# Einführung in das Planen

zu lösende Aufgaben:

- Finde Folge von Aktionen,
- um aus einer gegebenen Anfangssituation
- eine Zielsituation zu erreichen,

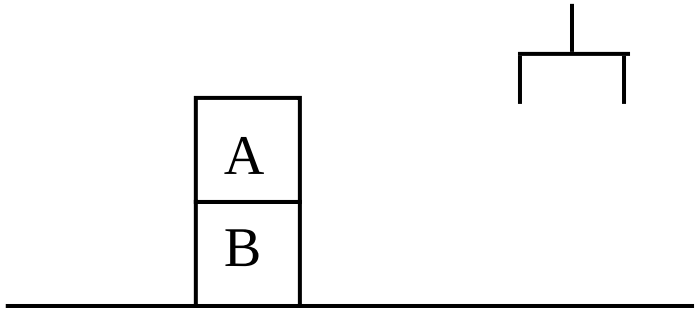
  wobei die Aktionen aus einer vorgegebenen Menge sind

Z.B. Handlungsplan für einen Roboter auf einer abstrakten Ebene.

# Planen in der Klötzchen-Welt (Blocks World)

Eine Modellwelt,
zur Untersuchung des Vorgehens
der Planungsalgorithmen und
der entstehenden Problematiken

**Blockwelt**, oder auch Klötzchenwelt.



# Planen in der Klötzchen-Welt

Annahmen:

- Auf einem Tisch stehen
  Würfel-Klötzchen auf- bzw. nebeneinander.
  Für alle Würfel ist genug Platz vorhanden.
- Es gibt einen Greifarm (Hand), der jeden (freien) Würfel
  hochheben und woanders hinstellen kann,
  d.h. entweder auf den Tisch, oder auf einen anderen Würfel.
- Ein Würfel hat drei Möglichkeiten:
  - auf dem Tisch, oder
  - auf einem anderen Würfel, oder
  - er wird vom Greifarm gehalten.
- typische Operationen sind: HochHeben($x$), Absetzen($x$),

# Situationslogik

Prädikatenlogik zum Behandeln veränderlicher Zustände

Prädikate werden um Situationsparameter erweitert
und dadurch veränderlich
(analog zu diskreter Zeit)

Inferenzen der Prädikatenlogik reichen aus.

Beispiel `StrasseNass(RobertMayerStr, S)`

# Situationsparameter in der Situationslogik

Situationsparameter sind strukturiert:

| | | |
|---|---|---|
| Situationsparameter: | Konstanten $S_0$ | |
| | Terme | Aktion(Situation, Parameter) |
| | Aktionen | Situation, Parameter $\rightarrow$ Situation |

$S_1 = Hochheben(S_0, A)$: Situation, nachdem in $S_0$ etwas hochgehoben wurde.

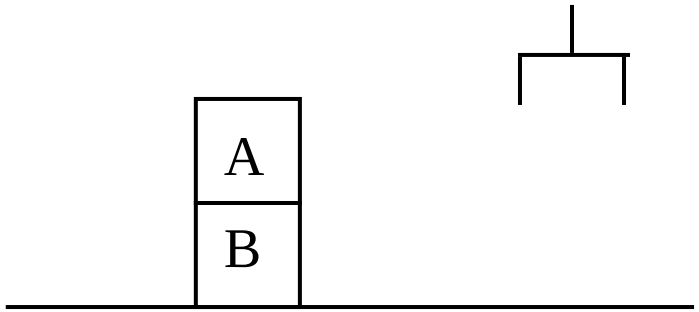
Diese Strukturierung erlaubt eine sehr allgemeine Formulierung.

# Klötzchenwelt: Beispiel-Situationen

Darstellung einer Situation der Klötzchenwelt
und einiger Eigenschaften der aktuellen Situation:

$$
\begin{array}{ll}
 & \mathsf{AufDemTisch}(B, S_0) \\
\wedge & \mathsf{Auf}(A, B, S_0) \\
\wedge & \mathsf{Hand}(\mathsf{Leer}, S_0) \\
\wedge & \mathsf{Frei}(A, S_0)
\end{array}
$$

Hierbei bezeichnen $A, B$ Namen von Klötzchen,
$S_0$ ist die aktuelle Situation.



# Operatoren der Klötzchenwelt

Operatoren, um die Klötzchenwelt zu manipulieren:
Zum Erzeugen neuer Zustände (Situationen)

**VomStapel** , **Aufheben(.)** , **Hinstellen(.)**

# Operator VomStapel

**VomStapel** nimmt einen Klotz vom Stapel

In der Situationslogik muss man dazu
eine Formel (ein Axiom) hinschreiben,
die den Effekt dieser Operation beschreibt:

# Operator VomStapel

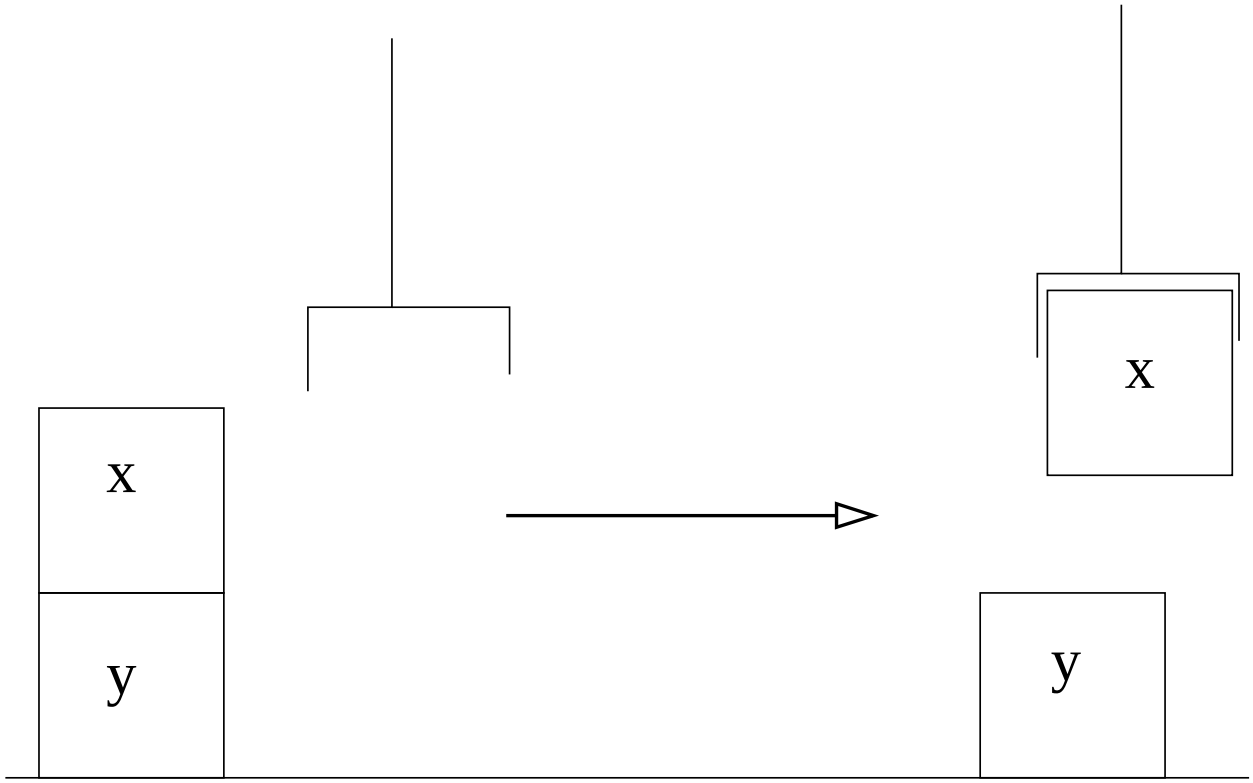
$\forall x, y, s : \mathsf{Hand}(\mathsf{Leer}, s) \wedge \mathsf{Frei}(x, s) \wedge \mathsf{Auf}(x, y, s)$

