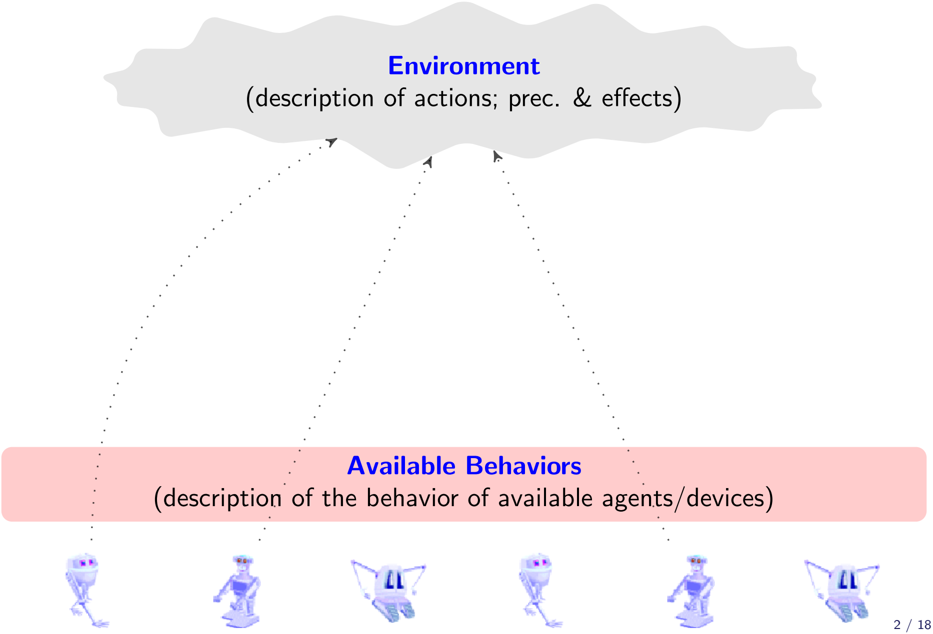
Composition of ConGolog Programs

Sebastian Sardina[1] Giuseppe De Giacomo[2]

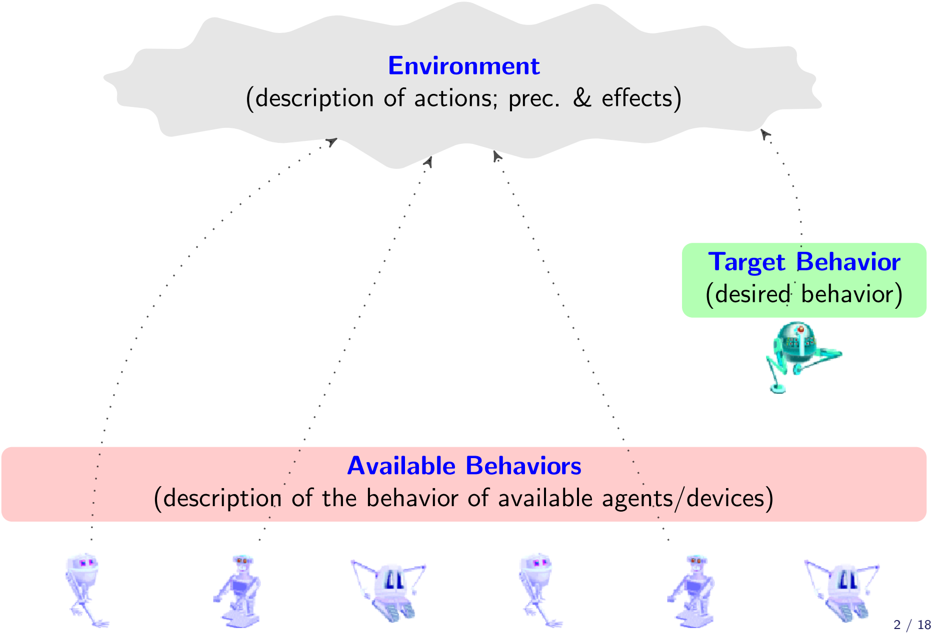
[1]Department of Computer Science and Information Technology
RMIT University, Melbourne, AUSTRALIA

[2]Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Sapienza Universita' di Roma, Rome, ITALY

