Planen in der realen Welt

Einführung

- Bis jetzt: Planen durch Suche (A*, Tiefensuche, ...)
- Jetzt: Planen in komplexeren Umgebungen

Beispiel: Buchhandlung. Ein Agent soll Russel, KI kaufen. In einer Online Buchhandlung muss der Agent ALLE möglichen Aktionen (im Milliardenbereich) auswerten bis ein ihm sagt, dass er den gewünschten Erfolg hat. (Das Buch mit der richtigen ISBN Nummer hat). Ein sensibler Planungsagent kann von seinem Ziel haben(ISBN...) rückwärts darauf schließen, dass Kaufen (ISBN...) die richtige Aktion ist. (ANSCHREIBEN)

Inhaltsverzeichnis

- Sprache für Planungsprobleme (STRIPS)
- Partiell ordnende Planer
- Planen mit Ressourcen
- Hierarchisches Task Netzwerk Planen (HTNP)
- Planen in nicht deterministischen Domänen
  - vollständig beobachtbar
  - partiell beobachtbar
- Ausführungsüberwachung
- stetiges Planen
- Multi-Agenten-Planen

STRIPS

- steht für Stanford Research Institute Problem Solver
- Ist eine Sprache zur Repräsentation von Planungsproblemen
- Repräsentiert Zustände, Ziele und Aktionen

STRIPS - Zustände

• Werden für Planer in Konjunktionen positiver Literale erster Stufe dargestellt
• zum Beispiel:
• *Arm* $\wedge$ *Unbekannt* ist ein unglücklicher Agent
• *Bei(Flugzeug1, Melburne)* $\wedge$ *Bei(Flugzeug2, Sydney)* kann einen Teil eines Transportproblems beschreiben
• Funktionen in Literalen wie *Bei(x,y)* oder *Bei(Professor(KI), Köln)* sind nicht erlaubt
• Annahme der geschlossenen Welt: Alle Bedingungen die nicht erwähnt werden, werden als falsch angenommen

STRIPS - Ziele

- Sind eine spezielle Art Zustände
- Werden identisch dargestellt
- Ein Zustand erfüllt ein Ziel, wenn alle Atome des Ziels auch im Zustand Vorkommen
- zum Beispiel:
  - *reich* $\wedge$ *berühmt* $\wedge$ *unglücklich* erfüllt das Ziel *reich* $\wedge$ *berühmt*

Ziele sind wie Zustände Konjunktionen positiver Grundliterale

Punkt 3 ist wahr, da alle literale positiv sein müssen.

STRIPS - Aktionen

- Besteht aus Namen, Vorbedingungen und Effekten
- Beispiel für ein Aktionsschema:

Ein Flugzeug von einem Ort an einen andern zu fliegen

Aktion(Fliegen(p,von,nach))

PRECOND: Bei(p,von) ∧ Flugzeug(p) ∧ Flughafen(von) ∧ Flughafen(nach)

EFFECT: ¬Bei(p,von) ∧ Bei (p,nach)

PRECOND: p ist bei von, p ist ein Flugzeug, von hat einen Flughafen und vnach hat einen Flughafen

EFFECT: p ist nicht mehr in von dafür ist p in nach

STRIPS - Aktionen

- Der Aktionsname mit den Parametern *(Fliegen(p,von,nach))* dient der Identifikation der Aktion
- Die Vorbedingungen (*PRECOND*) geben an welche Bedingungen ein Zustand aufweise muss, damit die Aktion ausgeführt werden kann
- Der Effekt gibt die Änderungen der Zustände an. Hier können auch negative Literale auftreten. Jedes Literal was nicht im Effekt erwähnt wird, bleibt unverändert.

Partiell geordnete Pläne

- Um zu verstehen, was partiell geordnete Pläne von Vollständig geordneten unterscheidet, benutzen wir als Beispiel das Schuhe anziehen.
- Als Planungsproblem in STRIPS sieht das so aus:

Ziel(RechterSchuhAn ∧ LinkerSchuhAn)
Initiieren()
Aktion(RechterSchuh,PRECOND:RechteSockeAn,EFFECT: RechterSchuhAn)
Aktion(RechteSocke, EFFECT:RechteSockeAn)
Aktion(LinkerSchuh,PRECOND:LinkeSockeAn,EFFECT: LinkerSchuhAn)
Aktion(LinkeSocke, EFFECT:LinkeSockeAn)

In jedem Konkretenplanungsproblem gibt es noch Initiieren(), welches definiert wann eine Problem gelöst werden muss.

Ein Planer sollte jetzt die Aktionsfolge RechteSocke, RechterSchuh und LinkeSocke, LinkerSchuh finden um die beiden Teile der Zielkonjunktion zu erfüllen.

Bleibt die Reihenfolge in der die vier Aktionen ausgeführt werden. Ein vollständig ordnender Planer würde, um später auf den besten Plan schließen zu können, sämtliche Möglichkeiten (6 Stück) berechnen. Ein partiell ordnender Planer ist in der Lage eine Abfolge anzugeben, in der zwei oder mehr Aktionen nebeneinander stehen ohne, dass gegeben ist, welche zuerst ausgeführt wird.

Zur Anzeige wird der QuickTime™
Dekompressor „“
benötigt.

Der partielle Plan wird als Aktionsgraph und nicht als Folge dargestellt.

RUSSEL SEITE 482

Partiell geordnete Pläne

- Die schriftliche Darstellung eines POP besteht aus Aktionen, Ordnungsbedingungen und kausalen Verknüpfungen
- Im Schuhanziehbeispiel:

Aktionen: {*RechterSchuh, LinkerSchuh, RechteSocke, LinkeSocke, Start, Ende}*
Ordnungen: {*RechteSocke<RechterSchuh, Linke Socke<LinkerSchuh}*
Verknüpfungen: {*RechteSocke --RechteSockeAn-> RechterSchuh, LinkeSocke --LinkeSockeAn-> LinkerSchuh, RechterSchuh --RechterSchuhAn-> Ende, LinkerSchuh --LinkerSchuhAn-> Ende}*

Aktionen enthalten alle Schritte des Plans, sie stammen logischerweise aus der Menge der Aktionen im Planungsproblem plus Start und Ende

Ordnungsbedingungen sortierten Aktionen mit „<„. A<B heißt, dass A vor B aber nicht unbedingt exakt vor B ausgeführt werden muss.

Als Verknüpfung bedeutet *A --p-> B „A erzielt p für B". P ist ein Effekt von A und Vorbedingung für B. Außerdem gibt eine Verknüpfung an, dass zwischen der Ausführung von A und , p wahr bleiben muss. Eine Aktion C die zu ¬p führt, darf also nicht zwischen A und B erfolgen.*

Planen mit Ressourcen

- STRIPS basiert auf dem Situationskonzept (zB bekannt aus Kap. 10)
- Das führt dazu, dass man nicht festhalten kann, wie lange ein Aktion dauert, oder wann sie stattfindet.
- Offensichtlich ist dies jedoch bei realen Problemen wichtig (Job Shop Scheduling)

Job Shop Scheduling ist eine Anwenungsgruppe, die Probleme löst, in denen jede Aktion eine bestimmte Zeit dauert, ggf. Ressourcen benötigt.

Planen mit Ressourcen

- Beispiel, zwei Autos zusammenbauen:

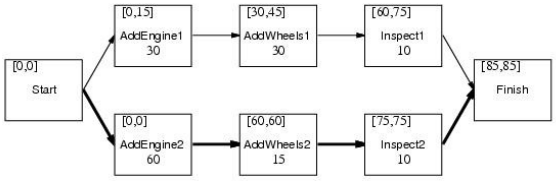
Init(Chassis(C1) ∧ Chassis(C2) ∧ Engine(E1,C1,30) ∧ Engine(E1,C2,60) ∧ Wheels(W1,C1,30) ∧ Wheels(W2,C2,15))
Goal(Done(C1) ∧ Done(C2))
Action(AddEngine(e,c,m)
PRECOND: *Engine(e,c,d) ∧ Chassis(c) ∧ ¬EngineIn(c)*
EFFECT: *EngineIn(c) ∧ Duration(d))*
Action(AddWheels(w,c)
PRECOND: *Wheels(w,c,d) ∧ Chassis(c)*
EFFECT: *WheelsOn(c) ∧ Duration(d))*
Action(Inspect(c)
PRECOND: *EngineIn(c) ∧ WheelsOn(c) ∧ Chassis(c)*
EFFECT: *Done(c) ∧ Duration(10))*

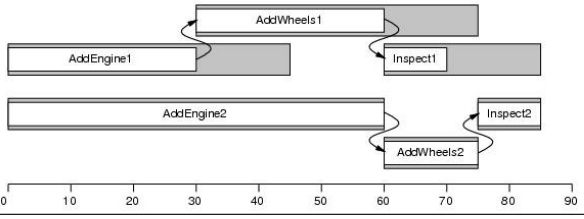
Das Auto besteht aus Chassis, Motor und Rädern. Der Motor muss vor den Rädern eingebaut werden. Zum Schluss wir das Ganze noch überprüft.

Zusätzlich zur bisherigen Notation, finden sich hier Zahlen im Ausgangszustand und Duration im EFFECT. Motor(E1,C1,30) bedeutet, dass Motor E1 in Chassis C1 gehört und 30 Minuten zum einbauen braucht. Duration gibt für jede Aktion an wie lange diese insgesamt dauert.

Planen mit Ressourcen

- Geben wir diese Problem einem POP, bekommen wir:





Oben der partiell geordnete Plan. In den Aktionen stehen die früheste (ES) und späteste (LS) Startzeit. Die Differenz aus beiden ist der Spielraum dieser Aktion. Der kritische Pfad, der für die Gesamtlänge des Plans verantwortlich ist hat immer 0 Spielraum (fett).

Unten der Gleiche Plane als Zeitgerade. Die freien grauen Rechtecke sind der Spielraum

Alle Zeiten werden ausgehen von ES Start = 0 mithilfe der duration Zeiten der Aktion berechnet.

SEITE 517

Planen mit Ressourcen

- Eine Komplexitätsstufe mehr: Ressourcen
- Jeder Motoreinbau benötigt die Motorwinde und es gibt nur zwei Mitarbeiter, die die Abschlusskontrolle durchführen können
- Erweiterung der Problemrepräsentation um RESOURCE: R(k)
- für eine Aktion werden k Einheitend der Resource R benötigt

Action(AddEngine(e,c,m)
PRECOND: *Engine(e,c,d)* $\wedge$ *Chassis(c)* $\wedge$ $\neg$*EngineIn(c)*
EFFECT: *EngineIn(c)* $\wedge$ *Duration(d))*
RESOURCE: *EngineHoist(1)*

Diese Erweiterung des Notation wird nur für wieder verwendbare Ressourcen wie Maschinen oder Personal benötigt.
Verbrauchsressourcen (Schrauben) können in der Standardnotation verarbeitet werden, da es ja einen Effekt auf die Menge gibt.

