Chapter MK:VI

VI. Planning and Configuration

- Agent Systems
- Deductive Reasoning Agents
- Planning Language

- Planning Algorithms
- State-Space Planning
- Plan Space Planning
- HTN Planning
- Complexity of Planning Problems
- Erweiterungen

- Konfigurierungsproblemstellung
- Konfigurierungsansätze

Erweiterungen

Kontrolle der Planausführung

Dreiecktabelle:

- Repräsentation von Plan und Voraussetzungen
- Kontrolle der auszuführenden Aktion eines Planes
  - Welcher maximale Block in der Dreieckstabelle ist im aktuellen Zustand erfüllt?
  - Wähle die *aktive Aktion*, die zu dem Block gehört, der dem Effekt des Planes am nächsten ist.
- Kontrolle des Wiederaufsetzens von Plänen
  - Unvorhergesehene Änderungen in der Umgebung können das Erreichen von Zielen durch strikte Ausführung des Planes verhindern.
  - Auswahl der aktiven Aktionen berücksichtigt Änderungen.

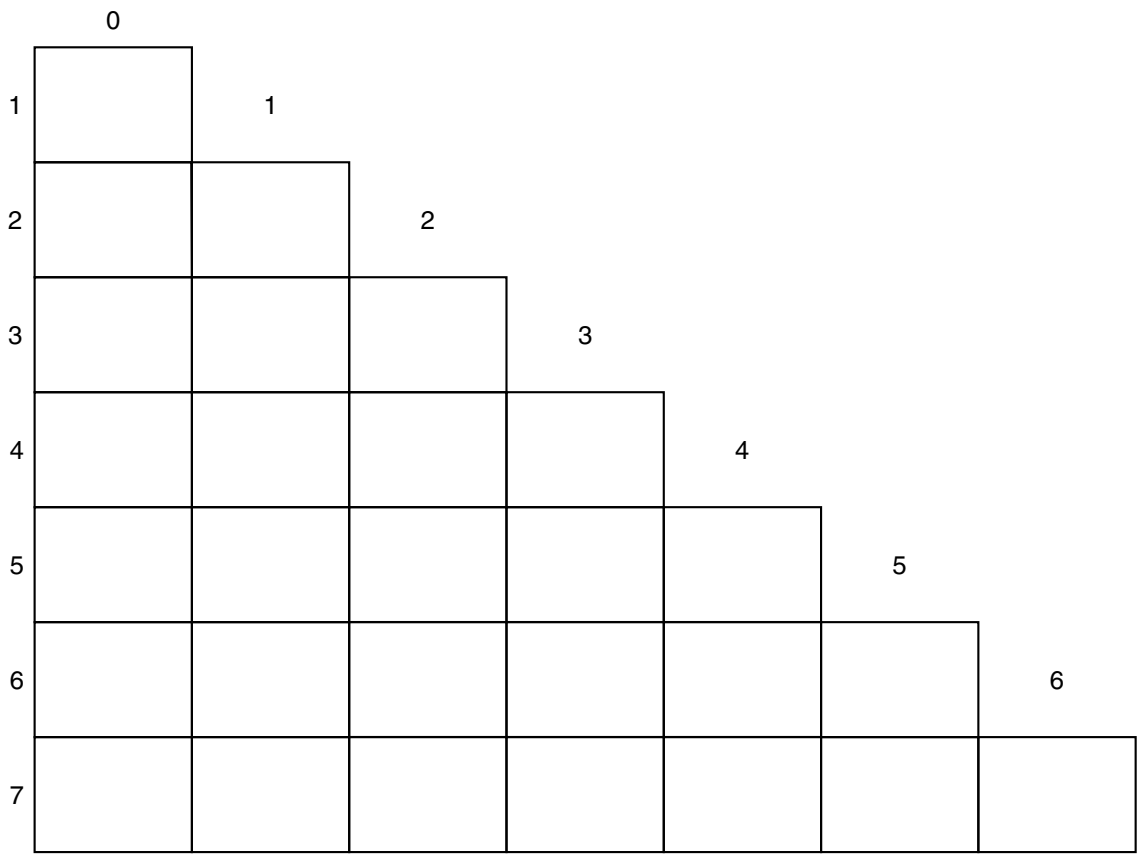
Schränkt man STRIPS nicht so weit ein wie hier, sind die maximalen Blöcke in der Dreieckstabelle nicht unbedingt (Teil-) Zustände, sondern können bei passender Instatiierung von Variablen (Teil-) Zustände beschreiben.

