Composition of ConGolog Programs

Sebastian Sardina\textsuperscript{1} \quad Giuseppe De Giacomo\textsuperscript{2}

\textsuperscript{1}Department of Computer Science and Information Technology
RMIT University, Melbourne, AUSTRALIA

\textsuperscript{2}Dipartimento di Informatica e Sistemistica “Antonio Ruberti”
Sapienza Universita’ di Roma, Rome, ITALY
Behavior Composition: The Basic Idea ...

**Environment**
(description of actions; prec. & effects)

**Available Behaviors**
(description of the behavior of available agents/devices)

**Available Devices**
(logic of existing devices; partially-controllable)

**Target Behavior**
(desired behavior)

**Controller**

- Broadcasting Channel

**Agents**

- Autonomous
- Deterministic
- Autonomous
- Nondeterministic
- Fully observable
- Partially controllable

**Scheduler**

**Robust Controller**

Now with unbounded data!
Behavior Composition: The Basic Idea ...

Environment
(description of actions; prec. & effects)

Target Behavior
(desired behavior)

Available Behaviors
(description of the behavior of available agents/devices)
Behavior Composition: The Basic Idea ...

Environment
(description of actions; prec. & effects)

Controller

Target Behavior
(desired behavior)

Available Behaviors
(description of the behavior of available agents/devices)
Behavior Composition: The Basic Idea ...

Environment
(description of actions; prec. & effects)

Controller

Target Behavior
(desired behavior)

Available Behaviors
(description of the behavior of available agents/devices)
Behavior Composition: The Basic Idea ...

Environment
(description of actions; prec. & effects)

Controller

Target Behavior
(desired behavior)

Available Behaviors
(description of the behavior of available agents/devices)

Now with unbounded data!
The ConGolog Composition Problem

Given:

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

Task:

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

Notable features:

- Programs may include non-deterministic points & may not terminate.
- Domain may be infinite.
- Programs may go over infinite states.
The ConGolog Composition Problem

Given:

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

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 $\delta_t$.

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 \( \delta_t \).

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

while True do {
  if (¬Playing ∧ (∃song)Pending(song)) then
    (π song, disk).{
      (Pending(song) ∧ InDisk(song, disk))?;
      select(song);
      load(disk);
      play(song)
    }
  else wait
}
Controller for a Music Jukebox

\[
\text{while True do } \{
\text{if } (\neg \text{Playing} \land (\exists \text{song}) \text{Pending(song)}) \text{ then }
\]
\[
(\pi \text{ song, disk}).\{
(\text{Pending(song)} \land \text{InDisk(song, disk)})?;
\text{select(song);}
\text{load(disk);}
\text{play(song)}
\}
\text{else wait}
\}
\]

- Domain tests relative to an action theory.
Controller for a Music Jukebox

```plaintext
while True do {
    if (¬Playing ∧ (∃song)Pending(song)) then
        (π song, disk).{
            (Pending(song) ∧ InDisk(song, disk))?
            select(song);
            load(disk);
            play(song)
        }
    else wait
}
```

- Domain tests relative to an action theory.
- Domain actions.
Controller for a Music Jukebox

```java
while True do {
    if (¬Playing ∧ (∃song)Pending(song)) then
        (π song, disk).{
            (Pending(song) ∧ InDisk(song, disk))?;
            select(song);
            load(disk);
            play(song)
        }
    else wait
}
```

- Domain tests relative to an action theory.
- Domain actions.
- Nondeterministic features.
Semantics for High-Level Programs

In terms of two predicates:

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

   \[
   \text{Trans}(\delta_1; \delta_2, s, \delta', s') \equiv \text{Trans}(\delta_1, s, \delta'_1, s') \land \delta' = \delta'_1; \delta_2 \lor \text{Final}(\delta_1, s) \land \text{Trans}(\delta_2, s, \delta', s').
   \]

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

   \[
   \text{Final}(\delta_1; \delta_2, s) \equiv \text{Final}(\delta_1, s) \land \text{Final}(\delta_2, s)
   \]
Informally:

*System S simulates system T if S can “match” all T’s moves, forever.*
Informally:

*System S simulates system T* if S can “match” all T’s moves, forever.

