Hierarchical Task Network Planning
Seminar „Planen“
Betreut durch Prof. Dr. Jürgen Dix

Lars Schittly
ls@tu-clausthal.de

14.06.2005

Inhaltsverzeichnis

Kapitel 1

Einleitung

Hierarchical Task Network - HTN Planning ist wie Klassisches Planen bei, dem jeder Zustand der Welt durch eine Menge von Atomen dargestellt wird und jede Aktion mit einem deterministischen Zustandsübergang korrespondiert.
HTN Planer unterscheiden sich von Klassischen Planern in dem was und wie sie es planen. Es ist nicht das Ziel, einen Zielzustand zu erreichen, sondern eine Menge von Aufgaben auszuführen.
Ein HTN Planer erhält hierbei als Eingabe Operatoren (ähnlich denen des Klassischen Planens) und Methoden, welche beschreiben, wie Aufgaben in Teilaufgaben zerlegt werden. Planen geschieht durch rekursives Zerlegen von Aufgaben in kleinere Teilaufgaben. Dies geschieht solange, bis primitive Aufgaben erreicht sind, die sich direkt durch Planungsoperatoren ausführen lassen. Die Kombination aus Operatoren und Methoden stellt das Wissen da, was man im Planungssystem integriert.
Nun ein einführendes Beispiel für eine DWR Problem.

1.1 Beispiel

In diesem Beispiel sollen die Container von den Paletten $p1a$, $p2a$ und $p3a$ auf die Paletten $p1c$, $p2c$ und $p3c$ bewegt werden. Die Anordnung der Container soll auf den Ziel-Paletten erhalten bleiben.

Eine mögliche Vorgehensweise wäre die Container einzeln auf Palette b zwischenzustapeln um sie von dort dann nach Palette c zu verlegen. Hier lassen sich Aufgaben wie 'move stack' definieren, wobei man move stack dann in eine Teilaufgabe zerlegen könnte, die darin besteht, solange auf der Startpalette noch Container stehen, den obersten Container zu nehmen und auf der Zielpalette abstellen.
Eine HTN Beschreibung würde die DWR Operatoren beinhalten, den Startzustand, die Aufgabe, die drei Stapel so zu bewegen, dass ihre Reihenfolge

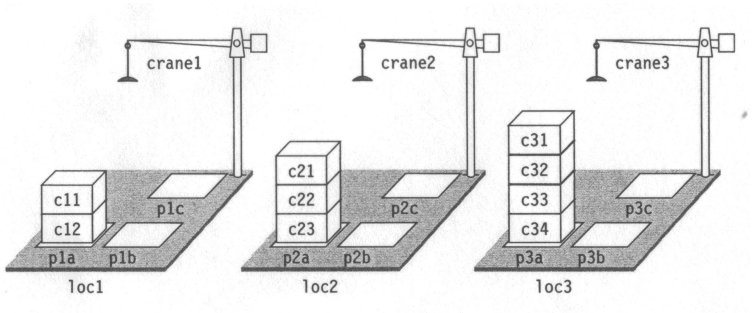


Abbildung 1.1: Startzustand des Planungsproblems

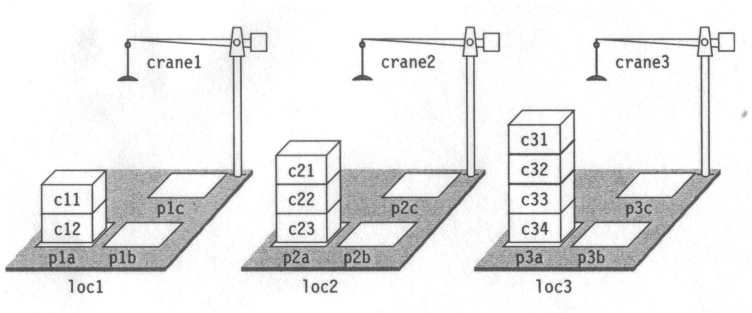


Abbildung 1.2: Ziel des Planungsproblems

erhalten bleibt und eine formale Beschreibung der eben genannten Vorgehensweise.
Im Folgenden werde ich zuerst auf das Simple Task Network eingehen, eine vereinfachte Form von HTN. Dabei werde ich erst neue Definitionen einführen und auch auf Bekanntes aus dem Klassischen Planen zurückgreifen. Darauf folgen dann Lösungsmöglichkeiten mit Beispielen und schließlich Hierarchical Task Networks. Dieses beinhaltet unter anderem auch Vergleiche zu anderen Planungssystemen, die Mächtigkeit von HTN und Erweiterungen. Am Ende folgt mit einer kurzen Zusammenfassung von Emergency Evacution Planning ein Blick auf ein sich noch in der Entwicklung befindendes System für Evakuierungsprozesse, welches HTN verwendet.

Kapitel 2

Simple Task Network

Beim Simple Task Network - STN (oder auch nur Task Network) handelt es sich um eine vereinfachte Version von Hierarchical Task Network. Die aus dem Klassischen Planen bekannten Definitionen von *terms*, *literals*, *operators*, *actions* und *plans* lassen sich in das Simple Task Network übernehmen. $\gamma(s, a)$ ist das Ergebnis einer Aktion a auf einen Zustand s und entspricht somit der Funktion des Klassischen Planens.
Gegenüber dem Klassischen Planen kommen *tasks*, *methods* und *task networks* neu hinzu, die für die Definition eines Planungsproblems und dessen Lösung benötigt werden.

