

# Grundlagen der Künstlichen Intelligenz

## **9 Handlungsplanung**

---

Planen im Situationskalkül, STRIPS-Formalismus,  
Nichtlineares Planen

*Volker Steinhage*

# Inhalt

---

- Handlungsplanung vs. Problemlösen
- Planen im Situationskalkül
- Der STRIPS-Formalismus
- Nichtlineares Planen
- Der POP-Algorithmus
- Variablenbindungen

# Handlungsplanung

---

Aufgabenstellung der Handlungsplanung:

- Gegeben:
  - eine *initiale Situation*,
  - eine Beschreibung der *Zielbedingungen* und
  - eine Menge von *möglichen Aktionen*.
- Gesucht:
  - ein *Handlungsplan* als *Sequenz von Aktionen*, die die initiale Situation in eine Situation überführt, in der die Zielbedingungen gelten.

# Ein einfacher, planender Agent

**function** SIMPLE-PLANNING-AGENT(*percept*) **returns** an *action*

**static:** *KB*, a knowledge base (includes action descriptions)

*p*, a plan, initially *NoPlan*

*t*, a counter, initially 0, indicating time

**local variables:** *G*, a goal

*current*, a current state description

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))

*current*  $\leftarrow$  STATE-DESCRIPTION(*KB*, *t*)

**if** *p* = *NoPlan* **then** \*

*G*  $\leftarrow$  ASK(*KB*, MAKE-GOAL-QUERY(*t*))

*p*  $\leftarrow$  IDEAL-PLANNER(*current*, *G*, *KB*)

**if** *p* = *NoPlan* **or** *p* is empty **then** *action*  $\leftarrow$  *NoOp*

**else**

*action*  $\leftarrow$  FIRST(*p*)

*p*  $\leftarrow$  REST(*p*)

TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))

*t*  $\leftarrow$  *t* + 1

**return** *action*

Vorgehen des Agenten:

- (1) Bestimmung eines Ziels
- (2) Berechnung eines Plans, um dieses Ziel vom aktuellen Zustand aus zu erreichen
- (3) Ausführen des Plans, bis der Zielzustand erreicht ist
- (4) Weiter bei (1)

Bemerkung: Sofern in  $G \leftarrow \text{ASK}(\text{KB}, \text{MAKE-GOAL-QUERY}(t))$  a priori alle Teilziele als eine Konjunktion generiert werden, werden die Teilziele auch bis zum leeren Plan (*p* is empty) abgearbeitet. Sofern nach Abarbeitung eines ersten Zieltes (*p* is empty) eine erneute Zielabfrage  $G \leftarrow \text{ASK}(\text{KB}, \text{MAKE-GOAL-QUERY}(t))$  das nächste Ziel aus der Wissensbasis generieren soll, muss die Bedingung der ersten bedingten Abfrage\* lauten: „**if** *p* = *NoPlan* **or** *p* is empty **then** ...“. Dann kann die Zielgenerierung und Plangenerierung erneut nach Abarbeitung eines Zieltes angestoßen werden.

# Handlungsplanung vs. Problemlösen durch Suche

Wesentlicher Unterschied: Explizite, logikbasierte Repräsentation

- Zustände/Situation: Durch logische Formeln beschriebene Weltzustände vs. geschlossene Datenstrukturen

~ Der Agent kann explizit über Objekte der Anwendungsdomäne sowie ihre Eigenschaften und Beziehungen reflektieren und kommunizieren.

- Zielbedingungen als logische Formeln vs. Zieltest (Black Box)

~ Der Agent kann explizit über Objekte (inkl. ihrer Eigenschaften und Beziehungen) seiner Ziele reflektieren.

- Operatoren: Axiome oder Transformation von Formeln vs. Modifikation von Datenstrukturen (Suchbäumen bzw. -graphen) durch Programme

~ Der Agent kann durch Inspektion der Operatoren Information über die Vorbedingungen und Effekte von Aktionen gewinnen.

# Planen als logische Inferenz (1)

Wir sahen: Handlungsplanung ist mit Hilfe des *Situationskalküls* formalisierbar.

Initialer Zustand (Aussagen über atomaren Formeln):

$$\text{At}(\text{home}, s_0) \wedge \neg \text{Have}(\text{milk}, s_0) \wedge \neg \text{Have}(\text{banana}, s_0) \wedge \neg \text{Have}(\text{drill}, s_0)$$

Operatoren (Nachfolgezustandsaxiome):

$$\forall a, s \text{ Have}(\text{milk}, \text{do}(a, s)) \Leftrightarrow \\ \{a = \text{buy}(\text{milk}) \wedge \text{Poss}(\text{buy}(\text{milk}), s) \vee \text{Have}(\text{milk}, s) \wedge a \neq \text{drop}(\text{milk})\}$$

Zielbedingungen (Anfrage):

$$\exists s \text{ At}(\text{home}, s) \wedge \text{Have}(\text{milk}, s) \wedge \text{Have}(\text{banana}, s) \wedge \text{Have}(\text{drill}, s)$$

Wenn der Initialzustand, alle Vorbedingungen und alle Nachfolgezustandsaxiome gegeben sind, liefert der *konstruktive Beweis* der existentiellen Anfrage einen *Plan*, der das Gewünschte macht.

## Planen als logische Inferenz (2)

---

- Bei *Vorwärtsableitung* (s. Algm. *FOL-FC-ASK*, Vorl. 8) werden vom Startzustand  $s_0$  aus Aktionen  $a$  gemäß ihren Nachfolgezustandsaxiome angewendet.
- Schrittweise entstehen neue Klauseln, die die Erfüllung von Prädikaten wie *Have* oder *At* über den Nachfolgezuständen  $do(a,s)$  formulieren.
- Automatisch sammeln sich in den Termen, die die Nachfolgezustände darstellen, die angewandten Aktionen (durch diese erfolgen ja die Zustandsüberführungen). Für den Lösungspfad ergibt sich im Beispiel folg. Variablenbindung für die Zustandsvariable  $s$ :

$do(go(home),do(buy(drill), do(go(hardware\_store),$   
 $do(buy(banana),do(buy(milk),do(go(super\_market),s_0))))))$ .

- In diesem Zustand sind alle Zielprädikate erfüllt und der *Plan* lautet:

$go(super\_market),buy(milk),buy(banana),go(hardware\_store),(buy(drill),go(home).$

# Planen als logische Inferenz (3)

---

Logische Inferenz führt nicht garantiert zum effizientesten Lösungsweg, weil

- 1) viele für ein Ziel **irrelevante Aktionen** zu untersuchen sind,
- 2) Vorteile der **Problemzerlegung** nicht nutzbar sind.

Grund: Verschachtelung von zustandsändernden Aktionen als Terme in zustandsbeschreibenden Prädikaten ist eine sehr dichte Repräsentation, die eine explizite und effiziente Auswahl der „richtigen“ Aktionen erschwert.

- 3) Unentscheidbarkeit der PL 1

~ **Spezialisiertes Inferenzsystem** für eingeschränkte Repräsentation

~ *Planungsalgorithmus*



# Der STRIPS-Formalismus

---

STRIPS: STanford Research Institute Problem Solver (Planer der 70er Jahre) <sup>(1)</sup>

- *Weltzustand* (inkl. initialer Zustand):
  - Aussagen über *Grundatomen*,
    - *keine Funktionssymbole* außer Konstanten,
    - interpretiert unter *Closed World Assumption* (auch *Standardinterpretation*)<sup>(2)</sup>.
- *Zielbedingungen*: Aussagen über *Grundatomen*.

