Approximation Based Reasoning and Conformant/Conditional Planning — Bridging Reasoning About Actions & Changes and Planning

Tran Cao Son

Department of Computer Science
New Mexico State University
Las Cruces, NM 88011, USA

ICAPS 2007

Reasoning about actions and changes (RAC)

Goal

How to represent actions and their effects? Reason about actions and their effects: what will be true/false after the execution of an action (an action sequence) in a given state? Illustration

Reasoning about actions and changes (RAC)

Goal

How to represent actions and their effects? Reason about actions and their effects: what will be true/false after the execution of an action (an action sequence) in a given state? Illustration

Activities

1. Development of languages for representing of dynamic domains (or actions and their effects)
2. Development of basic algorithms for computing successor states.
3. Considering of real-world domains (e.g. actions might have durations, non-deterministic, concurrent, etc.)

Reasoning about actions and changes (RAC)

Goal

How to represent actions and their effects? Reason about actions and their effects: what will be true/false after the execution of an action (an action sequence) in a given state? ▸ Illustration

Activities

1. Development of languages for representing of dynamic domains (or actions and their effects)
2. Development of basic algorithms for computing successor states.
3. Considering of real-world domains (e.g. actions might have durations, non-deterministic, concurrent, etc.)

Important Notions

1. State
2. Algorithms for computing of successor states

Planning

Goal

Development of domain-independent planner(s) for real-world applications: computing a plan to achieve a predefined goal

Goal

Development of domain-independent planner(s) for real-world applications: computing a plan to achieve a predefined goal

Activities

1. Development of several domain-independent planners (the algorithms for computing next state of the world (RAC) ensures correctness)
2. Development of techniques to improve the efficiency and scalability of planners.

Goal

Development of domain-independent planner(s) for real-world applications: computing a plan to achieve a predefined goal

Activities

1. Development of several domain-independent planners (the algorithms for computing next state of the world (RAC) ensures correctness)
2. Development of techniques to improve the efficiency and scalability of planners.

Important Considerations

1. Efficiency
2. Scalability

Realistic planning systems must be able to cope with

- incomplete information
- nondeterministic actions
- actions with durations
- actions that consume and produce resources
- deadlines of goals
- user preferences
- inconsistency of goals
- ...

Consequence

Each requirement represents a change in the “problem statement” for reasoning about actions and changes and/or planning.

RAC and Planning

Changes in problem statement (e.g. complete vs. incomplete initial state) lead to changes in

1. the notion of state (what is a state?) and/or
2. the basic algorithm (how to compute the successor state?)

in RAC and planning.

RAC and Planning

Changes in problem statement (e.g. complete vs. incomplete initial state) lead to changes in

1. the notion of state (what is a state?) and/or
2. the basic algorithm (how to compute the successor state?)

in RAC and planning.

Hypothesis

New algorithms for computing the next state will be needed in planning with complex domains (e.g. actions with durations, resources, etc.).

RAC and Planning

Changes in problem statement (e.g. complete vs. incomplete initial state) lead to changes in

1. the notion of state (what is a state?) and/or
2. the basic algorithm (how to compute the successor state?)

in RAC and planning.

Hypothesis

New algorithms for computing the next state will be needed in planning with complex domains (e.g. actions with durations, resources, etc.).

Study in RAC will play important role in the new frontier of planning.

RAC and Planning

Changes in problem statement (e.g. complete vs. incomplete initial state) lead to changes in

1. the notion of state (what is a state?) and/or
2. the basic algorithm (how to compute the successor state?)

in RAC and planning.

Hypothesis

New algorithms for computing the next state will be needed in planning with complex domains (e.g. actions with durations, resources, etc.).

Study in RAC will play important role in the new frontier of planning.

This tutorial: RAC in domains with static causal laws (state constraints) and planning with incomplete information and sensing actions.

Outline

Example [McCarthy, 1959]

- *Problem*: John is at home and his car is at home also. He wants to go to the airport (going to Providence to attend ICAPS 2007).

Example [McCarthy, 1959]

- *Problem*: John is at home and his car is at home also. He wants to go to the airport (going to Providence to attend ICAPS 2007).
- *Question*: What should John do?

Example [McCarthy, 1959]

- *Problem*: John is at home and his car is at home also. He wants to go to the airport (going to Providence to attend ICAPS 2007).
- *Question*: What should John do?
- *Solution*: Drive to the airport.

Example [McCarthy, 1959]

- *Problem*: John is at home and his car is at home also. He wants to go to the airport (going to Providence to attend ICAPS 2007).
- *Question*: What should John do?
- *Solution*: Drive to the airport.

Current Situation

This example can be encoded using any representation language developed for RAC and/or planning such as:

- situation calculus [McCarthy & Hayes, 1969]
- event calculus [Kowalski & Sergot, 1986]
- action languages [Gelfond & Lifschitz, 1993]
- fluent calculus [Thielscher, 2000]
- STRIPS [Fikes & Nilson, 1971]
- PDDL [Ghallab *et al.*, 1998]

Basic Ontologies (Situation Calculus, [McCarthy & Hayes, 1969])

- Situation: a complete state of the universe in an instance of time, often given by a set of facts
  - The fact "John is at home" is represented by the atom $at(john, home)$.
  - "His car is at home also" is another fact, that can be represented by the atom $at(car, home)$.
- Fluent: a function whose domain is the space of situations
  E.g. $at(john, home)$ is a Boolean function whose domain is the set of situations, $at(john, home)(s)$ is true says that "John is at home in situation s."
- Action: causes for changes from situations to situations
  E.g. $drive(home, airport)$ is an action that changes the situation in which John is at home to the situation in which John is at the airport.

Basic Ontologies (Situation Calculus, [Reiter, 2001])

- Situation: a possible history of the world
  - s_0 – initial situation.
  - $do(drive(home, airport), s_0)$ – situation after the execution of $drive(home, airport)$ in s_0.
- Fluent: a relation (a property of the world) whose (truth) value changes over time due to the execution of actions
  - $at(john, home)$ is a relation whose truth value changes – a *Boolean* fluent.
  - $number\_paper(john)$ is a relation whose value changes – a *functional* fluent.
- Action: causes for *all* changes in the world
  E.g. $drive(home, airport)$ is the *only action* that can change the world in our example.

Basic Ontologies (Action Languages, [GELFOND & LIFSCHITZ, 1993])

- Actions and fluents – same as in situation calculus in [REITER, 2001]
- Fluent literal – a fluent or its negation (a fluent preceeding by $\neg$) E.g. $at(john, home)$, $\neg at(john, home)$
- State: two commonly used definitions
  - a set of fluents or
  - a *complete* and *consistent* set of fluent literals, i.e., s is a state if for every fluent f
    - either f or $\neg f$ belongs to s; and
    - $\{f, \neg f\} \not\subseteq s$.

We will use the ontologies of action languages in this tutorial.

Action Language $\mathcal{AL}$ — Syntax

- Fluents: propositional symbols (e.g. $at(john, home)$, $at(john, airport)$, $at(car, home)$, and $at(car, airport)$)
- Actions: propositional symbols (e.g. $drive(home, airport)$ and $drive(airport, home)$) disjoint from fluents
- Laws:
  - *Dynamic law*: describes effects of actions

    $$drive(home, airport) \textbf{ causes } at(john, airport), at(car, airport)$$

  - *Static causal law*: represents the relationship between fluents

    $$\neg at(john, home) \textbf{ if } at(john, airport)$$

  - *Executability law*: encodes the conditions under which an action can be executed

    $$drive(home, airport) \textbf{ executable } at(john, home), at(car, home)$$

  - *Initial state*: a set of fluent literals

Action Theory — Syntax

Definition

An action theory is a pair $(\mathcal{D}, \delta)$ where

- $\mathcal{D}$, called an *action domain*, is a set of dynamic, static causal, and executability laws.
- δ, called the *initial state*, is a set of fluent literals.

$(\mathcal{D}_a, \delta_a)$—"Going to the Airport" Action Theory

$$\mathcal{D}_a = \left\{ \begin{array}{l} \mathit{drive}(\mathit{home}, \mathit{airport}) \textbf{ executable } \mathit{at}(\mathit{john}, \mathit{home}), \mathit{at}(\mathit{car}, \mathit{john}) \\ \mathit{drive}(\mathit{home}, \mathit{airport}) \textbf{ causes } \mathit{at}(\mathit{john}, \mathit{airport}), \mathit{at}(\mathit{car}, \mathit{airport}) \\ \mathit{drive}(\mathit{airport}, \mathit{home}) \textbf{ executable } \mathit{at}(\mathit{john}, \mathit{airport}), \mathit{at}(\mathit{car}, \mathit{airport}) \\ \mathit{drive}(\mathit{airport}, \mathit{home}) \textbf{ causes } \mathit{at}(\mathit{john}, \mathit{home}), \mathit{at}(\mathit{car}, \mathit{home}) \\ \neg \mathit{at}(\mathit{john}, \mathit{airport}) \textbf{ if } \mathit{at}(\mathit{john}, \mathit{home}) \\ \neg \mathit{at}(\mathit{car}, \mathit{airport}) \textbf{ if } \mathit{at}(\mathit{car}, \mathit{home}) \\ \neg \mathit{at}(\mathit{john}, \mathit{home}) \textbf{ if } \mathit{at}(\mathit{john}, \mathit{airport}) \\ \neg \mathit{at}(\mathit{car}, \mathit{home}) \textbf{ if } \mathit{at}(\mathit{car}, \mathit{airport}) \end{array} \right.$$

$$\delta_a = \{\mathit{at}(\mathit{john}, \mathit{home}), \mathit{at}(\mathit{car}, \mathit{home}), \neg \mathit{at}(\mathit{john}, \mathit{airport}), \neg \mathit{at}(\mathit{car}, \mathit{airport})\}$$

$\mathcal{AL}$ vs. PDDL (mostly a 1-1 correspondence, difference in static causal laws)

Domain: $\mathcal{D}_a$ in PDDL representation

```
(define (domain airport)
    (:predicates (at ?x ?y)
       (location ?x) (person ?p) (car ?c))
    (:action drive
       :parameters (?x ?y)
       :precondition (and (location ?x) (location ?y)
          (person ?p) (at ?p ?x)
          (car ?c) (at ?c ?x))
       :effect (and (at ?c ?y) (at ?p ?y)
          (not (at ?c ?x)) (not (at ?p ?x)))))
```

Problem: δ_a and Goal in PDDL representation

```
(define (problem airport-1-1) (:domain airport)
(:objects john car home airport)
(:init person(john) car(car) location(home) location(airport)
      at(john,home) at (car,home))
(:goal at(john,airport)))
```

$\mathcal{AL}$ vs PDDL

$\mathcal{AL}$	PDDL
Action	$\surd$
Fluent	Predicate
Effect	$\surd$
Executability condition	Precondition
Static causal law (allow cyclic)	Defined fluent or axiom (no cyclic)
Ground Instantiations (Variables: shorthand)	Typed Variables

$\mathcal{AL}$ vs PDDL

$\mathcal{AL}$	PDDL
Action	$\surd$
Fluent	Predicate
Effect	$\surd$
Executability condition	Precondition
Static causal law (allow cyclic)	Defined fluent or axiom (no cyclic)
Ground Instantiations (Variables: shorthand)	Typed Variables

Notes

1. Dealing directly with static causal laws is advantageous [THIEBAUX *et al.*, 2003].
2. Not many planners deal with static causal laws directly.