$\Rightarrow$

$$
\begin{aligned}
& \mathsf{Hand}(x, \mathsf{VomStapel}(x, y, s)) \\
\wedge\ & \mathsf{Frei}(y, \mathsf{VomStapel}(x, y, s)) \\
\wedge\ & \neg\mathsf{Hand}(\mathsf{Leer}, \mathsf{VomStapel}(x, y, s)) \\
\wedge\ & \neg\mathsf{Frei}(x, \mathsf{VomStapel}(x, y, s)) \\
\wedge\ & \neg\mathsf{Auf}((x, y, \mathsf{VomStapel}(x, y, s)))
\end{aligned}
$$

Hier ist $s$ die Situation vor Ausführen der Operation und $\mathsf{VomStapel}(x, y, s)$ die Situation nach Ausführen der Operation.

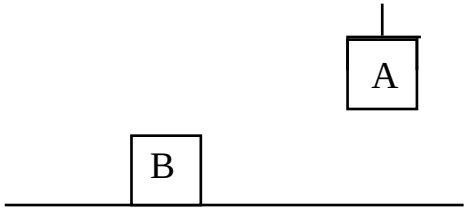
# Operator VomStapel



# Operatoren der Klötzchenwelt: Axiomatisierung

Wenn $S_0$ vorher und $S_1 :=$ VomStapel$(A, B, S_0)$,
dann erhalten wir als neue Fakten in $S_1$:

Hand$(A, S_1)$
Frei$(B, S_1)$
$\neg$Hand(Leer, $S_1$)
$\neg$Frei$(A, S_1)$
$\neg$Auf$(A, B, S_1)$



# Operatoren: Axiomatisierung

Gilt AufDemTisch$(B, S_1)$?

Leider aus den bisherigen Axiomen nicht folgerbar!

Man bräuchte **Rahmenaxiome,successor-state axioms**,
die pro Aktion exakt sagen, was sich alles **nicht** ändert:

$$\forall x, y, z, s : \text{AufDemTisch}(z, s) \wedge z \neq x$$
$$\Rightarrow$$
$$\text{AufDemTisch}(z, \text{VomStapel}(x, y, s))$$

# Das Rahmenproblem (Frame problem)

Wie beschreibt und behandelt man die Aspekte der Welt, die sich unter Operatoren nicht ändern?

Feinere Unterscheidung

- Rahmenproblem der Repräsentation (Sprache).
- Rahmenproblem der Inferenz.

Situationslogik: hat beide Varianten des Rahmenproblems: Bei der Formulierung der gültigen Fakten einer Situation und bei der Inferenz

In jedem Planungsformalismus muss das Rahmenproblem beachtet und gelöst werden.

# Rahmenproblem

Die Formulierung in Prädikatenlogik ist praktisch zu ineffizient:

Sehr viele Rahmenaxiome

sehr viele (scheinbar redundante) Fakten, die erhalten bleiben.

Versuch einer Abhilfe:

- automatische Generierung
- automatisches Verwenden in (Rahmenaxioms-) Inferenzen

Nachteile:

- für jede Aussage und Aktion ist nachzurechnen, ob sich durch die Aktion etwas geändert hat.
- unübersichtlich und unsicher
- die Axiome der Modellwelt sind schwer auf Korrektheit und Widerspruchsfreiheit zu prüfen:

# Qualifikationsproblem

Welche Eigenschaften muss man im Modell explizit beschreiben ?
Welche Eigenschaften lässt man weg?

Dieses Problem hat jede Modellierung,
da nicht alle Aspekte der realen Welt erfassbar sind

Im Modell ungenau oder wegabstrahiert sind:

Größe der Blöcke
Gewicht der Blöcke
Kann die Hand etwas verlieren ?
Stapel fällt bei Sturm um,
der Greifarm greift manchmal ungenau zu,
. . .

D.h. Man kann die Wirklichkeit nicht exakt im Modell erfassen,
Jedes Modell abstrahiert bestimmte Eigenschaften.

# Problematiken der Modellwelten

- Modelle abstrahieren immer von der realen Welt.

- „explizite Programmierung“.
  Muss alles im Voraus vollständig beschreiben und programmieren
  Verbesserung evtl. mit Methoden des maschinellen Lernens

- Logische Modelle gehen von einer Übereinstimmung der Welt mit dem Modell aus.
  Aber: die interne Repräsentation stimmt nicht ganz mit der Welt überein
  Erfordert: Kontrolle mittels Sensoren

# STRIPS-Planen

**St**anford **R**esearch **I**nstitute **P**roblem **S**olver

**Situationen** werden beschrieben durch:
Konjunktion von (positiven und negativen) Grund-Fakten.
(keine Variablen, keine Funktionssymbole)

Namen (Konstanten) für die Objekte.
Es gilt die unique-names assumption:
Verschiedene Namen sind verschiedene Objekte.

# STRIPS-Planen

Alternative zu negativen Fakten:

Closed-World Annahme:

- nur positive Fakten und werden gespeichert
- nicht vorhandene Fakten sind falsch

# STRIPS-Planen:Operatoren

Ein **Operator** hat folgende Komponenten:

- Menge von Variablen (allquantifiziert)
- Vorbedingungsteil (Konjunktion von Literalen, positiv oder negativ)
- Effekte des Operators: besteht aus positiven und negativen Fakten, die die entsprechenden Fakten in der Situation überschreiben sollen. Man kann das auch positiv als Dazu- und Löschliste gruppieren,

Eine konkrete Anwendung eines Operators muss vollinstanziiert sein, d.h. alle Variablen sind durch Konstanten substituiert.

# STRIPS-Planen:Plan

Ein linearisierter **Plan** ist eine Folge von vollinstanziierten Operatoren.

Anwendung eines vollinstanziierten Operators auf $S$:
Vorbedingungen des Operators müssen erfüllt ein
Ergebnis: Die Nachfolgesituation nach Abarbeitung der Dazu- und Löschlisten.

