Simulation von Wissen - Modellierung von Adaptivität

>> Achtung : Skript gibt den mündlichen Vortrag nur unvollständig wieder !!! <<<

1. Einführung

In der letzten Sitzung ist die Frage aufgeworfen worden, inwieweit Expertensysteme in der Lage sind, adaptive Prozesse zu modellieren. Es war grundsätzlich festgestellt worden, dass die Expertensystem-Entwicklungsumgebung CLIPS in der Lage ist, ihre Datenbasis während des Ablaufs durch *Wegnahme oder Hinzufügung von Fakten* dynamisch zu ändern. Selbst *Regeln können dynamisch eliminiert* werden. Es bleibt zu klären, was dies für die allgemeine Frage der Modellierung von adaptiven Prozessen bedeutet. Diese Frage soll in dieser Sitzung geklärt werden.

2. Argumentationsstrategie

Um die Mächtigkeit von Expertensystemen ähnlich denen von CLIPS einschätzen zu können, ist es sehr hilfreich, wenn man ihr Verhältnis zum Konzept der *Turingmaschine* bzw. zur *Universellen Turingmaschine* beschreiben kann. Innerhalb der Theorie der Berechenbarkeit (theoretische Informatik, Metamathematik) ist das Konzept der universellen Turingmaschine das stärkste Konzept, das bekannt ist. Eine universelle Turingmaschine kann jede beliebige Turingmaschine simulieren. Insofern die Turingmaschine bis heute das allgemeinste bekannte Konzept berechenbarer Konzepte ist, bedeutet die Aussage, dass eine universelle Turingmaschine jede beliebige Turingmaschine simulieren kann, dass die universelle Turingmaschine alle berechenbaren Prozesse simulieren kann, die sich überhaupt beschreiben lassen.

Die Überlegungen der heutigen Sitzung gehen nun in die Richtung, zu zeigen, dass man mit CLIPS Expertensysteme erzeugen kann, die jede beliebige Turingmaschine simulieren können. Sofern dies gelingt, würde dies bedeuten, dass die mit CLIPS darstellbaren Modelle mindestens so mächtig sein können wie eine universelle Turingmaschine. Daraus folgt, dass die Entscheidung, für eine Modellierung CLIPS zu benutzen statt irgendeinen anderen bekannten Formalismus zur Darstellung von Algorithmen, zumindest theoretisch keinerlei Einschränkung bedeuten würde. In weiteren Überlegungen wäre zu prüfen, in welchem Sinne Expertensysteme im Stile von CLIPS andere Arten von Nachteilen aufweisen, die ihrem Einsatz im Vergleich zu anderen Formalismen

entgegenstehen würden.

Bedenkt man, dass eine universelle Turingmaschine über die allgemeine Simulation von beliebigen Turingmaschinen hinaus prinzipiell auch in der Lage ist, jede zu simulierende Turingmaschine 'nach Bedarf' abzuändern, dann bedeutet dies, dass die Simulierbarkeit von Turingmaschinen die Fähigkeit zur Modellierung beliebiger berechenbarer adaptiver Prozesse mit einschließt. Alle adaptiven Prozesse, die sich überhaupt durch einen Computer modellieren lassen, können durch eine universelle Turingmaschine modelliert werden. Entsprechend gilt, dass ein Expertensystem von der Mächtigkeit von CLIPS, das sich wie eine universelle Turingmaschine verhalten kann, dann genauso --im Prinzip-- alle adaptiven Prozesse simulieren kann, die sich als berechenbare Prozesse darstellen lassen.

Analog wie im ersten Fall der universellen Turingbrechenbarkeit wäre auch in diesem speziellen Fall der Modellierung von adaptiven Prozessen zu prüfen, ob sich Expertensysteme besser oder schlechter für solch eine Modellierung eignen; ' bsser' bw. ' shlechter' wieder mit Blick auf zu definierende Kriterien der Praktikabilität.

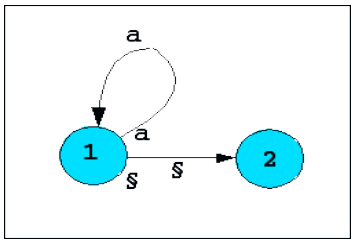
Der Weg in Richtung universelle Turingmaschine wird durch drei Stationen markiert: (i) Mealy-Automaten als Beispiele endlicher Automaten; (ii) Turingmaschinen; (iii) Universelle Turingmaschine.

