Universität Dortmund
Lehrstuhl Verteilte Systeme

Dr. Michael Zapf

Michael.Zapf@cs.uni-dortmund.de
Michael.Zapf@uni-kassel.de

vkiv@ls4.cs.uni-dortmund.de

Übung zur Vorlesung „Verteilt-kooperative Informationsverarbeitung“ - SS 2007

Lösungen zu Blatt 6

Aufgabe 6.1 – 10 Punkte

a) Modellieren Sie eine englische Auktion in KQML. [...]

„Modellieren“ ist hier so zu verstehen, dass demonstriert wird, wie die Auktion unter KQML ablaufen würde. Wir bedienen uns hier keiner speziellen Notation wie UML.

```
(subscribe :sender Bieter1 :receiver Auktionator :ontology kqml-ontology :language KQML
    :content (ask-one :sender Bieter1 :receiver Auktionator :ontology Auktion :language MeineSprache
            :content "Ware=XXX, aktuellesGebot?") )

(tell :sender Auktionator :receiver Bieter1 :ontology Auktion :language MeineSprache
    :content "Ware=XXX, aktuellesGebot=100 Euro")

(subscribe :sender Bieter2 :receiver Auktionator :ontology kqml-ontology :language KQML
    :content (ask-one :sender Bieter2 :receiver Auktionator :ontology Auktion :language MeineSprache
            :content "Ware=XXX, aktuellesGebot?“) )

(tell :sender Auktionator :receiver Bieter2 :ontology Auktion :language MeineSprache
    :content "Ware=XXX, aktuelles Gebot=100 Euro")

(subscribe :sender Bieter3 :receiver Auktionator :ontology kqml-ontology :language KQML
    :content (ask-one :sender Bieter3 :receiver Auktionator :ontology Auktion :language MeineSprache
            :content "Ware=XXX, aktuellesGebot?") )

(tell :sender Auktionator :receiver Bieter3 :ontology Auktion :language MeineSprache
    :content "Ware=XXX, aktuelles Gebot=100 Euro")

...

(ask-if :sender Bieter1 :receiver Auktionator :ontology Auktion :language MeineSprache
    :content "Ware=XXX, mein Gebot=110 Euro, akzeptiert?")

(tell :sender Auktionator :receiver Bieter1 :ontology Auktion :language MeineSprache
    :content "Ware=XXX, akzeptiere Gebot")

(tell :sender Auktionator :receiver Bieter1 :ontology Auktion :language MeineSprache
    :content "Ware=XXX, aktuelles Gebot=110 Euro")

(tell :sender Auktionator :receiver Bieter2 :ontology Auktion :language MeineSprache
    :content "Ware=XXX, aktuelles Gebot=110 Euro")

(tell :sender Auktionator :receiver Bieter3 :ontology Auktion :language MeineSprache
    :content "Ware=XXX, aktuelles Gebot=110 Euro")


(ask-if :sender Bieter2 :receiver Auktionator :ontology Auktion :language MeineSprache
    :content "Ware=XXX, mein Gebot=115 Euro, akzeptiert?")

(tell :sender Auktionator :receiver Bieter2 :ontology Auktion :language MeineSprache
    :content "Ware=XXX, akzeptiere Gebot")

(tell :sender Auktionator :receiver Bieter1 :ontology Auktion :language MeineSprache
    :content "Ware=XXX, aktuelles Gebot=115 Euro")
```

(tell :sender Auktionator :receiver Bieter2 :ontology Auktion :language MeineSprache
:content "Ware=XXX, aktuelles Gebot=115 Euro")

(tell :sender Auktionator :receiver Bieter3 :ontology Auktion :language MeineSprache
:content "Ware=XXX, aktuelles Gebot=115 Euro")

(ask-if :sender Bieter3 :receiver Auktionator :ontology Auktion :language MeineSprache
:content "Ware=XXX, mein Gebot=112 Euro")

(tell :sender Auktionator :receiver Bieter3 :ontology Auktion :language MeineSprache
:content "Ware=XXX, Gebot zu niedrig")

...

keine weiteren Gebote

...