Die Anwendung eines Plans $P = O_1, \ldots, O_n$ auf $S_1$ ergibt
Folge von Situationen $S_1, S_2, \ldots, S_{n+1}$ mit
$O_i(S_i) = S_{i+1}$ für alle $i$. Resultat: $S_{n+1}$
Bezeichnung: $P(S_1) = S_{n+1}$

# STRIPS-Planen: die Aufgabe

| | |
|---|---|
| Allgemein gegeben: | Menge von Operatoren<br>Mögliche Fakten (Prädikate) |
| Gegeben: | eine Startsituation $S_0$ und<br>eine Zielsituation $Z$ |
| Gesucht: | ein vollinstanziierter, linearisierter Plan $P$<br>mit $P(S_0) = Z$ |

Vereinfachung im Folgenden: nur positive Fakten.

# STRIPS-Planen: Operatoren: Beispiele

in der Klötzchenwelt ( blocks world)

# Operatoren:

**Aufheben**(x):

| | | |
|---|---|---|
| Vorbedingung: | | $AufDemTisch(x)$ |
| | | $Frei(x)$ |
| | | $Hand(Leer)$ |
| Effekt: | Dazu: | $Hand(x)$ |
| | Löschen: | $AufDemTisch(x)$ |
| | | $Frei(x)$ |
| | | $Hand(Leer)$ |

**Hinstellen(x)**:

| | | |
|---|---|---|
| Vorbedingung: | | $Hand(x)$ |
| Effekt: | Dazu: | $AufDemTisch(x)$ |
| | | $Frei(x)$ |
| | | $Hand(Leer)$ |
| | Löschen: | $Hand(x)$ |

# Operatoren:

**StapeIn(x,y)**:

| | | |
|---|---|---|
| Vorbedingung: | | $Hand(x)$ |
| | | $Frei(y)$ |
| Effekt: | Dazu: | $Auf(x,y)$ |
| | | $Frei(x)$ |
| | | $Hand(Leer)$ |
| | Löschen: | $Hand(x)$ |
| | | $Frei(y)$ |

**VomStapel(x,y)**:

| | | |
|---|---|---|
| Vorbedingung: | | $Auf(x,y)$ |
| | | $Frei(x)$ |
| | | $Hand(Leer)$ |
| Effekt: | Dazu: | $Hand(x)$ |
| | | $Frei(y)$ |
| | Löschen: | $Auf(x,y)$ |
| | | $Frei(x)$ |
| | | $Hand(Leer)$ |

# Implizite Annahmen

Meist sollen die Situationen implizite Annahmen erfüllen:

In der Klötzchenwelt:

- eine Situation ist eine Menge von positiven Grund-Fakten
  nicht erwähnte Grund-Fakten sind falsch.
- jedes Objekt existiert nur einmal
- Jeder Platz kann nur von einem Objekt besetzt sein
- Die Situation muss eine konsistente Beschreibung sein:
  z.B. $Auf(B, A), Auf(B, A)$ kann nicht gleichzeitig gelten.
- Es gibt nur die erwähnten Objekte

# Planungsalgorithmus: Regression

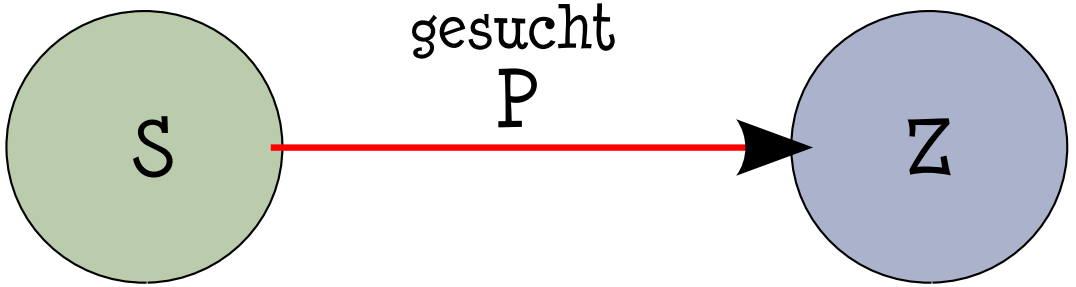
- Finde voll instanziierten Plan $P$ zu
- Anfang $S$, Ziel $Z$
- $S, Z$ sind Mengen von positiven Grundfakten
- $S \quad \stackrel{P}{\longrightarrow} \quad S \quad \supseteq Z$

# Planungsalgorithmus: Regression

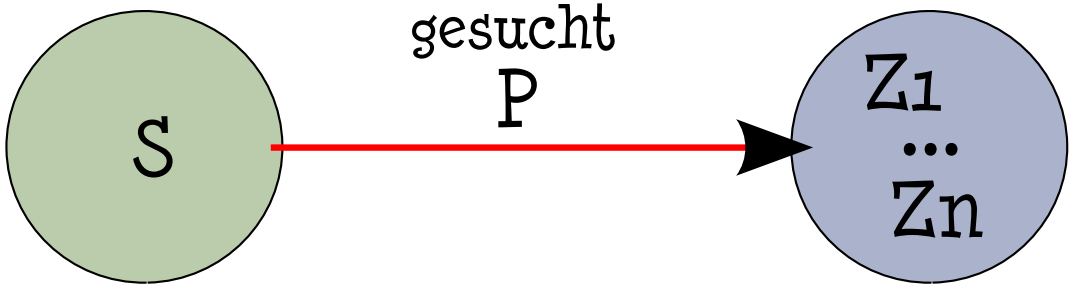
$Plane$(Z, S) (Z: Teilziele, S: Situation)

- P = []
- Solange es **offene** Teilziele gibt in Z
  - wähle **offenes** Ziel $Z_1$ aus Z
  - wähle Instanz O eines Operators, so dass $Z_1$ in der Dazu-Liste ist. und kein Effekt von O einem Teilziel aus Z widerspricht.
  - Erzeuge Plan P' zum Erfüllen der Vorbedingung von O durch rekursiven Aufruf:
    P' := $Plane$(O.Vorbedingung, S).
  - S := O(P'(S))
  - P := P++P'++[O]
- RETURN P

