Kapitel MK:VI

III. Planning

- Motivation
- Wissensrepräsentation
- Planungsalgorithmen
- Suche im Zustandsraum
- Suche im Planraum
- Komplexität
- Erweiterungen

Planungsalgorithmen

Eigenschaften

- Korrektheit

  Ein Planungsalgorithmus ist korrekt, wenn die gefundenen Pläne im Anfangszustand anwendbar sind und zu einem Zustand führen, der das Ziel umfasst.

- Vollständigkeit

  Ein Planungsalgorithmus ist vollständig, wenn er einen Plan findet, der das Planungsproblem löst, sofern ein solcher Plan existiert.

- Optimalität

  Ein Planungsalgorithmus ist optimal, wenn die gefundenen Pläne minimal (z.B. hinsichtlich Länge) unter allen Plänen sind, die das Planungsproblem lösen.
  (Optimalität wird meist vernachlässigt.)

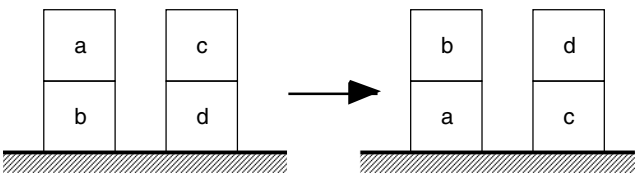
Planungsalgorithmen

Ansätze und Einschränkungen

- Lineares Planen vs. nicht-lineares Planen

  Ziele werden nacheinander bearbeitet in einer festen Reihenfolge. Beim nicht-linearen Planen können Ziele verschränkt werden (Interleaving, Zielmenge statt Zielstack).

  Lineares Planen ist möglich, falls Ziele unabhängig voneinander erreicht werden können, d.h. keine Operation zum Erreichen eines Zieles andere Ziele blockiert.



- Progression vs. Regression

  Die Suche kann vom Anfangszustand ausgehend vorwärts durchgeführt werden oder aber vom Ziel aus rückwärts.

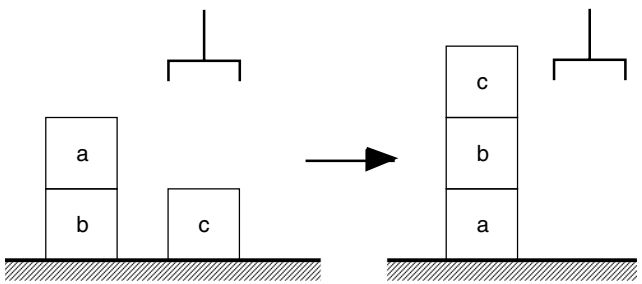
Planungsalgorithmen

Ansätze und Einschränkungen (Fortsetzung)

- Total geordnete Pläne vs. partiell geordnete Pläne

  Pläne können als Sequenz von Aktionen konstruiert werden. Pläne können aber auch als Linearisierungen einer Menge von Aktionen mit Reihenfolge-Constraints gebildet werden, wobei die Menge der Aktionen und der Constraints sukzessive vergrößert werden.

  Nicht jede Gesamtreihenfolge der Operationen aus partiellen Plänen zum Erreichen von Einzelzielen führt zum Gesamtziel, insbesondere müssen Teilpläne ggf. verschränkt werden.



- Zustandsraum-Suche vs. Planraum

  Im Zustandsraum wird der Suchraum durch Zustände gebildet mit Aktionen als Kanten, im Planraum besteht der Suchraum aus Plänen mit Plantransformationen als Kanten.

Planungsalgorithmen

Planen als Suche im Zustandsraum

- Der Suchraum ist ein Graph mit
  - Zustandsbeschreibungen als Knoten sowie
  - Aktionen als Basis für gerichtete Kanten.
- Ein Plan ist ein gerichteter Weg vom Anfangszustand zu einem Zustand, der das geforderte Ziel erfüllt.
- Planungsalgorithmen
  - Vorwärts-Suche (vom Anfangszustand ausgehend vorwärts)
    (Vorwärts-Planen, Forward-Search, Progression Planning)
  - Rückwärts-Suche (vom Ziel ausgehend rückwärts)
    (Rückwärts-Planen, Backward-Search, Regression Planning)
  - Rekursives STRIPS-Planen

Planungsalgorithmen

Planen als Suche im Planraum

- Der Suchraum ist ein Graph mit
  - Mengen von partiell instantiierten Operatoren und Constraints für Operatoren als Knoten sowie
  - Möglichkeiten zur Erweiterung einer Knotenbeschreibung als Basis für gerichtete Kanten.
- Aus den Informationen in einem Zielknoten kann ein gesuchter linearer Plan erzeugt werden.
- Planungsalgorithmen
  - Partial-Order Planning

Suche im Zustandsraum

Vorwärts-Suche im Zustandsraum (Progression Planning)

Um einen Plan für eine Liste von Zielen zu erzeugen, betrachten wir den Anfangszustand und versuchen durch Anwendung von Aktionen einen Zustand zu erreichen, der die Ziele umfasst.

Pseudocode:

- Beginne mit einem leeren Plan und dem Anfangszustand.
- Solange im aktuellen Zustand nicht alle Ziele enthalten sind:
  - Falls keine Aktion mehr anwendbar ist, brich die Suche ab (Fehlschlag).
  - Wähle eine anwendbare Aktion aus.
  - Stelle Aktion an das Ende des bisherigen Plans.
  - Bestimme den Folgezustand der Aktion als neuen aktuellen Zustand.
- Gib den Plan als Ergebnis zurück.

Suche im Zustandsraum

Vorwärts-Suche im Zustandsraum (Progression Planning) (Fortsetzung)

- Vorwärts-Suche ist *korrekt*.
  Ein Plan, der von einem Lauf des nicht-deterministischen Verfahrens zurückgegeben wird, ist eine Lösung des Planungsproblems.
- Vorwärts-Suche ist *vollständig*.
  Wenn ein Plan zur Lösung des Planungsproblems existiert, dann kann von einem Lauf des nicht-deterministischen Verfahrens ein solcher Plan gefunden werden.
- Deterministische Implementationen der Vorwärts-Suche:
  - Breitensuche
    (korrekt und vollständig; exponentieller Platzaufwand in maximal erlaubter Planlänge)
  - Best-First Suche (A*)
    (korrekt; vollständig bei guten Heuristiken; exponentieller Platzaufwand)
  - Tiefensuche
    (korrekt; vollständig bei beschränkter Planlänge; linearer Platzaufwand)
  - Greedy Search
    (korrekt; meist nicht vollständig; linearer Platzaufwand)

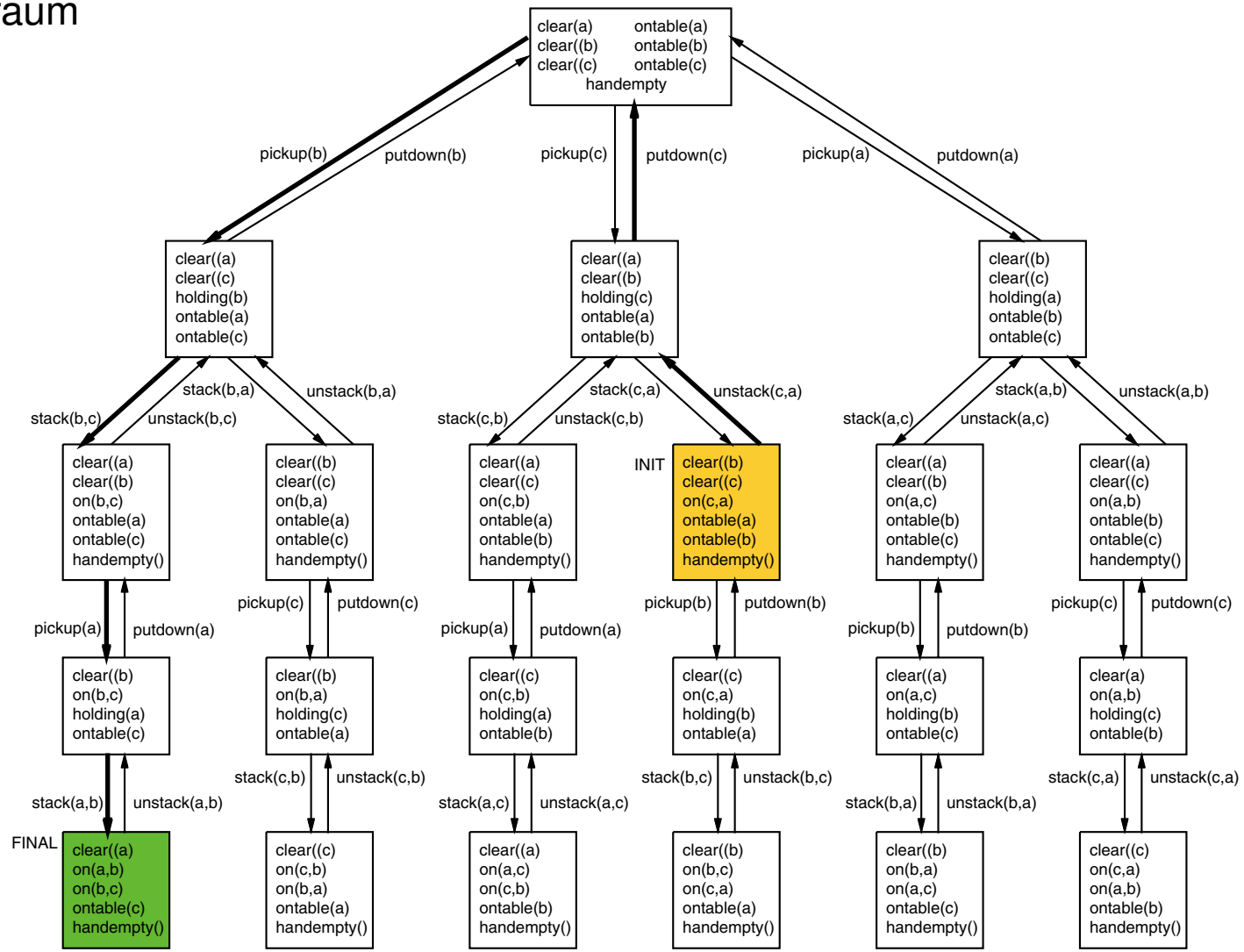
Bemerkungen:

- Der Verzweigungsgrad (Anzahl der anwendbaren Aktionen) spielt für den Suchaufwand eine große Rolle. Auch für die platzeffizienten Verfahren wie Tiefensuche führt ein hoher verzweigungsgrad zu einem hohen zeitlichen Aufwand bei der Durchsuchung von Pfaden, die zum überwiegenden Teil nicht zielführend sind. Die Informierung der Suche durch eine geeignete Heuristik (z.B. Abstand eines Zustands von einem Zielzustand) kann in der Best-First Suche entscheidend zur Effizienzsteigerung beitragen.

