Augmented Transition Networks (ATNs)

Notes on Augmented Transition Network Parsing

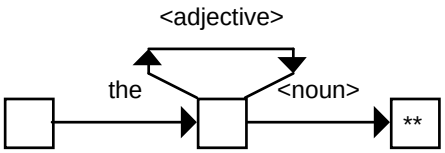
Transition network parsers can be viewed much like a finite state machine which is capable of recognizing *regular* languages. Allowing the networks to make recursive calls to other networks (or themselves) is a extension of transition network parsers to recursive transition network parsers (RTN parsers). These RTN parsers have the power of context-free grammars. A final, critical addition to RTN parsers is the ability to make changes to the contents of a set of registers associated with the networks, and whose transitions can be conditional on the contents of those registers. This is done by adding to each arc in the transition network an arbitrary condition which must be satisfied in order for the arc to be followed, and a set of structure building actions to be executed if the arc is followed.

Augmented transition network (ATN) parsers have the power of a Turing machine since they have changable registers and can transfer control depending on the state of those registers. Clearly they can handle any type of grammar which could possibly be parsed by any machine.

Let us now turn to some simple examples:

Example 1 - *a Finite State Transition Diagram*

The following finite state transition diagram, in which the start state is the left-most node and the final state is labeled **, will accept any sentence that begins with *the*, ends with a noun, and has an arbitrary number of adjectives in between.



Lets follow through the net with the input phrase *the pretty picture*. We start in the START state and proceed along the arc labeled the, because that is the left-most word in the input string. This leaves us in the middle box, with *pretty picture* left as our string to be parsed. After one loop around the adjective arc, we are again at the middle node, but this time with the string *picture* remaining. Since this word is a noun, we proceed to the FINAL node, **, and arrive there with no words remaining to be processed. Thus the parse is successful; in other words, our sample finite state transition diagram accepts this string.

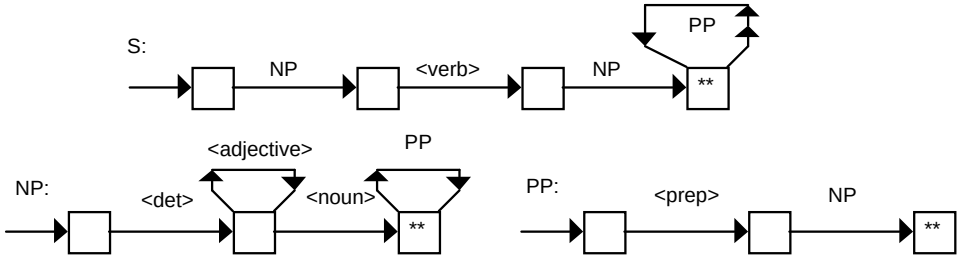
A natural extension to finite state transition diagrams provides a recursive mechanism such as the one mentioned above. A recursive transition network is a finite-state transition diagram in which the labels of any arc may include not only terminal symbols but also nonterminal symbols that denote the name of another subnetwork to be given temporary control of the parsing process.

The RTN operates similarly to a finite-state transition diagram. If a label on an arc is a terminal (word or word class), the arc may be taken, as in finite-state transition diagrams, if the word being scanned matches the label. For example, the word *ball* would match the arc labeled <noun> but not the one labeled <adjective>. Otherwise, if the arc is labeled with a nonterminal symbol, representing a syntactic construct (e.g., *Prepositional Phrase*) that corresponds to the name of another network, the current state

of the parse is put on a stack and control is transferred to the corresponding named subnetwork, which continues to process the sentence, returning control when it finishes.

Whenever an accepting state is reached, control is transferred to the node obtained by "popping the stack" (i.e., returning to the point from which the subnetwork was entered). If an attempt is made to pop an empty stack, and if the last input word was the cause of this attempt, the input string is accepted by the RTN; otherwise it is rejected. The effect of the arcs labeled with names of syntactic constructs is that an arc is followed only if a construction of the corresponding type follows as a phrase in the input string.

Example 2 - *a Recursive Transition Network* - Consider the following example of an RTN



Here *NP* denotes a noun phrase; *PP,* a prepositional phrase; *det,* a determiner; *prep,* a preposition; and *adjective,* an adjective. Accepting nodes are labeled **. If the input string is *The little boy in the swimsuit kicked the red ball,* the above network would put it into the following phrases:

NP:	The little boy in the swimsuit
PP:	in the swimsuit
NP:	the swimsuit
Verb:	kicked
NP:	the red ball

Notice that any subnetwork of an RTN may call any other subnetwork, including itself; in the example above, for example, the prepositional phrase contains a noun phrase. Also notice that an RTN may be nondeterministic in nature; that is, there may be more than one possible arc to be followed at a given point in a parse. Parsing algorithms handle nondeterminism by *parallel processing* of the various alternatives or by trying one of them and then *backtracking* if it fails.

An *augmented transition network* (ATN) is an RTN that has been extended in three ways:

(1) A set of *registers* has been added; these can be used to store information, such as partially formed *derivation trees,* between jumps to different subnetworks.