Planen mit Ressourcen

- Das komplette Problem:

Init(Chassis(C1) ∧ Chassis(C2) ∧ Engine(E1,C1,30) ∧ Engine(E1,C2,60) ∧ Wheels(W1,C1,30) ∧ Wheels(W2,C2,15) ∧ EngineHoists(1) ∧ WheelStations(1) ∧ Inspectors(2))

Goal(Done(C1) ∧ Done(C2))

Action(AddEngine(e,c,m)
PRECOND: *Engine(e,c,d) ∧ Chassis(c) ∧ ¬EngineIn(c)*
EFFECT: *EngineIn(c) ∧ Duration(d),*
RESOURCE*: EngineHoists(1))*

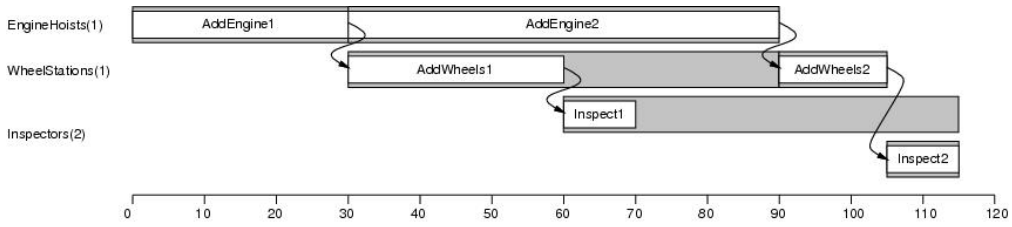
Action(AddWheels(w,c)
PRECOND: *Wheels(w,c,d) ∧ Chassis(c)*
EFFECT: *WheelsOn(c) ∧ Duration(d)*
RESOURCE*: WheelStations(1))*

Action(Inspect(c)
PRECOND: *EngineIn(c) ∧ WheelsOn(c) ∧ Chassis(c)*
EFFECT: *Done(c) ∧ Duration(10)*
RESOURCE*: Inspectors(1))*

Angegeben Ressourcen sind: 2 Inspektoren, eine Motorwinde und eine Radstation

Planen mit Ressourcen

- Die Lösung:



Links die Ressourcen. Es gibt 2 mögliche Zeitpläne, je nachdem welcher Motor zuerst eingebaut wird. Dies ist der optimale.

Obwohl das relativ einfach aussieht, ist es Rechentechnisch alles andere als das. Will man die optimale Durchlaufzeit finden, steigt die Rechenzeit dramatisch an. Realwelt Probleme sind aber leider wesentlich komplexer als dieses Beispiel.

Ein Problem von 1963 mit 10 Maschienen 10 Aufgaben mit je 100 Aktione brauchte 23 Jahre bis es jemand löste.

SEITE 519

Planen mit Ressourcen

- Die gezeigten Methoden unterteilen das Problem in eine Planungsphase, in der der POP Graph erstellt wird. Darauf folgt die Ablaufplanungsphase in der Zeit und Ressourcen berücksichtigt werden.
- Dies ist weit verbreitet, teilweise wird der erste Teil auch von Menschen erledigt, bevor ein System zu Einsatz kommt.
- Gibt es jedoch starke Ressoucenbeschränkungen, macht es meist Sinn diese bereits beim POP zu berücksichtigen.

Die Kombination der beiden Phasen wird durch eine Erweiterung des Planugnsschemas möglich. Wird aber jetzt (und im Russel) nicht weiter diskutiert.

Hierarchisches Task-Netzwerk-Planen

Hierarchisches Task-Netzwerk-Planen

- Weit verbreitete Idee im Umgang mit Komplexität: Hierarchische Zerlegung
- Zerleget Funktionen in kleinere Anzahl Aktionen
- Rechenaufwand für richtige Lösung wird geringer
- Im Bestfall führt das zu Planungsalgorithmen mit linearer statt exponentieller Zeit

Hierarchische Zerlegung ist die am weitesten verbreitete Idee im Umgang mit Komplexität.

Bspw: Komplexe Software zerlegt in hierarchische Subroutinen oder Objektklassen, Befehlsgewalten in Armeen, Regierungen und Unternehmen haben hierarchische Strukturen von Abteilungen, Büros, …

Vorteil: Eine Funktion wird auf jeder Ebene auf eine kleine Anzahl von Aktionen auf der nächstniedrigeren Ebene reduziert. Der Rechenaufwand um die richtige Anordnung der Aktivitäten für das Problem zu finden wird kleiner.

Nicht-hierarchische Methoden reduzieren hingegen eine Aufgabe auf eine große Anzahl von Einzelaktionen.

Im Bestfall führt das zu Planungsalgorithmen mit linearer statt exponentieller Zeit.

Hierarchisches Task-Netzwerk-Planen

- Ursprünglicher Plan wird in Teilaufgaben zerlegt
- Aktionszerlegung reduziert Aktionen höherer Ebenen auf partiell geordnete Menge von Aktionen auf niedrigeren Ebenen
- Bsp: „Haus bauen" in „Grundstück kaufen" und „Bauen".
- Zerlegung wird fortgesetzt, bis nur noch primitive Aktionen im Plan verbleiben
- Primitive Aktionen sind Aktionen die der Agent automatisch ausführen kann

Das HTN-Planen beschreibt, wie Aufgaben in Teilaufgaben zergliedert werden.

Pläne werden verfeinert, in dem Aktionszerlegungen vorgenommen werden.

Jede Aktionszerlegung reduziert eine Aktion auf höherer Ebene in eine partiell geordnete Menge von Aktionen auf niedriger Ebene.

Beispiel: Aufgabe „Haus bauen“ wird in die Teilaufgaben „Grundstück kaufen“ und „Bauen“ zerlegt.

Dieser Prozess wird fortgesetzt, bis nur noch primitive Aktionen im Plan verbleiben. Primitive Aktionen sind Aktionen, die der Agent automatisch ausführen kann.

Hierarchisches Task-Netzwerk-Planen

- Bei „reinem" HTN-Planen werden Pläne ausschließlich durch aufeinanderfolgende Aktionszerlegungen erzeugt
- HTN betrachtet Planen als Prozess eine Aktivitätsbeschreibung zu konkretisieren
- STRIPS-Aktionsbeschreibungen können in Aktionszerlegungen umgewandelt werden
- HTN als Erweiterung von POP betrachten
- Keine neue Notation, es können bestehende Konventionen verwendet werden

Bei „reinem“ HTN-Planen werden Pläne ausschließlich durch aufeinanderfolgende Aktionszerlegungen erzeugt. HTN betrachtet „Planen“ deshalb als Prozess eine Aktivitätsbeschreibung zu konkretisieren und nicht als Prozess des Aufbaus einer Aktivitätsbeschreibung.

Jede STRIPS-Aktionsbeschreibung kann in eine Aktionszerlegung umgewandelt werden.

Partiell ordnendes Planen kann als Sonderfall in der „reinen“ HTN-Planung betrachtet werden.

Für bestimmte Aufgaben, besonders neuartige konjunktive Ziele ist die reine HTN-Planung eher unnatürlich. Es sollte ein hybrider Ansatz verwendet werden, wobei Aktionszerlegungen als Planverfeinerungen im POP verwendet werden, zusätzlich zu den Standardoperationen der Bestimmung einer offenen Bedingung und der Konfliktlösung durch hinzufügen von Ordnungsbedingungen.

Wenn man HTN-Planen als Erweiterung des POP betrachtet, kann man die bereits bekannten Notationskonventionen verwenden und muss nicht eine neue Menge einführen.

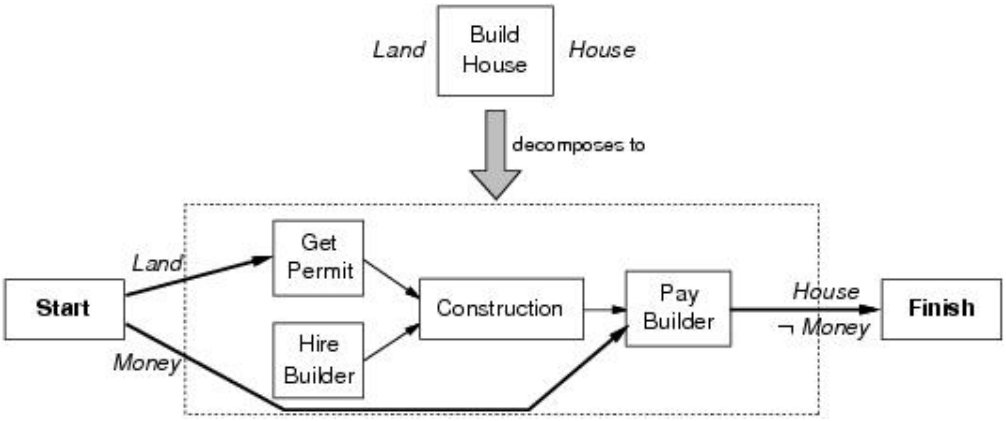
Repräsentation von Aktionszerlegungen

- Allgemeine Beschreibungen sind in der „plan library" hinterlegt
- Jede Methode ist ein Ausdruck der Form „Zerlegen(a, d)"
- Aussage: Aktion a wird in Plan d zerlegt, Plan d wird als partiell geordneter Plan repräsentiert

Allgemeine Beschreibungen zur Zerlegung eines Plans sind in der „Plan Library" (Planbibliothek) hinterlegt. Von dort werden sie extrahiert und instantiiert um die Bedürfnisse des zu konstruierenden Plans zu erfüllen. Jede Methode ist ein Ausdruck der Form „Zerlegen(a, d)". Dies besagt, das eine Aktion a in einen Plan d zerlegt werden kann. Der neue Plan d wird als partiell geordneter Plan repräsentiert

Repräsentation von Aktionszerlegungen

- Beispiel: Haus bauen



Beispiel: Haus bauen (Russell, Seite 522f)

Start-Aktion stellt alle Vorbedingungen von Aktionen im Plan bereit, die nicht durch andere Aktionen bereitgestellt werden. Diese werden auch als externe Bedingungen bezeichnet. In unserem Beispiel sind diese externen Vorbedingungen der Zerlegung Land und Geld.

Analog dazu verhalten sich die sog externen Effekte, wobei es sich bei diesen um Vorbedingungen für das Ende handelt.

Also alle Effekte der Aktionen, die nicht durch andere Aktionen negiert werden. Im Beispiel sind diese Haus und –Geld.

Einige HTN-Planer unterscheiden auch zwischen primären Effekten (Haus) und sekundären Effekten (-Geld).

