministic) target behavior as well as satisfying constraints imposed by the environment in which the behaviors are to be performed. These behaviors are abstractions that can represent a variety of mechanisms such as programs, robot actions, capabilities of software agents or physical devices, etc. As such, solutions to this problem have a wide field of applicability from composing web services [Berardi *et al.*, 2008] through to co-ordinating multiple robots or software agents [de Giacomo and Sardina, 2007; Sardina and de Giacomo, 2007; Sardina *et al.*, 2008; Sardina and de Giacomo, 2008]. The closest work to this paper is that of Sardina *et al.* [2008] which proposes a regression based technique to solving this problem where we present a progression based technique here and briefly consider the possibility of approximating the target behaviour.

For example, consider an urban search and rescue setting with three types of robots. *Scout robots* can search for victims and report their location. *Diagnosis robots* can assess victims and determine whether their condition requires special transportation. If not, this robot can guide victims to safety. *Rescue robots* can carry immobile victims to safety. We will elaborate this example using our framework in this paper.

This paper provides four main contributions that improve on previous approaches to this problem: 1) we provide a sound and complete algorithm for solving the behavior composition problem which works in the way of a forward search—this is in contrast to the proposal of Sardina et al. [2008] which can be seen as a backward search; 2) we have realized an implementation of our algorithm in Java which is the first known implementation of a solution to this problem; 3) our algorithm determines all possible schedulers for the target behavior (as can the proposals in [Berardi et al.*, 2008; Sardina* et al.*, 2008]); and, 4) our approach allows the definition of approximate solutions to the behavior composition problem which can be used when the target behavior cannot be realized by the available behaviors in the given environment.*

1 Background

The basic components of our framework are:

Environment which provides an abstract notion of the observable effects of actions and preconditions for actions.

Behaviors are skills or capabilities and are essentially programs that entities can perform.

Target behavior is the desired behavior that we would like to carry out.

Each of these components can be specified abstractly using automata as we shall see below. We now look at these components in more detail and introduce some other concepts that we require. In doing so we use the formulation of de Giacomo and Sardina [2007] and reproduce the important definitions from their paper that we require here with one modification. We allow multiple initial states where they do not. Otherwise the definitions are largely as in their paper and interested readers are pointed to this article for further details and intuitions behind this formulation of the behavior composition problem.

We begin with a definition of the *environment*. Note that incomplete knowledge about actions as well as failure of actions are dealt with by allowing the environment and the behaviours to be non-deterministic.

Definition 1.1 ([de Giacomo and Sardina, 2007]) *An* environment $\mathcal{E} = (\mathcal{A}, E, I_{\mathcal{E}}, \delta_{\mathcal{E}})$ *is characterized by the components:*

- $\mathcal{A}$ *a finite set of shared actions*
- E *a finite set of possible environment states*
- $I_{\mathcal{E}} \subseteq E$ *the set of initial states of the environment*
- $\delta_{\mathcal{E}} \subseteq E \times \mathcal{A} \times E$ *the transition relation among states:* $\delta_{\mathcal{E}}(e, a, e')$ *holds when action* a *performed in state* e *may lead the environment to successor state* e'.

Example 1.1 *Considering the urban search and rescue example introduced earlier. In the following example environment the set of actions* $\mathcal{A}$ = {diagnose, report, return, search, special, transport} *where each of the individual actions has the obvious interpretation given the description above. The set of states* $E = \{e_1, e_2, e_3, e_4\}$, *the initial state* $I_{\mathcal{E}} = \{e_1\}$ *and the transitions* $\delta_{\mathcal{E}}$ *are given in the following automaton.*

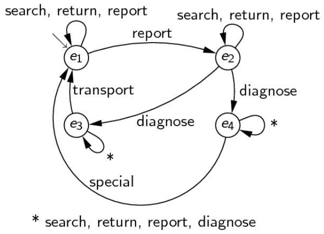

Figure 1: Environment

Next we turn to *behaviors* of which there are two types: *available behaviors* that specify the capabilities of entities that are under our control when it comes to scheduling and the *target behavior* that specifies the desired functionality.