2.1 Aufgaben und Methoden

Eine neue Art von Symbol ist das *task symbol*. Jedes Operator Symbol ist *task symbol* und zusätzlich gibt es noch so genannte *nonprimitiv task symbols*. Ein *task* ist ein Ausdruck der Form $t(r_1, \ldots, r_k)$, wobei t das *task symbol* ist und die r_i sind *terms*.
Ein *task* ist *ground*, wenn alle terms ground sind, ist dies nicht der Fall, ist er unground. Eine Aktion $a = (name(a), precond(a), effects(a))$ führt einen ground primitiv task t in einem Zustand s aus, wenn $name(a) = t$ und die Aktion a auf den Zustand s anwendbar ist.

2.1.1 Definition Simple Task Network

Ein Simple Task Network ist ein gerichteter Digraph $w = (U, E)$ mit den Knoten U und Kanten E. Jeder Knoten enthält einen *task* t_u. Der Graph w ist ground, wenn alle tasks t_u ground sind, andernfalls ist der Graph *unground*. Analog ist w *primitiv*, wenn alle tasks t_u primitiv sind, andernfalls ist w *nonprimitiv*.
Die Kanten E im Graph w definieren eine Teilordnung von U. Wenn ein Pfad von u nach v existiert, ist u ein Vorgänger von v ($u \prec v$). Falls diese

Teilordnung total ist, ist der Graph w total geordnet. In diesem Fall wäre eine Notation als Graph überflüssig, da man anstelle des Graphen auch die Folge der *tasks* schreiben könnte: $w =< t_1, \ldots, t_k >$. Dabei entspricht t_1 der Aufgabe im ersten Knoten, t_2 der im zweiten, usw.

2.1.2 Beispiel

In einer DWR Domäne seien $t_1 = take(crane2, loc1, c1, c2, p1)$ und $t_2 = put(crane2, loc2, c3, c4, p2)$. Sei $t_3 = move - stack(p1, \mathrm{q})$, wobei *move-stack* ein nichtprimitives task symbol ist. Dann sind t_1 und t_2 primitive tasks und t_3 ist ein nonprimitiv task.
Seien u_1, u_2 und u_3 Knoten und $t_i, i = \{1, 2, 3\}$ deren Aufgaben. Sei $w_1 = (\{u_1, u_2, u_3\}, \{(u_1, u_2), (u_2, u_3)\})$ und $w_2 = (\{u_1, u_2\}, \{(u_1, u_2)\})$. Dann sind w_1 und w_2 task networks. Da w_2 total geordnet, ist kann man es auch als $w_2 =< t_1, t_2 >$ schreiben. Hier ist w_2 auch ground und primitiv, daher korrespondiert es mit dem Plan $< a_1, a_2 >$, bei dem die Namen der Aktionen a_1 und a_2 t_1 und t_2 sind. Man kann normalerweise die Namen von Aktionen verwenden um auf die Aktionen selber zu verweisen; also würde der Plan <take(crane2, loc1, c1, c2, p1), put(crane2, loc2, c3, c4, p2)> lauten.

2.1.3 Definition STN Methoden

Eine STN Methode ist ein 4-Tupel
$m = (name(m), task(m), precond(m), network(m))$
Der Name der Methode $name(m)$ ist ein syntaktischer Ausdruck der Form $n(x_1, \ldots, x_k)$, bei dem n ein eindeutiges Methodensymbol ist und x_i alle Variablen Symbole sind, die irgendwo in der STN Methode m auftauchen. $task(m)$ ist eine nichtprimitive Aufgabe. $precond(m)$ ist eine Menge von Literalen, die alle erfüllt sein müssen, damit die Methode m angewendet werden kann. $network(m)$ ist ein *task network*, dessen tasks die *subtasks* von m genannt werden.
name(m) hat denselben Zweck wie im Klassischen Plan ein Planungsoperator (er lässt uns eindeutig auf Ersetzunginstanzen der Methoden verweisen, ohne dass man die Vorbedingungen und Effekte explizit angeben muss). task(m) gibt die Art der Aufgabe an, auf die die Methode m angewendet werden kann.
Wenn das Netzwerk total geordnet ist, so ist auch die m total geordnet und man kann wieder eine vereinfachende Schreibweise wählen.

2.1.4 Beispiel

Hier sieht man nun eine formale Beschreibung des Bespiels aus der Einleitung:

```
take-and-put(c, k, l1, l2, p1, p2, x1, x2):
    task:      move-topmost-container(p1, p2)
    precond:   top(c, p1), on(c, x1),    ; true if p1 is not empty
               attached(p1, l1), belong(k, l1),   ; bind l1 and k
               attached(p2, l2), top(x2, p2)      ; bind l2 and x2
    subtasks:  ⟨take(k, l1, c, x1, p1), put(k, l2, c, x2, p2)⟩

recursive-move(p, q, c, x):
    task:      move-stack(p, q)
    precond:   top(c, p), on(c, x)    ; true if p is not empty
    subtasks:  ⟨move-topmost-container(p, q), move-stack(p, q)⟩
               ;; the second subtask recursively moves the rest of the stack

do-nothing(p, q)
    task:      move-stack(p, q)
    precond:   top(pallet, p)    ; true if p is empty
    subtasks:  ⟨⟩   ; no subtasks because we are done

move-each-twice()
    task:      move-all-stacks()
    precond:     ; no preconditions
    network:     ; move each stack twice:
               u1 = move-stack(p1a,p1b), u2 = move-stack(p1b,p1c),
               u3 = move-stack(p2a,p2b), u4 = move-stack(p2b,p2c),
               u5 = move-stack(p3a,p3b), u6 = move-stack(p3b,p3c),
               {(u1, u2), (u3, u4), (u5, u6)}
```

2.1.5 Definition Anwendbarkeit

Eine Methodeninstanz m ist *applicable* (anwendbar) in einem Zustand s wenn $precond^+(m) \subseteq s$ und $precond^-(m) \cap s = \emptyset$
Zum Planen sind wir daran interessiert Methodeninstanzen zu finden, die im aktuellen Zustand anwendbar und für auszuführende tasks *relevant* sind.

