Agent Abstraction via Forgetting in the Situation Calculus

Kailun Luo[1], Yongmei Liu[1], Yves Lespérance[2], Ziliang Lin[1]

1.Dept. of Computer Science, Sun Yat-sen University
2.Dept. of Electrical Engineering and Computer Science, York University

ECAI 2020

Background

Abstraction plays an important role in Artificial Intelligence:

- heuristic, guide the search process in classical planning [Boddy, Fox, and Thiébaux 2007]
- method, form general solutions in generalized planning [Srivastava, Immerman, and Zilberstein 2008]
- technique, tackle state-space explosion in agent program verification [Mo, Li, and Liu 2016]

Background

Abstraction plays an important role in Artificial Intelligence:

- heuristic, guide the search process in classical planning [Boddy, Fox, and Thiébaux 2007]
- method, form general solutions in generalized planning [Srivastava, Immerman, and Zilberstein 2008]
- technique, tackle state-space explosion in agent program verification [Mo, Li, and Liu 2016]

Agent Abstraction
[Banihashemi, De Giacomo, and Lespérance 2017]

Background

Abstraction plays an important role in Artificial Intelligence:

- heuristic, guide the search process in classical planning [Boddy, Fox, and Thiébaux 2007]
- method, form general solutions in generalized planning [Srivastava, Immerman, and Zilberstein 2008]
- technique, tackle state-space explosion in agent program verification [Mo, Li, and Liu 2016]

Agent Abstraction
[Banihashemi, De Giacomo, and Lespérance 2017]

- expressive first-order framework based on the Situation Calculus and ConGolog

Background

Abstraction plays an important role in Artificial Intelligence:

- heuristic, guide the search process in classical planning [Boddy, Fox, and Thiébaux 2007]
- method, form general solutions in generalized planning [Srivastava, Immerman, and Zilberstein 2008]
- technique, tackle state-space explosion in agent program verification [Mo, Li, and Liu 2016]

Agent Abstraction
[Banihashemi, De Giacomo, and Lespérance 2017]

- expressive first-order framework based on the Situation Calculus and ConGolog
- abstraction of dynamic domains (action theories)

Agent abstraction [Banihashemi et al. 2017]

A high-level (HL) action theory, a low-level (LL) action theory and a refinement mapping:

Agent abstraction [Banihashemi et al. 2017]

A high-level (HL) action theory, a low-level (LL) action theory and a refinement mapping:

- the refinement mapping specifies abstraction:
  - HL fluent $\hookrightarrow$ LL formula, e.g.,

    $$all\_block\_ontable(s) \hookrightarrow \forall x.ontable(x, s)$$

  - HL action $\hookrightarrow$ LL program, e.g.,

    $$move\_to\_table(x) \hookrightarrow \pi y.unstack(x, y); putdown(x)$$

Agent abstraction [Banihashemi et al. 2017]

A high-level (HL) action theory, a low-level (LL) action theory and a refinement mapping:

- the refinement mapping specifies abstraction:
  - HL fluent $\hookrightarrow$ LL formula, e.g.,

    $$all\_block\_ontable(s) \hookrightarrow \forall x.ontable(x, s)$$

  - HL action $\hookrightarrow$ LL program, e.g.,

    $$move\_to\_table(x) \hookrightarrow \pi y.unstack(x, y); putdown(x)$$

- with the refinement mapping, m-bisimulation (similarity) between HL and LL models

Agent abstraction [Banihashemi et al. 2017]

A high-level (HL) action theory, a low-level (LL) action theory and a refinement mapping:

- the refinement mapping specifies abstraction:
  - HL fluent $\hookrightarrow$ LL formula, e.g.,

    $$all\_block\_ontable(s) \hookrightarrow \forall x.ontable(x, s)$$

  - HL action $\hookrightarrow$ LL program, e.g.,

    $$move\_to\_table(x) \hookrightarrow \pi y.unstack(x, y); putdown(x)$$

- with the refinement mapping, m-bisimulation (similarity) between HL and LL models
- based on m-bisimulation, sound abstraction and complete abstraction between HL and LL theories

Agent abstraction [Banihashemi et al. 2017]

A high-level (HL) action theory, a low-level (LL) action theory and a refinement mapping:

- the refinement mapping specifies abstraction:
  - HL fluent $\hookrightarrow$ LL formula, e.g.,