(insert :sender Auktionator :receiver Bieter1 :ontology Auktion :language MeineSprache
:content "Ware=XXX, Gebot=115 Euro, kein Zuschlag, Ende")

(insert :sender Auktionator :receiver Bieter2 :ontology Auktion :language MeineSprache
:content "Ware=XXX, Gebot=115 Euro, Zuschlag, Ende")

(insert :sender Auktionator :receiver Bieter3 :ontology Auktion :language MeineSprache
:content "Ware=XXX, Gebot=115 Euro, kein Zuschlag, Ende")

Hinweise (siehe auch Spezifikation):

ask-if erfordert eine Antwort, **tell** ist eine Antwort; **insert** erfordert keine Antwort. subscribe ist so vorgesehen, dass der Anfrager jene Performative im Inhalt nennt, die er an den Empfänger im Einzelfall richten würde. Entsprechend muss die Anfrage ein „ask-if/one/all“ sein, und der Empfänger antwortet mit einem tell.

Die hier verwendeten ask-ifs bewirken zugleich eine Änderung der VKB. Will man dies voneinander trennen, so muss man ein insert mit dem Gebot und dann ein ask-if schicken, ob das Gebot akzeptabel ist.

subscribe führt dazu, dass bei einer Änderung des entsprechenden Faktums in der VKB eine Nachricht versendet wird. Es wird in Implementierungen empfohlen, dass der Empfänger des subscribe eine erste Antwort als Vergleichsbasis schickt.

Achtung bei der Verwendung von **broadcast** (siehe KQML-Spezifikation): „[Broadcast] is a request to forward the <performative> [in the content] to all the agents that the :receiver knows of, i.e. to all agents that have registered (using register with the :receiver if the :receiver is a facilitator), or that the :receiver might know of. A broadcast is equivalent [...] to a series of forward messages to all such agents.“

Wenn man zur Modellierung der Kommunikation also broadcast wählt, sollte sich der Auktionator an einen Facilitator wenden. Deshalb wurde in dieser Lösung auf broadcast verzichtet.

b) Entwerfen Sie zu Ihrer Modellierung in a) eine entsprechende Realisierung. [...]

```
public class Auktionator extends Agent {

  Ontology ontAuc;
  Language lang;
  KQMLInterpreter kqmlInt;

 public Auktionator() {
   ontAuc = new AuctionOntology();
   kqmlInt = new KQMLInterpreter(ontAuc);
   lang = new Language(my_very_own_content_language);
  }

  public void invoke() {
    // warten auf Teilnehmer
    while (!timePrepareOver) {
       KQMLMessage km = getKQMLMessage(kqmlInt, messageSystem);
       if (km.getMessageType().equals(KQML.subscribe)) {
            if (lang.messageMatching(ontAuc, km.getContent(), thisAuction)) {
```

```
                bidderList.add(km.getSender());
            }
        }
    }
    currentBid = minimumBid;
    for (bidder in bidderList) submitMessage(new KQMLMessage(
                KQML.tell, bidder, currentBid, beginAuction));
    auctionRunning = true;
    while (auctionRunning) {
       KQMLMessage km = getKQMLMessage(kqmlInt, messageSystem);
       if (km.getMessageType().equals(KQML.askIf)
                && lang.messageMatching(ontAuc, km.getContent(), thisAuction)) {
            Sender bidder = km.getSender();
            Value val = lang.getValue(ontAuc, "Gebot");
            if (val <= lastVal) {
                submitMessage(new KQMLMessage(KQML.tell, bidder, "Gebot zu niedrig"));
            }
           else {
                submitMessage(new KQMLMessage(KQML.tell, bidder, "Gebot akzeptiert"));
            }
            for (bidder in bidderList) submitMessage(new KQMLMessage(
                KQML.tell, bidder, currentBid));
        } // if (km...)
       if (timeout) auctionRunning=false;
    }
     for (bidder in bidderList) {
         if (isWinner(bidder)) submitMessage(new KQMLMessage(KQML.insert, bidder, ontAuc.gotIt));
         else submitMessage(new KQMLMessage(KQML.insert, bidder, ontAuc.lostIt));
     }
  }
}
```

