Kapitel 7. Planen

7.1 Planen vs. Problem-Lösen

7.2 STRIPS

7.3 Partial-Order Planning

7.4 Conditional Planning

7.5 Erweiterungen

Zeitplan:

- 1 Doppelstunde für 7.1 - 7.2 (Def. des Plans),
- 1 Doppelstunde für 7.3 - 7.5.

1 Einleitung

2 Intelligente Agenten

3 Problem-Lösen durch Suchen

4 Informierte Suche

5 Wissenrepräsentation I

6 Wissenrepräsentation II

7 Planen

7.1 Planen vs. Problem-Lösen

Problem-lösender Agent: Die Effekte einer fixen Sequenz von Aktionen werden bestimmt.

Wissensbasierter Agent: Aktionen können ausgewählt werden.

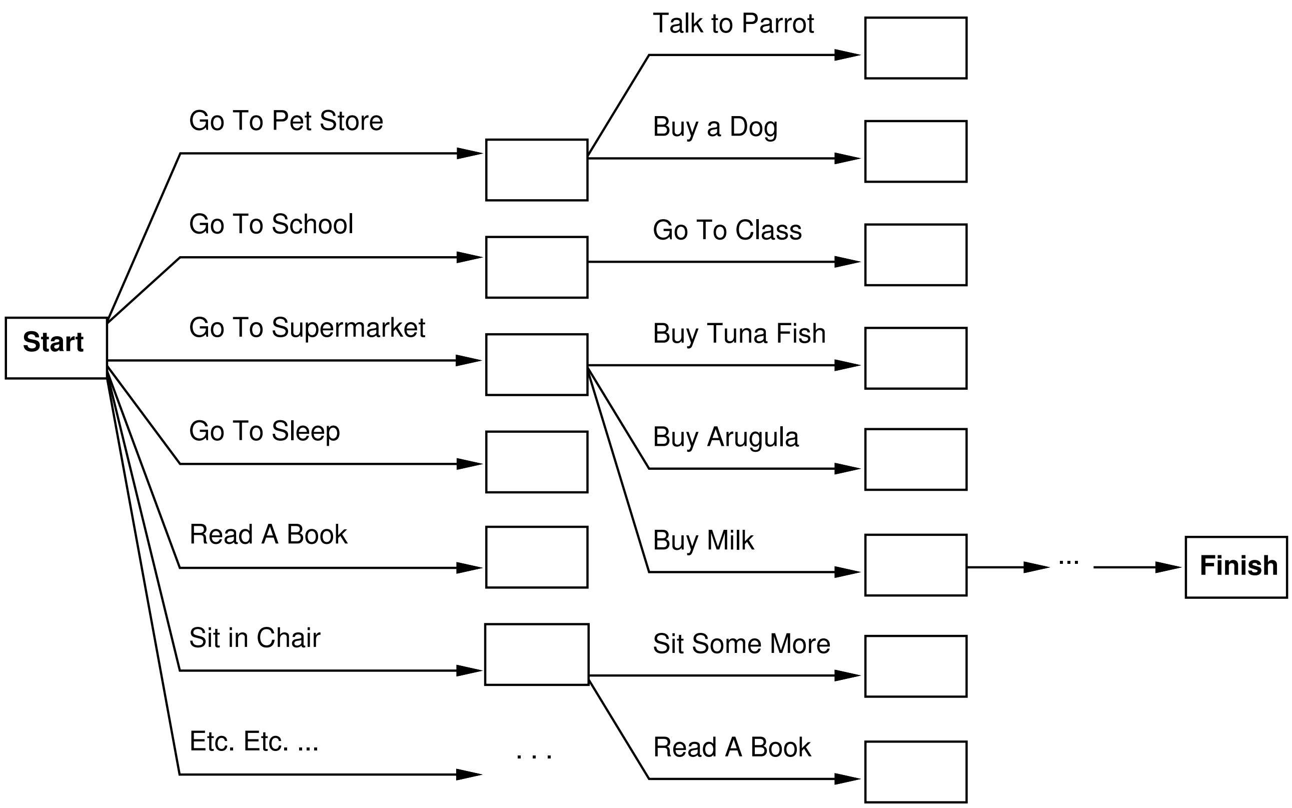
Beide zusammen versuchen wir zu einem **Planungsagenten** zu verschmelzen.

```
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 ← STATE-DESCRIPTION(KB, t)
   if p = NoPlan then
      G ← ASK(KB, MAKE-GOAL-QUERY(t))
      p ← IDEAL-PLANNER(current, G, KB)
   if p = NoPlan or p is empty then action ← NoOp
   else
      action ← FIRST(p)
      p ← REST(p)
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

Beispiel 7.1
Wir wollen eine frische Bananenmilch trinken und dann ein paar Löcher zuhause in die Wand bohren. Dazu brauchen wir Bananen, Milch und eine Bohrmaschine.

Wie geht ein Problem-lösender Agent damit um?



Planen unterscheidet sich vom Problem-Lösen in folgenden Punkten:

1. Benutze PL1 um Verbindungen zwischen Zuständen und Aktionen herzustellen. Wenn $have(Milk)$ das Ziel ist und $Buy(x) \rightarrow Have(x)$, so ist es sinnvoll, $Buy(Milk)$ zu betrachten.

   Dies vermeidet irrelevante Aktionen.

2. Beim Zusammenstellen eines Planes dürfen Aktionen beliebig addiert werden: es muß nicht in sturer Reihenfolge gehen. Achtung: Reihenfolge des Plans hat zunächst nichts mit der Reihenfolge der Ausführung zu tun.

   Dies setzt den Verzweigungsfaktor herab.

   Zustände sind Mengen von Sätzen, daher nicht notwendig vollständig.

3. Die meisten Teile der Welt sind unabhängig voneinander. $Have(Milk) \wedge Have(Bananas) \wedge Have(Drill)$.

   Dies ermöglicht Divide-et-Impera Strategien.

Planen im Situationskalkül

Anfangszustand: $At(Home, S_0)$, $\neg Have(Milk, S_0)$, $\neg Have(Bananas, S_0)$, $\neg Have(Drill, S_0)$.

Ziel: $\exists s \; (At(Home, s) \wedge Have(Milk, s) \wedge Have(Bananas, s) \wedge Have(Drill, s)$.

Axiome: z.B. über "Milch kaufen":

$$\begin{aligned} &\forall a, s \; Have(Milk, result(a, s)) \leftrightarrow \\ &(a = Buy(Milk) \wedge At(Supermarket, s)) \vee \\ &(Have(Milk, s) \wedge a \neq Drop(Milk)) \end{aligned}$$

Wir brauchen nun noch einen Term $Result'(l, s)$: die Situation die entsteht, wenn in s die Sequenz l von Aktionen ausgeführt wird.

$$\begin{array}{lll} \forall s & Result'([], s) & := \; s \\ \forall a, p, s & Result'([a|p], s) & := \; Result'(p, result(a, s)) \end{array}$$

Die Aufgabe ist nun ein p zu finden, mit
$At(Home, Result'(p, S_0)) \wedge Have(Milk, Result'(p, S_0)) \wedge Have(Bananas, Result'(p, S_0)) \wedge Have(Drill, Result'(p, S_0))$.

Probleme:

1. $T \vdash \phi$ ist nur semi-entscheidbar.
2. Ist p ein Plan, so auch $[Leere\_Aktion|p]$ und $[A, A^{-1}|p]$.

Also sollte man möglichst keinen allgemeinen Beweiser, sondern einen auf den speziellen Fall zugeschnittenen benutzen. Außerdem sollte man die Sprache einschränken.

Man mache sich den Unterschied klar zwischen

- Goal an Planner und
- Query an Theorembeweiser.

Im ersten Fall sucht man einen Plan derart, daß *nach Ausführung* des Planes das Goal gilt.

Im zweiten Fall fragt man ob die Query bezüglich der KB wahr gemacht werden kann: $KB \models \exists x \phi(x)$. Das dynamische steckt in den Termen. Die Logik selbst ist statisch.