Suche im Zustandsraum

Beispiel: Vorwärts-Suche in Blocks World

Suchraum



Suche im Zustandsraum

Rückwärts-Suche im Zustandsraum (Regression Planning)

Um einen Plan für eine Liste von Zielen zu erzeugen, betrachten wir alle zu erreichenden Ziele gleichzeitig. Wird ein Teilziel durch eine Aktion erreicht, stellen die Bedingungen der Aktion neue Teilziele dar.

Pseudocode:

- Beginne mit einem leeren Plan und dem Ziel.
- Solange die aktuellen Zielformeln im Anfangszustand nicht enthalten sind:
  - Wähle eine der nicht erfüllten aktuellen Zielformeln aus.
  - Falls keine Aktion diese Zielformel erreicht, brich die Suche ab (Fehlschlag).
  - Wähle eine Aktion, die die Zielformel erreicht.
  - Stelle Aktion an den Anfang des bisherigen Plans.
  - Das neue Ziel ist eine minimale Formelmenge, die Zustände beschreibt, in denen die gewählte Aktion ausführbar ist, so dass in den Folgezuständen **alle** aktuellen Zielformeln erfüllt sind.
- Gib den Plan als Ergebnis zurück.

Suche im Zustandsraum

Rückwärts-Suche im Zustandsraum (Regression Planning) (Fortsetzung)

- Die Rückwärts-Suche arbeitet nicht mit Zuständen (nur positive Literale) wie die Vorwärts-Suche sondern mit Zielen (positive und negative Literale). Ziele können als partiell bestimmte Zustände aufgefasst werden (CWA).
- Ausgangspunkt der Rückwärts-Suche ist die Umkehrung der Zustandsübergangsrelation (Weakest Precondition):
  $s' \xrightarrow{a}^{-1} s$ gilt gdw s minimal mit der Eigenschaft, dass für alle $s'' \in \mathit{states}(s_{goal})$ die Aktion a anwendbar ist und im jeweiligen Folgezustand das Ziel s' erreicht ist.
  Der Suchraum wird beschrieben durch das Ziel s_{goal} im Startknoten, stellvertretend für die Zustandsmenge $\mathit{states}(s_{goal})$, und die invertierte Übergangsrelation τ^{-1}.
- Wie im Fall der Vorwärts-Suche können deterministische Implementationen für die Suche in diesem Suchraum angeben werden.

Bemerkungen:

- Wann ein Ziel minimal im Sinne der Weakest Precondition heißt, ist hier offen gelassen worden. Im vorgestellten einfachen Fall des Regression Planning kann man die Anzahl der im Ziel enthaltenen positiven und negativen Literale verwenden.
- Während die Vorwärts-Suche Aktionen in den Plan einfügen könnte, die nicht zielführend sind, scheinen die bei der Rückwärts-Suche die in den Plan eingefügten Aktionen „notwendig“ für das Erreichen des Zieles zu sein. Tatsächlich ist die Suche in der beschriebenen Art aber nicht effizienter.

Suche im Zustandsraum

Beispiel: Rückwärts-Suche in Blocks World

- Konstanten: a, b, c, d
- Prädikate: *on*(x, y), *ontable*(x), *holding*(x), *clear*(x), *handempty*$()$
- Operatoren:

  ...

  operator: *unstack*(x, y)

  precond: *on*(x, y), *clear*(x), *handempty*$()$

  effects: *holding*(x), *clear*(y), $\neg$*on*(x, y), $\neg$*clear*(x), $\neg$*handempty*$()$
- Ziel: {*holding*(b), *clear*(c)}
- Bestimmung der Weakest Precondition:

  Substitution: $[x/b, y/c]$ Aktion: *unstack*(b, c)
  Neues Ziel: {*on*(b, c), *clear*(b), *handempty*$()$}

  Substitution: $[x/b, y/a]$ Aktion: *unstack*(b, a)
  Neues Ziel: {*on*(b, a), *clear*(b), *clear*(c), *handempty*$()$}

  Substitution: $[x/b, y/d]$ Aktion: *unstack*(b, d)
  Neues Ziel: {*on*(b, d), *clear*(b), *clear*(c), *handempty*$()$}

Suche im Zustandsraum

Bestimmung der Weakest Precondition

- s sei das aktuelle Ziel, also eine Menge von variablenfreien Literalen.
- a sei eine Aktion, also eine Grundinstanz eines Operators.
- Die Regression eines Literals $L \in s$ durch a ergibt
  1. *true*, falls $L \in$ effects(a),
  2. *false*, falls $\neg L \in$ effects(a),
  3. L in allen anderen Fällen.

  (Beachte, dass L positives oder negatives Literal sein kann.)
- Die Weakest Precondition von s für a besteht aus allen mittels Regression durch a gewonnenen Literalen des aktuellen Ziels s und precond(a).

Bemerkungen:

- Ein Literal *true* in der Weakest Precondition ist in jedem Zustand erfüllt, es kann weggelassen werden.
- Ein Literal *false* in der Weakest Precondition kann von keinem Zustand erfüllt werden, die Suche könnte abgebrochen und an einem Backtracking-Punkt fortgesetzt werden.

Suche im Zustandsraum

Beispiel: Rückwärts-Suche in Blocks World (Fortsetzung)

- Konstanten: a, b, c, d
- Prädikate: *on*(x, y), *ontable*(x), *holding*(x), *clear*(x), *handempty*$()$
- Operatoren:

  ...

  operator: *stack*(x, y)

  precond: *holding*(x), *clear*(y)

  effects: *on*(x, y), *clear*(x), $\neg$*clear*(y), $\neg$*holding*(x), *handempty*$()$
- Ziel: {*on*(b, c), *on*(a, b)}

  Aktion *stack*(x, y) kann Ziel *on*(b, c) liefern.
- Bestimmung der Weakest Precondition:

  Substitution: $[x/b, y/c]$ Aktion: *stack*(b, c)
  Neue Ziele: {*holding*(b), *clear*(c), *on*(a, b)}

➜ Neues Ziel nicht konsistent!

Suche im Zustandsraum

Beispiel: Rückwärts-Suche in Blocks World (Fortsetzung)

- Konstanten: a, b, c, d
- Prädikate: *on*(x, y), *ontable*(x), *holding*(x), *clear*(x), *handempty*$()$
- Operatoren:

  ...

  operator: *pickup*(x)
  precond: *ontable*(x), *clear*(x), *handempty*$()$
  effects: *holding*(x), $\neg$*ontable*(x), $\neg$*clear*(x), $\neg$*handempty*$()$

  operator: *putdown*(x)
  precond: *holding*(x)
  effects: *ontable*(x), *clear*(x), *handempty*$()$, $\neg$*holding*(x)
- Ziel: {*holding*(a), *clear*(b), *on*(b, c)}
- Aktion *pickup*(x) kann Ziel *holding*(a) liefern mit Substitution $[x/a]$
  Weakest Precondition: {*ontable*(a), *clear*(a), *handempty*$()$, *clear*(b), *on*(b, c)}
- Aktion *putdown*(x) kann Ziel *clear*(a) liefern mit Substitution $[x/a]$
  Weakest Precondition: {*holding*(a), *clear*(b), *on*(b, c)}

➜ Zyklus in der Folge der Ziele (partiellen Zustände)!

Bemerkungen:

- Für die Erkennung von Zyklen reicht es nicht aus festzustellen, ob Ziele mehrfach auftreten. Da das Ziel s_{goal} nur ein partiell festgelegter Zustand ist, können durch die Rückwärts-Suche Obermengen hiervon generiert werden. Obermengen eines Zieles stellen aber keinen Fortschritt dar.
- Bei der Rückwärts-Suche ist das Vermeiden von unsinnigen Aktionen wie *unstack*(a, a) enorm wichtig. Durch Einfügen zusätzlicher Constraints in den precond Teil, im Beispiel etwa $x \neq y$, lassen sich unerwünschte Spezialisierungen vermeiden. Ein Konsistenztest aller (Un-)Gleichungen vor dem Hintergrund der Unique Name Assumption ist einfach.