Fundamental Problems in RAC

- **The frame problem**: succinct representation of what does not change due to the execution of an action.
  E.g. John's home does not change its location after John's drove his car to the airport.

Fundamental Problems in RAC

- **The frame problem**: succinct representation of what does not change due to the execution of an action.
  E.g. John's home does not change its location after John's drove his car to the airport.
- **The qualification problem**: encoding the conditions under which an action can be executed.
  E.g. Normally, John can drive his car if he is at the same place as his car (Taken for granted: he has the key, his car will start, his car has enough gasoline, etc.)

Fundamental Problems in RAC

- **The frame problem**: succinct representation of what does not change due to the execution of an action.
  E.g. John's home does not change its location after John's drove his car to the airport.
- **The qualification problem**: encoding the conditions under which an action can be executed.
  E.g. Normally, John can drive his car if he is at the same place as his car (Taken for granted: he has the key, his car will start, his car has enough gasoline, etc.)
- **The ramification problem**: accounting for indirect effects of actions.
  E.g. If John's luggages are in his car then his luggages are at the airport after he executed the action of driving to the airport.

Fundamental Problems in RAC

- **The frame problem**: succinct representation of what does not change due to the execution of an action.
  E.g. John's home does not change its location after John's drove his car to the airport.
- **The qualification problem**: encoding the conditions under which an action can be executed.
  E.g. Normally, John can drive his car if he is at the same place as his car (Taken for granted: he has the key, his car will start, his car has enough gasoline, etc.)
- **The ramification problem**: accounting for indirect effects of actions.
  E.g. If John's luggages are in his car then his luggages are at the airport after he executed the action of driving to the airport.

Current Situation

Adequate solutions for the above problems have been proposed in different formalisms for various settings.

Key Ideas in Solving the Fundamental Problems in RAC

- **The frame problem**: the law of inertial "normally, a fluent's value does not change" (successor state axioms — one per fluent (e.g. [REITER, 2001])).
- **The qualification problem**: encodes only the minimal requirement for the action to be executed.
- **The ramification problem**: causal law "things do not change by themselves; there must be a reason for a fluent literal to change its value."

Action language $\mathcal{AL}$ (Semantics) — Intuition

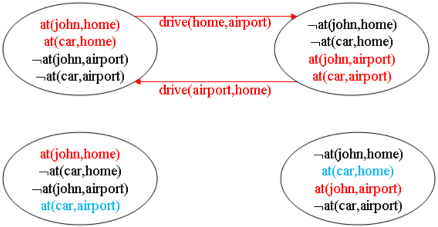
Given an action theory $(\mathcal{D}, \delta)$, the action domain $\mathcal{D}$ encodes a transition system consisting of elements of the form $\langle s_1, a, s_2 \rangle$ where s_1 and s_2 are states of the theory and a is an action that, when executed in s_1, changes the state of the world from s_1 into s_2. For example, in $(\mathcal{D}_a, \delta_a)$

$$\mathcal{D}_a = \begin{cases} drive(home, airport)\ \textbf{executable}\ \ at(john, home), at(car, john) \\ drive(home, airport)\ \textbf{causes}\ at(john, airport), at(car, airport) \\ drive(airport, home)\ \textbf{executable}\ \ at(john, airport), at(car, airport) \\ drive(airport, home)\ \textbf{causes}\ at(john, home), at(car, home) \\ \neg at(john, airport)\ \textbf{if}\ at(john, home) \\ \neg at(car, airport)\ \textbf{if}\ at(car, home) \\ \neg at(john, home)\ \textbf{if}\ at(john, airport) \\ \neg at(car, home)\ \textbf{if}\ at(car, airport) \end{cases}$$

a transition is

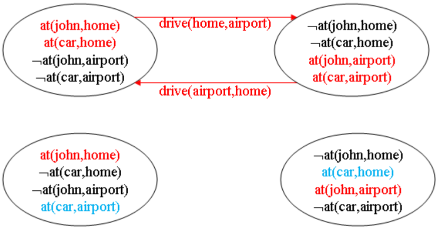
$$\begin{array}{c} \langle \{at(john, home), at(car, home), \neg at(john, airport), \neg at(car, airport)\}, \\ drive(home, airport), \\ \{\neg at(john, home), \neg at(car, home), at(john, airport), at(car, airport)\}\rangle \end{array}$$

Example of States and Transitions

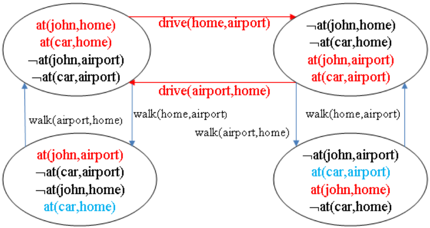


Going to the Airport

Example of States and Transitions



Going to the Airport



Adding the action *walk*(X, Y)

Action language $\mathcal{AL}$ (Semantics) I

States in $\mathcal{AL}$ theories

Let σ be a set of fluent literals.
σ satisfies a fluent literal l iff $l \in \sigma$ (denoted by $\sigma \models l$).
σ satisfies a set of fluent literals ψ iff $\psi \subseteq \sigma$ (denoted by $\sigma \models \psi$).
σ satisfies a static causal law φ **if** ψ if $\sigma \models \psi$ implies that $\sigma \models \varphi$.
$Cn_{\mathcal{D}}(\sigma)$, called the closure of σ, is the smallest set of literals that contains σ and satisfies all static causal laws in D.
Note: $Cn_{\mathcal{D}}(\sigma)$ might be inconsistent.

Definition

A state of an action domain $\mathcal{D}$ is a *complete* and *consistent* set of fluent literals which *satisfies all* static causal laws in $\mathcal{D}$ (i.e., $s = Cn_{\mathcal{D}}(s)$ and s is consistent and complete).

Action language $\mathcal{AL}$ (Semantics) II

Successor State

Given an action domain $\mathcal{D}$, a state s, and an action a.

- $de(a, s) = \{l \mid \mathcal{D}$ contains a **causes** l **if** φ and $s \models \varphi\}$ is called the *direct effects* of a in s.
- s' is a possible successor state of s after the execution of a in s if

$$s' = Cn_{\mathcal{D}}(de(a, s) \cup (s \cap s'))$$

Intuition

- $s \cap s'$ – inertial part
- $de(a, s)$ – direct effects of a
- $s' \setminus (de(a, s) \cup (s \cap s'))$ – indirect effects of a

Action language $\mathcal{AL}$ (Semantics) III

Example

For

$s_1 = \{at(john, home), at(car, home), \neg at(john, airport), \neg at(car, airport)\}$,

$s_2 = \{\neg at(john, home), \neg at(car, home), at(john, airport), at(car, airport)\}$

s_2 is a possible successor state of s_1 after the execution of $drive(home, airport)$ in s_1 because

$$de(drive(home, airport), s_1) = \{at(john, airport), at(car, airport)\}$$

$$s_1 \cap s_2 = \emptyset$$

and

$$Cn_{\mathcal{D}_a}(de(drive(home, airport), s_1) \cup (s_1 \cap s_2)) = s_2$$

Action language $\mathcal{AL}$ (Semantics) IV

Transition Function — Φ

$$\Phi : Actions \times States \rightarrow States$$

$$\Phi(a, s) = \begin{cases} \{s' \mid s' = Cn_{\mathcal{D}}(de(a, s) \cup (s \cap s'))\} \\ \text{ if } \mathcal{D} \text{ contains an executbtability} \\ \quad \text{law } a \textbf{ executable } \varphi \text{ and } s \models \varphi \\ \\ \Phi(a, s) = \emptyset \quad \text{otherwise} \end{cases}$$

Definition

a is executable in s if $\Phi(a, s) \neq \emptyset$. (The transition $\langle s, a, s' \rangle$ denotes that $s' \in \Phi(a, s)$.)

Action language $\mathcal{AL}$ (Semantics) V

Definition

For an action sequence $\alpha = a_1, \ldots, a_n$ and a state s, the extended transition function $\hat{\Phi}$ is defined by

$$\hat{\Phi}(\alpha, s) = \begin{cases} \{s\} & n = 0 \\ \bigcup_{s' \in \hat{\Phi}(\alpha_{n-1}, s)} \Phi(a_n, s') & \text{if } a \text{ is executable in } \hat{\Phi}(\alpha_{n-1}, s) \end{cases}$$

α is executable in s if $\hat{\Phi}(\alpha, s) \neq \emptyset$.

Definition

$(\mathcal{D}, \delta)$ entails the query φ **after** α, denoted by $(\mathcal{D}, \delta) \models \varphi$ **after** α, if φ is true in every state belonging to $\hat{\Phi}(\alpha, \delta)$.

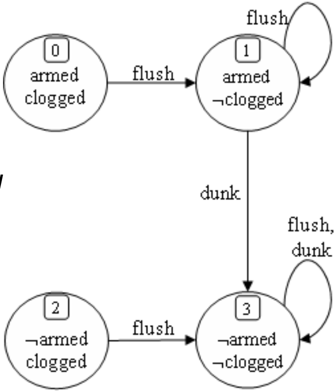
Bomb-In-The-Toilet Example

There may be a bomb in a package. Dunking the package into a toilet disarms the bomb. This action can be executed only if the toilet is not clogged. Flushing the toilet makes it unclogged.

