Wissensbasiertes Planen

Planungsaufgabe

gegeben:

- **Beschreibung des Anfangszustandes**
- **Beschreibung des Zielzustandes**
- **Beschreibung der möglichen Aktionen**

gesucht:

- **Folge (bzw. partielle Ordnung) von Aktionen, deren Ausführung aktuellen Zustand in Zielzustand überführt**

Entwicklung effizienter Planungssysteme

1. STRIPS-Aktionsrepräsentation
2. Naive Vorwärts- und Rückwärtsplaner
3. Partial-Order Planer: Suche im Raum unvollständiger Pläne
4. Graphplan: Planen mit Plangraphen
5. Heuristisches Planen

Planen im Situationskalkül

Anfangszustand:
$At(Home, S0) \wedge \neg Have(Milk,S0) \wedge \neg Have(Bananas, S0) \wedge \neg Have(Drill,S0)$

Wirkung von Aktionen:

$$\forall a, s.\ Have(Milk,do(a,s)) \iff a = Buy(Milk) \wedge At(Supermarket,s) \vee Have(Milk,s) \wedge a \neq Drop(Milk)$$

Aktionsfolge zu Zielzustand ermittelt durch Anfrage:
$\exists s.\ At(Home, s) \wedge Have(Milk, s) \wedge Have(Bananas, s) \wedge Have(Drill, s)$

Theorembeweiser liefert folgende Bindung für s:

do(Go(Home),
do(Buy(Drill),
do(Go(HardwareStore),
do(Buy(Banana),
do(Buy(Milk),
do(Go(Supermarket), S0))))))

Funktioniert im Prinzip, aber extrem ineffizient

1. STRIPS

versucht Ineffizienz durch Einschränkung der Sprache zu begegnen

Zustand: Menge von Grundatomen
Ziele: Menge von Atomen, kann Variablen enthalten:
At(x), Sells(x, Milk) entspricht: $\exists x. At(x) \wedge Sells(x, Milk)$

Beschreibung von Operatoren (= Aktionstypen) durch

Name	
Vorbedingung	**kann Aktion ausgeführt werden?**
Add-List	**welche Atome werden wahr?**
Delete-List	**welche Atome werden ungültig?**

Operatoren können Variablen enthalten, d.h. beschreiben Aktionstypen
ausführbare Aktionen immer Grundinstanzen von Operatoren

Variablen in Zielen nicht unbedingt nötig (später)

Einkaufen mit STRIPS

Operatoren

Name	Vorbedingung	Add-List	Delete-List
Buy(x)	At(y), Sells(y,x)	Have(x)	
Go(x, y)	At(x), ¬(x = y)	At(y)	At(x)

Hintergrundwissen: Sells(SM, Milk), Sells(SM, Bananas), Sells(HS, Drill)

Zustand	Aktion
At(Home)	Go(Home,SM)
At(SM)	Buy(Milk)
At(SM), Have(Milk)	Buy(Bananas)
At(SM), Have(Milk), Have(Bananas)	Go(SM,HS)
At(HS), Have(Milk), Have(Bananas)	Buy(Drill)
At(HS), Have(Milk), Have(Bananas), Have(Drill)	Go(HS,Home)
At(Home), Have(Milk), Have(Bananas), Have(Drill)	

Grundbegriffe

Sei a = (pre(a), add(a), delete(a)) eine Aktion mit pre(a) = pos(a) $\cup$ neg(a) Menge von positiven bzw. negativen Literalen, add(a) und delete(a) Mengen von Atomen; sei s ein Zustand:

- **a anwendbar in s falls pos(a) $\subseteq$ s und m $\notin$ s für jedes $\neg$m $\in$ neg(a),**

- **falls a in s anwendbar, so entsteht durch Ausführen von a in s der Zustand**

$$\gamma(a,s) = [s \cup add(a)] \setminus delete(a).$$

Sei g eine Menge von Zielen (ohne Variablen):

- **a ist relevant für g, falls add(a) $\cap$ g $\neq \emptyset$ und delete(a) $\cap$ g = $\emptyset$,**

- **falls a für g relevant ist, so ist die Zwischenzielmenge für a und g**

$$\gamma^{-1}(a,g) = [g \setminus add(a)] \cup precond(a),$$

- **falls a relevant für g und γ^{-1}(a,g) $\subseteq$ s, dann ist g $\subseteq$ γ(a,s).**

Eliminieren von Variablen aus Zielen

Das Ziel

$\exists x. At(x) \wedge Sells(x, Milk)$

wird ersetzt durch neues Symbol, etwa

Goal

dazu neuer Operator mit

Pre: At(x), Sells(x, Milk)
Add: Goal
Delete: leer

2. „Naive“ Vorwärts- und Rückwärtsplaner

Wie findet man die "richtige" Aktionsfolge für Ziel g_0?

Aktionen generieren Situationsraum

Progressionsplaner:

Wurzel Startzustand s_0
mit a beschriftete Kante von s zu s‘, falls a in s anwendbar und s‘ = $\gamma(a,s)$
Zielzustand jeder Zustand s mit $g_0 \subseteq s$

Problem: oft hoher Verzweigungsgrad

Regressionsplaner:

Wurzel Ziel g_0
mit a beschriftete Kante von g zu g‘, falls a relevant für g und g‘ = $\gamma^{-1}(a,g)$
Abbruchkriterium: Erreichen einer Zielmenge g mit $g \subseteq s_0$

Problem: verschiedene Ziele können interagieren

originales STRIPS-System ist unvollständiger Regressionsplaner

Nichtdeterministischer Vorwärtsplaner

Forward-Search(O, s_0, g)

s := s_0;

π := empty plan;

loop

> **if s satisfies g (g ⊆ s) then return π;**
>
> **applicable := {a | a instance of an operator in O applicable in s};**
>
> **if applicable = ∅ then return failure;**
>
> **nondeterministically choose an action a ∈ applicable;**
>
> **s := γ(a,s);**
>
> **π := π.a;**

Nichtdeterministischer Rückwärtsplaner

Backward-Search(O, s_0, g)

π := empty plan;

loop

> **if s_0 satisfies g ($g \subseteq s_0$) then return π;**
>
> **relevant := {a | a instance of an operator in O relevant to g};**
>
> **if relevant = $\emptyset$ then return failure;**
>
> **nondeterministically choose an action a $\in$ relevant;**
>
> **g := $\gamma^{-1}(a,g)$;**
>
> **π := a.π;**

3. Suche im Raum partieller Pläne

Grundidee:

- Anwendung von Verfeinerungsoperatoren, die bisherigen Plan erweitern
- dabei möglichst wenig Festlegungen (Reihenfolge und Variablenbindung)
- Starten mit dem unvollständigsten Plan
- Terminierung wenn vollständiger, konsistenter Plan erzeugt ist

Was macht partiellen Plan unvollständig?