Bemerkungen:

- Die Dreiecktabelle ist für eine Spezialisierung von STRIPS geschaffen worden, bei der die negativen Effekte (zulöschende Atome des Zustands) und die Vorbedingungen einer Aktion übereinstimmen. Diese Spezialisierung stellt keine Einschränkung dar, da "‘unverbrauchte"’ Bedingungen als positive Effekte modelliert werden können.

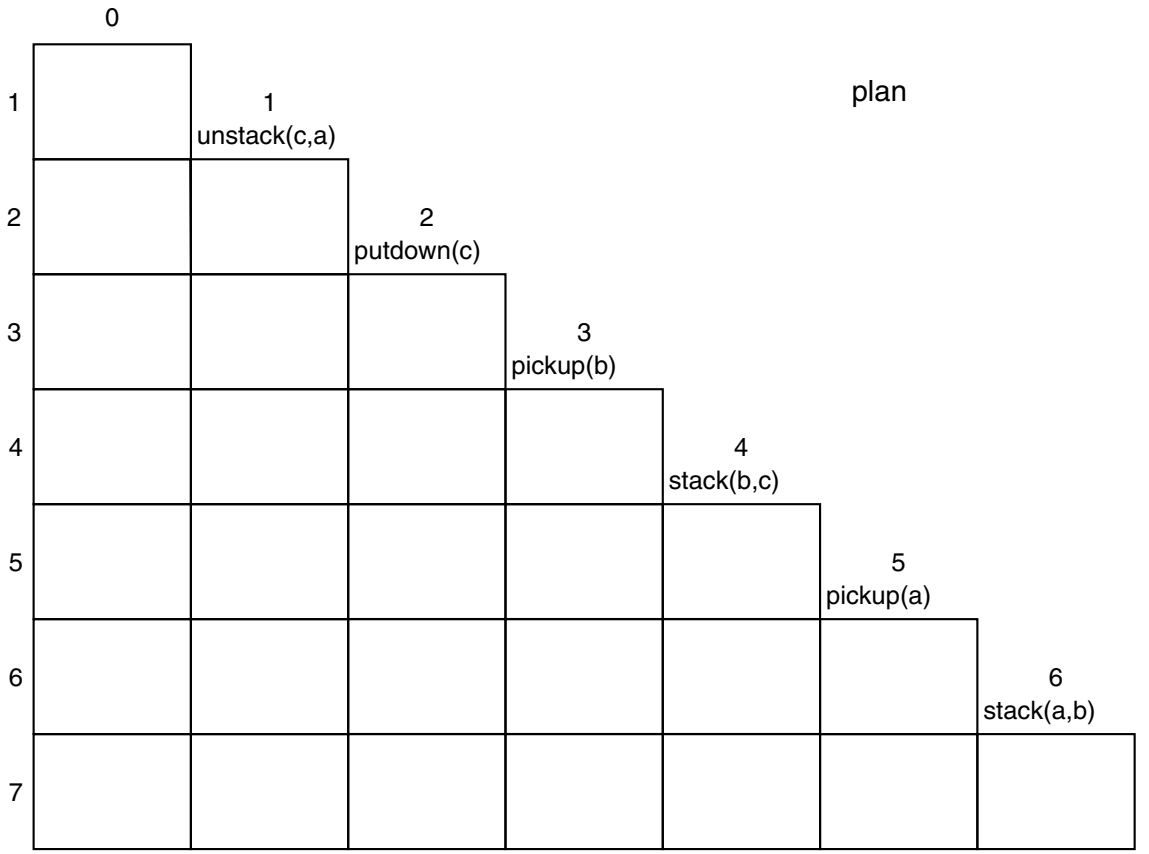
Erweiterungen

Beispiel Dreiecktabelle in Blocks World



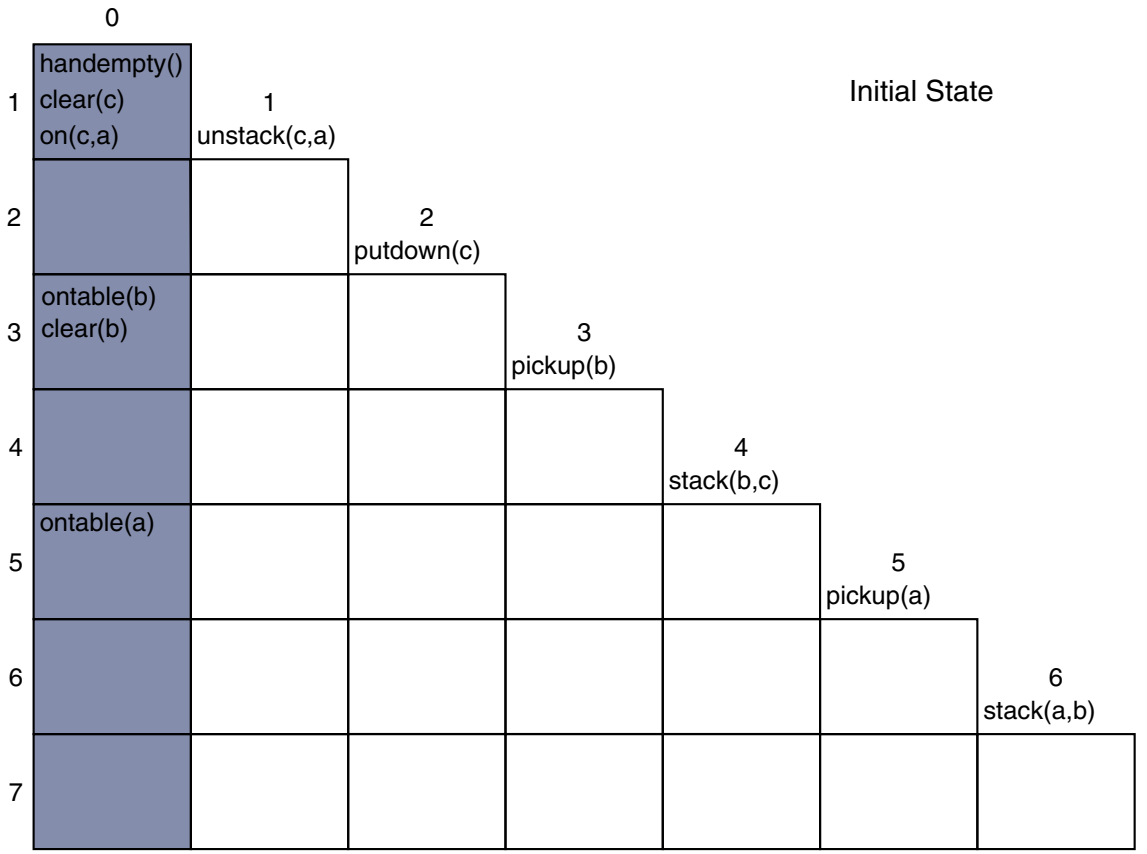
Erweiterungen

Beispiel Dreiecktabelle in Blocks World (continued)



