1 DATR-General Knowledge Representation

1.1 Was ist DATR ?

DATR ist eine Programmiersprache zur Wissensrepräsentation. Sie wurde 1989 von Roger Evans (University of Brighton) und Gerald Gazdar (University of Sussex) zur einfachen Darstellung von Lexikalischen Wissen entwickelt. Teile von Latein, Arabisch und nahezu allen europäischen Sprachen wurden erfolgreich in DATR abgebildet. DATR ist Turing und damit nicht nur auf die reine Wissensdarstellung beschränkt.
DATR benutzt Vererbung, eine natürliche Eigenschaft der Darstellung von Wissen. Die Vererbung ist orthogonal, das bedeutet Mehrfachvererbung ist möglich, sofern die ererbten Attribute disjunkt sind. Die Definition von Nichtmonotonie ist einfach möglich.
DATR ist unter den Linguisten weit verbreitet und es gibt zahlreiche Implementationen auf DOS und UNIX Rechnern.

1.2 Motivation zur Entwicklung von DATR

Die wichtigsten Darstellungen von Wissen sind Logik und Semantische Netze. Die Vorteile der Darstellung von Wissen in Logik liegen in der einfachen Syntax und klaren Definition der Semantik. Inferenzen lassen sich somit einfach ableiten. Es ist jedoch schwierig, Inferenzmaschinen zu bauen, die Ergebnisse effizient herleiten.
Ein Vorteil der Semantischen Netze liegt in der schnellen Ableitung von Inferenzen. Sie sind das Folgen von Pointern. Nachteile von Semantischen Netzen ist häufig das Fehlen einer formalen Semantik. Zur Verdeutlichung werden im folgenden noch einmal die Schwächen der frühen KI-Sprachen zusammengefaßt.
Semantische Netze sind schon seit etwa 1968 bekannt. 1975 prägte Marvin Minsky den Begriff FRAME. Sie dienen dem Wiedererkennen von Objekten, der Beantwortung von Fragen über stereotype Objekte und dem Handeln bei stereotypen Ereignissen.
In FRL hat jeder FRAME einen speziellen AKO SLOT. Ein FRAME erbt von jedem FRAME zu dem er einen AKO Link hat. Dieses kann zu Widerprüchen führen, wie folgendes Beispiel zeigt:

- Elephants are gray
- Royal elephants are elephants
- Royal elephants are not gray
- Clyde is a royal elephant
- Clyde is an elephant

Es ist nicht entscheidbar, welche Farbe Clyde hat. Damit ist die Semantik des Netzes nicht eindeutig festgelegt. In einigen Sprachen suchen Inferenzalgorithmen solange, bis sie einen Attributwert gefunden haben, der eine Anfrage befriedt. Diese Methode entspricht der *shortest path inference*. Dieser Weg ist indiskutable, da der so ererbte Wert oftmals nicht die echte Ober-Unterklassenhierachie wiedergibt, falls ein FRAME von mehreren anderen FRAMES erbt.
Auch bei NETL gibt es keine formale Semantik für das Netz. So kann es verschiedene Implementationen geben, denen das gleiche Netz zugrundeliegt, die aus diesem jedoch verschiedene Inferenzen ziehen. Die Bedeutung des Netzes wird also nur durch Analyse des Quelltextes klar. Diese Ansätze sind unbefriedigend.
Fehlend ist die explizite Beziehung zwischen der formalen Semantik und den Operationen auf dem Graphen. Um diese Problem zu lösen, hat man versucht, den Semantischen Netzen eine Semantik zu geben, indem man sie in Logik übersetzte. Bei Monotonen Semantischen Netzen ist dieses sehr einfach, da eine Teilmengenbeziehung einfach durch Implikation ausgedrückt werden kann. Schwierig wird es bei Nichtmonotonen Semantischen Netzen. Hier sei auf Touretzkys Arbeiten verwiesen. Er benutzt Modaloperatoren und führt für das Ableiten von Inferenzen eine dreiwertige Logik ein.
Aus den oben beschriebenen Schwächen ergeben sich folgende Anforderung an DATR:

