Proseminar Künstliche Intelligenz

Planning & Acting

Marc Rochel

- Juli 2000 -

1 EINLEITUNG

Planung und Ausführung sind verschiedene Dinge. Die Planung alleine liefert einen Plan, dessen Durchführung in der Domäne, der dem Planer gegebenen Welt, kein Problem darstellt. Eine Anwendung eines solchen Planes auf die reale Welt bringt einige Probleme mit sich, die nicht alleine durch Veränderung des Planers gelöst werden können. Dieses Referat beschäftigt sich mit dieser Thematik und beschreibt einige Lösungsansätze für diese Probleme.

1.1 PROBLEMATIKEN

Ein Planer im Allgemeinen ist domänenunabhängig. Er stellt sozusagen einen allgemeingültigen Algorithmus dar, einen Plan zu finden um in einer gegebenen Welt ein bestimmtes Ziel zu erreichen, ausgegangen von einem bestimmten Ausgangspunkt. Das Problem einer Anwendung auf die reale Welt liegt dabei in der Modellierung der Domäne. Die reale Welt ist derart komplex, dass für gegebene Probleme eine Abstraktion geschaffen werden muss, damit ein Planer die gestellte Aufgabe bewältigen kann. So kann es vorkommen, dass aus einer zu stark von der Realität abstrahierten Domäne ein Plan entsteht, dessen Ausführung in der Realität scheitert. So schlägt z.B. der Plan „Gehe zur Bank und hebe Geld vom Konto ab." zur Beschaffung von Geld fehl, wenn die Bank gerade geschlossen ist, wenn ein Agent versucht den Plan auszuführen. Dieses Problem ließe sich noch durch den bekannten Partitial-Order Planner lösen, erweitert man die Operation der Domäne „hebe Geld vom Konto ab" um die Vorbedingung „Bank ist geöffnet" und fügt zusätzlich noch eine Operation ein „warte bis die Bank geöffnet ist".[1]

Es gibt jedoch auch Problematiken in der Realität, die sich nicht mit Hilfe eines Planers und dem bisherigen Domänenmodell lösen lassen. Solch ein Modell beschreibt eine Welt statisch. Alle Angaben sind wahr und alle relevanten Daten zur Lösung des Problems sind bekannt. Dies ist in der Realität allerdings oft nicht der Fall. In der Welt des Planers ist er der Einzige, der durch Aktionen die Welt verändern kann. Aktionen anderer Agenten in derselben Welt, die evtl. bereits erreichte Vorbedingungen für das Ziel wieder zerstören, können nicht in dem erstellten Plan erfasst werden.

1.2 PRINZIPIELLE LÖSUNGSANSÄTZE

Erstellte Pläne des Partitial-Order-Planners werden vom Agent „blind" ausgeführt. Sie ermöglichen es nicht, auf Veränderungen in der Umwelt des Agenten einzugehen. Im folgenden werden zwei Lösungsansätze dafür vorgestellt:

1.2.1 BEDINGTE PLANUNG

Conditional Planning erweitert den Operationenumfang des Planers. Es stellt ihm Operationen zur Wahrnehmung der Welt, in der sich der Agent zur Laufzeit befindet zur Verfügung sowie Aktionen zur Fallunterscheidung. Dadurch können Pläne erstellt werden für Probleme, zu dessen Lösung Informationen benötigt werden, die zur Zeit der Planung noch nicht bekannt sind. Dies ermöglicht einem Agenten zum Beispiel das Gewicht zweier Würfel zu messen, wenn sein Ziel ist, den Leichteren auf den Schwereren zu stellen.

[1] Für die Ausführung solcher Pläne braucht man auch immer einen Agenten der diese Operationen beherrscht. Dies muss natürlich bei der Domänenmodellierung beachtet werden. Die Erweiterung in diesem Fall funktioniert nur, wenn der Agent tatsächlich eine Operation „warte bis die Bank geöffnet ist" ausführen kann. Aus der Sicht des Planers ist dies eine ganz gewöhnliche Operation, die einen Effekt hat, mit der er eine der Vorbedingungen zur Erreichung des Ziels erfüllt wird. Der Agent jedoch muss, um diese Operation auszuführen, so lange auf die Uhr schauen, bis es acht Uhr ist und die Bank öffnet. Wird dies nicht in einer Operation gekapselt, versagt ein herkömmlicher Partitial-Order Planner.

1.2.2 INTEGRIERTE PLANUNG UND AUSFÜHRUNG

Bisher wurde die Planung immer getrennt von der Ausführung betrachtet. Dieses Verfahren verfolgt einen anderen Ansatz und verbindet diese beiden Phasen. Es erlaubt eine Neuerstellung des Plans zur Laufzeit („Replanning“), sobald der bisherige Plan nicht mehr ausgeführt, bzw. das Ziel nicht mehr durch den aktuellen Plan erreicht werden kann. So kann zum Beispiel auch auf Eingriffe anderer Agenten reagiert werden, indem ein neuer Plan zum Erreichen des Ziels erstellt wird, ausgegangen von der neuen aktuellen Situation, die der andere Agent bewirkt hat.

2 BEDINGTE PLANUNG

2.1 SPRACHERWEITERUNGEN

Zunächst erweitern wir die Sprache zur Darstellung eines Plans, um die Fallunterscheidung formell notieren zu können.

If ({Condition}, [{Actions a}], [{Actions b}])

bedeutet, dass die Aktionen a ausgeführt werden, wenn die Bedingung wahr ist, sonst die Aktionen b.

Außerdem muss ein neuer Effekt eingeführt werden, der die Auswertung einer wahrnehmenden Aktion dem aktuellen Situationsstatus hinzufügt.

Op(ACTION: *a(x)*,
PRECOND: *b(x)*,
EFFECT: KnowsWhether(“*c(x)*”))

Hierbei ist zu beachten, dass x Vektor ist. Sei der aktuelle Situationsstatus S und der Situationsstatus nach der Ausführung von a(x) S’, dann bedeutet diese Operation: wenn Vorbedingung b(x) erfüllt ist (Mehrere Vorbedinungen können durch eine Funktion dargestellt werden.), überprüft der Agent, ob c(x) in seiner Welt gerade wahr ist und setzt den aktuellen Situationsstatus S’ = S ∧c(x) wenn c(x) wahr ist, sonst S’ = S ∧¬c(x).

Beispiel: Folgende Operation sei Bestandteil der Domäne:

Op(ACTION: *prüfeTür(t)*,
PRECOND: *stehevorTür(t)*,
EFFECT: KnowsWhether(“*Türoffen(t)*”))

Sei T1 eine Tür in der Domäne und der Situationsstatus S = stehevorTür(T1). Der Folgestatus wäre dann zur Ausführungszeit S’ = stehevorTür(T1) ∧ Türoffen(T1) wenn T1 in der Realität des Agenten offen ist, sonst S’ = stehevorTür(T1) ∧¬ Türoffen(T1).

