Kapitel 1

SHOP3

1.1 Installation

In diesem Abschnitt wird eine Einführung in das HTN-Planungssystem SHOP3 (Download) gegeben. Für weitere spezifische Programmierungsfragen wird auf das Handbuch (Manual) verwiesen. SHOP3 ist in Common Lisp programmiert. Zum besseren Syntax-Verästindnis sei daher erwähnt, dass alle Elemente (Variablen, Funktionen, Ausdrücke usw.) in Common Lisp in Klammern „(...)“ verschachtelt werden. Die Entwickler von SHOP3 empfehlen den Lisp-Compiler „Steel Bank Common Lisp“ (SBCL) und die Entwicklungsumgebung Emacs (Download) in Kombination mit der Erweiterung „Superior Lisp Interaction Mode for Emacs“ (SLIME) zu installieren. Ein Tutorial zur Installation gibt es hier (Tutorial). Außerdem wird vor dem Start der Programmierung in SHOP3 empfohlen, die Compiler-Einstellungen anzupassen, da SHOP3 bei komplexeren Domänen Speicherplatzprobleme verursacht (siehe Readme). Hierfür muss entweder in der Konfigurationsdatei von SBCL, diese lässt sich in der Benutzeroberfläche von Emacs mit der Tastenkombination *[Steuerung + X + F]* unter *„~/.sbclrc“* finden, oder in der Projektdatei selbst (vor dem eigentlichen Quellcode) folgender Code ergänzt werden:

```
(DECLAIM (OPTIMIZE (speed 3) (space 3)))
```

Algorithm 1: Compiler-Einstellungen für SHOP3

1.2 Domäne und Problembeschreibung

SHOP3 erhält, wie im Allgemeinen alle KI-Planungssysteme, zwei Dinge als Eingabe. Zum einen die Domäne, eine Beschreibung der Planungswelt mit allen Methoden, Operatoren und Axiomen und deren Relationen und zum anderen die Problembeschreibung, die aus zwei Listen besteht. Die erste Liste enthält alle Atome, die das Planungsproblem im Ausgangszustand definieren und die zweite Liste beschreibt den Zielzustand. Anders als bei klassischen Planungsproblemen, die beispielsweise im PDDL-Format beschrieben werden und bei denen der Zielzustand durch eine Liste der zu erreichenden Atome dargestellt wird, wird der Zielzustand bei SHOP3 (und auch anderen HTN-Planern) durch eine Aufgabenliste dargestellt, die aus abzuarbeitenden Methoden und Operationen besteht. Die Domäne wird mit dem Schlüsselwort *defdomain* definiert, gefolgt von dem Domänennamen („beispieldomäne“). Im Rumpf werden alle Methoden, Ope-

ratoren und Axiomen definiert. Externe Funktionen, die von der Domäne ausgehend aufgerufen werden, werden außerhalb implementiert:

(defdomain beispieldomäne
(*Domänenrumpf*)
)

Algorithm 2: Aufbau der Domäne

Eine Problembeschreibung beginnt mit dem Schlüsselwort *defproblem*, gefolgt von dem Problemnamen und dem Domänennamen, auf die sich das Problem bezieht. Im Rumpf stehen die Atomliste und die Aufgabenliste:

(defproblem beispielproblem beispieldomäne
(*Atomliste*)
(*Aufgabenliste*)
)

Algorithm 3: Aufbau der Problembeschreibung

Eine Problembeschreibung kann auch mit dem Schlüsselwort *make-problem*, gefolgt von dem Problemnamen, der Atomliste und der Aufgabenliste, zum Beispiel *(make-problem beispielproblem atomliste aufgabenliste)*, definiert werden.