In dieser Implementierung des Auktionators wird nach jedem Gebot der aktuelle Stand an alle Agenten geschickt, während in der Modellierung der Auktion in KQML unter a) auch parallele Gebote möglich waren. Dies kann beim Auktionator mittels Multithreading gelöst werden.

```
public class Bieter extends Agent {

   Ontology ontAuc;
   Language lang;
   KQMLInterpreter kqmlInt;

  public Bieter() {
     ontAuc = new AuctionOntology();
     kqmlInt = new KQMLInterpreter(ontAuc);
     lang = new Language(my_very_own_content_language);
   }

   public void invoke() {
      // Adresse des Auktionators schon bekannt
      submitMessage(new KQMLMessage(
                KQML.subscribe, auctioneer, lang.create(ontAuc.auction, "Ware=XXX"));
      // warte auf Antwort
      KQMLMessage km = getKQMLMessage(kqmlInt, messageSystem);
      if (km.getSender().equals(auctioneer) && km.getMessageType().equals(KQML.tell) &&
                lang.messageMatching(ontAuc, km, thisAuction)) {
          auctionRunning = true;
          while (auctionRunning) {
              waitSomeTime();
              latestBid = minimumBid;
              if (decideToIncrease(latestBid)) {
                 myBid = increase(latestBid);
```

```
                submitMessage(new KQMLMessage(
                        KQML.askIf, auctioneer, lang.create(ontAuc.content, "Gebot", myBid)));
            }
            km = getKQMLMessage(kqmlInt, messageSystem);
            if (km.getSender().equals(auctioneer) && km.getMessageType().equals(KQML.tell)) {
                if (lang.messageMatching(ontAuc, km, thisAuction)
                    && lang.messageMatching(ontAuc, km, "Gebot akzeptiert")) {
                   // Prima, Gebot ist akzeptiert
                }
                if (lang.messageMatching(ontAuc, km, thisAuction)
                    && lang.messageMatching(ontAuc, km, "Gebot zu niedrig")) {
                   // Gebot ist nicht akzeptiert
                }
                if (lang.messageMatching(ontAuc, km, currentBid)) {
                    latestBid = lang.getValue(ontAuc, "aktuelles Gebot");
                }
                if (lang.messageMatching(ontAuc, km, "Ende") {
                    auctionRunning=false;
                }
          } // if
        } // while
        if (lang.interpret(ontAuc, km).equals(ontAuc.gotIt)) {
             // Freude
        }
        else {
           // schade
        }
      } // if
    } // invoke
}
```

Zur Realisierung: Der darstellte Code ist natürlich noch lange nicht lauffähig, und meist lauert der Teufel im Detail.

KQML-Nachrichten werden über einen fiktiven KQML-Interpreter aus der textuellen Darstellung in eine leichter zu verarbeitende Form gebracht. Dieser könnte die Daten der Nachricht etwa in einem Baum oder einer Hashtabelle speichern.

Die Interpretation des Inhalts könnte in einem "Language"-Objekt mit einem zugehörigen "Ontology"-Objekt geschehen. Soll die Sprache auch für andere Zwecke nutzbar sein, so muss das Ontology-Objekt die für die Auktion wesentliche Verarbeitung leisten, etwa auch erwartete Nachrichten definieren.

Aufgabe 6.2 – 10 Punkte

a) Stellen Sie einen Vergleich zwischen den KQML-Befehlen und den FIPA-Befehlen auf. Geben Sie an, welche Befehle sich entsprechen oder im anderen Modell nicht vorkommen.