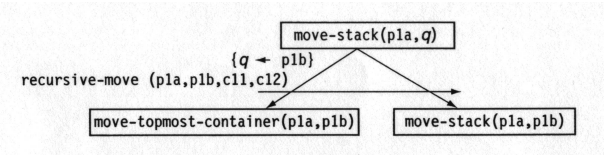
2.1.6 Definition *relevant*

Sei t ein task und m eine Methodeninstanz (egal ob ground oder unground), wenn eine Substitution σ existiert, so dass $\sigma(t) = task(m)$ gilt, dann ist m *relevant* für den task t und die Zerlegung von t durch m unter σ ist $\delta(t, m, \sigma) = network(m)$.
Falls m total geordnet ist, ließe sich die Zerlegung auch als $\delta(m, t, \sigma) = subtasks(m)$ schreiben.

2.1.7 Beispiel

Sei t der nichtprimitive task move-stack(p1a, q), s der Startzustand aus dem Beispiel in der Einleitung und m die Methodeninstanz recursive-move(p1a,

p1b, c11, c12). Dann ist m anwendbar auf s, *relevant* für t unter der Substitution $\sigma = \{\text{q} \leftarrow p1b\}$ und wird zerlegt in $\delta(t, m, \sigma) =< move - topmost - container(p1a, p1b), move - stack(p1a, p1b) >$
Auf diesem Bild erkennt man, dass sich die Zerlegung auch als AND-branch



darstellen lässt.
Hier wird gezeigt, wie move-stack(p1a, q) zerlegt wird, wobei die Methodeninstanz recursiv-move(p1a, p1b, c11, c12)verwendet wird. Die Verzweigung ist mit der Ersetzung und dem Namen der Methodeninstanz beschriftet. Der rechtsgerichtete Pfeil gibt die Reihenfolge der Teilaufgaben an.
Wenn wir eine Methode m benutzen um einen task t zu zerlegen, ist gewöhnlich dieser task Teil einen Knotens u im Netzwerk w. In diesem Fall werden wir ein neues *task network* $\delta(w, u, m, \sigma)$ bekommen, welches noch definiert wird.

2.1.8 Definition

Sei $w = (U, E)$ ein task network, u ein Knoten in w, der keine Vorgänger besitzt, und m eine Methode, die relevant für t_u unter einer Substitution σ ist. Sei $succ(u)$ die Menge aller direkten Nachfolger von u, $succ(u) = \{u' \in U | (u, u') \in E\}$. Bei $succ_1(u)$ sei die Menge aller direkten Nachfolger von u für die u der einzige Vorgänger ist. Sei (U', E') das Ergebnis der Entfernung von u und allen Kanten die u enthalten. Sei (U_m, E_m) eine Kopie des network(m). Wenn (U_m, E_m) nicht leer ist, dann ist das Ergebnis der Zerlegung von u in w durch m unter σ eine Menge von task networks:
$\delta(w, u, m, \sigma) = \{(\sigma(U' \cup U_m), \sigma(E_v)) | v \in subtasks(m)\}$
mit $E_v = E_m \cup (U_m \times succ(u)) \cup \{(v, u') | u' \in succ_1(u)\}$
sonst ist $\delta(w, u, m, \sigma) = \{(\sigma(U'), \sigma(E'))\}$

2.2 Probleme und Lösungen

2.2.1 Definition Planungsdomäne

Eine STN Planungsdomäne ist ein paar $D = (O, M)$ wobei O eine Menge von Operatoren ist und M eine Menge von Methoden. D ist eine total geordnete Planungsdomäne wenn M total geordnet ist.

2.2.2 Definition Planungsproblem

Ein STN Planungsproblem ist ein 4-Tupel $P = (s_0, w, O, M)$, wobei s_0 der Startzustand und w ein task network ist. $D = (O, M)$ ist eine STN Planungsdomäne.

2.2.3 Definition Lösung für ein Planungsproblem

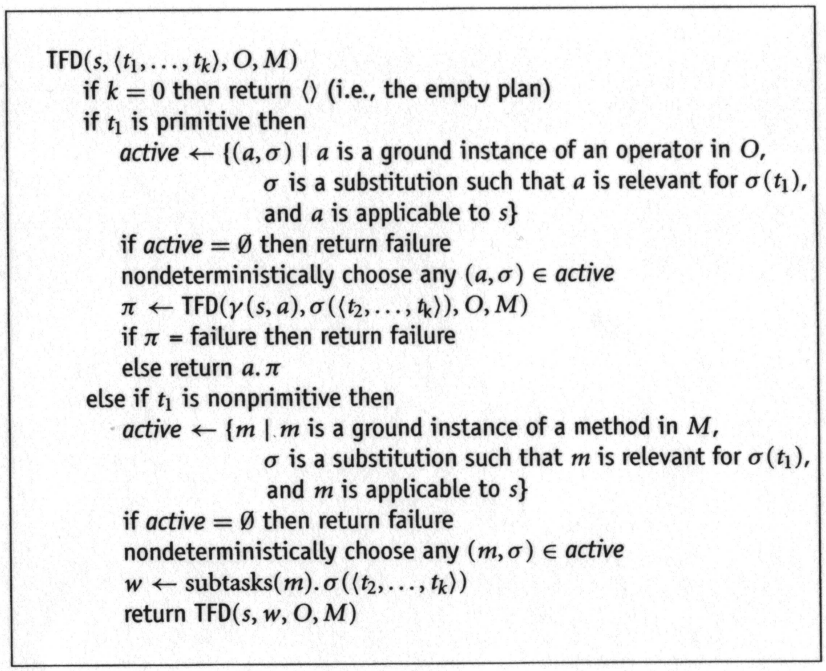
Sei $P = (s_0, w, O, M)$ ein Planungsproblem.
Wenn w leer ist, ist Π eine Lösung für P, wenn es selbst auch leer ist.
Wenn w nicht leer ist, und die erste Aktion auf den Startzustand angewendet werden kann gilt:
$\Pi =< a_2, \ldots, a_n >$ und $P' = (\gamma(s_0, a_1), w - \{u\}, O, M)$ (P' ist das Planungsproblem welches man durch Ausführen der ersten Aktion des Plans erhält und den entsprechenden Knoten entfernt).
Wenn der Knoten zu der Aktion nonprimitive ist und Vorgänger besitzt, existiert eine Lösung, wenn es relevante Methoden für den Knoten gibt und sich das task network so zerlegen lässt, dass ein primitiv task node ohne Vorgänger entsteht.

