

# Intelligente Handlungsplanung

## 3. Hierarchisches Planen

Prof. Dr. Susanne Biundo-Stephan

Institut für Künstliche Intelligenz, Universität Ulm

SS 2008



# Inhalt

- 1 Situationsabstraktion
- 2 Operatorabstraktion
- 3 Hierarchisches Planen

# Situationsabstraktion I

- Einführung von Abstraktionsebenen in den Zustandsraum
- Konstruktion von Plänen auf der Basis abstrakter Zustände
- Schrittweise Verfeinerung

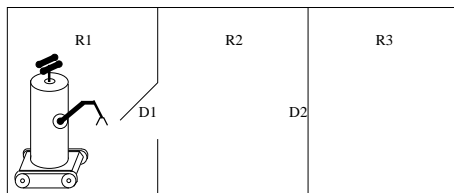
## *ABSTRIPS*

(Sacerdoti 1974)

Den Vorbedingungen der Operatoren werden *Kritikalitäten* zugeordnet. Je “unwichtiger” eine Vorbedingung oder je einfacher sie zu erreichen ist, desto geringer ist ihre Kritikalität.

# Situationsabstraktion II

## Beispiel:



*init:* In(R1), Open(D1), Closed(D2), Connects(R1,D1,R2),  
Connects(R2,D2,R3)

*goal:* In(R3)

*O:* **goto** ( $r_1, d, r_2$ )

*prec* : In( $r_1$ ), Open( $d$ ), Connects( $r_1, d, r_2$ )

*add* : In( $r_2$ )

*del* : In( $r_1$ )

**open** ( $d$ )

*prec* : Closed( $d$ )

*add* : Open( $d$ )

*del* : Closed( $d$ )

## Situationsabstraktion III

Konstruktion eines abstrakten Planes unter der Annahme, dass alle Vorbedingungen *unter* einer bestimmten Kritikalität bereits erfüllt sind.  
in(R3) wird erfüllt durch  
goto(R1,D1,R2) ; goto(R2,D2,R3).

- ⇒ Konstruktion eines detaillierteren Planes, der weitere Vorbedingungen des ersten Operators im abstrakten Plan erzeugt.
- ⇒ Konstruktion eines Planes für weitere Vorbedingungen des zweiten Operators im abstrakten Plan etc.

Vorbedingungen der Operatoren im abstrakten Plan

- ⇒ Inseln im Suchraum
  - Erzeugung der Inseln auf der ersten abstrakten Planungsebene
  - Herabsetzen der Kritikalität um 1
- ⇒ goto(R1,D1,R2); open(D2); goto(R2,D2,R3)

# Situationsabstraktion IV

**Strategie:** “*length-first*”

Auf jeder Abstraktionsebene wird ein kompletter Plan erzeugt.

**Alternative:** “*depth-first*”

- Kompletter Plan auf der ersten Ebene
- Identifikation der Inseln
- Entwicklung eines kompletten, *konkreten* Planes, der zur ersten Insel führt
  - ⇒ erstes, ausführbares Anfangsstück  
z.B. *sense-plan-act* Zyklus
- Ausführung des Anfangsstücks
- *Replanning* und Orientierung am ursprünglich erzeugten abstrakten Plan

# Situationsabstraktion V

- Feste Zuordnung der Kritikalitätswerte durch den Benutzer
  - ▶ partielle Ordnung auf den Prädikaten
  - ▶ gibt es einen “kurzen” Plan?
- Keine Flexibilität in Bezug auf das aktuelle Planungsproblem

# Situationsabstraktion VI

*ALPINE* (Knoblock 1994)

Vollautomatische Generierung von Abstraktionen aus der Problemspezifikation

Prinzip der “*ordered monotonicity*”

- Statische Bedingungen haben die höchste Kritikalität
- Interagierende Vorbedingungen liegen auf der gleichen Abstraktionsebene

## **Grund:**

Mögliche Konflikte sollen nicht durch Verfeinerung induziert werden

*siehe Mitschrift*

# Hierarchisches Planen I

**Ziel:** Lösung von *realistischen* Planungsproblemen

## **Anforderungen:**

- Ausdrucksstarker Repräsentationsformalismus
- Hierarchische Strukturierung des Planungsproblems
- Verwendung vorstrukturierter, abstrakter Lösungen
- Berücksichtigung von Ressourcen