*Beachte: Keine explizite Zustandsvariable wie im Situationskalkül.*

Es ist immer nur der aktuelle Weltzustand zugreifbar.

---

<sup>(1)</sup> Die mit STRIPS eingeführten Strukturen und Begriffe haben aber auch in aktuellen Forschungen ihre Relevanz.

<sup>(2)</sup> Closed-World-Annahme: Nicht explizit als *wahr* gegebene oder ableitbare Aussagen sind als *falsch* anzunehmen. Erübrigt das explizite Aufnehmen und Verarbeiten von negativen Faktenklauseln.

# STRIPS-Operatoren

Aktionen sind *Tripel*, bestehend aus:

- **Aktionsnamen**: Funktionsname mit Parametern (wie im Situationskalkül).
- **Vorbedingungen**: Konjunktion positiver Literale;  
müssen für Ausführbarkeit der Aktion gelten.
- **Effekte**: Konjunktion positiver und negativer Literale;  
positive Literale werden hinzugenommen (ADD Liste),  
*negative gelöscht* (DEL Liste) ( $\sim$  Behandlung des **Frame-Problems**).

Closed World  
Assumption

Beispiel:

Grafische Darstellung

Op( **Action**: Go(there)

**Precond**:  $\text{At}(\text{here}) \wedge \text{Path}(\text{here}, \text{there})$

**Effect**:  $\text{At}(\text{there}) \wedge \neg \text{At}(\text{here})$

$\text{At}(\text{here}), \text{Path}(\text{here}, \text{there})$

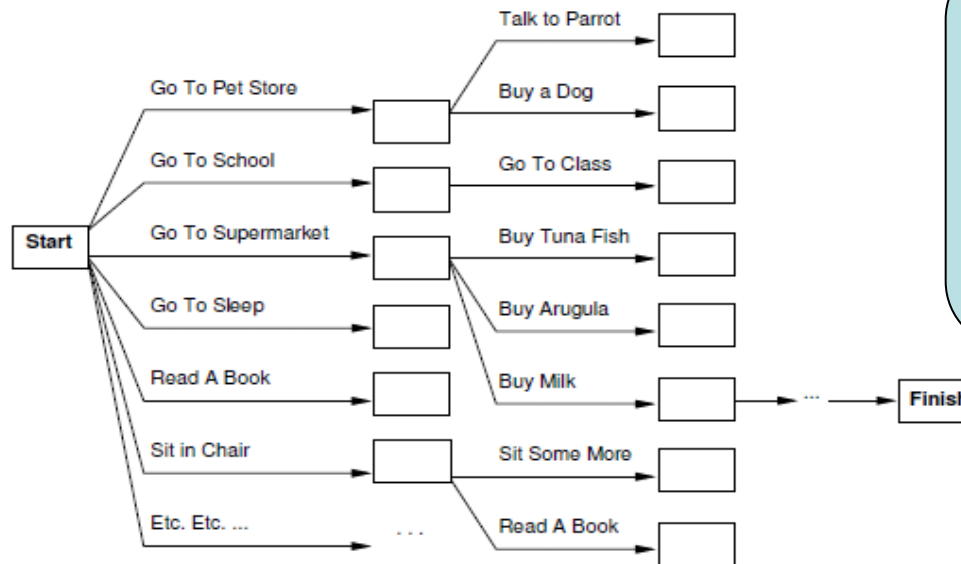
Go(there)

$\text{At}(\text{there}), \neg \text{At}(\text{here})$

# Suchen im Zustandsraum

Wir könnten jetzt durch den Zustandsraum (die Menge aller Zustände) suchen – und damit **Planen auf Suchen reduzieren**.

- Dabei können wir vorwärts suchen → **Progressionsplanung**:



Vorwärts- und Rückwärtssuche sind *vollständig* ordnende Suchverfahren, die Start und Ziel durch vollständig sequentielle Pläne verbinden → kein Ansatz zur Problemzerlegung.  
Effizienter: Teilpläne von Unterproblemen erstellen und diese zu Gesamtplan kombinieren.  
→ Arbeiten im „Raum der Pläne“  
→ *Divide and conquer*

- Alternativ könnten wir ausgehend vom Ziel die Suche rückwärts durchführen → **Regressionsplanung**.

*Möglich*, da die Operatoren genug Information zur Verfügung stellen und *meist effizienter*, da der Verzweigungsfaktor geringer ist.

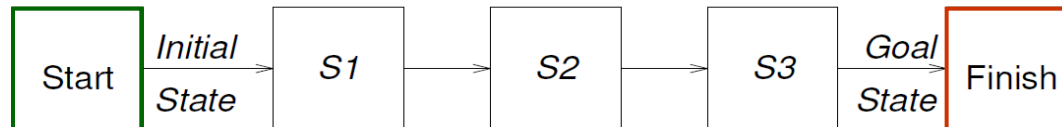
# Suchen im Raum aller Pläne

Statt im Zustandsraum suchen wir jetzt im **Raum aller Pläne**.

- Start ist ein **partieller Plan**, der nur einen **Startschritt** (mit **Startzustand als Effekt**) und einen **Zielschritt** (mit **Zielzustand als Vorbedingung**) enthält:



- Ziel ist ein **vollständiger Plan**, der das gegebene Problem löst:

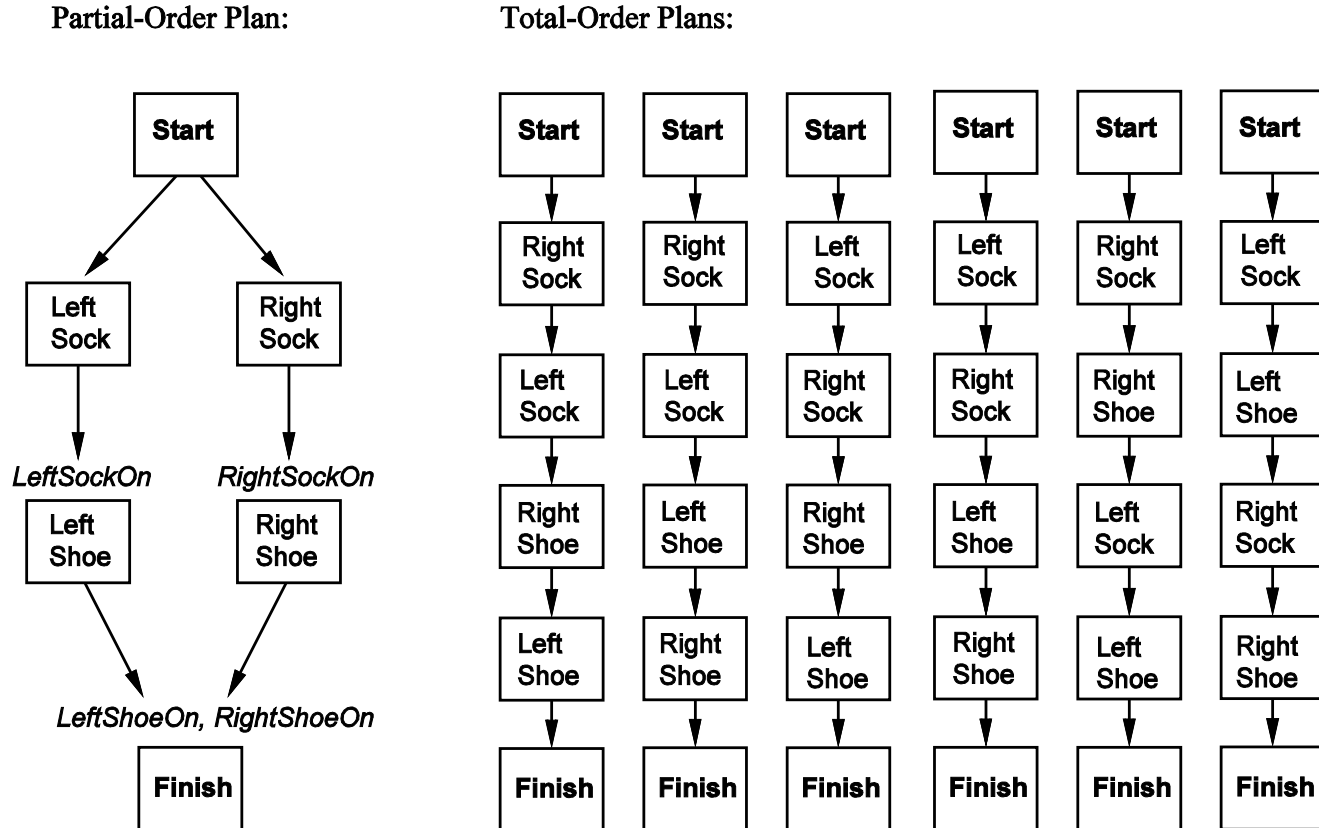


- Operatoren im Planraum:
  - (1) **Verfeinerungsoperatoren** machen den Plan konkreter (mehr Schritte, mehr Einschränkungen der Ordnung, Variablenbelegung, usw.)
  - (2) **Modifikationsoperatoren** modifizieren den Plan (z.B. durch Fehlerbehebung von potentiell fehlerhaften Startplänen)

# Plan = Sequenz von Aktionen?

Oft ist es nicht sinnvoll oder möglich, sich bei der Reihenfolge früh festzulegen

~ *nichtlineare* oder *partiell geordnete Pläne* (*least-commitment planning*)!



Links der partiell geordnete Plan P. Rechts die möglichen Linearisierungen von P durch zusätzliche Ordnungs-Einschränkungen (Ordnungs-Constraints).

# Repräsentation partiell geordneter Pläne

---

- Ein Planschritt = *STRIPS-Operator*
- Ein *Plan* besteht aus
  - 1) Menge von *Planschritten* mit partieller Ausführungsordnung ( $\prec$ ),  
wobei  $S_i \prec S_j$  gdw.  $S_i$  muss vor  $S_j$  ausgeführt werden.
  - 2) Menge von *Variablenbelegungen*  $x = t$ ,  
wobei  $x$  eine Variable und  $t$  eine Konstante oder Variable ist.
  - 3) Menge *kausaler Beziehungen* ( $\rightarrow^c$ )  
 $S_i \rightarrow^c S_j$  bedeutet „ $S_i$  erzeugt die Vorbedingung  $c$  für  $S_j$ “  
(impliziert  $S_i \prec S_j$ ).
- Lösungen für Planungsprobleme müssen *vollständig* und *konsistent* sein.

# Vollständigkeit und Konsistenz

*Vollständiger Plan:* Jede Vorbedingung eines Schritts wird erfüllt:

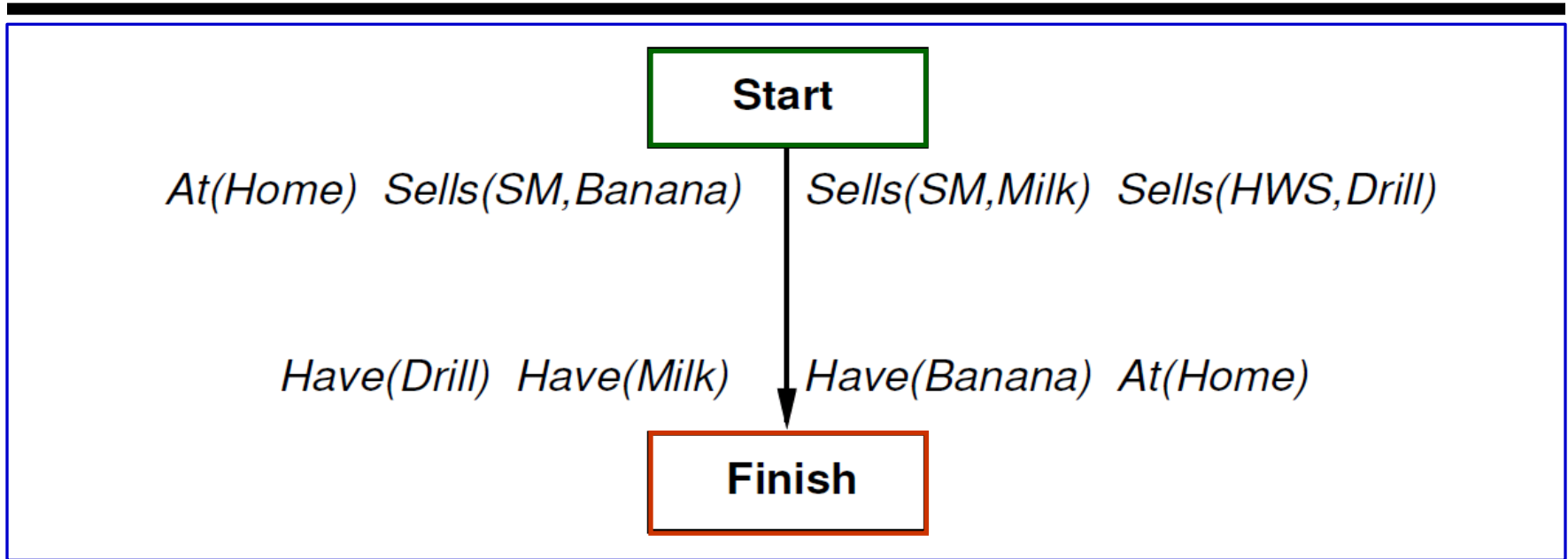
- 1)  $\forall S_j$  mit  $c \in \text{Precond}(S_j)$  gilt:  
 $\exists S_i$  mit  $S_i < S_j$  und  $c \in \text{Effects}(S_i)$  und
- 2) für jede Linearisierung des Plans gilt:  
 $\forall S_k$  mit  $S_i < S_k < S_j$ ,  $\neg c \notin \text{Effect}(S_k)$

*Konsistenter Plan:* widerspruchsfrei

- 1) wenn  $S_i < S_j$ , dann  $S_j \not< S_i$  und
- 2) wenn  $x = A$ , dann  $x \neq B$   
für Variable  $x$  und verschiedene  $A$  und  $B$   
(*Unique Name Assumption*)

Ein *vollständiger, konsistenter Plan* heißt *Lösung* eines Planungsproblems.

