Zustandsbasiertes Planen und STRIPS

Seminararbeit

von

Michael Räther

Angewandte Informatik (BA)

Januar 2007

Betreuer:

Prof. Dr. Schmid

Universität Bamberg

Fakultät Wirtschaftsinformatik / Angewandte Informatik
Kognitive Systeme

Inhaltsübersicht

1. Einleitung

Das Lösen von Problemstellungen ist eine klassische menschliche Disziplin. Tagtäglich werden wir mit immer neuen Problemen konfrontiert. Der technische Fortschritt hat uns inzwischen Möglichkeiten geschaffen, selbst schwierige Probleme maschinell lösbar zu machen. In der realen Welt verwenden wir meist feste Regeln und Schemata um nicht für jedes Problem eine Lösung suchen zu müssen. Wir bedienen uns hierbei also meist vorhandenen Wissen. Werden wir jedoch mit Problemstellungen konfrontiert, über deren Lösung wir keine Kenntnisse besitzen, müssen wir uns einen Plan zur Lösung des Problems erarbeiten. Hierbei merkt man schnell, dass ohne eine Auseinandersetzung mit den relevanten Tatsachen keine nützlichen Ergebnisse erzielt werden können. Will man Probleme gemeinsam mit anderen Individuen lösen wird außerdem deutlich, dass alle Beteiligten als Basis eine gemeinsame Ontologie benötigen, um die Semantik der Tatbestände zu vereinheitlichen. Dadurch wird vermieden, dass Individuen die Bedeutung mancher Tatsachen in anderen Kontext sehen und somit eventuell andere Schlüsse ziehen.

Wie auch in der realen Welt benötigt auch ein computergestützter Planer eine feste Definition der Tatsachen. Diese Darstellung der Welt wird als Repräsentation bezeichnet und bildet die Basis für automatisierte Lösungsverfahren. Es handelt sich hierbei um einen Formalismus der alle für das Problem relevanten Informationen liefert.

Werden wir mit Problemen aus uns unbekannten Sachgebieten konfrontiert neigen wir dazu, die richtige Lösung dadurch zu finden, indem wir alle Möglichkeiten ausprobieren. Dieses Prinzip ist in der Informatik als *Trial and error* bekannt und findet auch Einsatz bei der automatisierten Planung. Durch das Testen aller vorhandenen Möglichkeiten mit allen vorhandenen Objekten wird ein Suchraum geschaffen, der alle somit erreichten Zustände abbildet und in Beziehung setzt. Kennzeichnet man den Ausgangszustand und den unweigerlich mit dem Problem verbundenen Zielzustand im Zustandsraum (englisch state-space), lassen sich viele Wege finden, wie man das Ziel erreichen kann. Alle diese Wege durch den Suchraum kennzeichnen eine mögliche Lösung für das Problem.

Diese Arbeit befasst sich zunächst mit verschiedenen Möglichkeiten der Repräsentation eines Problems im Zustandsraum. Anhand des linearen Planers STRIPS sollen dann die Stärken und Schwächen des Planens im Zustandsraum deutlich gemacht werden.

2. Mengen-theoretische Repräsentation

Die Darstellung bzw. Definition einer Mengen-theoretischen Welt wird Domäne ∑ genannt und hat als Basis eine Menge mit endlichen Symbolen L. Darin enthalten sind sämtliche Fakten die in der Welt gelten können. In der Regel wird die Semantik eines Faktum erst durch die Beschreibung in Worten vollends deutlich. Um die Faktenmenge L jedoch übersichtlich zu halten werden hier in der Regel nur Symbole als Elemente aufgenommen und die Bedeutung dieser Symbole im weiteren Verlauf der Domänendefinition erläutert.

2.1 Definition einer Mengen-theoretischen Domäne

Auf der Basis dieser Faktenmenge kann nun die Domäne als Zustandübergangssystem[1] definiert werden. Dazu werden weiterhin die möglichen Zustände der definierten Welt und die gültigen Operationen benötigt. Außerdem ist eine Zustandübergangsfunktion zu definieren, die das Vorgehen bei der Manipulation der Welt formal festlegt. Formal kann die Domäne somit folgendermaßen definiert werden:

$$\sum = (S, A, y)$$

S beschreibt die Menge von möglichen Zuständen. Ein Zustand s besteht aus einer Menge von Fakten die in diesem Zustand gültig sind. Alle nicht genannten Fakten sind im Zustand nicht gültig[2]. Alle Zustände s die nicht Teil der Menge S sind sind in der definierten Welt nicht gültig.