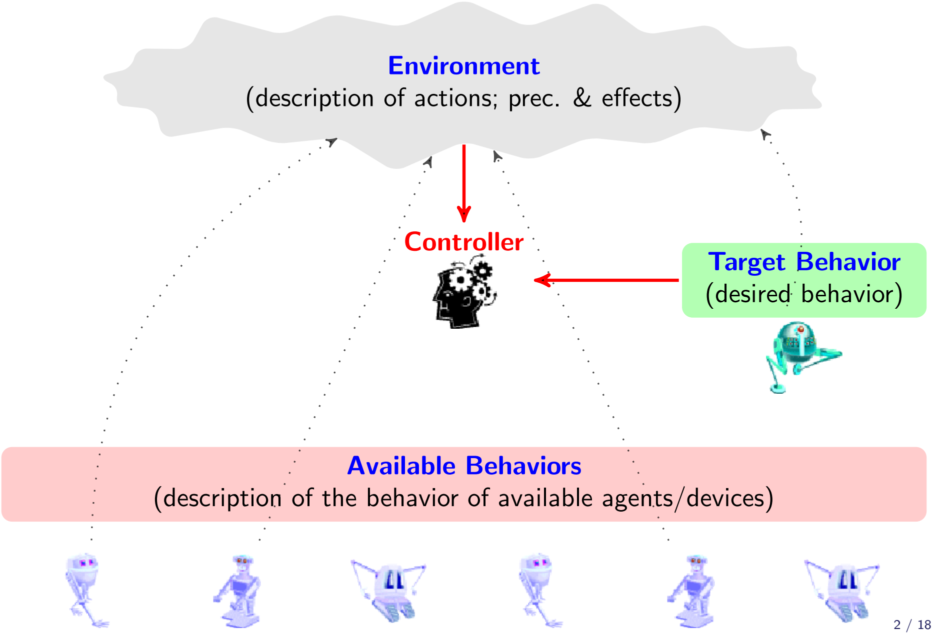
Behavior Composition: The Basic Idea ...



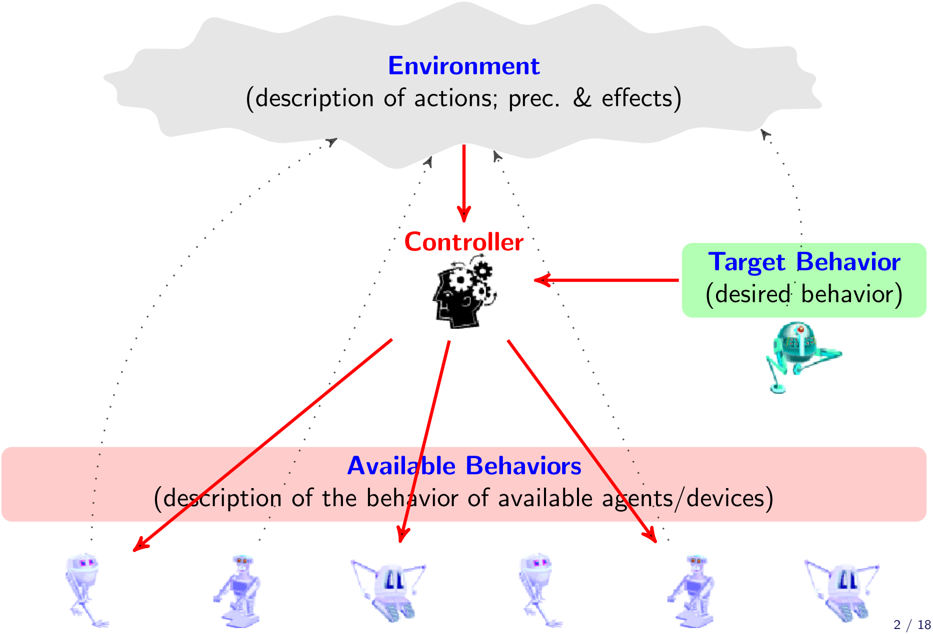
Behavior Composition: The Basic Idea ...



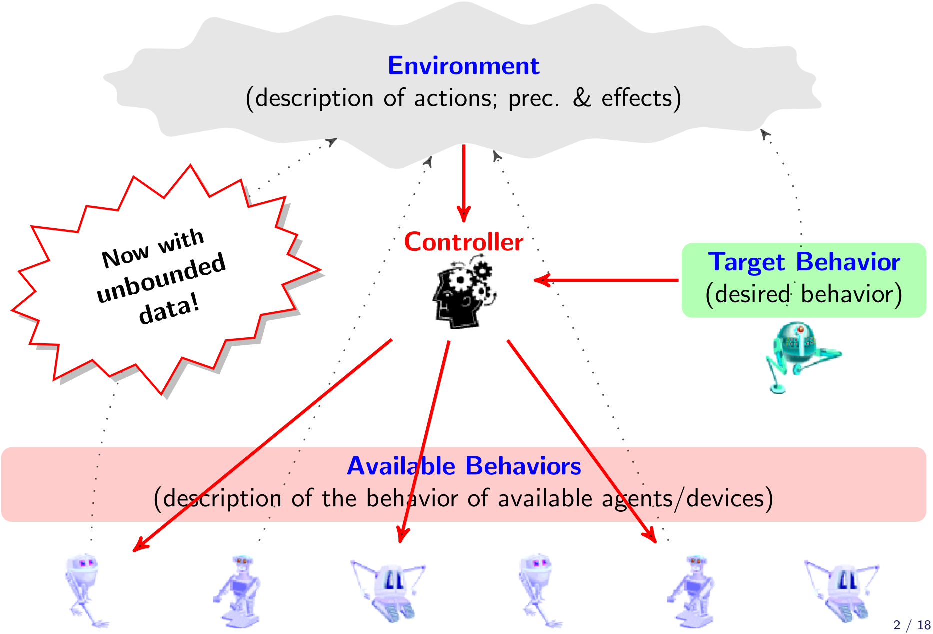
Behavior Composition: The Basic Idea ...



Behavior Composition: The Basic Idea ...



Behavior Composition: The Basic Idea ...