KQML	FIPA-ACL
achieve	request, request-when(ever)
advertise	propose
-	agree
ask-about/all/one	query-ref
ask-if	query-if
break	cancel (bedingt)
broadcast	proxy (bedingt)
broker-all/one	proxy (bedingt)
-	call-for-proposal
-	cancel

deny	disconfirm (bedingt)
delete-all/one	-
discard	cancel (bedingt)
eos	-
error	not-understood
evaluate	request, request-when(ever)
forward	proxy
generator	-
insert	inform (bedingt)
monitor	-
next	-
pipe	-
ready	-
recommend-all/one	-
recruit-all/one	-
-	refuse
register	-
reply	-
rest	-
sorry	failure (bedingt)
standby	-
stream-about/all	-
subscribe	subscribe
tell	confirm (bedingt), failure (bedingt), inform(-if,-ref) (bedingt), accept-/reject-proposal (bedingt)
transport-address	-
unregister	-
untell	-

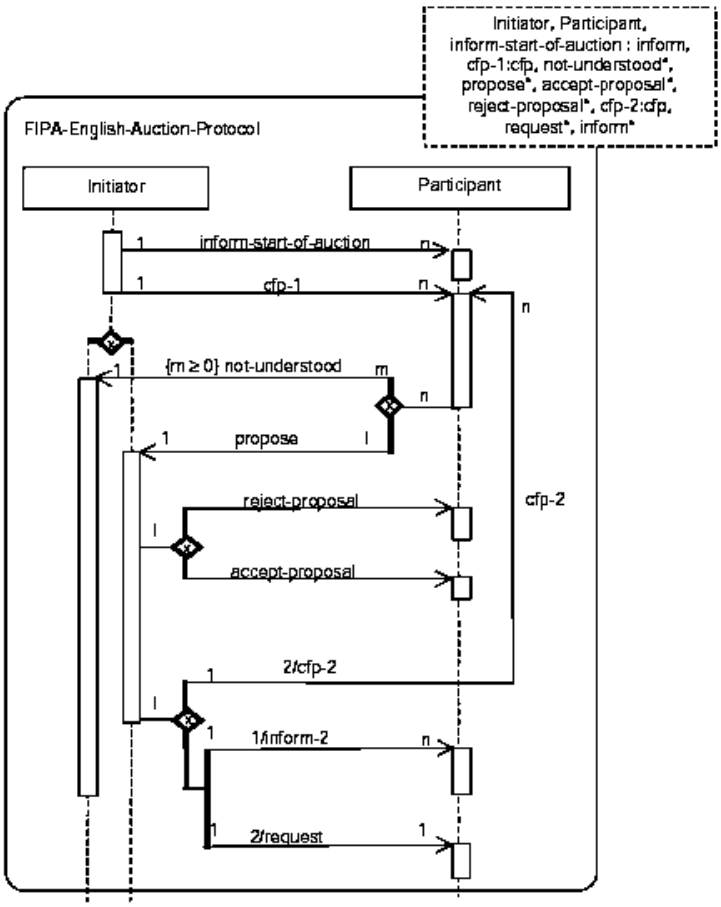
Viele der Punkte lassen sich deswegen nicht so ohne weiteres vergleichen, weil FIPA-ACL ziemlich strikte semantische Bedingungen aufstellt. So gibt es eigentlich keine Entsprechungen zu „insert/delete“, denn FIPA-ACL schreibt vor, dass der Absender tatsächlich an die Wahrheit der übermittelten Daten glaubt. Dies ist dann aber schon „tell“ in KQML. Eine Entsprechung zu „untell“ oder „delete“ gibt es daher auch nicht, denn „disconfirm“ kann nur angewendet werden, wenn der Absender an die Falschheit glaubt; bei KQML drückt der Absender mit „untell“ aus, dass die Information in dieser Form in seiner Wissensbasis nicht vorhanden ist.

b) Kann man den KQML-Ansatz aus Aufgabe 6.1 direkt übernehmen und auf FIPA-ACL umschreiben?

Das ist nicht so einfach. Im Allgemeinen findet man für bestimmte Befehle keine Entsprechung; in anderen Fällen sind die semantischen Bedingungen zu beachten. Die im obigen Beispiel verwendteen Performative lassen sich jedoch einfach umsetzen.

Sinnvoller ist jedoch die Nutzung der spezifischen Proposal-Befehle. Diese dienen nicht nur dazu, sich als Dienstleister vermitteln zu lassen, sondern ganz allgemein auszudrücken, dass der Absender willens ist, eine bestimmte Aktion durchzuführen – etwa einen bestimmten Betrag zu zahlen.