Die Menge A beschreibt die in der definierten Welt gültigen Aktionen. Eine Aktion a wird durch ein Tripel von Funktionen definiert:

precond (a)
effectsNegative (a)
effectsPositive (a)

Alle drei Funktionen liefern als Ergebnisse Mengen von Symbolen aus L. Für die Rückgabemengen von *effectsNegative (a)* und *effectsPositive (a)* gilt zusätzlich, dass die beiden Mengen zueinander disjunkt sind. Die Vereinigung der Symbole bzw. Fakten von *effectsNegative (a)* und *effectsPositive (a)* liefert also stets die leere Menge. Die Rückgabe der Funktion *precond (a)* gibt die Voraussetzungen an die erfüllt sein müssen, um die Aktion a auf einen Zustand s auszuführen. Ist diese Voraussetzung beim Ausgangszustand nicht gegeben, so kann die erwünschte Aktion nicht durchgeführt werden. Die Ergebnisse der Funktionen *effectsNegative (a)* und *effectsPositive (a)* kennzeichnen die eigentlichen Manipulationen der definierten Welt. Die drei Funktionen liefern stets eine Menge als Rückgabe und können somit auch jeweils mehrere Elemente enthalten. Aktionen stellen also Angaben über mögliche Manipulationen in der Welt zur Verfügung. Ihre genaue Verwendung ist in der Zustandübergangsfunktion, die einmalig für die Domäne definiert ist festgelegt.

1 state-transition-system
2 closed-world-assumption

Die Funktion y die beim Zustandsübergang verwendet wird benötigt als Parameter einen Ausgangszustand s und eine durchzuführende Aktion a. Anwendung der Funktion ist nur möglich, wenn die gegebene Aktion a auf den gegeben Zustand s anwendbar ist und a eine gültige Aktion für die definierte Welt darstellt. Die setzt voraus, dass der bei der Durchführung entstehende Folgezustand s_2 bereits in der Menge S enthalten ist. Die Zustandübergangsfunktion ist einmalig definiert und lautet in der Regel:

$$y(s,a) = (s - effectsNegativ(a)) \text{ UNION } effectsPositive(a)$$

Die in der Funktion verwendeten Operatoren sind Bestandteil der Mengenlehre und verdeutlichen den Ansatz der Mengen-theoretischen Repräsentation. Aus der Menge des Ausgangszustandes s sind zunächst die durch die Aktion a vorgegebenen negativen Effekte zu entfernen und dann die positiven Effekte hinzuzufügen. Die somit entstandene neue Faktenmenge beschreibt den Folgezustand s_{Folge}.

Die Definition der Problemdomäne ist also durch die Menge von möglichen Zuständen S, die Menge von gültigen Funktionen A und die einmalig für die Domäne definierte Zustandübergangsfunktion y definiert. Ein Planungsproblem benötigt eine Domäne, und einen Anfangs- und Endzustand. Der Anfangszustand s_0 ist Teil der in der Domäne definierten Zustandsmenge S. Der Zielzustand g gibt eine Menge von Fakten aus der in der Domäne definierten Faktenmenge L an, die gelten müssen, damit ein erreichter Zustand als Zielzustand klassifiziert werden kann.

Aufgrund des gegeben Startzustands s_0, den gegebenen Aktionen und der definierten Zustandübergangsfunktion können somit alle Folgezustände von s_0 gefunden werden. Für jeden neu erreichten Zustand werden dann erneut die Folgezustände ermittelt. Entsprechen die Fakten eines erreichten Zustands denen des gegebenen Zielzustands g, so kann der erreichte Zustand als Zielzustand klassifiziert werden und die Sequenz von Aktionen die zu diesen Zustand geführt haben zeigen einen gültigen Plan für das Problem.

```
L = { onground, onrobot, holding, at1, at2 }

Semantik der Fakten:

„onground“ bedeutetet der Container befindet sich auf den
Boden
„onrobot“ bedeutetet der Container befindet sich auf dem
Roboter
„holding“ bedeutetet der Container wird vom Kran in der
Luft gehalten
„at1“ bedeutet der Roboter befindet sich an Ort1
„at2“ bedeutet der Roboter befindet sich an Ort2

S = { s0 , ... , s5 }

wobei gilt::

s0 = { onground, at2 };
s1  = { holding, at2 };
s2  = { onground, at1 };
s3  = { holding, at1 };
s4   = { onrobot, at1 };
s5 = { onrobot, at2 }

A = { take, put, load, unload, move1, move2 }

wobei gilt:

take       = ( {onground},     {onground}, {holding}   )
put        = ( {holding},      {holding},  {onground}  )
load       = ( {holding, at1}, {holding},  {onrobot}   )
unload     = ( {onrobot, at2}, {onrobot},  {holding}   )
move1      = ( {at2},          {at2},      {at1}       )
move2      = ( {at1},          {at1},      {at2}       )
```