- Fluents: *armed*, *clogged*
- Actions: *dunk*, *flush*
- Action domain:

$$\mathcal{D}_b = \left\{ \begin{array}{l} \textit{dunk } \textbf{causes } \neg \textit{armed } \textbf{if } \textit{armed} \\ \textit{flush } \textbf{causes } \neg \textit{clogged} \\ \textit{dunk } \textbf{executable } \neg \textit{clogged} \\ \textit{flush } \textbf{executable } \textit{true}^* \end{array} \right.$$

(* — present unless otherwise stated)



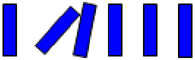
- Entailments

$$(\mathcal{D}_b, \{\textit{armed}, \textit{clogged}\}) \models \neg \textit{armed } \textbf{after } \langle \textit{flush}, \textit{dunk} \rangle$$

Dominoes Example

n dominoes $1, 2, \ldots, n$ line up on the table such that if domino i falls down then $i+1$ also falls down.

$$\mathcal{D}_d = \begin{cases} \textit{down}(n+1) \textbf{ if } \textit{down}(n) \\ \textit{touch}(i) \textbf{ causes } \textit{down}(i) \end{cases}$$



It can be shown that

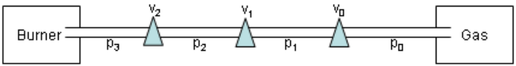
$$(\mathcal{D}_d, \delta_d) \models \textit{down}(n) \textbf{ after } \textit{touch}(i)$$

for every δ_d and i.

Gas Pipe

$n+1$ sections of pipe (pressured/unpressured) connected through n valves (opened/closed) connects a gas tank to burner. A valve can be opened only if the valve on its right is closed. Closing a valve causes the pipe section on its right side to be unpressured. The burner will start a flame if the pipe section connecting to it is pressured. The gas tank is always pressured.

- Fluents: *flame*, $opened(V)$, $pressured(P)$, $0 \leq V \leq n$, $0 \leq P \leq n+1$,
- Actions: $open(V)$, $close(V)$



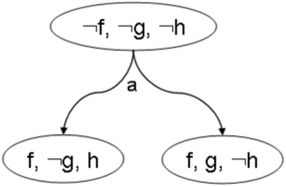
- Action domain:

$$\mathcal{D}_g = \begin{cases} open(I) \textbf{ executable } \neg opened(I+1) \\ open(I) \textbf{ causes } opened(I) \\ close(I) \textbf{ causes } \neg opened(I) \\ pressured(I+1) \textbf{ if } opened(I), pressured(I) \\ pressured(0) \textbf{ if } true \\ flame \textbf{ if } pressured(n+1) \end{cases}$$

Non-Deterministic $\mathcal{AL}$ Theories

Action theories in $\mathcal{AL}$ can be non-deterministic.

$$\mathcal{D}_n = \begin{cases} a \textbf{ causes } f \textbf{ if } \neg h, \neg g \\ h \textbf{ if } f, \neg g \\ g \textbf{ if } f, \neg h \end{cases}$$



Two successor states of $s_0 = \{\neg f, \neg g, \neg h\}$ after executing a:

$s_1 = \{f, \neg g, h\}$ and $s_2 = \{f, g, \neg h\}$

Planning and Complexity (Complete Information)

Definition (Planning Problem)

- Given: an $\mathcal{AL}$-action theory $(\mathcal{D}, \delta)$, where δ is a state of $\mathcal{D}$, and a set of fluent literals *G*.
- Determine: a sequence of actions α such that $(\mathcal{D}, \delta) \models$ *G* **after** α

From [LIBERATORE, 1997, TURNER, 2002]:

Theorem (Complexity)

- $(\mathcal{D}, \delta)$ *is deterministic: NP-hard even for plans of length 1, NP-complete for polynomial-bounded length plans (Classical Planning).*
- $(\mathcal{D}, \delta)$ *is non-deterministic:* Σ^2_P*-hard even for plans of length 1,* Σ^2_P*-complete for polynomial-bounded length plans (Conformant Planning in non-deterministic theories).*

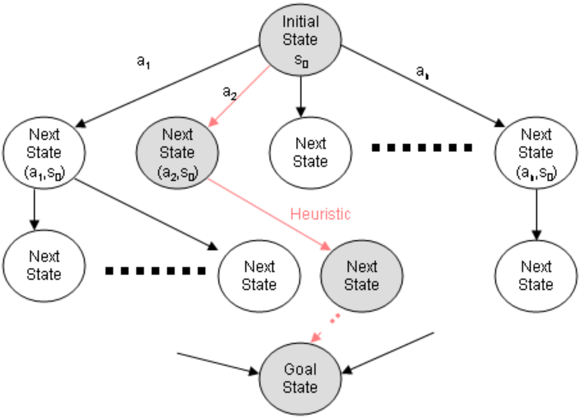
Planning Algorithms (Complete Information)

(1) *Heuristic search based approaches*
- State space: the search space is the set of possible states
- Plan space (partial order planning): the search space is the set of possible plans

(2) *Translation based approaches* (SAT-, model checking-, or answer set solvers).
- SAT: translation into a SAT instance
- Model checking: translation into a model checking problem
- Answer set programming: translation into a logic program

Search Based Approaches

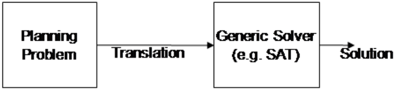
In search based planners, performance depends on how fast the search can be done ⇒ accuracy of heuristic is the key.



Heuristic Search Based Planners

Translation Based Approaches

In planners utilizing general theorem prover, performance depends on the performance of the general theorem prover.



Planning as Satisfiability

Pros and cons ((1) vs (2))

- (1) “independent” from the development in other communities, lots of good heuristics, easy to try out new heuristics
- (2) “dependent” from the development in other communities, heuristics are difficult to exploit in a systematic way
- (2) easier to deal with arbitrary domains than (1) (e.g. cyclic static causal laws)
- (2) easier to add “declarative domain knowledge”
- (2) easier to deal with “concurrent” actions than (1)

Approaches to Reasoning with Incomplete Information

Incomplete Information: initial state is not fully specified (e.g. δ in $(\mathcal{D}, \delta)$ might not be a state)

- Possible world approach (PSW): Extension of the transition function to a transition function over belief states.
- Approximation: Modifying the transition function to a transition function over approximation states.

Notation

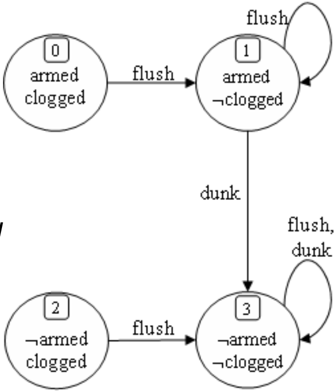
	Belief states (S and Σ)	Approximation states (δ and Δ)	
S	a set of states	a set of fluent literals	δ
Σ	a set of belief states	a set of approximation states	Δ

Example (Bomb-In-The-Toilet Revisited)

There may be a bomb in a package. Dunking the package into a toilet disarms the bomb. ...

- Fluents: *armed*, *clogged*
- Actions: *dunk*, *flush*
- Action domain:

$$\mathcal{D}_b = \begin{cases} \textit{dunk} \ \textbf{causes} \ \neg \textit{armed} \ \textbf{if} \ \textit{armed} \\ \textit{flush} \ \textbf{causes} \ \neg \textit{clogged} \\ \textit{dunk} \ \textbf{executable} \ \neg \textit{clogged} \end{cases}$$



- Initially, we know nothing about the value of *armed* and *clogged*.
- PWS: the initial belief state $S_0 = \{0, 1, 2, 3\}$.
- Approximation: the initial approximation state $\delta_0 = \emptyset$.

Definitions

Approximation state/Partial state: a set of fluent literals which is a part of some state.

Belief state: a set of states

For an action theory $(\mathcal{D}, \delta_0)$:

- Initial approximation state: δ_0 — a partial state
- Initial belief state:

$$S_0 = bef(\delta_0)$$

  where

$$bef(\delta) = \{s \mid \delta \subseteq s,\ s \text{ is a state}\}$$

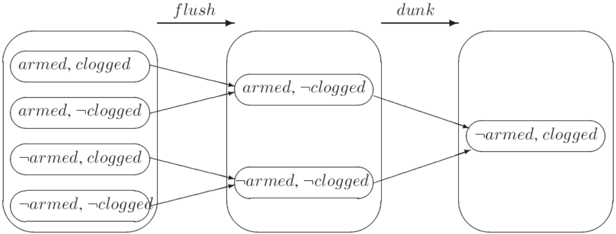
- A fluent formula φ true (false) in a belief state S if it true (false) in every state $s \in S$; it is unknown if it is neither true nor false in S.
- A fluent literal l is true (false) in an approximation state δ if $l \in \delta$ $(\neg l \in \delta)$; unknown, otherwise. The truth value of a fluent formula φ is defined in the usual way.