7.2 STRIPS

STRIPS steht für **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver.

Zustände: Konjunktion von Funktions-freien Grund-Literalen

Ziel: Konjunktion von Funktions-freien Literalen

Aktionen: STRIPS-Operatoren bestehen aus 3 Komponenten

1. *Description*, Name der Aktion,
2. *Precondition*, Konjunktion von Atomen,
3. *Postcondition*, Konjunktion von Literalen.

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

$$\boxed{\text{Go(there)}}$$

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

Definition 7.2 (Anwendbarer Operator)
Ein Operator Op ist anwendbar in einem Zustand s, falls seine Variablen so instantiiert werden können, daß alle Preconditions in s wahr sind: $Precond(s) \subseteq s$.

Im resultierenden Zustand gelten alle positiven Literale aus $Effect(Op)$ und alle Literale die in s gelten, bis auf die negierten Literale aus $Effect(Op)$.

Bisher kann man alles auch als Problem-Lösen sehen. Wir benutzen STRIPS als Repräsentationsformalismus und suchen nach einem Lösungspfad:

- Knoten im Suchbaum $\approx$ Situationen,
- Lösungspfade $\approx$ Pläne.

Idee: Suche im Raum aller Pläne!

Fange mit partiellem Plan an und erweitere sukzessive.

Dazu brauchen wir Operatoren die auf Plänen operieren. Wit unterscheiden zwischen 2 Arten:

Refinement-Op: Einem Plan werden Constraints angehängt. Ein Plan steht dann für die Menge aller vollständigen Pläne (analog steht $Cn(T)$ für $MOD(T)$).

Modification-Op: Alle anderen.

Wichtig: Eindringlich den Unterschied zum Problem-Lösen klarmachen. Suche im Raum aller Pläne, also eine Abstraktionsebene höher.

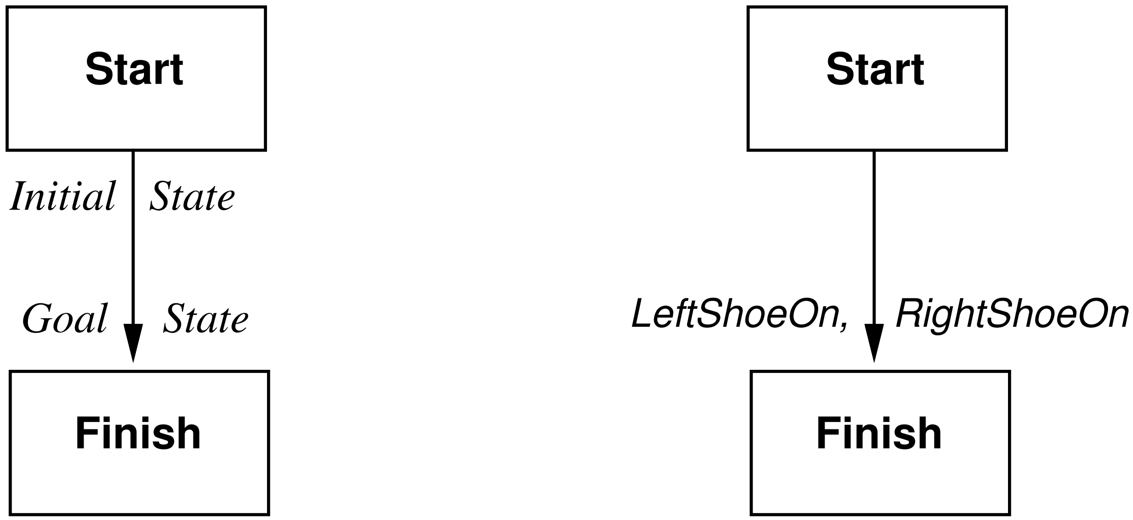
Wie repräsentiert man Pläne? Es gilt zwei wichtige Punkte zu beachten:

- *Variablen-Instantiierungen:* Instantiiere nur, falls notwendig. Nimm also stets den mgU.
- *Partielle Ordnung:* Sieh' zunächst von der genauen Reihenfolge ab (dadurch wird der Suchraum kleiner).

Definition 7.3 (Plan)
Ein Plan ist ein Quadrupel bestehend aus

1. *Einer Multi-Menge von* Schritten. *Ein Schritt ist ein Operator des Problems.*
2. *Einer Menge von* Ordnungs-Bedingungen *(auch* Constraints *genannt) an die Schritte. Diese haben die Form* $S_i \prec S_j$*:* S_i *muß vor* S_i *ausgeführt werden.*
3. *Einer Menge von* Instantiierungs-Bedingungen. *Diese haben die Form* $v = x$ *wobei* v *eine Variable in einem Schritt und* x *eine Konstante oder Variable ist.*
4. *Einer Menge von* kausalen Links. *Diese haben die Form* $S_i \xrightarrow{c} S_j$*:* S_i *stellt* c *sicher für* S_j.

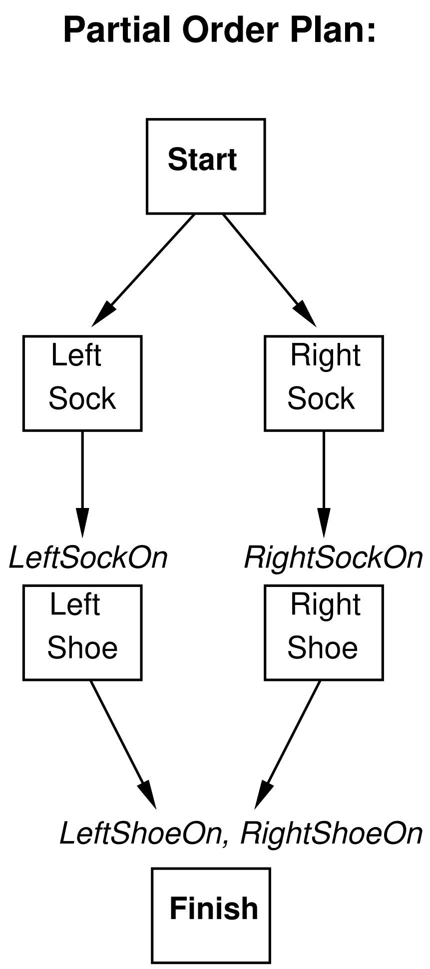
Um den Anfangsplan eines Problems darzustellen definieren wir 2 Operatoren START und FINISH wie folgt



(a) (b)

Als Anfangsplan eines Problems ergibt sich damit:

$$
\textbf{Plan(}\quad
\begin{array}{ll}
Steps: & \{S_1 : START, S_2 : FINISH\} \\
Orderings: & \{S_1 \prec S_2\} \\
Bindings: & \emptyset \\
Links: & \emptyset
\end{array}
\quad\textbf{)}
$$


Partial Order Plan:
Start
Left Sock
Right Sock
LeftSockOn
RightSockOn
Left Shoe
Right Shoe
LeftShoeOn, RightShoeOn
Finish


Total Order Plans:
Start
Right Sock
Left Sock
Right Shoe
Left Shoe
Finish
Start
Right Sock
Left Sock
Left Shoe
Right Shoe
Finish
Start
Left Sock
Right Sock
Right Shoe
Left Shoe
Finish
Start
Left Sock
Right Sock
Left Shoe
Right Shoe
Finish
Start
Right Sock
Right Shoe
Left Sock
Left Shoe
Finish
Start
Left Sock
Left Shoe
Right Sock
Right Shoe
Finish

Was ist eine Lösung?

Das ist einfach, falls wir nur voll instantiierte, linear geordnete Pläne betrachten: dann können wir es nämlich einfach überprüfen. In unserem Fall ist es aber komplizierter:

Definition 7.4 (Lösung eines Planes)
Eine Lösung ist ein vollständiger *und* konsistenter *Plan. Dabei bedeutet vollständig, daß jede Precondition eines Schrittes von einem anderen Schritt sichergestellt wird und konsistent, daß es keine Widersprüche in den Instantiierungs oder Ordnungs-Bedingungen gibt.*

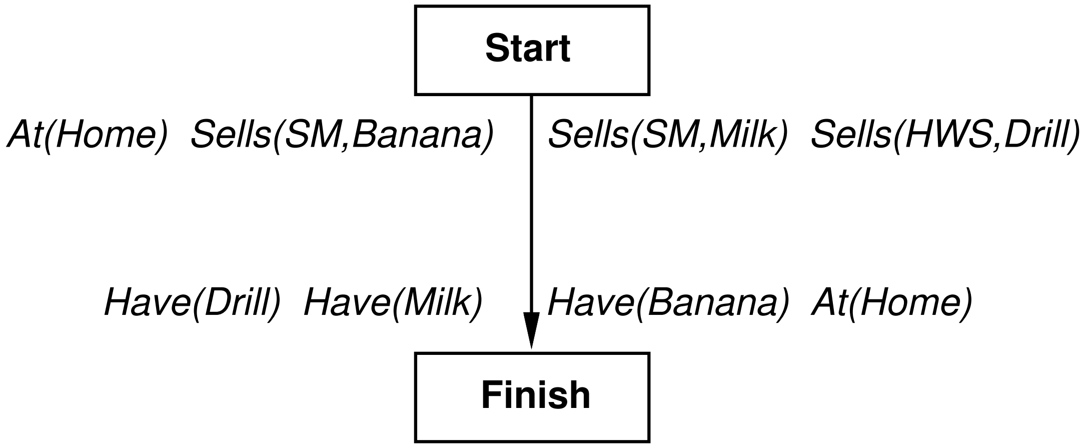
Noch präziser:

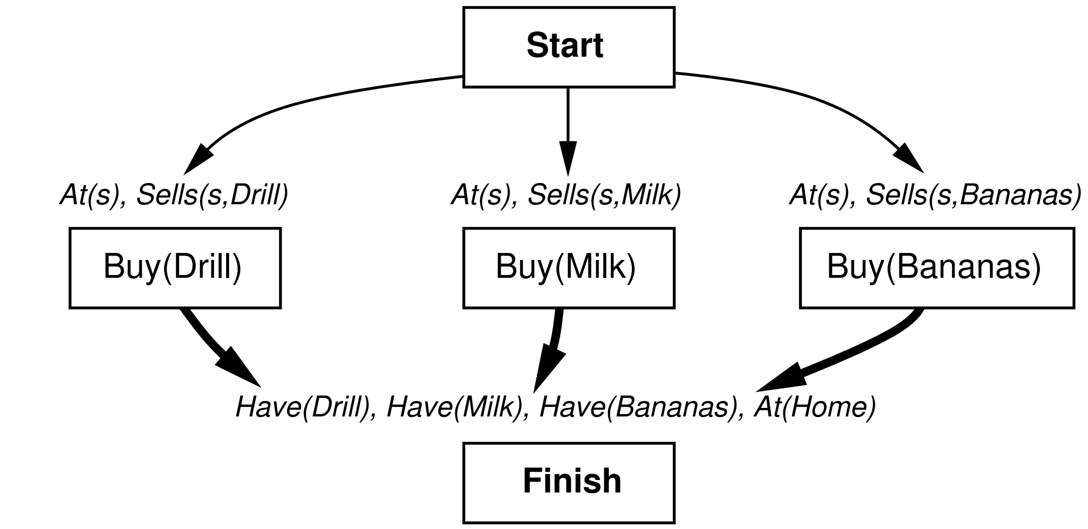
- "S_i stellt c sicher für S_j" bedeutet
  - $c \in Precondition(S_j)$,
  - $S_i \prec S_j$,
  - $\nexists S_k : \neg c \in Effect(S_k)$ mit $S_i \prec S_k \prec S_j$ in irgendeiner Linearisierung des Plans.
- "Keine Widersprüche" bedeutet
  - Weder ($S_i \prec S_j$ und $S_j \prec S_i$) noch ($v = A$ und $v = B$ für verschiedene Konstanten A, B).

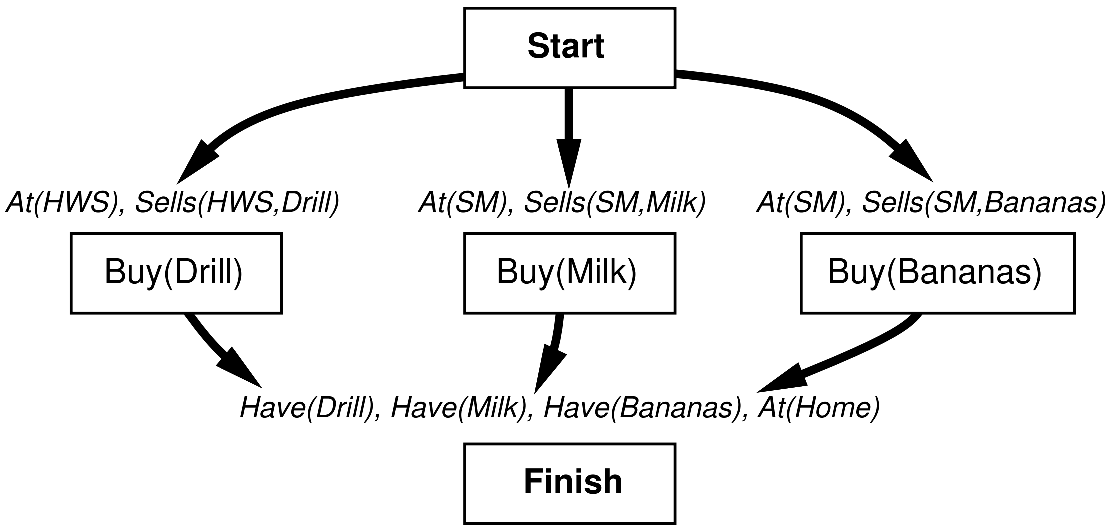
  Man beachte, daß dies nicht nur explizit gefordert wird. Die entsprechenden Aussagen können auch folgen, da $\prec$ und $=$ etwa transitiv sind.

7.3 Partial-Order Planning

Wir betrachten unser Bananen-Milch-Beispiel. Als Operatoren verwenden wir Buy und Go.

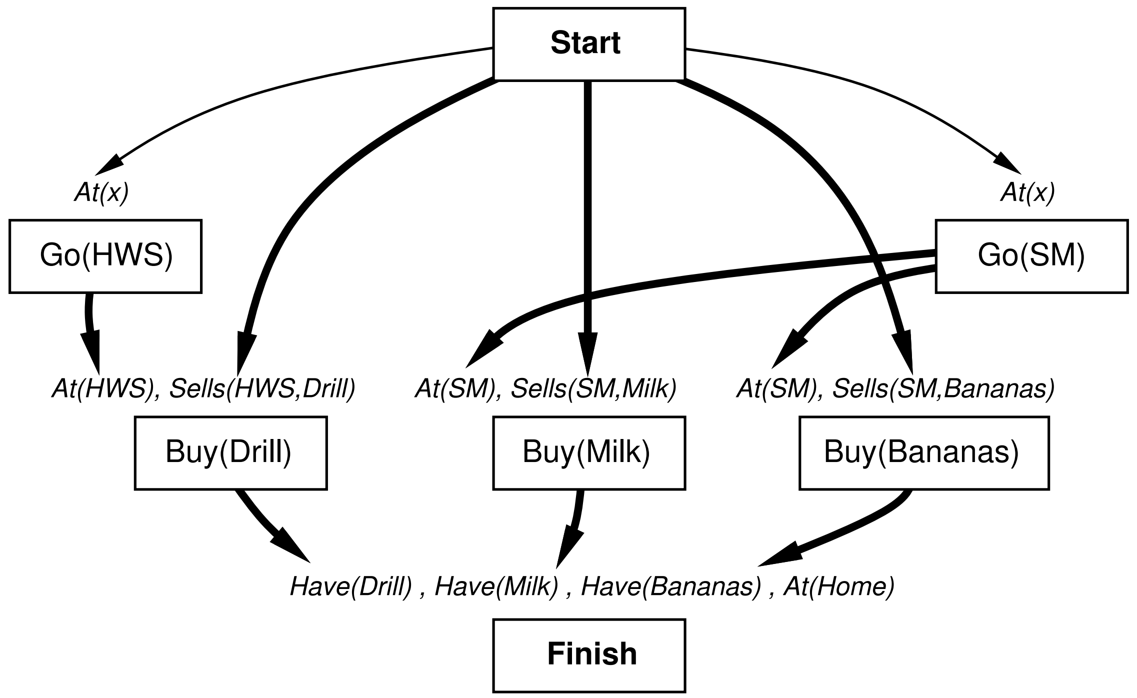


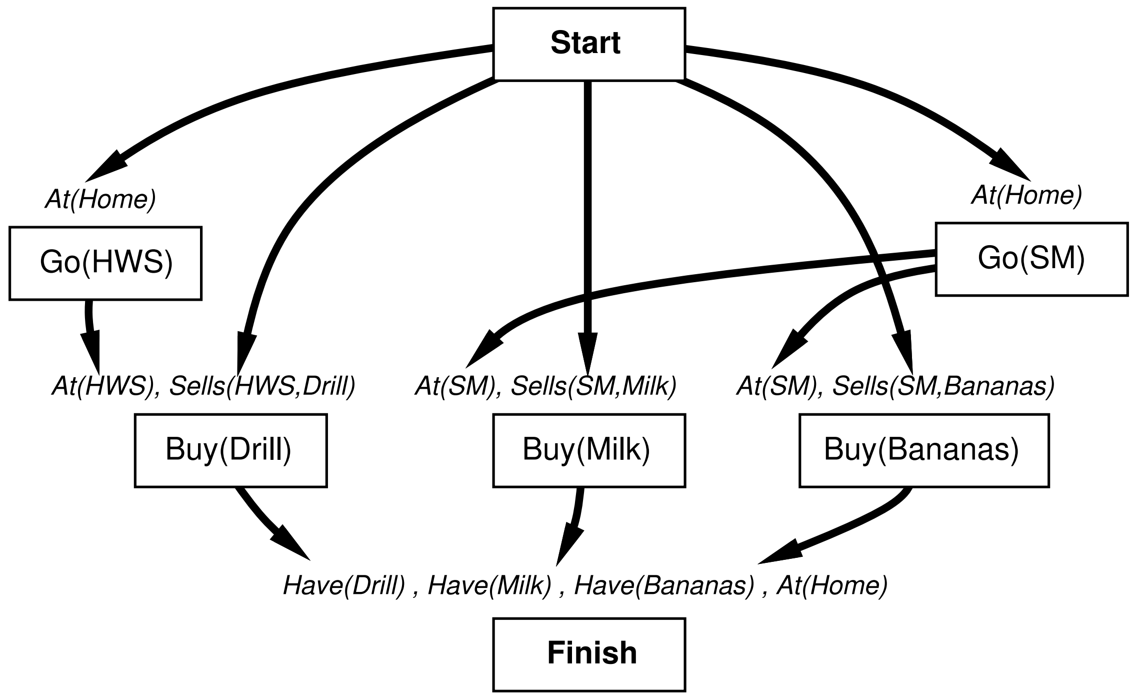




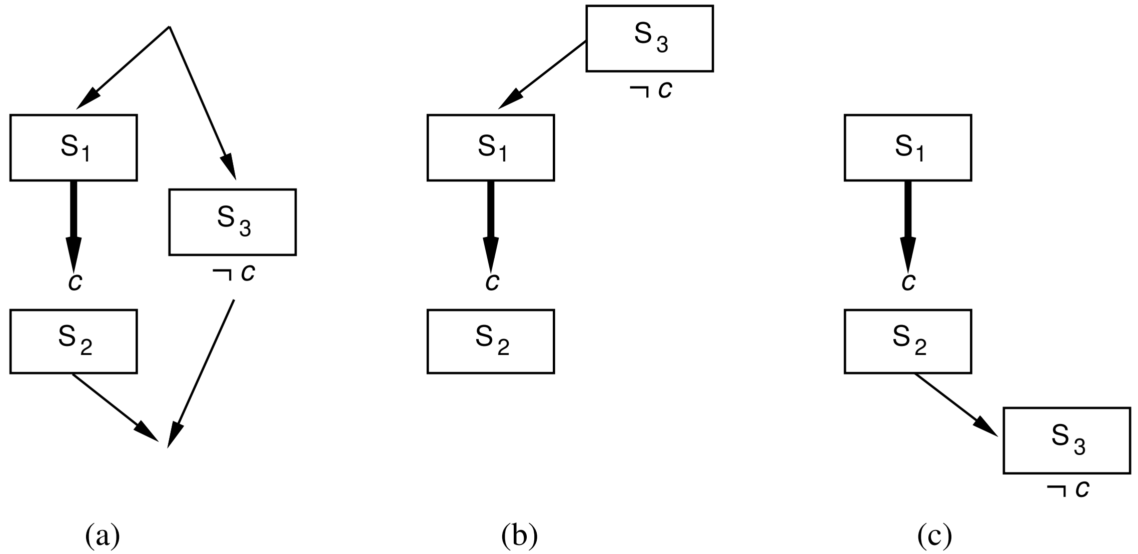
Kausale Links werden geschützt!

Kausale Links sind fett eingezeichnet. Sie schließen den entsprechenden Ordnungs-Constraint ein!


Start
At(x)
Go(HWS)
At(x)
Go(SM)
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) , Have(Milk) , Have(Bananas) , At(Home)
Finish

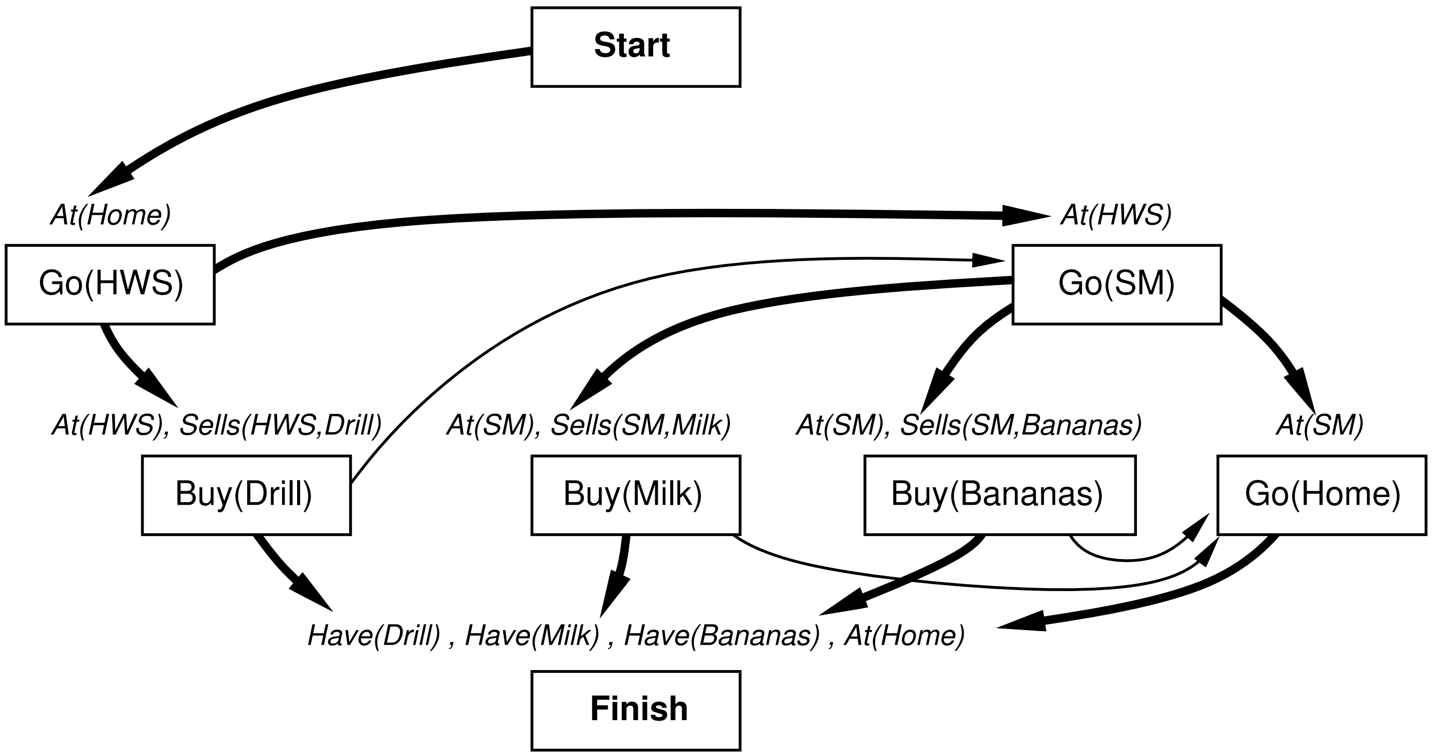

Start
At(Home)
Go(HWS)
At(Home)
Go(SM)
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) , Have(Milk) , Have(Bananas) , At(Home)
Finish

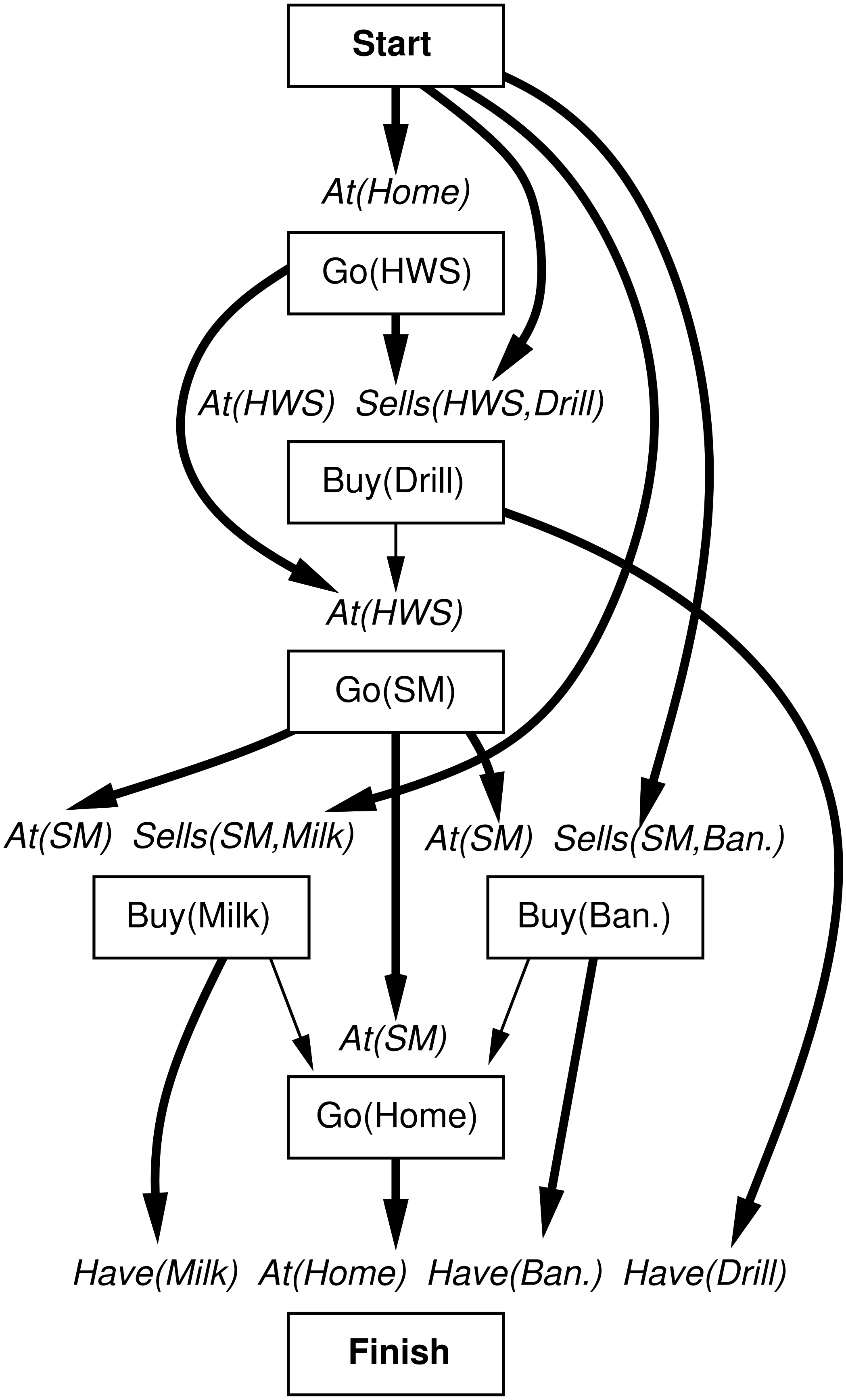
Der letzte partielle Plan kann nicht mehr erweitert werden. Wie stellt man das fest? Indem man feststellt, daß ein kausaler Link bedroht wird und die Drohung nicht aufgehoben werden kann!



Die Drohung kann aufgehoben werden, indem man die Drohung vor- oder nach-schaltet (das muss nicht immer gehen!).

Wir müssen einen Go-Schritt einführen, um die letzte Vorbedingung zu sichern. Aber wie können wir die Vorbedingung des Go-Schrittes sichern? Es gibt jetzt jede Menge Bedrohungen, von denen sich die meisten nicht auflösen lassen. Schließlich kommen wir zu




Start
At(Home)
Go(HWS)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM) Sells(SM,Milk)
At(SM) Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk) At(Home) Have(Ban.) Have(Drill)
Finish

Das führt also zu folgendem Algorithmus:

- In jeder Runde wird der Plan erweitert um die Vorbedingung eines Schrittes zu sichern. Dies geschieht durch Auswahl eines geeigneten Operators.
- Der entsprechende kausale Link wird eingeführt. Drohungen werden durch Ordnungs-Constraints abgewehrt (es gibt 2 Fälle: der neue Schritt bedroht bestehende oder bestehende bedrohen den neuen).
- Falls es keinen Operator gibt oder die Drohungen nicht abgewehrt werden können — backtracken.

Theorem 7.5 (POP)
POP ist vollständig und korrekt.

Ist unser Beispiel eigentlich fair? Wir hatten doch nur sehr wenige Operatoren (ganze 2) und brauchen uns daher nicht zu wundern, daß der Verzweigungsfaktor so klein ist!

Dieser Einwand ist nicht gerechtfertigt, da auch wenn wir noch 1000 andere irrelevante Operatoren zulassen, diese nicht betrachtet werden, da wir sie nicht benutzen können, um unsere Vorbedingungen zu sichern. POP ist eben sehr ziel-gerichtet und vermeidet absolut irrelevante Aktionen.

Warum wird eigentlich nicht über die Auswahl der zu sichernden Vorbedingung gebacktrackt? (siehe Aufgabe)