## **Planung komplexer Vorhaben:**

- Rundreisen
- Projekte
- Prozesse
- Hilfseinsätze

# Hierarchisches Planen II

- ⇒ Automatische und flexible Generierung umfangreicher Pläne
- ⇒ Nachvollziehbarkeit des Lösungsweges
- ⇒ Verwendung / Konkretisierung abstrakter Benutzerpläne
- ⇒ Vorgabe unterschiedlicher Lösungswege

# Hierarchische Dekomposition I

Planen auf verschiedenen Abstraktionsebenen

[*Go(Supermarket)*, *Buy(Milk)*, *Buy(Bananas)*, *Go(Home)*]

⇒ abstrakte Beschreibung der notwendigen Aktionen

[*Forward(1 cm)*, *Turn(1 degree)*, *Forward(1.5cm)*, ...]

⇒ tatsächlich auszuführende Aktionen

Planungssysteme für praktische Anwendungen verwenden

## Hierarchische Dekomposition

Dabei werden *abstrakte* Operatoren in einzelne Schritte zerlegt. Sie beschreiben (auf niedrigerer Abstraktionsstufe) einen Plan, der den Operator "implementiert"

Diese Dekompositionen werden in einer Planbibliothek abgelegt und nach Bedarf eingesetzt

# Hierarchische Dekomposition II

**Beispiel:** Bau eines Fachwerkhauses

*Build(House)* ← *ObtainPermit, HireBuilder,*  
*Construction, PayBuilder*

**Aufgaben:**

- Spezifikation *abstrakter Operatoren*
- Spezifikation *primitiver Operatoren*
- Spezifikation von *Dekompositionsmethoden*
- Verwendung bekannter Planungsverfahren (z.B. POP) auf allen Abstraktionsebenen

# Hierarchische Dekomposition III

## Voraussetzung:

- Erweiterung der Plansprache um das Konzept der abstrakten Operatoren
- Erweiterung des Planungsalgorithmus um das Konzept der *Ersetzung abstrakter Operatoren* durch geeignete Methoden

## Dekomposition von *Construction*

### *Decompose(Construction,*

*Plan( Steps :*      $\{ S_1 : Build(foundation), S_2 : Build(Frame),$   
                           $S_3 : Build(Roof), S_4 : Build(Walls),$   
                           $S_5 : Build(Interior) \}$

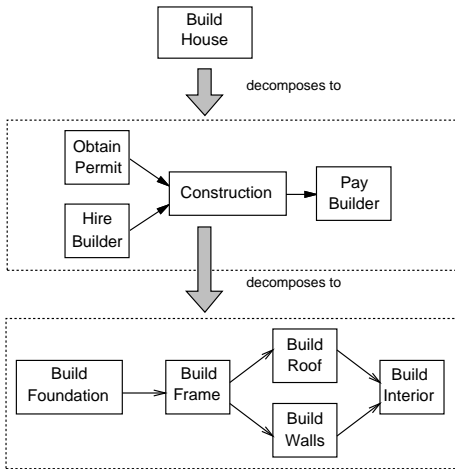
*Orderings :*  $\{ S_1 \prec S_2 \prec S_3 \prec S_5, S_2 \prec S_4 \prec S_5 \},$

*Bindings :*  $\{ \},$

*Links :*      $\{ S_1 \xrightarrow{Foundation} S_2, S_2 \xrightarrow{Frame} S_3, S_2 \xrightarrow{Frame} S_4,$   
                   $S_3 \xrightarrow{Roof} S_5, S_4 \xrightarrow{Walls} S_5 \} \}$

# Hierarchische Dekomposition IV

Hierarchische Dekomposition des Operators *Build(House)* in einen nicht-linearen Plan



# Hierarchische Dekomposition V

## Bemerkungen:

- Eine Dekompositionsmethode ist ein Unterprogramm oder *Makro* für einen Operator
  - ⇒ vgl. modularer Aufbau von *Programmen*
- Für jede Definition  $Decompose(o, p)$  kann ein Operator  $o'$  mit  $o' = \sigma o$ , für eine Substitution  $\sigma$ , durch den Plan  $\sigma p$  ersetzt werden
- Oft können abstrakte Operatoren auf *verschiedene* Weise “implementiert” werden

Falls die verschiedenen Teile eines abstrakten Planes weitgehend *unabhängig* voneinander sind, können sehr komplexe Pläne *modular* und *inkrementell* aus konkreten Teilplänen aufgebaut werden