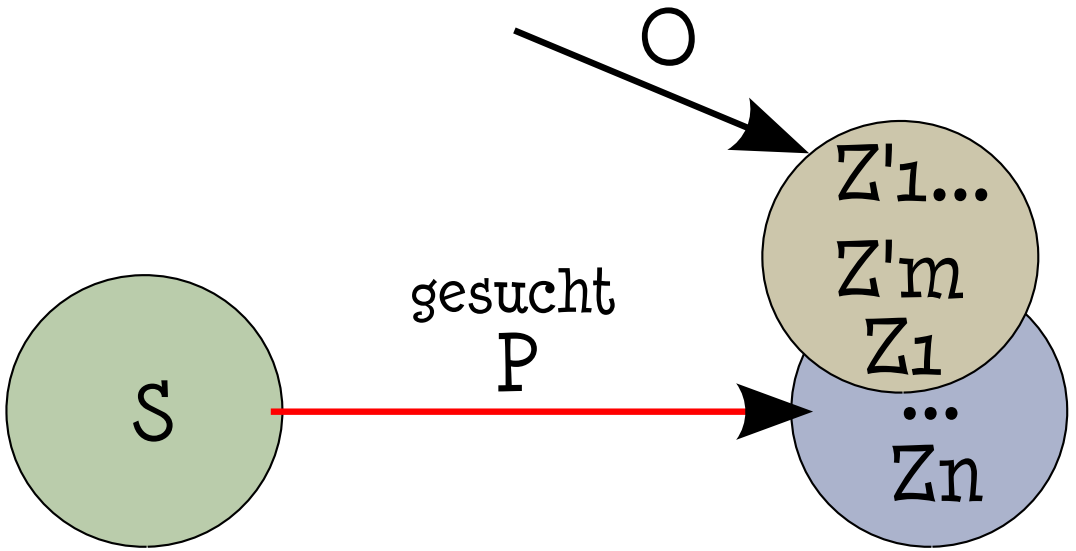
# Illustration



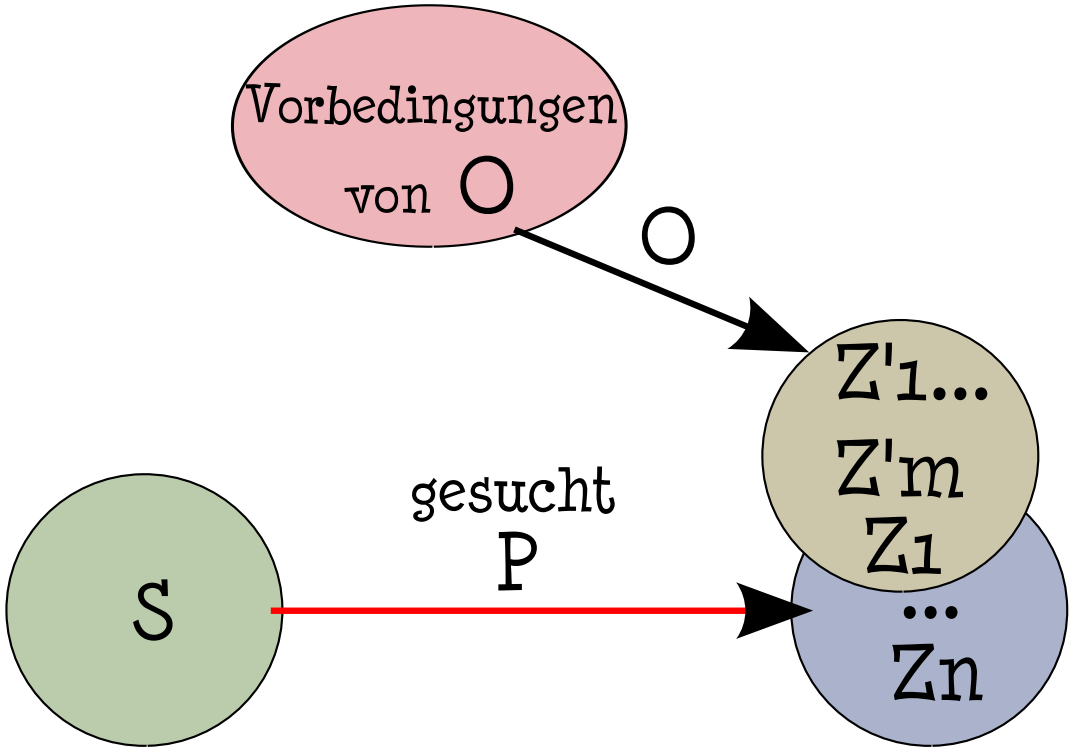
# Illustration



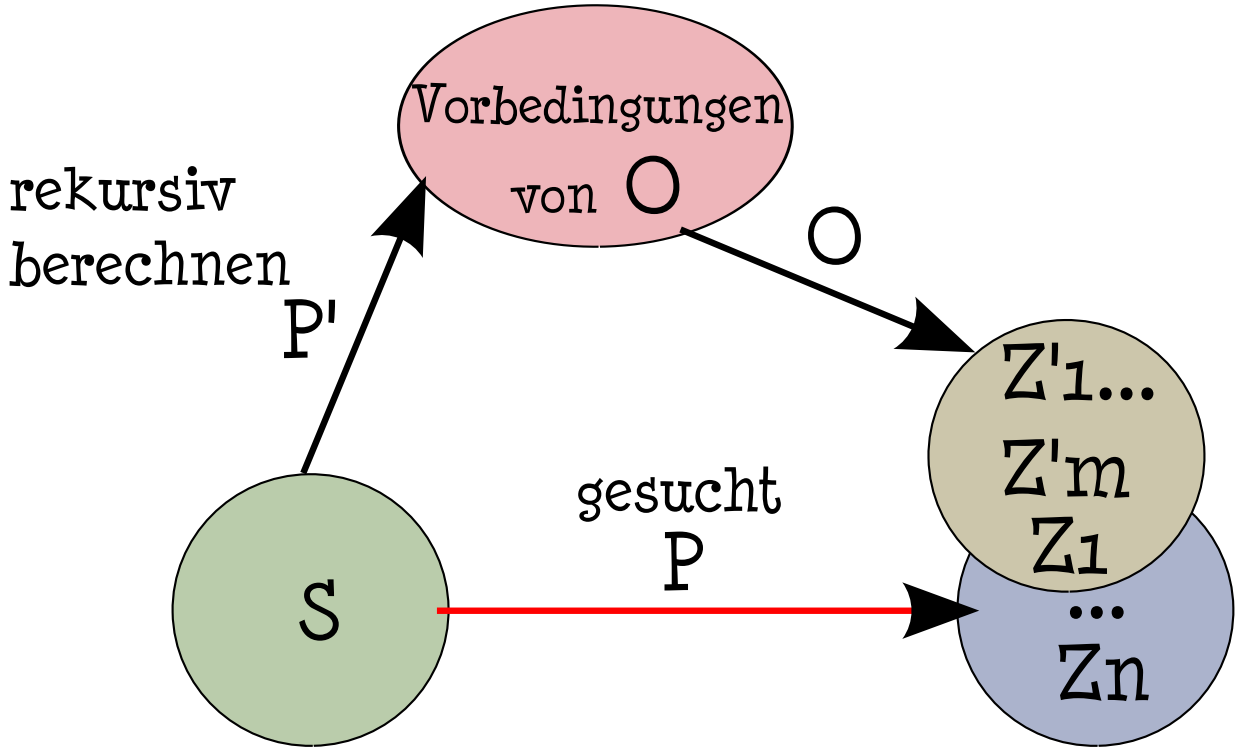
# Illustration



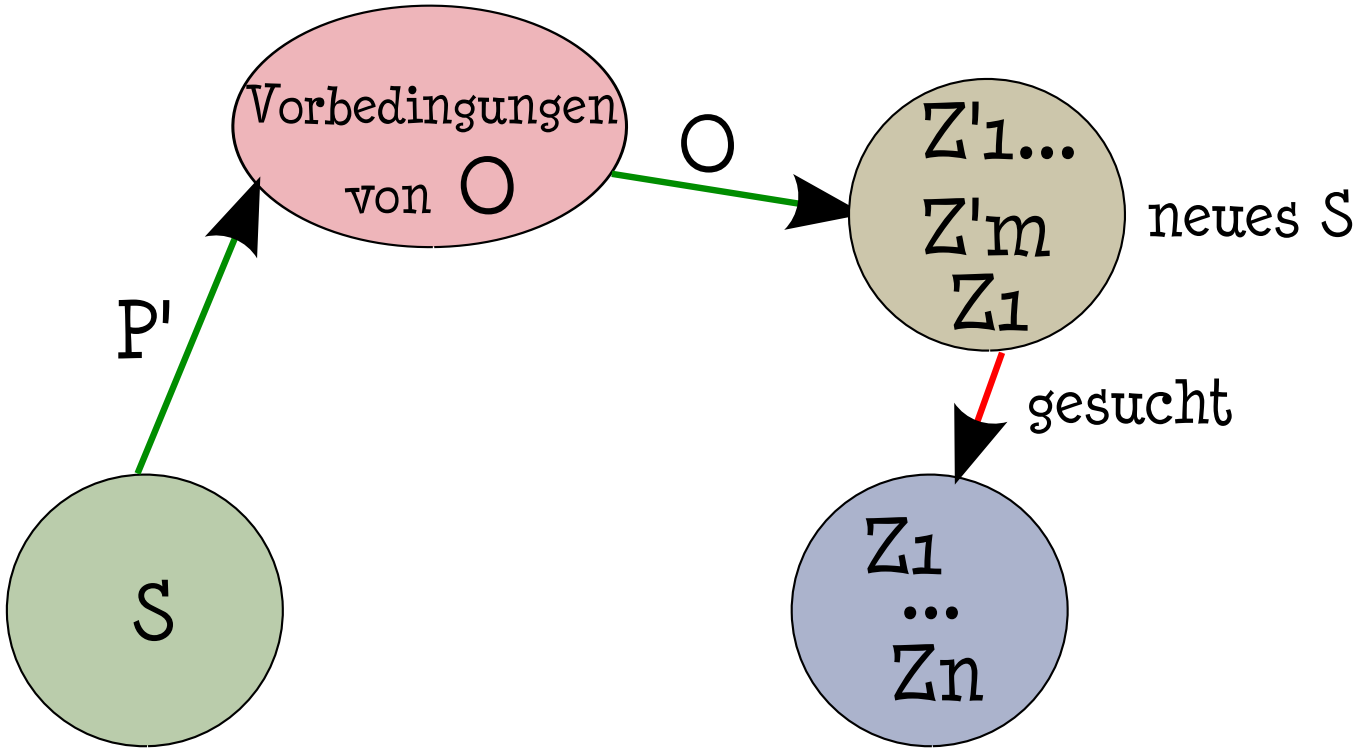
# Illustration



# Illustration



# Illustration



# Beispiel für einen Planungsablauf


Anfangssituation:

$Auf(B,A), Frei(B), Hand(Frei), Tisch(C), Tisch(A), Frei(C)$

Ziel: $Auf(A,B), Auf(B,C)$

Teilziel: $Auf(B,C)$

Operatorinstanz: $Stapeln(B,C)$

Vorbedingung: Hand(B), Frei(C)

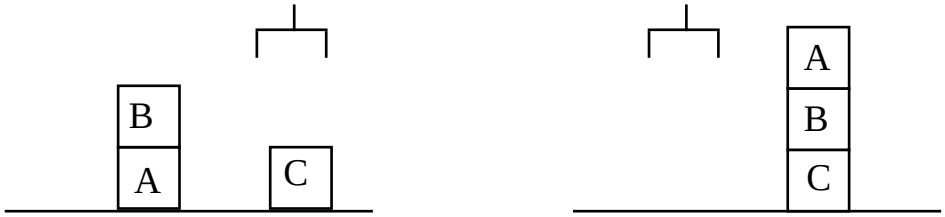
rekursives Planen: Hand(B), Frei(C)

Erfülle: Hand(B)

Operatoren: $VomStapel(B,x)$ oder $Aufheben(B)$

usw.

# Beispiel ...




Resultat-Plan:
VomStapel(B,A), Stapeln(B,C), Aufheben(A), Stapeln(A,B).

# STRIPS: Bemerkungen

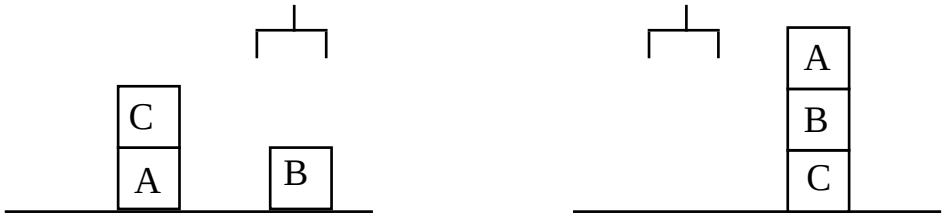
Implizite Annahmen und Einschränkungen dieser Methode:

- Linearitätsannahme:
  Einem Plan liegt eine totale (lineare) zeitliche Ordnung zugrunde. Der Planungsalgorithmus benutzt diese.

- Lösung des Rahmenproblems in STRIPS:
  Jeder Operator ändert genau das, was als sein Effekt angegeben ist. Alle nicht angegebenen Fakten bleiben erhalten.

- In der einfachen angegeben Prozedur oben:
  das Ziel wird direkt angesteuert, das Erfinden von neuen Zwischenzielen ist nicht möglich.

# Das Problem der Zwischenzielinteraktion (Sussman-Anomalie)




Nichtoptimale Pläne durch das Vorgehen des Planungsalgorithmus: Regression

rekursive Planung durch Erfüllung von Zwischenzielen

Sussman-Anomalie ist unabhängig von der Selektion der Zwischenziele

Grund ist die Vorgehensweise der Methode:
1. zuerst ein Teilziel vollständig erfüllen,
2. dann nächstes Teilziel erfüllen.

# Sussman-Anomalie - Beispiel

Die beste Variante ist:
VomStapel(C,A), Hinstellen(C), Hochheben(A),
Stapeln(A,B), VomStapel(A,B) Hinstellen(A),
Hochheben(B), Stapeln(B,C), Hochheben(A),
Stapeln(A,B).

Abhilfen:
Andere Suchstrategie, oder
„Gucklochoptimierung “(wie in einem Compiler) Das Planungsproblem hat Ähnlichkeit mit dem Erstellen imperativer Programme

# Andere Verfahren

## Suche mit Iterativem Vertiefen

Bem. : durch die Quasi-Breitensuche wird die Sussman-Anomalie verhindert.
ineffizient, da sehr viele Möglichkeiten probiert werden.

problemspezifische Konsistenzbedinungen des Gesamt-Zwischenziels
verhindert einige Ineffizienzen

## Voller Aufbau des Raumes aller Situation

## Codieren und danach Benutzung eines SAT-Beweisers

# Nichtlineare Pläne

Planen; partielle geordnete Pläne.