- Teilziele noch nicht erreicht
- Mit Plan kompatible Reihenfolge der Aktionen löscht nötiges Atom

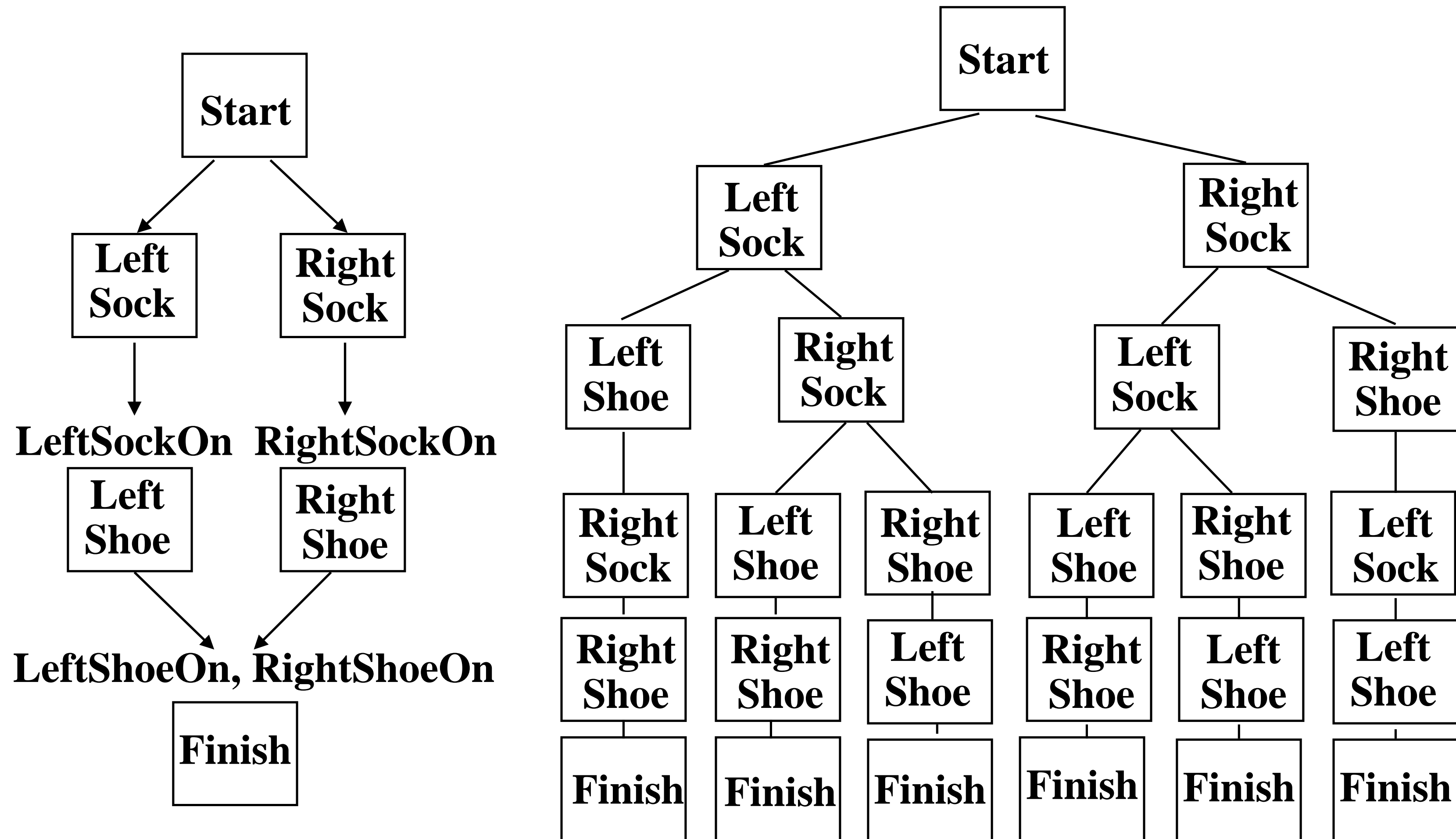
Wie kann man verfeinern?

- Aktion einfügen; Reihenfolge festlegen; Variable binden; neuer causal link

Ziel: Planen mit möglichst wenig Backtracking

Partiell geordnete Pläne

Anziehen von Socken und Schuhen: partiell geordneter Plan und Linearisierungen



Was ist ein (partieller) Plan?

Datenstruktur bestehend aus:

* **einer Menge von Planschritten, d.h. Operatoren**
* **einer (partiellen) Ordnungsrelation auf den Planschritten**
* **einer Menge von Variablenbindungen**
* **einer Menge von kausalen Verbindungen (causal links), die den Zweck der Schritte im Plan festhalten: causal link von Si zu Vorbedingung c von Sj bedeutet: Si wird ausgeführt, um diese Vorbedingung wahr zu machen.**

Partiell Geordnete Pläne, ctd

Lösung eines Planungsproblems: ein vollständiger, konsistenter Plan

Ein Plan ist vollständig, wenn alle Vorbedingungen aller Aktionen erreicht sind, d.h. :

wenn c Vorbedingung von Sj, dann gilt

1) es gibt Si < Sj mit Effekt c (causal link von Si zu c)

2) es gibt keine Linearisierung, so dass Si < Sk < Sj und Sk löscht c

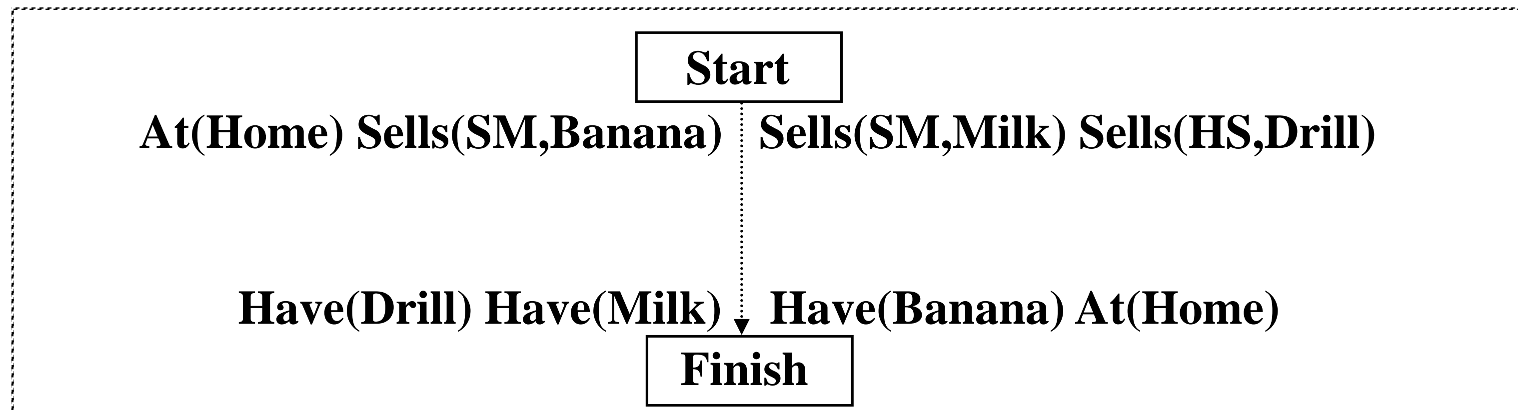
Ein Plan ist konsistent, wenn er keine widersprüchliche Information bzgl. Ordnung und Variablenbindungen enthält