Possible World Approach

- $S_0 = bef(\delta_0)$
-

$$\Phi^c(a,S) = \begin{cases} \emptyset & \text{if } a \text{ is not executable in some } s \in S \\ \bigcup_{s \in S} \Phi(a,s) & \text{otherwise} \end{cases}$$

- Φ^c extended to $\hat{\Phi}^c$ in the usual way
- $(\mathcal{D}, \delta_0) \models^P \varphi$ **after** α if φ is true in the final belief state
- Size of search space: n fluents $\rightarrow 2^{2^n}$ belief states



Conformant Planning and Complexity

Definition (Conformant Planning Problem)

- Given: an $\mathcal{AL}$-action theory $(\mathcal{D}, \delta)$, where δ is a partial state, and a set of fluent literals *G*.
- Determine: a sequence of actions α such that $(\mathcal{D}, \delta) \models$ *G* **after** α

From [BARAL *et al.*, 2000, LIBERATORE, 1997, TURNER, 2002]:

Theorem (Complexity)

- Conformant Planning*: $(\mathcal{D}, \delta)$ is deterministic: Σ^2_P-hard even for plans of length 1, Σ^2_P-complete for polynomial-bounded length plans.*
- Conformant Planning*: $(\mathcal{D}, \delta)$ is non-deterministic: Σ^3_P-hard even for plans of length 1, Σ^3_P-complete for polynomial-bounded length plans.*

Planning Systems for Incomplete Domains

	$\texttt{DLV}^{\mathcal{K}}$	MBP	CMBP	SGP	POND	CFF	KACMBP
Language	*K*	*AR*	*AR*	PDDL	PDDL	PDDL	SMV
Sequential	yes	yes	yes	no	yes	yes	yes
Concurrent	yes	no	no	yes	no	no	no
Conformant	yes	yes	yes	yes	yes	yes	yes

Table: Features of Planning Systems

Planning Systems for Incomplete Domains

- Heuristic search based planners (search in the space of belief states)
  - CFF: A belief state *S* is represented by the initial belief state (a CNF formula) and the action sequence leading to *S*. To check whether a fluent literal *l* is true is *S*, a call to a SAT-solver is made. (subset of) PDDL as input.
  - POND: Graph plan based conformant planner. (subset of) PDDL as input.
- Translation into model checking: KACMBP (CMBP) – Input is a finite state automaton. Employing BDD (Binary Decision Diagram) techniques to represent and search the automaton. Consider nondeterministic domains with uncertainty in both the initial state and action effects.
- Translation into logic programming: $\texttt{DLV}^{\mathcal{K}}$ is a declarative, logic-based planning system built on top of the $\texttt{DLV}$ system (an answer set solver).

General Considerations and Properties

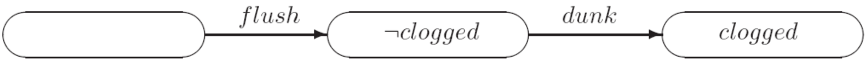
- Address the complexity problem of the possible world approach: give up completeness for efficiency in reasoning/planning
- Sound with respect to possible world semantics (formal proof is provided in some work)
- Representation languages and approaches are different
  - Situation calculus: [ETZIONI *et al.*, 1996, GOLDMAN & BODDY, 1994, PETRICK & BACCHUS, 2004]
  - Action languages: [SON & BARAL, 2001, SON & TU, 2006, SON *et al.*, 2005b]
  - Logic programming: [SON *et al.*, 2005a]

0-Approximation Approach [SON & BARAL, 2001]

- Initial partial state: δ_0
- Transition function is defined as

$$\Phi^0(a, \delta) = (\delta \cup de(a, \delta)) \setminus \neg pe(a, \delta)$$

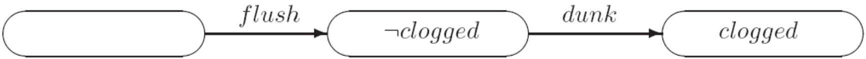
  where
  - $de(a, \delta)$ is the set of "direct effects" of a in δ
  - $pe(a, \delta)$ is the set of "possible effects" of a in δ
- $(\mathcal{D}, \delta_0) \models^0 \varphi$ **after** α if φ is true in the final partial state
- n fluents $\rightarrow 3^n$ partial states
- Incomplete
- No static causal laws



0-Approximation Approach – Example

$$\mathcal{D}_b = \begin{cases} \textit{dunk} \ \textbf{causes} \ \neg \textit{armed} \ \textbf{if} \ \textit{armed} \\ \textit{flush} \ \textbf{causes} \ \neg \textit{clogged} \\ \textit{dunk} \ \textbf{executable} \ \neg \textit{clogged} \end{cases}$$

- $\delta_0 = \emptyset$
  - *dunk* is not executable in δ_0
  - *flush* is executable in δ_0, $de(flush, \delta_0) = pe(flush, \delta_0) = \{\neg clogged\}$
  - $\Phi^0(flush, \delta_0) = \{\neg clogged\}$
- $\delta_1 = \{\neg clogged\}$
  - *dunk*, *flush* are executable in δ_1
  - $de(dunk, \delta_1) = \emptyset$ and $pe(dunk, \delta_1) = \{\neg armed\}$
  - $\Phi^0(dunk, \delta_1) = \{clogged\}$



Dealing with Static Causal Laws

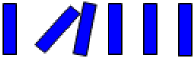
How will the 0-approximation fare in the dominoes example?

Dealing with Static Causal Laws

How will the 0-approximation fare in the dominoes example?
(Predictably: not so good!)

Dealing with Static Causal Laws

How will the 0-approximation fare in the dominoes example?
(Predictably: not so good!)



$$\mathcal{D}_d = \begin{cases} down(n+1) \textbf{ if } down(n) \\ touch(i) \textbf{ causes } down(i) \end{cases}$$

$\delta_0 = \emptyset$

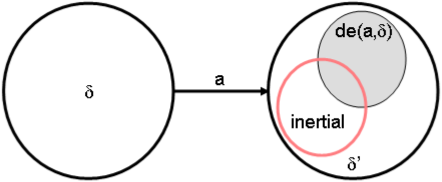
- $touch(i)$ is executable for every i
- $de(touch(i), \delta_0) = \{down(i)\}$ and $pe(touch(i), \delta_i) = \{down(i)\}$
- $\Phi^0(touch(i), \delta_0) = \{down(i)\}$

Intuitive result

$\{down(j) \mid i \leq j \leq n\} \subseteq \Phi^0(touch(i), \delta_0)$

Dealing with Static Causal Laws

$$\delta' = Cn_{\mathcal{D}}(de(a, \delta) \cup (\delta \cap \delta'))$$



The next state has three parts: (i) the direct effect $de(a, \delta)$; (ii) the inertial; (iii) the indirect effects (the closure of $Cn_{\mathcal{D}}$).

Dealing with Static Causal Laws

Question

What will be the inertial part?

Ideas

A literal does not change its value if it belongs to δ and

- either its negation cannot possibly hold in δ';
  $\Rightarrow$ possible holds approximation
- or it cannot possibly change in δ'
  $\Rightarrow$ possible change approximation

Φ^{ph} Approximation – Idea

A literal l possibly holds in the next state if

- it possibly holds in the current state (i.e., $l \notin \neg\delta$)
- it does not belong to the negation of the direct effect of the action (i.e., $l \notin \neg Cl_{\mathcal{D}}(de(a, \delta))$)
- there is some static causal law whose body possibly holds (i.e., there exists some static causal law l **if** φ such that φ possibly holds)

Φ^{ph} Approximation – Definition

$$E(a,\delta) = Cl_{\mathcal{D}}(e(a,\delta)) \quad [\text{always belongs to } \delta']$$

$$ph(a,\delta) = \bigcup_{i=0}^{\infty} ph^i(a,\delta) \quad [\text{possiblly holds in } \delta']$$

$$ph^0(a,\delta) = (pe(a,\delta) \cup \{l \mid \neg l \not\in \delta\}) \setminus \neg E(a,\delta)$$

OBS: if l **if** φ in $\mathcal{D}$ and φ possibly holds then l possibly holds.

$$ph^{i+1}(a,\delta) = ph^i(a,\delta) \cup \left\{ l \;\middle|\; \begin{array}{l} \exists[\, l \textbf{ if } \psi \,] \text{ in } \mathcal{D} \text{ s.t. } l \not\in \neg E(a,\delta), \\ \psi \subseteq ph^i(a,\delta), \neg\psi \cap E(a,\delta) = \emptyset \end{array} \right\}$$

Definition

- if a is not executable in δ then
$$\Phi^{ph}(a,\delta) = \emptyset$$
- otherwise,
$$\Phi^{ph}(a,\delta) = Cl_{\mathcal{D}}(\{l \mid l \not\in \neg ph(a,\delta)\})$$

Φ^{ph} Approximation – Example

$$\mathcal{D}_d = \begin{cases} down(i+1) \textbf{ if } down(i) \\ touch(i) \textbf{ causes } down(i) \end{cases}$$

Computation for $\delta_0 = \emptyset$

- $de(touch(i), \delta_0) = \{down(i)\}$ and $pe(touch(i), \delta_0) = \{down(i)\}$
- $E(touch(i), \delta_0) = \{down(j) \mid i \leq j \leq n\}$
- $ph^0(touch(i), \delta_0) = \{down(j) \mid 1 \leq j \leq n\} \cup \{\neg down(j) \mid 1 \leq j < i\}$
- $ph^k(touch(i), \delta_0) = \{down(j) \mid 1 \leq j \leq n\} \cup \{\neg down(j) \mid 1 \leq j < i\}$
- $\Phi^{ph}(touch(i), \delta_0) = \{down(j) \mid i \leq j \leq n\}$

Φ^{pc} Approximation – Idea

A literal l possibly changes if

- it is not in δ
- it is a possible effect a (i.e., there exists a dynamic law *a* **causes** l **if** φ and φ is not false in δ)
- it is a possibly indirect effect of a (i.e., there exists a static causal law l **if** φ and φ possibly changes)

Φ^{pc} Approximation

$$pc(a,\delta) = \bigcup_{i=0}^{\infty} pc^i(a,\delta)$$

$$pc^0(a,\delta) = pe(a,\delta) \setminus \delta$$

$$pc^{i+1}(a,\delta) \quad = \quad pc^i(a,\delta) \cup \left\{ l \;\middle|\; \begin{array}{l} \exists[\ l \textbf{ if } \psi\ ] \in \mathcal{D} \text{ s.t. } , l \notin \delta \\ \psi \cap pc^i(a,\delta) \neq \emptyset, \text{ and } \neg\psi \cap E(a,\delta) = \emptyset \end{array} \right\}$$

Definition

- if a is not executable in δ then
$$\Phi^{pc}(a,\delta) = \emptyset$$
- otherwise,
$$\Phi^{pc}(a,\delta) = Cl_{\mathcal{D}}(E(a,\delta) \cup (\delta \setminus \neg pc(a,\delta)))$$

Φ^{pc} Approximation – Example

$$\mathcal{D}_d = \begin{cases} down(i+1) \textbf{ if } down(i) \\ touch(i) \textbf{ causes } down(i) \end{cases}$$

Computation for $\delta_0 = \emptyset$

- $de(touch(i), \delta_0) = \{down(i)\}$ and $pe(touch(i), \delta_0) = \{down(i)\}$
- $E(touch(i), \delta_0) = \{down(j) \mid i \leq j \leq n\}$
- $pc^0(touch(i), \delta_0) = \{down(i)\}$
- $pc^1(touch(i), \delta_0) = \{down(i), down(i+1)\}$
- $pc(touch(i), \delta_0) = \{down(j) \mid i \leq j \leq n\}$
- $\Phi^{pc}(touch(i), \delta_0) = \{down(j) \mid i \leq j \leq n\}$