Abbildung 2.1: Mögliche Mengen-theoretische Repräsentation

In Abbildung 2.1 wird deutlich, dass die Voraussetzungen, negative sowie positive Effekte einer Aktion immer als eine Menge von Fakten definiert sind. Die Operation *load* benötigt zwei geltende Fakten, um ausgeführt werden zu können. Das Beispiel zeigt außerdem, dass die Mengen-theoretische Repräsentation sehr schnell zu komplexen Definitionen der Aktionen führen kann. Die Operationen *move1* und *move2* haben semantisch eigentlich nur eine Bedeutung, und zwar die Bewegung des Roboters von einen Ort zum anderen. Da jedoch die Repräsentation keine Verwendung von Variablen vorsieht, müssen hierfür zwei Aktionen definiert werden.

2.1 Problemlösung in einer Mengen-theoretischen Domäne

Nachdem die Domäne definiert wurde sind nun die relevanten Gegebenheiten der Welt bekannt und es ist nun möglich, Problemstellungen in dieser definierten Welt zu betrachten. Bei Problemen geht es darum, die Welt so zu manipulieren, dass die Gegebenheiten der Welt einen erstrebten Ziel entsprechen. Wir benötigen somit

einen Anfangszustand s_0 und einen Endzustand g.

$$P = (\sum, s_0, g)$$

Nun kann durch die Verwendung von möglichen Operationen auf Grundlage der in ∑ definierten Zustandübergangsfunktion y der Anfangszustand s_0 immer weiter manipuliert werden, bis irgendein Folgezustand die Fakten des Zielzustands g erfüllt. Dieses Vorgehen wird als Vorwärtssuche (Progression) bezeichnet und in Kapitel 4 näher erläutert.

Die dadurch entstehenden Zusammenhänge zwischen Zuständen, Aktionen und den resultierenden Folgezuständen lassen sich am Besten in einer Baumstruktur abbilden. Die Wurzel dieses Suchbaums bildet hierbei der initiale Zustand s_0. Die Endpunkte des Baums bilden die Zustände, die die Bedingungen die durch g gegeben sind erfüllen. Aufgrund Definition ist festgelegt das jeder Zustand, der die Fakten des Endzustands g erfüllt als Zielzustand zu klassifizieren ist:

$$s_{Folge} = \text{Zielzustand wenn g istTeilmenge } s_{Folge}$$

Da mehrere Zustände diese Forderung erfüllen können, kann der Suchbaum durchaus mehrere Endpunkte haben, die einer Lösung des Problems entsprechen.

Alle Sequenzen an Operationen die zu einen, als Zielzustand klassifizierten Folgezustand führen werden als Lösung bezeichnet. Es handelt sich hierbei um einen gültigen Plan für das zugrunde liegende Planungsproblem. Der Suchbaum macht deutlich, dass bei den meisten Problemen unterschiedliche Sequenzen von Aktionen zu einen Ziel führen können. Es können also mehrere gültige Pläne für das Lösen eines Problems existieren. Sind mehrere Lösungen verfügbar ist eine Klassifizierung nach ihrer Güte erforderlich, um eine gute Lösung von einer schlechten unterscheiden zu können. Die Länge der Sequenz ihrer Aktionen dient hierbei als Kriterium der Einteilung in:

- minimale Lösung
- maximale Losung

Eine minimale Lösung ist die Sequenz, die die wenigsten Aktionen benötigt, um das Ziel zu erreichen. Dies muss nicht heißen, dass eine minimale Lösung auch die geringste Zeit in Anspruch nimmt. Hierzu wäre eine zeitliche Klassifizierung aller verfügbaren Aktionen nötig. Geht man jedoch davon aus, dass ein computergestützter Planer pro Takt immer genau eine Aktion durchführen kann, so würde eine minimale Lösung auch weniger Takte und somit auch Zeit benötigen, um das erstrebte Ziel zu erreichen. Bei der maximalen Lösung ist die Anzahl der benötigten Aktionen verglichen zu den anderen gültigen Lösungen maximiert. Diese Klassifizierung ist also nur möglich, wenn andere Lösungen existieren, die das Ziel durch die Ausführung von weniger Operationen erreichen. Eine Maximallösung stellt somit die denkbar schlechteste Alternative zum Lösen eines Problems dar. Sollte kein gültiger Plan vorliegen, so ist das Problem in der entsprechenden Welt unlösbar.