3. CLIPS und Mealy-Automaten

Da eine Turingmaschine letztlich ein endlicher Automat mit einer zusätzlichen Eigenschaft ist, kann man in einem ersten Schritt den etwas einfacheren endlichen Automaten betrachten, um dann im nächsten Schritt zu zeigen, wie sich eine Turingmaschine simulieren lässt.

Ein Mealy-Automat ist eine von vielen möglichen Varianten von endlichen Automaten, die alle untereinander äquivalent sind.

Ein einfaches Programm für einen Mealy-Automaten wäre das folgende:

	Solange der Automat im Startzustand 1 das Zeichen ' a' liest, bleibt er im Zustand 1 und druckt das Zeichen ' a' auch aus. Sobald er ein leeres Zeichen ' §' liest, geht er in den Endzustand 2 über und hält an.
Alphabet ALPH = {a, §} mit ' §' als *leeres Zeichen*	<--- Alphabet
Zustände STATES = {1,2} ' 2' gilt als ein *Endzustand*.	<--- Zustände
<1,a,a,1> <1,§,§,2>	<--- Regeln

Eine mögliche Umsetzung des formalen Konzeptes eines Mealy-Automaten zum Akzeptieren von beliebigen vielen ' a' Zeichen könnte wie folgt aussehen:

Fakten

```
;;;*****************************
;;; is-sw-unit4-mealy1.clp
;;;*****************************************
;;;
;;; Vorgegeben wird ein Band mit Inhalt,
;;; ein Index idx fuer die Position im Band sowie
;;; der Zustandszaehler pc mit Startwert 1
;;;*********************************************
;;;
;;;**************************
;;;* FACTS                  *
;;;**************************
;;;
(deffacts band
  (pc 1)
  (idx 1)
  (band 1 a)
  (band 2 a)
  (band 3 +))
```

Anmerkung: Im CLIPS-Beispiel wurde das Zeichen ' §durch ' +ersetzt, da '§' ein Sonderzeichen ist und bei der Verarbeitung zu Problemen führt.

CLIPS-Programm des Mealy-Automaten mealy1

```
;;;*********************************
;;; RULES
;;;*********************************
(defrule mealy1
   ?p <- (pc ?x)
   ?i <- (idx ?y)
   ?z <- (band ?y a)
=>
   (retract ?i)
   (assert (idx (+ ?y 1)))
   (facts)
   (printout t "no end state pc = " ?x " index = " ?y " --> " (+ ?y 1)  crlf)
   )
 mealy2
   ?p <- (pc ?x)
   ?i <- (idx ?y)
   ?z <- (band ?y +)
=>
   (retract ?i)
   (assert (idx (+ ?y 1)))
   (retract ?p)
   (assert (pc (+ ?x 1)))
   (facts)
   (printout t "state change pc = " ?x " ---> " (+ ?x 1) " index = " ?y " --> " (+ ?y 1)    crlf)
   )
```

CLIPS-Simulation des mealy-Automaten mealy1

```
CLIPS> (clear)
CLIPS> (load is-sw-unit4-mealy1.clp)
Defining deffacts: band
Defining defrule: mealy1 +j+j+j
Defining defrule: mealy2 =j=j+j
TRUE
CLIPS> (reset)
<== Focus MAIN
==> Focus MAIN
==> Activation 0      mealy1: f-1,f-2,f-3
CLIPS> (facts)
f-0     (initial-fact)
f-1     (pc 1)
f-2     (idx 1)
f-3     (band 1 a)
f-4     (band 2 a)
f-5     (band 3 +)
For a total of 6 facts.
```

Nachdem das Programm mit (load ...) geladen wurde und die Fakten mit (reset) aktualisiert wurden, kann nun das Programm gestartet werden.

CLIPS-Simulation des mealy-Automaten mealy1
CLIPS> (run) FIRE 1 mealy1: f-1,f-2,f-3 ==> Activation 0 mealy1: f-1,f-6,f-4 f-0 (initial-fact) f-1 (pc 1) f-3 (band 1 a) f-4 (band 2 a) f-5 (band 3 +) f-6 (idx 2) For a total of 6 facts.