## Insbesondere:

Möglichkeit der *Wiederverwendung* mehr oder weniger konkreter Lösungen

# Hierarchische Dekomposition VI

Ein Plan  $p$  ist eine *korrekte Implementierung* eines Operators  $o$ , falls

- $p$  konsistent ist, d.h. es gibt keine widersprüchlichen Variablenbedingungen oder Ordnungsbeziehungen in  $p$ ,
- jeder Effekt von  $o$  durch mindestens einen Planschritt in  $p$  zugesichert und durch keinen nachfolgenden Schritt wieder zurückgenommen wird und
- jede Vorbedingung von Schritten in  $p$  durch einen Schritt in  $p$  erzeugt wird oder eine Vorbedingung von  $o$  ist.

# Hierarchische *Task* - Netze I

*Hierarchical Task Networks* (Erol & Hendler & Nau 1994)

*Tasks* entsprechen sowohl *Zielen* als auch *abstrakten Operatoren*

⇒ Dekomposition

⇒ *subgoaling*

⇒ *refinement*

primitive tasks: (STRIPS) - Operatoren

goal tasks: Ziele

compound tasks: komplexe Zielspezifikationen

task networks: Graphen

Knoten: tasks

Kanten: *constraints*: Beziehungen zwischen  
tasks, Variablenbedingungen

# Hierarchische Task - Netze II

**Methoden** *non-primitive tasks + task networks*

## HTN Planen

Ausgehend von einem *initialen* Task-Netz werden mit Hilfe geeigneter Methoden alle Tasks solange verfeinert, bis das Task-Netz nur noch primitive Tasks enthält, so dass alle Constraints erfüllt sind.

# HD-POP I

Erweiterung des POP-Algorithmus um hierarchische Dekomposition

- Neben Planschritten zur Erzeugung nicht unterstützter Vorbedingungen, müssen auch geeignete Methoden zur Dekomposition nicht-primitiver Operatoren gefunden werden  
⇒ Zwei Arten der Verfeinerung
- Pläne als Eingabe

```
function HD-POP(plan, operators, methods) returns plan  
inputs: plan, an abstract plan with start and goal steps (and possibly other steps)  
  
loop do  
  if SOLUTION?(plan) then return plan  
  Sneed, c ← SELECT-SUB-GOAL(plan)  
  CHOOSE-OPERATOR(plan, operators, Sneed, c)  
  Snonprim ← SELECT-NONPRIMITIVE(plan)  
  CHOOSE-DECOMPOSITION(plan, methods, Snonprim)  
  RESOLVE-THREATS(plan)  
end
```

**Lösungen:** Konsistente, vollständige, **primitive** Pläne.

## HD-POP II

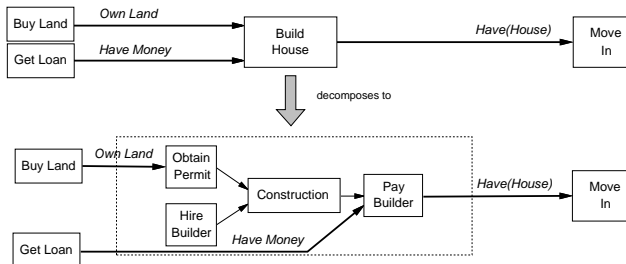
SOLUTION? prüft, ob jeder Planschritt einem primitiven Operator entspricht. Ist der Plan konsistent und vollständig?

CHOOSE-DECOMPOSITION wählt eine Methode aus und wendet sie auf den Plan an. Sei *method* die ausgewählte Methode zur Dekomposition von  $S_{nonprim}$

- *Steps* : Füge alle Schritte von *method* in den Plan ein und entferne  $S_{nonprim}$ .
- *Bindings* : Nimm alle Variablenbindungen aus *method* in den Plan auf. Falls ein Widerspruch entsteht → **fail**.
- *Orderings* : Folge der *least commitment* Strategie, indem jede Ordnungsbeziehung  $S_a \prec S_{nonprim}$  durch Ordnungsbeziehungen, die  $S_a$  vor die letzten Schritte von *method* legen, ersetzt wird. Ersetze entsprechend jede Beziehung  $S_{nonprim} \prec S_z$  durch Beziehungen, die  $S_z$  nach den ersten Schritten von *method* einordnen.