The ConGolog Composition Problem

Given:

1. An action theory $\mathcal{D}$;
2. n available programs $\delta_1, \ldots, \delta_n$;
3. a target program δ_t.

The ConGolog Composition Problem

Given:

1. An action theory $\mathcal{D}$;
2. *n* available programs $\delta_1, \ldots, \delta_n$;
3. a target program δ_t.

Task: find an orchestrator/delegator that coordinates the concurrent execution of the available programs so as to mimic/realize the target program.

The ConGolog Composition Problem

Given:

1. An action theory $\mathcal{D}$;
2. n available programs $\delta_1, \ldots, \delta_n$;
3. a target program δ_t.

Task: find an orchestrator/delegator that coordinates the concurrent execution of the available programs so as to mimic/realize the target program.

agent/plan coordination, virtual agents;
web-service composition; composition of business processes

The ConGolog Composition Problem

Given:

1. An action theory $\mathcal{D}$;
2. n available programs $\delta_1, \ldots, \delta_n$;
3. a target program δ_t.

Task: find an orchestrator/delegator that coordinates the concurrent execution of the available programs so as to mimic/realize the target program.

agent/plan coordination, virtual agents;
web-service composition; composition of business processes

Notable features:

- Programs may include *non-deterministic* points & may *not terminate*.
- Domain may be *infinite*.
- Programs may go over *infinite states*.

Controller for a Music Jukebox

$$
\begin{array}{l}
\textbf{while } \texttt{True} \textbf{ do } \{ \\
\quad \textbf{if } (\neg \mathit{Playing} \wedge (\exists \mathit{song}) \mathit{Pending}(\mathit{song})) \textbf{ then} \\
\quad\; (\pi\ \mathit{song}, \mathit{disk}).\{ \\
\qquad (\mathit{Pending}(\mathit{song}) \wedge \mathit{InDisk}(\mathit{song}, \mathit{disk}))?; \\
\qquad \mathit{select}(\mathit{song}); \\
\qquad \mathit{load}(\mathit{disk}); \\
\qquad \mathit{play}(\mathit{song}) \\
\quad \} \\
\quad \textbf{else } \mathit{wait} \\
\}
\end{array}
$$

Controller for a Music Jukebox

$$
\begin{array}{l}
\textbf{while } \texttt{True} \textbf{ do } \{ \\
\quad \textbf{if } (\neg \mathit{Playing} \land (\exists \mathit{song})\mathit{Pending}(\mathit{song})) \textbf{ then} \\
\quad\; (\pi\ \mathit{song}, \mathit{disk}).\{ \\
\quad\quad (\mathit{Pending}(\mathit{song}) \land \mathit{InDisk}(\mathit{song}, \mathit{disk}))?; \\
\quad\quad \mathit{select}(\mathit{song}); \\
\quad\quad \mathit{load}(\mathit{disk}); \\
\quad\quad \mathit{play}(\mathit{song}) \\
\quad \} \\
\quad \textbf{else } \mathit{wait} \\
\}
\end{array}
$$

- Domain tests relative to an action theory.

Controller for a Music Jukebox

$$
\begin{array}{l}
\textbf{while } \texttt{True} \textbf{ do } \{ \\
\quad \textbf{if } (\neg \mathit{Playing} \land (\exists \mathit{song}) \mathit{Pending}(\mathit{song})) \textbf{ then} \\
\quad\; (\pi\ \mathit{song}, \mathit{disk}).\{ \\
\qquad (\mathit{Pending}(\mathit{song}) \land \mathit{InDisk}(\mathit{song}, \mathit{disk}))?; \\
\qquad \mathit{select}(\mathit{song}); \\
\qquad \mathit{load}(\mathit{disk}); \\
\qquad \mathit{play}(\mathit{song}) \\
\quad \} \\
\quad \textbf{else } \mathit{wait} \\
\}
\end{array}
$$

- Domain tests relative to an action theory.
- Domain actions.

Controller for a Music Jukebox

$$
\begin{array}{l}
\textbf{while } \texttt{True} \textbf{ do } \{ \\
\quad \textbf{if } (\neg Playing \wedge (\exists song) Pending(song)) \textbf{ then} \\
\quad\quad (\pi\ song, disk).\{ \\
\quad\quad\quad (Pending(song) \wedge InDisk(song, disk))?; \\
\quad\quad\quad select(song); \\
\quad\quad\quad load(disk); \\
\quad\quad\quad play(song) \\
\quad \} \\
\quad \textbf{else } wait \\
\}
\end{array}
$$

- Domain tests relative to an action theory.
- Domain actions.
- Nondeterministic features.

Semantics for High-Level Programs

In terms of two predicates:

1. $Trans(\delta, s, \delta', s')$: program δ can *evolve one step* from situation s to situation s' with remaining program δ'.

$$
\begin{aligned}
&Trans(\delta_1;\delta_2, s, \delta', s') \equiv \\
&\qquad Trans(\delta_1, s, \delta_1', s') \land \delta' = \delta_1';\delta_2 \lor Final(\delta_1, s) \land Trans(\delta_2, s, \delta', s').
\end{aligned}
$$

2. $Final(\delta, s)$: program δ *may terminate* successfully in s.

$$Final(\delta_1;\delta_2, s) \equiv Final(\delta_1, s) \land Final(\delta_2, s)$$

Formalizing the Composition Problem: Simulation

Informally:

System S simulates system T if S can "match" all T's moves, forever.