# Beispiel



Aktionen:

Op (Action: **Go(there)**,  
Precond:  $\text{At}(\text{here})$ ,  
Effect:  $\text{At}(\text{there}) \wedge \neg \text{At}(\text{here})$ )

Op(Action: **Buy(x)**,  
Precond:  $\text{At}(\text{store}) \wedge \text{Sells}(\text{store}, x)$ ,  
Effect:  $\text{Have}(x)$ )

- Home, Banana, Milk, Drill, SM (Supermarket), HWS (Hardware Store) sind Konstante
- there, here, x, store sind Variable
- Initialer Plan: Start-Step  $\prec$  Finish-Step

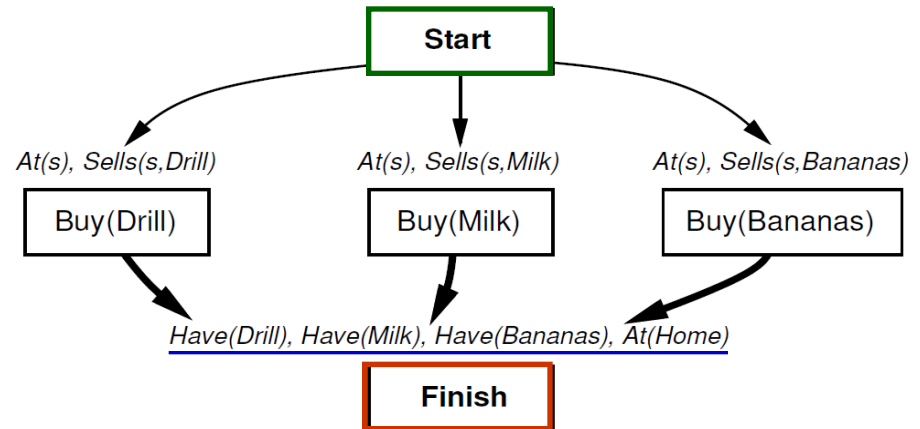


# Planverfeinerung (1)

## Prinzip der Regressionsplanung:

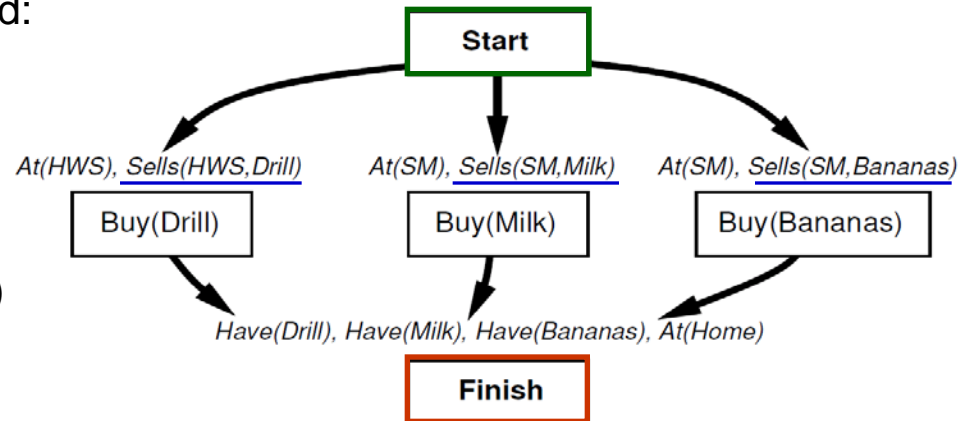
Erfülle *Have*-Prädikate des Ziels:

Buy(Drill) — Have(Drill) → Finish  
 Buy(Milk) — Have(Milk) → Finish  
 Buy(Ban.) — Have(Ban.) → Finish



Erfülle *Sells*-Prädikate durch Startzustand:

Start — Sells(HWS,Drill) → Buy(Drill)  
 Start — Sells(SM, Milk) → Buy(Milk)  
 Start — Sells(SM, Ban.) → Buy(Ban.)



Dünne Pfeile =  $\prec$ , Fette Pfeile =  $\xrightarrow{c}$

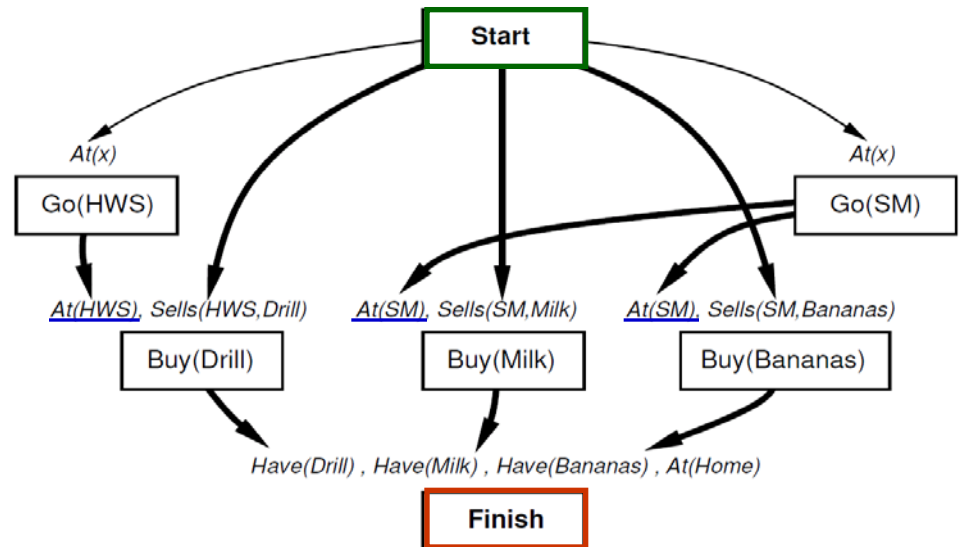
# Planverfeinerung (2)

Erfülle *At*-Prädikate der Buy-Aktionen:

$\text{Go(HWS)} \xrightarrow{\text{At(HWS)}} \text{Buy(Drill)}$

$\text{Go(SM)} \xrightarrow{\text{At(SM)}} \text{Buy(Milk)}$

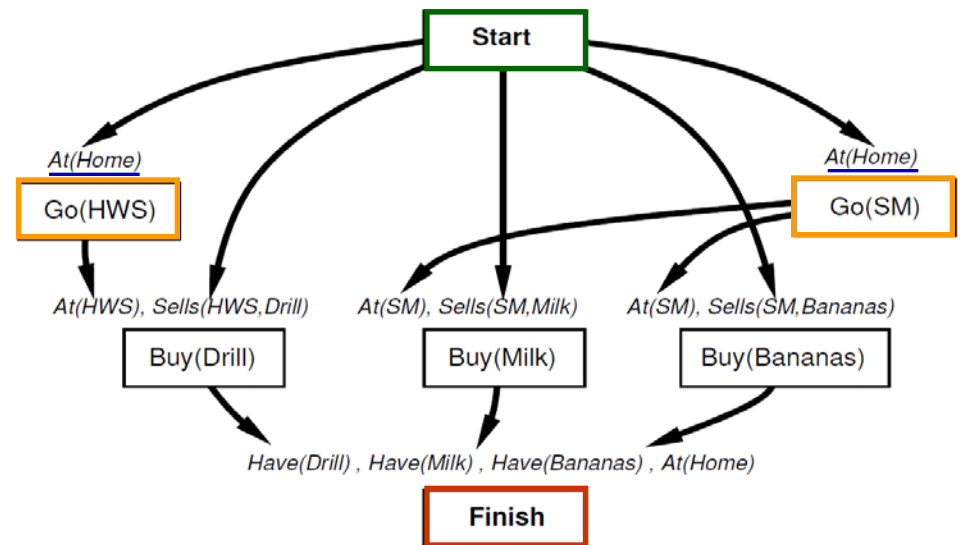
$\text{Go(SM)} \xrightarrow{\text{At(SM)}} \text{Buy(Ban.)}$