Definition 1.2 ([de Giacomo and Sardina, 2007]) Behavior $\mathcal{B} = (S, I_{\mathcal{B}}, G, \delta_{\mathcal{B}}, F)$ *over an environment* $\mathcal{E}$ *consists of:*

- S *a finite set of behavior states*
- $I_{\mathcal{B}} \subseteq S$ *the set of initial states of the behavior*
- G *a set of guards, which are Boolean functions* $g : E \rightarrow \{true, false\}$ *for environment states* E *of* $\mathcal{E}$
- $\delta_{\mathcal{B}} \subseteq S \times G \times \mathcal{A} \times S$ *transition relation, where* $\mathcal{A}$ *is the set of actions of* $\mathcal{E}$*; the* $G \times A$ *component is the* label *of the transition*
- $F \subseteq S$ *set of final states in which the behavior may halt.*

Note that available behaviors may also be non-deterministic and so may not be fully under the scheduler's control while the target behavior is deterministic.

Example 1.2 *Continuing our example, the automaton in Figure 2 specifies the target behavior that we wish to obtain.* $S = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, $I_{\mathcal{B}} = \{t_1\}$, $F = \{t_1\}$, *transitions are represented by the arcs of the automaton.*

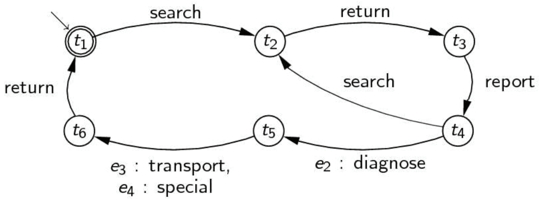

Figure 2: Target behavior

The following definitions will assist in our development of a *scheduler* that will compose the available behaviors according to the environment so as to achieve the target behavior. Runs describe transitions that can be realized by a behavior in the environment.

Definition 1.3 ([de Giacomo and Sardina, 2007]) *For behavior* $\mathcal{B} = (S, I_{\mathcal{B}}, G, \delta_{\mathcal{B}}, F)$ *and environment* $\mathcal{E} = (\mathcal{A}, E, I_{\mathcal{E}}, \delta_{\mathcal{E}})$, runs *of* $\mathcal{B}$ *on* $\mathcal{E}$ *are (possibly infinite) alternating sequences:* $(s^0, e^0)a^1(s^1, e^1)a^2 \cdots$, *where* $s^0 \in I_{\mathcal{B}}$ *and* $e^0 \in I_{\mathcal{E}}$, *and for every* i $(s^i, e^i)a^{i+1}(s^{i+1}, e^{i+1})$ *is such that:*

- *there is a transition* $(e^i, a^{i+1}, e^{i+1}) \in \delta_{\mathcal{E}}$
- *there is a transition* $(s^i, g^{i+1}, a^{i+1}, s^{i+1}) \in \delta_{\mathcal{B}}$, *such that* $g^{i+1}(e^i) = true$.

If the run is finite, i.e., $(s^0, e^0)a^1 \cdots a^l(s^l, e^l)$, *then* $s^l \in F$.

Traces are legal sequences of actions according to runs.

Definition 1.4 ([de Giacomo and Sardina, 2007]) *A* trace *is a sequence of pairs* (g, a), *where* $g \in G$ *is a guard of* $\mathcal{B}$ *and* $a \in A$ *an action, of the form* $t = (g^1, a^1) \cdot (g^2, a^2) \cdots$ *such that there is a run* $(s^0, e^0)a^1(s^1, e^1)a^2 \cdots$, *where* $g^i(e^{i-1}) = true$ *for all* i*. If trace* $t = (g^1, a^1) \cdots (g^l, a^l)$ *is finite, then there is a finite run* $(s^0, e^0)a^1 \cdots a^l(s^l, e^l)$ *with* $s^l \in F$*. We call* l *the* length *of* t.

A *system*, then, consists of the available behaviors that we can compose to realize the target behavior and the environment they must satisfy.

Definition 1.5 ([de Giacomo and Sardina, 2007]) System $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ *is formed by an environment* $\mathcal{E}$ *and* n *non-deterministic available behaviors* $\mathcal{B}_i$*. A* system configuration *is a tuple* $(s_1, \ldots, s_n, e)$ *specifying a snapshot of the system where behavior* $\mathcal{B}_i$ *is in state* s_i

and environment $\mathcal{E}$ in state e. The system has a specific component, called the scheduler *able to activate, stop, and resume behaviors at each time point.*

Example 1.3 *The component behaviors are given by the automata in Figure 3. The environment and target behavior are given in the previous examples.*

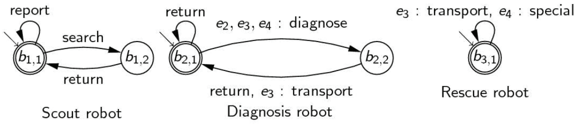

Figure 3: Component behaviors

The behavior composition problem can now be specified. We are provided with a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ and a deterministic target behavior $\mathcal{B}_0$ over $\mathcal{E}$. Our task is to schedule the available behaviors so that they realize the target behavior. At each point in time the scheduler can activate one of the available behaviors so that it can perform one action. In doing so, the scheduler ensures that this action is executable in the environment. We now formalize these notions.