- Explizite Theorie der Vererbung
- Expliziet Deklarative Semantik
- Effiziente und einfache Implementation
- Einfache Darstellung lexikalischer Einträge
- Einfache Darstellung von Generalisierung und Spezialisierung

1.3 Definition von Begriffen in DATR

Ein Beispiel für ein DATR Programm:

```
Knoten : Pfad == Wert
Knoten : Pfad =  Wert
```

Beispiel für eine Ableitung:

```
Anfrage: N:<q r>
Satz:    N:<q> == <a <b <c <d> e> f> g>

Ableitung:      N:<d r> = A1
           N:<c A1 e r> = A2
           N:<b A2 f r> = A3
           N:<a A3 g r> = A4

Theorem: N:<q r> = A4
```

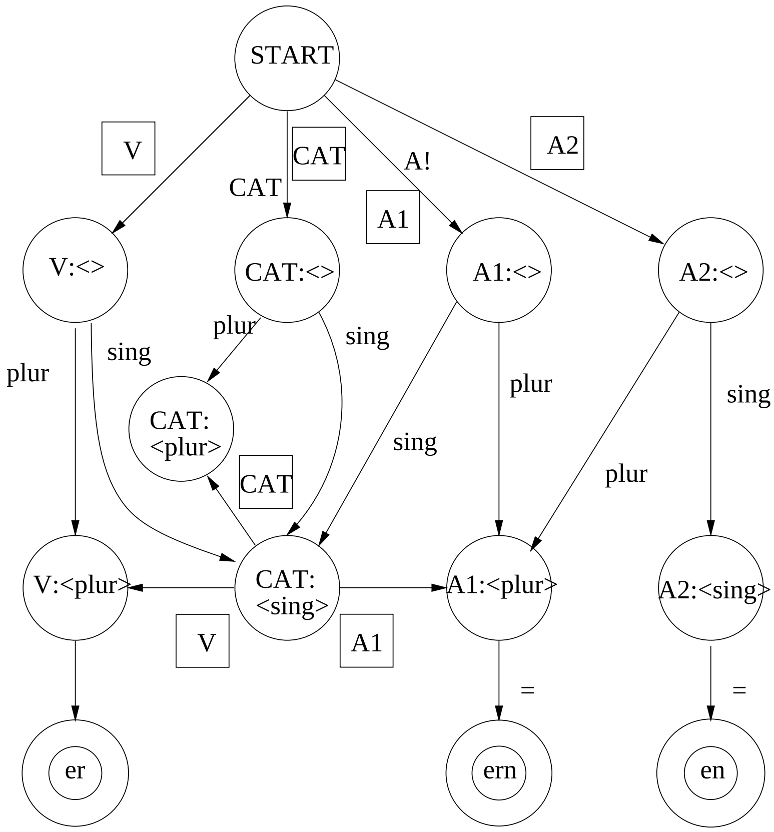
Zunächst einige Begriffe: Eine Repräsentation in DATR heißt eine DATR Theorie. Sie beschreibt das Wissen. Die Theorie besteht aus Sätzen. In Objektorientierten Sprachen würde man die Sätze wahrscheinlich als Objekte auffassen. Ein Satz besteht aus Knoten, Pfaden und Werten. Der Knoten ist vergleichbar mit dem Namen des Objektes, der Pfad mit dem Attributnamen und der Wert mit dem Attributwert. Der Wert kann dabei ein Verweis sein oder ein atomarer Wert. Handelt es sich um einen Verweis, so nennt man den Satz definitorisch. Die Pfad-Wertgleichung wird dann durch ein doppeltes Gleichheitszeichen beschrieben. Ein Satz mit einem atomaren Wert als rechte Seite heißt extensionaler Satz. Er wird als Theorem bezeichnet. Eine Ableitung in DATR ordnet einem Knoten-Pfad Paar einen Wert zu, so daß sich ein Theorem ergibt.
Jetzt soll eine einfache Ableitung in DATR beschrieben werden. Die Anfrage an die Theorie heißt `N:<q r>`. `N` ist der Knoten. Der Pfad wird immer in spitzen Klammern geschrieben. In der Theorie existiert ein Satz, dessen Pfad ein Präfix des Anfragepfades ist. Es wird immer der Satz mit dem längsten gemeinsamen Präfix ausgewählt. Im Beispiel also `N:<q>`. Die rechte Seite ist ein Verweis. Der Knotenname fehlt. Deshalb muß der richtige Name dem Kontext entnommen werden. In diesem Beispiel ist der Kontext `N`. Die rechte Seite besteht aus geschachtelten Pfaden. Die Auswertung beginnt immer von innen nach außen. Dabei werden die Verweise auf der rechten Seite um einen Suffix ergänzt (im Beispiel `r`), so daß gilt: der Pfad auf der linken Seite konkateniert mit dem Suffix ergibt den Anfragepfad. Der Suffix wird an