# HD-POP III

- *Links* : Ersetze jede kausale Beziehung  $S_i \xrightarrow{c} S_{nonprim}$  in *plan* durch eine Menge kausaler Beziehungen  $S_i \xrightarrow{c} S_m$ , wobei  $S_m$  ein Planschritt in *method* ist, so daß  $S_m$   $c$  als Vorbedingung hat, aber  $c$  nicht schon Vorbedingung eines Schrittes ist, der vor  $S_m$  liegt. Ersetze entsprechend jede Beziehung  $S_{nonprim} \xrightarrow{c} S_j$  in *plan* durch Beziehungen  $S_m \xrightarrow{c} S_j$ , sofern  $S_m$  ein Schritt aus *method* ist, der  $c$  als Effekt hat und es keinen Schritt nach  $S_m$  gibt, der ebenfalls  $c$  als Effekt hat.



# Hierarchische Dekomposition VII

Hierarchisches Planen erlaubt die Konstruktion sehr großer Pläne ohne den enormen Aufwand, der bei der Konstruktion solcher Pläne aus primitiven Operatoren notwendig wäre.

## **Beispiel:** Bau des Fachwerkhauses

Der aus vier Operatoren bestehende konsistente und vollständige Plan (S. 14) ist eine **abstrakte Lösung** des Problems.

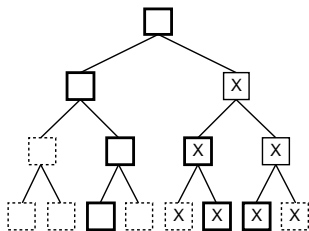
Die Verfeinerung dieses abstrakten Planes führt schrittweise zu einer **konkreten Lösung**, wobei *backtracking* über die Dekompositionsschritte vermieden werden kann.

# Hierarchische Dekomposition VIII

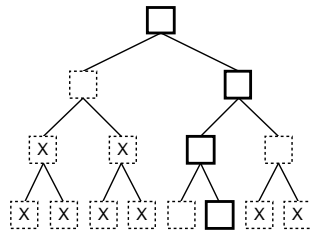
## Wichtige Eigenschaften:

**Downward Solution Property** Ist  $p$  eine abstrakte Lösung, dann gibt es eine primitive Lösung, von der  $p$  eine Abstraktion darstellt.

**Upward Solution Property** Ist ein abstrakter Plan  $p$  inkonsistent, dann gibt es keine primitive Lösung von der  $p$  eine Abstraktion darstellt.



(a) Downward Solution Property



(b) Upward Solution Property

# Hierarchische Dekomposition IX

Unter der Voraussetzung, daß die *Up-* und *Downward Solution* Eigenschaften gelten, reduziert sich der Planungsaufwand gegenüber nicht hierarchischen Verfahren beträchtlich.

Ein vereinfachtes Modell des Suchraumes:  
Annahmen

- Es gibt mindestens eine primitive Lösung mit  $n$  Schritten.
- $b$  sei die Anzahl sowohl der möglichen Dekompositionen für jeden abstrakten Schritt als auch der möglichen Operatoren zur Erzeugung einer Vorbedingung für einen primitiven Schritt.
- $s$  sei die Anzahl der Schritte in einer Methode und  $d$  die Lösungstiefe.

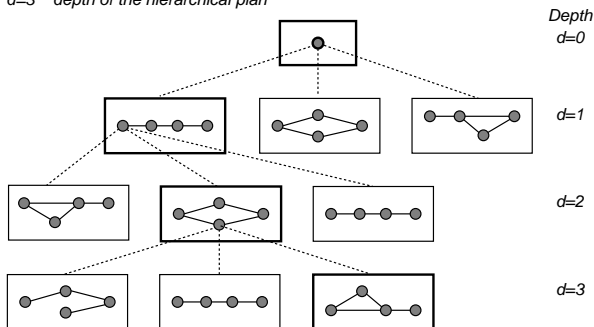
Nicht-hierarchisches Planen: Aufwand  $O(b^n)$

Hierarchisches Planen: Aufwand  $O(bs^d)$

# Hierarchische Dekomposition X

Die hierarchische Planungsstrategie sucht nur nach solchen Dekompositionen, die abstrakte Lösungen erzeugen.