Für die Realisierung von Zielen können nur primäre Effekte verwendet werden. Beide Arten könnten jedoch Konflikte mit anderen Aktionen im Plan verursachen, wodurch der Suchraum wesentlich verkleinert wird. Ebenso könnte die Entdeckung unerwarteter Pläne verhindert werden.

Repräsentation von Aktionszerlegungen

$Action(BuyLand,$ PRECOND:$Money,$ EFFECT:$Land \wedge \neg Money)$
$Action(GetLoan,$ PRECOND:$GoodCredit,$ EFFECT:$Money \wedge Mortgage)$
$Action(BuildHouse,$ PRECOND:$Land,$ EFFECT:$House)$

$Action(GetPermit,$ PRECOND:$Land,$ EFFECT:$Permit)$
$Action(HireBuilder,$ EFFECT:$Contract)$
$Action(Construction,$ PRECOND:$Permit \wedge Contract,$
EFFECT:$HouseBuilt \wedge \neg Permit)$
$Action(PayBuilder,$ PRECOND:$Money \wedge HouseBuilt,$
EFFECT:$\neg Money \wedge House \wedge \neg Contract)$

$Decompose(BuildHouse,$
$Plan($STEPS:$\{S_1 : GetPermit, S_2 : HireBuilder,$
$S_3 : Construction, S_4 : PayBuilder\}$
ORDERINGS:$\{Start \prec S_1 \prec S_3 \prec S_4 \prec Finish, \quad Start \prec S_2 \prec S_3\},$
LINKS:$\{Start \xrightarrow{Land} S_1, Start \xrightarrow{Money} S_4,$
$S_1 \xrightarrow{Permit} S_3, S_2 \xrightarrow{Contract} S_3, S_3 \xrightarrow{HouseBuilt} S_4,$
$S_4 \xrightarrow{House} Finish, S_4 \xrightarrow{\neg Money} Finish\}))$

Aktionsbeschreibung für das Hausbau-Problem

Aktionsbeschreibungen für das Hausbauproblem und eine detaillierte Zerlegung der Aktion HausBauen. Diese Beschreibungen verwenden ein vereinfachtes Konzept von Geld und eine optimistische Sichtweise der Bauunternehmer.

Repräsentation von Aktionszerlegungen

- Eine Zerlegung sollte eine korrekte Implementierung der Aktion sein
- Dann, wenn der Plan d ein vollständiger, konsistenter partiell geordneter Plan für das Problem ist, die Effekte der Aktion a mit den Vorbedingungen von a zu erzielen
- Zerlegung ist dann korrekt, wenn sie das Ergebnis eines logisch konsequenten partiell ordnenden Planers ist

Eine Zerlegung sollte eine korrekte Implementierung der Aktion sein. Dies ist dann gegeben, wenn ein Plan d ein vollständiger und konsistenter partiell geordneter Plan für das Problem ist, die Effekte der Aktion a mit den Vorbedingungen von a zu erzielen. Eine Zerlegung ist dann korrekt, wenn sie das Ergebnis eines logisch konsequenten partiell ordnenden Planers ist.

Repräsentation von Aktionszerlegungen

- Durch mehrere Zerlegungsmöglichkeiten verborgene Vorbedingungen und Effekte in STRIPS-Aktionsbeschreibung unvermeidbar
- Vorbedingungen und Effekte auf höchster Ebene grundsätzlich Untermenge der echten Vorbedingungen und Effekte jeder primitiven Implementierung
- Beschreibung auf höherer Ebene ignoriert alle internen Effekte der Zerlegung
- Beschreibung auf höherer Ebene spezifiziert keine Intervalle in der Aktivität in denen die Vorbedingungen und Effekte Gültigkeit besitzen

Berücksichtigt man, das eine Aktion auf höchster Ebene (HausBauen) mehrere mögliche Zerlegungen haben kann, ist es unvermeidbar, das ihre STRIPS-Aktionsbeschreibung bestimmte Vorbedingungen und Effekte dieser Zerlegung verbirgt.

Die Vorbedingungen und Effekte auf höchster Ebene sollten jeweils die Schnittmenge ihrer externen Vorbedingungen und externen Effekte der Zerlegung sein.

Anders: Die Vorbedingungen und Effekte auf höchster Ebene sind auf jeden Fall eine Untermenge der echten Vorbedingungen und Effekte jeder primitiven Implementierung.

Weitere Arten, Informationen zu verstecken:

-Beschreibung auf höherer Ebene ignoriert alle internen Effekte der Zerlegung vollständig

Bsp: Zerlegung HausBauen hat temp int Efffekte „Genehmigung“ und „Vertrag“.

-Beschreibung auf höherer Ebene spezifiziert nicht die Intervalle innerhalb der Aktivität, während denen die Vorbedingungen und Effekte auf höherer Ebene Gültigkeit besitzen.

Bsp: Vorbedingung „Land“ muss nur wahr sein, bis die Aktion „GenehmigungEinholen“ ausgeführt wurde

Bsp: „Haus“ ist erst wahr, nachdem „BauunternehmerBezahlen“ ausgeführt wurde

Repräsentation von Aktionszerlegungen

- „information hiding" ist wichtig, wenn hierarchisches Planen Komplexität reduzieren soll
- Es muss möglich sein, über Aktionen auf höherer Ebene schließen zu können, ohne auf Implementierungsdetails achten zu müssen
- Nachteil: Konflikte die nicht aus höheren Ebenen heraus erkannt werden können

„Information hiding" ist wichtig, wenn hierarchisches Planen Komplexität reduzieren soll. Es muss möglich sein, über Aktionen auf höherer Ebene schließen zu können, ohne unzählige Implementierungsdetails beachten zu müssen.

Nachteile:

Es könnte Konflikte zwischen internen Bedingungen einer höheren Aktion und internen Aktionen einer anderen geben, aber keine Möglichkeit dies aus der höheren Aktionsbeschreibungen heraus zu erkennen.

Wo primitive Aktionen vom Planungsalgorithmus als Punktereignisse behandelt werden können, haben höhere Aktionen ein temporäres – zeitliches – Ausmaß innerhalb dessen alle möglichen Dinge passieren können.

Repräsentation von Aktionszerlegungen

- Nachfolgerfunktion des POP ändern
- Sie soll auf den aktuellen partiellen Plan P anwendbar werden
- Neue Nachfolgerpläne bilden:
  - nicht-primitive Aktion a´ im Plan P auswählen
  - für jede Methode Zerlegen(a, d) aus der Planbibliothek a´ durch d´=SUBST(0, d) anwenden
  - Es gilt: a und a´ unifizieren mit der Substitution 0
- Für jede Zerlegung d´ gibt es die folgenden Schritte

Nachfolgerfunktion des POP ändern, so dass sie auf den aktuellen partiellen Plan P angewendet werden kann.

Neue Nachfolgerpläne werden gebildet, in dem eine nicht-primitive Aktion a´ im Plan P ausgewählt wird und dann für jede Methode Zerlegen(a, d) aus der Planbibliothek a´ durch d´=SUBST(0, d). Es gilt dabei, das a und a´ mit der Substitution 0 unifizieren.

Planer für Zerlegungen anpassen

3 Schritte

1. Möglichkeiten der gemeinsamen Nutzung?
2. Ordnungsbedingungen verbinden
3. Kausale Verknüpfungen verbinden

Planer für Zerlegungen anpassen

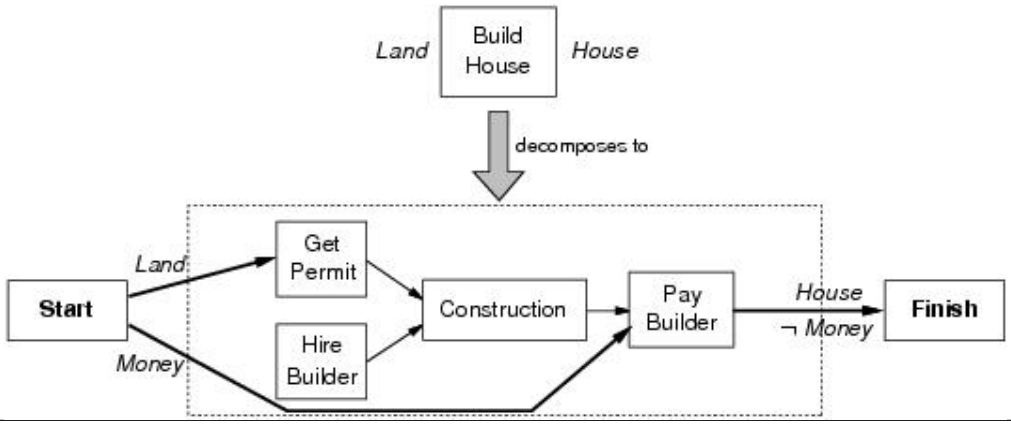
1. Aktion a´ aus P entfernen, dann für jeden Schritt s in der Zerlegung d´ eine Aktion auswählen, die die Aufgabe von s erfüllt und es dem Plan hinzufügt. Das kann entweder eine neue Instantiierung von s sein, oder ein existierender Schritt s´ aus P, der mit s unifiziert. Möglicherweise kann eine bereits existierende Aktion verwendet werden => gemeinsame Nutzung von Teilaufgaben (Subtask Sharing)

Zuerst wird die Aktion a´ aus dem Plan P entfernt. Dann wird für jeden Schritt s in der Zerlegung d´ eine Aktion ausgewählt, die die Aufgabe des Schritts s erfüllt und es dem Plan hinzufügt. Das kann entweder eine neue Instantiierung von s sein, oder ein bereits existierender Schritt s´ aus dem Plan P, der mit s unifiziert.

Möglicherweise kann eine bereits existierende Aktion verwendet werden. Dabei spricht man von gemeinsamer Nutzung der Teilaufgaben oder von Substask Sharing.

Planer für Zerlegungen anpassen

Beispiel für keine gemeinsame Nutzung von Teilaufgaben



Planer für Zerlegungen anpassen

2. Verbinden der Ordnungsbedingungen für a´ im ursprünglichen Plan mit den Schritten in d´.

   Es sollten zu strenge Ordnungen vermieden werden, da sonst mögliche Lösungen verhindert werden. Beste Lösung: Grund für die Ordnungs- bedingung angeben. Wird eine höhere Aktion expandiert können die Ordnungsbedingungen gelockert werden, solange sie konsistent mit dem Grund für die ursprüngliche Bedingung sind.

Planer für Zerlegungen anpassen

3. Kausale Verbindungen verknüpfen. Wenn B-p->a´ im ursprünglichen Plan eine kausale Verknüpfung war, ersetzen wir sie durch eine Menge kausaler Verknüpfungen von B zu allen Schritten in d´ mit den Vorbedingungen p die durch den Start-Schritt in der Zerlegung d bereitgestellt wurden.