Formalizing the Composition Problem: Simulation

Informally:

System S simulates system T if S can "match" all T's moves, forever.

Formally [Milner IJCAI'71]:

Given two labelled TSs $S = (\Sigma_S, A_S, \longrightarrow_S)$ and $T = (\Sigma_T, A_T, \longrightarrow_T)$, the simulation is the largest relation $Sim \subseteq \Sigma_S \times \Sigma_T$ such that:

if $Sim(s, t)$ holds (state s simulates state t), then:

if $t \xrightarrow{\alpha}_T t'$, then

there exists $s \xrightarrow{\alpha}_S s'$ and $Sim(s', t')$.

The Composition Problem: Simulation

$Sim(\delta_t, \delta_1, \ldots, \delta_n, s)$: available programs can *simulate* the target program in s.

$$Sim(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \\ \exists S.\big(S(\delta_t, \delta_1, \ldots, \delta_n, s) \land \forall \delta_t, \delta_1, \ldots, \delta_n, s.\Theta[S](\delta_t, \delta_1, \ldots, \delta_n, s)\big),$$

where

$$\Theta[S](\delta_t, \delta_1, \ldots, \delta_n, s) \stackrel{\text{def}}{=} \\ S(\delta_t, \delta_1, \ldots, \delta_n, s) \rightarrow \\ \big(Final(\delta_t, s) \rightarrow \bigwedge_{i=1,\ldots,n} Final(\delta_i, s)\big) \;\; \land \\ \big(\forall \delta'_t, s'\, Trans(\delta_t, s, \delta'_t, s') \rightarrow \\ \bigvee_{i=1,\ldots,n} \exists \delta'_i. Trans(\delta_i, s, \delta'_i, s') \land S(\delta'_t, \delta_1, \ldots, \delta'_i, \ldots, \delta_n, s')\big).$$

If the simulation holds then one can build an orchestrator generator based on it.

The Composition Problem: Simulation

$\mathit{Sim}(\delta_t, \delta_1, \ldots, \delta_n, s)$: available programs can *simulate* the target program in s.

$$
\begin{array}{l}
\mathit{Sim}(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \\
\qquad \exists S.\big(S(\delta_t, \delta_1, \ldots, \delta_n, s) \land \forall \delta_t, \delta_1, \ldots, \delta_n, s.\Theta[S](\delta_t, \delta_1, \ldots, \delta_n, s)\big),
\end{array}
$$

where

$$
\begin{array}{l}
\Theta[S](\delta_t, \delta_1, \ldots, \delta_n, s) \stackrel{\text{def}}{=} \\
\quad S(\delta_t, \delta_1, \ldots, \delta_n, s) \rightarrow \\
\qquad \big(\mathit{Final}(\delta_t, s) \rightarrow \bigwedge_{i=1,\ldots,n} \mathit{Final}(\delta_i, s)\big) \;\land \\
\qquad \big(\forall \delta_t', s'\, \mathit{Trans}(\delta_t, s, \delta_t', s') \rightarrow \\
\qquad\qquad \bigvee_{i=1,\ldots,n} \exists \delta_i'.\mathit{Trans}(\delta_i, s, \delta_i', s') \land S(\delta_t', \delta_1, \ldots, \delta_i', \ldots, \delta_n, s')\big).
\end{array}
$$

Second order!

If the simulation holds then one can build an orchestrator generator based on it.

The Technique

Relies on the following "tools"/notions:

1. Simulation approximates: [Tarski '55]
   - Check simulation in a finite way.
2. Regression mechanism: [Reiter'91; Pirri & Reiter'99]
   - Reason on formulas after action performance.
3. Characteristic graphs: [Classen&Lakemeyer KR'08]
   - Abstract (infinite) program states into a finite graph.

The Technique

Relies on the following "tools"/notions:

1. Simulation approximates: [Tarski '55]
   - Check simulation in a finite way.
2. Regression mechanism: [Reiter'91; Pirri & Reiter'99]
   - Reason on formulas after action performance.
3. Characteristic graphs: [Classen&Lakemeyer KR'08]
   - Abstract (infinite) program states into a finite graph.

Simulation Approximates

$Sim_k(\delta_t, \delta_1, \ldots, \delta_n, s)$:
the available programs can "simulate" k steps of the target program in s.

$$Sim_0(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv (Final(\delta_t, s) \rightarrow \bigwedge_{i=1,\ldots,n} Final(\delta_i, s)).$$

$$\begin{array}{l} Sim_{k+1}(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \\ \quad Sim_k(\delta_t, \delta_1, \ldots, \delta_n, s) \;\wedge \\ \quad \big(\forall \delta_t', s'. Trans(\delta_t, s, \delta_t', s') \rightarrow \\ \qquad \bigvee_{i=1,\ldots,n} \exists \delta_i'. Trans(\delta_i, s, \delta_i', s') \wedge Sim_k(\delta_t', \delta_1, \ldots, \delta_i', \ldots, \delta_n, s')\big). \end{array}$$