Um eine kompakte Repräsentation zu gewährleisten ist es hinderlich, dass die

Definition eines Problems eine komplette Domäne benötigt, da eine vollständige Definition einer Domäne auch die Definition aller gültigen Zustände impliziert. Bei komplexen Problemstellungen führt dies schnell zu Unübersichtlichkeit bei der Domänenfestlegung. Diese Schwierigkeit kann umgangen werden, indem das Problem nicht die vollständige Domäne ∑, sondern nur die darin enthaltenen Aktionen als Basis besitzt. Ausgehend vom Startzustand s_0 sind also alle durch Aktionen entstehenden Folgezustände gültig und können wiederum weiter expandiert werden. Bei ungenauer Definition der Operationen kann dieses Vorgehen jedoch zu inkonsistenten oder aussageschwachen Zustände führen. Ein inkonsistenter Zustand beinhaltet Fakten, die sich gegenseitig widersprechen.

```
s_inkonsitent     =        { holding, at2, onrobot }
s_aussageschwach  =        { holding }
```

Abbildung 2.2: inkonsistenter und aussageschwacher Zustand

Die Abbildung 2.2 zeigt einen Zustand, in dem der Container vom Kran gehalten wird und sich zudem auf dem Roboter befindet. Die semantische Bedeutung der in Abbildung 2.1 definierten Fakten macht deutlich, dass ein solcher Zustand sich widerspricht und somit nicht erwünscht ist. Der aussageschwache Zustand $s_{aussageschwach}$ gibt keinen Aufschluss über die Position des Roboters. Ein Expandieren dieses Zustands ist durch Anwendung der Operation *put* (Definition siehe Abbildung 2.1) zwar möglich, jedoch würde dadurch wiederum ein aussageschwacher Zustand entstehen. Es hängt von den in g definierten Zielbedingungen ab, ob $s_{aussageschwach}$ dennoch zu einen als Zielzustand klassifizierbaren Zustand expandiert werden kann. Beide Zustände wären für die in Abbildung 2.1 definierte Domäne zwar ungültig, weil dort alle erreichbaren Zustände in der Menge S definiert sind, könnten jedoch entstehen, wenn L undefiniert bliebe. Einziger Ansatzpunkt um diese Problem zu vermeiden wäre dann nur eine vollständig korrekte Definition der Operationen.

3. Klassische Repräsentation

Die Klassische Repräsentation ermöglicht den Einsatz von Variablen und macht somit vor allem bei der Definition der Operatoren verglichen zum Mengen-theoretischen Ansatz eine kompaktere Definition möglich. Somit ist die Klassische Repräsentation für die Darstellung von komplexen Welten weit besser geeignet.

3.1 Definition einer Klassischen Domäne

Als Basis für die Definition der Domäne dient auch beim Klassischen Ansatz eine Faktenmenge. Die Semantik der Fakten ist vergleichbar zur Mengen-theoretischen Repräsentation, jedoch wurde die Darstellung insofern erweitert, dass die Fakten nun durch instantiierte Prädikate dargestellt werden. Für die Instantiierung werden konstante Symbole benötigt, die in der Menge C zusammengefasst sind. Die Menge P stellt die definierten Prädikate bereit. Bei einem Prädikat handelt es sich um eine

Funktion die als Rückgabe den Datentyp Boolean liefert. Jede Instanz eines Prädikats kann also unterschiedliche Wahrheitswerte, also *true* oder *false* liefern. Das Ziel der Manipulation besteht nun darin, die Wahrheitswerte der Prädikate zu ändern. Die Werkzeuge für die Manipulation, die Operatoren, sind außerdem variabel definiert. Dies bedeutet ein Operator der die Aktion *bewege Container von Ort zu Ort* ausführt muss nun nur noch einmalig definiert ist und mit jeglichen verfügbaren Objekten des Typs *Container* und *Ort* instantiiert und ausgeführt werden kann. Dies setzt voraus, dass man jeder Konstante einen eindeutigen Typ zuweisen können muss, damit bei jeder Operatorinstanz Typsicherheit gegeben ist. In der klassischen Repräsentation behält auch die *closed-world-assumption* ihre Gültigkeit. Jede Prädikatinstanz die in einem Zustand nicht entweder als wahr oder falsch verzeichnet ist, gilt als Faktum das im gegebenen Zustand nicht gültig ist.