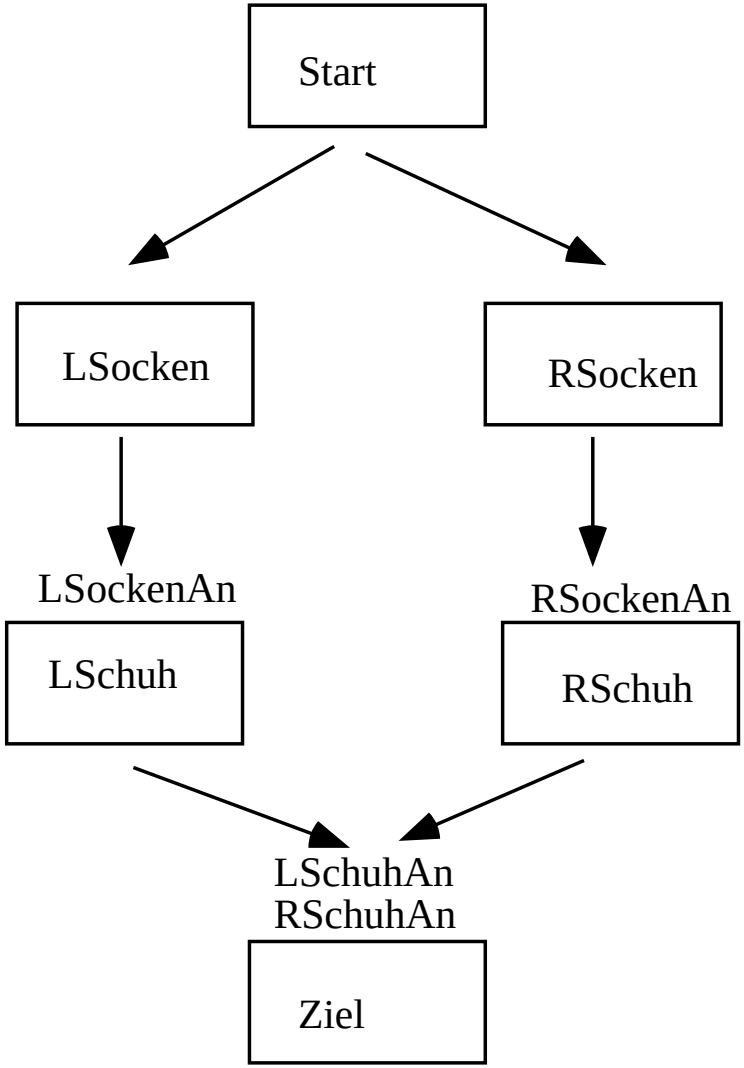
## Beispiel: Socken und Schuhe

Ziehe linke und rechte Socke und linken und rechten Schuh an.

Die Operatoren sind:

| Operator | Bedingung | Effekt |
|---|---|---|
| Start | | |
| LSockenAnziehen | | LSockenAn |
| RSockenAnziehen | | RSockenAn |
| LSchuhAnziehen | LSockenAn | LSchuhAn |
| RSchuhAnziehen | RSockenAn | RSchuhAn |
| Ziel | LSchuhAn, RSchuhAn | |

# Beispiel: Schuhe; der Plan



# Planen mit partieller Ordnung: POP

Wie vorher, aber: Verallgemeinerung der Pläne:
statt Linearisierung (bzw. Folge)
eine partielle Ordnung der instanziierten Operatoren

Vorteile:

- Der Planalgorithmus muss keine genaue Ordnung festlegen.
  Kann Backtracking beim Planen optimieren
- Der fertige Plan erlaubt verschiedene Linearisierungen.
  hat höhere Flexibilität.
  ermöglicht parallele Ausführung
  z.B mehrere Greifarme

# POP-Plan

## Definition

Ein Plan ist:

- Ein gerichteter Graph mit Knoten $K_i$.
- Startoperation ist die Wurzel, die Zieloperation die Senke
- Jeder Knoten ist mit genau einem Operator markiert,
  die Variablen sind jeweils neu umbenannt.
- Instanzbedingungen an die Variablen:
  nur $x = a$ bzw. $x = y$ sind möglich

Wir schreiben $K_i \prec K_j$, wenn es einen Weg von $K_i$ nach $K_j$ gibt.

# POP-Plan

Ein Plan ist *konsistent*, wenn der Graph azyklisch ist
und die Instanzbedingungen nicht widersprüchlich sind;
d.h. $x = a, x = b$ ist nicht daraus herleitbar.

Eine *Linearisierung eines Planes* ist eine Folge aller Knoten, die kompatibel mit dem gerichteten Graphen ist.
Normalerweise meint man dann nur die Folge der Operatoren.
(mittels topologischem Sort herstellbar)

Ein konsistenter Plan $P$ ist *vollständig*,
wenn jede Linearisierung auf jeder Situation
ohne Fehlschlagen ausführbar ist.

# POP-Plan

Ein Kriterium für Vollständigkeit ist:

Für jede Vorbedingung $c$ eines Operators am Knoten $K_i$ gibt es einen Knoten $K_j$, so dass $K_j \prec K_i$ gilt und $K_j$ hat $c$ als Effekt; und für alle weiteren Knoten $K_k$, die in einer Linearisierung zwischen $K_j$ und $K_i$ sein könnten, d.h. für die weder $K_k \prec K_j$ noch $K_i \prec K_k$ gilt, ist $\neg c$ nicht Effekt von $K_k$.

$\Rightarrow$ Jede Vorbedingung wird erfüllt ohne wieder zerstört zu werden. die parallele und widerspruchsfreie Ausführbarkeit des Plans ist garantiert.

# Partial-Order Planen

Operationen auf dem Graph bzw der partiellen Ordnung:

- Füge einen neuen Knoten (instanziierten Operator) hinzu
- Füge neue Kanten hinzu

bis der Plan konsistent und vollständig ist

Steuerungsmöglichkeit und Strategien sind komplexer
als beim linearen Planen.

# Beispiel: Milch+Bananen

Aufgabe: kaufe Milch, Bananen und einen Bohrer
Milch und Bananen gibt es im Supermarkt,
den Bohrer im Heimwerkerladen.

entwickle einen Plan,
so dass man von zu hause startet,
die beiden Geschäfte aufsucht,
die Artikel jeweils kauft
und wieder nach hause geht.