2.3 Total-Order STN Planning

Die Total-Order Forward Decomposition (TFD) Prozedur kann vollständig sortierte STN Probleme lösen. Der Algorithmus basiert direkt auf der Definition von STN und ist sowohl sound als auch complete für vollständig sortierte STN Planungsprobleme. Er liefert also nur echte Lösungen und findet eine Lösung, wenn eine existiert.
Wie forwardsearch berücksichtigt TFD nur Aktionen, deren Vorbedingungen für den aktuellen Zustand erfüllt sind. Und es werden, wie beim backwardsearch, nur relevante Operatoren benutzt. Durch diese Kombination wird die Effizienz der Suche gesteigert.
TFD erzeugt Aktionen in der Reihenfolge, in der sie ausgeführt werden (wie beim forwardsearch). Jedes mal, wenn eine Aufgabe erfüllt werden soll, ist bereits bis zu dieser Aufgabe geplant, wodurch der aktuelle Zustand der Welt bekannt ist.
Bei der Rückwärtssuche können unnötig viele ground instances eines Operators entstehen, TFD kann unnötig viele ground instances der Methoden erzeugen. Dieses Problem kann durch 'liften' gelöst werden.

2.4 Partial-Order STN Planning

Für die TFD kann es notwendig sein neue Methoden zu erstellen, damit die totale Ordnung gegeben ist. Diese Methoden zu erstellen ist nicht immer



```
TFD(s, ⟨t_1, ..., t_k⟩, O, M)
    if k = 0 then return ⟨⟩ (i.e., the empty plan)
    if t_1 is primitive then
        active ← {(a, σ) | a is a ground instance of an operator in O,
                           σ is a substitution such that a is relevant for σ(t_1),
                           and a is applicable to s}
        if active = ∅ then return failure
        nondeterministically choose any (a, σ) ∈ active
        π ← TFD(γ(s, a), σ(⟨t_2, ..., t_k⟩), O, M)
        if π = failure then return failure
        else return a.π
    else if t_1 is nonprimitive then
        active ← {m | m is a ground instance of a method in M,
                      σ is a substitution such that m is relevant for σ(t_1),
                      and m is applicable to s}
        if active = ∅ then return failure
        nondeterministically choose any (m, σ) ∈ active
        w ← subtasks(m).σ(⟨t_2, ..., t_k⟩)
        return TFD(s, w, O, M)
```

Abbildung 2.1: Total-Order Forward Decomposition Prozedur für Total-Order STN Planning

einfach. Bei der Partial-Order Forward Decomposition (PFD) müssen die Methoden nicht alle nacheinander ausführbar sein.

2.4.1 Beispiel

In diesem Beispiel sollen die Container c1 und c2 von l1 nach l2 bewegt werden. Der Zerlegungsbaum ist auf Abbildung 2.3 dargestellt.

Nun wird das Problem geringfügig abgeändert, r1 ist jetzt in der Lage mehrere Container auf einmal zu bewegen. Abbildung 2.4 zeigt wie ein Zerlegungsbaum aussehen würde, bei dem r1 mehrere Container gleichzeitig transportiert. Dieser Zerlegungsbaum ist nicht mehr total geordnet und einzelne Teilaufgaben überschneiden sich.

Wie Methoden zum Lösen dieses Problems aussehen könnten, zeigt Abbildung 2.5.

Abbildung 2.6 zeigt die Partitial-Order Forward Decomposition Prozedur, die eine Verallgemeinerung der TFD Prozedur ist, so dass STN Planungsprobleme nicht total geordnet sein müssen um sie lösen zu können. PFD ist eine direkte Implementierung der Definition einer Lösung eines STN Planungs-

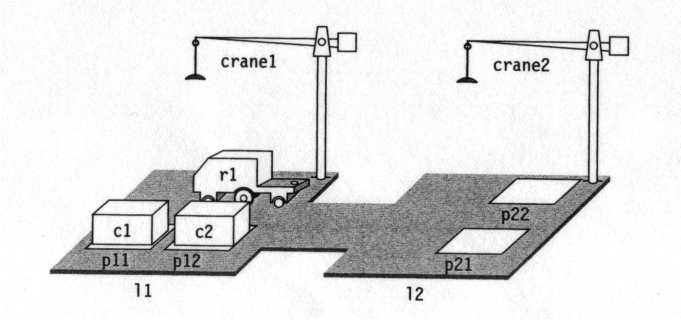


Abbildung 2.2: Startzustand eines DWR Problems bei dem zwei Container von l1 nach l2 bewegt werden sollen