Die Operatoren die für die Manipulation der Welt zuständig sind lassen sich ähnlich wie bei der Mengen-theoretischen Repräsentation auch nach einen festgelegten Schema erzeugen:

o = (name(o), precond(o), effects(o))

Ein Planungsoperator o besitzt somit einen eindeutigen Namen und setzt eine Voraussetzung (= *precondition*) voraus um ausgeführt werden zu können. Die eigentlichen Anweisungen zur Manipulation werden nun nicht mehr in negative und positive Effekte unterteilt, sondern als Literale unter *effects(o)* zusammengefasst. Die Instantiierung der Prädikate in den *preconditions* und *effects* wird durch die Eingabeparameter bestimmt. Um die Typsicherheit zu gewährleisten müssen entweder alle verwendeten Variablen einen Typ zugewiesen werden, oder jeder Operator muss eine Typbeschreibung beinhalten.

```
buildStargate ( p, m )
;; erstellt ein Sternentor von Planet m nach
   Planet m, kann nur ausgeführt werden wenn
   nötige Materialien aluminium zum Bau auf
   Planet m verfügbar sind
precond: stored(p,aluminium)
effects: ¬stored(p,aluminium), adjacent(p,m)
```

Abbildung 3.1: ein möglicher Operator in einer klassischen Planungsdomäne

```
buildStargate ( earth, mars )
precond: stored(earth,aluminium)
effects: ¬stored(earth,aluminium),
           adjacent(earth,mars)
```

Abbildung 3.2: eine mögliche Instantiierung des Operators aus Abbildung 3.1

Das in Abbildung 3.1 gezeigte Beispiel beinhaltet die nötige Typbeschreibung um die Prädikate typsicher instantiieren zu können. Neben Variablen wird auch die Konstante *aluminium* verwendet. Abbildung 3.2 zeigt **eine** mögliche Instantiierung mit Objekten vom Typ Planeten *earth* und *mars*. Sind mehr als zwei Objekte vom Typ definiert lässt sich der Operator *buildStargate* mit allen definierten Planeten durchführen. Will man die Möglichkeit voraussetzen, alle Planeten über ein Sternentor verbinden zu können, müsste man bei der Mengen-theoretischen

Repräsentation bei fünf verfügbaren Planeten zehn mal einen Operator definieren, der das Sternentor verwirklicht.

Der Zustandsübergang wird durch die einmalig definierte Zustandübergansfunktion definiert. Diese gleicht der bereits bei der Mengen-theoretischen Darstellung verwendeten Funktion und macht es möglich, Folgezustände durch Mengenalgebra zu erzeugen. Durch die nötige Instantiierung der Operatoren ist sichergestellt, dass die Prädikate der Folgezustände keine Variablen beinhalten, wodurch die Fakten jedes Zustands unmissverständlich festgelegt sind.

Eine Domäne benötigt wie beim Mengen-theoretischen Ansatz auch eine Zustandsmenge S, die Menge der erlaubten Operationen A (entspricht hier der Menge aller möglichen Operationen O mit allen möglichen Instantiierungen) und die einmalig definierte Zustandübergansfunktion y.
Ein Problem wird ebenfalls wie bei der Mengen-theoretischen Repräsentation formuliert:

$$P = (O, s_0, g)$$

3.2 Erweiterungen der Klassischen Repräsentation

Das Interesse komplexe Problemstellungen kompakt mit der klassischen Darstellung erfassen zu können machte es nötig, Erweiterungen zu entwickeln, die das zugrunde liegende Zustandübergangssystem nicht verletzen. Da die Mengenalgebra die Basis für einen Zustandsübergang ist, musste man sich somit auf die durch die Mengenlehre verfügbaren Möglichkeiten beschränken. Einige entscheidende Erweiterungen die Möglichkeiten der klassischen Repräsentation enorm steigern können sind:

- Operatoren mit Bedingungen
- quantifizierte Ausdrücke
- disjunkte Voraussetzungen