Formally [Milner IJCAI’71]:

*Given two labelled TSs* $S = (\Sigma_S, A_S, \rightarrow_S)$ *and* $T = (\Sigma_T, A_T, \rightarrow_T)$, *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* $t \xrightarrow{T} t'$, *then*

*there exists* $s \xrightarrow{S} s'$ *and* $Sim(s', t')$. 
The Composition Problem: Simulation

Sim(δ_t, δ_1, ..., δ_n, s): available programs can simulate the target program in s.

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

where

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

If the simulation holds then one can build an orchestrator generator based on it.
The Composition Problem: Simulation

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

\[
\text{Sim}(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \exists S. (S(\delta_t, \delta_1, \ldots, \delta_n, s) \land \forall \delta_t, \delta_1, \ldots, \delta_n, s. \Theta[S](\delta_t, \delta_1, \ldots, \delta_n, s)),
\]

where

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

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:**
   - Check simulation in a finite way. [Tarski ’55]

2. **Regression mechanism:**
   - Reason on formulas after action performance. [Reiter’91; Pirri & Reiter’99]

3. **Characteristic graphs:**
   - Abstract (infinite) program states into a finite graph. [Classen&Lakemeyer KR’08]
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**

\( \text{Sim}_k(\delta_t, \delta_1, \ldots, \delta_n, s): \)

the available programs can “simulate” \( k \) steps of the target program in \( s \).

\[
\text{Sim}_0(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv (\text{Final}(\delta_t, s) \rightarrow \bigwedge_{i=1,\ldots,n} \text{Final}(\delta_i, s)).
\]

\[
\text{Sim}_{k+1}(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \\
\text{Sim}_k(\delta_t, \delta_1, \ldots, \delta_n, s) \land \\
(\forall \delta'_t, s'. \text{Trans}(\delta_t, s, \delta'_t, s') \rightarrow \\
\bigvee_{i=1,\ldots,n} \exists \delta'_i. \text{Trans}(\delta_i, s, \delta'_i, s') \land \text{Sim}_k(\delta'_t, \delta_1, \ldots, \delta'_i, \ldots, \delta_n, s')).
\]
Simulation Approximates

\( \text{Sim}_k(\delta_t, \delta_1, \ldots, \delta_n, s) \): the available programs can “simulate” \( k \) steps of the target program in \( s \).

\[
\text{Sim}_0(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv (\text{Final}(\delta_t, s) \rightarrow \bigwedge_{i=1,\ldots,n} \text{Final}(\delta_i, s)).
\]

\[
\text{Sim}_{k+1}(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \text{Sim}_k(\delta_t, \delta_1, \ldots, \delta_n, s) \land \\
(\forall \delta'_t, s'. \text{Trans}(\delta_t, s, \delta'_t, s') \rightarrow \\
\bigvee_{i=1,\ldots,n} \exists \delta'_i. \text{Trans}(\delta_i, s, \delta'_i, s') \land \text{Sim}_k(\delta'_t, \delta_1, \ldots, \delta'_i, \ldots, \delta_n, s')).
\]

Proposition

For every \( k \geq 0 \), if

\[
\text{Sim}_k(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \text{Sim}_{k+1}(\delta_t, \delta_1, \ldots, \delta_n, s),
\]

then

\[
\text{Sim}_k(\delta_t, \delta_1, \ldots, \delta_n, s) \equiv \text{Sim}(\delta_t, \delta_1, \ldots, \delta_n, s).
\]
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 $\phi$ is true after doing action $\alpha$ in $s$.
   - $\mathcal{R}[\phi(do(\alpha, s))] = \phi'(s)$ action $\alpha$ 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 $\delta_{\text{music}}$

$$
\langle \pi \text{song} : \text{select(song)}, \nonumber \\
\text{Pending(song)} \land \text{InDisk(song, disk)} \land \neg \text{Playing} \rangle
$$