Suche im Zustandsraum

Beispiel: Rückwärts-Suche in Blocks World (Fortsetzung)

- Konstanten: a, b, c, d
- Prädikate: *on*(x, y), *ontable*(x), *holding*(x), *clear*(x), *handempty*$()$
- Operatoren:

  ...

  operator: *unstack*(x, y)

  precond: *on*(x, y), *clear*(x), *handempty*$()$

  effects: *holding*(x), *clear*(y), $\neg$*on*(x, y), $\neg$*clear*(x), $\neg$*handempty*$()$
- Ziel: {*holding*(b), *clear*(c)}
- Bestimmung der Weakest Precondition:

  Substitution: $[x/b]$ Aktion: *unstack*(b, y)

  Neues Ziel: {*on*(b, c), *clear*(b), *handempty*$()$}
  oder
  Neues Ziel: {*on*$(b, y), y \neq c$, *clear*(b), *clear*(c), *handempty*$()$}

➜ Lifting

Suche im Zustandsraum

Lifted Backward Search im Zustandsraum (Regression Planning)

Statt wie bei Backward Search Grundinstanzen für Operatoren zu betrachten, werden die Operatoren durch.Substitutionen nur so weit spezialisiert, wie es für das Erreichen eines Zieles nötig ist.

Pseudocode:

- Beginne mit einem leeren Plan und dem Ziel.
- Solange keine Grundinstanz der aktuellen Zielformeln im Anfangszustand enthalten ist:
  - Wähle eine der nicht erfüllten aktuellen Zielformeln aus.
  - Falls keine Aktion eine Grundinstanz dieser Zielformel erreicht, brich die Suche ab (Fehlschlag).
  - Wähle eine Instanz eines Operators mit neuen Variablen, die mit einem Most General Unifier σ als Substitution die Zielformel erreicht.
  - Stelle den durch σ partiell instantiierten Operator an den Anfang des bisherigen Plans.
  - Das neue Ziel ist eine minimale Formelmenge, deren Grundinstanzen Zustände beschreiben, in denen jeweils die zugehörige Grundinstanz des Operators ausführbar ist, so dass im Folgezustand die zugehörige Grundinstanz der aktuellen Zielformeln erfüllt ist.
- Gib eine passende Grundinstanz des Plan als Ergebnis zurück.

Suche im Zustandsraum

Bestimmung der Weakest Precondition für Lifting

- s sei das aktuelle Ziel, also eine Menge von (i.a. nicht variablenfreien) Literalen.
- $o(x_1, ..., x_m)$ sei ein Operator ($x_1, ..., x_m$ neu) und σ eine Instantiierung von o.
  (Beachte, dass σ auch Variablen von s ersetzen kann, also s spezialisieren kann!)
- Die Regression eines Literals $L \in s$ kann alternative Erweiterungen von σ ergeben:
  - $\sigma_1, \ldots, \sigma_p$ seien die MGUs, mit denen L erreicht werden kann,
    d.h. $\sigma_i(\sigma(L)) \in$ effects$(\sigma_i(\sigma(o)))$.
  - $\sigma_{p+1}, \ldots, \sigma_{p+n}$ seien die MGUs, mit denen L widersprochen werden kann,
    d.h. $\sigma_i(\sigma(\neg L)) \in$ effects$(\sigma_i(\sigma(o)))$.

  a. Jedes σ_i mit $i \in \{1, \ldots, p\}$ ergibt eine Regression von L durch $\sigma_i(\sigma(o))$ mit dem Ergebnis:
  true.

  b. Jedes σ_i mit $i \in \{p+1, \ldots, p+n\}$ ergibt eine Regression von L durch $\sigma_i(\sigma(o))$ mit dem Ergebnis:
  false.

  c. Jede Auswahl $(y_1, \ldots, y_{p+n})$ mit $y_i \neq \sigma_i(y_i)$ für $i \in \{1, \ldots, p+n\}$ ergibt eine Regression von L durch $\sigma(o)$ mit dem Ergebnis:
  $\sigma(L)$ und $y_1 \neq \sigma_1(y_1)$ und ... und $y_{p+n} \neq \sigma_{p+n}(y_{p+n})$ (nur $\sigma(L)$ für $n = 0$).

- Mit den alternativen Erweiterungen von σ werden Regressionen weiterer Literale $L' \in s$ durchgeführt.

Bemerkungen:

- Für die Auswahl von Variablen zum Verbot bestimmter MGUs in Schritt c) gibt es nur endlich viele Möglichkeiten, da nur endlich viele Variablen in einem MGU zweier Formeln verändert werden können. Die Auswahl einer unveränderten Variablen führt zu einer Bedingung $x \neq x$, die niemals erfüllt werden kann, so dass solche Fälle unbeachtet bleiben können.
- Zu beachten ist, dass jede Operation mit neuen, bisher noch nicht verwendeten Variablennamen angewendet wird. Sonst wären die bestimmten MGUs zu speziell.
- Insgesamt wird bei einer nicht-deterministischen Auswahl bei den Alternativen insgesamt eine Substitution aufgebaut und fortlaufend ergänzt.
- Die Bestimmung der Regression über eine Operation mit Lifting erlaubt verschiedene Alternativen. Diesen Nicht-Determinismus muss ein deterministisches Verfahren durch Verwalten von Choice Points nacheinander überprüfen.

Suche im Zustandsraum

Bestimmung der Weakest Precondition für Lifting (Fortsetzung)

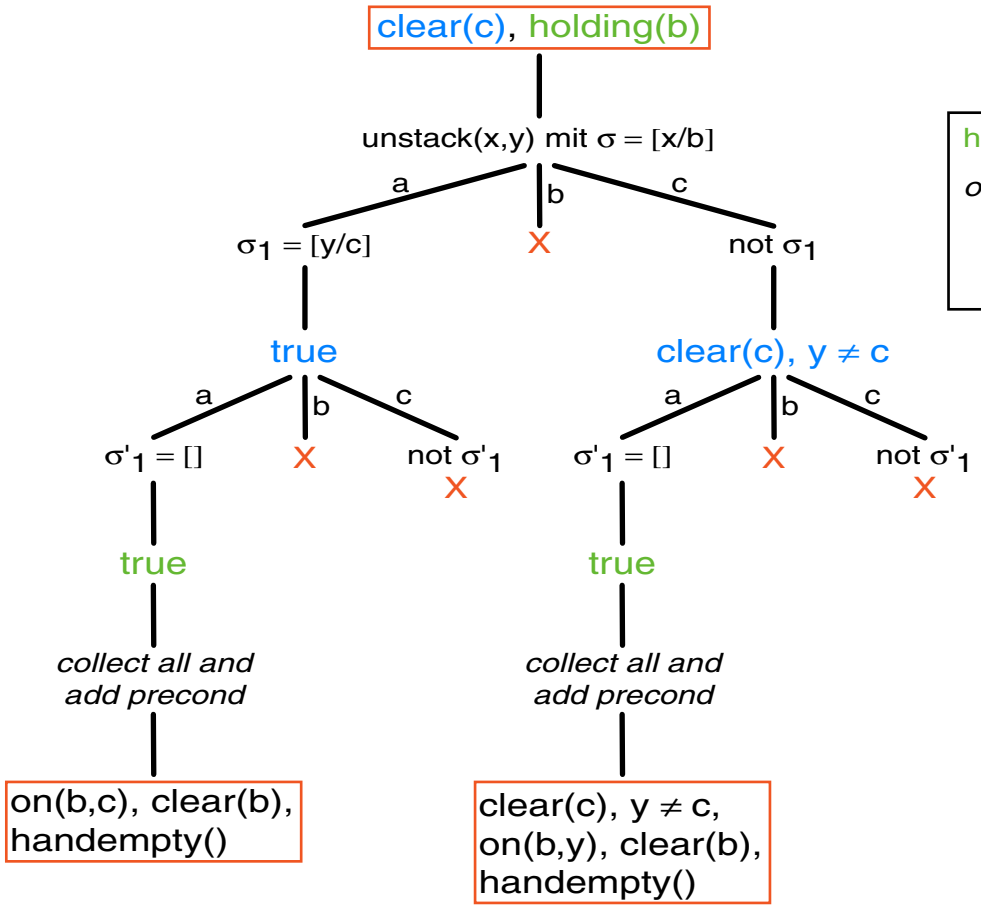
- Die Operation o und eine Anfangssubstitution σ ergibt sich aus der Auswahl eines Literals aus dem aktuellen Ziel s, das mit der Operation o unter Verwendung von σ erreicht werden kann.
- Die verschiedenen alternativen Weakest Preconditions ergeben sich, indem man für jedes weitere Literal des aktuellen Ziels eine der Regressionen aus den Fällen a.), b.) oder c.) anwendet und die Vorbedingungen precond$(\sigma'(o))$ unter der Gesamtsubstitution hinzufügt.

➜ Die Anzahl der Alternativen ist abhängig von der Größe der Menge effects(o). Verschiedene Pläne ergeben sich durch unterschiedliche Operatorauswahlen und unterschiedliche Instantiierungen.

➜ Da die Ziele während des Regression-Planning mit Lifting Variablen enthalten können, fassen sie so eine Vielzahl möglicher Ziele zusammen, die sich ohne Lifting als Grundinstanzen ergäben. Der Verzweigungsgrad des Suchraumes ist mit Lifting also geringer.