Definition 1.6 ([de Giacomo and Sardina, 2007]) *Let $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ be a system, $\mathcal{E} = (\mathcal{A}, E, I_\mathcal{E}, \delta_\mathcal{E})$ the environment, $\mathcal{B}_i = (S_i, I_{\mathcal{B}_i}, G_i, \delta_i, F_i)$ available behaviors and $\mathcal{B}_0 = (S_0, \{s_0^0\}, G_0, \delta_0, F_0)$ the target behavior. A* system history *is an alternating sequence of system configurations and actions $h = (s_1^0, \ldots, s_n^0, e^0) \cdot a^1 \cdot (s_1^1, \ldots, s_n^1, e^1) \cdots (s_1^{l-1}, \ldots, s_n^{l-1}, e^{l-1}) \cdot a^l \cdot (s_1^l, \ldots, s_n^l, e^l)$ where:*

- *$s_i^0 \in I_{\mathcal{B}_i}$ for $i \in \{1, \ldots, n\}$, i.e., behaviors start in an initial state*
- *$e^0 \in I_\mathcal{E}$, i.e, the environment starts in an initial state*
- *at each step $0 \leq k \leq l$, there is an $i \in \{1, \ldots, n\}$ where $(s_i^k, g_i^{k+1}, a^{k+1}, s_i^{k+1}) \in \delta_i$ and for all $j \neq i$, $s_j^{k+1} = s_j^k$, i.e, at each step in the history only one of the behaviors—$\mathcal{B}_i$—has made a (legal) transition*
- *at each step $0 \leq k \leq l$, we have $(e^k, a^{k+1}, e^{k+1}) \in \delta_\mathcal{E}$, i.e, the environment also makes a legal transition.*

Definition 1.7 ([de Giacomo and Sardina, 2007]) *A* scheduler *is a function $P : \mathcal{H} \times \mathcal{A} \rightarrow \{1, \ldots, n, u\}$ that, for history $h \in \mathcal{H}$ ($\mathcal{H}$ is the set of all system histories as defined above) and action $a \in \mathcal{A}$, returns the behavior (via its index) scheduled to perform the action.*

de Giacomo and Sardina [2007] allow the scheduler to also return the value u to specify that no behavior can perform the action after the history. We now formalize what constitutes a solution to the behavior composition problem.

Definition 1.8 ([de Giacomo and Sardina, 2007]) *Let $t = (g^1, a^1) \cdot (g^2, a^2) \cdots$ be a trace of the target behavior. A scheduler program P* realizes *the trace t iff for all l and all system histories $h \in \mathcal{H}_{t,P}^l$ (see below) such that $g^{l+1}(e_h^l) = true$ in the last environment state e_h^l of h, we have that $P(h, a^{l+1}) \neq u$ and $\mathcal{H}_{t,P}^{l+1}$ is non-empty. The set of system histories $\mathcal{H}_{t,P}^l$ is inductively defined:*

- *$H_{t,P}^0 = \bigcup_{e_0 \in I_\mathcal{E}, s_{10} \in I_{\mathcal{B}_1}, \ldots, s_{n0} \in I_{\mathcal{B}_n}} \{(s_{10}, \ldots, s_{n0}, e_0)\}$*
- *$H_{t,P}^{l+1}$ is the set of l + 1-length system histories of the form $h \cdot a^{l+1} \cdot (s_1^{l+1}, \ldots, s_n^{l+1}, e^{l+1})$ such that:*
  - *$h \in \mathcal{H}_{t,P}^l$, where $(s_1^l, \ldots, s_n^l, e^l)$ is the last system configuration in h*
  - *a^{l+1} is an action where $P(h, a^{l+1}) = i$, with $i \neq u$, i.e., the scheduler specifies action a^{l+1} at system history h to be executed in behavior $\mathcal{B}_i$*
  - *$(s_i^l, g, a^{l+1}, s_i') \in \delta_i$ with $g(e^l) = true$, i.e., behavior $\mathcal{B}_i$ evolves from its current state s_i^l to state s_i' wrt the (current) environment state e^l*
  - *$(e^l, a^{l+1}, e^{l+1}) \in \delta_\mathcal{E}$, i.e., the environment may evolve from current state e^l to e^{l+1}*
  - *$s_i^{l+1} = s_i'$ and $s_j^{l+1} = s_j^l$, for $j \neq i$, i.e., only behavior $\mathcal{B}_i$ is allowed to perform a step.*