   LandKaufen-Land->HausBauen

   wird durch

   LandKaufen-Land->Genehmigung

   ersetz.□

Planer für Zerlegungen anpassen

Weil höhere Aktionen Informationen über ihre endgültigen primitiven Implementierungen verstecken, wird eine weitere Änderung im POP-Algorithmus gebraucht.

Weil höhere Aktionen Informationen über ihre endgültigen primitiven Implementierungen verstecken, wird eine weitere Änderung im POP-Algorithmus gebraucht. Insbesondere führt der ursprüngliche POP-Algo Backtracking durch die Fehler durch wenn der aktuelle Plan einen nicht auflösbaren Konflikt enthält. Ein nicht auflösbarer Konflikt besteht dann, wenn eine Aktion einen Konflikt mit einer kausalen Verknüpfung erzeugt, aber nicht vor oder nach der Verknüpfung eingeordnet werden kann.

Bei höheren Aktionen können scheinbar unauflösbare Konflikte manchmal durch Zerlegung und Neuanordnung der konflikterzeugenden Aktionen gelöst werden. Dadurch kann es passieren, das man einen vollständigen und konsistenten primitiven Plan erhält, obwohl es keinen vollständigen und konsistenten höheren Plan gibt.

Das bedeutet, das ein vollständiger HTN-Planer auf viele Kürzungsmöglichkeiten verzichten muss die einem herkömmlichen POP-Planer zur Verfügung gestanden hätten.

Alternativ: Kürzen und hoffen, das keine Lösung verpasst wird.

Probleme bei HTN-Planern

Reines HTN-Planen ist unentscheidbar, da selbst die kürzeste HTN-Lösung beliebig lang werden kann, obwohl der zugrundeliegende Zustandsraum endlich ist. Es ist nicht möglich, die Suche nach fester Zeit zu terminieren. Die Schwierigkeit besteht darin, das Aktionszerlegungen rekursiv sein können. Es gibt jedoch 3 Möglichkeiten, die HTN-Pläne zu verkürzen:

Reine HTN-Pläne sind unentscheidbar, auch wenn der zugrunde liegende Zustandsraum endlich ist. Das ist natürlich nicht im Sinne des Erfinders, da man ja durch HTN-Planen Effizienz gewinnen will. Die Schwierigkeit besteht darin, das die Aktionszerlegungen rekursiv stattfinden können => HTN-Pläne können beliebig lang werden. Selbst die kürzeste HTN-Lösung kann beliebig lang sein, so das eine Terminierung nach fester Zeit nicht möglich ist.

Probleme bei HTN-Planern

1) Rekursion kann ausgeschlossen werden, da sie nur in sehr wenigen Domänen wirklich benötigt wird. Dadurch werden HTN-Pläne endlich und können aufgelistet werden

Probleme bei HTN-Planern

2) Die Länge der Lösungen an denen wir interessiert sind kann begrenzt werden. Durch den endlichen Zustandsraum ist sichergestellt, das ein Plan mit mehr Schritten als es Zustände gibt, eine Schleife enthält. Es entsteht kaum Verlust, wenn solche Lösungen ausgeschlossen werden und die Suche wird kontrolliert

Probleme bei HTN-Planern

3) Es kann der hybride Ansatz, der HTN- und POP-Planen kombiniert, übernehmen. Partiell ordnendes Planen kann erkennen, ob es einen Plan gibt – der hybride Ansatz ist klar entscheidbar

Probleme bei HTN-Planern

- Bei der dritten Möglichkeit ist Vorsicht geboten. POP kann primitive Aktionen zufällig verbinden – das kann dazu führen, das schwer verständliche und nicht dem hierarchischen Aufbau entsprechende Pläne entstehen.
- Umgehbar durch: Kosten-/Rabattfunktion

Die hybride Suche sollte gesteuert werden, um solche Ergebnisse zu vermeiden. Ein geeigneter Kompromiss ist es, das Hinzufügen von neuer Aktionen benachteiligt werden, d.h. das Aktionszerlegungen bevorzugt behandelt werden. Allerdings muss beachtet werden, das nicht beliebig lange HTN-Pläne entstehen bevor die primitiven Aktionen hinzugefügt werden.

Dies kann man durch den Einsatz einer Kostenfunktion bzw. Rabattfunktion erreichen. Je höher der Rabatt für neu eingefügte Aktionszerlegungen ist, um so mehr erinnert die Suche an reines HTN-Planen.

Hierarchische Pläne sind in der Realität viel einfacher auszuführen und bei Fehlern einfacher zu korrigieren.

Warum HTN?

- HTN funktioniert in der Praxis
- Erlaubt dem menschlichen Experten, wichtiges Wissen zur Ausführung komplexer Aufgaben bereitzustellen
- Große Pläne können so mit geringem Rechenaufwand konstruiert werden
- Lernfähigkeit des Agenten, bereits geplante Pläne in Bibliothek speichern und wiederverwenden => Agent wird kompetenter
- Wichtiger Aspekt: Verallgemeinerung (Kap. 19)

Warum verwendet man dennoch HTN-Planer?

HTN-Planer funktionieren in der Praxis.

Der menschliche Experte kann dem Agenten wichtiges Wissen zur Ausführung der komplexen Aufgabe in der Plan-Bibliothek bereitstellen. Dadurch können Pläne mit geringen Rechenaufwand konstruiert werden. Auch ist der Agent lernfähig, das heißt das er Pläne in seiner Plan-Bibliothek speichert und sie für spätere Planungen verwenden kann. Der Agent wird so kompetenter was seine Lösungsstrategien angeht.

Menschen verwenden den selben Ansatz.

Ein wichtiger Aspekt sollte hierbei nicht ausser Acht gelassen werden: Verallgemeinerung. Die konstruierten Methoden sollten problemspezfische Details eliminieren. Siehe Kapitel 19.

Planen in nicht deterministischen Domänen

• Bis jetzt: Planung in vollständig beobachtbaren, statischen, deterministischen Domänen
• Agenten konnten Planen und Ausführen und sicher sein, dass der Plan Erfolg hat
• Das entspricht leider nicht ganz der realen Welt
• Agenten müssen mit unvollständigen und/oder falschen Informationen Arbeiten können

• -> Bedingtes Planen

Vollständig beobachtbare Domänen

• Der Agent kennt immer den Aktuellen Zustand
• nicht deterministisch: d.h. der Agent kann nicht vorhersagn, ob seine Aktionen Erfolg haben.

• Ein Agent in einer solchen Umgebung baut (zur Planungszeit) bedingte Aktionen in seinen Plan ein, die beim Ausführen den Zustand der Umgebung überprüfen und zu einer Entscheidung kommen.
• Das Problem ist das Erstellen eines solchen Plans

Vollständig beobachtbare Domänen

Beispiel: Staubsaugerwelt

- Aktionen sind: *Links, Rechts, Saugen*
- Aussagen über Zustände: *InL, InR, SauberL, SauberR*
- Dazu werden in Effekten Disjunktionen erlaubt. Eine Aktion kann damit mehrere Ausgänge haben.

- z.B. Die Bewegung nach Links schlägt manchmal fehl

Vorher: *Aktion(Links, PRECOND: InR, EFFECT: InL)*
Mit Bedingung: *Aktion(Links, PRECOND: InR, EFFECT: InL ∨ InR)*

Die Bewegung schlägt zB fehl weil ein Hinderniss oä getroffen wird. Der Agent kann nicht 100% sicher sein, dass die Aktion erfolg hat.

Vollständig beobachtbare Domänen

Beispiel: Staubsaugerwelt

- Außerdem führen wir bedingte Effekte ein
- diese werden mit „when" dargestellt

Aktion(Saugen, PRECOND:, EFFECT: (when InL:SauberL) ∧ (when InR:SauberR)

- Notation um bedingte Schritte darzustellen:
- if/then/else

IF InL ∧ SauberL THEN Rechts ELSE Saugen

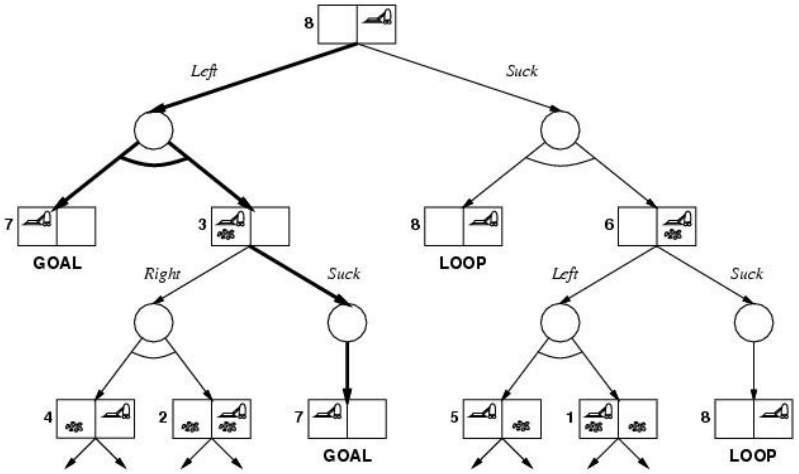
Vollständig beobachtbare Domänen

Beispiel: Doppel-Murphy-Sauger

- Ausgangszustand: Welt ist Sauber und der Agent rechts
  $SauberL \wedge SauberR \wedge InR$
- Ziel: Welt ist Sauber und der Agent links
  $SauberL \wedge SauberR \wedge InL$
- Der Agent verschmutzt manchmal saubere Felder wenn er dorthin geht und saugt er ein sauberes Feld, wird dieses manchmal schmutzig

Vollständig beobachtbare Domänen

Doppel-Murphy-Sauger



Suchbaum der Doppel Murphy Sauger Welt

Zustandsknoten (Saugerwelt) sind ODER Knoten, an denen eine Aktion gewählt werden muss.

Zufallsknoten (Kreise) sind UND Knoten. Jedes mögliche Ergebnis muss verarbeitet werden.

Die Lösung ist hier: (Anschreiben?) [Links, IF InL $\wedge$ SauberL $\wedge$ *SauberR THEN [] ELSE Saugen*]

SEITE 536

Vollständig beobachtbare Domänen

Und/Oder Graphen

• Wie gesehen, lassen Probleme in nicht deterministischen, vollständig einsehbaren Domänen auf UND/ODER Graphen führen
• Bereits gesehen in Kap. 7, statt Inferenzschritten Aktionen, aber der Algorithmus zur Problemlösung bleibt gleich.