Suche im Zustandsraum

Beispiel 1: Weakest Precondition mit Lifting in Blocks World



Ziel

holding(b)	erreichbar mit σ = [x/b] durch
operator:	unstack(x,y)
precond :	on(x,y), clear(x), handempty()
effects :	holding(x), clear(y), ¬on(x,y), ¬clear(x), ¬handempty()

Alternative Weakest Preconditions

Suche im Zustandsraum

Beispiel 2: Weakest Precondition mit Lifting in Blocks World

- Situation: Blocks World mit Ziel $\{\mathit{holding}(b), \mathit{ontable}(c), \mathit{clear}(a), \neg\mathit{clear}(z)\}$
- Ausgewählte Zielformel $\mathit{holding}(b)$
- Ausgewählte Operation $\mathit{unstack}(x, y)$ mit Substitution $\sigma = [x/b]$

  operator: $\mathit{unstack}(x, y)$
  precond: $\mathit{on}(x, y), \mathit{clear}(x), \mathit{handempty}()$
  effects: $\mathit{holding}(x), \mathit{clear}(y), \neg\mathit{on}(x, y), \neg\mathit{clear}(x), \neg\mathit{handempty}()$

- Regression von $\mathit{ontable}(c)$ mit $\mathit{unstack}(x, y)$ und $\sigma = [x/b]$
  Da das Prädikat *ontable* in effects($\mathit{unstack}(x, y)$) nicht auftritt, kommt nur Fall c.) für eine Regression in Frage. Da keine Substitutionen nach Fall a.) oder b.) existieren, wird nur $\mathit{ontable}(c)$ in die Weakest Precondition übernommen.
- Regression von $\mathit{clear}(a)$ mit $\mathit{unstack}(x, y)$ und $\sigma = [x/b]$
  Eine Erweiterung der Substitution um $[y/a]$ liefert diese Zielformel. Keine Substitution kann einen Widerspruch zur der Zielformel liefern. Im Fall c.) würde die Weakest Precondition um $\mathit{clear}(a)$ und $y \neq a$ ergänzt. Im Fall a.) erfolgt die Erweiterung von σ zu $[x/b, y/a]$.

Suche im Zustandsraum

Beispiel 2: Weakest Precondition mit Lifting in Blocks World (Fortsetzung)

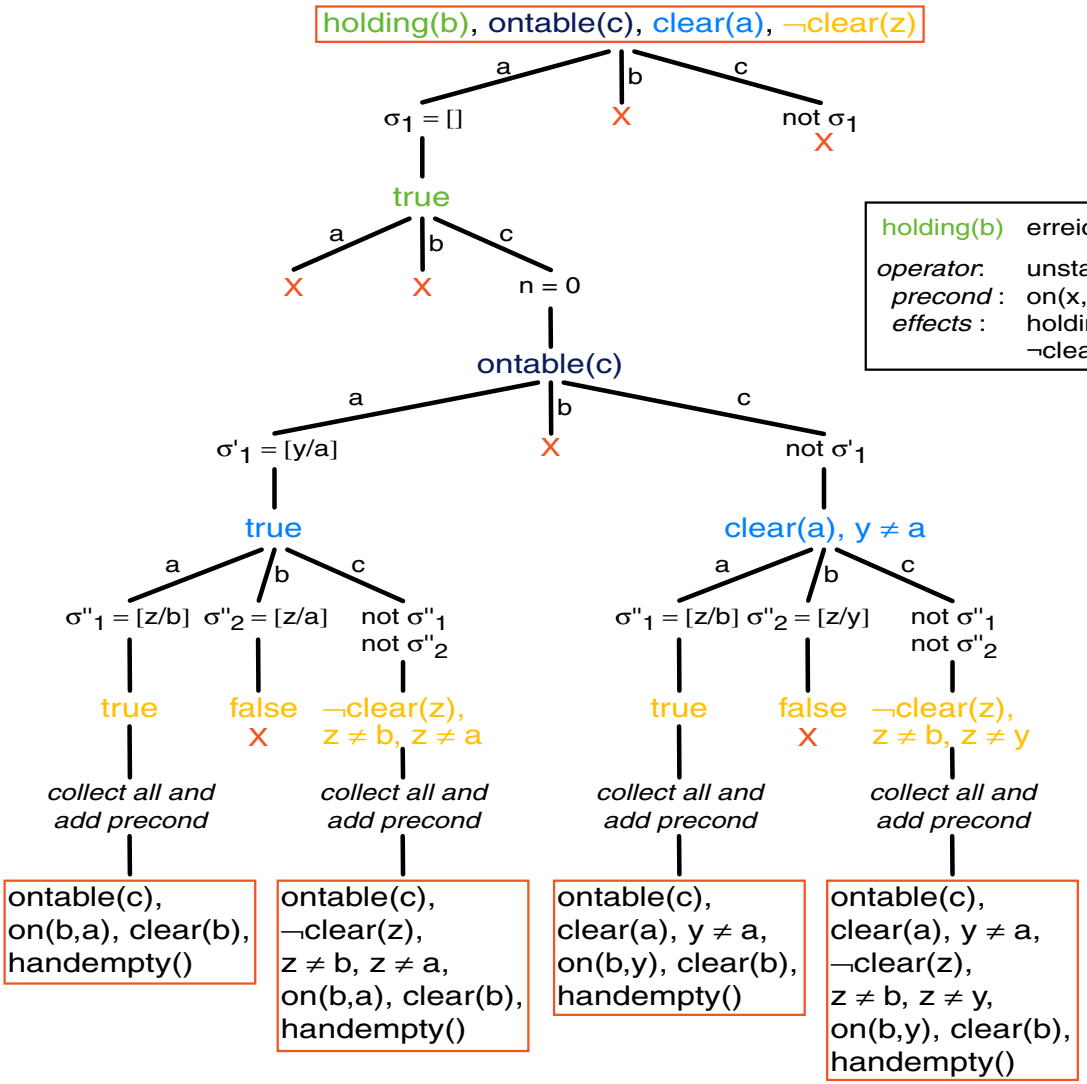
- Situation: Blocks World mit Ziel $\{\mathit{holding}(b), \mathit{ontable}(c), \mathit{clear}(a), \neg\mathit{clear}(z)\}$
- Ausgewählte Zielformel $\mathit{holding}(b)$
- Ausgewählte Operation $\mathit{unstack}(x, y)$ mit Substitution $\sigma = [x/b]$

  operator: $\mathit{unstack}(x, y)$
  precond: $\mathit{on}(x, y), \mathit{clear}(x), \mathit{handempty}()$
  effects: $\mathit{holding}(x), \mathit{clear}(y), \neg\mathit{on}(x, y), \neg\mathit{clear}(x), \neg\mathit{handempty}()$

- Regression von $\neg\mathit{clear}(z)$ mit $\mathit{unstack}(x, y)$ und $\sigma = [x/b]$
  Eine Erweiterung der Substitution um $[z/y]$ liefert die Negation dieser Zielformel, gehört also zu Fall b.). Eine Erweiterung der Substitution um $[z/b]$ liefert die Zielformel, gehört also zu Fall a.). Im Fall c.) würde die Weakest Precondition um $\neg\mathit{clear}(z)$ und $z \neq a$ und $z \neq b$ ergänzt.
- Regression mit $\mathit{unstack}(x, y)$ und $\sigma = [x/b, z/b]$
  Hinzugefügen der Vorbedingungen des Operators liefert die Weakest Precondition
  $\{\mathit{ontable}(c), \mathit{clear}(a), y \neq a, \mathit{on}(b, y), \mathit{clear}(b), \mathit{handempty}()\}$
  y ist als existenzquantifizierte Variable aufzufassen.

Suche im Zustandsraum

Beispiel 2: Weakest Precondition mit Lifting in Blocks World (Fortsetzung)



Bemerkungen:

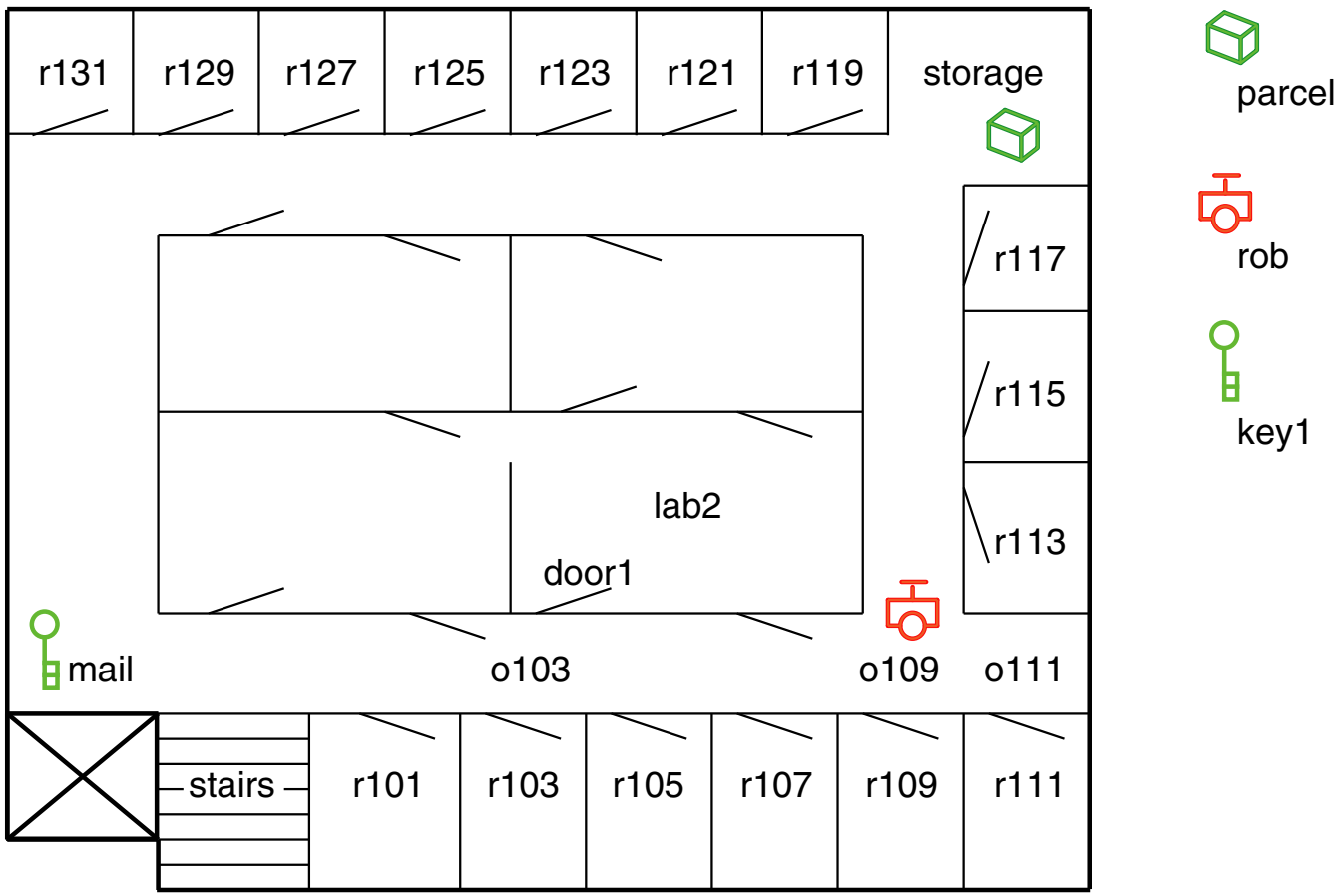
- Es ist zu beachten, dass Ersetzungen für Variablen, die bereits im Ziel vorkommen, Auswirkungen auf die bisher durchgeführte Plansuche haben. Auch dort, insbesondere im bisherigen Plan sind die Ersetzungen durchzuführen.