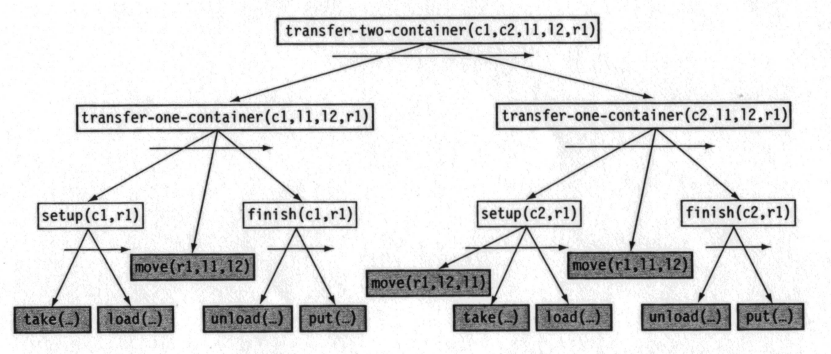


Abbildung 2.3: Zerlegungsbaum zu Abbildung 2.2

problems. Es kann daher gezeigt werden, dass PFD sound und complete ist. PFD kann auch 'geliftet' werden um eine Prozedur mit kleinerem Suchraum zu erzeugen.

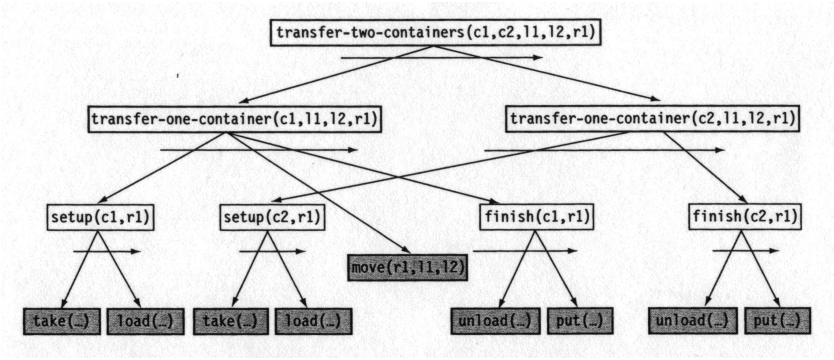


Abbildung 2.4: Dieser geänderte Zerlegungsbaum ist nicht mehr total geordnet

transfer2(c_1, c_2, l_1, l_2, r) *; method to transfer c_1 and c_2*
  task: transfer-two-containers(c_1, c_2, l_1, l_2, r)
  precond: *; none*
  subtasks: ⟨transfer-one-container(c_1, l_1, l_2, r),
  transfer-one-container(c_2, l_1, l_2, r)⟩

transfer1(c, l_1, l_2, r) *; method to transfer c*
  task: transfer-one-container(c, l_1, l_2, r)
  precond: *; none*
  network: u_1 = setup(c, r), u_2 = move-robot(r, l_1, l_2),
  u_3 = finish(c,r), $\{(u_1, u_2), (u_2, u_3)\}$

move1(r, l_1, l_2) *; method to move r if it is not at l_2*
  task: move-robot(l_1, l_2)
  precond: at(r, l_1)
  subtasks: ⟨move(r, l_1, l_2)⟩

move0(r, l_1, l_2) *; method to do nothing if r is already at l_2*
  task: u_0 = move-robot(l_1, l_2)
  precond: at(r, l_2)
  subtasks: ⟨⟩ *; i.e., no subtasks*

do-setup(c, d, k, l, p, r) *; method to prepare for moving a container*
  task: setup(c, r)
  precond: on(c, d), in(c, p), belong(k, l), *;attached(p,l), at (r,l)*
  network: u_1 = take(k, l, c, d, p), u_2 = load(k, l, c, r), $\{(u_1, u_2)\}$

unload-robot(c, d, k, l, p, r) *; method to finish after moving a container*
  task: finish(c, r)
  precond: attached(p, l), loaded(r,c), top(d, p), belong(k, l), at (r,l)
  network: u_1 = unload(k, l, c, r), u_2 = put(k, l, c, d, p), $\{(u_1, u_2)\}$

Abbildung 2.5: So könnten die passenden Methoden aussehen

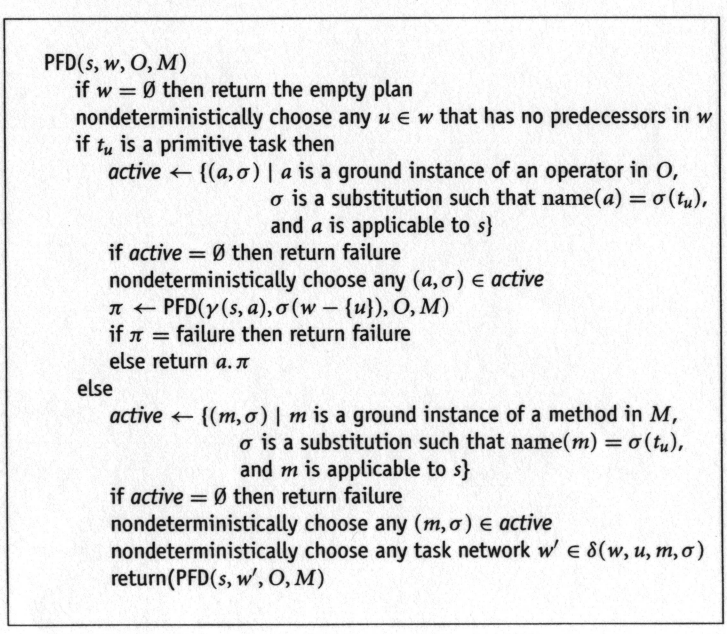


Abbildung 2.6: PFD Prozedur für STN Planning

Kapitel 3

HTN Planning

In STN werden zwei Arten von Bedingungen mit einer Methode verbunden: *ordering constraints* und *preconditions*. Ordering constraints werden explizit als Kanten eines gerichteten Graphen im task network repräsentiert. Im task network werden die preconditions nicht wirklich eingehalten, sie werden erzwungen, indem man einfach nur task networks erstellt, die die erforderlichen Bedingungen erfüllen.
Hierarchical Task Network - HTN ist nun eine Verallgemeinerung von STN. HTN gewährt der Planungsfunktion mehr Freiheiten, wie task networks konstruiert werden. Dazu werden Buchführungsmechanismen benötigt, die die Bedingungen repräsentieren, die der Planungsalgorithmus nicht erzwungen hat.