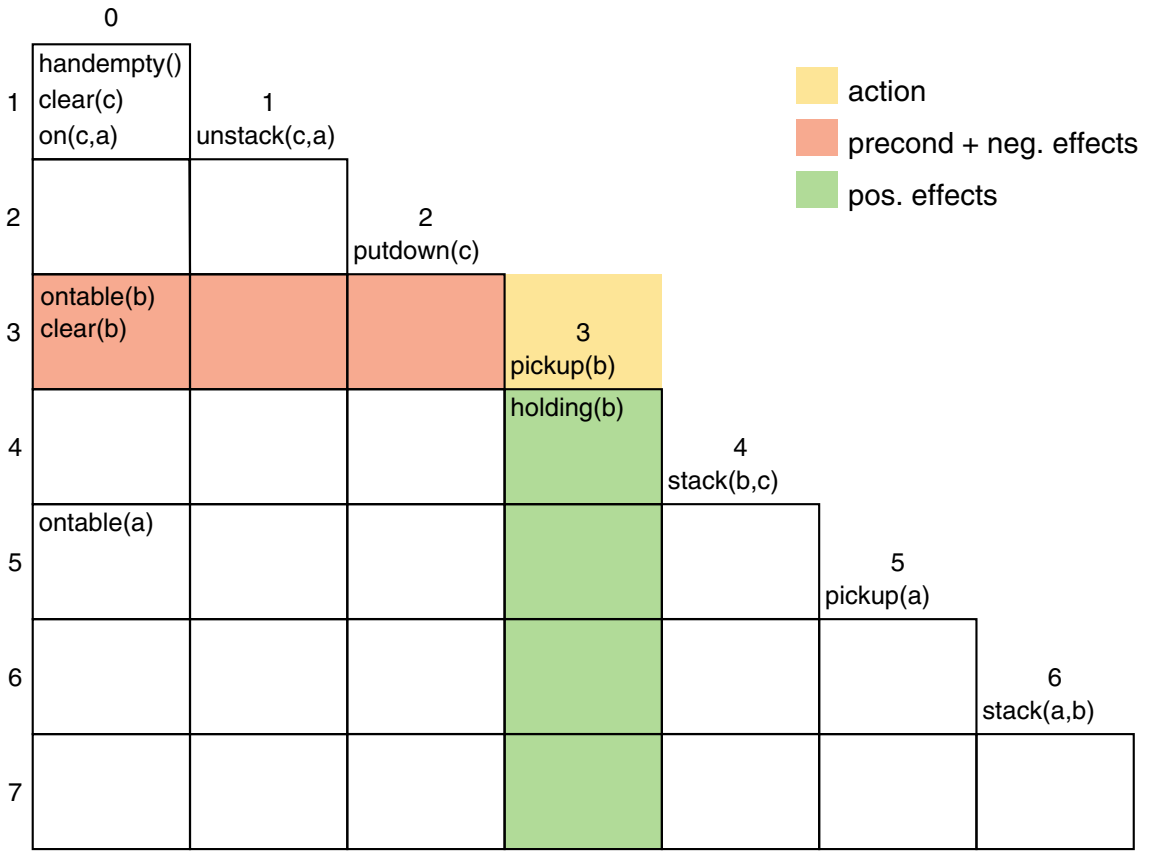
Erweiterungen

Beispiel Dreiecktabelle in Blocks World (continued)