Es gibt eine bereits fertig definierte Englische Auktion in FIPA. Das Diagramm sieht in AgentUML folgendermaßen aus (aus http://www.fipa.org/specs/fipa00031/XC00031F.pdf):



c) Welche Bedeutung haben die semantischen Angaben (Vor- und Nachbedingung) bei FIPA-ACL für die Implementierung? Wie würden Sie diese Bedingungen in Ihre Implementierung einfließen lassen?

Diese semantische Spezifikation gab Anlass zur Kritik, wie von Finin und Labrou geäußert. Denn als Agentenautor hat man nun beispielsweise sicherzustellen, dass der Agent wirklich nur dann ein Confirm verwendet, wenn er weiß, dass der Adressat sich seiner Sache nicht sicher ist und er selbst an die Richtigkeit glaubt. Das bedeutet eigentlich schon, dass man den Agenten mit entsprechendem Verhalten versehen muss, was über die eigentliche Absicht, eine gemeinsame Sprache zu definieren, hinausgeht.

Im Programm müssten – wollte man es richtig machen – tatsächlich Informationen über den „Geisteszustand“ der anderen Agenten vorliegen. Man dürfte dann nicht einfach ein „inform“ losschicken, sondern müsste das Wissen prüfen. Entspechend müsste die Datenbasis nach Durchführung aktualisiert werden. Denn erst dies würde in einer Schleife verhindern, dass beispielsweise zweimal ein „inform“ mit denselben Daten an denselben Agenten geschickt wird.

Aufgabe 6.3 – 10 Punkte

Informieren Sie sich bei www.fipa.org über die Inhaltssprache FIPA-RDF. Nennen Sie

- die wesentlichen Eigenschaften von FIPA-RDF (was kann man ausdrücken, welchen Sprachumfang hat es usw.)
- ein einfaches Beispiel, wie Sie Wissen mittels FIPA-RDF einem anderen Agenten mitteilen können.

(aus http://www.w3.org/TR/rdf-primer/)

> *RDF (Resource Description Framework) ist eine Sprache zur Repräsentation von Information über Webressourcen im WWW. Sie zielt besonders auf die Repräsentation von Daten wie Titel, Autor, Änderungsdatum, Copyright, Lizenzbedingungen ab, oder auch die Verfügbarkeitszeiträume von gewissen gemeinsamen Ressourcen. Man kann die Idee der Ressource auch dergestalt verallgemeinern, dass auch Informationen über Dinge auf dem Web gefunden werden können, die ihrerseits nicht im Web vorhanden sind. Beispiele sind Preisangaben in Online-Shops, Benutzerpräferenzen für die Lieferung von Waren und anderes.*

FIPA-RDF ist eine Spezifikation, wie RDF als Inhaltssprache für FIPA-ACL-Nachrichten, die zwischen Agenten ausgetauscht werden, genutzt werden kann.

FIPA-RDF definiert drei allgemeine Konzepte:

– Objekte (Objects)

– Aussagen (Propositions)

– Aktionen (Actions)

FIPA-RDF sieht Erweiterbarkeit vor. Die grundlegende Version ist fipa-rdf0; weitere Varianten sind dann fipa-rdf1, fipa-rdf2 und so weiter. FIPA-RDF kann in Form eines XML-Dokuments kodiert werden. Andere Kodierschemata sind verfügbar.

Ausdrucksmöglichkeiten

FIPA-RDF verfolgt ein Entity-Relationshop-Modell, das also Entitäten (Objekte) durch Beziehungen miteinander verknüpft. Dabei zeichnet RDF eines der Objekte als Subjekt aus (welches die Ressource bezeichnet), das andere Objekt ist der Wert. Die Beziehung wird durch ein zweistelliges Prädikat dargestellt.

Die Verbindung von einem Subjekt mit einem Wert über die Beziehung wird Aussage genannt, und eine Menge von RDF-formulierten Aussagen über eine Ressource ist die RDF-Beschreibung (description).

