

INTERFACING A SEMANTIC NET WITH AN AUGMENTED TRANSITION NETWORK

Joachim H. Laubsch
Institut fuer Informatik
Universitaet Stuttgart
West-Germany

The main advantages of semantic ATN-grammars are efficient execution and easy implementation in small discourse domains. The drawback lies in the growth of the semantic grammar if the domain of discourse is to be extended. The thesis of this paper is that a higher level of modularity for grammar rules can be attained by making semantic ATNs sharable through schemata (called "concepts") of the knowledge base. A workspace of activatable subATNs guides the parser in choosing among competing PUSH arcs. Expected subATNs are entered into this workspace either indirectly from the schemata through a filtering mechanism, or directly as a side effect of actions on arcs which instantiate a concept.

1. Introduction

Question-answering systems such as LUNAR [13], SOPHIE [4], PLANES [12], and various applications of LIFER [7] contributed valuable methodological tools for the development of a natural language user interface. Of these, semantic ATN-grammars and semantic nets have been successfully used in our group in several small projects [1,11]. The drawback of semantic grammars lies in the growth of the ATN if the discourse domain is enlarged. We discuss an approach to reduce the task-specificity of semantic ATNs by interfacing them with an inheritance network which performs many of the tasks usually embedded in tests and actions of a semantic grammar.

An object-oriented representation language - called OBJTALK [9] - will be used to illustrate the interface between ATN and semantic net. OBJTALK was inspired by SMALLTALK [6], FRL [10], KRL [3], KLONE [2], and DIRECTOR [8].

The basic ideas of object-oriented knowledge representation are:

- (1) objects are active data structures with behavioral traits, such as
 - slot & filler - pairs (where the filler may be defaulted, restricted etc) and
 - methods consisting of a pattern to receive a message and a body describing what is to be done.
- (2) behavioral traits are inherited in a **generalisation hierarchy** (or network). An object is either a **class** or an **instance**. Each object has one or several superclasses from which it inherits behavioral traits recursively up to a **root** node.
- (3) An object will exhibit behavior if it is sent a message.

2. Procedural attachment of ATNs to concepts in an inheritance net

In order to achieve semantic-driven parsing, a semantic SubATN should be activated whenever a particular **word** or **concept** is found. The basic mechanism is to attach **trigger-keys** to a schema that state which words or concepts put some named subATNs in a preferred activatable state. Once activated, a subATN may access the slots of the schema in tests or actions, and/or pop a structure used to fill slots of that schema. The inheritance structure between concepts is defined such that semantic subATNs are sharable among many concepts.

A concept definition in OBJTALK has the form:

```
(Concept <concept-name>
  (Individuates <name of a super-concept> ...)
  (Generic-properties
    <Slot>: <Filler description>
    ...)
  (Trigger-atn <Trigger-keys> ; attached ATN
    [<subATN-node> <production>] ...))
```

The Trigger-atn form is a special case of pattern-directed message receipt by an object. The Trigger-keys serve as activation keys for a **bottom up** instantiation of the current concept through words or other concepts. The form of **Trigger-keys** is:

```
(triggered-by: {(Words <lexical entry> ...)
  {(Concepts <concept name> ...)}}
```

If

- (1) a concept has been **partially** instantiated (an

instance of this concept or one lower in the hierarchy was made, but not all obligatory slots are given values yet), or

(2) one of the **concepts** mentioned in trigger-keys has been instantiated, or

(3) one of the **words** mentioned in trigger keys was entered into the chart during a lexical prepass,

then the concept's Trigger-atn form is evaluated. This has the effect that all subATN-nodes named in it are given a preferred activation status (leading the parser to try PUSHes to those subATNs first). The slots in the concept can be accessed from the ATN analogous to registers. The register **self** contains the name of the activating concept.

The <production> describes what will be done with the result of the activated subATN in case it was left by a POP arc. Its form is:

```
[(<Filter> ...) <action> ...]
```

The convention is that the value returned by a POP is a nested a-list. The filters match such an a-list, thereby binding filter variables. A (sequence of) path-indicator(s) [!<symbol> ...] is used to find the embedded value in an a-list.

The form of the <Filter> is

```
($!<symbol> {$!<symbol> ...} .  
  <list of constants, variables and slot-values>)
```

All filters have to match successfully. A fail action in <action> will try alternative matches if possible or finally fail back to the ATN. A success deactivates the current <subATN node>.