Properties of Φ^{ph} and Φ^{pc} Approximations

- Behave exactly as 0-approximation in action theories without static causal laws
- Sound but incomplete (proofs in [TU, 2007])
- Support parallel execution of actions (formal proofs available)
- Incompatibility between Φ^{ph} and Φ^{pc} $\Rightarrow$ could union the two to create a better approximation
- Deterministic: $\Phi^{A}(a, \delta)$ can be computed in polynomial-time
- Polynomial-length planning problem w.r.t Φ^{A} is NP-complete
- Could improve the approximations

Computing the Φ^{ph} Approximation

RESPH($\mathcal{D}$,a,δ)
INPUT: A domain description $\mathcal{D}$, an action a, and a partial state δ
OUTPUT: $\Phi^{ph}(a, \delta)$

```
BEGIN
    de = ∅        pe = ∅        lit = F ∪ ¬F
    for each dynamic causal law [a causes l if ψ] in D do
        if ψ possibly holds in δ then
            pe = pe ∪ {l}
            if ψ holds in δ then
                de = de ∪ {l}
    E = CLOSURE(D, de)
    ph = (pe ∪ (lit \ ¬δ)) \ ¬E
    repeat
        stop = true
        for each static causal law [l if ψ] in D do
            if l ∉ ¬E, ψ ⊆ ph, ¬ψ ∩ E = ∅, and l ∉ ph then
                ph = ph ∪ {l}        stop = false
    until stop
    return CLOSURE(D, lit \ ¬ph)
END
```

Figure: An algorithm for computing Φ^{ph}

Computing the Φ^{pc} Approximation

RESPC($\mathcal{D}$,a,δ)
INPUT: A domain description $\mathcal{D}$, an action a, and a partial state δ
OUTPUT: $\Phi^{pc}(a, \delta)$
BEGIN
    $de = \emptyset$ $\quad$ $pc = \emptyset$
    for each dynamic causal law [a **causes** l **if** ψ] in $\mathcal{D}$ **do**
        if ψ possibly holds in δ **then**
            if $l \notin \delta$ **then**
                $pc = pc \cup \{l\}$
            if ψ holds in δ **then**
                $de = de \cup \{l\}$
    $E = $ CLOSURE$(\mathcal{D}, de)$
    repeat
        $stop = true$
        for each static causal law [l **if** ψ] in $\mathcal{D}$ **do**
            if $\neg\psi \cap E = \emptyset$ and $\psi \cap pc \neq \emptyset$ and $l \notin \delta$ **then**
                $pc = pc \cup \{l\}$ $\quad$ $stop = false$
    until $stop$
    return CLOSURE$(\mathcal{D}, E \cup (\delta \setminus \neg pc))$
END

Figure: An algorithm for computing $\Phi^{pc}(a, \delta)$

What is good about the approximation?

Theorem (Complexity)

Conformant Planning*: $(\mathcal{D}, \delta)$ is deterministic: NP-complete for polynomial-bounded length plans.*

Consequence

If $(\mathcal{D}, \delta)$ is complete, planners can use the 0-approximation (lower complexity) instead of the possible world semantics. In fact, classical planners can be used to solve conformant planning (change in the computation of the next state.)

Approximation Based Conformant Planners

- Earlier systems [ETZIONI *et al.*, 1996, GOLDMAN & BODDY, 1994]: approximation is used in dealing with sensing actions (context-dependent actions and non-deterministic outcomes)
- PKS [PETRICK & BACCHUS, 2004] is very efficient (plus: use of domain knowledge in finding plans)
- CpA and CPASP [SON *et al.*, 2005b, SON *et al.*, 2005a] are competitive with others such as CFF, POND, and KACMBP in several benchmarks
- Incompleteness

Application in Conformant Planning

- CPASP:
  - Logic programming based
  - Uses Φ^{ph} approximation
  - Can generate both concurrent plans and sequential plans
  - Can handle disjunctive information about the initial state
  - Competitive with concurrent conformant planners and with others in problems with short solutions
- CPA:
  - Forward, best-first search with simple heuristic function (number of fulfilled subgoals)
  - Provides users with an option to select the approximation
  - Generates sequential plans only
  - Can handle disjunctive information about the initial state
  - Competitive with other state-of-the-art conformant planners

Experiments — Planning with concurrent actions I

Gas Pipe

Problem	$\mathcal{C}$-PLAN	DLV$^{\mathcal{K}}$	CPASP
Gaspipe^p(3)	-	0.08	0.40
Gaspipe^p(5)	-	0.17	0.75
Gaspipe^p(7)	-	0.44	1.22
Gaspipe^p(9)	-	17.44	3.17
Gaspipe^p(11)	-	-	8.83

Experiments — Planning with concurrent actions II

Cleaner

Problem	$\mathcal{C}$-PLAN	DLV$^{\mathcal{K}}$	CPASP
Cleaner^p(2,2)	0.05	0.07	0.26
Cleaner^p(2,5)	0.12	0.06	0.30
Cleaner^p(2,10)	0.06	0.07	0.30
Cleaner^p(4,2)	0.06	0.19	0.77
Cleaner^p(4,5)	0.09	0.80	0.93
Cleaner^p(4,10)	0.13	237.63	1.16
Cleaner^p(6,2)	0.11	4.47	1.98
Cleaner^p(6,5)	0.19	986.73	2.94
Cleaner^p(6,10)	0.35	-	3.73

Experiments — Planning with concurrent actions III

Bomb In The Toilet

Problem	$\mathcal{C}$-PLAN	DLV$^{\mathcal{K}}$	CPASP
$BT^p(2,2)$	0.07	0.07	0.11
$BT^p(4,2)$	0.05	0.09	0.26
$BT^p(6,2)$	1.81	3.06	0.34
$BT^p(8,4)$	4.32	10.52	0.24
$BT^p(10,4)$	-	-	1.91
$BTC^p(2,2)$	0.05	0.05	0.13
$BTC^p(4,2)$	0.07	0.90	0.30
$BTC^p(6,2)$	7.51	333.27	0.67
$BTC^p(8,4)$	-	-	0.50
$BTC^p(10,4)$	-	-	1192.45

Experiments — Sequential Planning I

Cleaner

Problem	KACMBP	POND	CFF	CPA^{ph}	CPA^{pc}
Cleaner(2,5)	0.01	0.17	0.03	0.01	0.00
Cleaner(2,10)	0.08	0.85	0.07	0.03	0.02
Cleaner(2,20)	0.62	15.87	0.15	0.19	0.07
Cleaner(2,50)	13.55	-	0.80	2.76	0.92
Cleaner(2,100)	185.39	-	5.72	22.71	7.51
Cleaner(5,5)	0.01	1.46	0.11	0.07	0.04
Cleaner(5,10)	0.09	12.86	0.24	0.26	0.16
Cleaner(5,20)	7.82	214.83	0.85	1.78	0.88
Cleaner(5,50)	227.82	-	14.36	26.66	11.66
Cleaner(5,100)	-	-	-	214.27	92.81

Experiments — Sequential Planning II

Logistics

Problem	KACMBP	POND	CFF	CPA^{ph}	CPA^{pc}
Log(2,2,2)	0.19	1.11	0.03	0.15	0.16
Log(2,3,3)	355.96	11.89	0.06	8.95	9.543
Log(3,2,2)	2.10	4.02	0.06	11.87	4.54
Log(3,3,3)	29.8	24.66	0.12	409.68	435.55
Log(4,3,3)	-	40.12	0.14	-	-

Experiments — Sequential Planning III

Ring

Problem	KACMBP	POND	CFF	CPA^{ph}	CPA^{pc}
Ring(2)	0.00	0.15	0.06	0.00	0.00
Ring(3)	0.00	0.08	0.23	0.01	0.01
Ring(4)	0.00	0.25	3.86	0.02	0.02
Ring(5)	0.00	0.96	63.67	0.03	0.04
Ring(10)	0.02	-	-	1.01	1.05
Ring(15)	0.04	-	-	6.76	6.10
Ring(20)	0.15	-	-	27.44	22.68
Ring(25)	0.32	-	-	79.58	64.60

Experiments — Sequential Planning IV

Bomb In The Toilet with Uncertainty

Problem	KACMBP	POND	CFF	CPA^{ph}	CPA^{pc}
BTUC(10,1)	0.01	0.07	0.05	0.01	0.01
BTUC(20,1)	0.05	0.57	0.17	0.07	0.03
BTUC(50,1)	0.51	28.69	5.33	0.82	0.33
BTUC(100,1)	3.89	682.33	121.8	6.24	2.36
BTUC(10,5)	0.09	0.65	0.07	0.04	0.02
BTUC(20,5)	0.30	7.28	0.16	0.18	0.09
BTUC(50,5)	1.66	348.28	4.70	1.90	0.83
BTUC(100,5)	6.92	-	113.95	12.13	5.266
BTUC(10,10)	0.30	2.50	0.05	0.07	0.04
BTUC(20,10)	0.97	27.69	0.13	0.40	0.19
BTUC(50,10)	5.39	960.00	4.04	3.74	1.63
BTUC(100,10)	35.83	-	102.56	20.94	9.80

Experiments — Sequential Planning V

Domino

Problem	KACMBP	POND	CFF	CPA^{ph}	CPA^{pc}
Domino(10)	0.01	1.72	0.05	0.00	0.00
Domino(50)	0.27	-	4.44	0.00	0.00
Domino(100)	2.56	-	-	0.01	0.01
Domino(200)	29.10	-	-	0.02	0.02
Domino(500)	-	-	-	0.06	0.06
Domino(1000)	-	-	-	0.20	0.20
Domino(2000)	-	-	-	0.63	0.65
Domino(5000)	-	-	-	3.81	4.01

$\mathcal{AL}$ vs. PDDL — Revisited

1. PDDL domains can be translated into $\mathcal{AL}$ domains — 1-to-1
2. $\mathcal{AL}$ domains can be translated into PDDL — might need to introduce additional actions (only polynomial number of actions)

Consequence

Planners using PDDL as their representation language can make use of the approximations.