3.1 Task Networks

Die task network Definition im HTN ist eine Verallgemeinerung der Definition, die STN verwendet. Ein task network ist ein Paar $w = (U, C)$. U sind die task nodes und C ist eine Menge von Constraints. Jede Bedingung in C spezifiziert eine Anforderung, die von jedem Plan erfüllt werden muss, der ein Planungsproblem löst.

3.1.1 Definition Task Networks

Sei Π eine Lösung für ein task network $w = (U, C)$. $U' \subseteq U$ sei eine Menge von task nodes im task network w. A sei die Menge aller Aktionen $a_i \in \Pi$, so dass im Zerlegungsbaum von Π a_i ein Nachfolger eines Knotens aus U' ist.
Dann ist $first(U', \Pi)$ die Aktion $a_i \in A$, die als erstes eintritt, und analog ist $last(U', \Pi)$ die Aktion $a_k \in A$ die als letztes Eintritt.
Hier werden vier Arten von Bedingungen betrachtet:

- Ein *precedence constraint* (Reihenfolge) ist ein Ausdruck der Form $u \prec v$, mit u und v als task nodes, ist identisch zu einer Kante im STN.
- Ein *before-constraint* ist die Verallgemeinerung einer precondition im STN. $before(U', x)$ x muss wahr sein bevor $first(U, \Pi)$ eintritt. Beispiel: $before(\{u\}, at(r2, l2))$ Roboter 2 muss an Platz 2 sein bevor der task von Knoten u ausgeführt wird.
- Ein *after-constraint* verhält sich analog zu einem before, bei einem $after(U, X)$ muss das x wahr sein, bevor $last(U, \Pi)$ eintritt.
- Ein *between-constraint* hat die Form $between(U', U'', x)$ und x muss wahr sein, nachdem $last(U, \Pi)$ eintritt, aber noch bevor $first(U, \Pi)$.

3.1.2 HTN Methoden

Die Definition von Methoden im HTN Planning verallgemeinert die Definition der Methoden aus STN. Eine HTN Methode ist ein 4-Tupel:
$m = (name(m), task(m).subtasks(m), constr(m))$.
Dabei ist *name(m)* ein Ausdruck der Form $n(x_1, \ldots, x_k)$ mit einem eindeutigen n und $x_1 \ldots, x_k$ sind die Symbolvariablen die in m auftreten. *task(m)* ist ein nichtprimitiver task und *(subtasks(m), constr(m))* ist ein task network.
Sei $w = (U, C)$ ein task network, $u \in U$ ist ein task node mit dem task t_u und m ist eine Instanz einer Methode aus M mit $task(m) = t_u$. Dann zerlegt m u in $subtasks(m')$ und erzeugt folgendes task network: $\delta(w, u, m) = ((U - \{u\}) \cup subtasks(m'), C' \cup constr(m'))$ mit C' als modifizierte Version von C für die gilt:

- Für jede Reihenfolgebedingung die u enthält, ersetze es mit dem constraint, das die Knoten des subtasks(m') enthält.
- Für jedes before-, after-, between-constraint ersetze U' mit $U' - \{u\} \cup subtasks(m')$.

3.1.3 HTN Probleme uns Lösungen

HTN Planungsdomänen sind identisch mit STN Planungsdomänen mit der Ausnahme, dass sie HTN Methoden verwenden. Eine HTN Planungsdomäne ist ein Paar $D = (O, M)$ und ein HTN Planungsproblem ist ein 4-Tupel $P = (s_0, w, O, M)$. Dabei ist s_0 der Startzustand, w das initiale task network, O eine Menge von Operatoren und M eine Menge von HTN Methoden.
Ein Plan ist eine Lösung eines Problems. Wenn $w = (U, C)$ primitive ist, dann ist ein Plan eine Lösung für P, wenn es eine ground instance (U', C') von (U, C) sowie eine totale Ordnung der Knoten aus U' gibt, so dass folgende Bedingungen gelten:

transfer2(c_1, c_2, l_1, l_2, r) *;; method to move c_1 and c_2 from pile p1 to pile p2*
task: transfer-two-containers(c_1, c_2, l_1, l_2, r)
subtasks: u_1 = transfer-one-container(c_1, l_1, l_2, r),
u_2 = transfer-one-container(c_2, l_1, l_2, r)
constr: $u_1 \prec u_2$

transfer1(c, l_1, l_2, R) *;; method to transfer c*
task: transfer-one-container(c, l_1, l_2, r)
subtasks: u_1 = setup(c, r), u_2 = move-robot(r, l_1, l_2), u_3 = finish(c, r)
constr: $u_1 \prec u_2$, $u_2 \prec u_3$

move1(r, l_1, l_2) *;; method to move r if it is not at l_2*
task: move-robot(l_1, l_2)
subtasks: u_1 = move(r, l_1, l_2)
constr: before({u_1}, at(r, l_1))

move0(r, l_1, l_2) *;; method to do nothing if r is already at l_2*
task: u_0 = move-robot(l_1, l_2)
subtasks: *;; no subtasks*
constr: before({u_0}, at(r_1, l_2))

- Die Aktionen in Π sind nach den Knoten $u_1 \dots u_k$ benannt.
- Π ist im Startzustand ausführbar.
- Die Reihenfolge hält sich an die constraints in C'.
- before-, after- und between-constraints müssen eingehalten werden.