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

  plan ← MAKE-MINIMAL-PLAN(*initial, goal*)
  loop do
    if SOLUTION?(*plan*) **then return** *plan*
    S_{need}, c ← 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*)

---

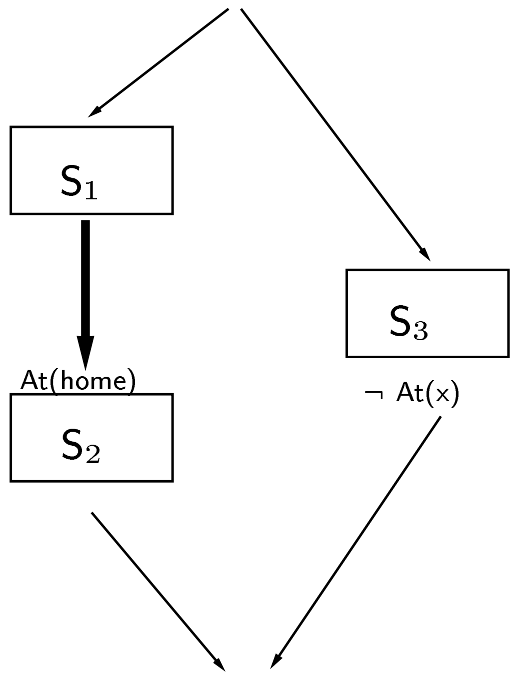
procedure RESOLVE-THREATS(*plan*)

  for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
    choose either
      Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
      Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
    if not CONSISTENT(*plan*) **then fail**
  end

Bisher haben wir uns überhaupt nicht um Variablen-Substitutionen gekümmert.

Angenommen, S_1 sichert die $At(home)$ Vorbedingung eines anderen Schrittes S_2 und es gibt einen paralellen Schritt S_3 mit der Nachbedingung $\neg At(x)$. Ist dies eine Bedrohung?

Wir nennen solch eine Drohung *möglich* und ignorieren sie zunächst, behalten sie aber im Hinterkopf. Falls aber irgendwann einmal x mit $home$ instantiiert wird, dann entsteht eine echte Drohung die aufgelöst werden muss.



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

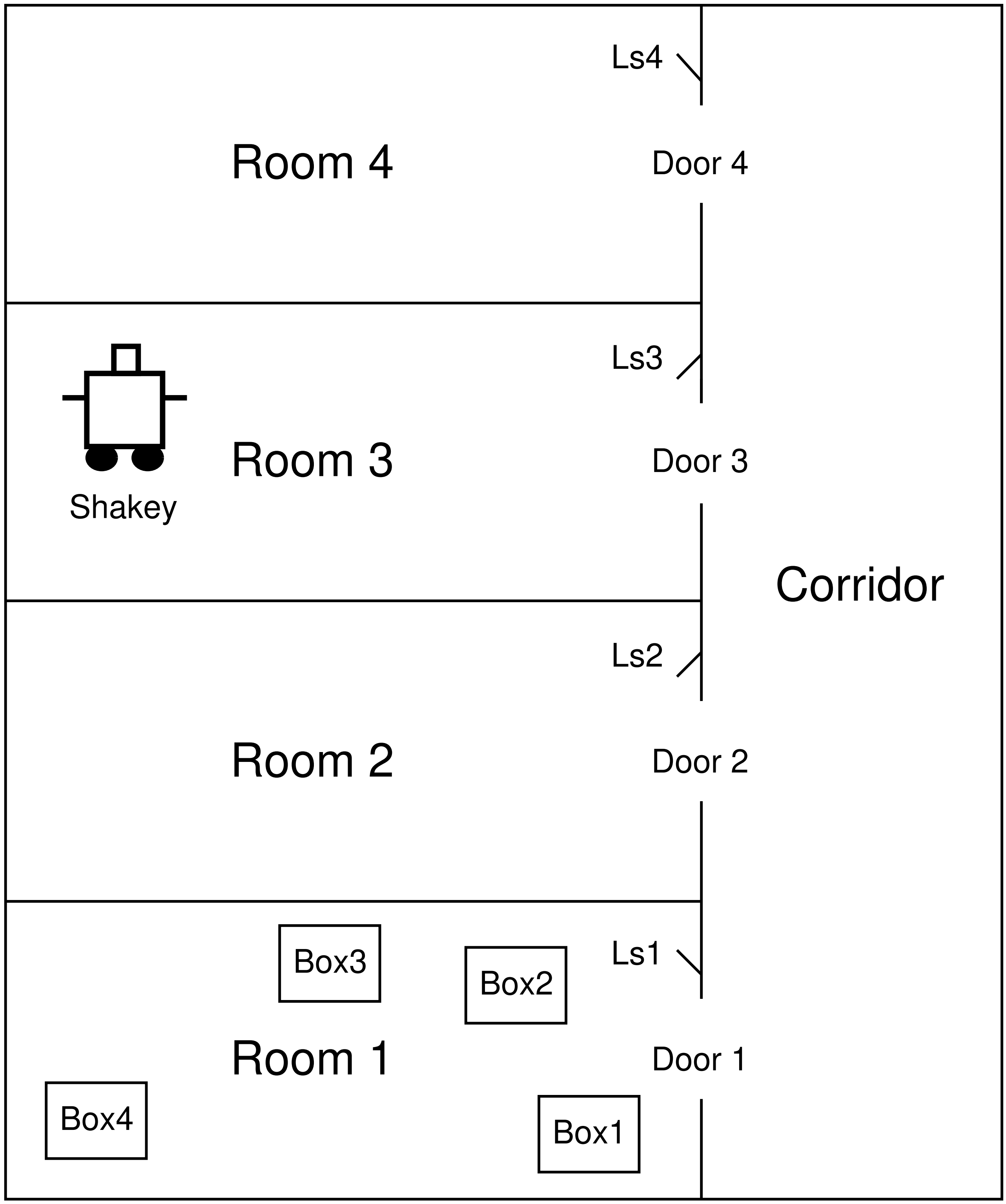
---

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
            Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
            Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
        if not CONSISTENT(*plan*)
          then fail
      end
    end
  end

STRIPS wurde ursprünglich für SHAKEY entworfen, einen kleinen mobilen Roboter. SHAKEY wurde duch 6 Operatoren beschrieben: $Go(x)$, $Push(b, x, y)$, $Climb(b)$, $Down(b)$, $Turn\_On(ls)$, $Turn\_Off(ls)$.

Damit SHAKEY den Lichtschalter an oder ausstellen kann, muss er auf einer Box stehen. Damit SHAKEY bei einer Go-Aktion nicht von einer Box fällt, ist $On(Shakey, floor)$ eine Vorbedingung der Go-Aktion.


Ls4
Room 4
Door 4
Ls3
Room 3
Door 3
Shakey
Corridor
Ls2
Room 2
Door 2
Box3
Box2
Ls1
Room 1
Door 1
Box4
Box1

7.4 Conditional Planning

Was tun, wenn die Welt nicht vollständig zugänglich ist? (Wo bekommt man Milch? Milch-Preis hat sich verdoppelt, man hat nicht genug Geld dabei!)

Idee: Führe neue Aktionen ein um gewisse Gegebenheiten abzufragen und entsprechend zu reagieren. (Wieviel kostet die Milch?)

Ein platter Reifen

Op($Action: \ Remove(x),$
$Prec: \ On(x),$
$Effect: \ Off(x) \land Clear\_Hub(x) \land \neg On(x)$
)

Op($Action: \ Put\_on(x),$
$Prec: \ Off(x) \land Clear\_Hub(x),$
$Effect: \ On(x) \land \neg Clear\_Hub(x) \land Off(x)$
)

Op($Action: \ Inflate(x),$
$Prec: \ Intact(x) \land Flat(x),$
$Effect: \ Inflated(x) \land \neg Flat(x)$
)

Ziel: $On(x) \land Inflated(x),$

Anfangszustand: $Inflated(Spare) \land Intact(Spare) \land Off(Spare) \land On(Tire_1) \land Flat(Tire_1).$

Was liefert unser POP?

Da $Intact(Tire_1)$ nicht vorhanden, liefert POP als Plan

$$[Remove(Tire_1), Put\_on(Spare)].$$

Würden Sie das auch so machen? Besser wäre es doch, einen konditionalen Plan zu liefern:

Falls $Intact(Tire_1)$ dann: $Inflate(Tire_1)$.

Dazu müssen wir konditionale Schritte schon bei der Planerstellung erlauben:

Definition 7.6 (Konditionale Schritte)
Ein konditionaler Schritt in einem Plan hat die Form

$$\mathit{If}(\{Condition\}, \{Then\_Part\}, \{Else\_Part\})$$

Der entsprechende Planungsagent sieht dann so aus:

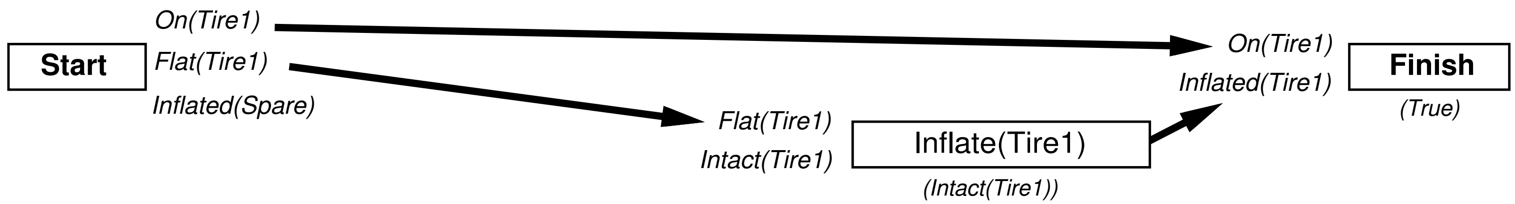
```
function CONDITIONAL-PLANNING-AGENT(percept) returns an action
  static: KB, a knowledge base (includes action descriptions)
          p, a plan, initially NoPlan
          t, a counter, initially 0, indicating time
          G, a goal