Erfülle *At*-Prädikate der Go-Aktionen durch Startzustand:

$\text{Start} \xrightarrow{\text{At(Home)}} \text{Go(HWS)}$

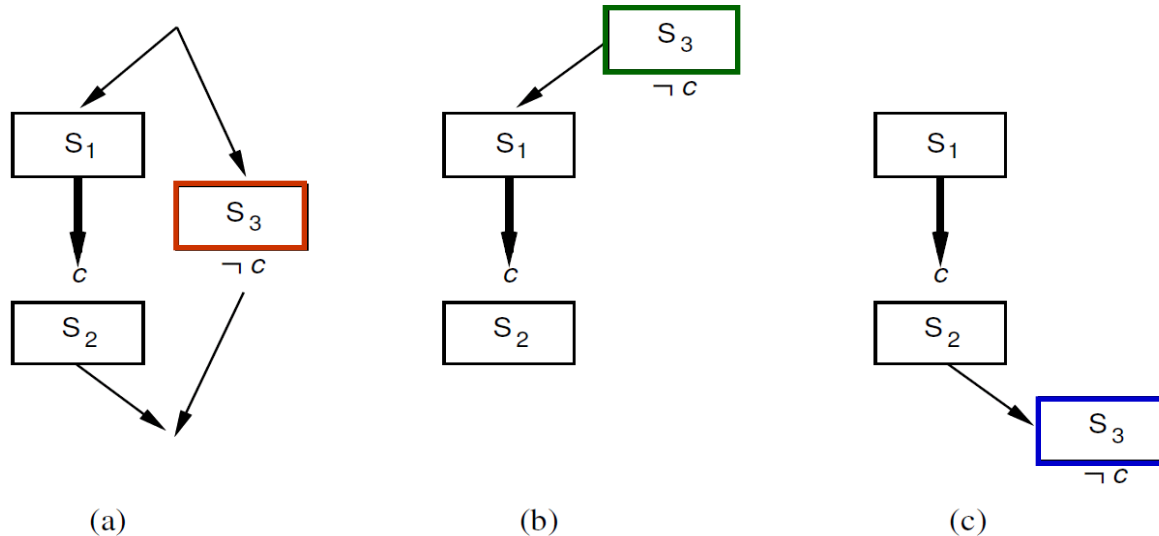
$\text{Start} \xrightarrow{\text{At(Home)}} \text{Go(SM)}$



**Konflikt** (Def.[Vollständigkeit] (2)):

Go(HWS) löscht Vorbedingung At(Home) von Go(SM) und umgekehrt!

# Schutz kausaler Beziehungen



a) Konflikt:  $S_3$  „bedroht“ die kausale Beziehung zwischen  $S_1$  und  $S_2$

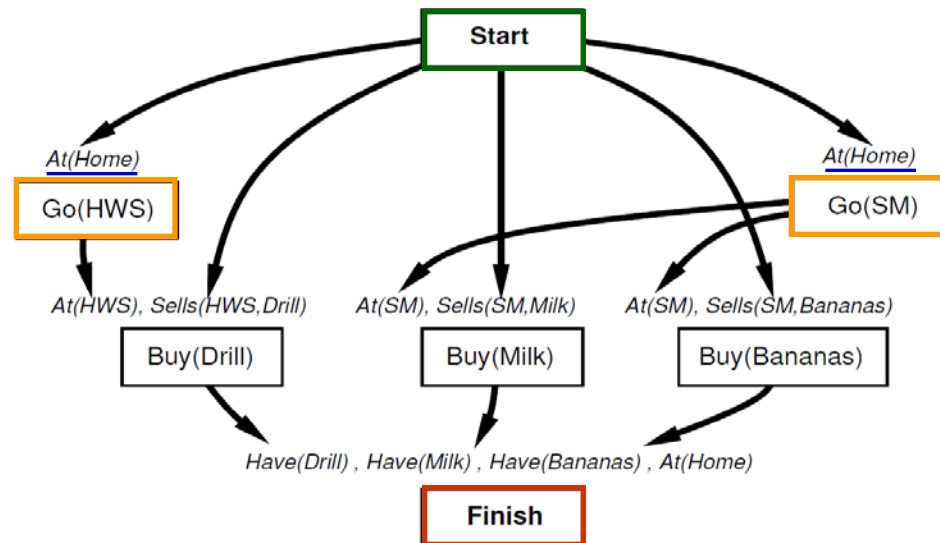
↪ erster Versuch der Konfliktlösung durch *Reihenfolgeänderung*:

b) **Demotion**: den „Bedroher“ vor die kausale Beziehung legen.

c) **Promotion**: den „Bedroher“ hinter die kausale Beziehung legen.

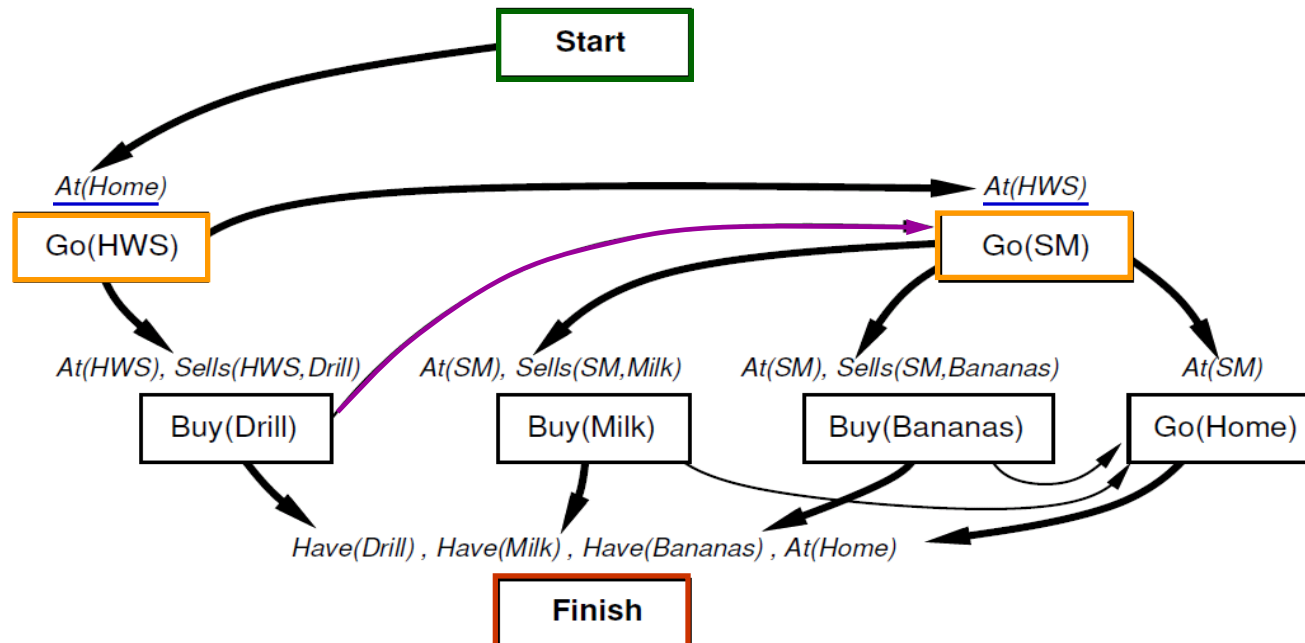
# Planverfeinerung (3)

- Aber: hier können wir den Konflikt nicht durch Reihenfolgeänderung auflösen.
- Grund: Wir haben einen wechselseitig kausalen Konflikt in der gemeinsamen Instanziierung  $x = Home$  der Vorbedingungen  $At(x)$  von  $Go(HWS)$  und  $At(x)$  von  $Go(SM)$  erzeugt, da beide Schritte dieses Prädikat als Effekt negieren.