Wenn mindestens ein task aus w nonprimitive ist, also w nonprimitive ist, existiert eine Lösung, wenn es eine Abfolge von task decompositions gibt, die auf w angewendet das Netz w' bilden, bei dem Π eine Lösung ist.

3.2 HTN im Vergleich und Mächtigkeit

Mit STN kann man nicht entscheidbare Probleme ausdrücken, und damit auch mit HTN, was ja nur eine Verallgemeinerung von STN ist. STN ist auch dann noch in der Lage nicht entscheidbare Probleme auszudrücken, wenn es keine Variablen enthält und total geordnet ist.
Dies bedeutet, dass beide, HTN und STN, ausdrucksfähiger als das Klassische Planen sind, da es im Klassischen Planen nicht möglich ist nicht entscheidbare Probleme auszudrücken.

Dieses Beispiel basiert auf der Fähigkeit der Methoden sich gegenseitig und auch selbst rekursiv aufrufen zu können. Man kann mit Hilfe von antizyklischen Restriktionen die Rekursion auf eine endliche Zahl von Aufrufen beschränken.
Wenn man diesen STN noch weiter einschränken würde, indem man die

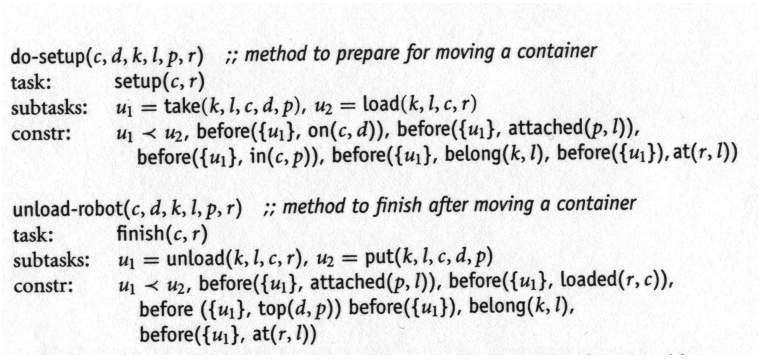


Abbildung 3.1: Beispiel für HTN Methoden

```
method1():
    task:      task1()
    precond: (no preconditions)
    subtasks: op1(), task1(), op2()

method2():
    task:      task1()
    precond: (no preconditions)
    subtasks: (no subtasks)
```

subtask-Liste der Methoden auf höchstens einen nichtprimitiven task beschränkt und dieser am Ende stehen muss, dann würde die Ausdrucksfähigkeit dieses STN's der des Klassischen Planen entsprechen.

3.3 Erweiterungen für HTN

So wie Klassisches Planen erweitert werden kann, ist dies auch für HTN möglich. In TFD oder PFD Erweiterungen aufzunehmen ist einfach, da sie vom Startzustand ausgehend vorwärts planen und daher immer den aktuellen Zustand der Welt kennen. Es ist möglich beliebige Berechnungen auf den aktuellen Zustand anzuwenden, wie zum Beispiel numerische Berechnungen, grundsätzliche Folgerungen oder domänenspezifische Anwendungen wie einen Datenbankzugriff.
Function Symbols: Die Argumente für ein Atom oder eine Aufgabe sind nicht länger auf konstante oder Variable Symbole beschränkt. Es können beliebig komplizierte Terme verwendet werden.
Axioms: Um Axiome einbinden zu können benötigt man einen Theorembeweiser als Unterfunktion der Planungsprozedur. Hier würden sich beispiels-

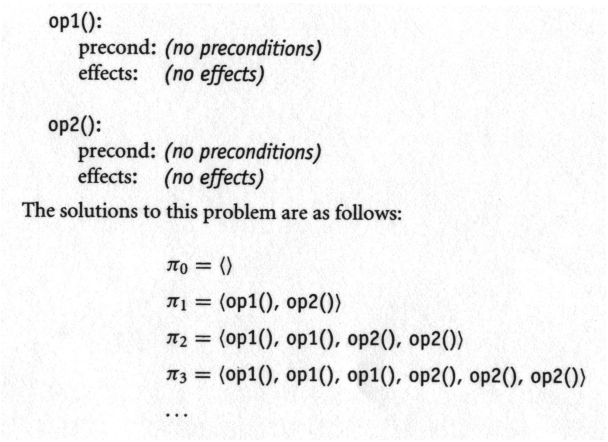


Abbildung 3.2: Beispiel eines total geordneten STN Planungsproblems, dessen Lösungsmenge eine kontextfreie, nicht reguläre Sprache ist

weise Hornklauseln mit einem fertigen Hornklausel-Theorembeweiser einsetzen lassen.
Attached Procedures: Bestimmte Terme oder Prädikatensymbole sollen von bestimmten zusätzlichen Prozeduren ausgewertet werden. Dieses braucht man unter anderem für numerische Berechnungen oder Anfragen an externe Informationsquellen.
High-Level Effects:Erlauben es einem Anwender in der Planungsdomänenbeschreibung verschiedenartige nonprimitive tasks oder deren Methoden zu vereinbaren, was die Effizienz verbessern kann.
External Preconditions: Wenn ein task von einer Methode zerlegt wird und mindestens eine Aktion eine Bedingung benötigt, die von keinem der tasks des Teilplans eingehalten wird, ist diese Bedingung extern und muss an einer anderen Stelle im Plan erreicht werden.
Time: Ermöglicht Zeitpläne, mit denen Aktionen um eine Ausführungszeit erweitert werden können, welche sich unter Umständen nicht überlappen dürfen.