Motivation Example

- Action domain:

$$\mathcal{D}_b = \left\{ \begin{array}{l} \textit{dunk} \textbf{ causes } \neg \textit{armed} \textbf{ if } \textit{armed} \\ \textit{flush} \textbf{ causes } \neg \textit{clogged} \\ \textit{dunk} \textbf{ executable } \neg \textit{clogged} \end{array} \right\}$$

- Initial State: $\delta_0 = \emptyset$

- If δ_0 is splitted into $\Delta_1 = \{\{\textit{armed}\}, \{\neg \textit{armed}\}\}$ then

$$(\mathcal{D}_b, \Delta_1) \models^0 \neg \textit{armed} \textbf{ after } \langle \textit{flush}, \textit{dunk} \rangle$$

See why?

- Splitting δ_0 into $\Delta_2 = \{\{\textit{clogged}\}, \{\neg \textit{clogged}\}\}$ does not help:

$$(\mathcal{D}_b, \Delta_2\}) \not\models^0 \neg \textit{armed} \textbf{ after } \langle \textit{flush}, \textit{dunk} \rangle$$

See why?

Questions

Given an action theory $(\mathcal{D}, \delta_0)$ and a fluent formula φ,

- When $\models^0$ is complete?, i.e., when

$$(\mathcal{D}, \delta_0) \models^P \varphi \textbf{ after } \alpha \Leftrightarrow (\mathcal{D}, \delta_0) \models^0 \varphi \textbf{ after } \alpha$$

  for every sequence of actions α?
- How to make it complete? what fluents whose values need to be considered in the beginning in order for 0-approximation to be complete?

Why important?

- If $\models^0$ is complete then the 0-approximation can be used instead of the possible world approach (reasoning process does not need to examine all possible initial states of the domain.)
- If $\models^0$ is incomplete then $(\mathcal{D}, \delta_0)$ can be transformed into a complete one.

When is $\models^0$ complete?

$(\mathcal{D}, \delta_0) \models \varphi$ after α?

- Possible World Approach: Our knowledge is a belief state (set of possible states) *bel*(δ_0)
- 0-approximation: Our knowledge is a partial state δ

When is $\models^0$ complete?

$(\mathcal{D}, \delta_0) \models \varphi$ after α?

- Possible World Approach: Our knowledge is a belief state (set of possible states) *bel*(δ_0)
- 0-approximation: Our knowledge is a partial state δ

Basic Idea

Characterize when reasoning with *bel*(δ_0) is the same as reasoning with δ (w.r.t. φ) — δ provides enough knowledge for reasoning about φ.

▸ Illustration

When is $\models^0$ complete?

$(\mathcal{D}, \delta_0) \models \varphi$ after α?

- Possible World Approach: Our knowledge is a belief state (set of possible states) $bel(\delta_0)$
- 0-approximation: Our knowledge is a partial state δ

Basic Idea

Characterize when reasoning with $bel(\delta_0)$ is the same as reasoning with δ (w.r.t. φ) — δ provides enough knowledge for reasoning about φ.

Illustration

Approach

- Dependency $\lhd$ between literals: $l \lhd g$ implies that to reason about l, may need to know g
- Reducibility: $S \gg_\varphi \delta$ if there exists a state $s \in S$ such that φ does not depend on $s \setminus \delta$

Dependencies and Reducibility I

- A literal l depends on a literal g, written as $l \triangleleft g$, if
  - $l = g$ or $\neg l \triangleleft \neg g$, or
  - there exists a **causes** l **if** ψ such that $g \in \psi$, or
  - there exists $l \triangleleft h$ and $h \triangleleft g$.
- An action a depends on a literal l, written as $a \triangleleft l$, if either
  - there exists a **executable** ψ such that $\neg l \in \psi$, or
  - there exists a literal g such that $a \triangleleft g$ and $g \triangleleft l$.

Example

$$\mathcal{D}_b = \left\{ \begin{array}{l} \textit{dunk } \textbf{causes } \neg \textit{armed } \textbf{if } \textit{armed} \\ \textit{dunk } \textbf{causes } \textit{clogged} \\ \textit{flush } \textbf{causes } \neg \textit{clogged} \\ \textit{dunk } \textbf{executable } \neg \textit{clogged} \end{array} \right\}$$

- $\neg armed \triangleleft \neg armed$ as $\triangleleft$ is reflexive
- $\neg armed \triangleleft armed$ because of the first statement

Dependencies and Reducibility II

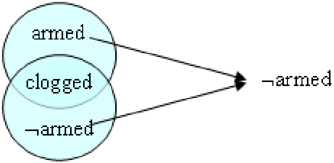
- A disjunction $\gamma = l_1 \vee \cdots \vee l_n$ depends on a literal g, written as $\gamma \lhd g$, if there exists some l_i such that , written as $l_i \lhd g$.
- A belief state S is reducible to δ w.r.t. $\varphi = \gamma_1 \wedge \cdots \wedge \gamma_n$, denoted by $S \gg_\varphi \delta$ if
  - δ is a subset of every state s in S,
  - for $1 \leq i \leq n$, there exists a state $s \in S$ such that $\gamma_i \ntriangleleft (s \setminus \delta)$, and
  - for any action a, there exists a state $s \in S$ such that $a \ntriangleleft (s \setminus \delta)$.

Example

- For $\delta = \{clogged\}$ (or $\{\neg clogged\}$), $bef(\delta) \not\gg_{\neg armed} \delta$ as $\neg armed \lhd s \setminus \delta$ for every $s \in bef(\delta)$
- But, for $\delta = \{armed\}$ (or $\{\neg armed\}$), $bef(\delta) \gg_{\neg armed} \delta$ as $\neg armed \ntriangleleft s \setminus \delta$ for some $s \in bef(\delta)$ (e.g. $s = \{clogged, armed\}$).

Example (Summary)

$$\mathcal{D}_b = \left\{ \begin{array}{l} \textit{dunk } \textbf{causes } \neg armed \textbf{ if } armed \\ \textit{dunk } \textbf{causes } clogged \\ \textit{flush } \textbf{causes } \neg clogged \\ \textit{dunk } \textbf{executable } \neg clogged \end{array} \right\}$$

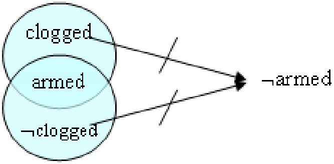


- Dependencies:
  - $\neg armed \lhd \neg armed$
  - $\neg armed \lhd armed$
- Reducibility:
  - For $\delta = \{clogged\}$ (or $\{\neg clogged\}$),

    $$bef(\delta) \not\gg_{\neg armed} \delta$$

  - But, for $\delta = \{armed\}$ (or $\{\neg armed\}$)

    $$bef(\delta) \gg_{\neg armed} \delta$$



Condition for Completeness of 0-approximation

Theorem

Let $(\mathcal{D}, \delta_0)$ *be an action theory without static causal laws and* φ *be a fluent formula. If* $bef(\delta_0) \gg_\varphi \delta_0$ *then for every sequence of actions* α*,*

$$(\mathcal{D}, \delta_0) \models^P \varphi \textbf{ after } \alpha \Leftrightarrow (\mathcal{D}, \delta_0) \models^0 \varphi \textbf{ after } \alpha$$

Condition for Completeness of 0-approximation

Theorem

Let $(\mathcal{D}, \delta_0)$ *be an action theory without static causal laws and* φ *be a fluent formula. If* $bef(\delta_0) \gg_{\varphi} \delta_0$ *then for every sequence of actions* α*,*

$$(\mathcal{D}, \delta_0) \models^P \varphi \textbf{ after } \alpha \Leftrightarrow (\mathcal{D}, \delta_0) \models^0 \varphi \textbf{ after } \alpha$$

Examples

- Cannot say whether $(\mathcal{D}_1, \{\{clogged\}\}) \models^P \neg armed \textbf{ after } \alpha$ iff $(\mathcal{D}_1, \{\{clogged\}\}) \models^0 \neg armed \textbf{ after } \alpha$ for every α as $bef(\{clogged\}) \not\gg_{\neg armed} \{clogged\}$
- But, $(\mathcal{D}_1, \{\{armed\}\}) \models^P \neg armed \textbf{ after } \alpha$ iff $(\mathcal{D}_1, \{\{armed\}\}) \models^0 armed \textbf{ after } \alpha$ for every α as $bef(\{armed\}) \gg_{\neg armed} \{armed\}$

How to make $\models^0$ complete?

- **Basic Idea**: find a set of fluents F, called decisive set, to split δ_0 into Δ_0 such that for each $\delta \in \Delta_0$,

$$\mathit{bef}(\delta) \gg_\varphi \delta$$

as by the completeness theorem, this guarantees

$$(\mathcal{D}, \delta_0) \models^P \varphi \textbf{ after } \alpha \Leftrightarrow (\mathcal{D}, \Delta_0) \models^0 \varphi \textbf{ after } \alpha$$

  - Example: $\{\mathit{armed}\}$ is a decisive set for $\emptyset$ w.r.t. $\varphi = \neg \mathit{armed}$ but $\{\mathit{clogged}\}$ is not

How to make $\models^0$ complete?

- **Basic Idea**: find a set of fluents F, called decisive set, to split δ_0 into Δ_0 such that for each $\delta \in \Delta_0$,

$$\mathit{bef}(\delta) \gg_\varphi \delta$$

as by the completeness theorem, this guarantees

$$(\mathcal{D}, \delta_0) \models^P \varphi \textbf{ after } \alpha \Leftrightarrow (\mathcal{D}, \Delta_0) \models^0 \varphi \textbf{ after } \alpha$$

  - Example: $\{\mathit{armed}\}$ is a decisive set for $\emptyset$ w.r.t. $\varphi = \neg \mathit{armed}$ but $\{\mathit{clogged}\}$ is not
- We developed an algorithm for computing such a decisive set
  - based on analyzing dependency relationships
  - most of the time returns a minimal one
  - runs in polynomial time

Computing A Decisive Set

Algorithm

DECISIVE($(\mathcal{D}, \delta_0), \varphi$)
INPUT: an action theory $(\mathcal{D}, \delta_0)$ and a formula $\varphi = \gamma_1 \wedge \cdots \wedge \gamma_n$
OUTPUT: a decisive set for δ_0 w.r.t. φ

1. BEGIN
2. $F = \emptyset$
3. compute dependencies between literals
4. compute dependencies between actions and literals
5. **for** each fluent f unknown in δ_0 **do**
6. **if** there exists $1 \leq i \leq n$ s.t. γ_i depends on both f and $\neg f$ or
7. an action a s.t. a depends on both f and $\neg f$
8. **then** $F = F \cup \{f\}$
9. **return** F;
10. END

Dealing with Static Causal Laws

Definition

Let $\mathcal{D}$ be an action domain. A fluent literal l depends on a fluent literal g, written as $l \lhd g$, if and only if one of the following conditions holds.