A scheduler program P realizes *target behavior $\mathcal{B}_0$ if it realizes all its traces.*

2 The expansion algorithm

Our algorithm for computing a scheduler program continually passes through several phases. States are *expanded*, possibly *instantiated* by matching them with existing states and those states violating the constraints imposed by the target behavior are *deleted*. The idea is to start with a state representing all possible initial configurations and expand states in a way that every configuration reachable from a configuration in the current state by performing an action according to a transition in the target behavior is represented by one of the expanded children of the current states. To avoid calculating an infinite graph we use an instance relation to 'close' the graph back to an already existing state. de Giacomo and Sardina [2007] show that when a target behavior can be realized it can be done so with a finite number of states therefore justifying our ability to produce finite graphs (albeit with loops). We could expand states that end up in a configuration where we are not able to perform all actions possible in the target behavior. These configurations (i.e., the states representing them) have to be deleted from our graph. This is achieved by a *marking algorithm.* Since the marking algorithm can mark states where there are instances of the marked states in the graph, expansion and marking steps have to be performed alternately until either no states with instances in the graph are marked or the initial state is marked (i.e., so no scheduler exists). We now formalize the notions we require for our algorithm.

States. A state for our expansion algorithm differs from those of behaviors and the environment and must contain a number of properties. First it represents a state in the target behavior. Second the states and the relations between them represent a set of system histories and thus a state contains a system configuration that can be seen as a predecessor step in every system history belonging to that certain state. Third we need to know which behavior should execute which action in the scheduler program. Therefore a state contains an action that is to be performed in the predecessor configuration and a number representing the index of the behavior that is supposed to perform the action. Finally a state contains a set of configurations combined with a set of actions. This set contains every reachable configuration from the predecessor configuration (after executing the action in this state) combined with every action that must be performable in the reached state of the target behavior. Using this formalization we need a special treatment for the first state when there is no predecessor configuration and action. Thus we define a distinguished action $start$ and a distinguished state $null$ to be added to the set of actions $\mathcal{A}' = \mathcal{A} \cup \{start\}$ and the sets of available behavior states and environment states: $S'_x = S_x \cup \{null\}$ for all $x \in \{1, \ldots, n, \mathcal{E}\}$. As technical properties we also enumerate the states (to distinguish them even if their other properties are identical) and give them a Boolean label for the marker algorithm described below. Formally:

Definition 2.1 *A* <u>state</u> *for our expansion algorithm wrt a given system* $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ *and target behavior* $\mathcal{B}_0$ *over* $\mathcal{E}$ *is an element* $s \in \mathbb{N}_0 \times \{\mathit{true}, \mathit{false}\} \times \mathit{States}$ *where* $\mathit{States} = S_0 \times \{0, \ldots, n\} \times \mathcal{A}' \times S'_1 \times \ldots \times S'_n \times S'_{\mathcal{E}} \times 2^{S_1 \times \ldots \times S_n \times S_{\mathcal{E}} \times 2^{\mathcal{A}}}$.

For a state $s = (m, l, (s_0, b, a, s_1, \ldots, s_n, e, O))$ the real state information without technical properties like a state number is just $(s_0, b, a, s_1, \ldots, s_n, e, O)$, where the O stands for obligations. The intuition behind the notion of obligation is, that every combination of a system configuration and an action from a corresponding set in O can be seen as an obligation for peforming the next step of a trace. Our algorithm develops a finite graph $SG = (V, E)$ with $V \subseteq \mathbb{N}_0 \times \{\mathit{true}, \mathit{false}\} \times \mathit{States}$ representing one or more schedulers.

Successor Actions. We are interested in being able to perform all actions possible in a current state of the target behavior. Therefore, given a behavior $\mathcal{B}$ over an environment $\mathcal{E}$ we define the set of *successor actions* for a state $s \in S$ in the environment state $e \in E$ as $\mathit{SuccessorActions}(s, e) := \{a \in \mathcal{A} | \exists s' \in S : \exists g \in G : \delta_{\mathcal{B}}(s, g, a, s') \wedge g(e)\}$.