Und/Oder Graphensuche

function AND-OR-GRAPH-SEARCH(*problem*) **returns** *a conditional plan* or *failure*
return OR-SEARCH(INITIAL-STATE[*problem*], *problem*, [])

function OR-SEARCH(*state, problem, path*) **returns** *a conditional plan* or *failure*
if GOAL-TEST[*problem*](*state*) **then return** the empty plan
if *state* is on *path* **then return** failure
for *action,state_set* in SUCCESSORS[*problem*](*state*) **do**
plan ← AND-SEARCH(*state_set, problem,* [*state* | *plan*])
if *plan* ≠ failure **then return** [*action* | *plan*]
return *failure*

function AND-SEARCH(*state_set, problem, path*) **returns** *a conditional plan* or *failure*
for each s_i in *state_set* **do**
$plan_i$ ← OR-SEARCH(s_i, *problem,path*)
if *plan* = failure **then return** failure
return [**if** s_1 **then** $plan_1$ **else if** s_2 **then** $plan_2$ **else** … **if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]

Und/Oder Graphensuche

- Der Algorithmus gibt das optimale Ergebnis zurück:

[Links, IF InL ∧ SauberL ∧ SauberR THEN [] ELSE Saugen]

- Pläne ähneln immer mehr Programmen die ggf. sogar Schleifen enthalten

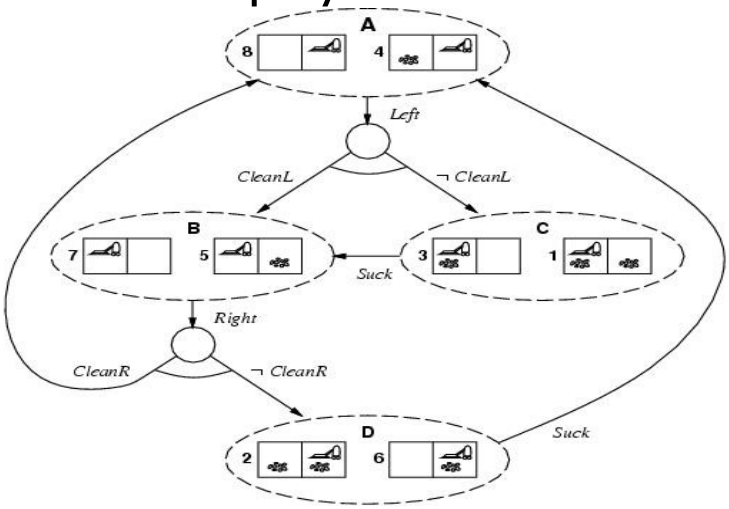
Schleifen treten zB in der Dreifach Murphy Welt auf, in der die Aktion Links nicht immer zum Erfolg führt. Der Agent muss diese also so oft wiederholen, bis sie Erfolgreich ist. Dies kann im schlechtesten Fall auch zu Endlosschleifen führen.

Partiell beobachtbare Domänen

- Werden im Prinzip wie vollständig beobachtbare mit UND/ODER Graphen behandelt.
- An den ODER Knoten befinden sich keine Zustände, sondern Glaubenszustände (siehe Kap. 3.6)
- Im Beispiel:
  - Der Agent kann sich zu Beginn *soweit er weiß* in 2 Zuständen befinden:
    $InR \wedge SauberL \wedge SauberR$ ODER $InR \wedge \neg SauberL \wedge SauberR$
  - Notation über das Wissen des Agenten erfolgt als K(x), wobei K für Knows steht und x für eine Zustandsbeschreibung. Hier:
    $K(InR) \wedge K(SauberR)$
  - Alle nicht aufgeführten Zustandsbeschreibungen (*SauberL*) können sowohl *true* als auch *false* sein

In diesem Beispiel kann der Agent nur feststellen, ob das Feld auf dem er sich befindet sauber ist; er hat keine Informationen über das andere Feld

Wechsel Murphy Welt



In dieser Welt, gilt dass Der Agent beim verlassen eines Feldes manchmal Schmutz hinterlässt.

Oben: Teil des UND/ODER Graphen dieser Welt ABCD sind Glaubenszustände des Agenten. Jeder Glaubenszustand hat 2 mögliche Zustände, da von den 3 Zustandsbeschreibungen SauberL, SauberR und InR/InL immer zwei in der Wissensbasis des Agenten sind.

SEITE 539

Ausführungsüberwachung und Neuplanen

Ausführungsüberwachung und Neuplanen

- Ausführungsüberwachungs-Agent prüft Wahrnehmungen
- Problem: unbegrenzte Indeterminanz
- Aktionsüberwachung ist unabdingbar

2 Arten der Ausführungsüberwachung:

1) Aktionsüberwachung
2) Planüberwachung

Der ausführungsüberwachende Agent prüft seine Wahrnehmungen.

Murphys Gesetz besagt, das selbst gut geplante Pläne häufig fehlschlagen.
Unbegrenzte Indeterminanz - es gibt immer unerwartete Zustände für die der Agent keine fehlerhafte Aktionsbeschreibungen hat.

$\Rightarrow$ Aktionsüberwachung ist unabdingbar

2 Arten der Ausführungsüberwachung:

1) Aktionsüberwachung. Einfache, aber schwache Form
2) Planüberwachung. Komplexere, aber effektivere Form

der Überwachung.

Neuplanender Agent weiß, was zu tun ist wenn Unerwartetes passiert: Neuen Plan erstellen oder alten Plan „reparieren“ (einen Weg vom unerwarteten Zustand zurück in den Plan finden)

Ausführungsüberwachung und Neuplanen

- Einfacher planender Agent führt Plan „dumm" schrittweise aus.
- Neu planender Agent beobachtet noch nicht ausgeführte Aktionen im Plan und den Gesamtplan. Prüft nach jedem Schritt seine Wahrnehmungen. Bei Änderungen versucht er, eine neue Aktionsfolge zu planen, die ihn in den Ursprungsplan zurückbringt

Der einfache planende Agent legt sich einen Plan unter Beachtung der Vorbedingungen zurecht und beginnt dann mit der schrittweisen Ausführung der Aktionen innerhalb des Plans. Er ignoriert unerwartete Veränderungen der Vorbedinungen.

Der neu planende Agent hingegen verhält sich etwas anders:

Er beobachtet sowohl den noch nicht ausgeführten Teil des Plans als auch den vollständigen Plan. Er prüft nach jedem Schritt seine Wahrnehmungen um festzustellen ob sich Vorbedingungen des Plans unerwartet geändert haben. Ist dies der Fall, geht der Agent im Plan zurück und versucht eine Aktionsfolge neu zu planen, die ihn schließlich in den ursprünglichen Plan zurückbringen soll.

Ausführungsüberwachung und Neuplanen

Schematische Darstellung eines neu planenden Agenten

```
function REPLANNING-AGENT(percept) returns an action
    static: KB, a knowledge base (+ action descriptions)
            plan, a plan initially []
            whole_plan, a plan initially []
            goal, a goal
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept,t))
    current ← STATE-DESCRIPTION(KB,t)
    if plan = [] then return the empty plan
            whole_plan ← plan ← PLANNER(current, goal, KB)
    if PRECONDITIONS(FIRST(plan)) not currently true in KB then
            candidates ← SORT(whole_plan,ordered by distance to current)

            find state s in candidates such that
                    failure ≠ repair ← PLANNER(current, s, KB)
            continuation ← the tail of whole_plan starting at s
            whole_plan ← plan ← APPEND(repair, continuation)
    return POP(plan)
```


Der Neuplaner bemerkt, das die Vorbedingung der ersten Aktion in „plan“ durch den aktuellen Zustand nicht erfüllt ist.

Anschließend ruft er den Planer auf um einen neuen Unterplan zu erzeugen => „korrektur“ der von der aktuellen Situation zu einem Zustand s „gesamt_plan“ führt.

In diesem Beispiel befindet sich der Zustand s einen Schritt hinter dem aktuell verbleibenden „plan“. Deswegen wird der gesamte Plan beobachtet und nicht nur der verbleibende Teil.

Im Allgemeinen wird s so nah wie möglich am aktuellen Zustand gewählt. Die Verbindung von „korrektur“ und dem Teil von „gesamt_plan“ ab s, den wir als „fortsetzung“ bezeichnen, bildet den neuen Plan und der Agent kann die Ausführung wieder aufnehmen.

Ausführungsüberwachung und Neuplanen

- Beispiel: Stuhl und Tisch derselben Farbe erzeugen. Der Stuhl ist blau, der Tisch ist grün. Es gibt Eimer mit den Farben Blau und Rot.
- Problemdefinition:

Init(Color(Chair, Blue) ∧Color(Table,Green) ∧ ContainsColor(BC,Blue) ∧ PaintCan(BC) ∧ ContainsColor(RC,Red) ∧ PaintCan(RC))
Goal(Color(Chair,x) ∧ Color(Table,x))
Action(Paint(object, color)
PRECOND: *HavePaint(color)*
EFFECT: *Color(object, color))*
Action(Open(can)
PRECOND: *PaintCan(can) ∧ ContainsColor(can,color)*
EFFECT: *HavePaint(color))*

Ein Agent soll Stuhl und Tisch in der gleichen Farbe erzeugen. Der Stuhl ist blau, der Tisch ist grün. Es gibt 2 Eimer mit den Farben Rot und Blau.

Ausführungsüberwachung und Neuplanen

- Der Planer des Agenten sollte folgenden Plan erzeugen:
- [Start, Öffnen(BlauerEimer); Malen(Tisch, Blau); Ende]
- Jetzt kann der Agent den Plan ausführen. Einfache Agenten würden es als Erfolg werten, wenn die einzelnen Schritte des Plans erfolgreich ausgeführt wurden.
- Der ausführungsbeobachtende Agent muss die Vorbedingung des Ende-Schritts prüfen.

Der Planer des Agenten sollte jetzt folgenden Plan erzeugen:

[Start, Öffnen(BlauerEimer); Malen(Tisch, Blau); Ende]

Jetzt kann der Agent den Plan ausführen. Einfache Agenten würden es als Erfolg werten, wenn die einzelnen Schritte des Plans erfolgreich ausgeführt wurden.

Der ausführungsbeobachtende Agent hingegen muss die Vorbedingung des Ende-Schritts prüfen. Diese besagt, das beide Möbel die selbe Farbe haben.

Ausführungsüberwachung und Neuplanen

- Beispiel: Vergessener grüner Fleck auf dem Tisch
- Korrektur: Agent erkennt, das der aktuelle Zustand identisch mit der Vorbedingung zu „Malen" ist. Er wählt die leere Folge „korrektur" für „reparatur" und macht einen neuen Plan für die [Malen; Ende]-Folge, die er zuvor schon probiert hat.
- Schleife, bis der Tisch in der Wahrnehmung vollständig Blau ist.

Der Agent nimmt wahr, das nach der Ausführung des Plans einen grünen Fleck auf dem Tisch gibt.

Er erkennt, das der aktuelle Zustand identisch mit der Vorbedingung zur Aktion „Malen“ ist und wählt eine leere Folge „korrektur“ als Reparatur des Plans. Dann macht er einen neuen Plan der die [Malen; Ende]-Folge enthält, die er zuvor schon ausprobiert hat. Dies geschieht solange in einer Schleife, bis der Agent wahrnimmt das der Tisch blau ist.