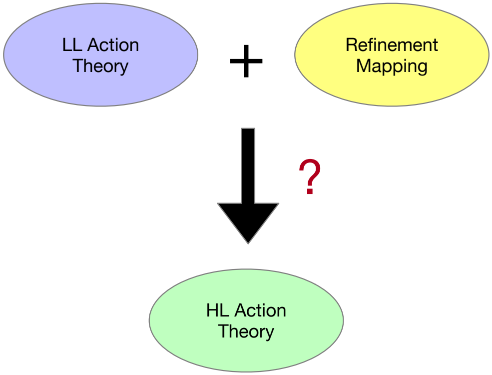
    $$all\_block\_ontable(s) \hookrightarrow \forall x.ontable(x, s)$$

  - HL action $\hookrightarrow$ LL program, e.g.,

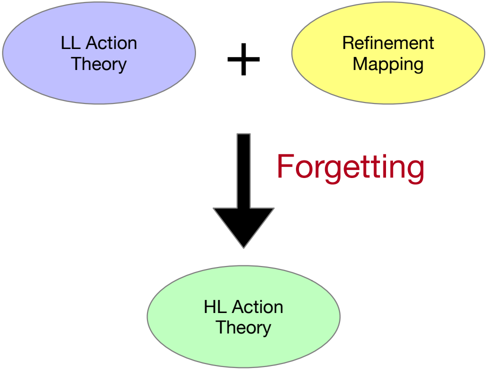
    $$move\_to\_table(x) \hookrightarrow \pi y.unstack(x, y); putdown(x)$$

- with the refinement mapping, m-bisimulation (similarity) between HL and LL models
- based on m-bisimulation, sound abstraction and complete abstraction between HL and LL theories
- sound abstraction: reasoning at HL $\hookrightarrow$ reasoning at LL

Problem



Problem


LL Action Theory
+
Refinement Mapping
Forgetting
HL Action Theory

Forgetting [Lin and Reiter 1994]

- Motivation: progression, i.e., update the knowledge base after an action was performed
- Intuition: omit the information of the symbols while retaining all the facts that are 'irrelevant' to these symbols

Forgetting [Lin and Reiter 1994]

- Motivation: progression, i.e., update the knowledge base after an action was performed
- Intuition: omit the information of the symbols while retaining all the facts that are ’irrelevant’ to these symbols
- Forgetting a predicate in a FO theory : not FO-definable [Lin and Reiter 1994]
- Forgetting a proposition in the propositional case is computable

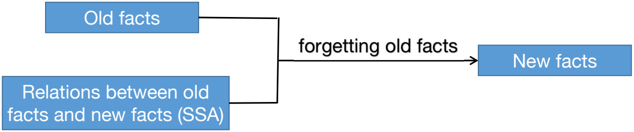
Contribution

Given a LL action theory, and a refinement mapping, we show

1. conditions: abstractions are representable by deterministic action theories and (Markovian) basic action theories
2. how to compute abstractions (HL theories) via forgetting

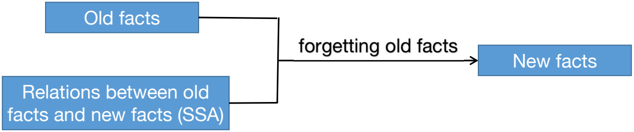
General idea

- Progression via forgetting:

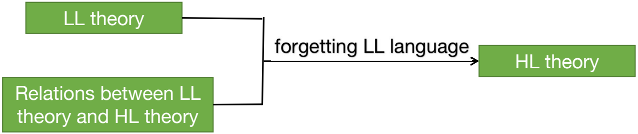


General idea

- Progression via forgetting:



- Abstraction via forgetting:



Characterize abstraction via forgetting

Theorem

Given a low-level theory $\mathcal{D}_l$ and a non-deterministic uniform (NDU) refinement mapping m, we have that

$$forget(\mathcal{D}_l \cup \mathcal{D}'_{una} \cup \mathcal{D}_m; \mathcal{A}_l \cup \mathcal{F}_l \cup \{\mathbb{B}\})$$

is a sound and complete abstraction of $\mathcal{D}_l$.

- NDU: different executions of programs at the LL should be indistinguishable at the HL (deterministic action)

Characterize abstraction via forgetting

Theorem

Given a low-level theory $\mathcal{D}_l$ and a non-deterministic uniform (NDU) refinement mapping m, we have that

$$forget(\mathcal{D}_l \cup \mathcal{D}'_{una} \cup \mathcal{D}_m; \mathcal{A}_l \cup \mathcal{F}_l \cup \{\mathbb{B}\})$$

is a sound and complete abstraction of $\mathcal{D}_l$.