Expansion Function. We define a function $\mathit{expand} : \mathbb{N}_0 \times \{\mathit{true}, \mathit{false}\} \times \mathit{States} \to 2^{\mathit{States}}$ to calculate the children of a state. Intuitively, $\mathit{expand}(s)$ calculates a set consisting of one state for every obligation in s (as mentioned above), where these states represent all possible outcomes (and new obligations) after performing the respective action in the respective system configuration. So given a state $s = (m, l, (s_0, b, a, s_1, \ldots, s_n, e, O))$ wrt a given system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ and target behavior $\mathcal{B}_0$ over $\mathcal{E}$ the expansion of s is defined as $\mathit{expand}(s) = \{(s'_0, b', a', s'_1, \ldots, s'_n, e', O') | (s'_1, \ldots, s'_n, e', A') \in O, a' \in A', O' = \{(s^o_1, \ldots, s^o_n, e^o, \mathit{SuccessorActions}(s'_0, e^o))| \delta_{\mathcal{E}}(e', a', e^o), \exists g^{b'} \in G_{b'} : \delta_{\mathcal{B}_{b'}}(s'_{b'}, g^{b'}, a', s^o_{b'}) \wedge g^{b'}(e'), s^o_i = s'_i \forall i \neq b'\} \neq \emptyset, \exists g \in G_0 : \delta_{\mathcal{B}_0}(s_0, g, a', s'_0) \wedge g(e')\}$.

Instance Relation. To obtain a finite graph representing the scheduler program, we need an *instance relation* that can be used to match states with existing ones and so add edges in the graph to already existing states. Instances represent the same state in the target behavior. Given two states $s^1 = (m^1, l^1, (s^1_0, b^1, a^1, s^1_1, \ldots, s^1_n, e^1, O^1))$ and $s^2 = (m^2, l^2, (s^2_0, b^2, a^2, s^2_1, \ldots, s^2_n, e^2, O^2))$ wrt a given system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ and target behavior $\mathcal{B}_0$ over $\mathcal{E}$ the relation $\mathit{instance}(s^1, s^2)$ (s^1 is an instance of s^2) holds iff $m^1 \neq m^2 \wedge l^1 \wedge l^2 \wedge s^1_0 = s^2_0 \wedge \forall (s^i_1, \ldots, s^i_n, e^i, A^1) \in O^1 \exists (s^i_1, \ldots, s^i_n, e^i, A^2) \in O^2 : A^1 \subseteq A^2$. s^1 is then called *instance child* and s^2 is called *instance father*. Intuitively, this means that we have an instance if the set of obligations for one state (s^1) is a subset of the other (s^2). The initial part of the condition ($m^1 \neq m^2 \wedge l^1 \wedge l^2$) is only required for technical reasons so that states already marked false are ignored as are identical states.

Complete Expansion Step Algorithm. Given a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ and target behavior $\mathcal{B}_0$ over $\mathcal{E}$, a finite graph $SG = (V, E)$ with $V \subseteq \mathbb{N}_0 \times \{\mathit{true}, \mathit{false}\} \times \mathit{States}$ and $E \subseteq V \times V$ and a set of states to expand $\mathit{TODO} \subseteq V$, the complete expansion step algorithm is:

```
 1: procedure: expansion_step($SG, \mathcal{S}, \mathcal{B}_0, TODO$)
 2: while $TODO \neq \emptyset$ do
 3:   $NEW := \emptyset$
 4:   for $s \in TODO$ do
 5:     if $\exists s' \in V : instance(s, s')$ then
 6:       $E := E \cup \{(s, s')\}$ with $instance(s, s') \wedge \forall s'' \in V : \neg(instance(s', s'') \wedge Obligation(s') \neq Obligation(s''))$
 7:     else
 8:       $C := expand(s)$
 9:       for $c \in C$ do
10:         $s' := (x, true, c)$ where $x$ is fresh
11:         $V := V \cup \{s'\}$
12:         $E := E \cup \{(s, s')\}$
13:         $NEW := NEW \cup \{s'\}$
14:       end for
15:     end if
16:   end for
17:   $TODO := NEW$
18: end while
```

Line 5 of the algorithm checks whether the state s currently under consideration for expansion has an instance. If so, line 6 adds an instance edge in a way that the instance father is not an instance child of another instance father again. Otherwise we obtain the children of the state to be expanded at line 8 and add each one to the graph SG (lines 11–13).