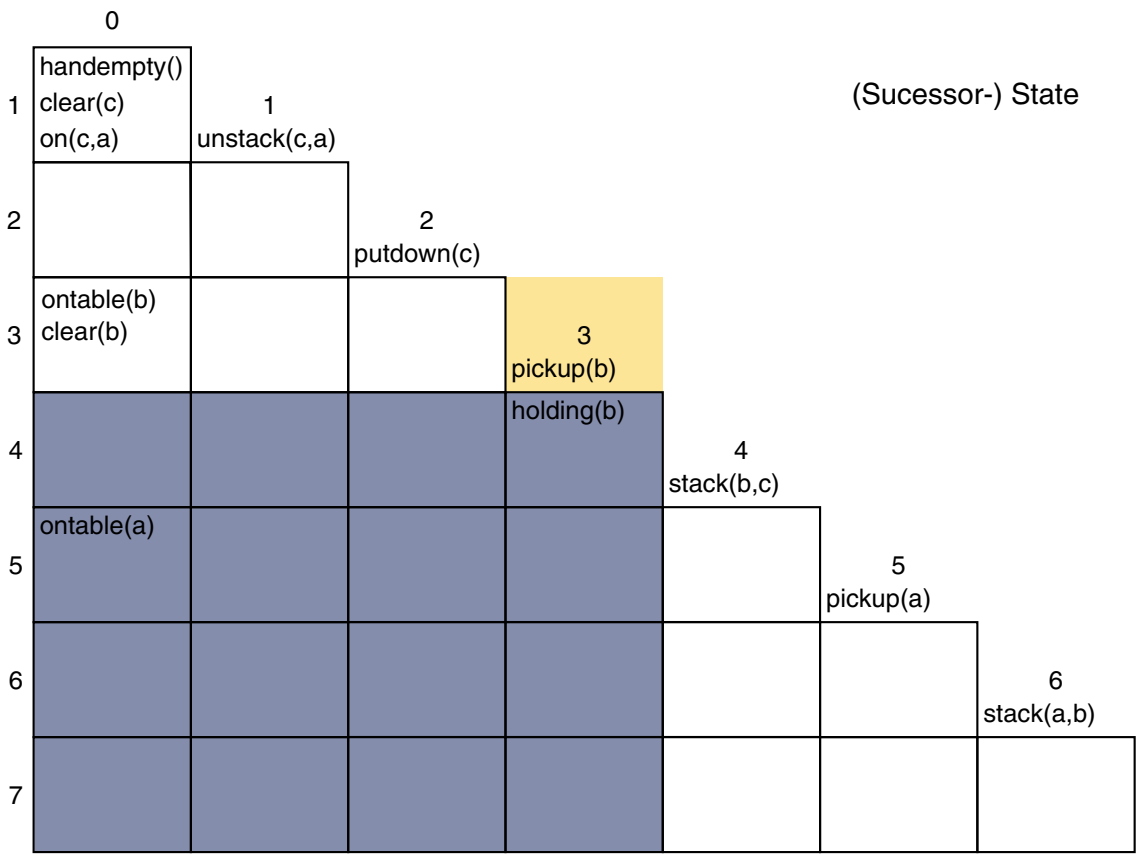
Erweiterungen

Beispiel Dreiecktabelle in Blocks World (continued)