Grundidee des Verfeinerungsansatzes:

starte mit unvollständigem Plan

vervollständige ihn schrittweise, ohne ihn inkonsistent zu machen

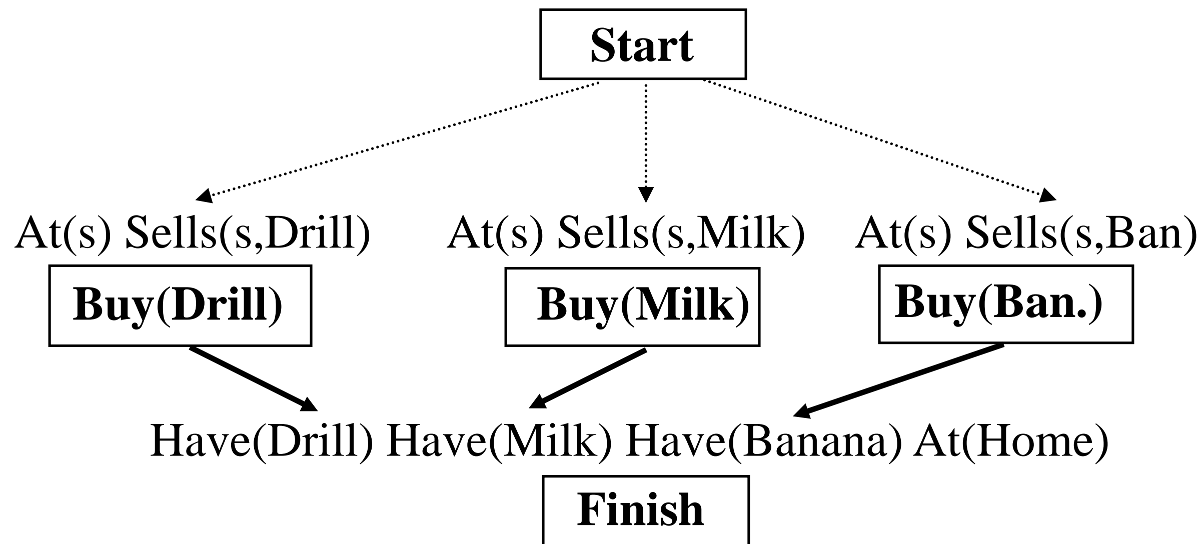
Einkaufen: der unvollständigste Plan



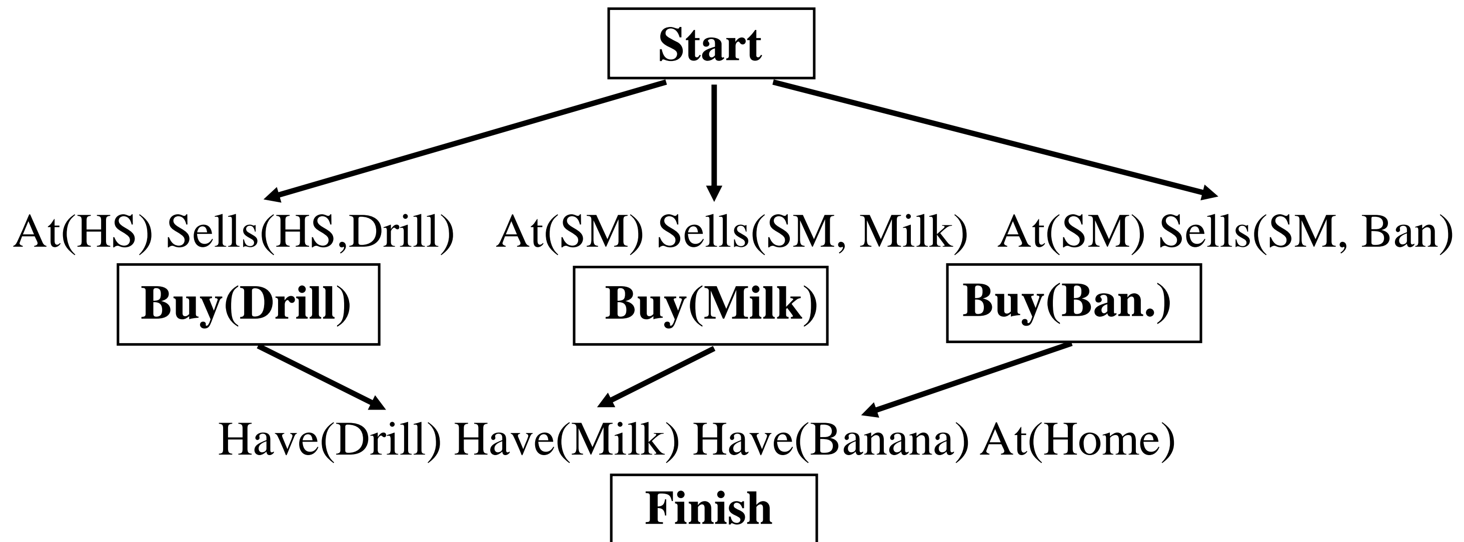
Unvollständig, weil nicht alle Vorbedingungen von Finish erfüllt