Diese Erkenntnis ließe sich mit der obigen Definition der if-Operation in einem Plan ausnutzen um das weitere Vorgehen zu entscheiden, zum Beispiel:

If ({stehevorTür(T1) ∧Türoffen(T1)}, [gehedurch(T1)], [öffneTür(T1), gehedurch(T1)])

2.2 BESCHREIBUNG DES AGENTEN

Damit ein Agent solch einen Plan mit Fallunterscheidungen ausführen kann, muss er gegenüber dem normalen Agenten der POP-Pläne ausführt, erweitert werden:

```
procedure Conditional-Planning-Agent
        plan=CPOP     // Plan erstellen durch Conditional Partitial-Order-Planner
        wiederhole
                nextAction=First(plan)
                solange nextAction eine if-Aktion ist
                        wenn die Bedingung der if-Aktion wahr ist:
                                plan=Thenpart(nextAction)+Rest(p)
                        sonst
                                plan=Elsepart(nextAction)+Rest(p)
                        nextAction=First(plan)
                ende
                wenn nextAction keine Aktionen enthält, höre auf
                sonst führeaus(nextAktion)
        ende
```

Interpreter für Bedingte Pläne
(First(p) liefert die erste Aktion des Planes p, Rest(p) liefert den Rest des Planes p, ohne die erste Aktion)

Dies ist nicht weiter kompliziert und soll nur verdeutlichen, wie ein Agent einen Bedingten Plan ausführen soll. Die Funktion „führeaus“ entspricht der Funktion zur Ausführung einer Aktion eines Agenten, der einen vom normalen Partital-Order-Plan ausführt.

2.3 BESCHREIBUNG DES BEDINGTEN PLANERS

Ein Conditional Partital-Order-Planner (CPOP) geht bei der Erstellung eines Plans zunächst so vor, wie der POP, d.h. er beginnt mit einem minimalen Plan mit Start und Ziel. Dabei gilt für den Start der in der Domäne definierte Situationsstatus und für Finish gibt es die zu erfüllenden Vorbedingungen, um das Ziel zu erreichen.[2] Es werden Operationen eingefügt, die als Effekt Vorbedingungen des Ziels erfüllen und anschließend Operationen, die die Vorbedingungen dieser Operationen erfüllen; und so weiter. Im Unterschied zum POP steht dem CPOP jedoch die Möglichkeit zur Verfügung eine Operation in den Plan zu integrieren, die den Agenten zur Laufzeit veranlasst, zu überprüfen, ob ein notwendiges Teilziel a() schon erreicht wurde. Der Planer muss nun beide Fälle berücksichtigen, denn beide können zur Laufzeit eintreten und es muss aus dem Plan für den Agenten ersichtlich sein, wie er in beiden Fällen reagieren soll. Der einfache Fall ist, das Teilziel a() ist erreicht. Dann kann der bisher erstellte Plan weiterentwickelt werden. Im anderen Fall, wenn Teilziel a() nicht erfüllt ist, muss ein alternativer Plan erstellt werden, denn der sich bisher in Entwicklung befindende Plan benötigt ja genau, dass a() erfüllt ist.

CPOP realisiert dies wie folgt: Ein Plan kann mehrere Finish-Steps enthalten, die durch ihren Kontext unterschieden werden. Dabei gilt: Ein Finish steht in Kontext k(), wenn k() zur Laufzeit (genauer: zur Überprüfungszeit, vergleiche Abschnitt 2.4) wahr sein muss, damit dieses Finish erreicht wird. Im Bezug auf oben stellt der Planer also das bereits existierenden Finish-Steps in den Kontext a(), bzw. erweitert deren Kontext um a() und erstellt den Plan für Kontext a() zu Ende. Anschließend erstellt er ein neues Finish unter dem Kontext, der noch nicht berücksichtigt wurde (Also alle bereits betrachteten Kontexte negiert und „logisch-und verknüpft“.) und

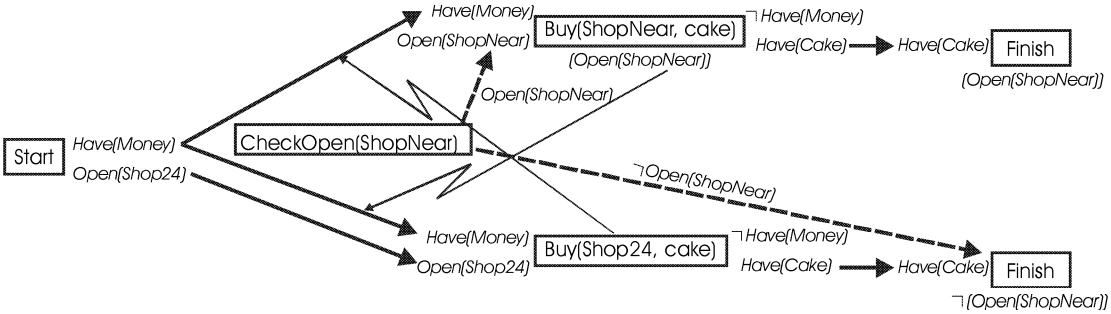
[2] Eine Vorbedingung kann auch als Teilziel betrachtet werden, das noch erfüllt werden muss um das eigentliche Ziel zu erreichen, denn es müssen alle Vorbedingungen der Operationen erfüllt werden um einen vollständigen Plan zu erhalten. Oder eben: Es müssen alle Teilziele erreicht worden sein, um einen vollständigen Plan zu erhalten.

beginnt, auch für das Finish in diesem Kontext einen Plan zu erstellen. Der Algorithmus ist beendet, wenn alle Kontextmöglichkeiten abgedeckt und keine Vorbedingungen offen sind.

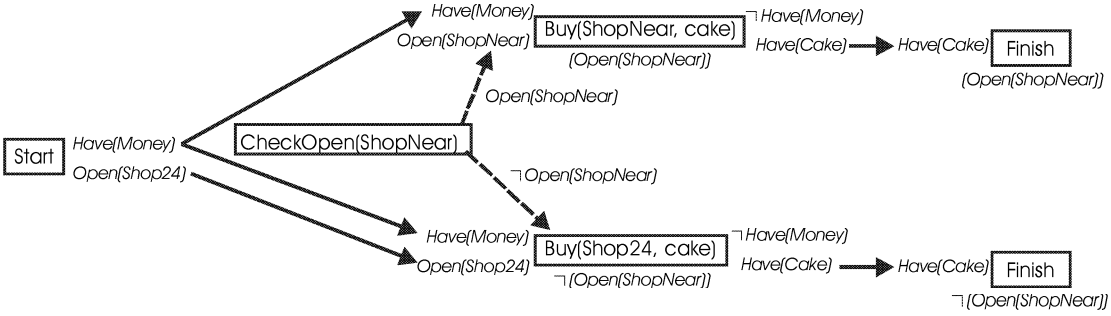
Was noch nicht besprochen wurde sind die Threats. Der CPOP besitzt zur Lösung eines Threats eine weitere Möglichkeit neben Promotion und Demotion: Conditioning. Threats zwischen verschiedenen Kontexten sind irrelevant, da zur Ausführungszeit nur ein Kontext des Plans abgearbeitet wird. Ein echter Threat kann gelöst werden, indem der causal link und die Operation zwischen denen der Threat besteht in verschiedene Kontexte gestellt werden. Betrachten wir folgendes Beispiel zur Veranschaulichung:

Angenommen das Ziel sei, einen Kuchen zu besitzen. Es gibt zwei Geschäfte in denen Kuchen gekauft werden kann. „Shop24" ist rund um die Uhr geöffnet. „ShopNear" nicht, dafür näher.