Ressourcen und Beziehungen werden häufig mit URIs gekennzeichnet (um Verwechslungen zu vermeiden). Die zugeordneten Werte werden als Zeichenketten gelesen.

Die XML-Repräsentation von FIPA-RDF sieht dabei so aus:

```
<fipa:Proposition>
   <rdf:subject>Verteilt-kooperative Informationsverarbeitung</rdf:subject>
   <rdf:predicate rdf:resource="http://meineBezugsadresse/schema#Dozent"/>
   <rdf:object>Michael Zapf</rdf:object>
   <fipa:belief>true</fipa:belief>
</fipa:Proposition>
```

und ist so zu lesen, dass der Dozent von „Verteilt-kooperative Informationsverarbeitung“ Michael Zapf ist. Was ein „Dozent“ ist, habe ich durch den URI festgelegt. Diese Aussage wird als „wahr“ bewertet.

Aktionen

Aktionen werden aus der Beschreibung der Aktion, der Nennung des Akteurs sowie ggf. des bearbeiteten Objekts gebildet. Es ist möglich, dass sich Aktionen auf keine anderen Objekte oder auf mehrere andere Objekte beziehen.

Beispiel:

```
<fipa:Action rdf:ID="Aktion1">
  <fipa:actor>Michael</fipa:actor>
  <fipa:action>schreiben</fipa:action>
  <fipa:argument>Musterlösung</fipa:argument>
</fipa:Action>
```

Die Implementierung der Aktion kann referenziell (Verweis auf den URI der Implementierung) oder unmittelbar (durch ein Skript) präsentiert werden.

fipa-rdf1 erweitert fipa-rdf0 durch die Einführung von *Regeln*. Eine Regel besteht aus einem Teil, der eine Menge von Daten auswählt, sowie einer Beschreibung, welche Operationen auf dem ausgewählten Teil ausgeübt werden.

Sprachelemente

(subject/predicate/object entstammen dem Standard-RDF)

fipa-rdf0:Proposition

fipa-rdf0:belief

fipa-rdf0:Action

fipa-rdf0:act

fipa-rdf0:actor

fipa-rdf0:argument

fipa-rdf0:done

fipa-rdf0:result

fipa-rdf0:implementedBy

fipa-rdf0:Code

fipa-rdf0:language

fipa-rdf0:binding

fipa-rdf0:code-uri

fipa-rdf0:script

fipa-rdf1:Rule

fipa-rdf1:selection

fipa-rdf1:manipulation

fipa-rdf1:selection-result

fipa-rdf1:implementedAs

Beispiel

Um einem anderen Agenten Wissen mitzuteilen, benötigt man eine Kommunikationssprache wie FIPA-ACL. So könnte folgender Sprechakt geeignet sein, einem Agenten mitzuteilen, dass ich mich soeben eingeloggt habe.

```
(inform :sender MeinAgent
        :receiver DeinAgent
        :language FIPA-RDF
        :content (
           <?xml version="1.0"?>
           <rdf:RDF    xmlns:rdf="http://www.w3.org/TR/REC-rdf-syntax"
                       xmlns="http://www.fipa.org/schemas/fipa-rdf0">
              <Proposition>
                 <rdf:subject>Michael Zapf</rdf:subject>
                 <rdf:predicate>einloggen</rdf:predicate>
                 <rdf:object>System</rdf:object>
                 <belief>true</belief>
              </Proposition>
        )
)
```

Aufgabe 6.4 – 10 Punkte

Wir wollen einen Agenten, genauer einen Saugroboter, zum Staubsaugen einsetzen. Glücklicherweise sammelt sich in unserem Raum der Staub nur an diskreten Punkten an. [...]

a) Beschreiben Sie die Operationen mittels der STRIPS-Notation.

Die Aufgabenstellung scheint nicht viele Regeln aufzubieten, aber die Formulierung als Prädikate bewirkt, dass eine Reihe von Varianten für jede Stelle im Raum entstehen. Alle Prädikate müssen als wahr oder falsch evaluiert werden.