1. $l = g$
2. $\mathcal{D}$ contains a dynamic causal law [a **causes** l **if** ψ] such that $g \in \psi$.
3. $\mathcal{D}$ contains a static causal law [l **if** ψ] such that $g \in \psi$.
4. There exists a fluent literal h such that $l \lhd h$ and $h \lhd g$.
5. The complement of l depends on the complement of g, i.e., $\neg l \lhd \neg g$.

Completeness Condition for Simple Theories

A belief state S is reducible to δ w.r.t. $\varphi = \gamma_1 \wedge \cdots \wedge \gamma_n$, denoted by $S \gg_\varphi \delta$ if

- δ is a subset of every state s in S,
- for $1 \leq i \leq n$, there exists a state $s \in S$ such that $\gamma_i \ntrianglelefteq (s \setminus \delta)$, and
- for any action a, there exists a state $s \in S$ such that $a \ntrianglelefteq (s \setminus \delta)$.

Definition

An action theory $(\mathcal{D}, \delta_0)$ is *simple* if every static causal law in $\mathcal{D}$ is of the form l **if** g.

Theorem

Let $(\mathcal{D}, \delta_0)$ be a simple action theory and φ be a fluent formula. If $bef(\delta_0) \gg_\varphi \delta_0$ then for every sequence of actions α,

$$(\mathcal{D}, \delta_0) \models^P \varphi \textbf{ after } \alpha \Leftrightarrow (\mathcal{D}, \delta_0) \models^0 \varphi \textbf{ after } \alpha$$

Reasoning with disjunctive information

Reasoning with disjunctive information can be done similar to reasoning in the presence of incomplete information since the knowledge of a reasoner can be represented by a belief states.

- Not a problem with reasoning but representation for possible world approach $\Rightarrow$ compact representation of the initial belief state or belief states during the reasoning process is useful (e.g. CFF)
- For approximation based reasoning: compacting a belief state into a single partial state causes losing of information $\Rightarrow$ expansion into set of partial states if completeness is required (e.g. CpA)
- Completeness condition still holds

Bomb-In-The-Toilet Domain

Problem	KACMBP	POND	CFF	CPA$^+$
Bomb(5,1)	0.00	0.03	0.03	0.00
Bomb(10,1)	0.01	0.07	0.05	0.00
Bomb(20,1)	0.05	0.57	0.17	0.03
Bomb(50,1)	0.51	28.69	5.33	0.31
Bomb(100,1)	3.89	682.33	121.8	2.28
Bomb(5,5)	0.04	0.10	0.04	0.00
Bomb(10,5)	0.09	0.65	0.07	0.02
Bomb(20,5)	0.30	7.28	0.16	0.07
Bomb(50,5)	1.66	348.28	4.70	0.68
Bomb(100,5)	6.92	-	113.95	4.50
Bomb(5,10)	0.11	0.35	0.03	0.01
Bomb(10,10)	0.30	2.50	0.05	0.05
Bomb(20,10)	0.97	27.69	0.13	0.15
Bomb(50,10)	5.39	960.00	4.04	1.26
Bomb(100,10)	35.83	-	102.56	7.44

Why sensing actions?

- Some properties of the domain can be observed after some sensing actions are executed
    - Cannot decide whether a package contains a bomb until we use a special device to detect it
    - A robot cannot determine an obstacle until it uses a sensor to detect it

Why sensing actions?

- Some properties of the domain can be observed after some sensing actions are executed
  - Cannot decide whether a package contains a bomb until we use a special device to detect it
  - A robot cannot determine an obstacle until it uses a sensor to detect it
- Two important questions:
  - What is a plan?

Why sensing actions?

- Some properties of the domain can be observed after some sensing actions are executed
  - Cannot decide whether a package contains a bomb until we use a special device to detect it
  - A robot cannot determine an obstacle until it uses a sensor to detect it
- Two important questions:
  - What is a plan?
  - How to reason about sensing actions?

Extending $\mathcal{AL}$ to handle sensing actions

- Allow knowledge-producing laws of the form

  ***a* determines θ**

  "if sensing action *a* is executed, then the values of $l \in \theta$ will be known"
- New language is called $\mathcal{AL}_\mathcal{K}$

Why sensing actions? — Example

- One bomb, two packages; exactly one package contains the bomb
- Initially, the toilet is not clogged. No *flush* action.
- Bomb can be detected by only by *X-ray*.

$$\mathcal{D}_2 = \left\{ \begin{array}{l} \textbf{oneof } \{armed(1), armed(2)\} \\ dunk(P) \textbf{ causes } \neg armed(P) \\ dunk(P) \textbf{ causes } clogged \\ \textbf{impossible } dunk(P) \textbf{ if } clogged \\ x{-}ray \textbf{ determines } \{armed(1), armed(2)\} \end{array} \right\}$$

- No conformant plan for

$$\mathcal{P}_1 = (\mathcal{D}_2, \{\neg clogged\}, \{\neg armed(1), \neg armed(2)\})$$

What is a plan in the presence of sensing actions?

- Conditional Plans: take into account contingencies that may arise
  - If a is a non-sensing action and $\langle\beta\rangle$ is a conditional plan then $\langle a, \beta\rangle$ is a conditional plan
  - If a is a sensing action that senses literals $l_1, \ldots, l_n$, and $\langle\beta_i\rangle$ is a conditional plan then

$$\left\langle a, \textbf{cases}\begin{pmatrix} l_1 \rightarrow \beta_1 \\ \ldots \\ l_n \rightarrow \beta_n \end{pmatrix}\right\rangle$$

  is a conditional plan

Example of Conditional Plan

$$\left\langle x\!-\!ray, \textbf{cases}\begin{pmatrix} armed(1) \rightarrow dunk(1) \\ armed(2) \rightarrow dunk(2) \end{pmatrix} \right\rangle$$

is a solution of

$$\mathcal{P}_1 = (\mathcal{D}_2, \{\neg clogged\}, \{\neg armed(1), \neg armed(2)\})$$

How to reason about sensing actions?

- Must take into account different outcomes of sensing actions
  - Transition function: $\mathit{Actions} \times \mathit{Partial\ States} \rightarrow 2^{\mathit{Partial\ States}}$

How to reason about sensing actions?

- Must take into account different outcomes of sensing actions
  - Transition function: *Actions* $\times$ *Partial States* $\rightarrow$ $2^{\textit{Partial States}}$
- For each $A \in \{ph, pc\}$, we define a transition function Φ_S^A as follows
  - for a non-sensing action a, Φ_S^A is the same as Φ^A
  - for a sensing action a, each partial state in Φ_S^A corresponds to a literal that is sensed by a
- Result in four different approximations of $\mathcal{AL}_\mathcal{K}$ domain descriptions

How to reason about sensing actions?

- Must take into account different outcomes of sensing actions
  - Transition function: *Actions* $\times$ *Partial States* $\rightarrow$ $2^{\textit{Partial States}}$
- For each $A \in \{ph, pc\}$, we define a transition function Φ_S^A as follows
  - for a non-sensing action a, Φ_S^A is the same as Φ^A
  - for a sensing action a, each partial state in Φ_S^A corresponds to a literal that is sensed by a
- Result in four different approximations of $\mathcal{AL_K}$ domain descriptions
- Entailment $\models_S^A$

$$(\mathcal{D}, \delta_0) \models_S^A \varphi \textbf{ after } \alpha$$

if φ is true in every final partial state of the execution of α

How to reason about sensing actions?

- Must take into account different outcomes of sensing actions
  - Transition function: $\textit{Actions} \times \textit{Partial States} \rightarrow 2^{\textit{Partial States}}$
- For each $A \in \{ph, pc\}$, we define a transition function Φ^A_S as follows
  - for a non-sensing action a, Φ^A_S is the same as Φ^A
  - for a sensing action a, each partial state in Φ^A_S corresponds to a literal that is sensed by a
- Result in four different approximations of $\mathcal{AL}_{\mathcal{K}}$ domain descriptions
- Entailment $\models^A_S$

$$(\mathcal{D}, \delta_0) \models^A_S \varphi \textbf{ after } \alpha$$

  if φ is true in every final partial state of the execution of α
- Properties
  - Φ^A_S can be computed in polynomial time

How to reason about sensing actions?