=> Einfügen von weiteren Aktionen

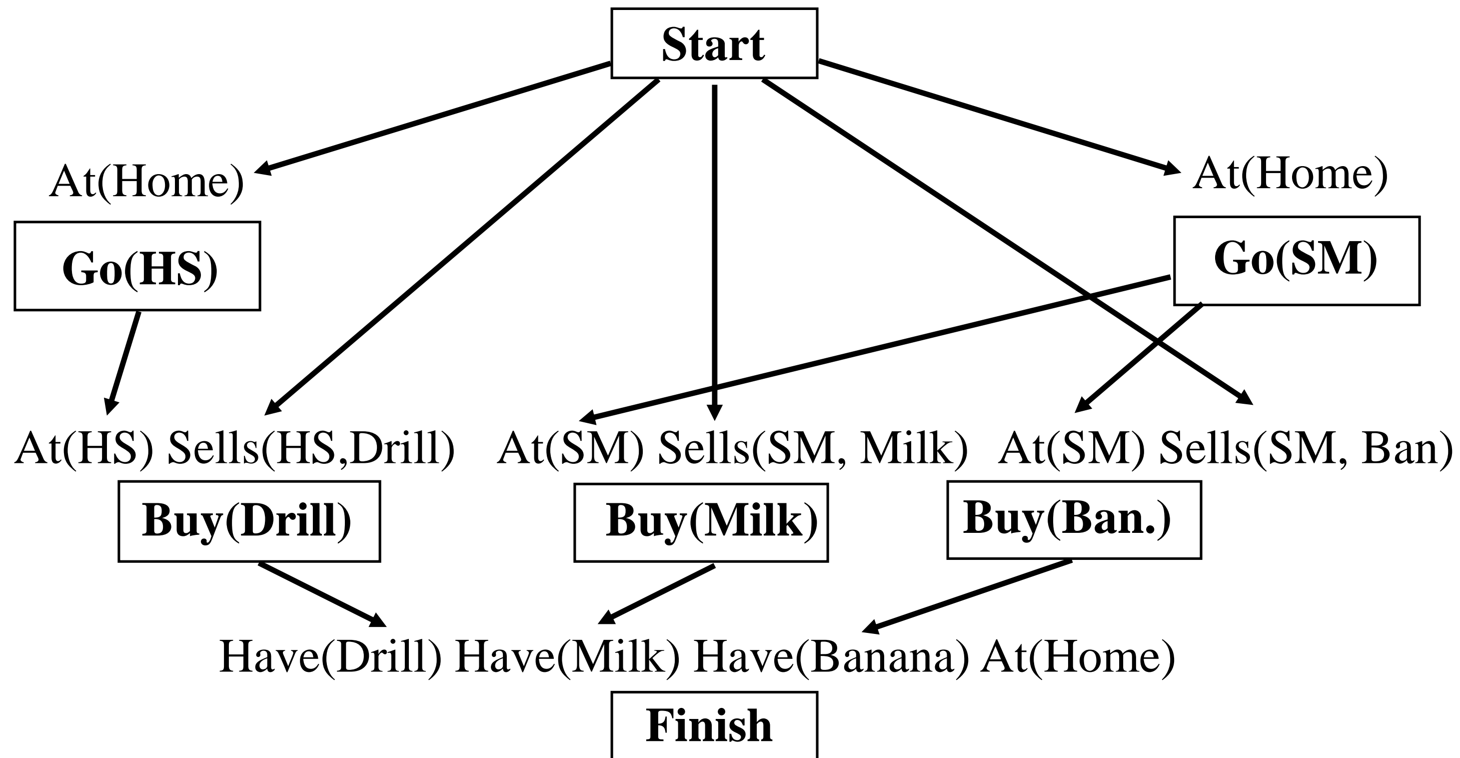
(Binden von Variablen bzw. Festlegen von Reihenfolge nicht anwendbar)



Verfeinerung, die 3 Zielbedingungen erfüllt, dicke Pfeile "causal links" zu Vorbedingungen späterer Aktionen, diese werden "geschützt". Unten Instanzierung:



Ein unvollständiger, nicht verfeinerbarer Plan



Problem manchmal durch Umordnen zu beheben
hier nur durch Rücknahme einer früheren Entscheidung (backtracking):
alternative Variablenbelegung der Vorbedingung von Go(SM)

Weitere Verfeinerung

Start

At(Home)

Go(HS)

At(HS)

Go(SM)

At(HS) Sells(HS,Drill)

Buy(Drill)

At(SM) Sells(SM, Milk)

Buy(Milk)

At(SM) Sells(SM,Ban)

Buy(Ban.)

At(SM)

Go(Home)

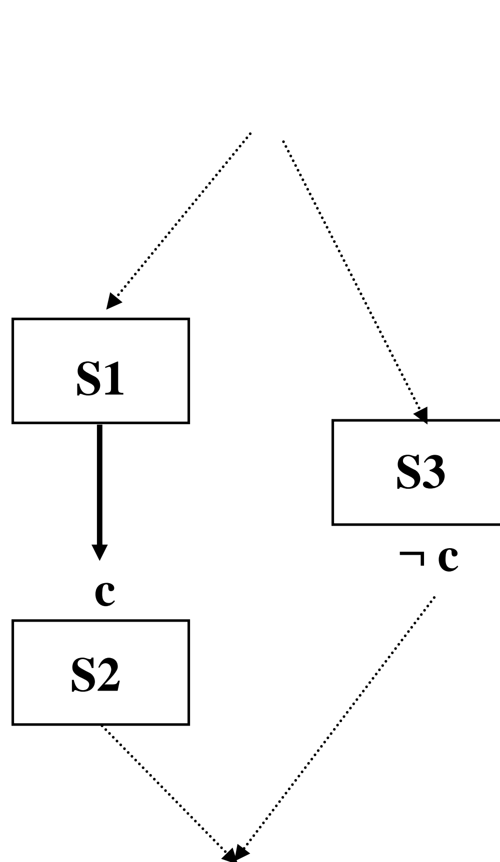
Have(Drill) Have(Milk) Have(Banana) At(Home)

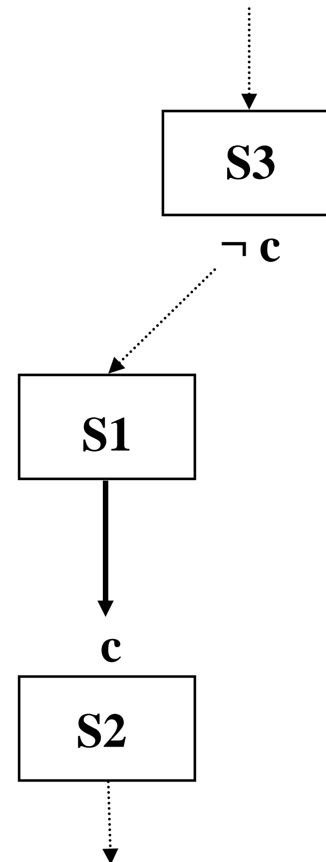
Finish

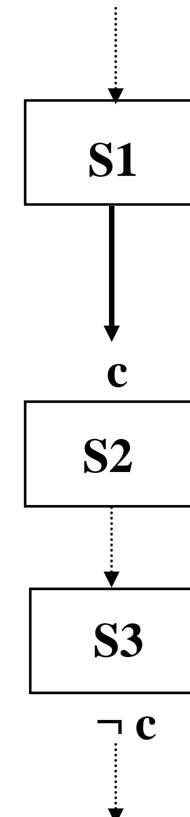
Aber: **Go(SM) kann vor Buy(Drill) ausgeführt werden, Go(Home) vor Buy(Milk) oder Buy(Ban.)**