Beschreibung der Welt

- F = { feld00, feld10, feld20, feld01, feld11, feld21, feld02, feld12, feld22 }
- schmutz(X): X ist verschmutzt, X aus F
- roboter(X): Auf X steht der Roboter, X aus F
- orientierung(X) mit X aus { N, S, W, O }: Orientierung des Roboters

Anfangsbedingung:

C={ roboter(feld00), orientierung(N), schmutz(feldmn), ... }

Operationen des Roboters:

- drehe_nord (_links, _rechts), drehe_sued (_links, _rechts), drehe_west (_links, _rechts), drehe_ost (_links, _rechts)
- schritt_nord_xy, schritt_sued_xy, schritt_west_xy, schritt_ost_xy, x,y=0,1,2
- sauge

Daraus die Definitionen

drehe_nord_links()
pre: { orientierung(O) }
del: { orientierung(O) }
add: { orientierung(N) }

drehe_nord_rechts()
pre: { orientierung(W) }
del: { orientierung(W) }
add: { orientierung(N) }

drehe_sued_links()
pre: { orientierung(W) }
del: { orientierung(W) }
add: { orientierung(S) }

drehe_sued_rechts()
pre: { orientierung(O) }
del: { orientierung(O) }
add: { orientierung(S) }

drehe_west_links()
pre: { orientierung(N) }
del: { orientierung(N) }
add: { orientierung(W) }

drehe_west_rechts()
pre: { orientierung(S) }
del: { orientierung(S) }
add: { orientierung(W) }

drehe_ost_links()
pre: { orientierung(S) }
del: { orientierung(S) }
add: { orientierung(O) }

drehe_ost_rechts()
pre: { orientierung(N) }
del: { orientierung(N) }
add: { orientierung(O) }

Für n=0,1,2 bilde folgende Regeln (Operatorschemata):

schritt_nord_n1()
pre: { orientierung(N), roboter(feldn1) }
del: { roboter(feldn1) }
add: { roboter(feldn2) }

schritt_nord_n0()
pre: { orientierung(N), roboter(feldn0) }
del: { roboter(feldn0) }
add: { roboter(feldn1) }

schritt_sued_n2()
pre: { orientierung(S), roboter(feldn2) }
del: { roboter(feldn2) }
add: { roboter(feldn1) }

schritt_sued_n1()
pre: { orientierung(S), roboter(feldn1) }
del: { roboter(feldn1) }
add: { roboter(feldn0) }

schritt_west_1n()
pre: { orientierung(W), roboter(feld1n) }
del: { roboter(feld1n) }
add: { roboter(feld0n) }

schritt_west_2n()
pre: { orientierung(W), roboter(feld2n) }
del: { roboter(feld2n) }
add: { roboter(feld1n) }

schritt_ost_0n()
pre: { orientierung(O), roboter(feld0n) }
del: { roboter(feld0n) }
add: { roboter(feld1n) }

schritt_ost_1n()
pre: { orientierung(O), roboter(feld1n) }
del: { roboter(feld1n) }
add: { roboter(feld2n) }

sauge()
pre: { schmutz(X), roboter(X) }
del: { schmutz(X) }
add: {}

b) Stellen Sie einen Algorithmus vor, den der Roboter auszuführen hat. Lassen Sie ihn probelaufen! (3 Punkte)

- Weltmodell gegeben
- solange Ziel nicht erreicht ist
  - wähle eine Operation P (nach einer bestimmten Strategie oder zufällig)
  - prüfe, ob die Vorbedingung von P erfüllt ist. Dazu gehe die Menge der Formeln der Reihe nach durch und prüfe, ob jede einzelne Formel zurzeit wahr ist. Verwendet man nur simple Formeln (ohne Konnektoren), dann genügt es zu prüfen, ob die Formeln in einer Liste auftauchen. Wenn ja:
    - lösche die logischen Formeln im Weltmodell, die in del angegeben sind
    - füge jene Formeln hinzu, die in add gegeben sind
    - prüfe, ob das Ziel erreicht ist

Das Problem an diesem naiven Algorithmus ist, dass der Roboter möglicherweise überhaupt keinen Fortschritt macht. Er könnte sich endlos in Feld (0,0) im Kreise drehen, weil diese Möglichkeit immer besteht, unabhängig von irgendwelchen Vorbedingungen.