Das Einbringen von Bedingungen in die Operatoren bedeutet, dass wenn ein Operator aufgrund der erfüllten Vorbedingungen ausgeführt wird, sich manche Effekte nur äußern, wenn hierfür wiederum Vorbedingungen erfüllt sind. Wie Abbildung 3.3 zeigt werden diese weiteren Vorbedingungen durch den englischen Ausdruck *when* gekennzeichnet und die bei Erfüllung eintretenden Effekte werden mit *then* eingeleitet. Mit der Verwendung von quantifizierten Ausdrücken ist es möglich, Effekte unabhängig der Instantiierung für alle Objekte des gleichen Typs zu initiieren. Der Allquantor wird mit *forall* und einer folgenden Variable des gewünschten Typs definiert. Für alle Objekte die die *when*-Bedingung erfüllen werden dann die nach *then* definierten Effekte durchgeführt. In Abbildung 3.4 würden demnach **alle** Raumschiffe die sich im Orbit befinden direkt zum nun neu erreichbaren Planeten katapultiert. Quantifizierte Ausdrücke auch helfen, den Initialzustand oder die Zielbedingung kompakt zu definieren.

```
buildStargate ( p, m, s )
;; erstellt ein Sternentor von Planet m nach
   Planet m, kann nur ausgeführt werden wenn
   nötige Materialien aluminium zum Bau auf
   Planet m verfügbar sind; befindet sich ein
   Raumschiff s im Orbit von Planet p, wird
   dieses direkt zum Planeten m katapultiert
precond: stored(p,aluminium)
effects: ¬stored(p,aluminium), adjacent(p,m),
         ( when at(s,p) then at(s,m), ¬at(s,p) )
```

Abbildung 3.3: konditionale Erweiterung des Operators aus Abbildung 3.2

```
buildStargate ( p, m, s )
;; erstellt ein Sternentor von Planet m nach
   Planet m, kann nur ausgeführt werden wenn
   nötige Materialien aluminium zum Bau auf
   Planet m verfügbar sind; befinden sich
   Raumschiffe s im Orbit von Planet p, werden
   diese alle direkt zum Planeten m katapultiert
precond: stored(p,aluminium)
effects: ¬stored(p,aluminium), adjacent(p,m),
         ( forall s when at(s,p) then at(s,m),
                                      ¬at(s,p) )
```

Abbildung 3.4: Operator aus Abbildung 3.3 mit Allquantor

Der Gebrauch von disjunkten Voraussetzungen äußert sich in der Verwendung von aus der Mengenlehre bekannten Symbolen für Konjunktion und Disjunktion in den *preconditions*. Er ermöglicht die logische Verknüpfung verschiedener Vorbedingungen.

Bei der Verwendung der Erweiterungen sind wie in der Programmierung (foreach-, if-Schleifen) beliebige Kapselungen möglich. Da die Operationen dadurch sehr schnell unüberschaubar werden können, sollten bei Verwendung von Erweiterungen stets der semantische Hintergrund der Aussage bedacht werden. Feststehende Relationen können hierbei als Kontrollfunktion dienen.

4. STRIPS

STRIPS ist ein linearer Planer der 1971 von Fikes und Nilson zur Steuerung eines Roboters entwickelt wurde. Als Grundlage dient die Klassische Repräsentation mit Erweiterungen. Um den Algorithmus der bei der Zielsuche verwendet wird besser zu verstehen, müssen zunächst die beiden grundsätzlichen Alternativen betrachtet werden, um Lösungen bzw. Pläne im Zustandsraum zu finden. Hierbei handelt es sich also um Algorithmen, die den Zustandsbaum expandieren.

4.1 Allgemeine Suchalgorithmen

Die eigentlichen Mechanismen um gültige Pläne zu evaluieren stellen die unterschiedlichen Suchalgorithmen dar. Ihre Aufgabe ist es, in einem Durchlauf von einen gegebenen Zustand die möglichen ausführbaren Operationen (Betrachtung der *preconditions*) zu ermitteln und die durch deren Anwendung entstehenden Folgezustände zu erstellen. Die Algorithmen verwenden iterative oder rekursive Techniken, um nach Auffindung von Folgezuständen erneute Durchläufe auf die neu entstandenen Zustände zu starten. Möchte man Probleme bei der Suche vermeiden oder die Suche optimieren bietet sich die Möglichkeit, den zugrunde liegenden Suchalgorithmus anzupassen. Eine nützliche Anpassung hierbei ist eine Erweiterung die Schleifen erkennt und vermeidet.