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  current ← STATE-DESCRIPTION(KB, t)
  if p = NoPlan then p ← CPOP(current, G, KB)
  if p = NoPlan or p is empty then action ← NoOp
  else
     action ← FIRST(p)
     while CONDITIONAL?(action) do
        if ASK(KB, CONDITION-PART[action]) then p ← APPEND(THEN-PART[action], REST(p))
        else p ← APPEND(ELSE-PART[action], REST(p))
        action ← FIRST(p)
     end
     p ← REST(p)
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

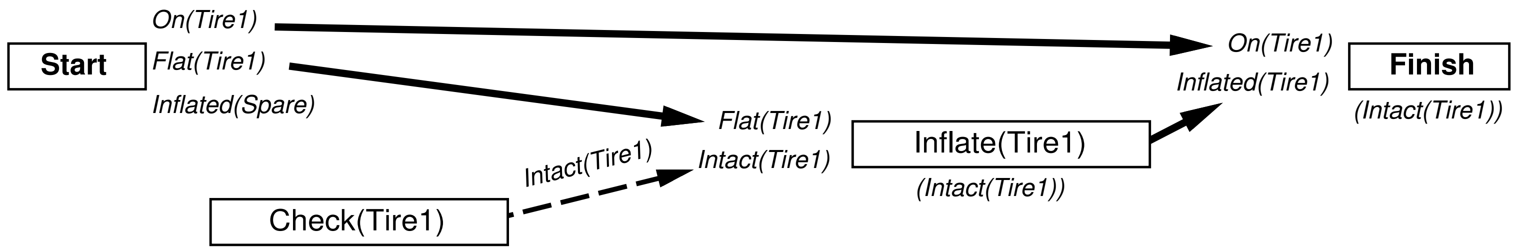
Der Agent muß beim Ausführen des Planes wissen (also erst zur Laufzeit) ob die entsprechende If-Bedingung gilt oder nicht. Daher führt man neue Aktionen zur Überprüfung ein:

Op(
$$\begin{array}{ll} Action: & Check\_Tire(x), \\ Prec: & Tire(x), \\ Effect: & \text{“Man weiss, ob } Intact(Tire) \text{ oder nicht.”} \end{array}$$
)

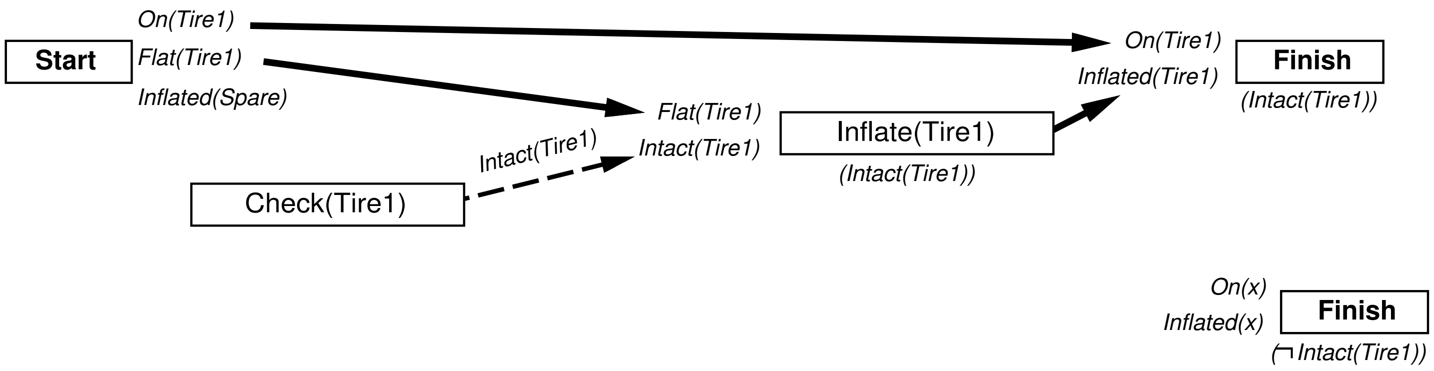
Definition 7.7 (Kontext)
Wir assoziieren mit jedem Schritt einen Kontext*: das ist die Menge der Bedingungen (Conditions), die gelten müssen, damit der Schritt ausgeführt werden kann.*