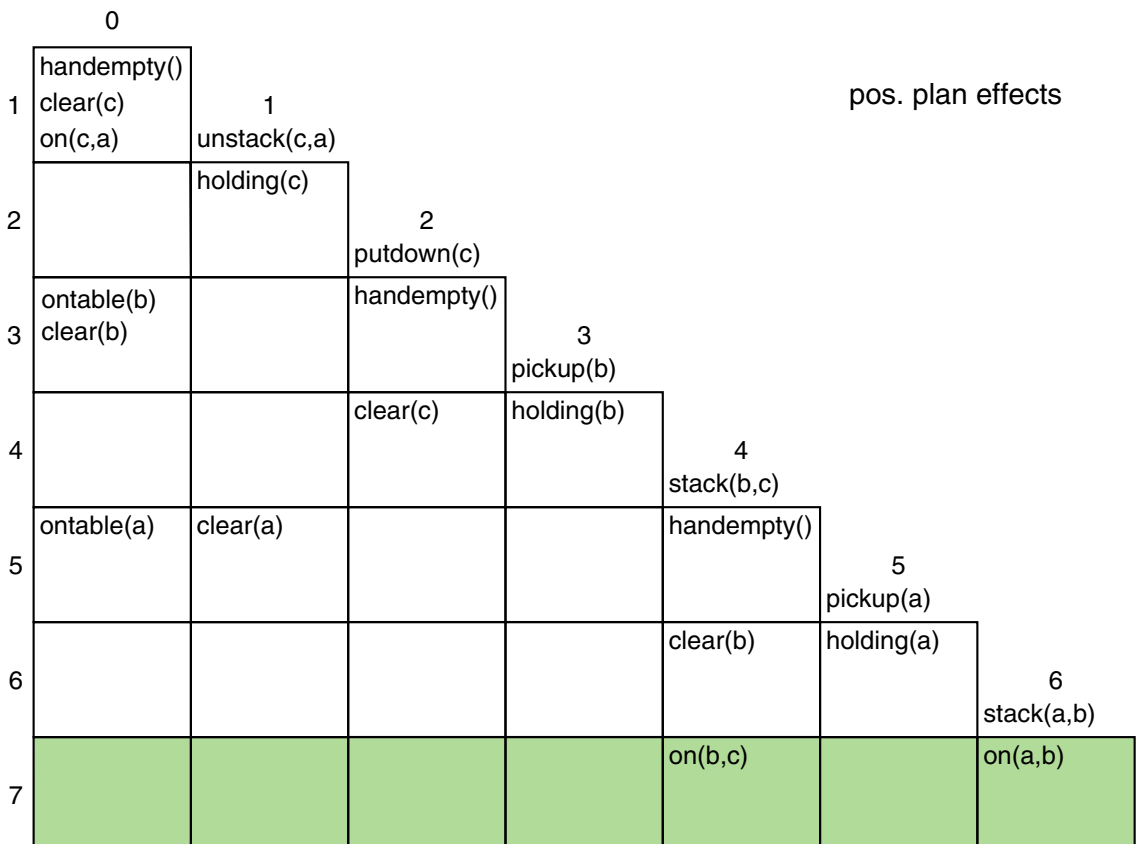
Erweiterungen

Beispiel Dreiecktabelle in Blocks World (continued)



Erweiterungen

Beispiel Dreiecktabelle in Blocks World (continued)



Erweiterungen

STRIPS Grenzen

- Statische Umgebung

  Veränderungen der Welt können nur über externe Agenten modelliert werden. Bewegung (Newtons erstes Gesetz) wird nicht modelliert, sondern nur Veränderungen.

- Sequentialität

  Simultane Aktionen durch mehrere Agenten können nicht modelliert werden. Aktionen werden als Einzelaktionen hintereinander ausgeführt und so im Modell eventuell Zustände erreicht, die in der Realität nicht eintreten.

- Universalität

  Nicht die Ausführung, sondern die Ergebnisse von Aktionen werden im Modell beschrieben. Die durch die Ausführung möglicherweise auftretenden Probleme müssen als eigene Aktionen modelliert werden.

Bemerkungen:

- Alle STRIPS Modellierungen können auch im Situation Calculus vorgenommen werden.
- Der Situation Calculus ist mächtiger als die STRIPS Sprache. (Beispiel: Lege alles ab.)
- Die Einbettung von STRIPS in den Situation Calculus kann durch ein Prädikat *holds*(C, S) (C ist wahr in Situation S) beschrieben werden (Prolog-ähnliche Notation).

  C wird von der Aktion A erzeugt:

  $$\textit{holds}(C, \textit{result}(A, W)) \leftarrow \textit{preconditions}(A, PL) \wedge \textit{hold\_sall}(PL, W) \wedge \textit{add\_list}(A, AL) \wedge \textit{member}(C, AL)$$

  C war zuvor wahr und ist nicht in der Löschliste von A:

  $$\textit{holds}(C, \textit{result}(A, W)) \leftarrow \textit{preconditions}(A, PL) \wedge \textit{holds\_all}(PL, W) \wedge \textit{delete\_list}(A, DL) \wedge \textit{notin}(C, DL) \wedge \textit{holds}(C, W)$$

- *holds*(C, S) beschreibt in der obigen Version nur, wann eine Aussage C nach einer Aktion gilt. Zusätzlich müsste man prüfen, ob C durch Regeln hergeleitet werden kann oder ob C im Anfangszustand gilt.
- Die Realisierung von *holds_all*(PL, W) erfolgt in üblicher Weise durch sukzessives Prüfen der Liste:

  $$\textit{holds\_all}([P|PL_R], W) \leftarrow \textit{holds}(P, W) \wedge \textit{holds\_all}(PL_R, W)$$
  $$\textit{holds\_all}([\,], W)$$

  PL_R soll die restlichen Goals aus PL außer P enthalten.

Erweiterungen

Beispiel: STRIPS Grenzen bei impliziten Effekten (Ferber)

- Zustand
  $s_1 = \{\mathit{Glass}(g), \mathit{Empty}(g), \mathit{Tap}(t), \mathit{Closed}(t), \mathit{Under}(g,t)\}$
- Operatoren

  operator : $\mathit{openTap}(x)$

  precond : $\mathit{Tap}(x), \mathit{Closed}(x)$

  effects : $\neg\mathit{closed}(T), \mathit{Flows}(x, \mathit{water}), \mathit{open}(x)$

  ...
- Folgezustand der Aktion $\mathit{openTap}(t)$
  $s_2 = \{\mathit{Glass}(g), \mathit{Empty}(g), \mathit{Tap}(t), \mathit{Open}(t), \mathit{Flows}(t, \mathit{water}), \mathit{Under}(g,t)\}$

➜ Das Glas bleibt leer!