Dabei ist zu beachten, dass FIRE 1 sich auf die erste Faktenmenge bezieht, in der (idx 1) noch die Nr.2 hatte; nach erster Ausführung wird (idx 1) entfernt und als (idx 2) neu mit Nr.6 eingeführt!

CLIPS-Simulation des mealy-Automaten mealy1
no end state pc = 1 index = 1 --> 2 FIRE 2 mealy1: f-1,f-6,f-4 ==> Activation 0 mealy2: f-1,f-7,f-5 f-0 (initial-fact) f-1 (pc 1) f-3 (band 1 a) f-4 (band 2 a) f-5 (band 3 +) f-7 (idx 3) For a total of 6 facts

Activation ist immer die Vorbereitung auf das nächste ' FIRE' Dabei zeigt sich jetzt, dass bei (idx 3) die Regel *mealy1* nicht mehr angewendet werden kann, dafür aber die Regel *mealy2*.

CLIPS-Simulation des mealy-Automaten mealy1

```
.
no end state pc = 1 index = 2 --> 3
FIRE    3 mealy2: f-1,f-7,f-5
f-0     (initial-fact)
f-3     (band 1 a)
f-4     (band 2 a)
f-5     (band 3 +)
f-8     (idx 4)
f-9     (pc 2)
For a total of 6 facts.
state change pc = 1 ---> 2 index = 3 --> 4
<== Focus MAIN
CLIPS>
```

Dieses Beispiel zeigt, dass sich ein mealy-Automat im Prinzip mittels CLIPS modellieren lassen. Auf die Verallgemeinerung, nämlich einen kompletten mealy-Simulator für beliebige mealy-Programme zu schreiben, wird hier verzichtet. Natürlich wäre dies möglich (siehe Simulator für Mealy-Automat für die grundsätzlichen Anforderungen).

4. CLIPS und Turingmaschine

Der Unterschied zwischen einer Turingmaschine und einem Mealy-Automaten besteht in zwei Eigenschaften: (i) Einmal kann eine Turingmaschine ihren Output auf das gleiche Band schreiben, auf dem auch der Input steht. Sie kann damit durch ihr Verhalten *auf sich selbst zurückwirken*! (ii) Zum anderen kann eine Turingmaschine den Lese- und den Schreibkopf in *beliebige Richtungen* bewegen. Damit kann sie sowohl beliebige *Kopien* anlegen und damit eine Form von *Gedächtnis* realisieren, sie kann aber auch beliebige Bandinhalte *überschreiben*, was impliziert, sie kann beliebige *Änderungen* vornehmen; damit ist grundsätzlich die Möglichkeiten gegeben, *Lernen*, sofern es auf Änderungen beruht, zu realisieren.

Fragt man sich, ob man mit CLIPS auch Turingmaschinen simulieren kann, dann kann man diese Frage aufgrund der bisherigen Daten positiv beantworten.

Denn die Eigenschaft (i), dass die Turingmaschine den Output auf das Inputband schreiben kann, wird dadurch realisiert, dass die Fakten, die ein CLIPS-Programm erzeugt, jenen Fakten zugefügt werden kann, die das Inputband repräsentieren.

Die Eigenschaft (ii), dass der Lese- und Schreibkopf sich in beliebige Richtungen bewegen kann, wird dadurch realisiert, dass aus der Faktenmenge über Regeln beliebige Fakten des Bandes ausgewählt werden können.

Also gibt es prinzipiell keine Hindernisse, um mit einem CLIPS-Programm eine Turingmaschine zu simulieren.

5. CLIPS und Universelle Turingmaschine

Da eine universelle Turingmaschine ja nichts anderes ist als eine Turingmaschine, die beliebige Turingmaschinen simulieren kann, folgt aus den vorausgehenden Überlegungen, dass man mit einem CLIPS-Programm auch eine universelle Turingmaschine simulieren kann.

6. Lernende Programme

Nach diesen Vorüberlegungen kann man sich jetzt nochmals der Frage zuwenden, Welche Eigenschaften eines CLIPS-Programms genau für die Lernfähigkeit notwendig sind und -- weiterführender-- was genau nun für einen Wissensassistenten an Eigenschaften gefordert werden müsste, damit er die Form der Lernfähigkeit besitzt, die für die von den Anwendungsfällen her sichtbar werdenden Eigenschaften notwendig und hinreichend wären.