1.3 Plansuche

In SHOP3 ist bereits standardmäßig ein Solver implementiert, der automatisiert die Aktionspläne zu den modellierten Domänen und Problembeschreibungen erstellt. Der Solver wird mit der Funktion *find-plans*, gefolgt von einem Apostroph und dem Namen der Problemschreibung aufgerufen (siehe Algorithmus 4). Zusätzlich können der Funktion verschiedene Argumente mitgegeben werden. Mit *:which* und einem Wert *:wert* kann die Art der Suche festgelegt werden. Folgende Werte sind möglich:

- :first - Tiefensuche. Der erste gefundene Plan wird ausgegeben. (Default-Wert)
- :all - Tiefensuche. Alle Pläne werden ausgegeben.
- :shallowest - Tiefensuche. Der Erste und kürzeste Plan wird ausgegeben.
- :all-shallowest - Tiefensuche. Alle und der kürzeste Plan wird ausgegeben.
- :id-first - Iterative Tiefensuche. Der erste gefundene Plan wird ausgegeben.
- :id-all - Iterative Tiefensuche. Alle und der kürzeste Plan wird ausgegeben.
- :random - Zufällige Suche. Nicht für den normalen Gebrauch gedacht.

Mit *:verbose* gefolgt von *:stats*, *:plans* oder *:long-plans* können erweiterte Informationen zu den Plänen ausgegeben werden. Der Parameter *:stats* (Default-Wert) gibt nur zusammenfassende Informationen in einer Tabelle zu allen Plänen aus, während *:plans* die zugehörige Aktionsliste und *:long-plans* die zugehörige Aktionsliste mit den jeweiligen Kosten pro Aktion ausgibt. Außerdem sucht die *find-plans*-Funktion solange nach einem Plan, bis alle Zerlegungsmöglichkeiten bei der Plansuche durchgeführt worden

sind oder alternativ die Suche mit dem Parameter *:time-limit n*, einer Zeitbegrenzung für die Suche, frühzeitig gestoppt wird (wobei n das Zeitlimit in Sekunden angibt). Folgender Beispiel-Funktionsaufruf gibt alle kürzesten Pläne, die innerhalb von 10 Sekunden gefunden werden mit der zugehörigen Aktionsliste aus:

(find-plans 'beispielproblem :which :all-shallowest :verbose :plans :time-limit 10)

Algorithm 4: Aufruf der find-plans-Funktion

```
Defining problem BEISPIELPROBLEM ...
------------------------------------------------------------------------
Problem #<SHOP3::PROBLEM BEISPIELPROBLEM> with :WHICH = :ALL-SHALLOWEST, :VERBOSE = :PLANS

Totals: Plans Mincost Maxcost Expansions Inferences  CPU time  Real time
          2       0       0          5          1     0.001      0.001

Plans:
(((!DO OPERATOR-1)) ((!DO OPERATOR-2)))
```

Abbildung 1.1: Beispiel-Planausgabe

Ohne die Domäne und eine Problembeschreibung würde dieser Funktionsaufruf jedoch einen Fehler zurückgeben. Neben *:time-limit n* gibt es noch weitere Konfigierungsmöglichkeiten für die Plansuche, wie etwa maximale Kosten *:optimize-cost c* für die Plansuche festzulegen (vgl. Kapitel 3.2.4 im Manual).

1.3.1 Atome

Atome werden in SHOP3 wie folgt dargestellt:

(predicate t_1 t_2 ... t_n)

Algorithm 5: Atome in Shop

Das *predicate* ist die Bezeichnung des Atoms. Jedes t_n repräsentiert einen Term des Atoms und kann entweder eine Zahl, eine Konstante, eine Variable oder eine Liste sein. Folgende Atome definieren zum Beispiel ein Fahrzeug oder die Entfernung zwischen zwei Orten: *(truck truck1)* und *(distance shovel1 crusher1 800)*.

1.4 Axiome

In Algorithmus 6 auf der nächsten Seite ist der Aufbau eines Axioms dargestellt. In SHOP3 schlussfolgern Axiome Tatsachen aus den Informationen des aktuellen Weltmodells. a und E sind logische Ausdrücke. a wird abgeleitet, wenn eines der E wahr ist. Die Namen der E_n sind optional.

```
(:- a
    name1 E1
    name2 E2
    ...
)
```

Algorithm 6: Axiome in Shop

Beispielsweise würde das folgende Axiom ableiten, dass ein Fahrzeug den Status vollgeladen hat, wenn das Fahrzeug nicht leer ist:

```
(:- (status ?truck loaded) ((payload ?truck ?val) (not (= ?val 0))))
```