Schutz von kausalen Verbindungen

S2 braucht c, delete-list von S3 enthält c

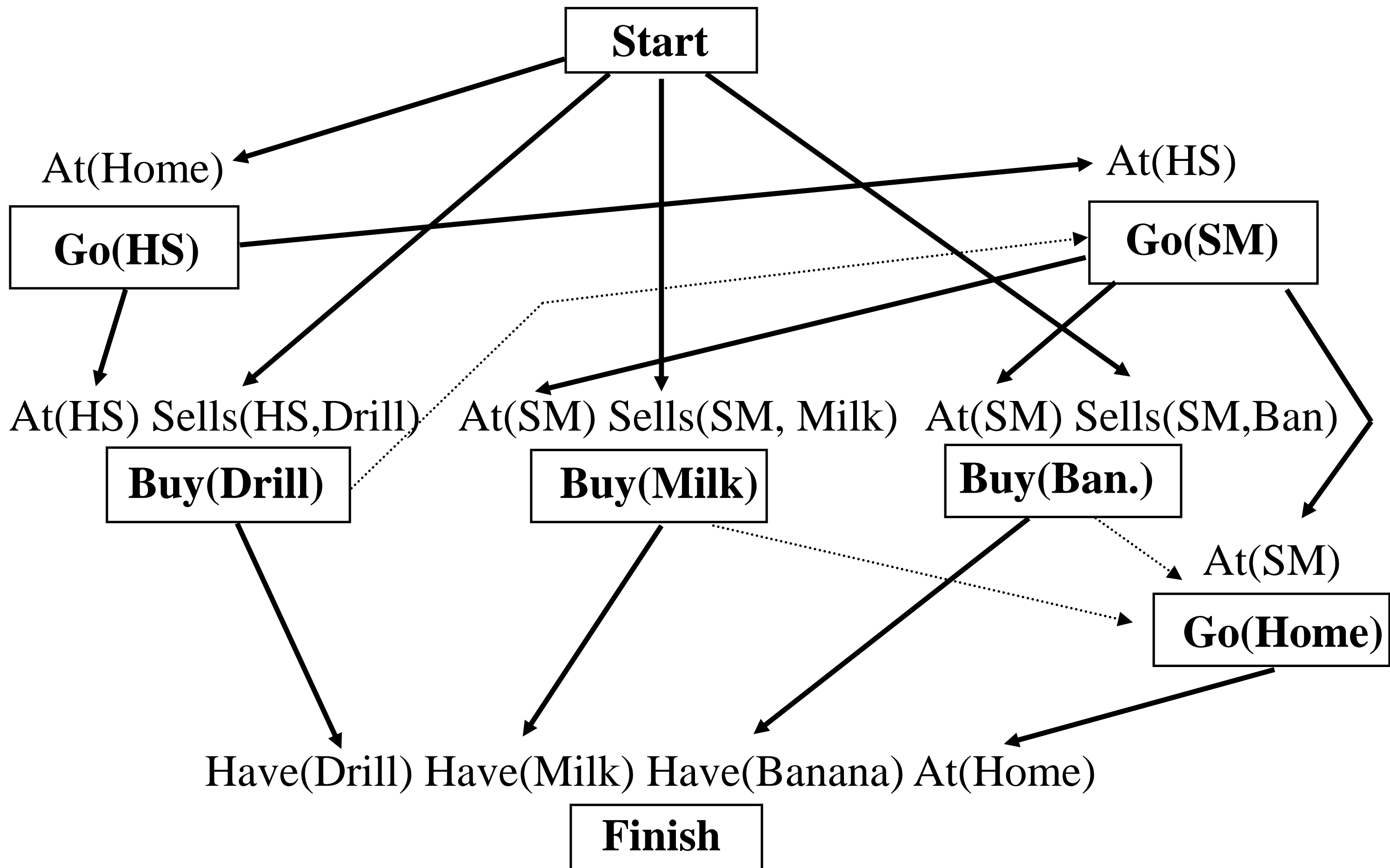






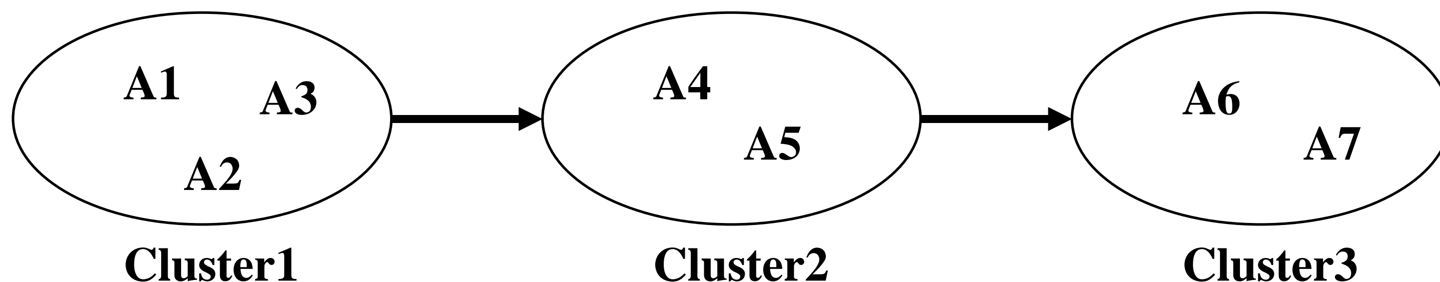
unvollständiger Plan **Demotion** **Promotion**

Der endgültige Plan



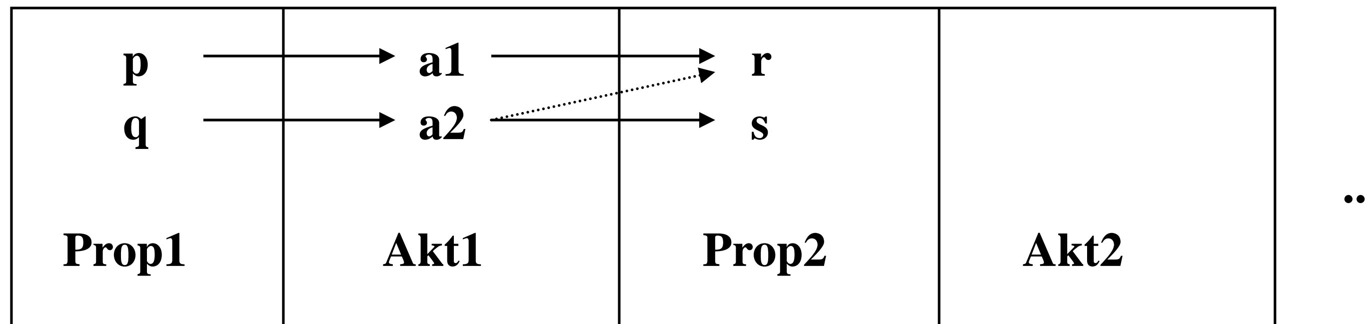
4. Graphplan

- **STRIPS-Repräsentation für Aktionen und Zustände**
- **verwendet Plangraphen, um Suche möglichst einzuschränken**
- **Pläne nicht notwendig total geordnet (wie beim Vorwärtsplanen)**
- **aber auch nicht beliebige partielle Ordnungen (wie PO-Planning)**
- **sondern Zwischenlösung:**
  - **Es gibt Cluster von Aktionen,**
  - **Reihenfolge innerhalb Cluster beliebig**
  - **Cluster selbst total geordnet**



Der Plangraph

- enthält alle möglicherweise ausführbaren Aktionen bzw. wahren Atome
- berücksichtigt nicht alle Interaktionen
- erzeugt in polynomialer Zeit
- führt zu erheblicher Einschränkung des Suchraums



Der Plangraph

gerichteter Level-Graph (Kanten aus einem Level nur zum nächsten Level)

2 Arten von Knoten (Aktionen und Propositionen)

3 Arten von Kanten (add, delete, precondition)

1. **Aktions-Level Akt_i folgt auf Propositions-Level $Prop_i$ Propositions-Level $Prop_{i+1}$ auf Aktions-Level Akt_i.**
2. **jede Aktion in Akt_i hat (eingehende) Kanten von ihren Vorbedingungen in $Prop_i$, sowie Kanten zu den Elementen ihrer add und delete-Listen in $Prop_{i+1}$.**
3. **A kommt in Akt_i vor, wenn alle Vorbedingungen in $Prop_i$ vorkommen und sich dort nicht ausschließen, und dann kommen auch alle Nachbedingungen von A in $Prop_{i+1}$ vor.**

Aktionen no-op(a): einmal erzeugtes Atom a bleibt in allen weiteren Levels erhalten

Wechselseitiger Ausschluss

Aktionen:

Interferenz: A_1 und A_2 schließen sich aus, wenn A_1 (bzw. A_2) eine Vorbedingung oder einen Add-Effekt von A_2 (bzw. A_1) löscht.