Hierbei ist zu beachten, das die Schleife nicht durch eine explizite Wiederholung im Plan entsteht, sondern durch den Prozess der Planung-Ausführung-Neuplanung.

Ausführungsüberwachung und Neuplanen

- Aktionsüberwachung ist eine einfache Methode der Ausführungsüberwachung
- Führt manchmal zu wenig intelligentem Verhalten
- Aktionsüberwachung erkennt Fehler erst, wenn die Aktion bereits ausgeführt wurde

Die Aktionsüberwachung ist eine einfache Methode der Ausführungsüberwachung, führt allerdings manchmal zu wenig intelligentem Verhalten. Sie kann Fehler erst erkennen, wenn es bereits zu Spät ist, also dann, wenn die Aktion bereits ausgeführt wurde.

Bsp.: Sollen der Tisch und der Stuhl rot angemalt werden, ist aber zu wenig Farbe für beide Möbel vorhanden, erkennt die Aktionsüberwachung den Fehler erst, nach dem der Tisch oder der Stuhl angemalt wurde. Der Fehler kann jetzt nicht mehr verhindert werden und muss extrem aufwendig korrigiert werden (hier: andere Farbe nehmen, Stuhl/Tisch noch mal mit anderer Farbe anmalen).

Ausführungsüberwachung und Neuplanen

- Fehler müssen erkannt werden, sobald ein Zustand erreicht ist, von dem aus der restliche Plan nicht mehr funktioniert
- Dies erledigt die Planüberwachung
- Prüft Vorbedingungen für den gesamten restlichen Plan, außer denen die durch andere Schritte im Plan erzeugt werden
- Beendet die Ausführung eines aussichtslosen Plans so früh wie möglich, statt weiterzumachen bis der Fehler tatsächlich auftritt

Man muss also den Fehler bereits erkennen, wenn ein Zustand erreicht ist von dem aus der gesamte restliche Plan nicht mehr funktioniert.

Dies erledigt die Planüberwachung. Sie prüft die Vorbedingungen für den gesamten restlichen Plan, außer denen die später im Plan durch die anderen Schritte erzeugt werden und beendet die Ausführung eines aussichtslosen Plans so früh wie möglich, anstatt weiterzumachen bis der Fehler tatsächlich auftritt.

Dies schützt den Agenten vor Sackgassen, von wo aus das Ziel nicht mehr zu erreichen wäre.

Diese Eigenschaft macht den Agenten intelligenter als einen Mistkäfer, der nicht erkennen kann ob er die Mistkugel noch besitzt. Unser Agent kann das erkennen und anfangen, eine neue Kugel zu rollen.

Ausführungsüberwachung und Neuplanen

- Es ist relativ einfach, den Planungsalgorithmus so zu ändern, das er an jedem Punkt die Vorbedingungen für den Erfolg des verbleibenden Plans hinzufügt
- Wenn zusätzlich geprüft wird, ob der aktuelle Zustand die gesamten zukünftigen Planvor-bedingungen erfüllt kann die Planüberwachung den sog. „glücklichen Zufall" benutzen

Es ist mit nur wenig Aufwand verbunden den Planungsalgorithmus so zu verändern, das er an jedem Punkt die Vorbedingungen für den verbleibenden Plan hinzufügt. Wird zusätzlich geprüft, ob der aktuelle Zustand die gesamten zukünftigen Planvorbedingungen erfüllt, kann die Planüberwachung den sogenannten „glücklichen Zufall“ benutzen.

Bsp: Wenn ein anderer Agent den Tisch rot anmalt während „unser“ Agent den Stuhl rot anmalt sind die endültigen Vorbedingungen des Plans erfüllt. Unser Agent kann das erkennen und malt den Tisch nicht noch einmal rot an, sondern beendet seine Arbeit früher.

Ausführungsüberwachung und Neuplanen

- In partiell beobachtbaren Umgebungen komplizierter
  - Es können Dinge schief laufen, ohne das der Agent das mitbekommt
  - Es kann sein, das für die Prüfung von Vorbedingungen Sensoreingaben nötig sind und diese Eingaben geplant und überwacht werden müssen,…

  Schlechtester Fall: Komplexer Plan für Sensoreingaben nötig und der Agent will alle Vorbedingungen überwachen => er schafft es vielleicht nie, seine Aufgabe zu erledigen

In partiell beobachtbaren Umgebungen ist die Überwachung komplizierter. Es können Dinge schief laufen, ohne das der Agent Kentnis davon erlangt und es kann für die Überprüfung der Vorbedingungen erforderlich sein, das Sensoreingaben verarbeitet werden müssen. Diese Sensoreingaben müssen auch wieder geplant werden, entweder zur Planungszeit (bedingtes Planen) oder zur Ausführungszeit.

Schlechtester Fall:

Für die Sensoreingaben ist ein komplexer Plan nötig, der Überwacht werden muss und für den Sensoreingaben nötig sind => wenn der Agent alle Vorbedingungen überprüfen will, kann es passieren das der Agent nie seine Aufgabe(n) erledigt.

Er sollte sich auf die Überprüfung wichtiger Variablen beschränken, für die wahrscheinlich ist das etwas schief läuft und deren Überprüfung nicht zu aufwendig ist.

Probleme bei Ausführungsüberw./Neuplanung

- Probleme entstehen, wenn wiederholte Versuche des Agenten nutzlos sind
- Lösungen:
  1. Zufällig einen Plan aus der Menge der Korrekturpläne auswählen
  2. Variationen in Korrekturplänen
  3. Lernfähigkeit (Kapitel 21)

Probleme entstehen wenn

Wiederholte Versuche des Agenten nutzlos sind, d.h. sie durch eine Vorbedingung oder einen Effekt blockiert sind, die er nicht erkennen kann

Mögliche Lösungen:

Zufällig einen Plan aus der Menge der Korrekturpläne auswählen

Variationen in den Korrekturplänen sind sinnvoll, wenn der Agent nicht zwischen dem echt nichtdeterministischen und dem nutzlosen Fall unterscheiden kann

Weitere Lösung:

Lernen. Nach einigen Versuchen sollte der Agent in der Lage sein, Aktionsbeschreibungen zu ändern. Diese Lösungsmöglichkeit wird in Kapitel 21 beschrieben.

Probleme bei Ausführungsüberw./Neuplanung

- Agent kann nicht in Echtzeitumgebungen arbeiten
- Zeit für Neuplanungen ist unbegrenzt
- Zeit für Entscheidungsfindung ist unbegr.
- Kann keine neuen Ziele formulieren
- Kann keine neuen Ziele neben seinem aktuellen Ziel akzeptieren
- Lösung dieser Probleme: Stetiges Planen

Trotz dieser Verbesserungen hat der Agent einige teils gravierende Nachteile

1) Er kann nicht in Echtzeitumgebungen arbeiten
2) Die Zeit für Neuplanungen ist nicht begrenzt
3) Die Zeit, die für eine Entscheidung für eine Aktion benötigt wird ist nicht begrenzt
4) Er kann keine neuen Ziele selbsttätig formulieren
5) Er kann keine neuen Ziele neben seinem aktuellen Ziel akzeptieren

Die Lösung dieser Probleme wird im Kapitel „Stetiges Planen“ erarbeitet.

Stetiges Planen

Stetiges Planen

- In diesem Abschnitt entwerfen wir einen Agenten, der unendlich lange in einer Umgebung verbleibt
- Kein „Problemlöser"
- Planer und Ausführungsüberwacher als ein Prozess vorstellbar: stetiger Planungsagent

Entwerfen eines Agenten, der unendlich lange in einer Umgebung bleibt.

⇒Kein „Problemlöser" mit einem bestimmten Ziel, sondern ein Agent, der eine Folge sich ständig ändernder Zielformulierungsphasen, Planung und Ausführung durchlebt

Planer und Ausführungsüberwacher als ein Prozess vorstellbar => stetiger Planungsagent

Stetiges Planen

- Agent ist aktiver Bestandteil des Plans
- Seine Aktivitäten beinhalten u.A.
  - Ausführung bestimmter Schritte des Plans
  - Verfeinerungen des Plans
  - Änderungen des Plans
- Bei erster Zielformulierung keine ausführbaren Aktionen
  - verbringt Zeit mit der Erstellung eines partiell geordneten Plans
  - Beginnt möglicherweise mit der Ausführung des Plans, wenn unabhängige Unterziele existieren

Der Agent ist Teil des Wegs durch den Plan, also aktiver Bestandteil.

Seine Aktivitäten beinhalten u.A.

-Ausführung bestimmter Schritte des Plans die gerade zur Ausführung stehen

-Verfeinerungen des Plans – Vorbedingungen erfüllen, Konflikte auflösen,

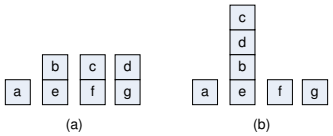
-Änderungen des Plans – zusätzliche Informationen die während der Ausführung aufgegriffen wurden

Bei erster Zielformulierung keine ausführbereiten Aktionen bereit, der Agent verbringt also einige Zeit mit der Erstellung eines partiellen Plans.

Er kann mit der Ausführung des noch nicht fertigen Plans beginnen, wenn er unabhängige Unterziele erreichen kann. Dabei überwacht er seine Umwelt und aktualisiert sein Weltmodell, auch wenn die Überlegungen noch andauern.

Stetiges Planen

Beispiel: Blockwelt-Domäne



Aktion „bewegen(x,y)“ bewegt Block x auf Block y wenn beide frei sind.

Action(*Move(x,y),*
PRECOND: *Clear(x)* ∧ *Clear(y)* ∧ *On(x,z)*
EFFECT: *On(x,y)* ∧ *Clear(z)* ∧ ¬*On(x,z)* ∧ ¬*Clear(y)*

Ein Beispiel: Blockwelt, die Steine sollten gestapelt werden (Ausgangszustand a, Zielzustand d).

Aktion „bewegen(x, y)“ wird benötigt, die den Block x auf den Block y bewegt, Vorausgesetzt beide Blöcke sind oben frei.

Stetiges Planen

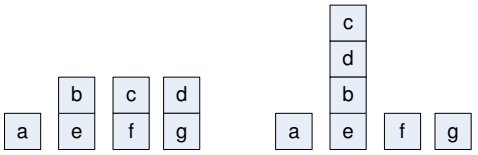
- Ziel des Agenten: auf(c, d) $\wedge$ auf(d, b)
- Plan wird inkrementell erzeugt, für jeden Schritt ist die Zeitdauer begrenzt
- Nach jedem Schritt NoOp als Aktion zurückgeben, Agent prüft seine Wahrnehmungen

Das Ziel des Agenten soll sein, das der Block d auf dem Block c, und das der Block d auf dem Block b steht.