1.5 Operatoren

In SHOP3 repräsentieren die Atome den aktuellen Weltzustand und definieren in der Atomliste der Problembeschreibung den Startzustand. Auf die Atome in der Problembeschreibung wendet SHOP3 bei der Planzerlegung die Operatoren und Methoden an. Die Operatoren beschreiben in SHOP3 die primitiven Aktionen. Diese können nicht weiter zerlegt werden und bilden die Aktionen in der Aktionsliste des finalen Plans ab. Operatoren werden wie folgt implementiert:

```
(:op (!beispieloperator ?argument1 ?argument2 ... )
    :add ( Add-Liste )
    :delete ( Delete-Liste )
    :precond ( Vorbedingungen )
    :cost Kostenfunktion
)
```

Algorithm 7: Operatoren in Shop

Vor dem Namen des Operators steht ein Ausrufezeichen. In SHOP3 gibt es „Interne Operatoren“, die mit zwei Ausrufezeichen vor dem Namen beginnen, diese werden wie unsichtbare Aktionen im Plan ausgeführt. Das heißt, dass ein Interner Operator zwar angewendet und den Zustand der Modellwelt verändert, jedoch nicht im Plan ausgegeben wird und damit für interne Berechnungen nützlich ist. Optional können dem Operator Argumente übergeben werden. Argumente beginnen mit einem Fragezeichen. Die Vorbedingungen des Operators bestehen aus logischen Ausdrücken, wenn die Vorbedingungen leer sind, wird standardmäßig *true* zurück gegeben. SHOP3 bietet einige Möglichkeiten in den Vorbedingungen, wie zum Beispiel numerische Berechnungen oder externe Funktionsaufrufe. Ein logischer Ausdruck kann zum Beispiel ein Atom, eine Negation *(not (beispielatom wert))*, eine Konjunktion *(and a_1 a_2 ... a_n)*, eine Disjunktion *(or a_1 a_2 ...a_n)*, eine Implikation *(a->b)* - wenn *a* gilt, dann muss *b* erfüllt sein, ein Universeller Quantifizierer *(forall ...)* - Bedingungen auf eine Menge von Atomen, eine Sammlung von Atomwerten in einer Liste *(setof ...)*, oder eine Variablenzuordnung *(assign ?beispielvariable (+ 2 3))* sein. Anwendungsbeispiele für einen universellen Quantifizierer und die *setof*-Funktion sind auf der nächsten Seite dargestellt.

```
(forall ?wert
        (atom ?wert)
        ((<= 0 ?wert) (<= 3 ?wert)))
```

Algorithm 8: Universelle Quantifizierer in Shop

In diesem Beispiel müssen alle Werte der Atome mit der Form *(atom ?wert)* größer gleich 0 und kleiner gleich 3 sein, damit der Universelle Quanitifizierer erfüllt wird.

```
(setof ?neuer-wert
        ((atom ?wert)
         (<= 0 ?wert) (assign ?neuer-wert (+ ?wert 1)))
        ?liste)
```

Algorithm 9: Setof-Funktion in Shop

Die *setof*-Funktion in SHOP3 erstellt im Allgemeinen einen Satz bzw. eine Liste von Variablenwerten. In diesem Beispiel werden alle Werte der Atome mit der Form *(atom ?wert)*, dessen Wert größer gleich 0 ist, mit 1 addiert und der Liste *?liste* hinzugefügt. Alle diese beschriebenen Funktionen für die Vorbedingungen werden im Handbuch (Manual) im Abschnitt 4.5 definiert. Außerdem kann mit einer *:sort-by*-Funktion, die am Anfang der Methode deklariert wird, bestimmte Atome vor den Variablenbindungen in den Vorbedingungen sortiert werden. Für die Anwendung der *:sort-by*-Funktion wird im Handbuch (Manual) auf Abschnitt 4.6.2 verwiesen. Aus der Erfahrung des Autors wird allerdings abgeraten die *:sort-by*-Funktion zu verwenden, da bei komplexeren Planungsdomänen dadurch viele Auswahlpunkte und Stack-Rahmen für mögliche Planzerlegungen entstehen und es zu Speicherplatzproblemen während der Plansuche kommt. Mit *:first* kann festgelegt werden, dass nur die erste gefundene Variablenbindung berücksichtigt wird. Befinden sich auf dem Heap beispielsweise die Atome *(atom 1)* und *(atom 2)* und in den Vorbedingungen steht als Ausdruck *(atom ?irgendeinwert)*, so kann der Variable *?irgendeinwert* 1 oder 2 zugeordnet werden, und wenn *:first* deklariert wird nur 1. Alle Variablen, die in der Add-Liste und der Delete-Liste verwendet werden, müssen entweder als Argument dem Operator übergeben oder hier definiert werden. Alle Elemente *:add*, *:delete*, *:precond* und *:cost* im Operator sind optional. Standardmäßig verursacht jeder Operator (auch die Internen Operatoren) die Kosten 1. Mit *:cost* können definierte Kosten für den Operator festgelegt werden.