Suche im Zustandsraum

Lifted Backward Search

Prolog-Implementierung nach Poole, Mackworth, Goebel

Anwendungsbeispiel



Suche im Zustandsraum

Lifted Backward Search

Prolog-Implementierung nach Poole, Mackworth, Goebel (Fortsetzung)

- Beschreibung von Operatoren, z.B. *move*$(Ag, Pos1, Pos2)$

  preconditions(*move*$(Ag, Pos1, Pos2)$,
  [*autonomous*(Ag), *adjacent*$(Pos1, Pos2)$, *sitting_at*$(Ag, Pos1)$]).
  achieves(*move*$(Ag, Pos1, Pos2)$, *sitting_at*$(Ag, Pos2)$).
  deletes(*move*$(Ag, Pos1, Pos2)$, *sitting_at*$(Ag, Pos1)$).

- Beschreibung statische Fakten und abgeleiteter Eigenschaften (Auswahl)

  at$(Obj, Pos) \leftarrow$ [*sitting_at*(Obj, Pos)].
  at$(Obj, Pos) \leftarrow$ [*autonomous*$(Ag), Ag \neq Obj$, *carrying*(Ag, Obj), *at*(Ag, Pos)].

  adjacent(*o109*, *o103*) $\leftarrow$ [ ].
  ...
  opens(*key1*, *door1*) $\leftarrow$ [ ].
  ...

- Beschreibung des Anfangszustandes

  holds(*sitting_at*(*rob*, *o109*), *init*).
  holds(*sitting_at*(*parcel*, *storage*), *init*).
  holds(*sitting_at*(*key1*, *mail*), *init*).

Suche im Zustandsraum

Lifted Backward Search

Prolog-Implementierung nach Poole, Mackworth, Goebel (Fortsetzung)

- *holds_all*(GL, S) ist wahr, wenn jedes Element der Goalliste GL im Zustand S erfüllt ist.
- *consistent*(GL) ist wahr, wenn die Goalliste GL konsistent ist, d.h. wenn die Goalliste GL in einem Zustand des Suchraums erfüllt werden kann..
- *choose_goal*(G, GL) ist wahr, wenn G ein nichtdeterministisch gewähltes Element der Goalliste GL ist.
- *choose_action*(A, G) ist wahr, wenn A eine nichtdeterministisch gewählte Aktion ist, die das Ziel G erreicht.
- *weakest_precondition*(A, GL, WP) ist wahr, wenn WP die schwächste Vorbedingung ist, die unmittelbar vor Aktion A gelten muss, damit die Goalliste GL unmittelbar nach der Aktion A erfüllt ist.

weakest_precondition$(A, [\,], P) \leftarrow$ *preconditions*(A, P).

weakest_precondition$(A, [G|GL], P_2) \leftarrow$
weakest_precondition$(A, GL, P_1) \wedge$ *regress*(G, A, P_1, P_2).

Suche im Zustandsraum

Lifted Backward Search

Prolog-Implementierung nach Poole, Mackworth, Goebel (Fortsetzung)

- *regress*(G, A, P_1, P_2) ist wahr, wenn P_2 die schwächste Vorbedingung ist, die P_1 umfasst und unmittelbar vor Aktion A gelten muss, damit das Goal G unmittelbar nach der Aktion A erfüllt ist.

  regress$(G, A, P, P) \leftarrow$ *on_add_list*(A, G).

  regress$(G, A, P, [G|P]) \leftarrow$ *not_on_add_list*$(A, G) \wedge$ *not_on_add_list*(A, G).

- *solve*(GL, W) ist wahr, wenn W ein Zustand ist, in dem alle Ziele der Liste GL erfüllt sind.

  solve$(GL,$ *init*$) \leftarrow$ *holds_all*$(GL,$ *init*$)$.

  solve$(GL,$ *do*$(A, W)) \leftarrow$
  consistent$(GL) \wedge$
  choose_goal$(G, GL) \wedge$
  choose_action$(A, G) \wedge$
  weakest_precondition$(A, GL, NGL) \wedge$
  solve(NGL, W).

Suche im Zustandsraum

Rekursives STRIPS-Planen

Um einen Plan für eine Liste von Zielen zu erzeugen, wählen wir ein Teilziel aus, erzeugen einen Plan zum Erreichen dieses Zieles hängen daran einen Plan zum Erreichen der restlichen Ziele an.

Pseudocode:

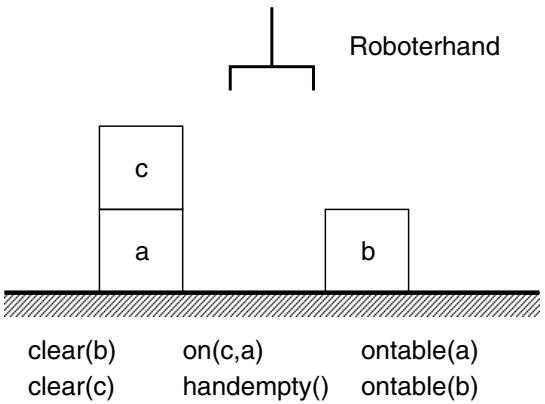
- Beginne mit einem leeren Plan, dem Anfangszustand als aktueller Zustand und dem Ziel als aktuelle Menge von Zielformeln.
- Solange die aktuellen Zielformeln im aktuellen Zustand nicht enthalten sind:
  - Wähle eine der nicht erfüllten aktuellen Zielformeln aus.
  - Falls keine Aktion diese Zielformel erreicht, brich die Suche ab (Fehlschlag).
  - Wähle eine Aktion, die die Zielformel erreicht.
  - Rekursion: Bestimme einen Teilplan vom aktuellen Zustand zum Erreichen der Vorbedingungen der Aktion als Ziel.
  - Hänge an den Teilplan die gewählte Aktion an.
  - Bestimme den aktuellen Zustand als Folgezustand der Anwendung des Teilplanes.
  - Hänge den Teilplan an den bisherigen Plan an.
- Gib den Plan als Ergebnis zurück.

➜ Divide and Conquer

Suche im Zustandsraum

Beispiel Sussman Anomalie in Blocks World

- Anfangszustand:

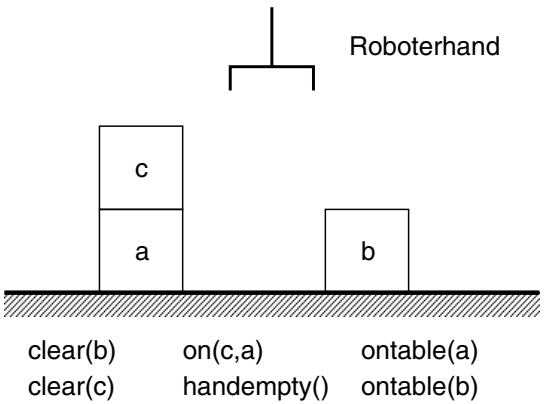


- Anfangszustand:
  $\{\mathit{clear}(b), \mathit{clear}(c), \mathit{on}(c, a), \mathit{handempty}(), \mathit{ontable}(a), \mathit{ontable}(b)\}$
- Ziel: $\{\mathit{on}(b, c), \mathit{on}(a, b)\}$

Suche im Zustandsraum

Beispiel Sussman Anomalie in Blocks World

- Anfangszustand:



- Anfangszustand:
  $\{\mathit{clear}(b), \mathit{clear}(c), \mathit{on}(c,a), \mathit{handempty}(), \mathit{ontable}(a), \mathit{ontable}(b)\}$
- Ziel: $\{\mathit{on}(b,c), \mathit{on}(a,b)\}$

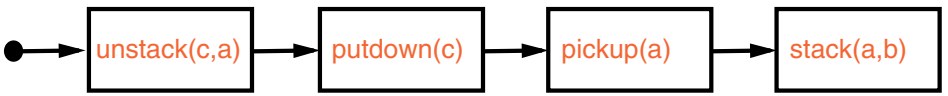
- Rekursives STRIPS-Planen:
  Erste Zielformel $\mathit{on}(a,b)$ muss nach Erreichen wieder zerstört werden.
  Erste Zielformel $\mathit{on}(b,c)$ muss nach Erreichen wieder zerstört werden.