Wählt der Planer, den Kuchen in „Shop24" zu kaufen, ist das Problem trivial, da keine Threats auftreten. Also nehmen wir an, dass der Planer zunächst versucht des Ziel have(cake) zu erreichen indem er in „ShopNear" einkauft. Dann erreicht er nach einigen Iterationen seines Planungsalgorithmuses folgenden noch nicht fertigen Plan:



Es existiert ein Threat zwischen der Operation buy(Shop24, cake) und dem causal link zwischen dem Start-Step und buy(ShopNear, cake) und vice versa. Durch Anwendung von Conditioning, also durch Zuweisen verschiedener Kontexte zu den sich threatenden Operationen und causal links, erhält man folgenden Plan:



Es folgt der CPOP-Algorithmus in Pseudocode. Erweiterungen gegenüber dem POP sind die „Alternative context generation" und die Möglichkeit des „Conditioning" zur Threat-Auflösung.

```
function CPOP(initial, goals, operators) returns plan
    plan = MAKE-PLAN(initial, goals)
    loop do
        Termination:
        if there are no unsatisfied preconditions
            and the contexts of the finish steps are exhaustive
                then return plan

        Alternative contect generation:
        if the plans for existing finish steps are complete and have
            contexts C1 … Cn then add a new finish step with
            a context not (C1 or … or Cn)
            this becomes the current context

        Subgoal selection and addition:
        find a plan step Sneed with an open precondition c

        Action selection:
        choose a step Sadd from operators or STEPS(plan) that
            adds c or knowledge of c and has a context
            compatible with the current context
        if there is no such step
            then fail
        add Sadd --c--> Sneed to LINKS(plan)
        add Sadd < Sneed to ORDERINGS(plan)
        if Sadd is a newly added step then
            add Sadd to STEPS(plan)
            add Start < Sadd < Finish to ORDERINGS(plan)

        Threat resolution:
        for each step Sthreat that potentially threatens any causal link
            Sadd --c--> Sneed with a compatible context do
            choose one of
                Promotion: Add Sthreat < Si to ORDERINGS(plan)
                Demotion: Add Sj < Sthreat to ORDERINGS(plan)
                Conditioning:
                    find a conditional step Scond possibly befor both
                        Sthreat and Sj, where
                    1. the context of Scond is compatible with the
                        contexts of Sthreat and Sj;
                    2. the step has outcomes consistent with Sthreat
                        and Sj, respectively add conditional links
                        for the outcomes from Scond to Sthreat
                        and Sj augment and propagate the
                        contexts of Sthreat and Sj
            if no choice is consistent
                then fail
        end
    end
```

Der CPOP Algorithmus

2.4 ANMERKUNGEN

Wird im Pseudocode fail aufgerufen bedeutet das nicht unbedingt, dass der Planer keinen Plan findet. Es bedeutet nur, dass die ausgewählte Kombination von Operationen nicht zum Ziel führt. Wird also fail bei der Ausführung des Planers erreicht, tritt das Backtracking in Aktion und wählt, abhängig vom Backtrackingalgorithmus, an einer Stelle, an der zwischen mehreren

Möglichkeiten zu wählen war (im Pseudocode „choose“), eine andere Option. Erst wenn alle Kombinationen von Möglichkeiten nicht zum Erfolg führen, steht fest, dass kein Plan gefunden werden kann.

Ein besonders Augenmerk verdient meiner Ansicht nach noch der Zusammenhang zwischen Kontext und den Teilzielen eines Finish-Steps. Oberflächlich betrachtet könnte zum Beispiel eine Domäne, in der als Ziel eine Tür geschlossen werden soll definiert ist, der Planer allerdings nicht weiß ob die Tür zur Startzeit der Ausführung des Plans offen oder geschlossen ist. Der CPOP erstellt hierfür einen Plan mit zwei Finish-Steps, eins im Kontext „die Tür geschlossen“ und eins im Kontext – ja was eigentlich? Die Tür ist zwar am Anfang geöffnet aber am Ende geschlossen. Was wäre, wenn der Planer zum Schluss eine Aktion in den Plan einfügt, die später nochmals überprüft, ob die Tür denn wirklich offen oder geschlossen ist? Dann müsste es ein Finish geben mit dem Kontext „Tür ist offen“ aus der ersten Überprüfung und „Tür ist geschlossen“ aus der zweiten. Aber eine Bedingung und deren Negierung können nicht gleichzeitig wahr sein. Zur Aufklärung der Problems: Der CPOP geht von 2 fundamentalen Grundsätzen aus bei der Erstellung seiner Pläne: 1. Seine Operationen funktionieren immer, d.h. Effekte, die eine Operation verspricht, treten immer ein, wird die Operation ausgeführt. 2. Der Agent wird bei der Ausführung des Planes nicht gestört.

Setzt man dies Beides voraus, kann man ohne Probleme einem Finish seinen Kontext zuweisen wie definiert, denn ein CPOP wird niemals eine bereits durchgeführte Überprüfung wiederholen, selbst wenn sich deren Ergebnis ändern würde. Im obigen Beispiel würde also der Planer nach einmaliger Überprüfung der Tür immer wissen, ob sie offen oder geschlossen ist, da die beiden Grundsätze gelten und somit eine Statusänderung der Tür immer nur vom Agent selber verursacht werden kann, was eine wiederholte Überprüfung überflüssig macht. Somit kann eine Bedingung nicht-a() Teil des Kontexts und a() Vorbedingung eines Finish-Steps sein.

Interessant ist auch die Betrachtung der Erstellung eines Conditional Plans aus einer anderen Perspektive. Im Prinzip erstellt der Planer ein Programm, das der Agent zur Laufzeit ausführt. So betrachtet kann man den Planer vergleichen mit einem Programmierer, der ein Programm für einen Computer schreibt, das ein gegebenes Problem löst. Ein CPOP ist damit also auch einfach als Programmierer zu sehen. Fallunterscheidungen beherrscht er (If-Operator) und um eine while-Schleife kann man ihn noch erweitern, worauf an dieser Stelle nicht genauer eingegangen wird. Dies reicht aus, um jedes primitiv-rekursive Programm zu implementieren.