Legal and Illegal States. Given a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ and target behavior $\mathcal{B}_0$ over $\mathcal{E}$ and a finite graph $SG = (V, E)$ with $V \subseteq \mathbb{N}_0 \times \{\mathit{true}, \mathit{false}\} \times \mathit{States}$ and $E \subseteq V \times V$ we call a state $s = (m, l, (s_0, b, a, s_1, \ldots, s_n, e, O)) \in V$ *legal* iff

- $l = \mathit{true}$,
- if s_0 is a final state of $\mathcal{B}_0$ then $s_1, \ldots, s_n$ are final states of $\mathcal{B}_1, \ldots, \mathcal{B}_n$,
- if s has a parent state, then the parent s is legal and either
  - s is an instance of another state: $\exists (s, s') \in E : \mathit{instance}(s, s')$, or
  - s can fulfill all its obligations by its children: $\forall o = (s'_1, \ldots, s'_n, e', A') \in O : \forall a' \in A' : \exists c = (m^c, l^c, (s^c_0, b^c, a^c, s^c_1, \ldots, s^c_n, e^c, O^c)) \in V : E(s, c) \wedge l^c \wedge e' = e^c \wedge s'_1 = s^c_1 \wedge \ldots \wedge s'_n = s^c_n \wedge a' = a^c$

The parent state of s is the only state s_p with $E(s_p, s) \wedge \neg instance(s_p, s)$ (only the root state has no parent state). Consequently we call s *illegal* iff s is not legal. Intuitively, an illegal state is either not achievable by performing a trace or at some point while performing a trace achieving this state, there is at least one trace with the same beginning up to that point which cannot be performed by the system anymore. We will be interested in a special set of illegal states in the graph SG, namely those illegal states with a $true$ label and at least one child which is not an instance father of that state (when this child is labeled with $true$). These are exactly the states in which we cannot perform every trace of the target behaviour anymore. So we define the set $Illegal(SG) := \{v = (m, l, (s_0, b, a, s_1, \ldots, s_n, e, O)) \in V |$ v is illegal , $l = true, \exists v' = (m', l', (s'_0, b', a', s'_1, \ldots, s'_n, e', O')) \in V : E(v, v') \wedge \neg instance(v, (m', true, (s'_0, b', a', s'_1, \ldots, s'_n, e', O')))\}$.

Furthermore we will be interested in those illegal states with a $true$ label that have become illegal just because all states of which it is an instance have become illegal or simply have not been expanded yet. Thus we define the set $Remainder(SG) := \{v = (m, l, (s_0, b, a, s_1, \ldots, s_n, e, O)) \in V | v$ is illegal, $l = true, \forall v' = (m', l', (s'_0, b', a', s'_1, \ldots, s'_n, e', O')) \in V : E(v, v') \Rightarrow instance(v, (m', true, (s'_0, b', a', s'_1, \ldots, s'_n, e', O')))\}$.

Marker Algorithm The marker algorithm simply marks all illegal states from our special set after each expansion step so that they can be subsequently deleted. Thus, given a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ and target behavior $\mathcal{B}_0$ over $\mathcal{E}$ and a finite graph $SG = (V, E)$ with $V \subseteq \mathbb{N}_0 \times \{true, false\} \times States$ and $E \subseteq V \times V$ we have:

```
procedure: mark($SG, \mathcal{S}, \mathcal{B}_0$)
while $Illegal(SG) \neq \emptyset$ do
    for $s = (m, l, (s_0, b, a, s_1, \ldots, s_n, e, O)) \in Illegal(SG)$ do
        $l := false$
    end for
end while
```

This algorithm simply determines the illegal states and marks each corresponding vertex in SG with the label $false$.

Complete Expansion Algorithm. For initializing our expansion algorithm we need to compute all possible start configurations. Thus, given a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ and target behavior $\mathcal{B}_0 = (S_0, s^0_0, G_0, \delta_{\mathcal{B}_0}, F_0)$ over $\mathcal{E}$ we define the root state $s_{root} := (0, true, (s^0_0, 0, start, null, \ldots, null, null, O_{root}))$ where $O_{root} := \{(s_1, \ldots, s_n, e, SuccessorActions(s^0_0, e)) | s_1 \in I_{\mathcal{B}_1}, \ldots, s_n \in I_{\mathcal{B}_n}, e \in I_{\mathcal{E}}\}$. Now the complete expansion algorithm is:

```
1:  procedure: expansion_algorithm($\mathcal{S}, \mathcal{B}_0$)
2:  $SG := (\{s_{root}\}, \emptyset)$
3:  $TODO := \{s_{root}\}$
4:  while $s_{root}$ is labeled with $true \wedge TODO \neq \emptyset$ do
5:      expansion_step($SG, \mathcal{S}, \mathcal{B}_0, TODO$)
6:      mark($SG, \mathcal{S}, \mathcal{B}_0$)
7:      $TODO := Remainder(SG)$
8:  end while
9:  if $s_{root}$ is labeled with $true$ then
10:     return $SG$
11: else
12:     return "computation failed"
13: end if
```

The core of the algorithm (lines 4–8) repeatedly applies the expansion step and marker algorithms. Provided the node corresponding to the initial state in the graph SG is not labeled $false$, there is at least one scheduler capable of realizing the target behavior.