- NDU: different executions of programs at the LL should be indistinguishable at the HL (deterministic action)
- $\mathcal{D}'_{una}$: LL and HL actions all stand for different actions

Characterize abstraction via forgetting

Theorem

Given a low-level theory $\mathcal{D}_l$ and a non-deterministic uniform (NDU) refinement mapping m, we have that

$$forget(\mathcal{D}_l \cup \mathcal{D}'_{una} \cup \mathcal{D}_m; \mathcal{A}_l \cup \mathcal{F}_l \cup \{\mathbb{B}\})$$

is a sound and complete abstraction of $\mathcal{D}_l$.

- NDU: different executions of programs at the LL should be indistinguishable at the HL (deterministic action)
- $\mathcal{D}'_{una}$: LL and HL actions all stand for different actions
- $\mathcal{D}_m$: relation between LL and HL theories, axiomatization of m-bisimulation

Characterize abstraction via forgetting

Theorem

Given a low-level theory $\mathcal{D}_l$ and a non-deterministic uniform (NDU) refinement mapping m, we have that

$$forget(\mathcal{D}_l \cup \mathcal{D}'_{una} \cup \mathcal{D}_m; \mathcal{A}_l \cup \mathcal{F}_l \cup \{\mathbb{B}\})$$

is a sound and complete abstraction of $\mathcal{D}_l$.

- NDU: different executions of programs at the LL should be indistinguishable at the HL (deterministic action)
- $\mathcal{D}'_{una}$: LL and HL actions all stand for different actions
- $\mathcal{D}_m$: relation between LL and HL theories, axiomatization of m-bisimulation
- $\mathcal{A}_l$: LL actions; $\mathcal{F}_l$: LL fluents

But...

- The result is a complex second-order theory

But...

- The result is a complex second-order theory
  - may not be representable as a basic action theory (BAT)

But...

- The result is a complex second-order theory
  - may not be representable as a basic action theory (BAT)
  - BATs have the form $\Sigma \cup \mathcal{D}_{una} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{S_0}$

But...

- The result is a complex second-order theory
  - may not be representable as a basic action theory (BAT)
  - BATs have the form $\Sigma \cup \mathcal{D}_{una} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{S_0}$
- Compute abstractions in the form of BATs via forgetting?

But...

- The result is a complex second-order theory
  - may not be representable as a basic action theory (BAT)
  - BATs have the form $\Sigma \cup \mathcal{D}_{una} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{S_0}$
- Compute abstractions in the form of BATs via forgetting?
  - We can compute each part separately

Compute abstraction via forgetting

Compute $\mathcal{D}^h_{S_0}$, $\mathcal{D}^h_{ap}$ and $\mathcal{D}^h_{ss}$ separately

Theorem

Given a low-level BAT $\mathcal{D}_l$ and a Markovian refinement mapping m, let T be $\Sigma \cup \mathcal{D}^h_{una} \cup \mathcal{D}^h_{S_0} \cup \mathcal{D}^h_{ap} \cup \mathcal{D}^h_{ss}$, where

- $\mathcal{D}^h_{S_0} \doteq \mathit{forget}(\mathcal{D}_{S_0} \cup \Phi_m[S_0]; \mathcal{F}_l)$;
- $\mathcal{D}^h_{ap}$ contains the set of sentences: for any $\mathbf{A} \in \mathcal{A}_h$,
$$\begin{aligned} Poss(\mathbf{A}(\vec{x}), s) \equiv\ & \mathit{forget}(\mathit{prog}[\mathit{Init}(s), \pi^*_m] \\ & \wedge \mathcal{R}[\top(s), m(\mathbf{A})(\vec{x})] \wedge \Phi_m[s]; \mathcal{F}_l); \end{aligned}$$
- $\mathcal{D}^h_{ss}$ contains: for any $\mathbf{A} \in \mathcal{A}_h$ and $\mathbf{F} \in \mathcal{F}_h$,
$$\begin{aligned} \mathbf{F}(\vec{y}, do(\mathbf{A}(\vec{x}), s)) \equiv\ & \mathit{forget}(\mathit{prog}[\mathit{Init}(s), \pi^*_m] \\ & \wedge \mathcal{R}[\mathbf{F}(\vec{y}, s), m(\mathbf{A})(\vec{x})] \wedge \Phi_m[s]; \mathcal{F}_l). \end{aligned}$$

Then T is a sound and complete abstraction of $\mathcal{D}_l$.