2.5 BEISPIEL

Als Anwendungsbeispiel für einen Conditional Planner betrachten wir folgende Domäne: Es soll ein Plan erstellt werden, mit dem ein Agent von zu Hause zu seiner Universität gelangt. Ihm stehen dazu folgende Operationen zur Verfügung:

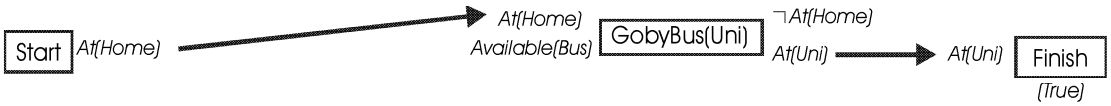
Op(ACTION: *GobyBus(x)*,
PRECOND: *At(y)* ∧ *Available(Bus)*,
EFFECT: *At(x)* ∧ ¬ *At(y)*")) , wobei x≠y

Op(ACTION: *Walk(x)*,
PRECOND: *At(y)*,
EFFECT: *At(x)* ∧ ¬ *At(y)*")) , wobei x≠y

Op(ACTION: *CheckAvailable(x)*,
PRECOND: *True*,
EFFECT: KnowsWhether("*Available(x)*"))

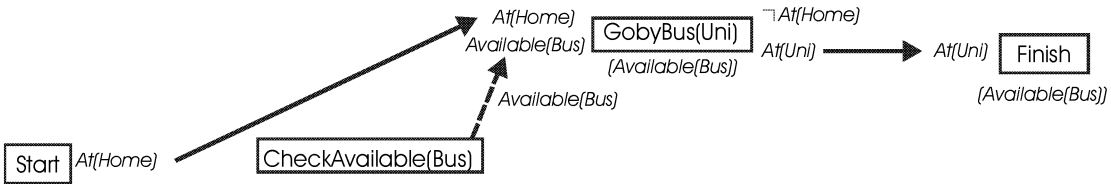
Der Planer beginnt mit folgendem Plan:

Der Start-Step hat als Effekt „At(Home)“ da sich der Agent zu Anfang zu Hause befindet. Der Finish-Step hat „At(Uni)“ als Vorbedingung, da das Ziel ist in der Uni zu sein. Kontext des Finish-Steps ist „True“, da noch keine Unterscheidungs-Steps im Plan integriert sind. Jetzt wird im Prinzip vorgegangen wie beim gewöhnlichen POP-Algorithmus, bis zu folgender Stelle:

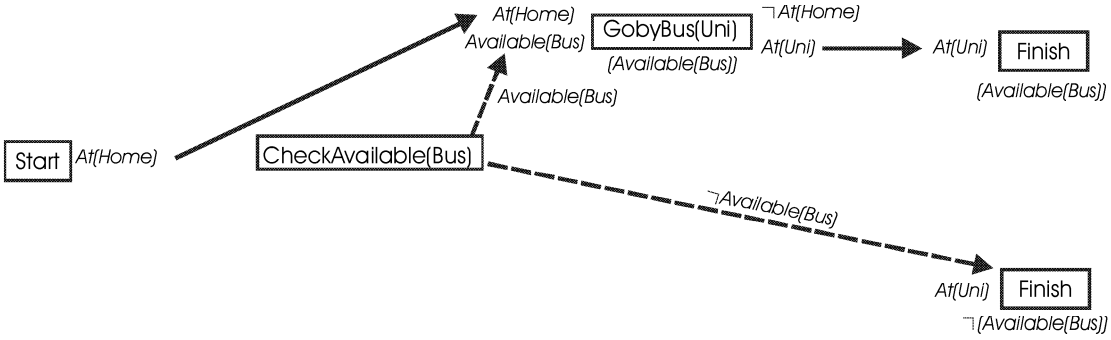


Der Planer hätte auch direkt „Walk“ anstelle von „GobyBus“ nehmen können um „At(Uni)“ zu erfüllen. Allerdings würde dann der CPOP-Plan genauso aussehen wie ein POP-Plan für dieses Problem. Außerdem wäre es nicht realistisch, jeden Tag zur Universität zu laufen, obwohl der Bus zur Verfügung steht. Wir nehmen also an, dass unser CPOP-Algorithmus die Vorzüge bestimmter Operationen erkennt und somit, wenn die Wahl frei steht, den Agenten eher Bus fähren als laufen lässt.

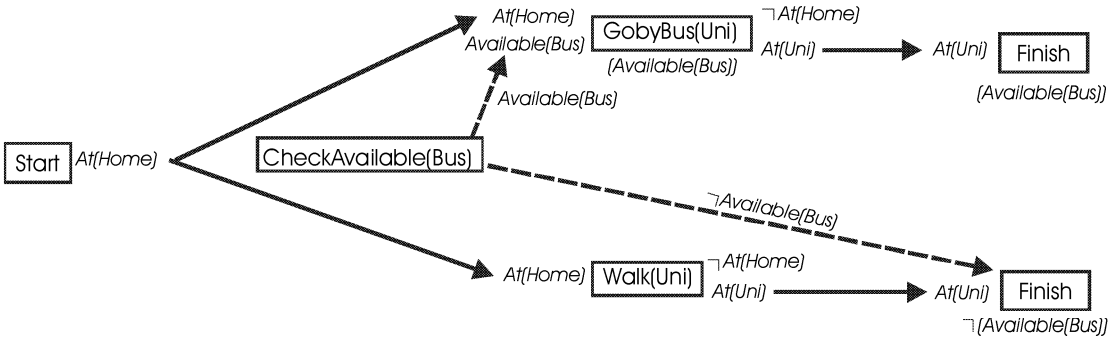
Um das Teilziel Available(Bus) zu erreichen, steht dem Planer eine Operation zur Verfügung: „CheckAvailable(Bus)“. Setzt er diese Operation ein, muss er die bereits im Plan enthaltenen Schritte, die als Vorbedingung „Available(Bus)“ haben, in den selben Kontext stellen. Der Plan sieht dann wie folgt aus:



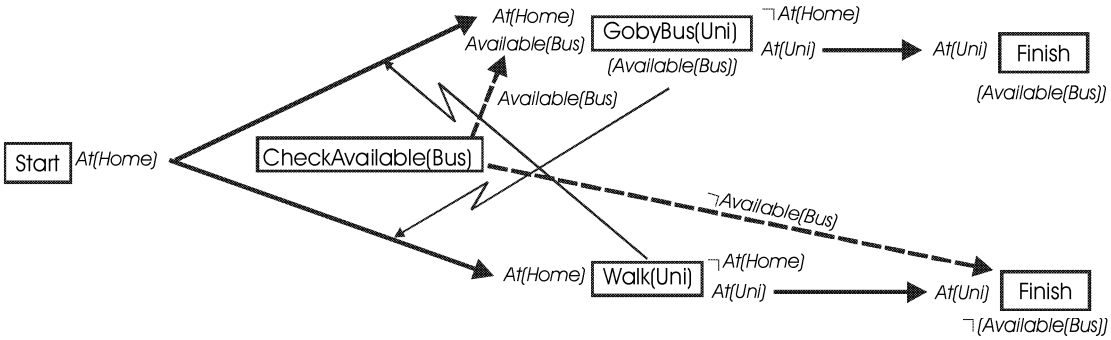
Anschließend wird die „Alternative context generation“ durchgeführt, so dass jeder Kontext im Plan abgedeckt ist:



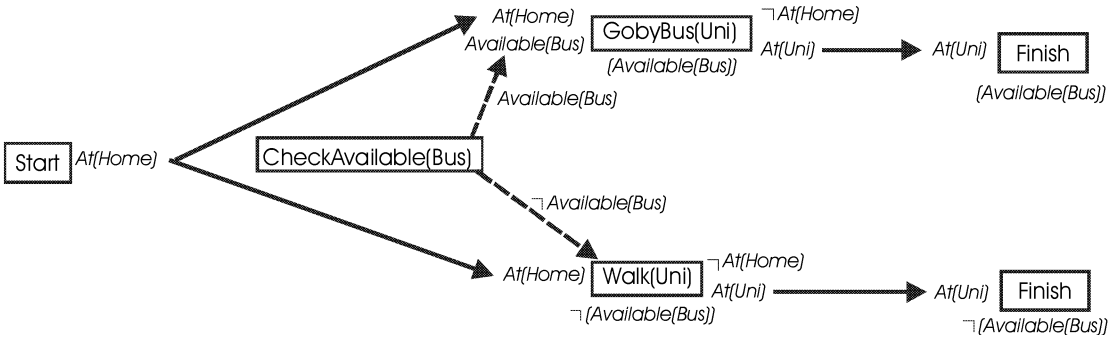
Nun wird der Plan um Schritte erweitert, die die Vorbedingungen dieses Finish-Steps in dessen Kontext erfüllen.



So sieht der Plan aus, wenn der Planer gerade vor der nächsten „Threat resolution"-Phase steht. Es bestehen zwei Threats in dem Plan (angedeutet durch die Blitze):



Ein POP würde diese versuchen durch Promotion oder Demotion zu lösen. Dem CPOP steht jedoch zusätzlich das Conditioning zur Verfügung. Dies angewandt ergibt folgenden vollständigen Plan:



3 INTEGRIERTE PLANUNG UND AUSFÜHRUNG

3.1 VORBEMERKUNG

Die bisherigen Planer gingen immer davon aus, dass der Agent alleine in einer Welt agiert und seine Operationen immer erfolgreich ausgeführt werden. Schlägt aufgrund dieser Annahme eine der Operationen eines Planes in der Realität fehl, so bekommt der Agent dies nicht mit. Stattdessen führt er seinen Plan einfach stur bis zum Ende aus und wird davon ausgehen, dass er Erfolg hatte.

Um dies zu vermeiden gibt es die Möglichkeit des „execution monitoring". Dies veranlasst den Agenten nach jeder Ausführung einer Operation zu überprüfen, ob die erwarteten Effekte eingetreten sind, die Operation also fehlerfrei verlief. Eine Alternative ist das „action monitoring". Hierbei wird vor der Ausführung der jeweils nächsten Operation deren

Vorbedingungen überprüft. Gelten sie nicht alle, schlug etwas fehl. Im Gegensatz zum „execution monitoring“ wird bei diesem Verfahren nicht unbedingt sofort ein Fehler bei der Ausführung einer Operation festgestellt, nämlich dann, wenn ein Teilziel, das die fehlgeschlagene Operation erreichen sollte keine Vorbedingung der nächsten Operation sondern erst einer späteren ist. Andererseits ermöglicht „action monitoring“ auf nicht vom Agent verursachte Veränderungen der Welt zu reagieren. Erreicht z.B. eine Operation eine für eine spätere Operation notwendige Vorbedingung so wird erkannt, wenn diese Bedingung nicht mehr wahr ist, wenn die spätere Operation zur Ausführung kommt.

3.2 BESCHREIBUNG DES SYSTEMS

Einen Schritt weiter geht folgender Ansatz: Planung und Ausführung werden nicht mehr getrennt durchgeführt, sondern in einem System gleichzeitig, zur Laufzeit des Systems. Es wird also, anschaulich gedacht, vor der Ausführung eines jeden Schrittes ein neuer Plan erstellt, ausgehend von der aktuellen Situation und ein Schritt davon ausgeführt. Somit kann auf alle Zwischenfälle, seien es andere Agenten die in derselben Welt agieren oder fehlerhafte Ausführung von Operationen, reagiert werden.

Dieses System benutzt grundlegend dafür einen POP. Allerdings wird aus Effizienzgründen nicht wirklich andauernd ein neuer Plan erstellt, sondern ein Plan, der der aktuellen Situation angepasst wird. Somit ist das System nur zu Beginn etwas länger mit der Erstellung eines Planes beschäftigt.

Der Trick ist nun, dass ein Agent nicht unbedingt warten muss, bis ein Plan komplett durchdacht ist um Operationen auszuführen. Stattdessen können Aktionen, die auf jeden Fall Bestandteil eines fertigen Planes wären, sofort ausgeführt werden. Aus der neuen Situation und dem Ergebnis kann man zudem evtl. Erkenntnisse für die weitere Planung erhalten (z.B. durch Sensing Actions). Ein Schritt wird als ausführbar eingestuft, wenn alle Vorbedingungen im Startschritt gelten, kein anderer Schritt unbedingt vor diesem ausgeführt werden muss und der Schritt keinen Threat in einem anderen causal link verursacht.

Nachdem also der POP zu Beginn ausgeführt wurde, wiederholt sich Folgendes, bis der Plan nur noch aus Start und Finish besteht und es keine offenen Teilziele mehr gibt:

1. Mögliche offene Teilziele im Plan werden durch causal links zu früher auszuführenden Schritten oder durch neu hinzugefügte Schritte, die dieses Teilziel erfüllen, verbunden und dadurch auftretende Threats durch Promotion und Demotion beseitigt.

2. Anschließend werden alle causal links die vom Start zu einem beliebigen Schritt führen und eine Aussage a() schützen, die nicht mehr in Start gilt, entfernt.

3. Ein causal link wird von Schritt b nach Schritt c entfernt und durch einen causal link von Schritt a nach Schritt c ersetzt (wobei er natürlich dieselbe Aussage schützt), wenn a vor b ausgeführt wird und dadurch keine zusätzlichen Threats entstehen (extending a causal link).

4. Nun werden alle redundant actions, d.h. alle Schritte, die mit keinem causal link verbunden sind und somit überflüssig für die Ausführung des Plans sind, entfernt.

5. Zuletzt wird ein ausführbarer Schritt gesucht, ausgeführt und vom Plan, neben allen causal links von und zu diesem Schritt, entfernt.

Wichtig ist hierbei die Reihenfolge der Aktionen. Zum Beispiel ein Ausführen von Punkt 3 vor Punkt 1 würde in manchen Fällen in umfangreicheren Plänen resultieren.