Table 11.1 Complexity of PLAN-EXISTENCE for HTN planning.

Restrictions on nonprimitive tasks	*Must the HTNs be totally ordered?*	*Are variables allowed?*	
		No	*Yes*
None	No	Undecidable[a]	Undecidable[a,b]
	Yes	In EXPTIME; PSPACE-hard	in DEXPTIME;[d] EXPSPACE-hard
"Regularity" (≤ 1 nonprimitive task, which must follow all primitive tasks)	Does not matter	PSPACE-complete	EXPSPACE-complete[c]
No nonprimitive tasks	No	NP-complete	NP-complete
	Yes	Polynomial time	NP-complete

[a] Decidable if we impose acyclicity restrictions.
[b] Undecidable even when the planning domain is fixed in advance.
[c] In PSPACE when the planning domain is fixed in advance, and PSPACE-complete for some fixed planning domains.
[d] DEXPTIME means double-exponential time.

Abbildung 3.3: Mögliche Komplexität von HTN Planern

Kapitel 4

HTN in der Praxis und Zusammenfassung

Die Grundidee von HTN planning wurde vor über 25 Jahren durch die Arbeit von Sacerdoti und im Tate's Nonlin planner entwickelt. HTN planning wird häufiger in Planungsanwendungen benutzt als andere planning applications. Beispiele sind u.a.:

- Produktionsplanung
- Krisenmanagement und Logistik
- Planen/Zeitplanung in der Raumfahrt
- Evakuierungsplanung
- Das Spiel Bridge
- Robotik

In komplexen Anwendungen können HTN Pläne tausende von Knoten besitzen, die dann aber ohne eine passende Darstellung unübersichtlich werden. Verschiedene HTN Planer verfügen über GUIs um das Erstellen, Betrachten, Verfolgen und Überwachen von Planungsprozessen zu unterstützen. Es ist für die Darstellung besonders praktisch, wenn sich der decomposition tree auf unterschiedlichen Abstraktionsebenen betrachten lässt.
Die bekanntesten Domänenunabhängigen HTN Planungssysteme sind:

- Nonline (war das erste)
- SIPE-2
- O-Plan
- UMCP (eine Implementierung des ersten Planungsalgorythmus', der beweisbar sound und complete war)

- SHOP2 (hat 2002 einen Preis gewonnen)

Mehrere dieser Systeme haben die auch hier beschriebenen Erweiterungen implementiert.
Emergency Evacuation Planning ist ein Planungsystem, welches auf HTN basiert, das Experten bei Evakuierungen unterstützen soll. Es muss ohne eine vollständige Beschreibung der Planungsdomäne arbeiten können, da diese in der Realität nicht möglich ist. Die Formulierung eines Evakuierungsplanes kann sehr komplex werden, im Normalfall sind dabei mehrere hundert verschiedene Aufgaben zu erledigen, die dazu auch noch von vielen Faktoren abhängen. Die Art der Gefahr, die zur Verfügung stehenden Ressourcen, die Geographie des zu evakuierenden Gebiets, das Wetter oder mögliche politische Probleme sind nur ein paar Faktoren, die die zu erledigenden Aufgaben beeinflussen.
Aufgrund dieser Probleme muss das Planen von Experten durchgeführt werden. Hierarchical Interactive Case Based Architecture for Planning - kurz HICAP - ist ein Tool, das entwickelt worden ist, um menschliche Experten beim Planen von Evakuierungen zu unterstützen. Da die Pläne sehr hierarchisch sind benutzt HICAP HTNs um Pläne zu repräsentieren. HICAP integriert in sich einen task decomposition editor, einen Hierarchical Task Editor (HTE) mit einem gemischt intuitiven Planungssystem, SHOP mit integriertem NaCoDAE (SiN). Mit HTE werden tasks editiert und mit SiN werden die HTN Pläne verfeinert. HICAP selbst befindet sich noch in der Entwicklungsphase.
Während eines typischen Planungsabschnitts betrachtet der Benutzer zuerst die top-level tasks und verfeinert diese wenn notwendig. Der sucht sich dann selbst die Aufgaben aus die zerlegt werden sollen oder arbeitet mit bereits zerlegten Aufgaben. Arbeiten mit dem Programm kann auch durch vordefinierte Frage-Antwort Fälle erfolgen, die HICAP notwendige Informationen zum Planen geben. Auf die Frage nach dem Wetter könnte man beispielsweise mit 'schlecht' antworten, was die Konsequenz hätte, das der Einsatz von Hubschraubern nicht möglich wäre.
Ein System benötigt vordefiniertes Wissen und zusätzlich das Wissen von Experten, die vor Ort sind. Dies muss dann beides vom System vereint werden. Zusätzlich muss es mit einer unvollständigen Welt arbeiten können.
Der Hauptvorteil von HTN gegenüber Klassischen Planern ist die fortschrittliche Wissensrepresentation und die Fähigkeit zum Schlussfolgern. Der primäre Nachteil ist, dass der Autor neben den Planungsoperatoren auch noch die Methoden schreiben muss. Aber gerade in den Methoden liegt die stärke von HTN. Sie eignen sich hervorragend für die Repräsentation von Expertenwissen.

Anhang A

Literaturverzeichnis

Malik Ghallab, Dana Nau, Paolo Traverso: Automated Planning (theory and practice), Morgan Kaufmann Publishers, Mai 2004