Simulation Approximates

$Sim_k(\delta_t, \delta_1, \ldots, \delta_n, s)$:
the available programs can "simulate" k steps of the target program in s.

$$Sim_0(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv (Final(\delta_t, s) \rightarrow \bigwedge_{i=1,\ldots,n} Final(\delta_i, s)).$$

$$\begin{aligned} & Sim_{k+1}(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \\ & \quad Sim_k(\delta_t, \delta_1, \ldots, \delta_n, s) \ \wedge \\ & \quad \big(\forall \delta_t', s'. Trans(\delta_t, s, \delta_t', s') \rightarrow \\ & \qquad \bigvee_{i=1,\ldots,n} \exists \delta_i'. Trans(\delta_i, s, \delta_i', s') \wedge Sim_k(\delta_t', \delta_1, \ldots, \delta_i', \ldots, \delta_n, s')\big). \end{aligned}$$

Proposition

For every $k \geq 0$, if

$$Sim_k(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv Sim_{k+1}(\delta_t, \delta_1, \ldots, \delta_n, s),$$

then

$$Sim_k(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv Sim(\delta_t, \delta_1, \ldots, \delta_n, s).$$

The Technique

Relies on the following "tools"/notions:

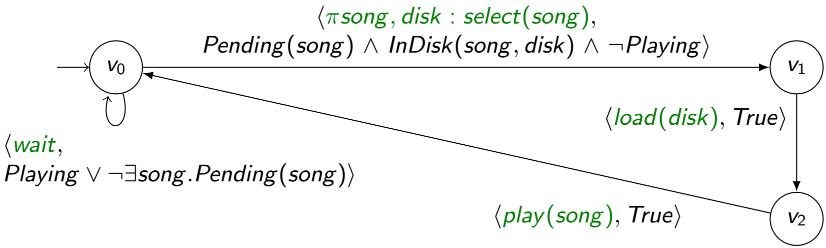
1. Simulation approximates: [Tarski '55]
   - Check simulation in a finite way.
2. Regression mechanism: [Reiter'91; Pirri & Reiter'99]
   - Reason on formulas after action performance.
   - computes what has to be true in situation s so that ϕ is true after doing action α in s.
   - $\mathcal{R}[\phi(do(\alpha, s))] = \phi'(s)$ action α has been eliminated!
3. Characteristic graphs: [Classen&Lakemeyer KR'08]
   - Abstract (infinite) program states into a finite graph.

The Technique

Relies on the following "tools"/notions:

1. Simulation approximates: [Tarski '55]
   - Check simulation in a finite way.
2. Regression mechanism: [Reiter'91; Pirri & Reiter'99]
   - Reason on formulas after action performance.
3. Characteristic graphs: [Classen&Lakemeyer KR'08]
   - Abstract (infinite) program states into a finite graph.

Characteristic Graph for δ_{music}



$\delta_{music} \doteq$
while *True* **do** {
  if $(\neg Playing \wedge (\exists song) Pending(song))$ **then**
    $\pi song, disk.\{$
      $(Pending(song) \wedge InDisk(song, disk))?;$
      $select(song);$
      $load(disk);$
      $play(song)$
  }
  else *wait*
}

$v_0 = \langle \delta_{music}, \; False \rangle$

$v_1 = \langle load(disk); play(song), \; False \rangle$

$v_2 = \langle play(song), \; False \rangle$

The Composition Problem: Simulation

$Sim(\delta_t, \delta_1, \ldots, \delta_n, s)$: available programs can *simulate* the target program in s.

$$
\begin{aligned}
&Sim(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \\
&\qquad \exists S. \big(S(\delta_t, \delta_1, \ldots, \delta_n, s) \land \forall \delta_t, \delta_1, \ldots, \delta_n, s. \Theta[S](\delta_t, \delta_1, \ldots, \delta_n, s) \big),
\end{aligned}
$$

where

$$
\begin{aligned}
&\Theta[S](\delta_t, \delta_1, \ldots, \delta_n, s) \stackrel{\text{def}}{=} \\
&\quad S(\delta_t, \delta_1, \ldots, \delta_n, s) \rightarrow \\
&\qquad \big(Final(\delta_t, s) \rightarrow \textstyle\bigwedge_{i=1,\ldots,n} Final(\delta_i, s) \big) \ \land \\
&\qquad \big(\forall \delta'_t, s' \, Trans(\delta_t, s, \delta'_t, s') \rightarrow \\
&\qquad\qquad \textstyle\bigvee_{i=1,\ldots,n} \exists \delta'_i. Trans(\delta_i, s, \delta'_i, s') \land S(\delta'_t, \delta_1, \ldots, \delta'_i, \ldots, \delta_n, s') \big).
\end{aligned}
$$

If the simulation holds then one can build an orchestrator generator based on it.