$v_0 \rightarrow v_1$

$$
\langle \text{wait}, \nonumber \\
\text{Playing} \lor \neg \exists \text{song}.\text{Pending(song)} \rangle
$$

$v_1 \rightarrow v_2$

$$
\langle \text{load(disk)}, \text{True} \rangle
$$

$$
\langle \text{play(song)}, \text{True} \rangle
$$

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

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

$v_2 = \langle \text{play(song)}, \text{False} \rangle$

$\delta_{\text{music}} \triangleq$

while True do {
    if ($\neg \text{Playing} \land \exists \text{song.}\text{Pending(song)}$) then
        $\pi \text{song}, \text{disk.}$
        $(\text{Pending(song)} \land \text{InDisk(song, disk)})$?;
        select(song);
        load(disk);
        play(song)
    } else wait
}
The Composition Problem: Simulation

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

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

where

$\Theta[S](\delta_t, \delta_1, \ldots, \delta_n, s) \overset{\text{def}}{=} S(\delta_t, \delta_1, \ldots, \delta_n, s) \rightarrow$

$\left(Final(\delta_t, s) \rightarrow \bigwedge_{i=1,\ldots,n} Final(\delta_i, s)\right) \land$

$\left(\forall \delta'_t, s' \ Trans(\delta_t, s, \delta'_t, s') \rightarrowight.$

$\left.\bigvee_{i=1,\ldots,n} \exists \delta'_i. \ Trans(\delta_i, s, \delta'_i, s') \land S(\delta'_t, \delta_1, \ldots, \delta'_i, \ldots, \delta_n, s')\right)$.

If the simulation holds then one can build an orchestrator generator based on it.
Algorithm $\text{SYM SIM}(\delta^0_t, \delta^0_1, \ldots, \delta^0_n)$

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 $\phi$: “the target program in $v_t$ is simulated by the available programs in $\langle v_1, \ldots, v_n \rangle$.”
Algorithm \textsc{SymSim}(\delta^0_t, \delta^0_1, \ldots, \delta^0_n)

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 \(\phi\): “the target program in \(v_t\) is simulated by the available programs in \(\langle v_1, \ldots, v_n \rangle\).”

1. \(X_0 := \{ \langle (\delta_t, \gamma_t), (\delta_1, \gamma_1), \ldots, (\delta_n, \gamma_n), \gamma_t \rightarrow \wedge_{i=1}^n \gamma_i \rangle \mid (\delta_j, \gamma_j) \text{ in } \mathcal{G}_{\delta_j} \}\)

- 0-step simulation: check for termination “mimicking.”
Algorithm SYMSIM($\delta^0_t, \delta^0_1, \ldots, \delta^0_n$)

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 $\phi$: “the target program in $v_t$ is simulated by the available programs in $\langle v_1, \ldots, v_n \rangle$.”

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

2. At every step, compute $\text{NEXT}[X]$: “one step refinement” of the simulation:

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

   - $\phi_{\text{new}}$: we can safely mimic (any) single action from the target.
Algorithm SYMSIM($\delta^0_t$, $\delta^0_1$, \ldots, $\delta^0_n$)

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 $\phi$: “the target program in $v_t$ is simulated by the available programs in $\langle v_1, \ldots, v_n \rangle$.”

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

2. At every step, compute $\text{NEXT}[X]$: “one step refinement” of the simulation:

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

   - $\phi_{new}$: 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^0_t, \delta^0_1, \ldots, \delta^0_n)\)

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 $\phi$: “the target program in $v_t$ is simulated by the available programs in $\langle v_1, \ldots, v_n \rangle$.”