Abschließend wird noch ein Tipp für die Parameter der Operatoren gegeben. Da die Operaten im Plan ausgegeben werden, können die Argumente zusätzliche Informationen über die aufgerufenen Aktionen und den Planverlauf geben, wie im folgenden Beispiel zu sehen ist:

Option 1: **(:op (!drtive-to ?truck ?destination))**
Option 2: **(:op (!drtive-to ?truck ?start ?destination ?starttime))**
...
Plan 1: ... **(!drtive-to truck1 b)** ...
Plan 2: ... **(!drtive-to truck1 a b 600)** ...

1.6 Methoden

In SHOP3 repräsentieren die Methoden die höheren Aktionen, die zerlegt werden. Eine Methode wird in der Aufgabenliste in weitere Methoden und Operatoren zerlegt. Ein finaler Plan besteht jedoch immer nur aus den primitiven Aktionen, den Operatoren.

Eine Methode wird wie folgt definiert:

(:op (beispielmethode ?argument1 ?argument2 ...**)**
zerlegung1
(*Vorbedingungen*)
(*Aufgabenliste*)
zerlegung2
(*Vorbedingungen*)
(*Aufgabenliste*)
...
)

Algorithm 10: Methoden in Shop

Im Rumpf der Methode ist „zerlegung1“ der Namen einer möglichen Zerlegung. Zu jeder Zerlegung muss eine Vorbedingungs- und eine Aufgabenliste definiert werden, diese können auch leer sein. Die *Vorbedingungen* haben dieselben Funktionen wie bei den Operatoren. Methoden dürfen beliebig viele Zerlegungen haben. Bei der Dekomposition behandelt SHOP3 die Zerlegung wie ein *else*. Eine weitere zentrale Eigenschaft von SHOP3 ist die Ordnung der Aufgaben bei der Zerlegung. Die abzuarbeitende Aufgabenliste einer Methoden kann entweder vollständig oder partiell geordnet sein. Partiell geordnete Aufgabenlisten erlauben es, die Methoden simultan zu zerlegen. Mit *:ordered* und *:unordered* wird die Ordnung definiert. Standardmäßig ist jede Aufgabenliste geordnet. Folgendes Beispiel wird dafür betrachtet:

(:method (beispielmethode1)
zerlegung-m1 () (:unordered (!operator1) (!operator2)))
(:method (beispielmethode2)
zerlegung-m2 () (:unordered (!operator3) (!operator4)))
(:method (high-level-aktion)
zerlegung-ha () (:unordered (beispielmethode1) (beispielmethode2)))

Angenommen es soll *(beispielmethode1)* zerlegt werden, so gibt es zwei Lösungen und die Pläne lauten: *((!operator1) (!operator2))* und *((!operator2) (!operator1))*. Wäre hingegen *(beispielmethode1)* als *:ordered* definiert, gäbe es nur eine mögliche Zerlegung: *((!operator1) (!operator2))*. Wird die Methode *(high-level-aktion)* zerlegt, so gibt es sogar 24 verschiedene Möglichkeiten diese Methode zu zerlegen: *((!operator1) (!operator3) (!operator2) (!operator4))* oder *((!operator1) (!operator3) (!operator4) (!operator2))* usw..

Abschließend sei noch ergänzend gesagt, dass in SHOP3 Methoden zum Debuggen in der Domäne definiert werden können, mit denen beispielsweise alle Atome des aktuelle Weltmodells ausgegeben werden *(print-current-state)* und das Debuggen erleichtert werden soll. Dies wird im Handbuch (Manual) im Abschnitt 3.4 beschrieben.