Eine Schleife entsteht durch die Ausführung einer Aktion auf einen Folgezustand, die als Ergebnis wieder den Eingangszustand liefert. Wenn es die Definition der Operationen zulässt, können sogar Sequenzen entstehen, die immer wieder die gleichen Zustände liefern. Hierbei spricht man von Endlosschleifen die nie zu einer Lösung führen können. Möchte man diese Probleme vermeiden, muss man Änderungen im Suchalgorithmus vornehmen. Bereits abgearbeitete Zustände müssen gespeichert werden und alle neu entstandene Zustände dürfen erst persistiert werden, wenn sichergestellt ist, dass nicht bereits ein gleicher Zustand gespeichert ist. Entspricht ein entstandener Folgezustand einen Zustand, der bereits besucht wurde liegt eine Schleife vor und die Ausführung der Aktion die den bereits vorliegenden Zustand erzeugt wird nicht durchgeführt.

Bei den beiden folglich vorgestellten Algorithmen zur Vorwärtssuche und Rückwärtssuche sind derlei Verbesserungen nicht integriert.

4.1.1 Vorwärtssuche (Progression)

Die Vorwärtssuche ist der einfachste, weil menschlich intuitivste Planungsalgorithmus zum Finden von Lösungen in einer definierten Problemstellung. Die Progression ist ein nicht-deterministisches Verfahren. Diese Aussage bezieht sich auf die Wahl der Aktion die in einer Rekursion für die Expansion verwendet wird. Gibt es also für einen Zustand mehrere mögliche Aktionen, so entscheidet der Algorithmus welche Operation er ausführt. Dies bedeutet letztendlich, dass keine Schleifenerkennung und Vermeidung integriert ist. Der Algorithmus basiert auf der Klassischen Repräsentation.

```
Forward-search (O, s0, g)
     s ← s0
     π ← the empty plan
     loop
          if s satisfies g then return π
          applicable ← { a | a is a ground instance of an
                             operator in O, and
                             precond(a) is true in s }
          if applicable = emptySet then return failure
          nondeterministically choose an action a isElem
                    applicable
          s ← y(s,a)
          π ← π.a
```

Abbildung 4.1: Progressionsalgorithmus als Schleife

Falls der Starzustand nicht bereits die Zielbedingungen erfüllt wird zunächst die Menge der aufgrund ihrer passenden Vorbedingung ausführbaren Aktionen erstellt. Zunächst müssen also sämtliche Operatoren instantiiert werden. Dann wird zufallsbedingt irgendeine dieser gültigen Aktionen ausgeführt und der neue Zustand wird erstellt. Außerdem wird die verwendete Aktion zu der Sequenz hinzugefügt und ist somit als ein Teil des Plans gespeichert. Die Schleife (oder alternativ Rekursion) wird nun mit den neu entstandenen Zustand durchlaufen. Zu Beginn des Durchlaufs muss geprüft werden, ob der aktuell betrachtete Zustand den Bedingungen des Zielzustands g entspricht. Trifft dies zu wird der bisher evaluierte Plan, also die bisherige Aktionssequenz zurückgegeben. Aufgrund der nicht-deterministischen Wahl der Aktionen liefern andere Ausführungsstränge des Algorithmus andere Lösungen für das Problem, sofern sie nicht aufgrund einer Schleife nicht terminieren.

4.1.2 Rückwärtssuche (Regression)

Bei der Rückwärtssuche dient der gegebene Endzustand als Anfangspunkt für den Algorithmus. Der Vorteil liegt hierbei darin, dass der Zielzustand in der Regel weniger Literale enthält als der Anfangszustand und somit weniger Schritte durchgeführt werden müssen.

```
Backward-search (O, s0, g)
     π ← the empty plan
     loop
          if s0 satisfies g then return π
          relevant ← { a | a is a ground instance of an
                           operator in O that is relevant
                           for g }
          if relevant = emptySet then return failure
          nondeterministically choose an action a isElem
                    applicable
          π ← a.π
          g ← y-1(g,a)
```

Abbildung 4.2: Regressionsalgorithmus als Schleife

4.2 STRIPS-Algorithmus

Das Hauptproblem der in 4.1 gezeigten Algorithmen war die Größe des Suchraums. Dies liegt daran, dass keine Heuristik eingebaut ist, die die Expansion von Zuständen welche sicherlich nicht zu einer Lösung beitragen können verhindert. Der STRIPS-Algorithmus ist ein angepasster Regressionsalgorithmus der Techniken beinhaltet, um dieses Problem zu minimieren. Im Gegensatz zu den vorgestellten Algorithmen zur Vorwärts- und Rückwärtssuche ist der STRIPS-Algorithmus jedoch nicht vollständig. Dies bedeutet, dass nicht alle möglichen Lösungen gefunden werden können.

```
Ground-STRIPS (O, s0, g)
π ← the empty plan
      loop
            if s0 satisfies g then return π
            A ← { a | a is a ground instance of an operator
                              in O, and a is relevant for g }
            if A = emptySet then return failure
            nondeterministically choose an action a isElem A
            π‘ ← Ground-STRIPS (O, s, precond(a))
            if π‘ = failure then return failure
            ;; if we get here, then π‘ achieves precond(a) from s
            s ← y(s, π‘)
            ;; s now satisfies precond(a)
            s ← y(s,a)
            π ← π. π‘.a
```