Algorithm $\textsc{SymSim}(\delta_t^0, \delta_1^0, \ldots, \delta_n^0)$

Computes relation X containing tuples of the form $\langle v_t, v_1, \ldots, v_n, \phi \rangle$:

- $v_t, v_1, \ldots, v_n$ are nodes in the characteristic graphs.
- FO formula ϕ: *"the target program in v_t is simulated by the available programs in $\langle v_1, \ldots, v_n \rangle$."*

Algorithm $\textsc{SymSim}(\delta_t^0, \delta_1^0, \ldots, \delta_n^0)$

Computes relation X containing tuples of the form $\langle v_t, v_1, \ldots, v_n, \phi \rangle$:

- $v_t, v_1, \ldots, v_n$ are nodes in the characteristic graphs.
- FO formula ϕ: *"the target program in v_t is simulated by the available programs in $\langle v_1, \ldots, v_n \rangle$."*

1. $X_0 := \{\langle (\delta_t, \gamma_t), (\delta_1, \gamma_1), \ldots, (\delta_n, \gamma_n), \gamma_t \rightarrow \bigwedge_{i=1}^n \gamma_i \rangle \mid (\delta_j, \gamma_j) \text{ in } \mathcal{G}_{\delta_j^0}\}$
   - 0-step simulation: check for termination "mimicking."

Algorithm $\textsc{SymSim}(\delta_t^0, \delta_1^0, \ldots, \delta_n^0)$

Computes relation X containing tuples of the form $\langle v_t, v_1, \ldots, v_n, \phi \rangle$:

- $v_t, v_1, \ldots, v_n$ are nodes in the characteristic graphs.
- FO formula ϕ: *"the target program in v_t is simulated by the available programs in $\langle v_1, \ldots, v_n \rangle$."*

1. $X_0 := \{\langle (\delta_t, \gamma_t), (\delta_1, \gamma_1), \ldots, (\delta_n, \gamma_n), \gamma_t \rightarrow \bigwedge_{i=1}^{n} \gamma_i \rangle \mid (\delta_j, \gamma_j) \text{ in } \mathcal{G}_{\delta_j^0}\}$
   - 0-step simulation: check for termination "mimicking."
2. At every step, compute $\textsc{Next}[X]$: "one step refinement" of the simulation:

$$\textsc{Next}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new} \rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old} \rangle \in X\}.$$

   - ϕ_{new}: we can safely mimic (any) single action from the target.

Algorithm $\textsc{SymSim}(\delta_t^0, \delta_1^0, \ldots, \delta_n^0)$

Computes relation X containing tuples of the form $\langle v_t, v_1, \ldots, v_n, \phi\rangle$:

- $v_t, v_1, \ldots, v_n$ are nodes in the characteristic graphs.
- FO formula ϕ: *"the target program in v_t is simulated by the available programs in $\langle v_1, \ldots, v_n\rangle$."*

1. $X_0 := \{\langle(\delta_t, \gamma_t), (\delta_1, \gamma_1), \ldots, (\delta_n, \gamma_n), \gamma_t \rightarrow \bigwedge_{i=1}^{n} \gamma_i\rangle \mid (\delta_j, \gamma_j) \text{ in } \mathcal{G}_{\delta_j^0}\}$
   - 0-step simulation: check for termination "mimicking."
2. At every step, compute $\textsc{Next}[X]$: "one step refinement" of the simulation:

$$\textsc{Next}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new}\rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old}\rangle \in X\}.$$

   - ϕ_{new}: we can safely mimic (any) single action from the target.
3. X represents the *approximates* of the simulation, refined at each iteration.

Algorithm $\textsc{SymSim}(\delta_t^0, \delta_1^0, \ldots, \delta_n^0)$

Computes relation X containing tuples of the form $\langle v_t, v_1, \ldots, v_n, \phi \rangle$:

- $v_t, v_1, \ldots, v_n$ are nodes in the characteristic graphs.
- FO formula ϕ: *"the target program in v_t is simulated by the available programs in $\langle v_1, \ldots, v_n \rangle$."*

1. $X_0 := \{\langle (\delta_t, \gamma_t), (\delta_1, \gamma_1), \ldots, (\delta_n, \gamma_n), \gamma_t \rightarrow \bigwedge_{i=1}^{n} \gamma_i \rangle \mid (\delta_j, \gamma_j) \text{ in } \mathcal{G}_{\delta_j^0}\}$
   - 0-step simulation: check for termination "mimicking."
2. At every step, compute $\textsc{Next}[X]$: "one step refinement" of the simulation:

$$\textsc{Next}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new} \rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old} \rangle \in X\}.$$

   - ϕ_{new}: we can safely mimic (any) single action from the target.
3. X represents the *approximates* of the simulation, refined at each iteration.
4. Stop when $X = \textsc{Next}[X]$.

$\textsc{Next}[X]$: One-step refinement of the simulation

$$\textsc{Next}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new}\rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old}\rangle \in X\},$$

$$\phi_{new} = \bigwedge_{v_t \xrightarrow[\psi_t]{\pi\vec{x}\ \alpha_t} v_t' \in E_t} \Big(\forall \vec{x}.\psi_t[s] \wedge Poss(\alpha_t, s) \rightarrow \bigvee_{i=1}^{n} \bigvee_{v_i \xrightarrow[\psi_i]{\pi\vec{y}.\alpha_i} v_i' \in E_i \wedge \langle v_t', v_1, \ldots, v_i', \ldots, v_n, \phi_i\rangle \in X} \exists \vec{y}.\alpha_t = \alpha_i \wedge \psi_i[s] \wedge \mathcal{R}[\phi_i(do(\alpha_i, s))]\Big).$$

For every potential target evolution from v_t to v_t' via action α_t, ...

if it can be done in the program (ψ_t holds) and the action α_t is possible, ...

then some available prog. δ_i can evolve from v_i to v_i' via action α_i such that:

1. the action α_i can be matched to α_t;
2. the program can indeed do the step;
3. after doing the step, we are still in simulation.