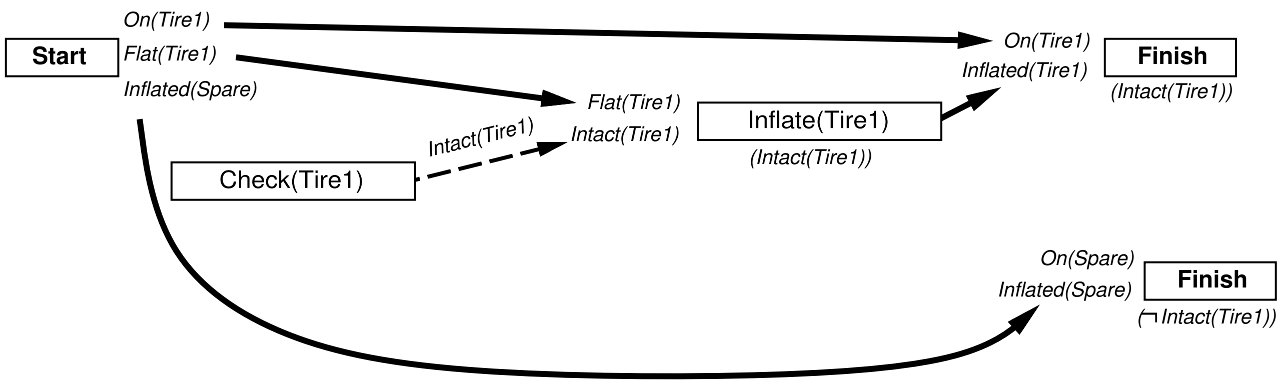
Hier würde POP schon backtracken, da $Flat(Tire_1)$ nicht gezeigt werden kann. Daher führt man eine neue Sorte von Links ein: *conditional links*.