Anschließend der Algorithmus im Pseudocode. Der erste Block enthält einige Initialisierungen, z.B. wird der aktuelle Situationsstatus ermittelt und zu Beginn ein initialer Plan erstellt. Die vorangegangenen fünf Punkte spiegeln sich direkt in den folgenden Absätzen wieder:

function SITUATED-PLANNING-AGENT(*percept*) **returns** an *action*
static: *KB*, knowledge base (includes action descriptions)
p, a plan, initially *NoPlan*
t, a counter, initially 0, indicating time
G, a goal

TELL(*KB*,MAKE-PERCEPT-SENTENCE(*percept*, *t*))
current = STATE-DESCIPTION(*KB,t*)
EFFECTS(START(*p*)) = *current*
if *p* = *NoPlan* **then**
G = ASK(*KB*,MAKE-GOAL-QUERY(*t*))
p = MAKE-PLAN(*current*, *G*, *KB*)
action = *NoOp* (the default)

Termination:
if there are no open preconditions and p has no steps other than START and FINISH **then**
p = *NoPlan* and skip remaining steps

Resolving standard flaws:
resolve any open condition by adding a causal link from any existing possibly prior step or a new step
resolve potential threats by promotion or demotion

Remove unsupported causal links:
if there is a causal link START $\xrightarrow{c}$ *S* protecting a proposition *c* that no longer holds in START **then**
remove the link and any associated bindings

Extend causal links back to earliest possible step:
if there is a causal link *Sj* $\xrightarrow{c}$ *Sk* such that another step *Si* exists with *Si* < *Sj* and the link *Si* $\xrightarrow{c}$ *Sk* is safe **then**
replace *Sj* $\xrightarrow{c}$ *Sk* with *Si* $\xrightarrow{c}$ *Sk*

Remove redundant actions:
remove any step *S* that supplies no causal links

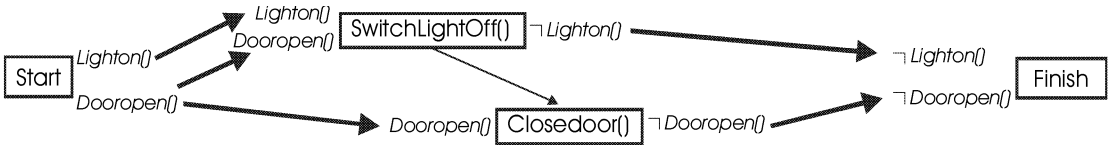
Execute actions when ready for execution:
if a step *S* in the plan other than FINISH satisfies the following:
(a) all preconditions satisfied by START;
(b) no other steps necessarily between Start and *S*; and
(c) *S* does not threaten any causal link in *p* **then**
add ordering constraints to force all other steps after *S*
remove *S* from *p*, and all causal links to and from *S*
action = the action in *S*

TELL(KB,MAKE-ACTION-SENTENCE(*action*,*t*))
t = *t* +1
return ***action***

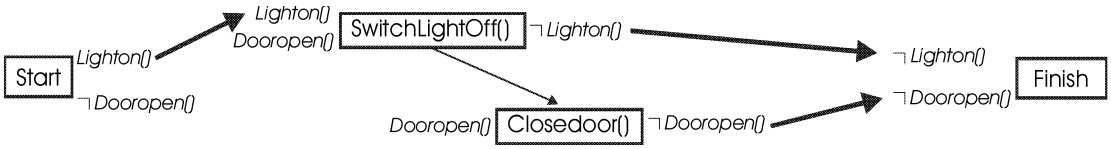
Ein Agent mit integriertem Planungssystem.

3.3 BEISPIEL

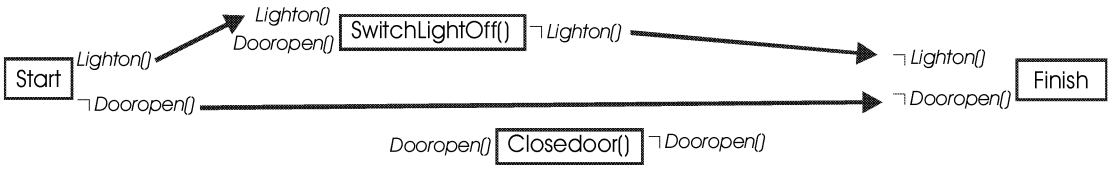
Sei die Domäne wie folgt modelliert: Das Ziel ist es, in einem Raum das Licht auszuschalten und die Tür zu schließen. Zu Beginn ist das Licht an und die Tür ist offen. Zu beachten ist: Um das Licht ausmachen zu können, muss die Tür geöffnet sein. (Somit ist sichergestellt, dass der Agent nach Erreichen seines Ziels sich außerhalb des Raumes befindet. Darauf wird nicht weiter eingegangen.) Der Planer des Systems erstellt zunächst folgenden Plan:



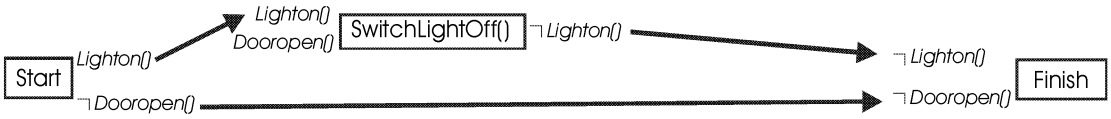
Das System könnte jetzt „SwitchLightOff()“ ausführen, da alle notwendigen Vorbedingungen vom Start-Step erfüllt werden, keine Schritte zwischen Start und „SwitchLightOff()“ ausgeführt werden müssen, und der Schritt keinen Threat verursacht. In dieser Art könnte das System alle Schritte des Plans der Reihe nach ausführen, und das Ziel würde erreicht. Aber nehmen wir an, ein anderer Agent verändert die Welt. Lassen wir zum Beispiel einen anderen Agenten die Tür schließen, bevor unser Agent seinen Schritt ausführt. Dann sieht der Plan nach der Phase „Remove unsupported causal links“ folgenderweise aus:



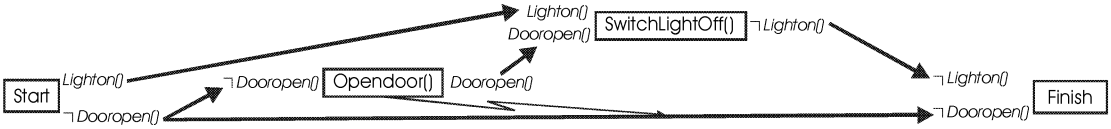
In der folgenden Phase „Extending causal links“ wird der Link von „Closedoor()“ zu Finish erweitert zu einem Link von Start zu Finish:



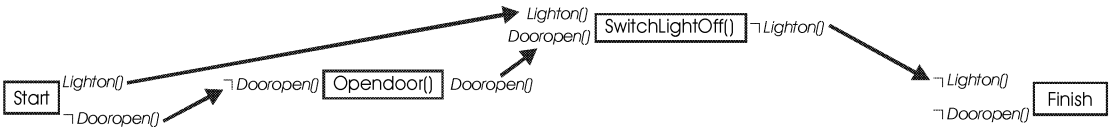
Closedoor() wurde damit zu einer „redundant action“, und wird in der nächsten Phase des Algorithmusses entfernt:



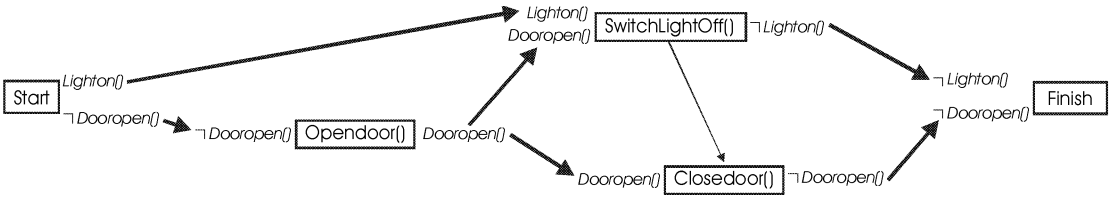
In der nächsten Iteration wird versucht die offene Vorbedingung „Dooropen()“ zu erfüllen. Dazu wird ein neuer Step eingefügt und es entsteht ein Threat.