Theorem 2.1 *The algorithms presented above satisfy the following formal properties:*

1. *the expansion step algorithm and instance relations preserve the possibility of realizing all traces of the target behavior in all ways possible for the system*
2. *the marker algorithm marks as false exactly those states achievable by performing a trace, where at some point while performing this trace there is at least one trace with the same beginning up to that point which cannot be performed by the system any more*
3. *each of the three algorithms terminates.*

Histories Represented by Calculated Graph. The intuitive connection between the graph calculated by our algorithm and the schedulers it represents lies in the system histories that can be simulated by the graph. These histories correspond directly to traces of the target behavior and thus we can read schedulers realizing the target behavior from our graph by looking at the system histories we can simulate with it. The simulation of histories is done by performing the respective action with the respective behavior in every state we reach by following the edges of the graph (and doing nothing on instance edges). Formally, the finite graph $SG = (V, E)$ calculated by the expansion algorithm represents all system histories h with $h = (s^0_1, \ldots, s^0_n, e^0) \cdot a^1 \cdot (s^1_1, \ldots, s^1_n, e^1) \cdots (s^{l-1}_1, \ldots, s^{l-1}_n, e^{l-1}) \cdot a^l \cdot (s^l_1, \ldots, s^l_n, e^l)$ where we have $\exists (s^0_1, \ldots, s^0_n, e^0, A) \in O_{root}$, $s^0 = s_{root}$ and there exist $s^1 = (m^{s^1}, true, (s^{s^1}_0, b^{s^1}, a^{s^1}, s^{s^1}_1, \ldots, s^{s^1}_n, e^{s^1}, O^{s^1})), \ldots, s^l = (m^{s^l}, true, (s^{s^l}_0, b^{s^l}, a^{s^l}, s^{s^l}_1, \ldots, s^{s^l}_n, e^{s^l}, O^{s^l})) \in V$ such that $(s^i_1, \ldots, s^i_n, e^i, A^i) \in O^{s^i}$, $(s^{i-1}_1, \ldots, s^{i-1}_n, e^{i-1}) = (s^{s^i}_1, \ldots, s^{s^i}_n, e^{s^i})$, $a^i \in A^{s^{i-1}}$ and $((s^{i-1}, s^i) \in E \wedge a^i = a^{s^i} \wedge \forall s' \in V : \neg(E(s^i, s') \wedge instance(s^i, s'))) \vee \exists s_{inst} \in V : ((s^{i-1}, s_{inst}) \in E \wedge (s_{inst}, s^i) \in E \wedge instance(s_{inst}, s^i) \wedge a^i = a_{inst})$ for all $i \in \{1, \ldots, l\}$. The set of all histories represented by SG is denoted by $\mathcal{H}(SG)$.

Represented Schedulers. A finite graph $SG = (V, E)$ calculated by the expansion algorithm represents all functions $P : \mathcal{H} \times \mathcal{A} \rightarrow \{1, \ldots, n, u\}$ with $P(h, a) \in Next(h, a)$ if $h \in \mathcal{H}(SG)$, where $Next(h, a) := \{b \in \{1, \ldots, n\} | \exists s_{next} = (m, true, (s_0, b, a, s_1, \ldots, s_n, e, O)) \in V : (reached(h, SG), s_{next}) \in E\}$ if this set is not empty and $Next(h, a) := \{1, \ldots, n, u\}$ otherwise for all $h \in \mathcal{H}$ and $a \in \mathcal{A}$.

Example 2.1 *Completing our example, the scheduler in Figure 4 gives the realization of the target behavior.*

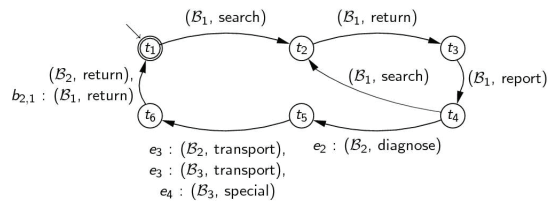
($\mathcal{B}_1$, search)
($\mathcal{B}_1$, return)
t_1
t_2
t_3
($\mathcal{B}_2$, return),
$b_{2,1}$: ($\mathcal{B}_1$, return)
($\mathcal{B}_1$, search)
($\mathcal{B}_1$, report)
t_6
t_5
t_4
e_3 : ($\mathcal{B}_2$, transport),
e_3 : ($\mathcal{B}_3$, transport),
e_4 : ($\mathcal{B}_3$, special)
e_2 : ($\mathcal{B}_2$, diagnose)

Figure 4: Result

In general, more than one scheduler may be possible and the expansion algorithm calculates a graph representing all schedulers realizing the target behavior.