Erweiterungen

Beispiel: STRIPS Grenzen bei impliziten Effekten (Ferber) (continued)

- Zustand
  $s_1 = \{\mathit{Glass}(g), \mathit{Empty}(g), \mathit{Tap}(t), \mathit{Closed}(t), \mathit{Under}(g,t)\}$
- Folgezustand
  $s_2 = \{\mathit{Glass}(g), \mathit{Empty}(g), \mathit{Tap}(t), \mathit{Open}(t), \mathit{Flows}(t, \mathit{water}), \mathit{Under}(g,t)\}$
- Notwendige zweite Aktion $\mathit{fillGlassUnderTap}(x,y)$

  operator : $\mathit{fillGlassUnderTap}(x,y)$

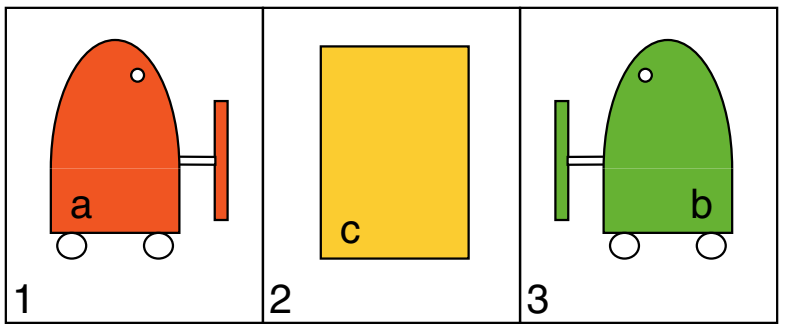
  precond : $\mathit{Tap}(x), \mathit{Glass}(y), \mathit{Under}(y,x), \mathit{Flows}(t, \mathit{water}), \mathit{Empty}(y)$

  effects : $\neg\mathit{Empty}(y), \mathit{Filled}(y, \mathit{water})$

➜ Keine Repräsentation von Kausalität zwischen Aktionen in STRIPS!

Erweiterungen

Beispiel: STRIPS Erweiterung auf Multiagentensysteme (Ferber)



- Zustand

  $s1 = \{\mathit{Cube}(c), \mathit{Agent}(a), \mathit{Agent}(b), \mathit{At}(c, 2), \mathit{At}(a, 1), \mathit{At}(b, 3)\}$

➜ Aktion $=$ Einflüsse und Reaktion

Erweiterungen

Beispiel: STRIPS Erweiterung auf Multiagentensysteme (Ferber) (continued)

Aktion = Einflüsse und Reaktion

- Einfluss
  Menge von Formeln, die den Aktionsversuch eines Agenten im Hinblick auf den aktuellen Zustand beschreibt.
- Operatoren

  operator : *move*(x, y, d)

  precond : *Agent*(x), *Cube*(y)

  postcond : *Push*(y, d)

  ...

  postcond beschreibt der Einfluss des Operators auf die Umgebung.

➜ postcond(*move*$(a, c, +1)) = \{$*Push*$(c, +1)\}$

Erweiterungen

Beispiel: STRIPS Erweiterung auf Multiagentensysteme (Ferber) (continued)

Aktion $=$ Einflüsse und Reaktion

- Zusammenfassung der Einflüsse gleichzeitiger Aktionen

  $\mathit{Exec} : (A, \|) \times S \to I$

  $$\mathit{Exec}(a, s) = \begin{cases} \mathit{postcond}(a) & \text{falls } \mathit{precond}(a) \text{ erfüllt in } s \\ \emptyset & \text{sonst} \end{cases}$$

  $$\mathit{Exec}(a_1 \parallel a_2, s) = \mathit{Exec}(a_1, s) \cup \mathit{Exec}(a_2, s)$$

➜ $\text{Exec}(\mathit{move}(a, c, +1) \parallel \mathit{move}(b, c, -1)) = \{\mathit{Push}(c, +1)\} \cup \{\mathit{Push}(c, -1)\}$

Erweiterungen

Beispiel: STRIPS Erweiterung auf Multiagentensysteme (Ferber) (continued)

Aktion $=$ Einflüsse und Reaktion

- Bestimmung des Gesamteffektes *React* : $S \times I \rightarrow S$

  React :

  precond : $\mathit{At}(c, x)$

  influence : $\mathit{Push}(c, d)$

  effects : $\neg\mathit{At}(c, x), \mathit{At}(c, x + \sum d)$

  wobei $\sum d$ die Kombination aller dieser Einflüsse bezeichnet.