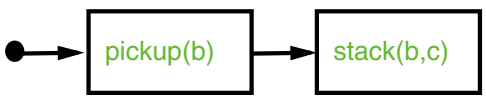
➜ Rekursives STRIPS-Planen kann nicht den kürzesten Plan liefern.

Suche im Zustandsraum

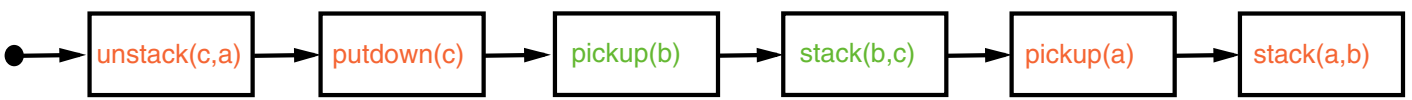
Beispiel Sussman Anomalie in Blocks World (Fortsetzung)

- Partielle Pläne zum Erreichen der Teilziele $on(b, c)$ und $on(a, b)$





- Korrekter Gesamtplan



Suche im Zustandsraum

Unabhängigkeit von Zielen

Problem:
Ziele sind im allgemeinen nicht unabhängig, d.h. das Erreichen von Zielen kann andere, bereits erreichte Teilziele wieder zerstören.

Beispiel:

Zielzustand: $\{\mathit{on}(b, c), \mathit{on}(a, b)\}$

Startzustand: $\{\mathit{clear}(a), \mathit{clear}(c), \mathit{on}(a, b), \mathit{handempty}(), \mathit{ontable}(b), \mathit{ontable}(c)\}$

Suche im Zustandsraum

Unabhängigkeit von Zielen

Problem:
Ziele sind im allgemeinen nicht unabhängig, d.h. das Erreichen von Zielen kann andere, bereits erreichte Teilziele wieder zerstören.

Beispiel:

Zielzustand: $\{\mathit{on}(b, c), \mathit{on}(a, b)\}$

Startzustand: $\{\mathit{clear}(a), \mathit{clear}(c), \mathit{on}(a, b), \mathit{handempty}(), \mathit{ontable}(b), \mathit{ontable}(c)\}$

Lösungsansätze:

- Schutz erreichter Ziele
  $\leadsto$ unvollständig, eventuell kein erfolgreicher Plan mehr vorhanden
- Erneutes Erreichen von zerstörten Teilzielen (Lösung im originalen STRIPS-Planen)
  $\leadsto$ eventuell kein kürzester Plan

Suche im Zustandsraum

Rekursives STRIPS-Planen

Prolog-Implementierung nach Poole, Mackworth, Goebel

- *achieve_all*$(GL, W1, W2)$ ist wahr, wenn $W2$ ein Zustand ist, der aus dem Erreichen aller Ziele der Liste GL resultiert ausgehend vom Zustand $W1$.

  achieve_all$([\,], W1, W1)$.

  achieve_all$([G|GL], W1, W3) \leftarrow$
  achieve$(G, W1, W2) \wedge$ *achieve_all*$(GL, W2, W3)$.

  Die Übergabe der Ziele als Liste gibt eine bestimmte Reihenfolge vor. Wenn verschiedene Zielreihenfolgen betrachtet werden sollen, dann muss die Zerlegung der Liste in Einzelziel und andere Ziele in alternativer Weise realisiert werden.

- *achieve*$(G, W1, W2)$ ist wahr, wenn $W2$ ein Zustand ist, der aus dem Erreichen des Zieles G resultiert ausgehend vom Zustand $W1$.

  achieve$(G, W, W) \leftarrow$ *holds*(G, W).

  achieve$(G, W1, W2) \leftarrow$
  clause$(G, B) \wedge$ *achieve_all*$(B, W1, W2)$.

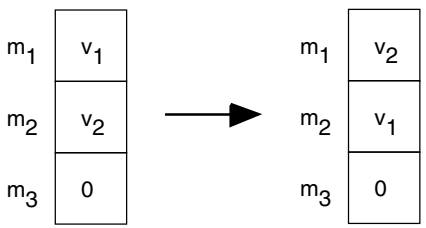
  achieve$(G, W1,$ *result*$(A, W2)) \leftarrow$
  add_list$(A, AL) \wedge$ *member*$(G, AL) \wedge$ *preconditions*$(A, CL) \wedge$
  achieve_all$(CL, W1, W2)$.

Suche im Zustandsraum

Beispiel: Swapping Problem

Modellierung des Swap von Variablenwerten mittels Assign-Anweisungen

- Konstanten: $m_1, m_2, m_3, v_1, v_2, 0$
- Prädikate: *contains*(x, y)
- Operatoren:
  operator: *assign*(x_1, y_1, x_2, y_2)
  precond: *contains*(x_1, y_1), *contains*(x_2, y_2)
  effects: $\neg$*contains*(x_1, y_1), *contains*(x_1, y_2)



- Anfangszustand: {*contains*(m_1, v_1), *contains*(m_2, v_2), *contains*$(m_3, 0)$}
- Ziel: {*contains*(m_1, v_2), *contains*(m_2, v_1)}
- Plan: (*assign*$(m_3, 0, m_2, v_2)$, *assign*(m_2, v_2, m_1, v_1), *assign*(m_1, v_1, m_3, v_2))

➜ Originales STRIPS-Planen kann das Problem nicht lösen!

Bemerkungen:

- Die originale Version des STRIPS-Algorithmus verwendet Lifting. Backtrack-Punkte sind in dem Verfahren die Auswahl der Zielformel und die Auswahl der Aktion zum Erreichen dieser Zielformel. Um eine Zielformel im aktuellen Zustand wiederzufinden (Schleifenbedingung), können zwar verschiedene Substitutionen probiert werden, aber der Versuch, sie mit einem Operator zu erreichen, unterbleibt, wenn eine Substitution möglich ist! (Siehe auch Nilsson S. 298-307.)
- Die Prolog Implementation nach Poole, Mackworth, Goebel hat den beschriebenen Nachteil nicht.

Suche im Planraum

Unterschiede

- Suche im Zustandsraum
  - Knoten = Zustände der Welt
  - Kanten = Aktionen, die Zustände überführen
  - Plan = Pfad durch Zustandsraum
- Suche im Planraum
  - Knoten = Partielle Pläne
  - Kanten = Plantransformationen
  - Plan = Element des Zielknotens

➜ Partial-Order Planning ist Suche im Planraum unter Verwendung einer Least Commitment Strategie.

Bemerkungen:

- Für Lifted Backward Search im Zustandsraum und analoges Vorgehen beim Partial Order Planning im Planraum kann die Verwendung von Variablen als Abkürzung der Verwendung von Mengen von Zuständen bzw. Mengen von partiellen Plänen in den Knoten auffassen.

Suche im Planraum

Partielle Pläne

Ein partieller Plan ist ein Tupel (AS, OS, CS) mit:

- AS ist eine Menge von Instanzen von Aktionen.
- OS definiert eine partielle Ordnung auf AS.
  $a_1 < a_2$ meint, dass a_1 vor a_2 ausgeführt werden muss.
- CS ist eine Menge von kausalen Links (a_{adds}, P, a_{pre}), die Abhängigkeiten zwischen Aktionen beschreiben:
  - P ist eine Eigenschaft (atomare Formel),
  - a_{adds} ist Instanz einer Aktion, die P erzeugt, d.h. P ist in der *effects*-Liste von a_{adds},
  - a_{pre} ist Instanz einer Aktion, die P als Vorbedingung benötigt, d.h. P ist in der *precond*-Liste von a_{pre},
  - $a_{adds} < a_{pre}$ ist Teil von OS.

Ein partieller Plan (AS', OS', CS') ist Erweiterung des Plans (AS, OS, CS), wenn gilt $AS \subseteq AS'$, $OS \subseteq OS'$ und $CS \subseteq CS'$

Bemerkungen:

- Aus Gründen der Einfachheit betrachten wir zwei weitere Aktionen *start* und *finish*, die den Startzustand über die *effects*-Liste bzw. die zu erreichenden Ziele als *precond*-Liste darstellen.
- Da nicht ausgeschlossen ist, dass dieselbe Aktion mehrfach in einem Plan auftritt, betrachten wir in AS Instanzen von Aktionen, d.h. die verschiedenen Anwendungen einer Aktion sind unterscheidbar. Alle Aktionen außer *start* und *finish* können mehrfach in AS auftreten.
- Zu jedem Tupel $(a_{adds}, P, a_{pre}) \in CS$ wird durch den Algorithmus eine Beziehung $a_{adds} < a_{pre}$ in OS erzeugt. Die partielle Ordnung der Aktionsinstanzen wird also zum Teil durch die kausalen Abhängigkeiten zwischen den Aktionen definiert.

Suche im Planraum

Partielle Pläne

Eine Instanz einer Aktion a gefährdet den kausalen Link (a_{adds}, P, a_{pre}) des partiellen Plans (AS, OS, CS), wenn $\neg P$ in der *effects*-Liste von a enthalten ist.

Ein partieller Plan (AS, OS, CS) ist sicher, wenn für jede Instanz einer Aktion $a \in AS$, die einen kausalen Link $(a_{adds}, P, a_{pre}) \in CS$ gefährdet, gilt $a < a_{adds}$ oder $a_{pre} < a$ ist in OS.