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

2. At every step, compute $\text{Next}[X]$: “one step refinement” of the simulation:

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

   - $\phi_{\text{new}}$: 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 $X = \text{Next}[X]$. 
\textbf{Next}[X]: One-step refinement of the simulation

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

\[ \phi_{new} = \bigwedge_{v_t} \pi_x \psi_t \alpha_t v_t' \in E_t \]

\[ \left( \forall \vec{x}. \psi_t[s] \land \text{Poss}(\alpha_t, s) \rightarrow \bigvee_{i=1}^n \bigvee v_i \pi_y \alpha_i v_i' \in E_i \land \langle v_t', v_1', ..., v_i', ..., v_n, \phi_i \rangle \in X \right. \]

\[ \left. \exists \vec{y}. \alpha_t = \alpha_i \land \psi_i[s] \land R[\phi_i(\text{do}(\alpha_i, s))] \right) \].

For every potential target evolution from \( v_t \) to \( v_t' \) via action \( \alpha_t \), ...

if it can be done in the program (\( \psi_t \) holds) and the action \( \alpha_t \) is possible, ... 

then some available prog. \( \delta_i \) can evolve from \( v_i \) to \( v_i' \) via action \( \alpha_i \) such that:

1. the action \( \alpha_i \) can be matched to \( \alpha_t \);
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

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

\[ \phi_{new} = \bigwedge_{v_t} \pi_{\vec{x}} \xrightarrow{\alpha_t}_{\psi_t} v_t \in E_t \]

\[
\left( \forall \vec{x}. \psi_t[s] \land \text{Poss}(\alpha_t, s) \rightarrow \bigvee_{i=1}^n \bigvee_{v_i} \pi_{\vec{y}} \xrightarrow{\alpha_i}_{\psi_i} v_i \in E_i \land \langle v_t', v_1', \ldots, v_i', \ldots, v_n', \phi_i \rangle \in X \right)
\]

\[
\exists \vec{y}. \alpha_t = \alpha_i \land \psi_i[s] \land \mathcal{R}[\phi_i(do(\alpha_i, s))] \right). \]

For every potential target evolution from $v_t$ to $v_t'$ via action $\alpha_t$, ...

if it can be done in the program ($\psi_t$ holds) and the action $\alpha_t$ is possible, ...

then some available prog. $\delta_i$ can evolve from $v_i$ to $v_i'$ via action $\alpha_i$ such that:

1. the action $\alpha_i$ can be matched to $\alpha_t$;
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**

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

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

For every potential target evolution from $v_t$ to $v_t'$ via action $\alpha_t$, ...

if it can be done in the program ($\psi_t$ holds) and the action $\alpha_t$ is possible, ...

then some available prog. $\delta_i$ can evolve from $v_i$ to $v_i'$ via action $\alpha_i$ such that:

1. the action $\alpha_i$ can be matched to $\alpha_t$;

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]

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

\[
\phi_{new} = \bigwedge_{v_t} \pi_{x}^{\alpha_t} v_t' \in E_t \left( \forall \vec{x}. \psi_t[s] \land \text{Poss}(\alpha_t, s) \rightarrow \bigvee_{i=1}^{n} \bigvee_{v_i} \pi_{\vec{y}}^{\alpha_i} v_i' \in E_i \land \langle v_t', v_1, \ldots, v_i', \ldots, v_n, \phi_i \rangle \in X \right)
\]

\[
\exists \vec{y}. \alpha_t = \alpha_i \land \psi_i[s] \land R[\phi_i(\text{do}(\alpha_i, s))] \right).
\]

For every potential target evolution from \(v_t\) to \(v_t'\) via action \(\alpha_t\), ...

if it can be done in the program (\(\psi_t\) holds) and the action \(\alpha_t\) is possible, ...

then some available prog. \(\delta_i\) can evolve from \(v_i\) to \(v_i'\) via action \(\alpha_i\) such that:

1. the action \(\alpha_i\) can be matched to \(\alpha_t\);
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

\[
\text{Next}[X] = \{ \langle \nu_t, \nu_1, \ldots, \nu_n, \phi_{old} \land \phi_{new} \rangle \mid \langle \nu_t, \nu_1, \ldots, \nu_n, \phi_{old} \rangle \in X \},
\]

\[
\phi_{new} = \bigwedge_{\nu_t} \pi^{\overrightarrow{x}}_{\psi_t} \alpha_t \nu'_t \in E_t \\
\left( \forall \overrightarrow{x}. \psi_t[s] \land \text{Poss}(\alpha_t, s) \rightarrow \\
\bigvee_{i=1}^n \bigvee_{\nu_i} \pi^{\overrightarrow{y}}_{\psi_i} \alpha_i \nu'_i \in E_i \land \langle \nu'_t, \nu_1, \ldots, \nu'_i, \ldots, \nu_n, \phi_i \rangle \in X \\
\exists \overrightarrow{y}. \alpha_t = \alpha_i \land \psi_i[s] \land R[\phi_i(\text{do}(\alpha_i, s))]. \right)
\]

For every potential target evolution from $\nu_t$ to $\nu'_t$ via action $\alpha_t$, ...

- if it can be done in the program ($\psi_t$ holds) and the action $\alpha_t$ is possible, ...

  then some available prog. $\delta_i$ can evolve from $\nu_i$ to $\nu'_i$ via action $\alpha_i$ such that:

1. the action $\alpha_i$ can be matched to $\alpha_t$;
2. the program can indeed do the step;
3. after doing the step, we are still in simulation.
**$\mathbf{NEXT}[X]$**: One-step refinement of the simulation

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

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

For every potential target evolution from $v_t$ to $v'_t$ via action $\alpha_t$, ...

if it can be done in the program ($\psi_t$ holds) and the action $\alpha_t$ is possible, ...

then some available prog. $\delta_i$ can evolve from $v_i$ to $v'_i$ via action $\alpha_i$ such that:

1. the action $\alpha_i$ can be matched to $\alpha_t$;
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

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

\[
\phi_{new} = \land_{v_t} \frac{\pi_x}{\psi_t} \frac{\alpha_t}{v_t'} v_t' \in E_t
\]

\[
\left( \forall \vec{x}. \psi_t[s] \land Poss(\alpha_t, s) \rightarrow \right.
\]

\[
\bigvee_{i=1}^{n} \bigvee_{v_i} \frac{\pi_{\vec{y}_i}}{\psi_i} \frac{\alpha_i}{v_i'} v_i' \in E_i \land \langle v_t', v_1, \ldots, v_i', \ldots, v_n, \phi_i \rangle \in X
\]

\[
\exists \vec{y}. \alpha_t = \alpha_i \land \psi_i[s] \land R[\phi_i(do(\alpha_i, s))]\right).
\]

For every potential target evolution from \(v_t\) to \(v_t'\) via action \(\alpha_t\), ...

if it can be done in the program (\(\psi_t\) holds) and the action \(\alpha_t\) is possible, ...  

then some available prog. \(\delta_i\) can evolve from \(v_i\) to \(v_i'\) via action \(\alpha_i\) such that:

1. the action \(\alpha_i\) can be matched to \(\alpha_t\);
2. the program can indeed do the step;
3. after doing the step, we are still in simulation.
Technical Results

Theorem

If algorithm $\text{SYMSIM}(\delta_0^t, \delta_1^0, \ldots, \delta_n^0)$ terminates returning the set $X$. Then,

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

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

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

Theorem

If algorithm $\text{SYM SIM}(\delta_0^t, \delta_0^1, \ldots, \delta_0^n)$ terminates returning the set $X$. Then,

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

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 \textit{on-the-fly} using FO entailment only (on $\mathcal{D}_{S_0}$).

Idea:

after a request, jump to a configuration $\langle \nu_t, \nu_1, \ldots, \nu_n, \phi \rangle \in X$ for which $\phi$ holds.
Conclusions

Reasoning on unbounded data and processes is a challenge for CS since it leads to 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.

Main research direction for future work:
1. Incomplete information about the initial situation.
2. Offline vs online interpreters (see literature on high-level programs in AI).
3. Identify cases in which the technique becomes sound & complete.
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.