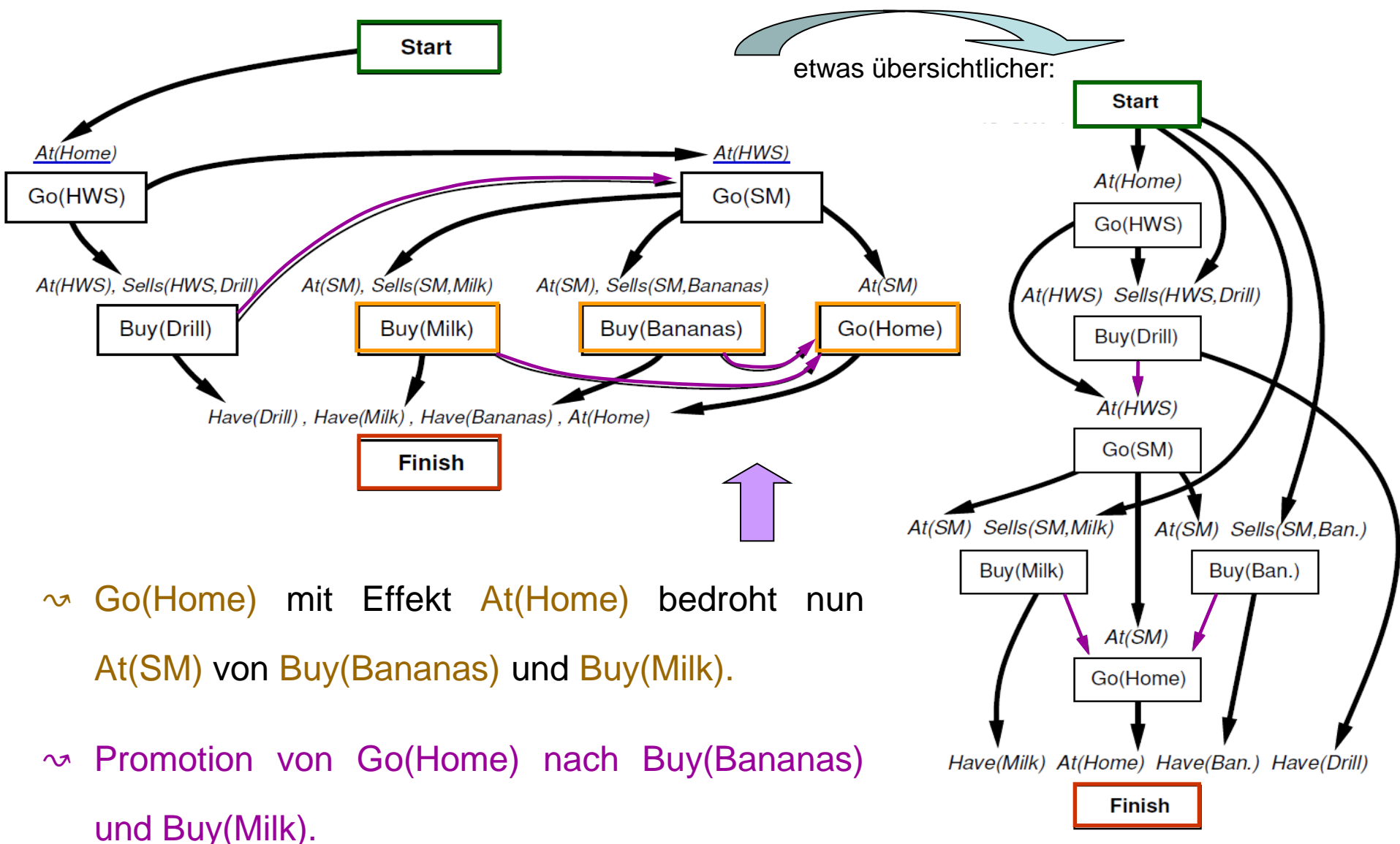


# Planverfeinerung (4)

- Alternative: Bei der **Instanziierung** der Vorbedingung  $At(x)$  der Aktion  $Go(SM)$  wählen wir jetzt  $x = HWS$  anstelle von  $x = Home$ .
- ↪  $Go(SM)$  mit Effekt  $At(SM)$  bedroht nun  $At(HWS)$  von  $Buy(Drill)$ .
- ↪ Promotion von  $Go(SM)$  nach  $Buy(Drill)$ .



# Die vollständige Lösung



# Der Partial-Order-Planning-Algorithmus POP

**function** POP(*initial*, *goal*, *operators*) **returns** *plan*

*plan*  $\leftarrow$  MAKE-MINIMAL-PLAN(*initial*, *goal*)

**loop do**

**if** SOLUTION?(*plan*) **then return** *plan*

$S_{need}, c \leftarrow$  SELECT-SUBGOAL(*plan*)

    CHOOSE-OPERATOR(*plan*, *operators*,  $S_{need}, c$ )

    RESOLVE-THREATS(*plan*)

**end**

---

**function** SELECT-SUBGOAL(*plan*) **returns**  $S_{need}, c$

  pick a plan step  $S_{need}$  from STEPS(*plan*)

    with a precondition  $c$  that has not been achieved

**return**  $S_{need}, c$

---

**procedure** CHOOSE-OPERATOR(*plan*, *operators*,  $S_{need}, c$ )

**choose** a step  $S_{add}$  from *operators* or STEPS(*plan*) that has  $c$  as an effect

**if** there is no such step **then fail**

  add the causal link  $S_{add} \xrightarrow{c} S_{need}$  to LINKS(*plan*)

  add the ordering constraint  $S_{add} \prec S_{need}$  to ORDERINGS(*plan*)

**if**  $S_{add}$  is a newly added step from *operators* **then**

    add  $S_{add}$  to STEPS(*plan*)

    add  $Start \prec S_{add} \prec Finish$  to ORDERINGS(*plan*)

---

**procedure** RESOLVE-THREATS(*plan*)

**for each**  $S_{threat}$  that threatens a link  $S_i \xrightarrow{c} S_j$  in LINKS(*plan*) **do**

**choose** either

*Demotion*: Add  $S_{threat} \prec S_i$  to ORDERINGS(*plan*)

*Promotion*: Add  $S_j \prec S_{threat}$  to ORDERINGS(*plan*)

**if not** CONSISTENT(*plan*) **then fail**

**end**

# Eigenschaften des POP-Algorithmus

---

Man kann zeigen:

- **Korrektheit von POP:** Jedes Resultat des POP-Algorithmus ist ein vollständiger, konsistenter Plan.
- **Vollständigkeit von POP:** Falls Breitensuche oder iterative Tiefensuche benutzt wird, findet der Algorithmus eine Lösung, falls eine existiert.