Eine Agenda AG für einen partiellen Plan (AS, OS, CS) ist die Menge der Subgoals der Form (P, a) mit $a \in AS$ und

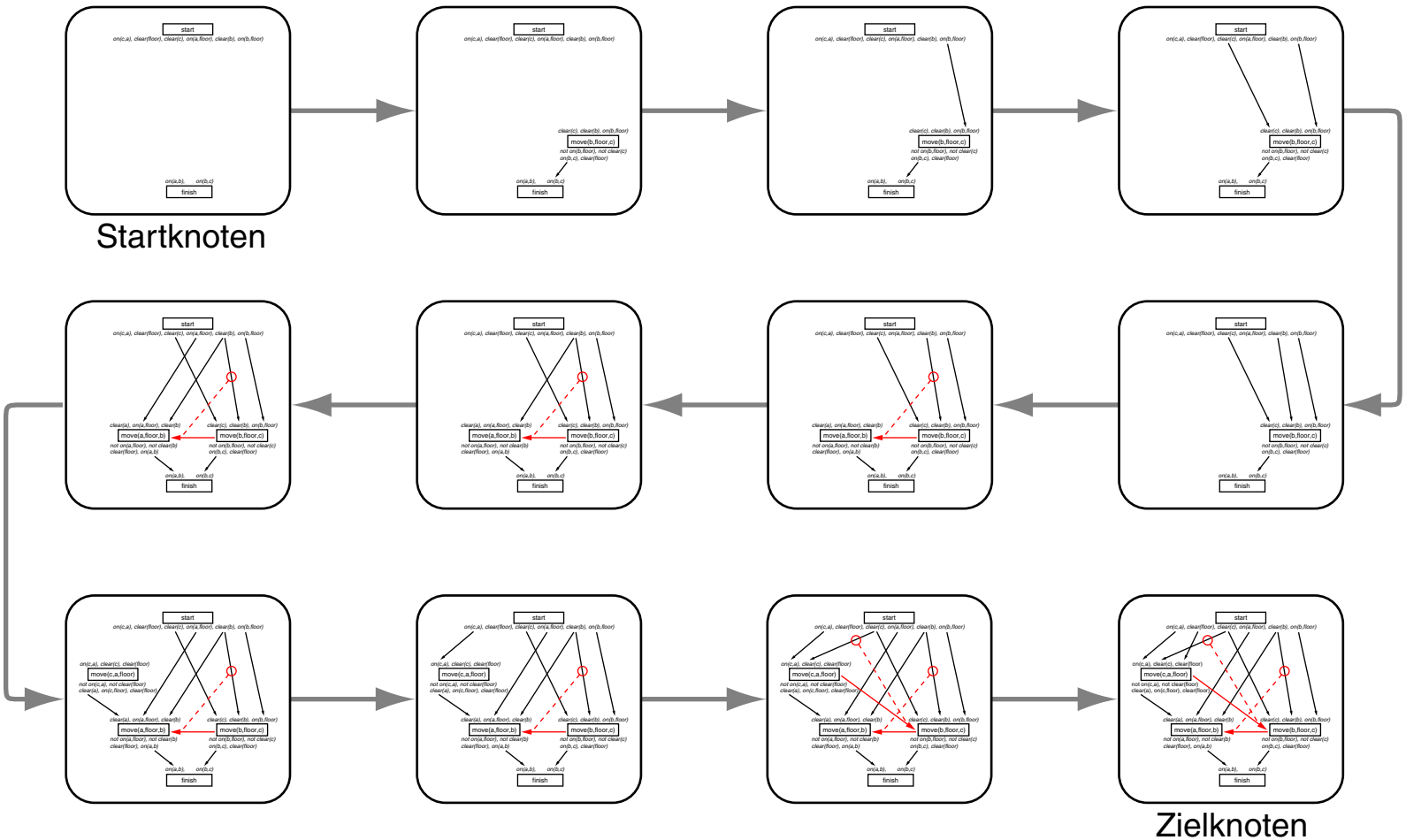
- P ist in der *precond*-Liste von a sowie
- kein kausaler Link (a', P, a) existiert in CS mit einem $a' \in AS$.

Ist die Agenda abgearbeitet, so nennen wir den erzeugten partiellen Plan (AS, OS, CS) vollständig, wenn er sicher ist. Ein vollständiger partieller Plan kann bereits durch (AS, OS) beschrieben werden.

Ein Plan ist konsistent, wenn OS konsistent ist, d.h. $OS \not\models a < a$ für jedes a.

Suche im Planraum

Beispiel: Lösungspfad im Planraum für Partial-Order Planning



Suche im Planraum

Partial-Order Planning: Erzeugung partieller Pläne

Wir beginnen den partiellen Plan mit den Aktionen *start* und *finish* in AS und der Relation *start* $<$ *finish* in OS. CL ist zu Anfang leer. Die Agenda besteht aus den zu erreichenden Zielen als Vorbedingung für *finish*.

Der Anfangsplan wird schrittweise erweitert:

- Wähle ein Subgoal (P, a) aus der Agenda und lösche es aus der Agenda.
- Wähle eine Instanz einer Aktion a' mit P in der *effects*-Liste.
- Füge $a' < a$ der Ordnungsrelation OS hinzu.
- Füge den kausalen Link (a', P, a) zu CS hinzu.
- Wenn a' einen kausalen Link $(a_1, Q, a_2) \in CS$ gefährdet oder eine Instanz einer Aktion $a'' \in AS$ den Link (a', P, a) gefährdet, füge $a' < a_1$ bzw. $a'' < a'$ (Demotion) oder $a_2 < a'$ bzw. $a < a''$ (Promotion) zu OS hinzu.
- Wenn a' noch nicht in AS enthalten ist, füge a' zu AS hinzu und füge für alle Elemente der *precond*-Liste entsprechende Subgoals zur Agenda hinzu.

Nichtdeterminismus: Der Algorithmus wählt Instanzen von Aktionen für Elemente der Agenda und Anordnungen von Aktionen, die gefährdete kausale Links sichern.

Bemerkungen:

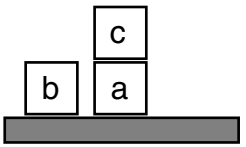
- Die ausgewählte Instanz einer Aktion a' zur Erzeugung des Zieles P kann bereits in AS enthalten sein.

Suche im Planraum

Beispiel: Partial-Order Planning für Sussman's Beispiel

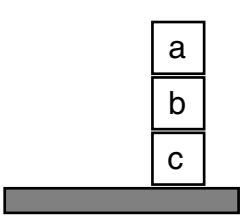
start

on(c,a), clear(floor), clear(c), on(a,floor), clear(b), on(b,floor)