- Must take into account different outcomes of sensing actions
  - Transition function: $\textit{Actions} \times \textit{Partial States} \rightarrow 2^{\textit{Partial States}}$
- For each $A \in \{ph, pc\}$, we define a transition function Φ^A_S as follows
  - for a non-sensing action a, Φ^A_S is the same as Φ^A
  - for a sensing action a, each partial state in Φ^A_S corresponds to a literal that is sensed by a
- Result in four different approximations of $\mathcal{AL_K}$ domain descriptions
- Entailment $\models^A_S$

$$(\mathcal{D}, \delta_0) \models^A_S \varphi \textbf{ after } \alpha$$

  if φ is true in every final partial state of the execution of α
- Properties
  - Φ^A_S can be computed in polynomial time
  - the polynomial-length conditional planning: NP-complete

$\mathcal{AL}_\mathcal{K}$ Approximations

Definition

- If a is not executable in δ then
$$\Phi^A_S(a, \delta) = \emptyset$$
- If a is a non-sensing action then
$$\Phi^A_S(a, \delta) = \begin{cases} \emptyset & \text{if } \Phi^A(a, \delta) \text{ is consistent} \\ \{\Phi^A(a, \delta)\} & \text{otherwise} \end{cases}$$
- If a is a sensing action associated with
$$a \textbf{ determines } \theta$$
then
$$\Phi^A_S(a, \delta) = \{Cl_\mathcal{D}(\delta \cup \{g\}) \mid g \in \theta \text{ and } Cl_\mathcal{D}(\delta \cup \{g\}) \text{ is consistent}\}$$

Application in Conditional Planning

- Conditional Planning Problem: $\mathcal{P} = (\mathcal{D}, \delta_0, \mathcal{G})$
  A solution of $\mathcal{P}$ is a conditional plan α such that
  $$(\mathcal{D}, \delta_0) \models^P \mathcal{G} \textbf{ after } \alpha$$

Application in Conditional Planning

- Conditional Planning Problem: $\mathcal{P} = (\mathcal{D}, \delta_0, \mathcal{G})$
  A solution of $\mathcal{P}$ is a conditional plan α such that
  $$(\mathcal{D}, \delta_0) \models^P \mathcal{G} \textbf{ after } \alpha$$
- ASCP:
  - Implemented in logic programming
  - Approximation: Φ_S^{pc}
  - Can generate both concurrent plans and sequential plans
  - Soundness and completeness of ASCP are proved
  - Competitive with some other conditional planners

Experiments

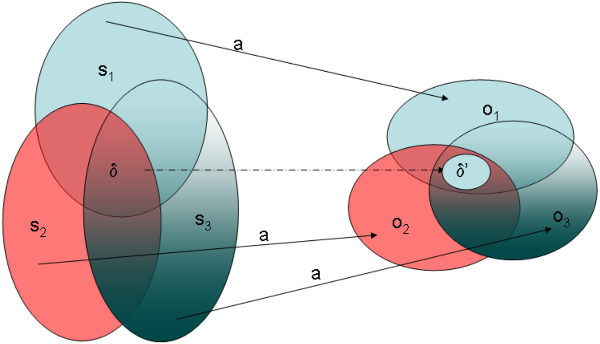
Problem	Min. Plan	ASCP		SGP	POND	MBP
		cmodels	smodels			
BTS1(4)	4x4	0.808	1.697	0.22	0.189	0.048
BTS1(6)	6x6	5.959	83.245	2.44	0.233	0.055
BTS1(8)	8x8	25.284	-	24.24	0.346	0.076
BTS1(10)	10x10	85.476	-	-	0.918	0.384
BTS2(4)	4x4	1.143	3.858	0.32	0.198	0.067
BTS2(6)	6x6	19.478	1515.288	3.23	0.253	2.163
BTS2(8)	8x8	245.902	-	25.5	0.452	109.867
BTS2(10)	10x10	345.498	-	-	1.627	178.823
BTS3(4)	4x4	1.099	5.329	0.44	0.195	1.93
BTS3(6)	6x6	7.055	-	3.89	0.258	147.76
BTS3(8)	8x8	56.246	-	28.41	0.549	-
BTS3(10)	10x10	248.171	-	-	2.675	-
BTS4(4)	4x4	1.696	3.556	0.64	0.191	-
BTS4(6)	6x6	13.966	149.723	4.92	0.264	-
BTS4(8)	8x8	115.28	-	30.34	0.708	-
BTS4(10)	10x10	126.439	-	-	4.051	-

Table: Performance of ASCP on the Bomb domains

Lessons Learned

1. Study in reasoning about actions and changes might provide useful ways for dealing with planning in complex domains
2. Approximations can compensate for the inaccuracy of heuristics
3. Approximations can be useful when the computation of the next state is more complicated
4. Completeness conditions can be used to deal with sensing actions in conditional planners: deciding when to execute a sensing action?

Intuition

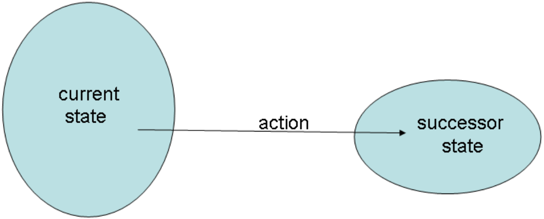


$\delta' = \Phi^0(a, \delta) \subseteq \Phi(a,s_1) \cap \Phi(a,s_2) \cap \Phi(a,s_3) = \Omega$

$l \in \Omega \backslash \delta'$ ➔ l ‘depends’ on something in $s_1\backslash\delta$ or $s_2\backslash\delta$ or $s_3\backslash\delta$

Return

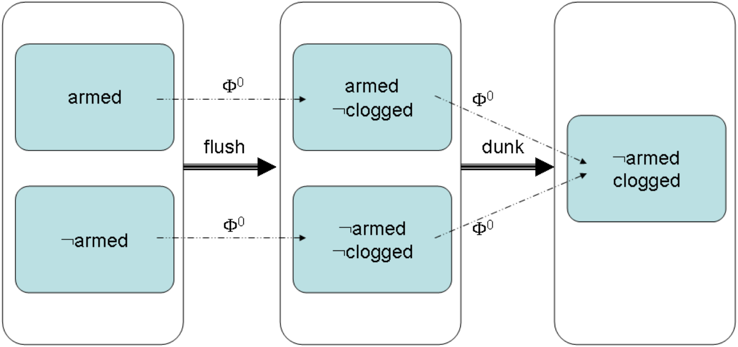
Illustration



1. *Representation*: how to represent actions and their effects?
2. *Reasoning*: how to compute the successor state(s)?

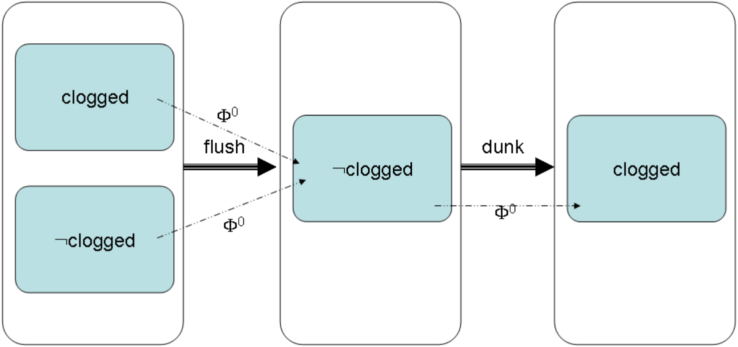
Return

Splitting $\emptyset$ to $\{armed\}$ and $\{\neg armed\}$ works



Return

Splitting $\emptyset$ to $\{clogged\}$ and $\{\neg clogged\}$ does not work



Lack of information about armed prevents the 0-approxiamtion in reaching the conclusion

Return

Acknowledgment

Thanks to Tu Phan for helping with the preparation of the slides.

References I

- Baral, C., Kreinovich, V., & Trejo, R. 2000.
  Computational complexity of planning and approximate planning in the presence of incompleteness.
  Artificial Intelligence, **122**, 241–267.

- Etzioni, O., Golden, K., & Weld, D. 1996.
  Sound and Efficient Closed-World Reasoning for Planning.
  Artificial Intelligence, **89**, 113–148.

- Fikes, R., & Nilson, N. 1971.
  STRIPS: A new approach to the application of theorem proving to problem solving.
  Artificial Intelligence, **2**(3–4), 189–208.

- Gelfond, M., & Lifschitz, V. 1993.
  Representing actions and change by logic programs.
  Journal of Logic Programming, **17**(2,3,4), 301–323.

References II

- Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., & Wilkins, D. 1998.
  PDDL — The Planning Domain Definition Language. Version 1.2.
  Tech. rept. CVC TR98003/DCS TR1165. Yale Center for Comp, Vis and Ctrl.
- Goldman, R., & Boddy, M. 1994.
  Representing uncertainty in simple planners.
  Pages 238–245 of: KR 94.
- Kowalski, R., & Sergot, M. 1986.
  A logic-based calculus of events.
  New Generation Computing, **4**, 67–95.
- Liberatore, P. 1997.
  The Complexity of the Language *A*.
  Electronic Transactions on Artificial Intelligence, **1**(1–3), 13–38.

References III

McCarthy, J. 1959.
Programs with common sense.
Pages 75–91 of: Proceedings of the Teddington Conference on the Mechanization of Thought Processes.
London: Her Majesty's Stationery Office.

McCarthy, J., & Hayes, P. 1969.
Some philosophical problems from the standpoint of artificial intelligence.
Pages 463–502 of: Meltzer, B., & Michie, D. (eds), *Machine Intelligence*, vol. 4.
Edinburgh: Edinburgh University Press.

References IV

Petrick, Ronald P. A., & Bacchus, Fahiem. 2004.
Extending the Knowledge-Based Approach to Planning with Incomplete Information and Sensing.
Pages 2–11 of: Proceedings of the Sixth International Conference on Automated Planning and Scheduling, 2004.

Reiter, R. 2001.
KNOWLEDGE IN ACTION: Logical Foundations for Describing and Implementing Dynamical Systems.
MIT Press.

Son, T.C., & Baral, C. 2001.
Formalizing sensing actions - a transition function based approach.
Artificial Intelligence, **125**(1-2), 19–91.

References V

Son, Tran Cao, & Tu, Phan Huy. 2006.
On the Completeness of Approximation Based Reasoning and Planning in Action Theories with Incomplete Information.
Pages 481–491 of: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning.

Son, Tran Cao, Tu, Phan Huy, Gelfond, Michael, & Morales, Ricardo. 2005a.
An Approximation of Action Theories of *AL* and its Application to Conformant Planning.
Pages 172–184 of: Proceedings of the the 7th International Conference on Logic Programming and NonMonotonic Reasoning.

References VI

- Son, Tran Cao, Tu, Phan Huy, Gelfond, Michael, & Morales, Ricardo. 2005b.
  Conformant Planning for Domains with Constraints — A New Approach.
  Pages 1211–1216 of: Proceedings of the the Twentieth National Conference on Artificial Intelligence.

- Thiebaux, S., Hoffmann, J., & Nebel, B. 2003.
  In Defense of PDDL Axioms.
  In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03).

References VII

Thielscher, M. 2000 (Oct.).
The Fluent Calculus: A Specification Language for Robots with Sensors in Nondeterministic, Concurrent, and Ramifying Environments.
Tech. rept. CL-2000-01. Computational Logic Group, Department of Computer Science, Dresden University of Technology.

Turner, H. 2002.
Polynomial-length planning spans the polynomial hierarchy.
Pages 111–124 of: Proc. of Eighth European Conf. on Logics in Artificial Intelligence (JELIA'02).

Tu, P.H. 2007.
Reasoning and Planning with Incomplete Information in the Presence of Static Causal Laws.
Ph.D Dissertation, NMSU 2007.