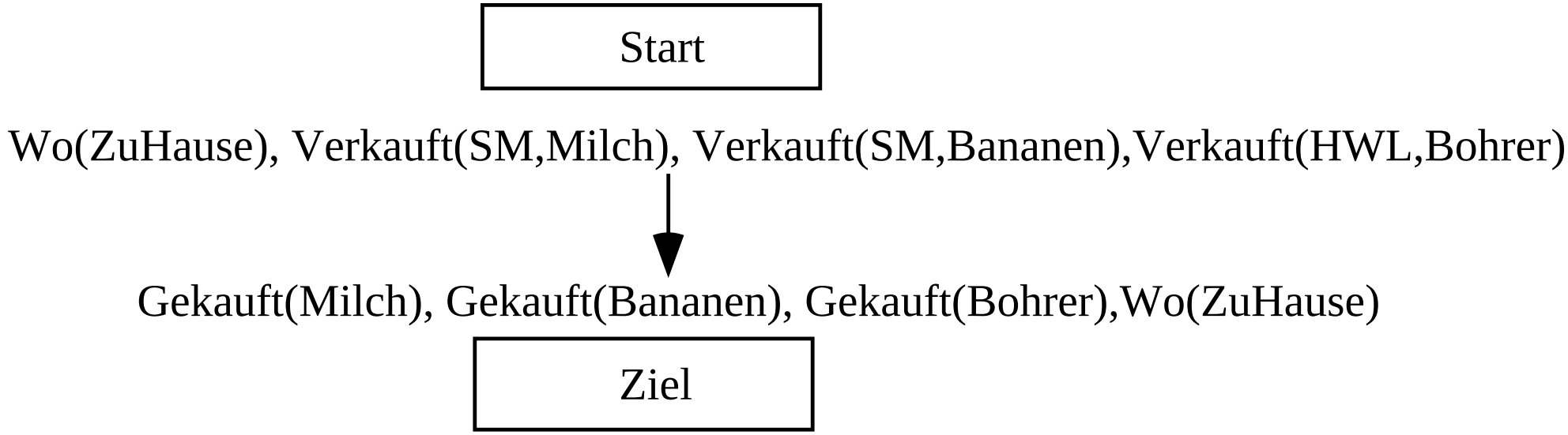
# Beispiel: Milch+Bananen

| | |
|---|---|
| Start | Effekte: Wo(ZuHause), Verkauft(SM,Milch), Verkauft(SM,Bananen), Verkauft(HWL,Bohrer) |
| Ziel | Bedingung: Gekauft(Milch), Gekauft(Bananen), Gekauft(Bohrer) , Wo(ZuHause) |
| Op: Gehe($dort$) | Bedingung: Wo($hier$), $hier \neq dort$;<br>Effekte: $\neg$Wo($hier$), Wo($dort$)<br>Variablen sind $hier, dort$ |
| Op: Kaufe($artikel$) | Bedingung: Wo($laden$), Verkauft($laden, artikel$)<br>Effekte: Gekauft($artikel$)<br>Variablen sind $laden, artikel$. |

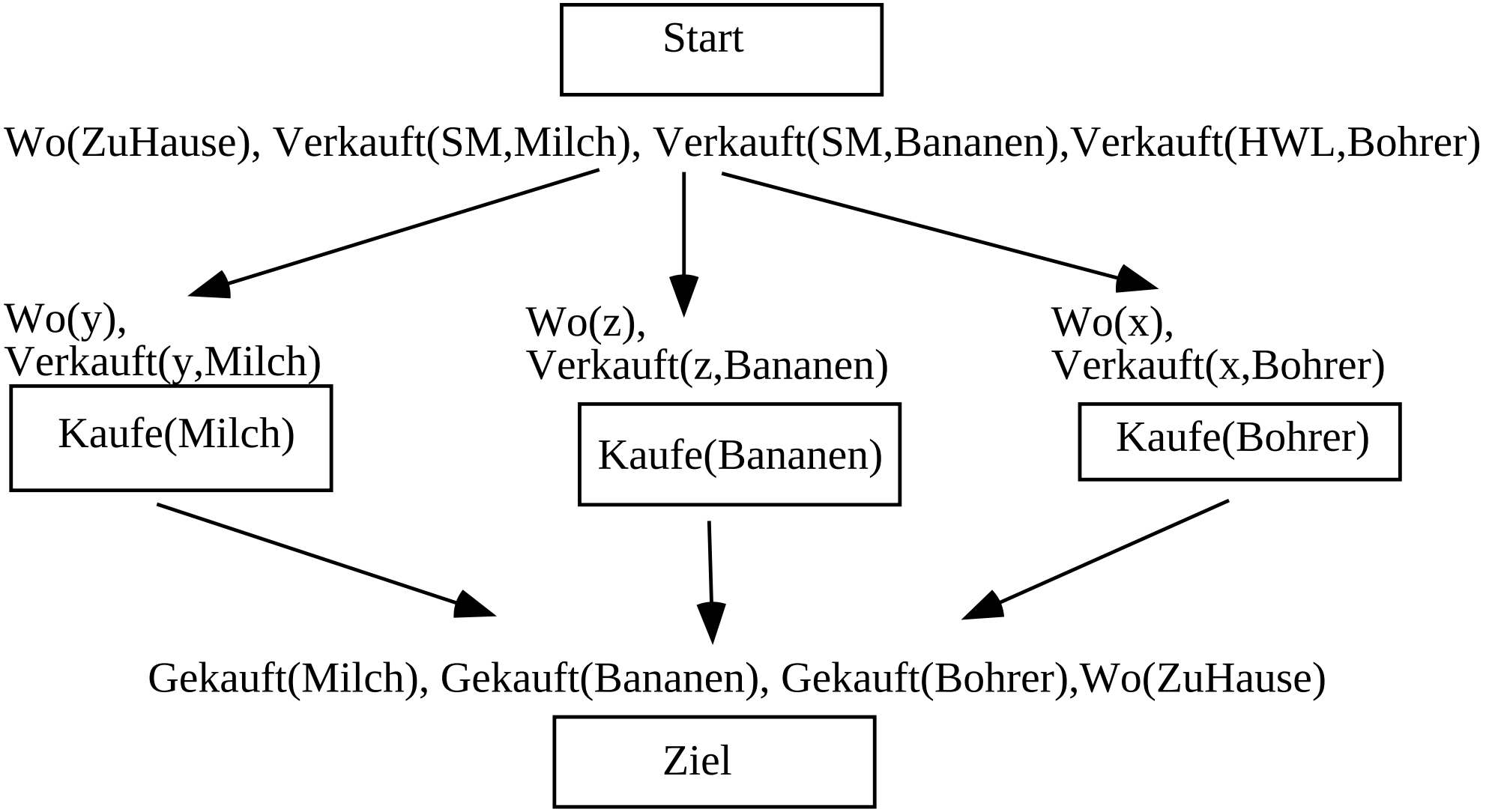
# Beispiel: Milch+Bananen

Planungsstart mit konsistenten Plan,
bestehend nur aus Start und Ziel mit Start $\prec$ Ziel