$\textsc{Next}[X]$: One-step refinement of the simulation

$$\textsc{Next}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new}\rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old}\rangle \in X\},$$

$$\phi_{new} = \bigwedge_{v_t \xrightarrow[\psi_t]{\pi\vec{x}\ \alpha_t} v_t' \in E_t} \Big(\forall \vec{x}.\psi_t[s] \wedge Poss(\alpha_t, s) \rightarrow \bigvee_{i=1}^{n} \bigvee_{v_i \xrightarrow[\psi_i]{\pi\vec{y}.\alpha_i} v_i' \in E_i \wedge \langle v_t', v_1, \ldots, v_i', \ldots, v_n, \phi_i\rangle \in X} \exists \vec{y}.\alpha_t = \alpha_i \wedge \psi_i[s] \wedge \mathcal{R}[\phi_i(do(\alpha_i, s))]\Big).$$

For every potential target evolution from v_t to v_t' via action α_t, ...

if it can be done in the program (ψ_t holds) and the action α_t is possible, ...

then some available prog. δ_i can evolve from v_i to v_i' via action α_i such that:

1. the action α_i can be matched to α_t;
2. the program can indeed do the step;
3. after doing the step, we are still in simulation.

$\textsc{Next}[X]$: One-step refinement of the simulation

$$\textsc{Next}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new}\rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old}\rangle \in X\},$$

$$\phi_{new} = \bigwedge_{v_t \xrightarrow[\psi_t]{\pi\vec{x}\ \alpha_t} v'_t \in E_t} \Big(\forall\vec{x}.\psi_t[s] \wedge Poss(\alpha_t, s) \rightarrow \bigvee_{i=1}^{n} \bigvee_{v_i \xrightarrow[\psi_i]{\pi\vec{y}.\alpha_i} v'_i \in E_i \wedge \langle v'_t, v_1, \ldots, v'_i, \ldots, v_n, \phi_i\rangle \in X} \exists\vec{y}.\alpha_t = \alpha_i \wedge \psi_i[s] \wedge \mathcal{R}[\phi_i(do(\alpha_i, s))]\Big).$$

For every potential target evolution from v_t to v'_t via action α_t, ...

if it can be done in the program (ψ_t holds) and the action α_t is possible, ...

then some available prog. δ_i can evolve from v_i to v'_i via action α_i such that:

1. the action α_i can be matched to α_t;
2. the program can indeed do the step;
3. after doing the step, we are still in simulation.

$\text{NEXT}[X]$: One-step refinement of the simulation

$$\text{NEXT}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new} \rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old} \rangle \in X\},$$

$$\phi_{new} = \bigwedge_{v_t \xrightarrow[\psi_t]{\pi\vec{x}\ \alpha_t} v'_t \in E_t} \Big(\forall \vec{x}.\psi_t[s] \wedge Poss(\alpha_t, s) \rightarrow \bigvee_{i=1}^{n} \bigvee_{v_i \xrightarrow[\psi_i]{\pi\vec{y}.\alpha_i} v'_i \in E_i \wedge \langle v'_t, v_1, \ldots, v'_i, \ldots, v_n, \phi_i \rangle \in X} \exists \vec{y}.\alpha_t = \alpha_i \wedge \psi_i[s] \wedge \mathcal{R}[\phi_i(do(\alpha_i, s))] \Big).$$

For every potential target evolution from v_t to v'_t via action α_t, ...

if it can be done in the program (ψ_t holds) and the action α_t is possible, ...

then some available prog. δ_i can evolve from v_i to v'_i via action α_i such that:

1. the action α_i can be matched to α_t;
2. the program can indeed do the step;
3. after doing the step, we are still in simulation.

$\textsc{Next}[X]$: One-step refinement of the simulation

$$\textsc{Next}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new}\rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old}\rangle \in X\},$$

$$\phi_{new} = \bigwedge_{v_t \xrightarrow[\psi_t]{\pi\vec{x}\ \alpha_t} v_t' \in E_t} \Big(\forall\vec{x}.\psi_t[s] \wedge Poss(\alpha_t, s) \rightarrow \bigvee_{i=1}^{n} \bigvee_{v_i \xrightarrow[\psi_i]{\pi\vec{y}.\alpha_i} v_i' \in E_i \wedge \langle v_t', v_1, \ldots, v_i', \ldots, v_n, \phi_i\rangle \in X} \exists\vec{y}.\alpha_t = \alpha_i \wedge \psi_i[s] \wedge \mathcal{R}[\phi_i(do(\alpha_i, s))]\Big).$$

For every potential target evolution from v_t to v_t' via action α_t, ...

if it can be done in the program (ψ_t holds) and the action α_t is possible, ...

then some available prog. δ_i can evolve from v_i to v_i' via action α_i such that:

1. the action α_i can be matched to α_t;
2. the program can indeed do the step;
3. after doing the step, we are still in simulation.