Markov property: the executability conditions and the effects of actions are fully determined by the present state of the system

Compute abstraction via forgetting

Compute $\mathcal{D}^h_{S_0}$, $\mathcal{D}^h_{ap}$ and $\mathcal{D}^h_{ss}$ separately

Theorem

Given a low-level BAT $\mathcal{D}_l$ and a Markovian refinement mapping m, let T be $\Sigma \cup \mathcal{D}^h_{una} \cup \mathcal{D}^h_{S_0} \cup \mathcal{D}^h_{ap} \cup \mathcal{D}^h_{ss}$, where

- $\mathcal{D}^h_{S_0} \doteq \mathit{forget}(\mathcal{D}_{S_0} \cup \Phi_m[S_0]; \mathcal{F}_l)$;
- $\mathcal{D}^h_{ap}$ contains the set of sentences: for any $\mathbf{A} \in \mathcal{A}_h$,
  $$\begin{aligned} Poss(\mathbf{A}(\vec{x}), s) \equiv \mathit{forget}(&prog[Init(s), \pi^*_m] \\ &\wedge \mathcal{R}[\top(s), m(\mathbf{A})(\vec{x})] \wedge \Phi_m[s]; \mathcal{F}_l); \end{aligned}$$
- $\mathcal{D}^h_{ss}$ contains: for any $\mathbf{A} \in \mathcal{A}_h$ and $\mathbf{F} \in \mathcal{F}_h$,
  $$\begin{aligned} \mathbf{F}(\vec{y}, do(\mathbf{A}(\vec{x}), s)) \equiv \mathit{forget}(&prog[Init(s), \pi^*_m] \\ &\wedge \mathcal{R}[\mathbf{F}(\vec{y}, s), m(\mathbf{A})(\vec{x})] \wedge \Phi_m[s]; \mathcal{F}_l). \end{aligned}$$

Then T is a sound and complete abstraction of $\mathcal{D}_l$.

Markov property: the executability conditions and the effects of actions are fully determined by the present state of the system

Construct HL action precondition $Poss(\mathbf{A}(\vec{x}), s)$

$$forget(prog[Init(s), \pi_m^*] \wedge \mathcal{R}[\top(s), m(\boldsymbol{A})(\vec{x})] \wedge \Phi_m[s]; \mathcal{F}_l)$$

1. $prog[Init(s), \pi_m^*]$ traverses all m-reachable situations (state-constraint-like formula)
2. $\mathcal{R}[\top(s), m(\mathbf{A})(\vec{x})]$ computes the executable condition for program $m(\mathbf{A})$
3. $\Phi_m[s]$ denotes relations between HL and LL fluents

Construct HL action precondition $Poss(\mathbf{A}(\vec{x}), s)$

$$forget(prog[Init(s), \pi_m^*] \wedge \mathcal{R}[\top(s), m(\boldsymbol{A})(\vec{x})] \wedge \Phi_m[s]; \mathcal{F}_l)$$

1. $prog[Init(s), \pi_m^*]$ traverses all m-reachable situations (state-constraint-like formula)
2. $\mathcal{R}[\top(s), m(\mathbf{A})(\vec{x})]$ computes the executable condition for program $m(\mathbf{A})$
3. $\Phi_m[s]$ denotes relations between HL and LL fluents

In the propositional case, $Poss(\boldsymbol{A}(\vec{x}), s)$ is always computable

Conclusion & Future work

Given a LL action theory and a refinement mapping:

- how to characterize abstractions via forgetting under the non-deterministic uniform condition
- how to compute abstractions of the form BATs under the Markovian restriction (propositional case, computable)

Conclusion & Future work

Given a LL action theory and a refinement mapping:

- how to characterize abstractions via forgetting under the non-deterministic uniform condition
- how to compute abstractions of the form BATs under the Markovian restriction (propositional case, computable)

Future work:

- extensions to HL theories that involve non-deterministic and non-Markovian actions
- sufficient conditions under which abstractions are always FO-definable

Conclusion & Future work

Given a LL action theory and a refinement mapping:

- how to characterize abstractions via forgetting under the non-deterministic uniform condition
- how to compute abstractions of the form BATs under the Markovian restriction (propositional case, computable)

Future work:

- extensions to HL theories that involve non-deterministic and non-Markovian actions
- sufficient conditions under which abstractions are always FO-definable
- application of agent abstraction: in planning, provide a mapping as a guide
- study of automated generation of these mappings

Thank you!