Inkompatible Vorbedingungen: A_1 und A_2 schließen sich aus in Akt_i, wenn A_1 eine Vorbedingung a benötigt, A_2 eine Vorbedingung b, und a und b schließen sich aus in $Prop_i$.

Propositionen:

p und q schließen sich aus in $Prop_i$, wenn jede Aktion in Akt_{i-1} mit add-Kante zu p jede Aktion in Akt_{i-1} mit add-Kante q ausschließt

Graphplan-Algorithmus

i := 1;

$Prop_1$:= Anfangszustand;

while kein Plan gefunden [und letzter Level nicht gleich vorletztem] do

> **erweitere Plangraph um Akt_i und $Prop_{i+1}$;**
>
> **prüfe, ob es gültigen Plan der Länge i gibt;**
>
> **i := i + 1;**

if Plan gefunden then gib Plan aus else ‚kein Plan existiert'

Beachte: Plan der Länge i kann mehr als i Aktionen haben (parallele Aktionen)!

Erweiterung um Aktions-Level Akt_i:

1. **Füge Aktion A in Akt_i ein, wenn alle Vorbedingungen von A in $Prop_i$ enthalten und nicht als sich ausschließend markiert sind.**
2. **Füge alle no-op Aktionen und precondition links ein.**
3. **generiere für jede Aktion A in Akt_i Liste aller Aktionen, die A ausschließt**

Erweiterung um Propositions-Level $Prop_{i+1}$:

1. **Füge alle Atome aus Nachbedingungen der Aktionen in Akt_i ein, zusammen mit entsprechenden Add- und Delete-Links**
2. **Markiere p, q in $Prop_{i+1}$ als sich ausschließend, wenn alle Aktionen, die p generieren, und alle, die q generieren, sich ausschließen**

Plansuche auf Basis des Plangraphen

Gegeben Menge g von Zielen in t

> **suche sich nicht ausschließende Menge von Aktionen in t-1, die alle Ziele in g wahrmachen.**
>
> **deren Vorbedingungen sind Ziele in t-1, usw.**

erhebliche Performanzsteigerung gegenüber Planraum-Planern:

> **wechselseitiger Ausschluss, parallele Pläne, Merken unerreichbarer Zielmengen, keine Instanzierung bei Suche**

Weiterer Vorteil: Terminierung bei unlösbaren Problemen:

> **es muss Propositions-Level geben, so dass alle weiteren identisch sind (dieselben Propositionen und Ausschlüsse)**
>
> **Wenn Plan nicht bis dahin gefunden, so gibt es keinen**

5. Planen mit heuristischer Suche: der FF-Planer

- „naive“ Vorwärtsplaner durchsuchen Zustandsraum „blind“
- keine Heuristiken für Auswahl des zu expandierenden Zustands
- für das Planen gibt es höchst informative Heuristiken
- Erfolg von FF (bestes System planning competitions 2000/2002) beruht auf

  Verwendung einer Heuristik basierend auf relaxiertem Problem;
  Heuristik schätzt, wie viele Aktionen nötig sind, um von Zustand zu Zielzustand zu kommen;
  berücksichtigt positive Interaktionen von Aktionen

  Verwendung einer geeigneten lokalen Suchstrategie, die Aspekte systematischer Suche verwendet, um lokale Minima zu vermeiden

- Vollständigkeit und Optimalität zugunsten von Effizienz aufgegeben

Local Search

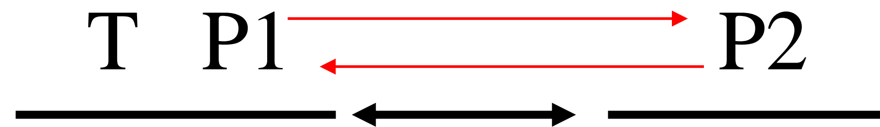
(nachfolgende englische Folien von J. Hoffmann, dem Entwickler von FF)

- State space = set of (world) states reachable from the initial state
- State spaces are exponentially large => heuristic *h* estimates goal distance
- Local Search: iteratively try to make an improvement on *h* by looking at local surroundings
- We define: a heuristic and a search scheme

FF - Heuristic

- General idea: relax problem, solve relaxation in each search state
- In STRIPS planning: ignore negative effects [Bonet et al. 1997]
- Optimal relaxed solution length „*h+* “ admissible but NP-hard to compute [Bylander 1994]
- Implementations use approximation techniques:
  - Bonet et al: approximate *h+* by additionally ignoring positive interactions
  - FF: approximate *h+* by running a relaxed version of GRAPHPLAN [Blum & Furst 1997]
    h(s) = number of actions in plan for the relaxed problem computable in polynomial time

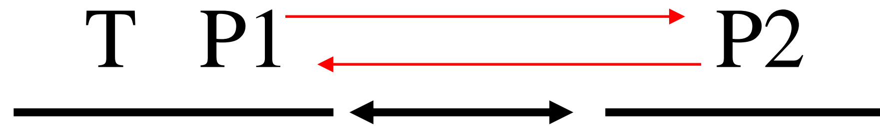
Example: Exchange P1 and P2 Using Truck T



- Plan:

Load T P1 Left
Drive T Left Right
Unload T P1 Right
Load T P2 Right
Drive T Right Left
Unload T P2 Left

Relaxed Example: No Delete Lists: Left True After Drive T Left Right



- Relaxed Plan: Load T P1 Left
  Drive T Left Right
  Unload T P1 Right
  Load T P2 Right
  Unload T P2 Left

FF - Search

- "Enforced hill-climbing":
  - start in S = initial state, stop when $h(S) = 0$
  - in a state S, do breadth first search for S' such that $h(S') < h(S)$
- Motivation: in experiments, often better states nearby, but not among immediate successors

6. Erweiterungen der bisherigen Ansätze

- **Hierarchisches Planen = Planen mit unterschiedlichem Abstraktionsgrad**

Go(HS) - Buy(Drill) - Go(SM) - Buy(Milk) - Buy(Ban.) - Go(Home)

StandUp - Take(Key) - Open(Door) - Walk(Outside) - Close(Door) ...