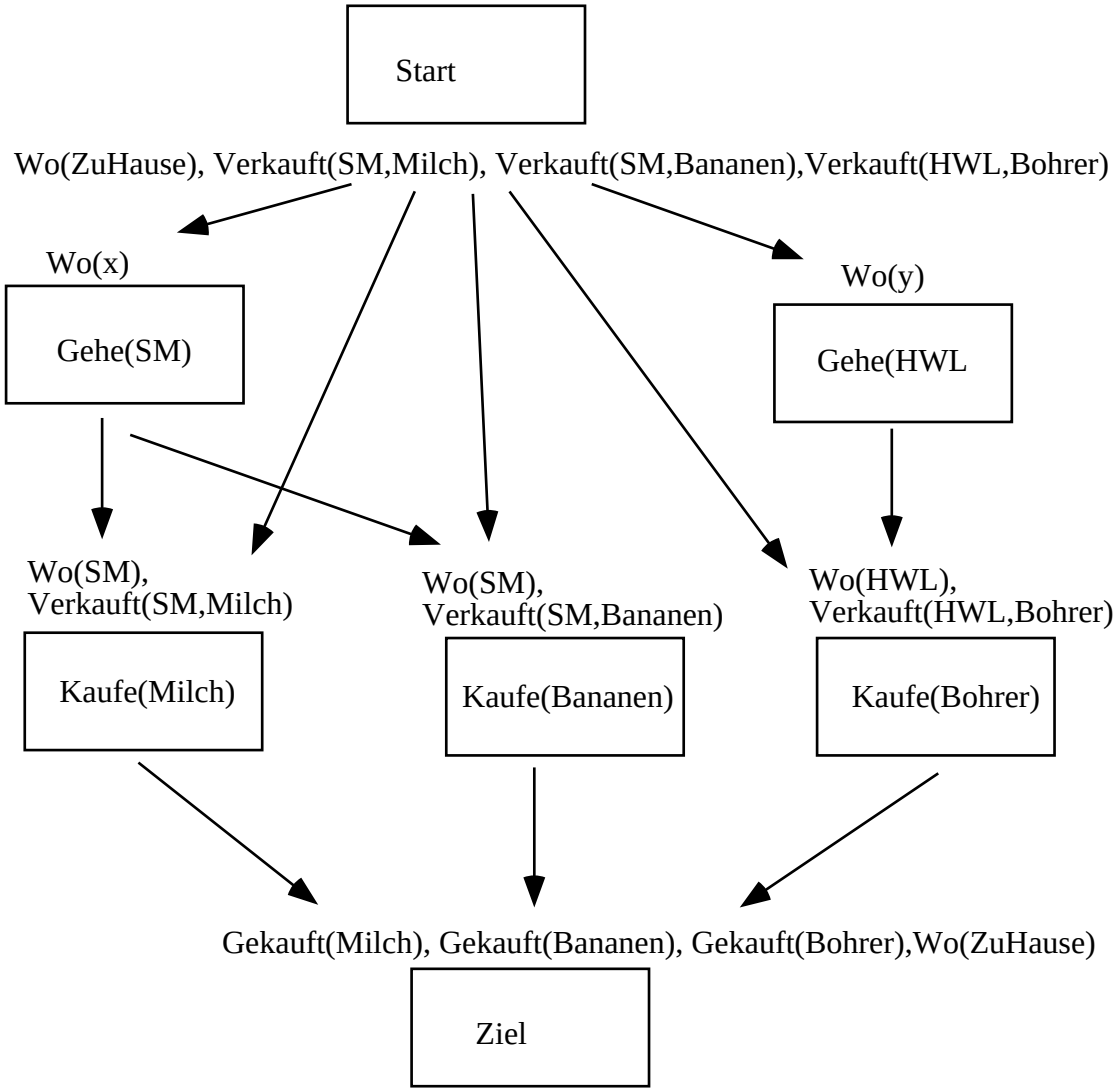
Der partielle Plan wird nach und nach erweitert.



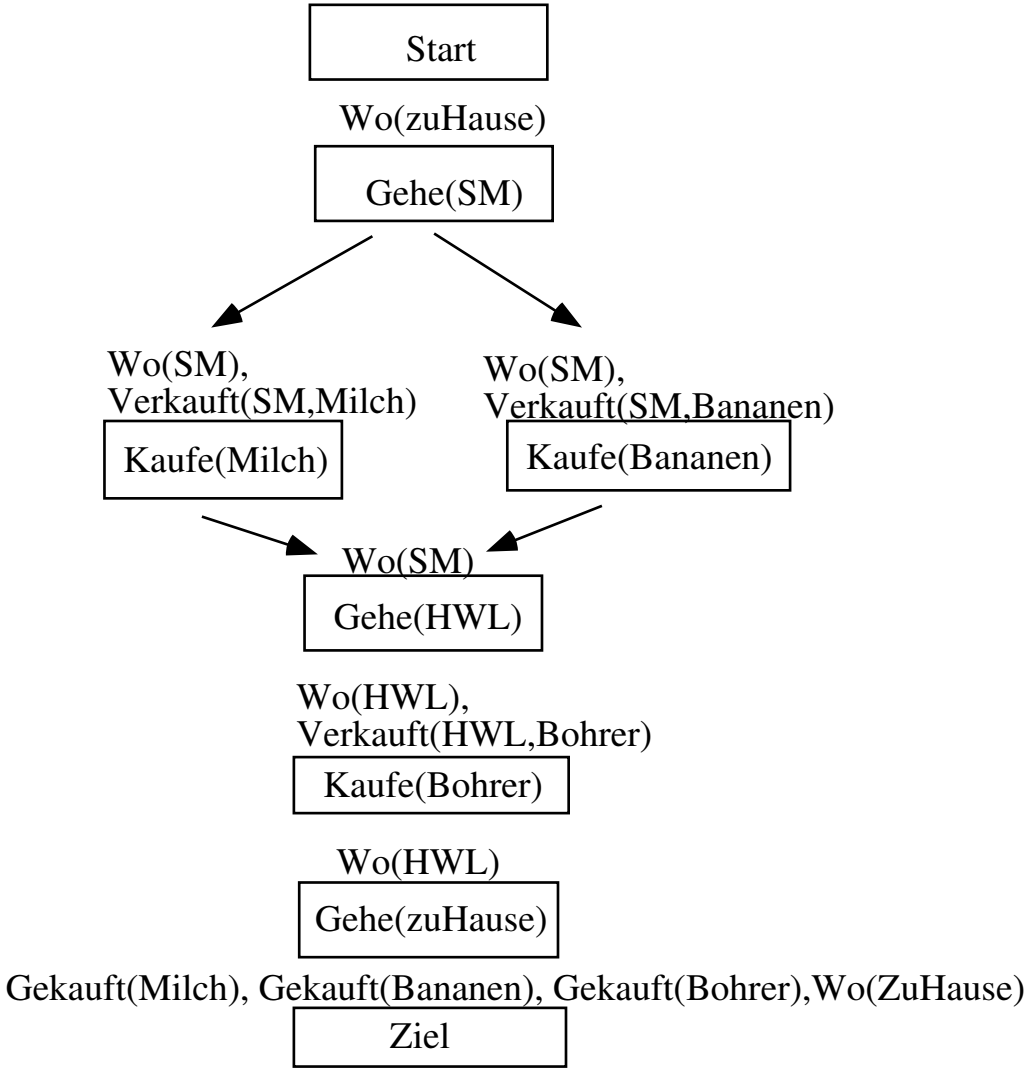
# Beispiel: Milch+Bananen:2



# Beispiel: Milch+Bananen:3



# Beispiel: Milch+Bananen:4



# Erweiterungen

POP-Planen erweitern: Operatoren in Plänen nicht voll instanziieren (siehe Russel und Norvig)

**Hierarchisches Planen**: Man hat verschiedene Abstraktionsebenen der Pläne.
Die Aktionen und Zustände sind entsprechend zu erweitern.

Planung durch Schrittweise Verfeinerungen.

z.B. Reiseroute von der Uni Frankfurt nach Chicago.

Zuerst macht man einen groben Plan, dann eine Feinplanung .

**Zeit-Constraints**:
Dauer von Aktion bzw. Ressourcenbeschränkungen,
Es gibt Analogien zum Job Shop Scheduling (Operations Research-Methoden?)