jeden Verweis auf der rechten Seite angehängt, unabhängig von vorherigen Auswertungen der Verweise. Ist der Wert auf der rechten Seite ein atomarer Wert, so wird der Suffix nicht angehängt.
Ein Kontext in DATR besteht aus einem Knoten und einem Pfad. Man unterscheidet zwischen drei Arten von Kontexten: lokaler Kontext, globaler Kontext und externer globaler Kontext. Jeder Kontext besteht aus einem Knoten und einem Pfad. Zu Beginn einer Anfrage werden alle drei Kontexte auf den Anfrageknoten bzw. Anfragepfad gesetzt. Die Verweise auf der rechten Seite eines Satzes können entweder aus einem Knoten-Pfad Paar, einem Knoten oder einem Pfad bestehen. Alle drei Verweisarten können lokal oder global sein. Globale Verweise werden in Anführungsstriche gesetzt. Ist der Verweis ein globaler oder lokaler Knoten, so wird er um einen globalen oder lokalen Pfad ergänzt, der dem globalen oder lokalen Kontext entnommen wird. Entsprechendes gilt, wenn der Verweis aus einem Pfad besteht. Durch einen Verweis wird immer ein Kontext geändert. Bei Angabe eines lokalen/globalen Knotens wird dieser der aktuelle lokale/globale Knotenkontext. Entsprechendes gilt falls der Verweis ein Pfad oder ein Knoten-Pfad Paar ist.
Der Externe Globale Kontext wird zu keinem Zeitpunkt verändert. Er stellt also immer die ursprüngliche Anfrage dar.

1.4 Semantik von DATR

Neben der Operationalen Semantik von DATR, dargestellt als Prolog Source Code, findet man in der Literatur auch eine Denotationale Semantik. Auf diese möcht ich hier nicht eingehen. Interessant ist jedoch die Möglichkeit, eine DATR Theorie als eine Art Endlichen Automaten darzustellen. Der Automat wurde um einen Speicher für einen Knoten erweitert. Im nachfolgenden Beispiel ist dieser Speicher durch ein Rechteck gekennzeichnet. Kreise entsprechen Zuständen, doppelte Kreise akzeptierten Zuständen. Bei Endlichen Automaten sind Kanten mit Wörtern markiert. Den Kanten darf gefolgt werden, wenn ein Präfix der Eingabe diesem Wort entspricht. Bei dem folgenden Automaten gibt es zwei zusätzliche Kantentypen: Kanten mit einem Rechteck und Kanten mit einem Rechteck und einem Wort. Kanten mit einem Rechteck und einem Wort darf gefolgt werden, wenn ein Eingabepräfix dem Wort entspricht. Zusätzlich wird der Inhalt des Rechecks der Inhalt des Speichers. Einer Kante, die nur mit einem Rechteck markiert ist, darf nur gefolgt werden, wenn der Inhalt des Rechtecks gleich dem Inhalt des Speichers ist.
Beispiel:

$$
\begin{array}{llll}
\text{CAT:} & < sing > & == & \text{“}< plur >\text{”} \\
\text{V:} & < sing > & == & \text{CAT} \\
 & < plur > & == & \text{er.} \\
\text{A1:} & < sing > & == & \text{CAT} \\
 & < plur > & == & \text{ern.} \\
\text{A2:} & < sing > & == & \text{en} \\
 & < plur > & == & \text{A1.}
\end{array}
$$