Man beachte, daß alle Fälle abgedeckt werden müssen. Also:

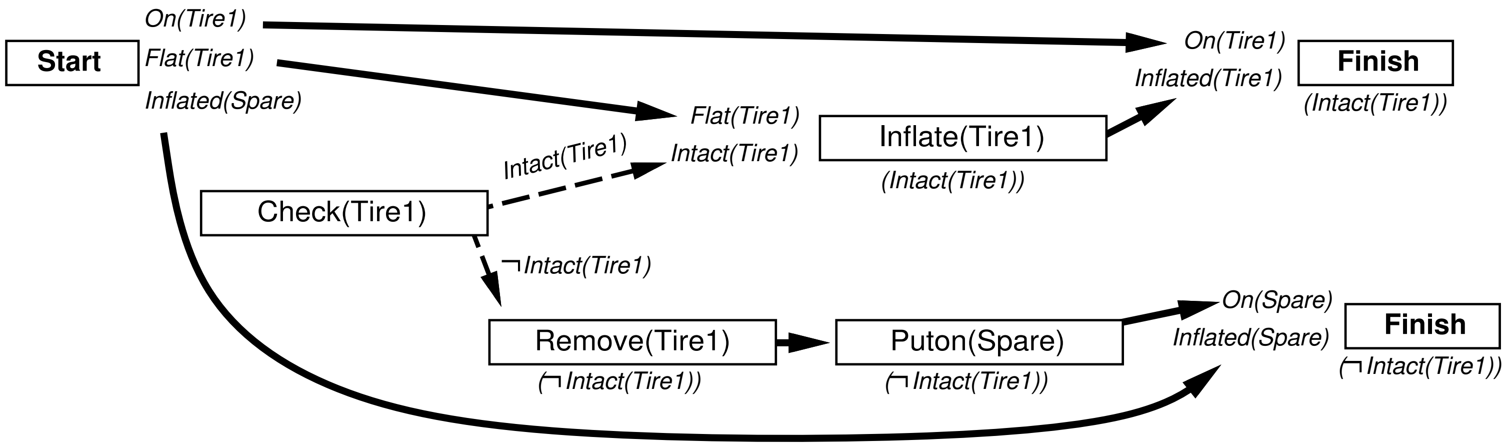


Wie können wir nun $Inflated(x)$ im FINISH-Schritt wahr machen? Das Hinzunehmen des Schrittes $Inflate(Tire_1)$ macht keinen Sinn, da ja die Vorbedingungen + Kontext inkonsistent miteinander sind.



Anstatt 13.06 könnte man auch einen $Inflate(Spare)$ Schritt einführen, aber das führt dann zu nichts.

Es fehlt jetzt nur noch $On(Spare)$.



Achtung: zunächst steht “True” unter $Remove(Tire_1), Put\_on(Spare)$. Dann gibt es aber eine Drohung mit dem oberen Link. **Diese kann aufgelöst werden, indem die Kontexte inkompatibel gemacht werden. Die entsprechenden Kontexte vererben sich.** Genauer: Suche einen konditionalen Schritt, dessen Nachbedingung die Kontexte inkompatibel macht und damit Drohungen auflöst.

```
function CPOP(initial, goals, operators) returns plan

  plan ← MAKE-PLAN(initial, goals)
  loop do
      Termination:
         if there are no unsatisfied preconditions
            and the contexts of the finish steps are exhaustive
             then return plan
      Alternative context generation:
         if the plans for existing finish steps are complete and have contexts C1 ... Cn then
             add a new finish step with a context ¬ (C1 ∨ ... ∨ Cn)
             this becomes the current context
      Subgoal selection and addition:
         find a plan step S_need with an open precondition c
      Action selection:
         choose a step S_add from operators or STEPS(plan) that adds c or
                 knowledge of c and has a context compatible with the current context
         if there is no such step
             then fail
         add S_add --c--> S_need to LINKS(plan)
         add S_add < S_need to ORDERINGS(plan)
         if S_add is a newly added step then
             add S_add to STEPS(plan)
             add Start < S_add < Finish to ORDERINGS(plan)
      Threat resolution:
         for each step S_threat that potentially threatens any causal link S_i --c--> S_j
                  with a compatible context do
             choose one of
                 Promotion: Add S_threat < S_i to ORDERINGS(plan)
                 Demotion: Add S_j < S_threat to ORDERINGS(plan)
                 Conditioning:
                    find a conditional step S_cond possibly before both S_threat and S_j, where
                        1. the context of S_cond is compatible with the contexts of S_threat and S_j;
                        2. the step has outcomes consistent with S_threat and S_j, respectively
                    add conditioning links for the outcomes from S_cond to S_threat and S_j
                    augment and propagate the contexts of S_threat and S_j
             if no choice is consistent
                    then fail
         end
  end
```

7.5 Erweiterungen

1. Ein Plan kann auch aus folgenden Gründen schiefgehen

- Aktionen können unerwartete Effekte haben, aber diese können aufgezählt werden (als Disjunktion).
- Die unerwarteten Effekte sind unbekannt. Dann muss man neu planen: *Replanning*.

```
function REPLANNING-AGENT(percept) returns an action
  static: KB, a knowledge base (includes action descriptions)
          p, an annotated plan, initially NoPlan
          q, an annotated plan, initially NoPlan
          G, a goal

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  current ← STATE-DESCRIPTION(KB, t)
  if p = NoPlan then
      p ← PLANNER(current, G, KB)
      q ← p
      if p = NoPlan or p is empty then return NoOp
  if PRECONDITIONS(p) not currently true in KB then
      p′ ← CHOOSE-BEST-CONTINUATION(current, q)
      p ← APPEND(PLANNER(current, PRECONDITIONS(p′), KB), p′)
      q ← p
  action ← FIRST(p)
  p ← REST(p)
  return action
```

D.h. man nimmt wahr und plant nur, wenn sich etwas verändert hat.

2. Man verbinde *Replanning* und *Conditional Planning*. Planen und Ausführung werden miteinander integriert.

```
function SITUATED-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
          G, a goal

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  current ← STATE-DESCRIPTION(KB, t)
  EFFECTS(START(p)) ← current
  if p = NoPlan then
     G ← ASK(KB, MAKE-GOAL-QUERY(t))
     p ← MAKE-PLAN(current, G, KB)
  action ← NoOp (the default)

   Termination:
     if there are no open preconditions and p has no steps other than START and FINISH then
        p ← NoPlan and skip remaining steps

   Resolving standard flaws:
     resolve any open condition by adding a causal link from any existing
        possibly prior step or a new step
     resolve potential threats by promotion or demotion

   Remove unsupported causal links:
     if there is a causal link START --c--> S protecting a proposition c
       that no longer holds in START then
        remove the link and any associated bindings

   Extend causal links back to earliest possible step:
     if there is a causal link S_j --c--> S_k such that
       another step S_i exists with S_i < S_j and the link S_i --c--> S_k is safe then
        replace S_j --c--> S_k with S_i --c--> S_k

   Remove redundant actions:
     remove any step S that supplies no causal links

   Execute actions when ready for execution:
     if a step S in the plan other than FINISH satisfies the following:
        (a) all preconditions satisfied by START;
        (b) no other steps necessarily between START and S; and
        (c) S does not threaten any causal link in p then
           add ordering constraints to force all other steps after S
           remove S from p, and all causal links to and from S
           action ← the action in S

  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```