Theorem 2.2 *Given a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ and target behavior $\mathcal{B}_0$ over $\mathcal{E}$, the expansion algorithm returns a finite graph SG representing all schedulers which realize the target behavior if there exists at least one such scheduler. Otherwise it returns "computation failed".*

Approximations. The method presented here can easily be used to calculate approximations by changing the marker step in such a way that it doesn't mark all states where the simulation of the target behavior by the system is not possible in every case (and so would become illegal). In this way we can obtain results even in situations where there is no exact solution because the available behaviors are incapable of realizing the target behavior in the environment. Instead of not marking all states necessary for an exact solution, we can also extract information about which action must be performable in a certain configuration to realize the target behaviour. This information can be used to extend the available behaviours efficiently such that they are capable of realizing a desired target behaviour. Approximations can also be used to provide faster solutions at the expense of incompleteness. To compare the closeness of some approximations to an exact solution we can calculate the expected number of system configurations reached by simulating an arbitrary trace of the target behavior with a certain maximum length in which we cannot perform the next action of the trace. To do this we have to weight the edges of the target behavior with probabilities for the respective actions. Due to space restrictions we do not expand on this here but leave it to future work.

3 Conclusions

We have provided a sound and complete algorithm for solving the behavior composition problem. While there are existing algorithms in the literature, our algorithm provides a number of fundamental advantages. Firstly, our algorithm works in the way of a forward search (progression) in contrast to the proposal of [Sardina *et al.*, 2008] which is based on regression. Also, it does not require abstract implementations like that of [de Giacomo and Sardina, 2007] in terms of propositional dynamic logic which itself is difficult to implement.[1] Secondly, as a result of its simplicity, we implemented our algorithm in Java and have tested it on a number of representative problems. It is the only existing implementation of an algorithm for the behavior composition problem that we know of. While our algorithm is EXPTIME-complete like other existing approaches, the method we have used allows us to easily define the notion of an approximate solution and provide a metric for determining how close an approximation is to the target behavior. Approximations can be used when the target behavior is not able to be realized by the available behaviors in the given environment and also to determine how far the solution falls short of the target. Furthermore, if we wish to improve on the EXPTIME-complete worst-case time complexity of our algorithm, we could adopt approximations at the expense of incomplete solutions. Finally, our algorithm computes all possible schedulers capable of realizing the target behavior (as do the proposals in [Berardi *et al.*, 2008; Sardina *et al.*, 2008]).

In future work we will more fully develop the notion of approximation. One interesting avenue for research would be to develop an anytime version of our algorithm so that approximations can be developed and incrementally refined. This should not require a large modification to our existing algorithm.

References

[Berardi *et al.*, 2006a] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. Automatic web service composition: Service-tailored vs. client-tailored approaches. In *Proc. of AISC-06*, 2006.

[Berardi *et al.*, 2006b] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. Composing web services with nondeterministic behavior. In *Proc. of ICWS-06*, 2006.

[Berardi *et al.*, 2008] D. Berardi, F. Cheikh, G. de Giacomo, and F. Patrizi. Automatic service composition via simulation. *International Journal of Foundations of Computer Science*, 19(2):429–451, 2008.

[Calvanese *et al.*, 2008] D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella, and F. Patrizi. Automatic service composition and synthesis: The Roman model. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 31(3):18–22, 2008.

[de Giacomo and Sardina, 2007] G. de Giacomo and S. Sardina. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. of IJCAI-07*, pages 1866–1871, 2007.

[Sardina and de Giacomo, 2007] S. Sardina and G. de Giacomo. Automatic synthesis of a global behavior from multiple distributed behaviors. In *Proc. of AAAI-07*, pages 1063–1069, 2007.

[Sardina and de Giacomo, 2008] S. Sardina and G. de Giacomo. Realizing multiple autonomous agents through scheduling of shared devices. In *Proc. of ICAPS-08*, pages 304–311, 2008.

[Sardina *et al.*, 2008] S. Sardina, F. Patrizi, and G. de Giacomo. Behavior composition in the presence of failure. In *Proc. of KR-08*, pages 640–650, 2008.

[1] In fact, this is where our research started. In attempting to implement the algorithm of [de Giacomo and Sardina, 2007] we realized that there are no suitable PDL provers that would make this a straightforward task.