Abbildung 4.3: STRIPS-Algorithmus

Der STRIPS-Algorithmus verwendet die Mittel-Zweck-Analyse[3] um geeignete Operatoren zu finden, bei denen sich ein Expansion lohnt. Dies bedeutet, dass in jeder Rekursion nur Teilziele verfolgt werden, die durch die Voraussetzungen des letzten ausgeführten Operators bestimmt werden. Erfüllt ein Zustand alle Voraussetzungen eines Operators, so wird dieser Operator ausgeführt und diese Bindung wird nicht wieder gelöst. Ein Teilziel ist hierbei eine Teilmenge der gesamten Zielmenge. Da es es sich um eine Regression handelt wird diese Menge durch den Anfangszustand s_0 definiert.

Ein Problem besteht darin, dass während der Evaluierung des Suchraums hin zu einen Teilziel, ein bereits erreichtes Teilziel wieder zerstört wird. Aufgrund der Tatsache, dass die meisten Operatoren mehrere Effekte auslösen kann dieses Problem leicht verdeutlicht werden. Während STRIPS einen Operator anwendet von denen er einen Effekt benötigt um sein momentan verfolgtes Teilziel zu erreichen, wird als Nebeneffekt ein weiterer im Operator definierter Effekt ausgeführt. Ein bekanntes Beispiel das dieses Problem näher verdeutlicht ist die *Sussman Anomaly*, ein triviales Problem aus der Blockswelt.

3 means-end-analysis

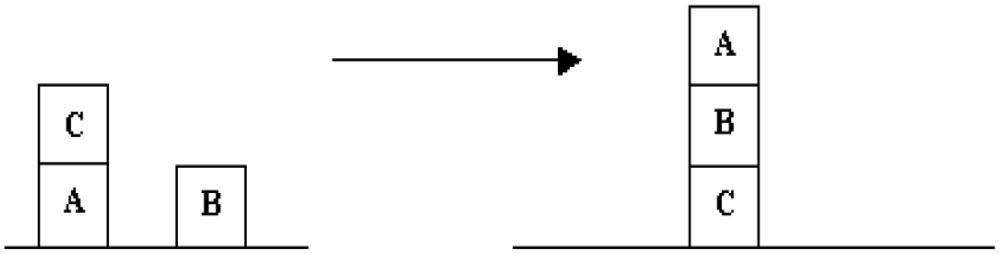

Abbildung 4.4: Initialzustand und Zielzustand der Sussman Anomaly

Eine Lösung dieser Problematik besteht im Zusammenführen der zwei Pläne für die Teilziele die unabhängig voneinander erreicht wurden. Bei zwei zu erreichenden Teilzielen evaluiert man zunächst die Aktionssequenz für das erste Teilziel und dann für das zweite. Die beiden Sequenzen werden nun zu einer vereinigt, indem sämtliche Aktionen aus $Sequenz_2$ an passender Stelle in $Sequenz_1$ eingefügt werden. Die so erhaltene Sequenz stellt eine Lösung dar, die beide Teilziele in einen Durchlauf erreicht.

5. Fazit

Die zustandsraumbasierte Suche von Lösungen war der erste Ansatz in der KI, um komplexere Problemstellungen zu lösen. Techniken um den Zustandsraum zu verkleinern, die Suche also zu optimieren wurden erst später bekannt. Während dieser Zeit wurden neue Ansätze entwickelt um die Suche im Zustandsraum zu verbessern. Moderne Algorithmen haben die Progression als Grundlage und werden stetig weiterentwickelt. STRIPS gilt somit schon seit einigen Jahren als veraltet, findet jedoch noch viel Gebrauch in der Forschung als verlässliche Methode neue Formeln zu überprüfen.

Literaturverzeichnis

[Fikes and Nilsson, (1971)] STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence, Vol. 2

[Russell and Norvig (2003)] Artificial Intelligence, A Modern Approach (2nd Ed.)

[Geffner (2000)] Functional Strips: a more flexible language for planning and problem solving. In Jack Minker (Ed.), Logic-Based Artificial Intelligence, Kluwer