- **Situationsabhängigen Effekte**

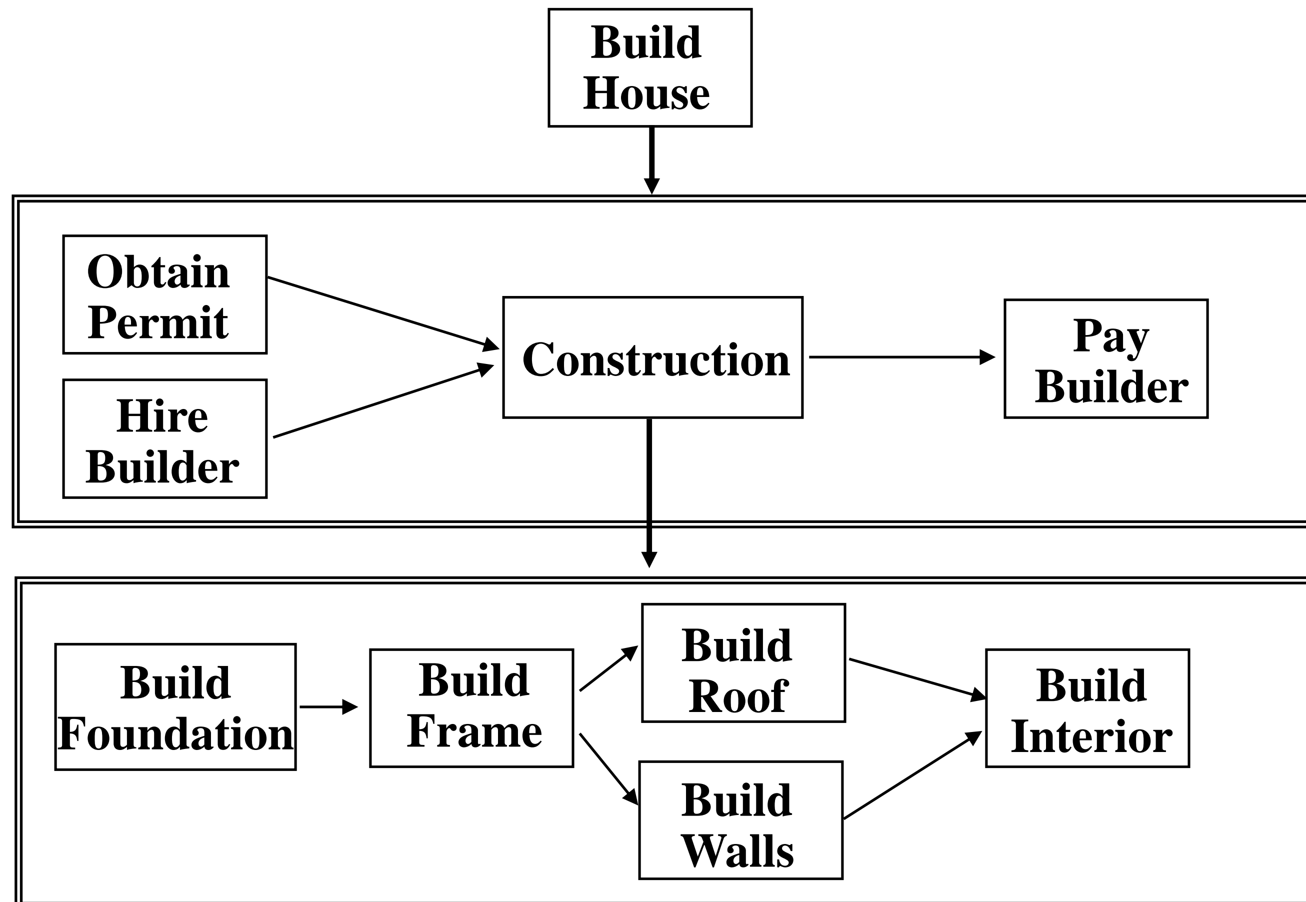
  Drehe(Lichtschalter) ==> Effekt: Licht an falls vorher aus, Licht aus sonst
- **Zeit**

  Aktionen haben Dauer, unterliegen Zeitbeschränkungen
- **Ressourcen**

  Aktionen verbrauchen Ressourcen, z.B. Kaufen kostet Geld

  können auch Ressourcen generieren, z.B. zur Bank gehen

Hierarchisches Planen: Beispiel



Operator Dekomposition

Plan P ist korrekte Dekomposition eines Operators O wenn

1) P konsistent ist

2) jeder Effekt von O in einem Schritt von P erreicht und nicht wieder zerstört wird

3) jede Vorbedingung eines Schrittes in P durch vorherigen Schritt erzielt wird oder Vorbedingung von O ist, und vor Ausführung des Schrittes nicht zerstört wird.

Sinnvoll, wenn wenig Interaktion zwischen einzelnen Dekompositionen

Wiederverwertbarkeit, Plan-Bibliotheken

Dekomposition solange, bis ausführbare Aktionen erreicht sind

Hier. Planen funktioniert, wenn eine der folgenden Bedingungen gilt:

1) für jede abstrakte Lösung gibt es eine zugehörige ausführbare Lösung (downward solution property, DSP)

2) kein inkonsistenter abstrakter Plan hat eine zug. ausführbare Lösung (upward solution property, USP)

Kostenersparnis durch HP: ein Beispiel

Annahmen:

es gibt Lösung mit 64 Schritten, jeweils 3 Operatoren anwendbar

Standardansatz: im schlimmsten Fall 3^{64} Operationen zu überprüfen

exponentiell in der Länge der Lösung

Hierarchischer Ansatz:

falls jeder abstrakte Operator in 4 Schritte dekomponiert wird, und es jeweils 3 Dekompositionen gibt, von denen genau 1 eine Lösung ist, dann sind im schlimmsten Fall $3 * 4 + 3 * 4 * 4 + 3 * 4 * 4 * 4$ Operationen zu überprüfen.

exponentiell in der Dekompositionstiefe

Bedingte Pläne

Beispiel Reifenpanne: Reifen R1 ist platt, es gibt Reserverad RR

Lösung: [Remove(R1),PutOn(RR)]

Aber was, wenn Reifen1 nicht kaputt ist, sondern nur unaufgepumpt?