3. Concept-driven control structure

We are using an ATN compiler developed by BURTON [5], modified to our MacLISP. The depth-first control structure of this parser leads to a large number of back-ups because, from a single node, many PUSH arcs are possible. We want to reduce the number of failing PUSHes and POPs by entering those subATNs occurring in evaluated Trigger-atn forms into a workspace of **activatable subnets (WAS)**.

A **morphological prepass** extracts wordclass, features etc and puts them into the **chart**. At this point, trigger-atn procedures associated as the words-part of trigger-keys may enter WAS. If the ATN machine reaches a node, containing PUSH arcs, a check is made to see whether any of them is among the expected subATNs in WAS. In case of a word-triggered subATN an additional test ensures that the triggering word has almost been reached by the scanner. This is psychologically motivated, since we think of parsing as basically a serial process with only a small look-ahead (of 2 - 4 words).

New trigger-atn procedures may be added to WAS in basically two ways:

(1) **indirectly** by processes in the inheritance network: A triggered subATN returns as the form of its POP arc an a-list which (partially) instantiates a concept. If instantiation is partial, the other subATNs stay activatable, else all are deactivated. Those attached subATNs where a concepts-part of trigger-keys contains an instantiated concept enter WAS. Also actions of productions may instantiate new concepts whose attached subATNs will become activatable.

(2) **directly** by the ATN as a side-effect of evaluating an action on an arc, if the action instantiates a new concept.

Indirect instantiation of concepts via a-lists allows a high generalizability of subnets - in the extreme to purely syntactic ATNs. On the other hand, **direct instantiation** presupposes that at the level of the ATN enough task-specific knowledge was coded to instantiate a concept, which in the extreme would result in a purely semantic grammar. The grammar programmer may choose between both alternatives. The ATN-programming style may change towards greater modularity, since PUSHes can be used more extensively.

4. Examples of an ATN <-> semantic net interface

Example 1: It shows **indirect instantiation** of concepts via a trigger-atn procedure attached to a particular attribute. The concept 'consumption' is defined as follows:

```
(Concept consumption  
  (Individuates Quant/attr/prod) ; an attribute  
  (Generic-properties  
    of/product: (class PRODUKT)  
    consumed/quantity: (class QUANTITY) ...)  
  (Trigger-atn  
    (triggered-by: (Words verbrauch- hoch Hoehe ...))  
    [Q/PROP/OBJ ; subATN for query of object's property  
      ((!OBJ1 (PROP: $?PROP))  
       (!OBJ2 (PROD/CODE: $?PROD))) | filter  
       (Ask PRODUKT make: P with: $-,OBJ1)  
       ; make an exemplar of a prod/code product  
action (Ask self set: of/product: $=P)  
       ; retain it in my of/product slot  
       (Ask P prop: $?PROP)  
       ; ask it how much it consumes on that  
       ; attribute ;)])
```

Q/PRP/OBJ can be shared by different attributes and products.

Example 2: This example illustrates how a subATN **directly instantiates** a concept which in turn makes further subATNs activatable. Consider sentences like:

```
>> Klaus faehrt mit dem Zug um 17:35  
    (Klaus goes by train at 17:35)
```

>> Klaus fliegt mit Gerhard von Stuttgart nach Berlin
(Klaus flies with Gerhard from Stuttgart to Berlin)

For concepts like **fly**, **drive** etc a **super-concept** **move** is defined:

```
(Concept fly
  (Individuals move)
  (Generic-properties
    instr: (default FLIGHT)
    origin: (constraint HAS-AIRPORT)
    destin: (constraint HAS-AIRPORT)))
```

```
(Concept move
  (Individuals act/people)
  (Generic-properties
    agent: (class PERSON)
    coagent: (class PERSON)
    instr: (constraint TRANSP/MED?)
    origin: (class LOC)
    destin: (class LOC)
    dep/time: (constraint (BEFORE * $=arr/time))
    arr/time: (constraint (BEFORE $=dep/time *))
    ...))
```