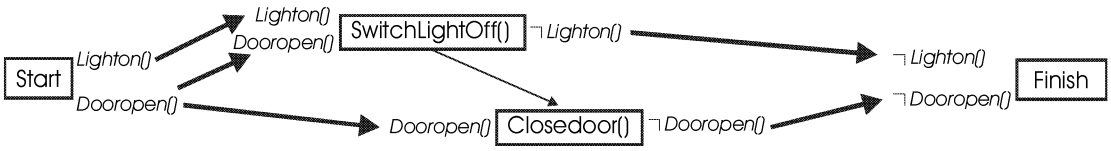
Dem System steht nun Promotion und Demotion zur Lösung des Threats zur Verfügung. Leider funktionieren in diesem Fall keine der beiden Möglichkeiten, da kein Schritt vor Start oder hinter Finish eingereiht werden kann. In dieser Situation tritt Backtracking in Aktion. Nehmen wir also einmal an, unser Backtrackingsystem wäre clever genug zu erkennen, dass der direkte Link von Start nach Finish entfernt werden muss. Dann erhalten wir:



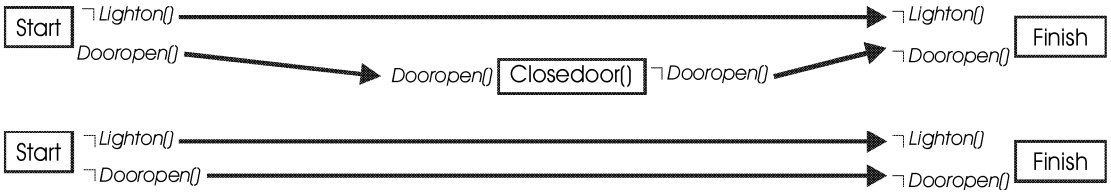
Da der direkte Link nicht funktioniert hat, probiert das System eine andere Möglichkeit aus, zum Beispiel:



„Opendoor()“ kann nun ausgeführt werden.



Wir erhalten den Plan, den das System sich zu Anfang überlegt hatte. Diesen kann es abarbeiten um das Ziel zu erreichen. Es hat erfolgreich auf Eingriffe anderer Agenten reagiert. Die nächsten Schritte sehen folgendermaßen aus, angenommen es greift kein anderer Agent ein:



Damit ist die Abbruchbedingung des Algorithmusses wahr und das Ziel erreicht.

4 REFLEKTION

Nachdem die beiden Ansätze von ihrem Vorgehen und ihrer Funktion besprochen wurden, gehe ich in diesem Kapitel einen Schritt weiter und erläutere einige Überlegungen in Bezug auf die Anwendung dieser Systeme, deren Vor- und Nachteile.

4.1 ERWEITERUNGEN GEGENÜBER EINES POP

Beide vorgestellten Techniken sind mächtiger im Finden eines Planes für ein gegebenes Ziel. Wir haben also nun Systeme, die selbstständig ein gegebenes Ziel in einer realen Domäne erreichen können. Dabei ermöglicht CPOP das Einfügen von Aktionen, die Informationen aus der Welt abfragen, wenn Informationen benötigt werden. Der planende Agent kann zudem auf unvorhergesehene Veränderungen in der Welt reagieren und immer noch zu seinem Ziel gelangen.

4.2 VERGLEICH CPOP MIT PLANENDEM AGENT

4.2.1 KOMPLEXITÄT

Der POP ist im Vergleich zu einem „search through situation space“ schon ein gewaltiger Geschwindigkeitsgewinn. CPOP benötigt zur Berechnung eines Plans hingegen mehr Zeit. Er

muss alle Fallunterscheidungen betrachten und mehrere Finish-steps mit ihrem jeweiligen Kontext berücksichtigen. Dies kann unter Umständen zu einer inakzeptablen Planungszeit führen. Betrachte man folgendes Beispiel:

Der Agent besitze 10 Schlüssel und soll damit 3 Türen aufschließen. Dabei passt jeweils ein Schlüssel in ein Schloss und die übrigen sieben passen in keines. Es steht eine Operation

Op(ACTION: *try(door,key),*
PRECOND: *True*,
EFFECT: KnowsWhether("*success(door,key)*"))

zur Verfügung. "success(door,key)" gilt, wenn Schlüssel key in Tür door passt.

CPOP baut nun einen Plan, in der er allen möglichen Schlüssel/Tür-Kombinationen einen Kontext zuweist und einen Plan für diesen Kontext erstellt. Aus der Kombinatorik folgt, dass der gesamte Plan

$$\frac{n!}{(n-k)!} = \frac{10!}{(10-3)!} = 10 * 9 * 8 = 720$$

Schritte beinhaltet. An der Formel erkennt man auch, dass das Problem über-exponentiell steigt mit der Anzahl der zur Verfügung stehenden Schlüssel. Damit ist das Problem, in Bezug zum CPOP, Element von O(n!).

Dieses Beispiel zeigt uns also, dass vom CPOP erstellte Pläne unter Umständen recht umfangreich werden können, obwohl es sich um ein gar nicht so kompliziertes Problem handelt.

Beauftragt man jedoch einen planenden Agenten mit demselben Auftrag, erhält man ein ganz anderes Ergebnis. Sein Planer wird, ausgehend von dem Startplan mit einem Start- und einem Finish-Step eine Operation sagen wir try(1,1) einfügen und diese ausführen, da sie nach den zuvor definierten Regeln eine ausführbare Operation ist. Gelingt ihm dies, ist ein Teilziel vom Finish, also eine Tür zu schließen, gelungen. Erst wenn dies nicht der Fall ist, wird überlegt, wie das Teilziel sonst noch erreicht werden könnte, z.B. durch try(1,2) usw.

So betrachtet ist die Anzahl der im Plan während der Ausführung erschienenen Schritte im schlechtesten Fall

$$n + (n-1) + (n-2) = 10 + 9 + 8 = 27$$

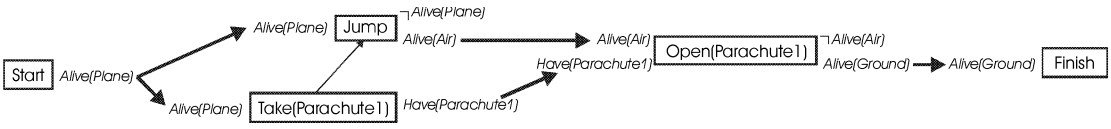
Man sieht leicht, dass für dieses Problem also ein planender Agent die effizientere Wahl zwischen den beiden Systemen ist. Der Trick liegt einfach darin, dass der weitere Verlauf der Aktionen erst geplant wird, wenn eine Bedingung, die den Verlauf entscheidet, ermittelt worden ist. Somit wird (größtenteils) nur der Teil eines entsprechenden CPOP-Plans berücksichtigt, der tatsächlich ausgeführt wird. Überflüssige Pfade für die aktuelle Situation werden nicht behandelt.