NEXT[X]: One-step refinement of the simulation

$$\text{NEXT}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new}\rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old}\rangle \in X\},$$

$$\phi_{new} = \bigwedge_{v_t \xrightarrow[\psi_t]{\pi\vec{x}\ \alpha_t} v_t' \in E_t} \Big(\forall \vec{x}.\psi_t[s] \wedge Poss(\alpha_t, s) \rightarrow \bigvee_{i=1}^{n} \bigvee_{v_i \xrightarrow[\psi_i]{\pi\vec{y}.\alpha_i} v_i' \in E_i \wedge \langle v_t', v_1, \ldots, v_i', \ldots, v_n, \phi_i\rangle \in X} \exists \vec{y}.\alpha_t = \alpha_i \wedge \psi_i[s] \wedge \mathcal{R}[\phi_i(do(\alpha_i, s))]\Big).$$

For every potential target evolution from v_t to v_t' via action α_t, ...

if it can be done in the program (ψ_t holds) and the action α_t is possible, ...

then some available prog. δ_i can evolve from v_i to v_i' via action α_i such that:

1. the action α_i can be matched to α_t;
2. the program can indeed do the step;
3. after doing the step, we are still in simulation.

$\textsc{Next}[X]$: One-step refinement of the simulation

$$\textsc{Next}[X] = \{\langle v_t, v_1, \ldots, v_n, \phi_{old} \wedge \phi_{new}\rangle \mid \langle v_t, v_1, \ldots, v_n, \phi_{old}\rangle \in X\},$$

$$\phi_{new} = \bigwedge_{v_t \xrightarrow[\psi_t]{\pi\vec{x}\ \alpha_t} v'_t \in E_t} \Big(\forall \vec{x}.\psi_t[s] \wedge Poss(\alpha_t, s) \rightarrow \bigvee_{i=1}^{n} \bigvee_{v_i \xrightarrow[\psi_i]{\pi\vec{y}.\alpha_i} v'_i \in E_i \wedge \langle v'_t, v_1, \ldots, v'_i, \ldots, v_n, \phi_i\rangle \in X} \exists \vec{y}.\alpha_t = \alpha_i \wedge \psi_i[s] \wedge \mathcal{R}[\phi_i(do(\alpha_i, s))]\Big).$$

For every potential target evolution from v_t to v'_t via action α_t, ...

if it can be done in the program (ψ_t holds) and the action α_t is possible, ...

then some available prog. δ_i can evolve from v_i to v'_i via action α_i such that:

1. the action α_i can be matched to α_t;
2. the program can indeed do the step;
3. after doing the step, we are still in simulation.

Technical Results

Theorem

If algorithm $\textsc{SymSim}(\delta_t^0, \delta_1^0, \ldots, \delta_n^0)$ *terminates returning the set* X*. Then,*

$$Axioms \models Sim(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \phi[s],$$

where $\langle(\delta_t, \gamma_t), (\delta_1, \gamma_1), \ldots, (\delta_n, \gamma_n), \phi\rangle \in X$*.*

Technical Results

Theorem

If algorithm $\textsc{SymSim}(\delta_t^0, \delta_1^0, \ldots, \delta_n^0)$ *terminates returning the set* X*. Then,*

$$Axioms \models Sim(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \phi[s],$$

where $\langle(\delta_t, \gamma_t), (\delta_1, \gamma_1), \ldots, (\delta_n, \gamma_n), \phi\rangle \in X$*.*

Moreover, we can construct a delegator controller to realize the composition *on-the-fly* using FO entailment only (on $\mathcal{D}_{S_0}$).

Idea:

after a request, jump to a configuration $\langle v_t, v_1, \ldots, v_n, \phi\rangle \in X$ for which ϕ holds.

Conclusions

Reasoning on unbounded data and processes is a challenge for CS since it leads *infinite* state systems.

- Standard approach: abstract to finite systems (see literature on Verification).
- Here instead we are proposing an alternative approach rooted in KR and AI.

Conclusions

Reasoning on unbounded data and processes is a challenge for CS since it leads *infinite* state systems.

- Standard approach: abstract to finite systems (see literature on Verification).
- Here instead we are proposing an alternative approach rooted in KR and AI.

Specifically:
Based on transforming the second-order formula for checking the dynamic property of simulation into a first-order one talking only about the static properties of the initial situation/DB.

Conclusions

Reasoning on unbounded data and processes is a challenge for CS since it leads *infinite* state systems.

- Standard approach: abstract to finite systems (see literature on Verification).
- Here instead we are proposing an alternative approach rooted in KR and AI.

Specifically:
Based on transforming the second-order formula for checking the dynamic property of simulation into a first-order one talking only about the static properties of the initial situation/DB.

Main research direction for future work:

1. incomplete information about the initial situation.
   - *offline* vs *online* interpreters (see literature on high-level programs in AI).
2. identify cases in which the technique becomes sound & complete.