Trigger-atn | explained below

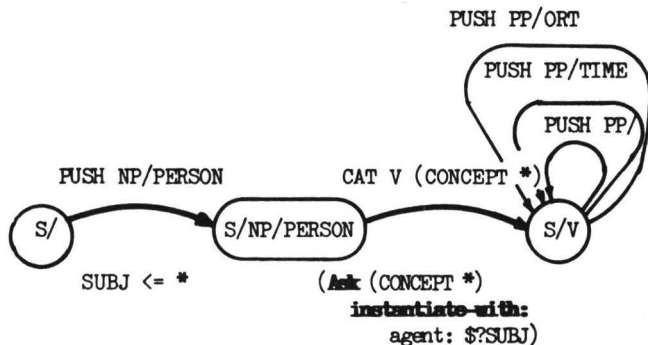


Fig.1: Triggering subATNs by instantiating a concept through an arc action

Parsing leads to instantiation of a concept (like **fly** or **drive**) as a side-effect of an arc action. This concept is a key in **move** and activates the subATNs contained in it:

```
(Trigger-atn
  (triggered-by (concepts fly drive walk ...))
  (PP/ORT [((($!HEAD $?ORT) ($!PREP nach))
    (Ask self set: destin: $?ORT))
    [((($!HEAD $?x) ($!PREP von))
    (Ask self set: origin $?ORT))]
  (PP/ [((($!HEAD $?x) ($!PREP mit))
    (cond ((Ask $?x TRANSP/MED: ?)
      (Ask self set: instr: $?x))
      ((Ask $?x PERSON: ?)
      (Ask self set: coagent: $?x))
      ((FAIL))))))
  (PP/TIME ...) ...)
```

Among the many PUSH arcs from S/V (see Fig. 1) the subset {PP/ORT PP/TIME PP/} is preferred, expressing the expectation to associate only a restricted subset of cases, each signalled by particular syntactic constructions.

5. Conclusions

The proposed ATN-semantic net interface allows higher generality and extensibility of the semantic grammar because we may use PUSH arcs more freely. There is increased work in the filtering phase (reducible by pattern-compilation). The interface has been tested in a semantic grammar for the micro-world of soccer. In our current research we concentrate on observing the grammar programming process in order to establish guidelines for semantic ATN design.

REFERENCES

- [1] Barth, K. "Zur Implementierung eines Lehrsystems fuer LISP." Diplomarb. 62, IFT-UNI Stuttgart, 1977.
- [2] Brachman, R.J. "A Structural Paradigm for Representing Knowledge." BEN Rep. No. 3605, Cambridge, 1978.
- [3] Bobrow, D.G. & Winograd, T. "An overview of KRL, a knowledge representation language." *Cognitive Science*. 1:1 (1977) 3-46.
- [4] Brown, J.S., Burton, R.R. & Bell, A.G. "SOPHIE - a Sophisticated Instructional Environment for teaching electronic trouble-shooting." BEN Rep. 2790, Cambridge, 1974.
- [5] Burton, R.R. "Semantic Grammar: An engineering technique for constructing natural language understanding systems." BEN Rep. 3453, Cambridge, 1976.
- [6] Goldberg, A. & Kay, A. (Eds.) "SMALLTALK-72 Instructional Manual." XEROX, PARC, Palo Alto, 1976.
- [7] Hendrix, G., Sacerdoti, E.D., Sagalowicz, D. & Slocum, J. "Developing a natural language interface to complex data." *ACM Trans. on Database Systems*. 3:2 (1978) 105-147.
- [8] Kahn, K.M. "DIRECTOR Guide." MIT-AI Memo 482, Cambridge, 1978.
- [9] Laubsch, J.H. "OBJTALK - eine auf semantischen Netzen basierende Sprache zur Darstellung von Wissen". MMK Memo 12, IFT-UNI Stuttgart, 1979.
- [10] Roberts, R.B. & Goldstein, I.P. "The FRL Manual." MIT-AI Memo 409, Cambridge, 1977.
- [11] Schairer, M. "Darstellung des zum Verstehen einfacher LISP Programme benoetigten Wissens in FRL." Diplomarbeit # 76, IFT-UNI Stuttgart, 1978.
- [12] Waltz, D.L. "An English Language Question Answering System for a Large Relational Database." *CACM*. 21:7 (1978) 526-539.
- [13] Woods, W.A., Kaplan, R.M. & Nash-Webber, B. "The LUNAR sciences natural language information system." BEN Rep. No. 2378, Cambridge, 1972.