Aber:

~ Instanziierungen von Variablen werden bislang nicht behandelt



# Variablen

---

Falls eine Variable in einem Literal eines „Effect“ auftaucht, z.B.  $\neg At(x)$ , kann dieses Literal eine *potentielle Bedrohung* für eine kausale Beziehung sein.

Konfliktauflösungen:

- (1) Durch *Gleichheits-Constraint* (z.B.  $x = HWS$ ) um nicht  $At(SM)$  zu bedrohen.
- (2) Durch *Ungleichheits-Constraint* (z.B.  $x \neq SM$ ). Nicht so einschränkend wie (1).  
Erfordert jedoch Spracherweiterung für Unifikation, die bislang nur mit Gleichheit arbeitet.
- (3) Später durchführen, wenn nach Variableninstanziierung wirklich nötig.

Wir wählen Letzteres.

# Behandlung von Variablen

**procedure CHOOSE-OPERATOR(*plan*, *operators*, *S<sub>needs</sub>*, *c*)**

choose a step  $S_{add}$  from *operators* or  $STEPS(plan)$  that has  $c_{add}$  as an effect  
such that  $u = UNIFY(c, c_{add}, BINDINGS(plan))$   
if there is no such step  
    then fail  
add  $u$  to  $BINDINGS(plan)$   
add  $S_{add} \xrightarrow{c} S_{need}$  to  $LINKS(plan)$   
add  $S_{add} \prec S_{need}$  to  $ORDERINGS(plan)$   
if  $S_{add}$  is a newly added step from *operators* then  
    add  $S_{add}$  to  $STEPS(plan)$   
    add  $Start \prec S_{add} \prec Finish$  to  $ORDERINGS(plan)$

Hier sind ggf.  
Alternativen möglich  
~ Backtracking

**procedure RESOLVE-THREATS(*plan*)**

for each  $S_i \xrightarrow{c} S_j$  in  $LINKS(plan)$  do  
    for each  $S_{threat}$  in  $STEPS(plan)$  do  
        for each  $c'$  in  $EFFECT(S_{threat})$  do  
            if  $SUBST(BINDINGS(plan), c) = SUBST(BINDINGS(plan), \neg c')$  then  
                choose either  
                    *Demotion*: Add  $S_{threat} \prec S_i$  to  $ORDERINGS(plan)$   
                    *Promotion*: Add  $S_j \prec S_{threat}$  to  $ORDERINGS(plan)$   
                if not  $CONSISTENT(plan)$   
                    then fail  
            end  
        end  
    end  
end

Führt zu vollständig instanziierten Plänen, falls Initialzustand variablenfrei und alle Operatoren alle Variablen in den Vorbedingungen benutzen.

Ansonsten muss *Solution?* geändert werden, indem nachträglich uninstanziierte Variablen konsistent instanziiert werden.

# Modellierung in STRIPS

---

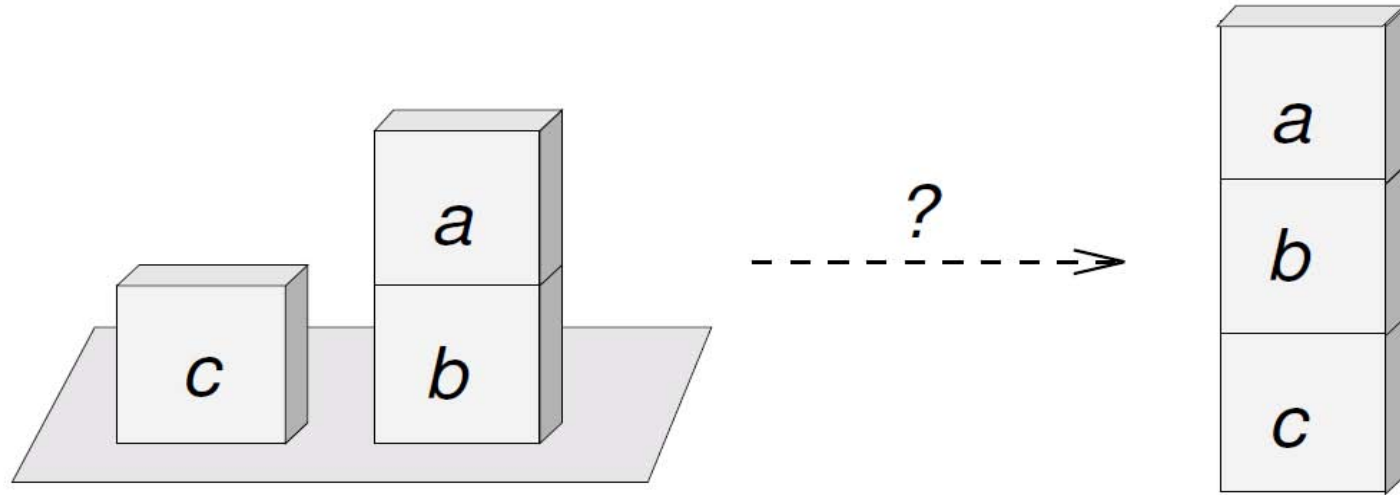
Ähnlich wie bisher auch schon ([Problemlösen](#), [PL1](#)), muss man bei der Modellierung von Aufgaben die folgenden Schritte durchführen:

- Entscheiden, worüber man sprechen will.
- Ein [Vokabular](#) für Bedingungen, Operatoren und Objekte festlegen.
- [Operatoren](#) kodieren.
- [Probleminstanzen](#) kodieren.

Dann kann ein Planer Lösungen produzieren.

# Klassisches Beispiel: Blockwelt

---



- Es gibt benannte Blöcke und eine Tischplatte in der Welt.
- Auf der Tischplatte können beliebig viele Blöcke stehen.
- Auf einem Block kann nur ein anderer Block stehen.
- Ein Block kann nur bewegt werden, wenn kein anderer Block auf ihm steht.
- Umstürzen von Blöcken usw. ist nicht erlaubt.

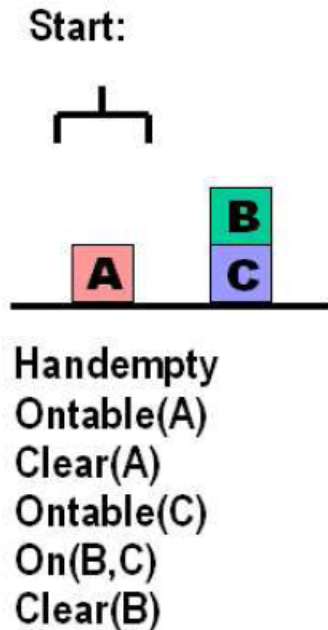
# Modellierung der Blockwelt

---

- Nur die Blöcke als Objekte modellieren, die Tischplatte ist implizit repräsentiert.
  - ~ Objekte:  $a, b, c$
- Man repräsentiert explizit, ob ein Block bewegt werden kann und ob er auf dem Tisch liegt.
  - ~ Prädikate:
    - $On(x, y)$ :  $x$  liegt auf  $y$
    - $OnTable(x)$ :  $x$  liegt auf der Tischplatte
    - $Clear(x)$ : Es liegt nichts auf  $x$
- Es gibt Operatoren, um Blöcke von Blöcken auf andere Blöcke zu bewegen.
- Es gibt Operatoren, um Blöcke vom Tisch auf einen anderen Block zu bewegen und umgekehrt.
  - ~ Operatoren:  $move(x, y, z)$ ,  $stack(x, y)$ ,  $unstack(x, y)$ , ...