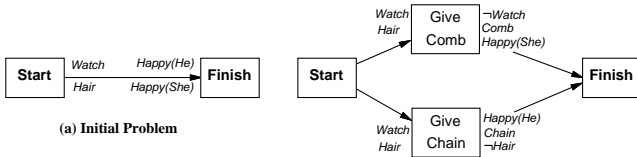
$b=3$  branching factor: number of decomposition methods per step  
 $s=4$  steps in a decomposition method  
 $d=3$  depth of the hierarchical plan



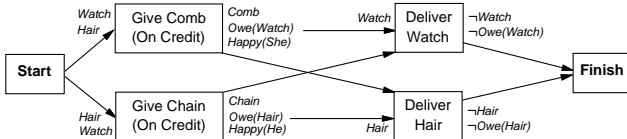
Ein nicht-hierarchischer Planer inspiziert  $3 \times 10^{30}$  Pläne, ein hierarchischer Planer 576 !

# Hierarchische Dekomposition XI

*The Gift of the Magi* (O. Henry)



(b) Abstract Inconsistent Plan



(c) Decomposition of (b) into a Consistent Solution

Hier gilt die Upward Solution Property nicht!

# Hierarchische Dekomposition XII

Die **Unique Main Subaction Condition** garantiert die *Upward Solution* Eigenschaft.

**UMS** Die Dekomposition enthält einen Planschritt, der alle Vorbedingungen und Effekte des abstrakten Operators umfaßt.

## Wichtig:

- Im allgemeinen können auch inkonsistente abstrakte Pläne zu konsistenten primitiven Plänen verfeinert werden, d.h. zu Lösungen führen.
- Im allgemeinen hat nicht jede abstrakte Lösung eine konkrete Entsprechung.

## Heuristik:

- Erzeuge **konsistente** Verfeinerungen.
- Verfeinere abstrakte **Lösungen**.

# Sharing und Merging I

CHOOSE-DECOMPOSITION fügt jeden Schritt der Dekompositionsmethode in den aktuellen Plan ein (*Merging*)

⇒ Divide-and-Conquer

Häufig erfordert die Lösung eines Problems jedoch, daß **ein** Planschritt **unterschiedlichen** Teilplänen angehört (*Sharing*).

## Beispiel:

*In die Flitterwochen fahren und Mutter sein*

⇒ *Merging* liefert keine Lösung, falls Scheidung nicht erlaubt ist.

**Sharing** Erweiterung von CHOOSE-DECOMPOSITION: Zur Instantiierung der Operatoren in der Dekomposition werden entweder neue Schritte in den Plan eingefügt oder aber bereits vorhandene verwendet.

# Sharing und Merging II

**Merging** Eine Dekomposition wird in den aktuellen Plan eingefügt. Anschließend erfolgt eine sogenannte **Plankritik**, die den entstandenen Plan modifiziert.

**Plankritiken** lösen Konflikte und nehmen Optimierungen vor.

## Fazit:

- *Plankritiken* oder *Sharing* sind unerlässlich. Ohne sie ist die Vollständigkeit hierarchischer Planungsverfahren nicht gewährleistet.
- *Plankritiken* haben gegenüber *Sharing* den Vorteil, daß sie eine klare Strukturierung des Suchraumes erlauben.

# Approximationshierarchien I

Unter hierarchischem Planen versteht man auch Ansätze, die mit **Approximationsabstraktion** arbeiten.

Abstrakte Operatoren entstehen dabei aus konkreten, indem (“unwichtige”) Vorbedingungen und Effekte eliminiert werden.

⇒ **Situationsabstraktion**

## Bemerkung:

Die “abstrakten” Versionen der Operatoren sind keine Abstraktionen im eigentlichen Sinne: Es sind Approximationen konkreter Operatoren. In entsprechenden Planungsansätzen werden die Vorbedingungen nach ihrer **Kritikalität** geordnet:

*Op(Action :*    *Buy(x),*  
                  *Effect :*    *Have(x) ∧ ¬Have(Money),*  
                  *Precond :* 3 : *Sells(store, x) ∧*  
                                  2 : *Have(Money) ∧*  
                                  1 : *At(store))*

# Approximationshierarchien II

Approximationshierarchien haben die *Unique Main Subaction* Eigenschaft: Alle Dekompositionen haben nur einen Planschritt und besitzen alle Vorbedingungen und Effekte des abstrakten Operators

⇒ *Upward Solution Property*

## **Nachteil:**

Die Kritikalität der Vorbedingungen ist fest vorgegeben und kann nicht an den jeweiligen Planungskontext angepaßt werden