Bessere Lösung: bedingter Plan

If (Intact(R1), [Inflate(R1)], [Remove(R1),PutOn(RR)])

Unterschiedliche Teilpläne, die bei Planausführung entsprechend den Bedingungen ausgewählt werden

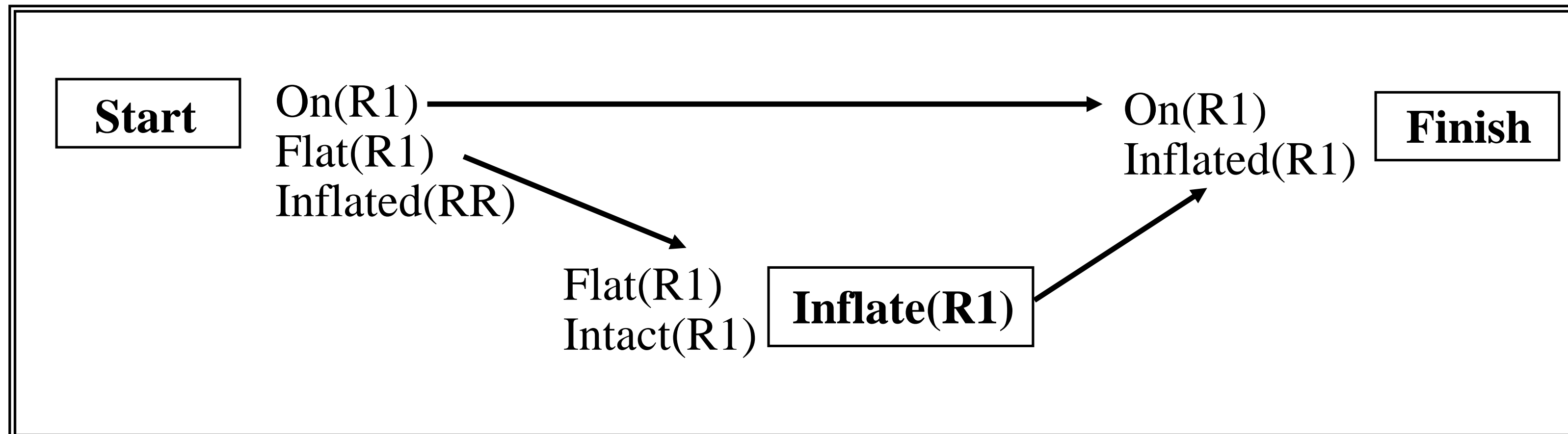
Gelten der Bedingungen muss bei Ausführung bekannt sein, evtl. müssen Aktionen (Wahrnehmungen, Tests) in Plan aufgenommen werden, die notwendige Information liefern

Repräsentation von Plänen wird um Kontexte für Aktionen erweitert

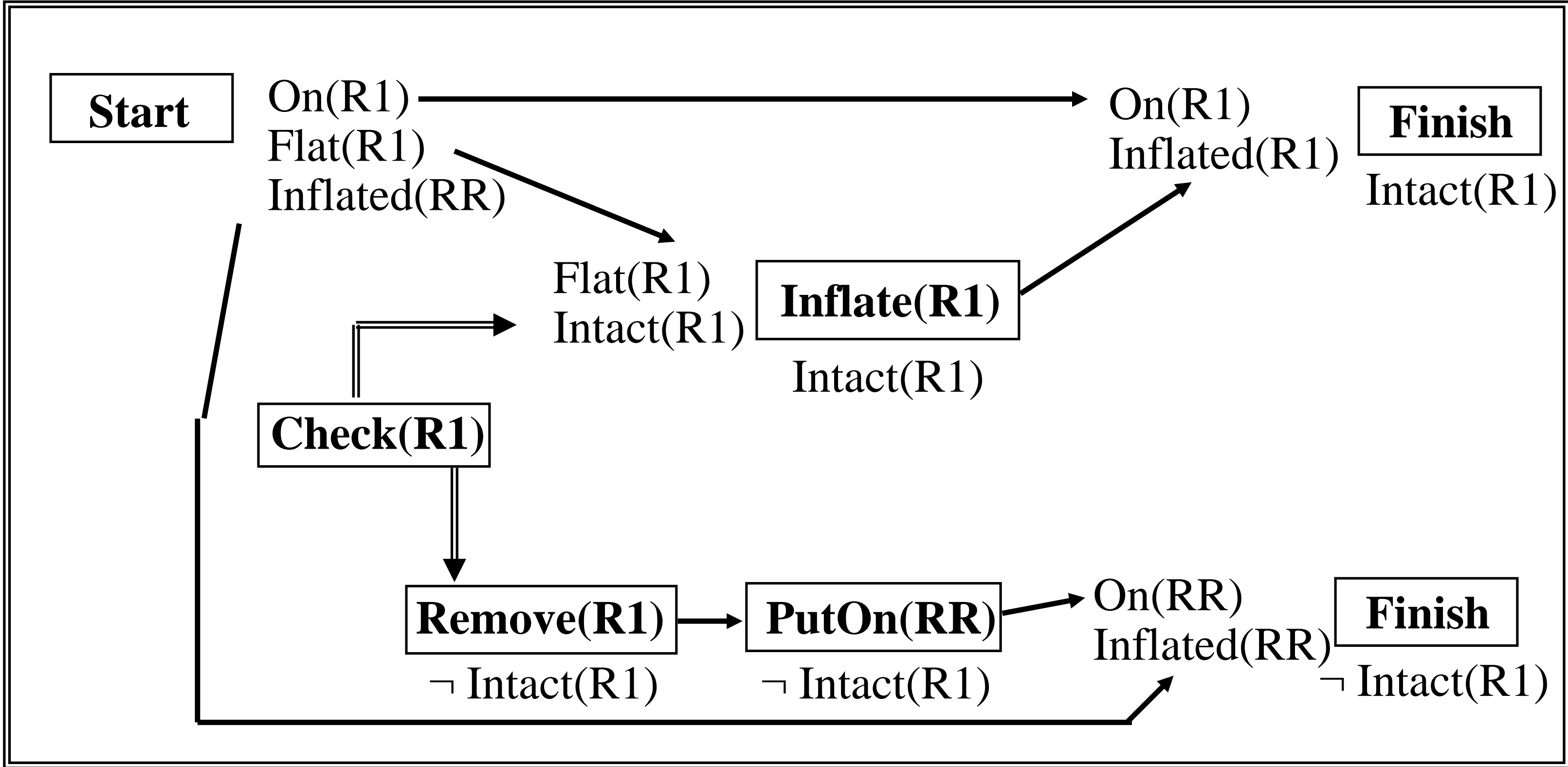
Ein Kontext ist eine Menge von Bedingungen, die gelten müssen, damit genau dieser Schritt ausgeführt wird

Bedingte Pläne: ein Beispiel

Ausgangssituation, wird durch Auswahl von Inflate und Instanziierung zu folgendem partiellen Plan. Da Intact(R1) nicht erzeugt werden kann, ist Standardansatz hier am Ende.



Einfügen einer Testaktion



Vollständiger Plan (bis auf Vorbed. Remove, PutOn)
Kontext jeweils unter der zugehörigen Aktion

Zusammenfassung Planen

- STRIPS Sprache (und einfache Erweiterungen) immer noch vorherrschend im Planungsbereich
- naive Vorwärts- und Rückwärtsplaner zu ineffizient
- Suche im Planraum statt im Zustandsraum führt oft zu besseren Ergebnissen
- wird häufig noch übertroffen von Ansätzen, die auf Plangraphen basieren
- Verwendung geeigneter Heuristiken und lokaler Suchstrategien liefert derzeit die besten Resultate bzgl. Effizienz
- Planer, die auf erweiterten STRIPS Sprachen beruhen, werden heute in komplexen Bereichen eingesetzt (Missionsplanung in der Raumfahrt, Fabrikation, etc.)