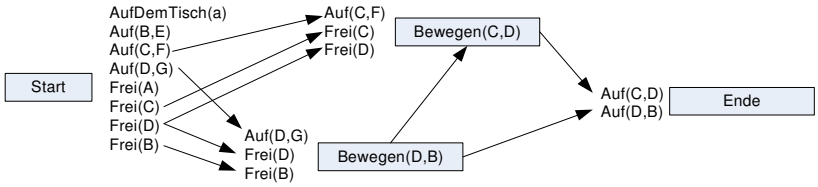
Dabei gehen wir davon aus, das dem Agenten dieses Ziel mitgeteilt wurde und er nicht selbst das Ziel formulieren musste.

Er erzeugt nun den Plan inkrementell. Für jeden Inkrementationsschritt ist die Zeitdauer begrenzt. Nach jedem Schritt gibt der Agent die Aktion „NoOp“ zurück und prüft wieder seine Wahrnehmungen.

Stetiges Planen



(a) Startzustand
(b) Zielzustand



Anfänglicher Plan

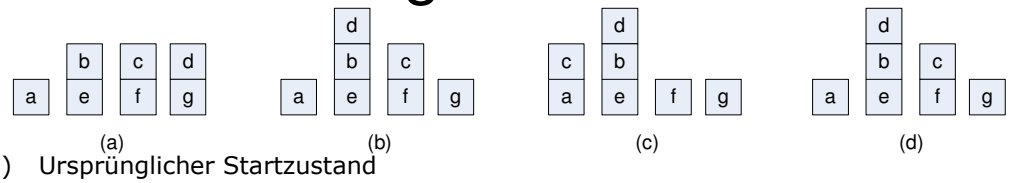
Zielformulierung: c auf d auf b stapeln.

Anfänglicher Plan, vom Agenten konstruiert. Ist nicht von einem normalen POP-erzeugten Plan unterscheidbar.

Stetiges Planen

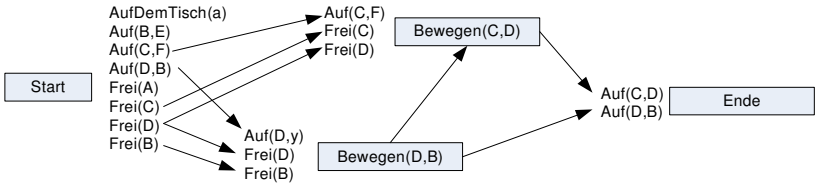
- Jetzt könnte der Agent seinen Plan ausführen, aber ein anderer Agent hat den Block D auf B gestellt.
- Der Agent erkennt das Problem, und aktualisiert seinen Plan entsprechend

Stetiges Planen



(a) (b) (c) (d)

(a) Ursprünglicher Startzustand
(b) Aktueller Zustand nach Störung
(c) Zwischenschritt nach Bewegen(C,D). Durch Fehler wurde c auf a fallen gelassen, Bewegen(C,D) wird erneut ausgeführt
(d) Endzustand



Neuer Plan

Zielformulierung: c auf d auf b stapeln.

Nachdem der Agent seinen ursprünglichen Plan durch eine externe Störung (d wurde auf b gestellt) nicht ausführen konnte, muss er seinen Plan korrigieren.

Die kausalen Verknüpfungen die Vorbedinungen für Bewegen(D,B) bereitgestellt haben

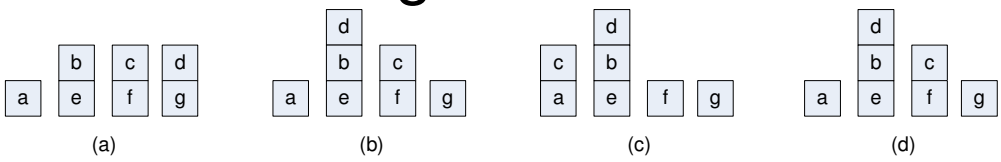
-Frei(B)

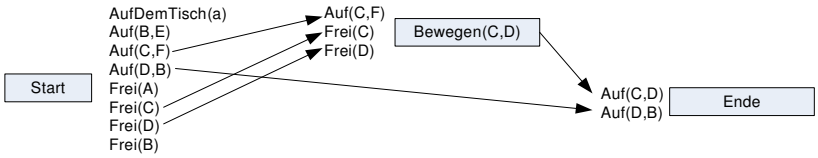
-Auf(D,y)

Sind ungültig geworden und müssen entfernt werden.

Der Plan ist jetzt unvollständig, da 2 Vorbedingungen nicht erfüllt sind und die Vorbedingung Auf(D,y) ist nicht instantiiert (deswegen y statt B!), da es keinen Grund zur Annahme gibt, das eine Bewegung von g aus passieren könnte (Der Platz über g ist leer, also kein Stein von g weg zu bewegen).

Stetiges Planen





Neuer Plan nach Erweiterung der kausalen Verknüpfung

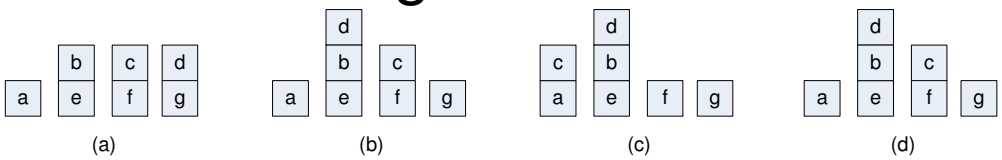
Jetzt kann der Agent die kausale Verknüpfung

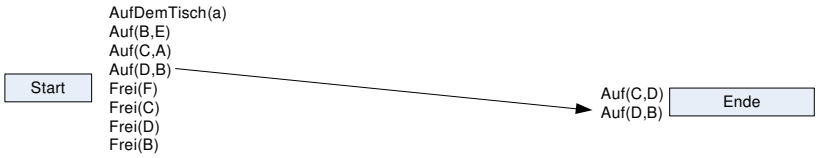
Bewegen(D,B) -> Ende

durch eine direkte Verbindung von Start und Ende ersetzen. Dies nennt man Erweiterung einer kausalen Verknüpfung.

Die Erweiterung erfolgt, wenn eine Vorbedingung durch einen früheren (statt späteren) Schritt im Plan bereitgestellt werden kann, ohne einen Konflikt zu erzeugen.

Stetiges Planen





Neuer Plan nach dem fehlerhaften Ausführen des Schritts „Bewegen(C,D)“. Neuer Zustand ist (c).

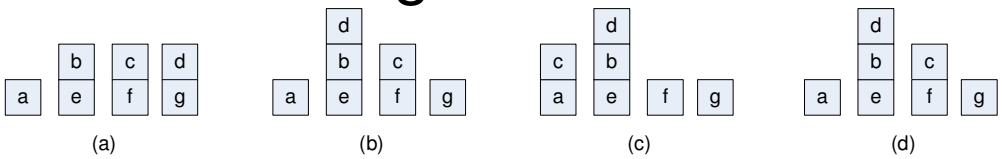
Nachdem der Agent die kausale Verknüpfung erweitert hat, kann er den Schritt „Bewegen(C,D)“ ausführen. Dieser Schritt wird dann aus dem Plan entfernt.

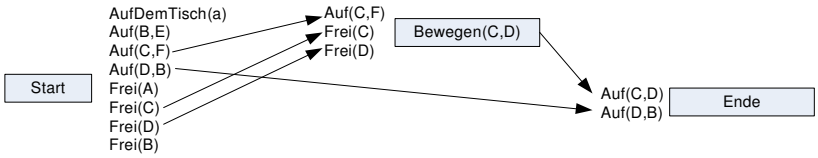
Leider hat der Agent den Block C nicht auf dem Block D abgelegt, sondern ihn versehentlich auf Block A fallengelassen.

Dadurch ergibt sich Zustand (c) und eine offene Bedingung für den Ende-Zustand (Auf(C,D)).

Diese offene Bedingung wird jetzt korrigiert.

Stetiges Planen



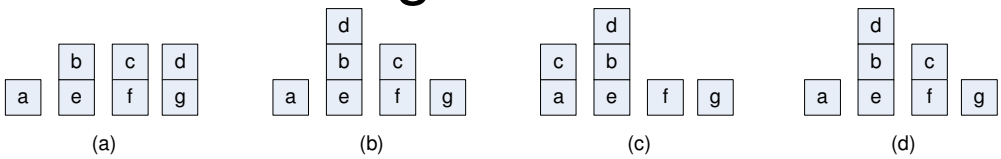


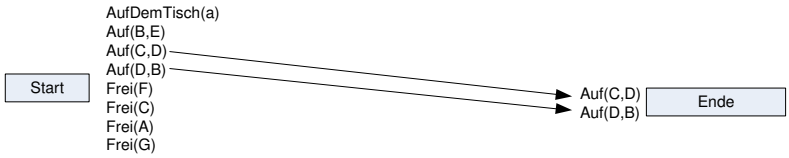
Korrigierter Plan, Agent führt erneut Schritt Bewegen(C,D) aus.

Erneut erfüllt Bewegen(C,D) die Zielbedingung und seine Vorbedingungen werden durch neue kausale Verknüpfungen vom Start-Schritt erfüllt.

Jetzt führt der Agent den Schritt Bewegen(C,D) aus und kommt in den Zielzustand.

Stetiges Planen





Endzustand (b), Zielbedingungen von Ende sind durch Start-Zustand erfüllt.

Jetzt sind beide Zielbedingungen von „Ende“ durch den Start-Zustand erfüllt.

Stetiges Planen

- Stetiges Planen ist dem POP-Planen sehr ähnlich
- POP-Algorithmus entfernt Defekte (offene Vorbedingungen und kausale Konflikte)
- Stetig planender Agent hat umfangreicheren Defekterkennungsbereich:
  - Fehlendes Ziel
  - Offene Vorbedingung
  - Kausaler Konflikt
  - Nicht unterstützte Verknüpfung
  - Redundante Aktion
  - Nicht ausgeführte Aktion
  - Unnötiges historisches Ziel

Der stetige Planer ist dem partiell ordnenden Planer sehr ähnlich.

Der POP-Algorithmus findet bei jeder Iteration einen Defekt im Plan und behebt ihn. Diese Defekte können offene Vorbedingungen oder kausale Konflikte sein.