4.2.2 BETRACHTUNG VERSCHIEDENER FÄLLE

Nach dem vorangegangenen Absatz könnte man zu dem Schluss kommen, dass ein planender Agent immer effizienter und besser ist als ein CPOP. Ist das jedoch wirklich der Fall? Um diese Frage zu beantworten betrachten wir folgendes Szenario:

Der Agent befindet sich zu Beginn in einem Flugzeug und soll abspringen und unten auf dem Boden heil ankommen. In dem Flugzeug befindet sich ein Fallschirm und ein Ersatzfallschirm.

Ein planender Agent würde zunächst seinen Planer starten und solch einen Plan erstellen:



Dabei bedeutet Alive(x), dass der Agent lebt und sich an Position x befindet. Als Erstes führt er Take(Parachute1) aus und anschließend Jump und zum Schluss Open(Parachute1). Idealerweise käme er so ans Ziel. Aber was passiert, wenn der Fallschirm sich nicht öffnet? Nun ja, nach dem fehlgeschlagenen Versuch den Fallschirm zu öffnen, wird der Agent mit einem Plan mit nur Start und Finish und dem unerfüllten Teilziel Alive(ground) konfrontiert sein. Ihm stehen auch keinerlei Aktionen zur Verfügung um sein Ziel noch zu erreichen. Somit wird er einen Fehlschlag, sein Ziel zu erreichen, melden, was in diesem Fall zugegebenermaßen recht fatale Folgen mit sich zieht.

Anders hingegen der mit einem CPOP erstellte Plan. Er kann einen Plan erstellen, indem er den Fall, dass der Parachute1 nicht aufgeht berücksichtigt. Da die Planung bei ihm vollständig vor der Ausführung irgendeiner Aktion durchgeführt wird, wird er für den Fall, dass Parachute1 zu öffnen fehlschlägt, noch den Reservefallschirm Parachute2 mitnehmen und somit praktisch eher sein Ziel erreichen als der planende Agent. Ein kleines Manko gibt es noch: Ein Agent der einen CPOP-Plan ausführt geht immer davon aus, dass er sein Ziel erreicht, da der Plan ja für alle Fälle ausgelegt ist. In diesem Fall würde ein Agent also nach einem Fehlschlag der Aktion den 2. Fallschirm zu öffnen gar nicht mitbekommen (ohne monitoring), dass er sein Ziel nicht erreichen wird. Er wird unwissend zerschellen.

Zu beachten ist dabei, dass die Domänen jeweils für den einzelnen Agenten leicht unterschiedlich definiert sein müssen damit dies so funktioniert:

In der Domäne die dem CPOP übergeben wird muss die Operation Open(x) so definiert sein, dass ein Effekt KnowsWeather(„Open(x)“) ist, damit nicht von Vornherein der Planer davon ausgehen kann, dass diese Funktion erfolgreich sein wird. Ferner muss noch berücksichtigt werden, dass ein Öffnen des 2. Fallschirms auf jeden Fall funktioniert, auch wenn es in der Realität nicht der Fall sein muss. Sonst würde ein CPOP gar keinen Plan liefern zur Lösung des Problems, da er keinen Plan finden kann, der für alle Fälle funktioniert.

Solch eine Domäne kann aber nicht dem planenden Agenten übergeben werden. Er würde einfach gar keinen Plan finden, da die Operation Open(x) nicht unbedingt erfolgreich sein muss und der Planer dieses Agenten keine Fallunterscheidung beherrscht. Domänen für diesen Agenten müssen Aktionen enthalten, mit denen er auf jeden Fall sein Ziel erreichen kann. Die beim CPOP berücksichtigte Fallunterscheidung wird durch eine dynamische Anpassung des auszuführenden Planes erreicht. In vielen Fällen funktioniert das auch, evtl. löst der Agent dann das Problem nicht auf die effektivste Weise (er muss vielleicht noch einmal irgendwohin zurück und etwas holen, das er doch noch benötigt, etc.). Doch in diesem Fall ist er in eine Sackgasse gerannt. Er realisiert, dass er so sein Ziel nicht erreicht und bräuchte jetzt den 2. Fallschirm, nur an den kann er jetzt nicht mehr herankommen.

Man kann also sagen, dass keiner der beiden Algorithmen der Bessere ist. Vielmehr sind sie unterschiedlich gut für verschiedene Probleme geeignet.

4.4 OFFENE PROBLEME

4.4.1 DOMÄNENMODELLIERUNG

Wie schon in der Einleitung erwähnt sind Domänen abstrakte Repräsentationen einer realen Welt, in der ein Agent Probleme lösen soll. Wie schon in dem Abschnitt zuvor kurz angedeutet

ist eines der größten Probleme, die Modellierung einer Domäne. Dies hat sich bereits daran gezeigt, dass je nachdem welches System benutzt wird unterschiedliche Domänen modelliert werden müssen, damit das Ziel erreicht wird. Somit muss man nicht nur entscheiden, wie stark man von der Realität abstrahiert, sondern welche Aktionen, Operationen, etc. man als verlässlich und welche als kritisch einstuft, obwohl in der Realität praktisch nichts 100%ig verlässlich ist. Die hier vorgestellten Algorithmen funktionieren ohne diese Einschränkungen nicht.

Was wahrscheinlich noch eine Verbesserung bringen würde, wäre eine Bewertung von Aktionen, z.B. die Wahrscheinlichkeiten, mit denen bestimmte Aktionen erfolgreich sind. Somit würde ein Planer Pläne erstellen, die mit höchster Sicherheit ans Ziel führen, indem er eher zuverlässige Schritte in den Plan integriert.

4.4.2 WAS NOCH ZU TUN BLEIBT

Systeme die auf den hier beschriebenen Agenten beruhen tragen immer das Risiko des Fehlschlags mit sich, obwohl das Problem offensichtlich hätte gelöst werden können. Was also noch aussteht ist eine Erweiterung dieser hier vorgestellten Techniken oder eine Neuschaffung eines Systems, das auf wesentlich mehr Eventualitäten unserer Welt reagieren kann. Wir haben gesehen, dass selbst der so weit fortgeschrittene planende Agent in der Fallschirmsituation versagt, obwohl für uns Menschen das Problem gar kein Thema gewesen wäre. Eine Idee wäre ein planender Agent mit integriertem CPOP. Die Probleme dabei im Detail müsste man sich noch überlegen, aber oberflächlich betrachtet ist leicht einzusehen, dass dieser Agent in der Fallschirmsituation den Ersatzfallschirm mitnehmen würde.

Russell, S. and Norvig, P.: *Artificial Intelligence – A Modern Approach*. Prentice-Hall International, 1995 (Chapter 13)