# Modellierung der Blockwelt

Eine etwas erweiterte Repräsentation der Blockwelt mit Berücksichtigung einer manipulierenden Roboterhand. Hier mit Angabe eines Start- und Zielzustandes.

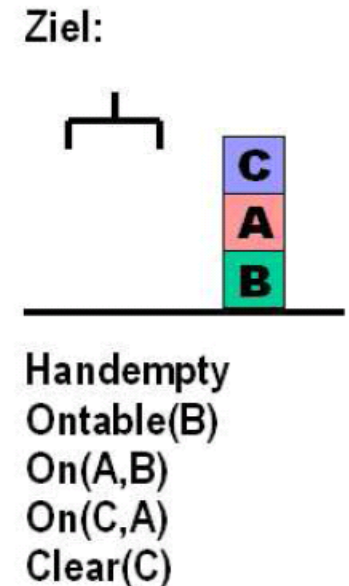


```
pickup(x)
  Cond & Del: Ontable(x), Clear(x), Handempty
  Add: Holding(x)

putdown(x)
  Cond & Del: Holding(x)
  Add: Ontable(x), Clear(x), Handempty

stack(x)
  Cond & Del: Holding(x), Clear(y)
  Add: On(x,y), Clear(x), Handempty

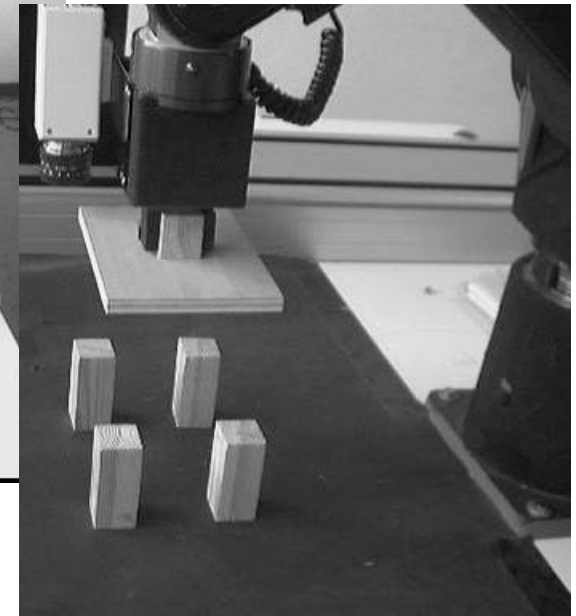
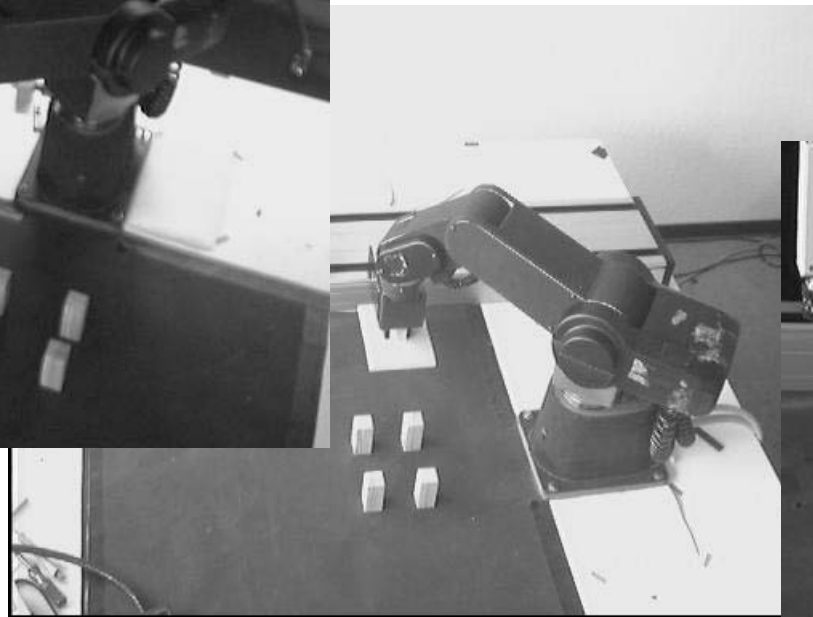
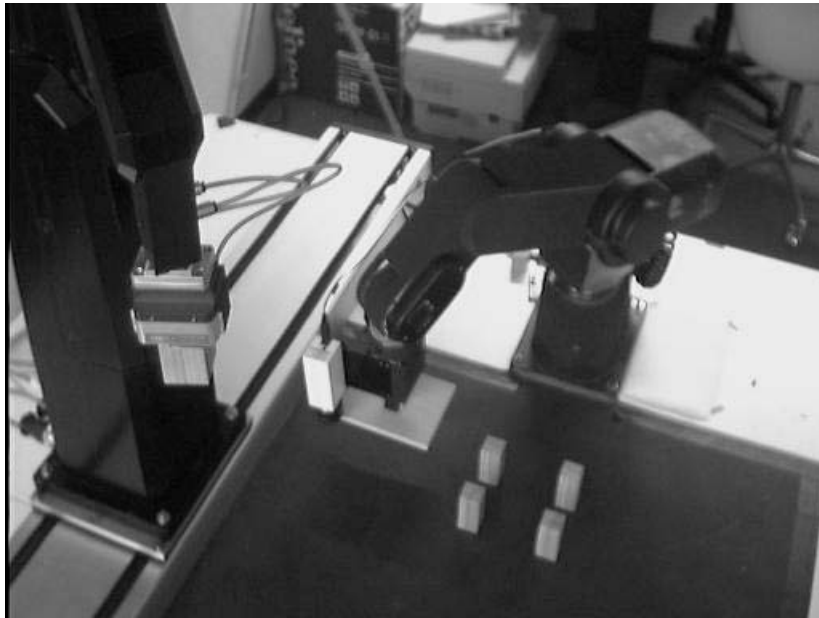
unstack(x)
  Cond & Del: On(x,y), Clear(x), Handempty
  Add: Holding(x), Clear(y)
```



# Laboranwendung der Blockwelt

---

Eine Variante der Blockwelt in einer Laboranwendung, in der ein Roboterarm eine Zielkonstellation erstellen soll.



# Zusammenfassung

---

- Im Prinzip ist Planung auf *logische Inferenz* im *Situationskalkül* reduzierbar; das ist aber sehr ineffizient.
- Handlungsplanung zeigt gegenüber Problemlösen eine *flexiblere Repräsentation*:
  - Unterscheidung zwischen *Zuständen* und *Aktionen* mit Vor-/Nachbedingungen
    - ~ Effiziente Kombination zwischen Zuständen und Aktionen möglich
  - Arbeiten im *Planungsraum*: flexibles Einfügen von erfüllenden Aktionen dort, wo nötig nach dem Least-Commitment-Prinzip (s.u.).
    - ~ Kein starres inkrementelles Planen von Start zu Ziel.
  - *Nichtlineares Planen* nach dem Prinzip der *geringsten Festlegung* (*least commitment*): Entscheidungen nur dann treffen, wenn nötig.
    - ~ Unterteilung in unabhängige Teilpläne.
  - Der POP-Algorithmus realisiert *nichtlineares* Planen, liefert partiell geordnete Pläne und ist *vollständig* und *konsistent*.