Der stetig planende Agent hingegen hat einen sehr viel weiter gefassten Bereich der Fehlerkorrekturen:

-Fehlendes Ziel

-Agent kann dem Ende-Zustand weitere neue Ziele hinzufügen

-Offene Vorbedingungen

-Offener Vorbedingung kausale Verknüpfung hinzufügen, wozu entweder eine existierende oder neue Aktion gewählt wird (wie beim POP)

-Kausaler Konflikt

-Für eine Kausale Verknüpfung wird eine Ordnungsbedingung oder eine Variablenbedingung gewählt, um den Konflikt aufzulösen (wie beim POP)

-Nicht unterstützte Verknüpfung

-Wenn eine kausale Verknüpfung mit einer Vorbedingung in „Start“ nicht mehr wahr ist, wird diese Verknüpfung entfernt. Damit wird verhindert, das eine Aktion mit falscher Vorbedingung ausgeführt wird.

-Redundante Aktion

-Wenn eine Aktion keine kausalen Verknüpfungen bereitstellt, wird sie und ihre Verknüpfungen entfernt

-Nicht ausgeführte Aktionen

-Wenn eine Aktion (nicht ENDE!) ihre Vorbedingungen in „Start“ erfüllt hat, keine anderen Aktionen vor ihr angesiedelt sind und sie keine Konflikte mit kausalen Verknüpfungen verursacht, kann diese Aktion und ihre kausalen Verknüpfungen entfernt und als nächste auszuführende Aktion zurückgegeben werden

-Unnötiges historisches Ziel

-Gibt es keine offenen Vorbedingungen und keine Aktionen im Plan, haben wir die Zielmenge erreicht. Die Ziele und die Verknüpfungen werden entfernt und neue Ziele zugelassen.

Multiagenten-Planen

Multiagenten-Planen

- Bisher nur Einzelagentenumgebungen betrachtet
- Unser Einzelagent könnte andere Agenten in sein Weltmodell aufnehmen, ohne seinen Algorithmus anzupassen
  - verschlechterte Leistung
  - Insbesondere stehen andere Agenten unserem Einzelagenten nicht indifferent gegenüber

Bisher haben wir nur Einzelagentenumgebungen betrachtet. Dort hat sich „unser“ Agent jeweils nur alleine bewegt, ohne äußere Einflüsse von anderen Agenten zu erfahren (Natur zählt nicht als Agent).

Der Einzelagent könnte andere Agenten nun in sein Weltmodell aufnehmen, ohne seinen zugrundeliegenden Algorithmus zu ändern.

Dies würde jedoch seine Leistung in vielen Fällen verschlechtern, weil der Umgang mit anderen Agenten nicht das selbe ist wie der Umgang mit der Natur. Insbesondere stehen andere Agenten unserem Einzelagenten nicht indifferent gegenüber, wie Beispielsweise die Natur es tut.

Multiagenten-Planen

- Multiagentenumgebungen können kooperativ oder konkurrierend sein
- Einfaches kooperatives Beispiel:
  - Tennis-Doppel
- Pläne für Aktionen jedes Spielers müssen erstellt werden
- Agenten müssen sich über den Plan einig sein
  - Koordination, möglicherweise über Kommunikation

Multiagenten können untereinander in ihren Umgebungen kooperativ oder konkurrierend sein. Ein einfaches Beispiel für kooperative Umgebungen ist das Tennis-Doppel.

Es müssen Pläne für die Aktionen jedes Spielers erstellt werden => garantiert nicht Erfolg => Agenten müssen sich über Plan einig sein => Koordination, möglich über Kommunikation

Multiagenten-Planen

Kooperation

- Agenten im Tennisdoppel haben das Ziel, das Spiel zu gewinnen
- Daraus entstehen Unterziele
  - Ball zurückschlagen
  - Sicherstellen, das einer von ihnen das Netz abdeckt

Multiagenten-Planungsproblem

Beide Agenten haben das Ziel, das Spiel zu gewinnnen.

Unterziele:

-Ball zurückschlagen

-Sicherstellen, das einer von ihnen das Netz abdeckt

Multiagenten-Planen

Multiagenten-Planungsproblem

Agents(A,B)
Init(At(A,[Left,Baseline])∧ At(B,[Right, Net]) ∧ Approaching(Ball,[Right, Baseline]) ∧ PArtner(A,B) ∧ Partner(B,A))
Goal(Returned(Ball) ∧ At(agent,[x,Net]))
Action(Hit(agent, Ball)
PRECOND: *Approaching(Ball,[x,y]) ∧ At(agent,[x,y]) ∧ Partner(agent, partner) ∧ ¬At(partner,[x,y])*
EFFECT: *Returned(Ball))*
Action(Go(agent,[x,y])
PRECOND: *At(agent,[a,b])*
EFFECT: *At(agent,[x,y]) ∧ ¬ At(agent,[a,b]))*

Das Tennisdoppel-Problem

2 Agenten spielen, können sich an 4 möglichen Positionen befinden

[links,grundlinie], [rechts, grundlinie], [links, netz], [rechts, netz]

Der Ball kann zurückgespielt werden, wenn sich genau ein Spieler an der richtigen Stelle befindet.

Multiagenten-Planen

- Diese Notation führt 2 neue Funktionsmerkmale ein: Agenten(A, B), und für jede Aktion wird explizit der betreffende Agent angegeben.
- Lösung für Multiagenten-Probleme ist der gemeinsame Plan, der aus Aktionen für alle Agenten besteht
- Lösung für das Tennisproblem

  A: [Go(A,[Right, Baseline]), Hit(A,Ball)]

  B: [NoOp(B), NoOp(B)]

  oder

  A: [Go(A,[Left, net), NoOp(A)]

  B: [Go(B,[Right, Baseline]), Hit(B, Ball)]

Diese Notation führt 2 neue Funktionsmerkmale ein:

-Agenten(A,B) deklariert, das es 2 teilnehmende Agenten gibt

-Für jede Aktion wird explizit der betreffende Agent angegeben

Multiagenten-Planen

- Beide Agenten müssen den selben Plan wählen, sonst schlägt entweder keiner den Ball zurück oder das Netz ist ungedeckt
- Existenz gemeinsamer Pläne garantiert nicht, dass das Ziel erreicht wird
- Koordinationsmechanismus wird gebraucht

Beide Agenten müssen sich für den selben Plan entscheiden, falls es mehrere gemeinsame Pläne für das Problem gibt, wie beim Tennisdoppel-Beispiel.

Die Existenz gemeinsamer Pläne garantiert nicht, dass das Ziel erreicht wird. Es wird ein Koordinationsmechanismus gebraucht. Ebenso muss es gemeinsames Wissen zwischen den Agenten geben.

Mehrkörper-Planen

- Mehrkörper-Planen basiert auf POP in vollständig beobachtbarer Umgebung
- Weiterer Aspekt: Umgebung ist nicht mehr statisch
  - Synchronisierung wird benötigt
- Plan 2 als gemeinsame Aktionen:
  [<Go(A,[Left,Net]), Go(B,[Right,Baseline]>;
  <NoOp(A), Hit(B, Ball)>]

Das Mehrkörper-Planen basiert auf POP in vollständig beobachtbaren Umgebungen, jedoch gibt es einen weiteren Aspekt: Die Umgebung ist nicht mehr statisch, da andere Agenten agieren können während ein bestimmter anderer Agent überlegt.

Dadurch wird eine Synchronisierung benötigt.

Ein gemeinsamer Plan besteht aus partiell geordneten Graphen gemeinsamer Aktionen.

Mehrkörper-Planen

- Alternative Darstellung durch implizite Definition gemeinsamer Aktionen
- Erweiterung der STRIPS-Aktionsbeschreibung durch „Liste nebenläufiger Aktionen"
- Beispiel:

Action(Hit(A, Ball)
CONCURRENT*: ¬Hit(B,Ball)*
PRECOND: *Approaching(Ball,[x,y]) ∧ At(A,[x,y])*
EFFECT: *Returned(Ball))*

Alternative Darstellung kann durch implizite Definition gemeinsamer Aktionen beschrieben werden.

STRIPS-Aktionsbeschreibung wird durch „Liste nebenläufiger Aktionen“ beschrieben.

Während der Aktion Schlagen von Agent A darf es keine weitere Aktion Schlagen des Agenten B geben.

Mehrkörper-Planen

- Repräsentation erlaubt es, einen POP-ähnlichen Planer zu erstellen.
- Es gibt 3 Unterschiede:
  - Neben der temporären Ordnungsaktion erlauben wir „nebenläufig" oder „vor- oder nebenläufig"
  - Wenn für eine Aktion nebenläufige Aktionen gefordert werden, müssen diese Aktionen instantiiert werden und neue oder im Plan vorhandene Aktionen verwendet werden
  - Untersagte Nebenläufigkeiten sind zusätzliche Bedingungsquellen. Jede Bedingung muss durch bedingte konflikterzeugende Aktionen aufgelöst werden, die vor- oder nachher ausgeführt werden
- Das verschafft uns einen POP für Mehrkörper-Domänen, auf den auch die vorherigen Ansätze (HTN, Neuplanen,…) angewendet werden können

Koordinationsmechanismen

- Einfachste Methode: Konventionen
- Eine Konvention ist eine beliebige Bedingung für die Auswahl eines Plans
- Konventionen können durch Evolution entstehen
- Gibt es keine geeignete Konvention, ist Kommunikation nötig
- Anzeigen des Plans durch kurze Aktionsfolgen führt auch zum Erfolg
- Auswahl eines erfolgreichen gemeinsamen Plans kann auf 2 Wege erfolgen
  - Entwickler beweist, das die Vorgehensweise erfolgreich ist
  - Agenten sind reaktiv und entscheiden selbst über den Plan

Die einfachste Methode, einen Plan auszuwählen ist die der Konventionen.

Eine Konvention ist eine beliebige Bedingung für die Auswahl eines Plans.

Bspw. würde die Konvention „Bleibe auf deiner Seite des Platzes“ dazu führen, das der zweite Tennisdoppel-Plan ausgewählt würde, während die Konvention „Ein Spieler bleibt immer am Netz“ zur Auswahl des ersten Plans führen würde.

Allgemein verbreitete Konventionen nennt man auch „soziale Gesetze“.

Konventionen können auch durch Evolutionsprozesse entstehen => Kolonien sozialer Insekten, gemeinsames genetisches Erbgut

Gibt es keine gemeinsame Konvention, ist die Nutzung von Kommunikation nötig.

Bspw. Tennisdoppel: Agent ruft „Meiner“ oder „Deiner“ um den bevorzugten Plan anzuzeigen. Entsprechende geeignete Kommunikationswege werden in Kap 22 beleuchtet.

Zur Erkennung eines bevorzugten Plans ist es nicht erforderlich, das verbale Kommunikation eingesetzt wird. Anzeigen des bevorzugten Plans durch eine kurze Aktionsfolge ist ausreichend, um den gemeinsamen Plan eindeutig zu erkennen.

Die Auswahl des geeigneten Plans kann entweder der Entwickler über einen Beweis erbringen, oder die Agenten können selbst reaktiv sein und selbst über den gemeinsamen Plan entscheiden und beweisen, das ihr Plan effektiv ist.

Fragen?