1.5 Inferenzregeln

Es gibt in DATR sieben Inferenzregeln. Im folgenden bezeichne A einen atomaren Wert, N und N_R seien Knoten und V ein Verweis und P und P_R seien Pfade. Die erste Regel beschreibt, daß ein definitorischer Satz zu einem extensionalen Satz abgeleitet werden kann, wenn die rechte Seite der Gleichung ein atomarer Wert ist. Folgende Regeln der Form $V \; \{ \; A/N_R : P_R \; \}$ bedeuten die Ersetzung jedes Vorkommens von $N_R : P_R$ in V durch A. Bei der dritten Regel wird der Knoten N_R um den lokalen Kontext P ergänzt und anschließend durch A ersetzt. In der vierten Regel wird ein lokaler Pfad um einen Knoten ergänzt, der dem lokalen Kontext entnommen wird, und dann ersetzt. Die restlichen drei Regeln haben die gleiche Bedeutung im globalen Kontext. Ein Knoten und/oder Pfad in Anführungsstrichen bedeutet, daß der globale Kontext auf diesen Wert geändert wird. Falls nur ein Pfad in Anführungsstrichen angegeben ist, so wird der fehlende Knoten dem globalen Kontext entnommen. Entsprechendes gilt bei fehlenden Pfad. Sind Knoten und Pfad angegeben, so wird lediglich der Kontext geändert.

Inferenzregeln (Evans & Gazdar 1989):

$$\begin{array}{lll}
N : P == A. & & \Rightarrow N : P = A. \\
N : P == V. & \wedge N_R : P_R = A. & \Rightarrow N : P == V\{A/N_R : P_R\}. \\
N : P == V. & \wedge N_R : P = A. & \Rightarrow N : P == V\{A/N_R\}. \\
N : P == V. & \wedge N : P_R = A. & \Rightarrow N : P == V\{A/P_R\}. \\
N : P == V. & \wedge N_R : P_R = A. & \Rightarrow N : P == V\{A/''N_R : P_R''\}. \\
N : P == V. & \wedge N_R : P_G = A. & \Rightarrow N : P == V\{A/''N_R''\}. \\
N : P == V. & \wedge N_G : P_R = A. & \Rightarrow N : P == V\{A/''P_R''\}.
\end{array}$$

1.6 DATR BNF

$$
\begin{array}{lcl}
< sentence > & ::= & < node >:< eqseq >. \\
< eqseq > & ::= & < equation > \mid < equation >< eqseq > \\
< equation > & ::= & < lhs >< rhs > \\
< lhs > & ::= & << atomseq >> \\
< rhs > & ::= & < extrhs > \mid < defrhs > \\
< extrhs > & ::= & =< atomval > \\
< atomval > & ::= & < atomseq > \mid (< atomseq >) \\
< atomseq > & ::= & < epsilon > \mid < atom >< atomseq > \\
< defrhs > & ::= & ==< descval > \\
< descval > & ::= & < descseq > \mid (< descseq >) \\
< desqsec > & ::= & < epsilon > \mid < desc >< desqsec > \\
< desc > & ::= & < atom > \mid < pointer > \\
< pointer > & ::= & \text{``} < spec > \text{``} \mid < spec > \\
< spec > & ::= & < node >:< descpath > \mid \\
 & & < node > \mid < descpath > \\
< descpath > & ::= & << descseq >> \\
< node > & ::= & < upper\_case >< char\_string > \\
< atom > & ::= & < not\_upper\_case >< char\_string > \mid \\
 & & 'any\_string' \\
< char\_string > & ::= & < epsilon > \mid \\
 & & < char >< char\_string > \\
< any\_string > & ::= & < epsilon > \mid \\
 & & < any >< any\_string > \\
< res\_char > & ::= & [:<>'' \; () >] \\
< char > & ::= & [I2CHAR(33)...I2CHAR(127)] \\
 & & -[< reschar >] \\
< any > & ::= & [I2CHAR(0)...I2CHAR(127)] \\
< upper\_case > & ::= & [A - Z] \\
< not\_upper\_case > & ::= & [< char >] - [< uper\_case >]
\end{array}
$$

1.7 Beispielprogramme

Bei der Auswertung gilt wird immer der Pfad mit dem längsten zur Anfrage passenden Präfix gewählt:

Beispiel: Bool'sche Logik

```
B:
<> == false                     (1)
<or> == true                    (2)
<if> == true                    (3)
<not false> == true             (4)
<and true true> == true         (5)
<if true false> == false        (6)
<or false false> == false       (7)

Anfrage:
B:<if <and <or true false> <and true true>> false>

Ableitung:
B:<if <and true true> false>           (2) (5)
B:<if true false>                      (5)
false                                  (6)


Anfrage:
B:<if <and true false> false>

Ableitung:
B:<if false false>                     (1)
true                                 (3)
```

Das folgende Beispiel demonstriert die Darstellung eines Endlichen Automaten in DATR:

```
S1:
  <subj 1 sg> == ni S2:<>  (1)
  <subj 2 sg> == u  S2:<>  (2)
  <subj 3 sg> == a  S2:<>  (3)
```

```
  <subj 1 pl> == tu S2:<>  (4)
  <subj 2 pl> == m  S2:<>  (5)
  <subj 3 pl> == wa S2:<>  (6)

S2:
  <past> == li S3:<>       (7)
  <futr> == ta S3:<>.      (8)

S3:
  <obj 1 sg> == ni S4:<>   (9)
  <obj 2 sg> == ku S4:<>   (10)
  <obj 3 sg> == m  S4:<>   (11)
  <obj 1 pl> == tu S4:<>   (12)
  <obj 2 pl> == wa S4:<>   (13)
  <obj 3 pl> == wa S4:<>.  (14)
S4:
 <like> == penda.

Anfrage:
S1:<subj 1 sg past obj 1 sg>

Ableitung:
ni S2:<past obj 1 sg> (1)
ni li S3:<obj 1 sg>   (7)
ni li ni
```

Im nächsten Beispiel werden Listen in DATR definiert. Die Listeneinträge sind durch ! getrennt. Variablen werden in DATR mit $ gekennzeichnet. Man kann sich vorstellen, daß Sätze mit Variablen so oft dubliziert werden, wie es verschiedene Werte für die Variablen gibt. So gibt es für (2) also folgende Sätze: $N :< firstT >= (T < first >)$, $N :< firsth >= (h < first >)$, $N :< firsto >= (o < first >)$, $N :< firstr >= (r < first >)$ ...

```
# vars $attribute: T h o r s t e n.

N:
  <first !)           == ()                          (1)
  <first $attribute> == ($attribute <first>)   (2)
  <rest>             == ()                          (3)
  <rest !>           == <copy>                      (4)
```

```
  <rest $attribute)  == <rest>                (5)
  <copy $attribute>  == $attribute <copy>     (6)
  <copy>             == ()                    (7)

Anfrage:
N:<first Th!orsten>

Ableitung:
N: T <first h!orsten>   (2)
N: Th <first !orsten>   (2)
N: Th                   (3)

Anfrage:
N:<rest T!horsten>

Ableitung:
N:<rest !horsten>      (5)
N:<copy horsten>       (4)
horsten                (6) (6) (6) (6) (6) (6) (7)
```

1.8 Pures DATR

Pure DATR ist semantisch äquivalent zu DATR. In Pure DATR wird auf Knoten und globale Vererbung verzichtet. Die Abbildung von DATR auf Pure DATR geschieht folgendermaßen: Der Knoten eines Satzes wird als Teil des Pfades aufgenommen. Danach werden soviele Varianten des Satzes gebildet, wie es verschiedene Knoten in der Theorie gibt. Jede Variante hat einen anderen Knoten als erstes Attribut des Pfades. Zu einem Satz `N:<b>` in DATR gibt es also n Sätze in Pure DATR: `<N1 N b> <N2 N b> ... <Nn N b>`. Damit kann auf den globalen Knoten über das erste Attribut eines Pfades jederzeit zugegriffen werden. Ähnlich wie der globale Knoten in den Pfad aufgenommen wurde, kann auch der globale Pfad in den Pfad aufgenommen werden. Mit den im vorherigen Abschnitt beschriebenen Sätzen zur Bearbeitung von Listen ist es möglich, den globalen Pfad über ein Trennzeichen hinter den lokalen Pfad anzuhängen und mit Listenfunktionen auf ihn zuzugreifen. Damit ist DATR auf lokale Vererbung beschränkt worden.