(2) Arcs, aside from being labeled by word classes or syntactic constructs, can have arbitrary *tests* associated with them that must be satisfied before the arc is taken.

(3) Certain *actions* may be "attached" to an arc, to be executed whenever it is taken (usually to modify the data structure returned).

This addition of registers, tests, and actions to the RTN's extends their power to that of Turing machines. ATN's offer a degree of expressiveness and naturalness not found in Turing machine formalisms and are a useful tool to apply them to natural language analysis.

The operation of an ATN is similar to that of an RTN except that if an arc has a test, then the test is performed first and the arc is taken only if the test is successful. Also if an arc has actions associated with it, then these operations are performed after following the arc. In this way, by permitting the parsing to be guided by the parse history (through the tests on the registers) an by allowing for a re-arrangement of the structure of the sentence during the parse (through the actions on the registers), ATNs are capable of building deep structure descriptions of a sentence in an efficient manner.

A limitation to the ATN approach is that the heavy dependence on syntax restricts the parser's ability to handle ungrammatical (although meaningful) utterances.

Example of Augmented Transition Network Parsing

An *augmented transition network* (ATN) is an RTN that has been extended in three ways:

(1) A set of *registers* has been added; these can be used to store information, such as partially formed *derivation trees,* between jumps to different subnetworks.

(2) Arcs, aside from being labeled by word classes or syntactic constructs, can have arbitrary *tests* associated with them that must be satisfied before the arc is taken.

(3) Certain *actions* may be "attached" to an arc, to be executed whenever it is taken (usually to modify the data structure returned).

- Registers - can contain items, flags, or structures
  - e.g., register named SUBJ could contain the NP that is the candidate for the subject of the sentence
  - PUSH arc saves the registers on a stack
  - POP arc restores the registers from the stack

- Tests - look for agreement (usually by examining registers)
  - could prevent trying for a match or an arc if it would lead to a failure
  - e.g., CAT NP// arc following a verb test could examine the VERB register to see if the verb can take an NP in that position

- Actions - taken if an arc match is found
  - e.g., - set or modify registers
    - HOLD - puts a constituent on a holding list from which it is later removed and placed in its correct structural pattern
    - (SETR <register> *)
      - set register to * (= current form) where the current form is (1) current word; (2) current stem; or (3)
  - complete structure from a former level POP
    - BUILDQ - builds structures
    - + = register
    - # = evaluates the LISP form
    - = * (current form)
    - @ = append what is found as follows

if register DET contains (DET THE) and the current word on CAT arc = "books" and register ADJS contains ((ADJ OLD) (ADJ DUSTY) (ADJ RED)) then (BUILDQ (@ (NP +) + ((N *) (Nu #))) DET ADJS (GETF NUMBER)) would produce the structure (NP (DET THE) (ADJ OLD) (ADJ DUSTY) (ADJ RED) (N BOOK) (NU PL)) - (TO <state>)

A Detailed Example: The mayor would not have wanted to be elected to the position of dog-catcher.

First set the constant DCL to indicate that the sentence is declarative, and control moves to S/NP. In state S/NP the CAT V arc cannot be taken with the word "would" in the input, but the CAT MODAL arc succeeds. The MODAL register is set to the form ((MODAL WILL)), using the root form of "would". The tense is recorded in the TNS register in the form (TNS PAST).

In state S/AUX we pick up the negative. The test on this arc ensures that the arc has not been taken before, thus making double negatives unacceptable. The NEG register is being used both as a flag for the test and as a piece of structure to be later incorporated in the parse structure. The test for "do" in the verb register is necessary to undo the effect of do-support in sentences like "Didn't I meet you in Pango-Pango?" If we were not permitted to have such a conditional action it would be necessary to have two CAT NEG arcs, one with the test for "do" and an unconditional action removing it from the V register and one with a condition that "do" not be in the V register.

:

```
(S COMP
  (NP (PRO SOMEONE))
  (TNS PAST)
  (VP (V ELECT)
    (NP (ART THE)
      (N MAYOR)
      (NU SG))
     (PP (PREP TO)
       (NP (ART THE)
         (N POSITION)
         (NU SG)
           (PP (PREP OF)
             (NP (N DOG-CATCHER)
               (NU SG)))))))
```

This structure is returned to the configuration we were in at the time of the PUSH from state VP/TO, and the remaining actions on that arc place the structure in the OBJ register. Since there are no more words in the input string, the PUSH PP/ and WRD BY arcs cannot be taken, so the POP arc creates the following structure to return as the values of the successful parse:

```
(S DCL
  (NP (ART THE) (N MAYOR) (NU SG))
  (TNS PAST PERFECT)
  (AUX (MODAL WILL) NEG)
  (VP (V WANT) (S COMP)
         (NP (PRO SOMEONE))
         (TNS PAST)
         (VP (V ELECT)
           (NP (ART THE) (N MAYOR) (NU SG))
             (PP (PREP TO) (NP ... )))))))
```