Testlauf:

- W= { roboter(feld00), orientierung(N), schmutz(feld12), schmutz(feld21) }

Wähle Regel schritt_nord_00

- W = { roboter(feld01), orientierung(N), schmutz(feld12), schmutz(feld21) }

Wähle Regel schritt_nord_01

- W = { roboter(feld02), orientierung(N), schmutz(feld12), schmutz(feld21) }

Wähle Regel drehe_ost_rechts

- W = { roboter(feld02), orientierung(O), schmutz(feld12), schmutz(feld21) }

Wähle Regel schritt_ost_02

- W = { roboter(feld12), orientierung(O), schmutz(feld12), schmutz(feld21) }

Wähle Regel sauge

- W = { roboter(feld12), orientierung(O), schmutz(feld21) }

Wähle Regel schritt_ost_12

- W = { roboter(feld22), orientierung(O), schmutz(feld21) }

Wähle Regel drehe_sued_rechts

- W = { roboter(feld22), orientierung(S), schmutz(feld21) }

Wähle Regel schritt_sued_22

- W = { roboter(feld21), orientierung(O), schmutz(feld21) }

Wähle Regel sauge

- W = { roboter(feld21), orientierung(O) }

Ab hier ist das Ziel erreicht, aber wir lassen den Roboter weiterlaufen.

Die angegebenen Formeln helfen eigentlich nur in „Mikroschritten“ und können ein Ziel nicht approximieren. Das heißt, der Roboter wird einfach durch puren Zufall irgendwann den Raum gesäubert haben.

Auf höherer Ebene (hier nicht in STRIPS darstellbar) könnte man Strategien definieren (z.B. erst Reihe 0, dann Reihe 1, dann Reihe 2), um das Ziel zu erreichen.

Deshalb ist es sinnvoll, einen Plan zu definieren (etwa so zu fahren: (0,0) → (0,1) → (0,2) → (1,2) → (1,1) → (1,0) → (2,0) → (2,1) → (2,2) und dabei ggf. zu saugen). Die STRIPS-Notation des gesamten Plans wäre dann

pre: { orientierung(N), roboter(feld00) }

del: { roboter(feld00), schmutz(X) } für alle X aus { feld00, feld01, ... }

add: { roboter(feld22)}

Man würde dann Aktionen ausführen lassen (etwa passende Drehungen und Bewegungen), um die Vorbedingung dieses Plans zu erfüllen. Die Nachbedingung dieses Plans ist, dass alle Felder sauber sind (was im Einzelnen zu begründen wäre) und dass der Roboter im Feld (2,2) steht.

c) Könnte man den Algorithmus auch so gestalten, dass er für viele (wenn nicht alle) solche Planungsprobleme einsetzbar ist? [...]

Der Algorithmus ist nicht spezifisch auf das Saugroboterproblem abgestimmt, also kann man ihn prinzipiell für jedes andere Problem auch einsetzen. Nimmt man diesen naiven Algorithmus, so müsste man einfach jedes Problem in Form von logischen Formeln darstellen (was seinerseits extrem aufwändig werden kann); dies wäre die einzige notwendige Anpassung. Dann ist diese Schleife so lange auszuführen, bis das Ziel erreicht wird (oder nicht mehr erreicht werden kann) – oder gemäß einer anderen, etwas intelligenteren Strategie.

Was nicht allgemein geleistet werden kann, ist die Zusammenstellung von Plänen. Diese sind rein problemabhängig.

Ein einfaches Beispiel, wie schwer das Finden von Plänen ist, ist das Spiel *Sokoban*. Dieses hat sehr einfache Regeln: Verschiebe alle Kisten im Raum auf vorgegebene Plätze. Jedoch kann man nur schieben, aber nicht ziehen. Ohne Backtracking-Verfahren (im Kopf spielt man die Züge voher durch) wird es in der Regel nicht gelingen, das Ziel zu erreichen.