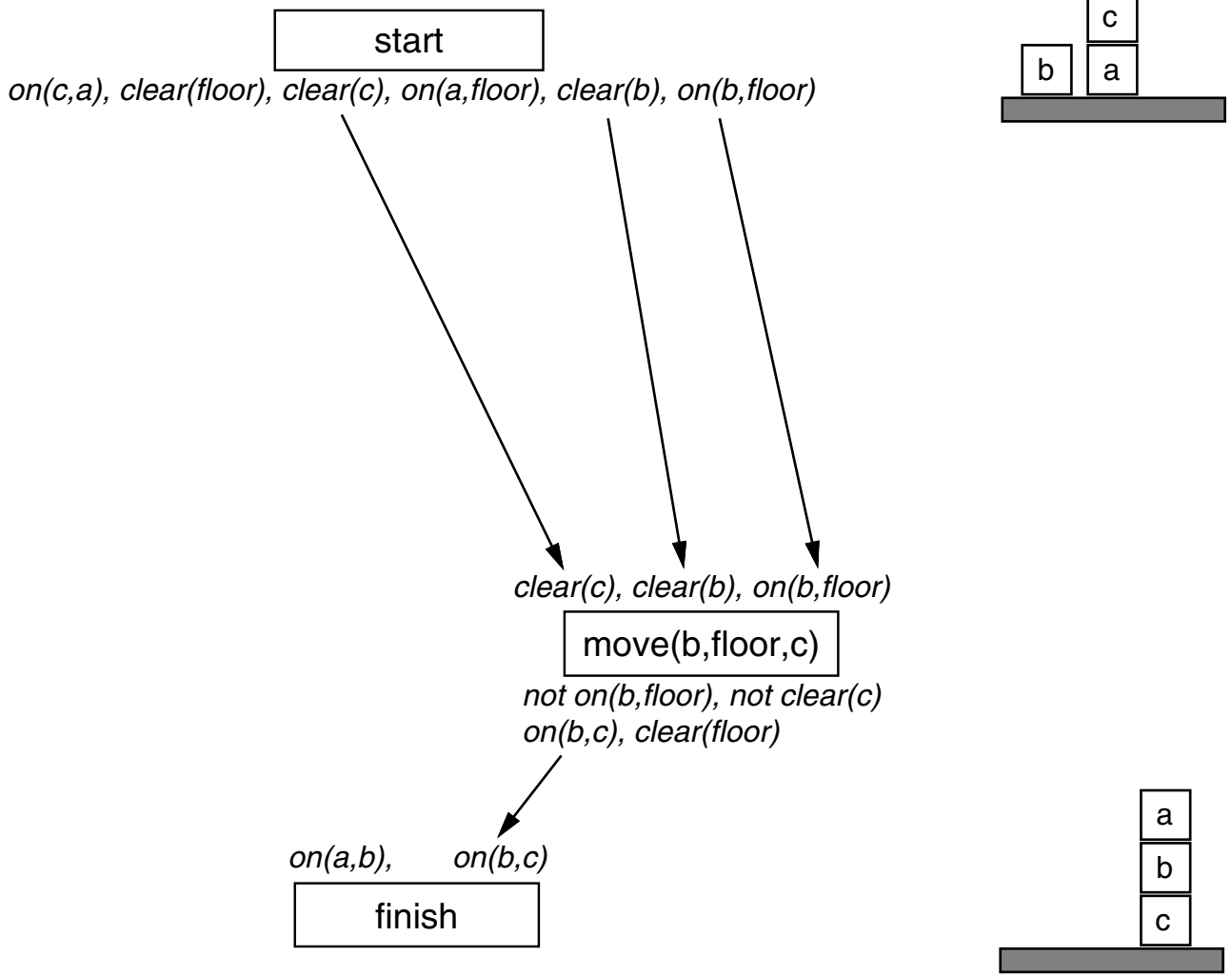
on(a,b), *on(b,c)*

finish



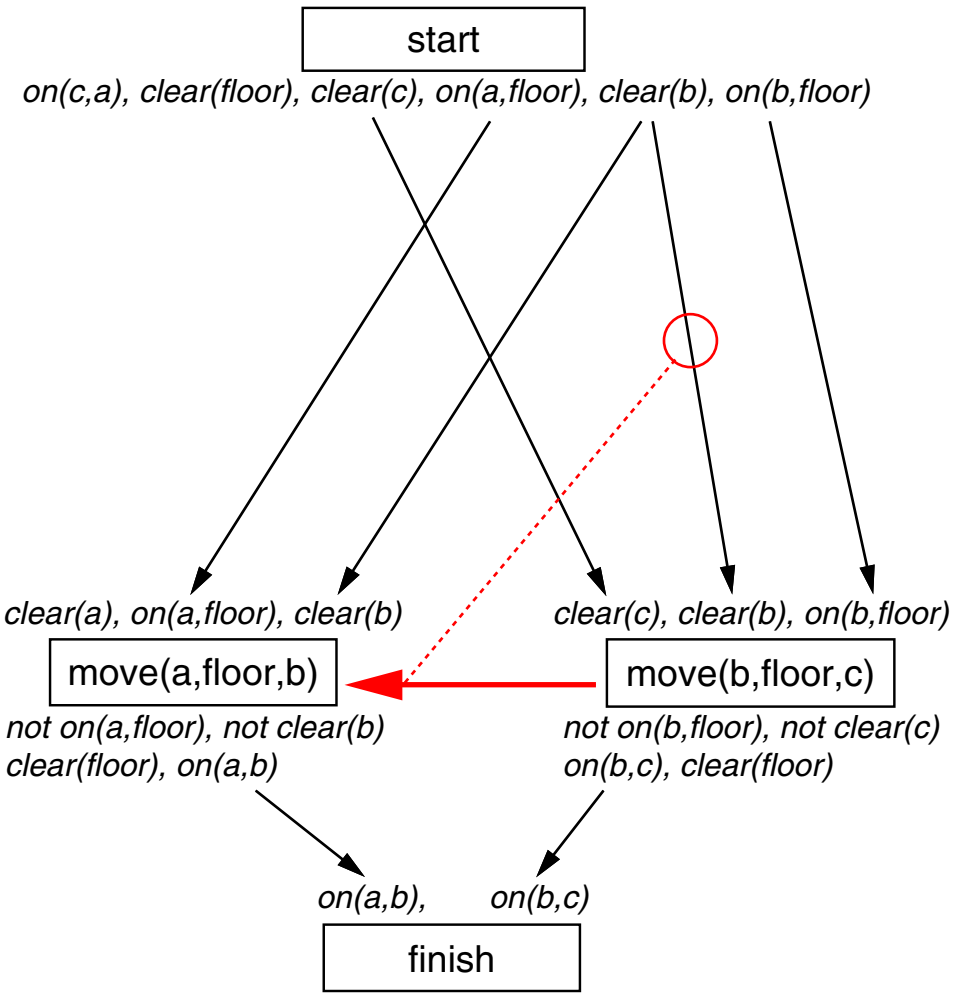
Suche im Planraum

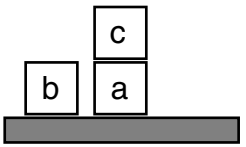
Beispiel: Partial-Order Planning für Sussman's Beispiel (Fortsetzung)



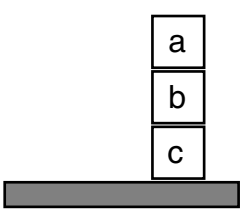
Suche im Planraum

Beispiel: Partial-Order Planning für Sussman's Beispiel (Fortsetzung)



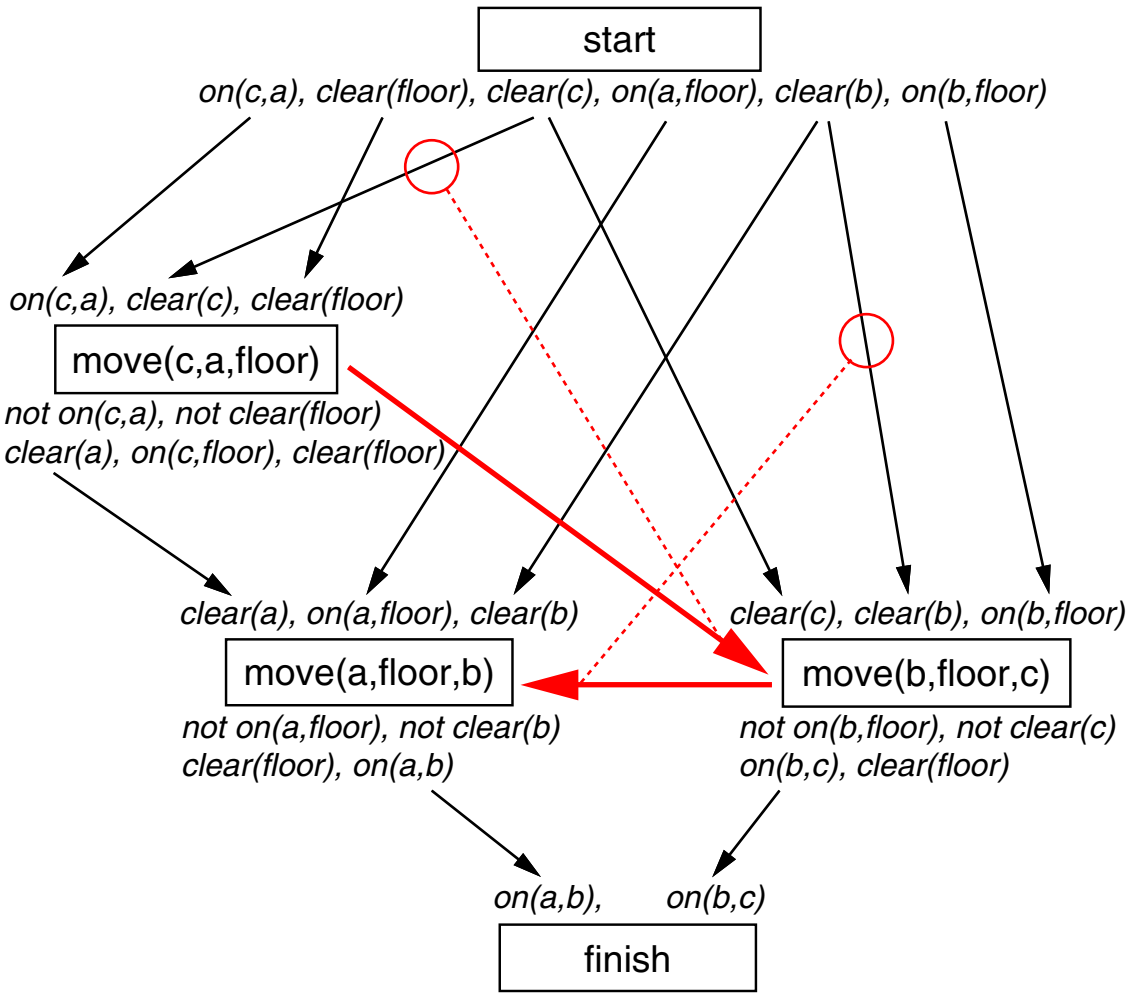


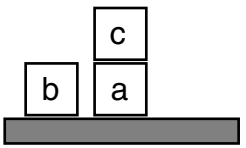
move(a,floor,b)
zerstört *clear(b)*
=> Anordnung nach
move(b,floor,c)



Suche im Planraum

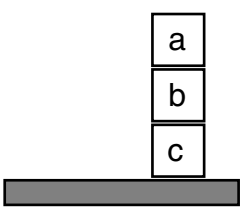
Beispiel: Partial-Order Planning für Sussman's Beispiel (Fortsetzung)





move(a,floor,b)
zerstört *clear(b)*
=> Anordnung nach
move(b,floor,c)

move(b,floor,c)
zerstört *clear(c)*
=> Anordnung nach
move(c,a,floor)



Bemerkungen:

- Die Modellierung der Blocks World erfolgt in der auf Seite **??** vorgestellten Weise. Beachten Sie die Bemerkungen dort zum Effekt $(\neg)$*clear*$($*floor*$)$ in z.B. der Aktion *move*$(c, a,$ *floor*$)$.

Neuere Algorithmen

Weitere Planungsverfahren

- GraphPlan (Blum & Furst 97)
  - Konstruieren und Verwalten eines Planungsgraphen
  - Verringerung des Suchaufwandes durch Information aus Planungsgraphen
- SatPlan (Kautz & Selman 92)
  - Übersetzung der Planungsaufgabe in aussagenlogische Problem
  - Anwendung eines effizienten SAT-Solvers zur Lösung