## **Beispiel:**

ABSTRIPS (1974) ist ein hierarchisches Planungssystem auf der Basis von STRIPS, das mit Approximationshierarchien arbeitet

*The Open Planning Architecture* (Tate & Drabble et al. 1985-95)

## Nichtlineares, hierarchisches Planen

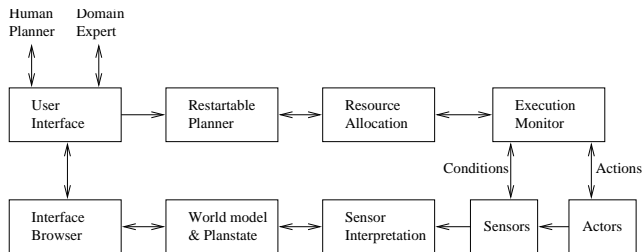
- Planspezifikation
- Plangenerierung
- Planausführung und Überwachung
- Benutzerinteraktion

⇒ Modularität

⇒ Erweiterbarkeit

(Interaktive) Suchraumkontrolle und Konfliktmanagement.

## Module und Datenfluss:

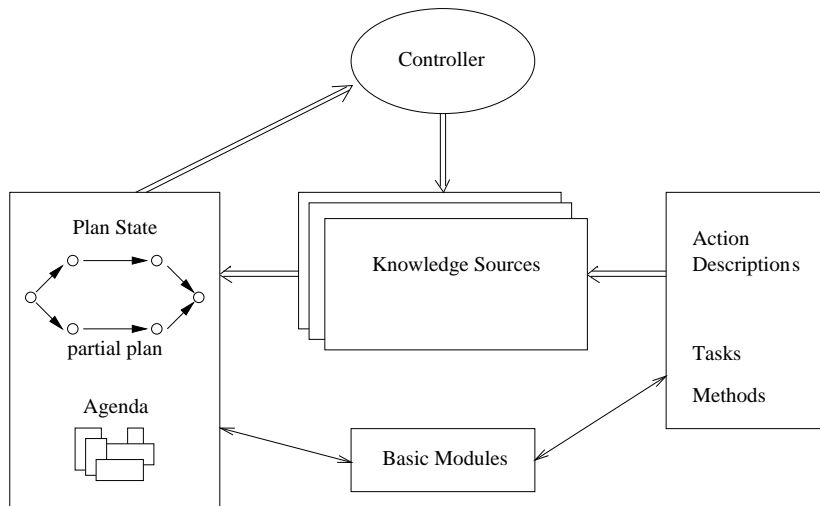


## Techniken:

- hierarchisches, nicht-lineares Planen, flexible Dekomposition von tasks
- Agenda basierte Organisation, Blackboard Architektur
- Least Commitment Strategie bzgl. der Variablenbindungen
- Behandlung von Zeit- und Ressourcen-Constraints

# O-Plan III

## Kontrolle:

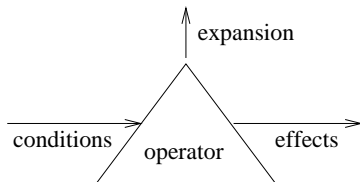


# O-Plan IV

- Agenda:
  - ▶ zu expandierende tasks
  - ▶ zu etablierende Vorbedingungen
  - ▶ Kontrollinformation (Reihenfolge von Schritten, Prioritäten)
- Partial Plan:
  - ▶ festgelegte Planschritte
  - ▶ Ordnungsbeziehungen
  - ▶ kausale Beziehungen
- Knowledge Sources:
  - ▶ Dekomposition von tasks
  - ▶ Etablierung kausaler Beziehungen
  - ▶ Analyse von Effekten
  - ▶ Etablierung von Variablenbedingungen
  - ▶ Ordnen (von Teilen) der Agenda
  - ▶ Benutzerinteraktion
- Controller:
  - ▶ Abarbeitung der Agenda

# O-Plan V

## Operatoren:



## Beispiel:

**schema** puton;

*vars* ?x = undef, ?y = undef, ?z = undef

*expands* { put ?x ontop of ?y }

*only\_use\_for\_effects* on(?x,?y) = true

clear(?y) = false

on(?x,?z) = false

clear(?z) = true

*conditions* only\_use\_for\_query on(?x,?z) = true

achievable clear(?y) = true

achievable clear(?x) = true

end schema.