  ...

➜ $s2 = \{\mathit{Cube}(c), \mathit{Agent}(a), \mathit{Agent}(b), \mathit{At}(c, 2), \mathit{At}(a, 1), \mathit{At}(b, 3)\}$

Erweiterungen

ADL: Erweiterung von STRIPS

	STRIPS	ADL
Zustand	pos. (und neg.) Grundliterale $p(a) \wedge q(b)$	pos. und neg. Grundliterale $p(a) \wedge q(b) \wedge \neg r(c)$
World Ass.	Closed World Assumption (Nicht genanntes ist falsch.)	Open World Assumption (Nicht genanntes ist unbekannt.)
Effekte	Löschen positiver Literale Setzen positiver Literale	Löschen positiver/negativer Literale Setzen positiver/negativer Literale bedingte Effekte *when* $p : q$
Ziele	Grundliterale $p(a)$ konjunktiv verknüpft $p(a) \wedge \neg q(b)$	existenzquantifiz. Literale $\exists X(p(X) \wedge q(X))$ konj. und disj. verknüpft $p(a) \wedge (q(b) \vee \neg r(c))$
Gleichheit	(unterstützt)	unterstützt $Y \neq a$
Sorten	(unterstützt)	unterstützt a : *block*

Erweiterungen

PDDL: Erweiterung von ADL (McDermott)

- Numerische Variablen:
  Zahlenwerte statt Wahrheitswert, Testen und Verändern der Werte

```
(define (domain jug-pouring)
  (:requirements :typing :fluents)
  (:types jug)
  (:functions
    (capacity ?j - jug)
    (amount ?j - jug))

(:action pour
  :parameters (?jug1 ?jug2 - jug)
  :precondtion (>= (capacity ?jug2)
      (+ (amount ?jug1) (amount ?jug2)))
  :effect (and (assign (amount ?jug1) 0)
      (increases (amount ?jug2) (amount ?jug1)))

)
```

- Komplexe Zielfunktionen

Erweiterungen

PDDL: Erweiterung von ADL (McDermott) (continued)

- Zeitabhängige Aktionen:
  Effekte können einige Zeit nach dem Ausführen der Aktion (die keine Zeit benötigt) eintreten.

```
(:durative-action fly
  :parameters (?p - plane ?t - traveller
      ?a ?b - location)
  :duration (= ?duration (flight-time ?a ?b))
  :condition (and (at start (at ?p ?a)) (at start (at ?t ?a))
        (over all (aboard ?t ?p)) (over all (inflight ?p))
        (at start (>= (fuel-level ?p)
          (* (flight-time ?a ?b) (consumption-rate ?p)))))
  :effect (and (at start (aboard ?t ?p))
      (at start (not (at ?t ?a)))   (at start (not (at ?p ?a)))
      (at start (inflight ?p))
      (at end (at ?p ?b))   (at end (at ?t ?b))
      (at end (not (inflight ?p)))
      (at end (not (aboard ?t ?p)))
      (at end (decrease (fuel-level ?p)
        (* (flight-time ?a ?b) (consumption-rate ?p))))
    )

)
```

Erweiterungen

PDDL: Erweiterung von ADL (McDermott) (continued)

- Zeitabhängige Aktionen:
  diskrete Zeit vs. kontinuierliche Zeit

```
(decrease (fuel-level ?p) (* #t (consumption-rate ?p)))
```

  im Gegensatz zu

```
(at end (decrease (fuel-level ?p)
      (* (flight-time ?a ?b) (consumption-rate ?p))))
```