

Bonaventure Undergraduate Robotics Laboratory
Technical Report 2
September, 1997

Parsing as Search 2:
An Easy to Understand ATN Interpreter
Robert M. Harlan, David M. Patrone, Scott L. Alexander

We use the representations generated by the ATN to build calls to a SHRDLU-like problem solving system. This system consists of a robot and a set of blocks. The robot is able to carry out commands and to answer questions about its actions and its world generally. In this paper we discuss how the ATN generates representations of commands given in English which can in turn be translated into calls to the problem solver.

Calls to the SHRDLU system, the target of the translation, take the form

(<primitive function>, <obj1>, <obj2>, ... <objn>)

where the primitive function is one supplied by the system (e.g., put-on, pick-up) and the arguments are symbolic atoms naming objects (e.g., b1, b2, p1). Thus, the call

(put-on 'b1 'b2)

is an instruction to put block b1 on block b2, which may necessitate intermediate actions or may be impossible.

To build calls to the problem solver, more than a purely syntactic representation characteristic of a pure ATN grammar is required. The representation generated by the ATN must identify the action to be carried out (the syntactic verb) and the object to be acted upon (the syntactic direct object).¹ Some commands, such as put-on, require a destination as well (the syntactic indirect object). Other commands, such as pick-up, do not, for the destination is implicit. In this problem domain, actions are more or less directly named by the syntactic verb. Objects, on the other hand, are described.

The representations generated by the ATN are input to a reference resolver, which identifies the objects described. The output of the reference resolver is either a call to the problem solver or a report of ambiguity if objects cannot be identified uniquely.

The ATN output for sentence (1), 'pick up the red block,' is given in figure 1. The STRUCTURE slot, action-obj, indicates that the object of the action must be named but the destination is implicit. The ACTION and OBJ slots have been filled. The ACTION slot contains the action named in the sentence, the OBJ slot a description of the object that fills that role. The description contains information regarding the type of reference (definite or indefinite) made and the type of attributes used to describe the object for subsequent use by the reference resolver.

The representation of sentence (2), 'put the green pyramid on the blue block,' is given in figure 2. The STRUCTURE slot indicates that both the OBJ and DEST slots must be filled for the action 'put-on'.

Prepositional phrases are interpreted as further object descriptions. The representation of (3), 'Put the blue block on the red block down,' is given in figure 3. Note that 'on' has been interpreted as preposition heading a prepositional phrase rather than as a particle as in (2).

The ATN must reject sentences such as (2*) 'Put the red block on' which violate the semantic constraints of actions. It also must reject sentences that violate particle agreement such as (1*) 'Pick on the red block'.

The representations built by the ATN contain sufficient

```
(S
  (STRUCTURE action-obj-dest)
  (ACTION put-on)
  (OBJ
    (NP
      (REF definite)
      (ADJS (green color))
      (HEAD pyramid)
    )
  )
  (DEST
    (NP
      (REF definite)
      (ADJS (blue color))
      (HEAD block)
    )
  )
)
```

Figure 2
Representation of 'Put the green pyramid on the blue block.'

```
(S
  (STRUCTURE action-obj)
  (ACTION put-down)
  (OBJ
    (NP
      (REF definite)
      (ADJS (red color))
      (HEAD block)
    )
    (PP
      (RELATION supported-by
        (NP
          (REF definite)
          (ADJS (blue color))
          (HEAD block)
        )
      )
    )
  )
  (DEST nil)
)
```

Figure 3
Representation of 'Put the red block on the blue block down'

information for the reference resolver to build a call to the problem solver. To examine how the representations are built, we first review the original RTN system which was modified to build them.

3. The RTN Interpreter and Grammar

The RTN interpreter uses a depth-first search engine to apply a RTN grammar to an input stream to compute whether or not the input conforms to the grammar. The search engine maintains a stack of parse states, each of which represents a viable state in the parse of the input. On each iteration the stack is popped, and an expansion function computes which state(s) can be reached from that state. The popped state is replaced by its successors (or by no states if the state is a dead end). The parse terminates successfully when a terminal transition network node has been reached and there are no remaining words to parse.

The algorithm for using the depth-first search engine to apply an RTN grammar to input is:

1. Initialize the stack of parse states with the start state for the grammar.
2. While the parse is not complete and there are still states to examine,
 - 2a. Generate all of the parse states that can be reached from the current state.
 - 2b. Replace the current state with new states

The LISP implementation of the algorithm is given in figure 4.

```
(defun interpret-rtn(stack)
  (cond
    ;;no more states to examine
    ((null stack) nil)
    ;;top of stack is terminal state
    ((goalp (car stack)) t)
    ;;add new paths to top of stack
    (t (interpret-rtn
        (append (expand(car stack))
              (cdr stack))))))
  )
```

Figure 4
LISP implementation of the RTN parse algorithm

To illustrate the algorithm, we use the RTN formalism to specify a grammar for accepting commands, discuss the information that must be contained in a parse state, and examine the expansion function that generates successor states.

The RTN grammar for accepting simple commands such as (1) 'Pick up the red block' is given in figure 5. There are three types of arcs that allow transitions between nodes in the transition nets:

- **CAT arcs**, which test the syntactic category of the next word in the words remaining to be parsed. If the syntactic category of the word matches that required by the arc, the word is removed from the stream and the current node is assigned the next node in the network. An example is the arc between VP and VP1, which is traversed when the next input word has the category verb.
- **PUSH arcs**, which require the successful traversal of another network. If the network named in the arc is successfully traversed, the current node is assigned the next node in the network. An example is the arc between S and S1, which is traversed when a VP

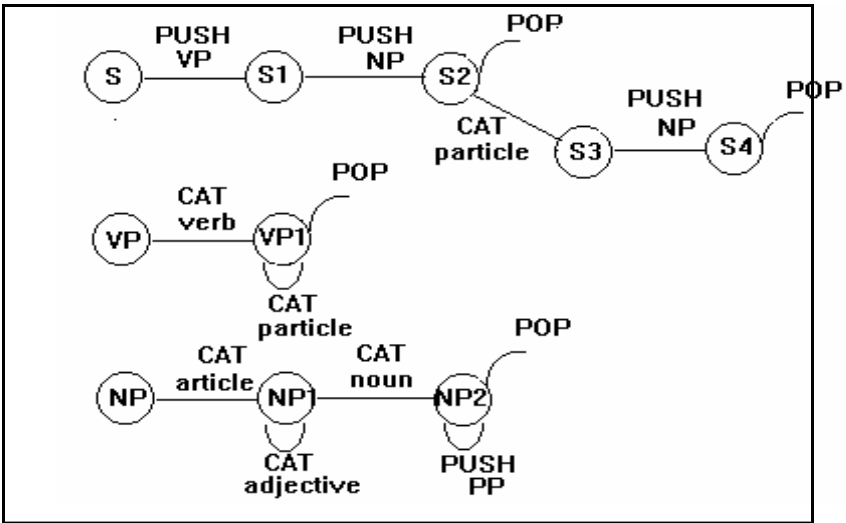


Figure 5
RTN grammar for accepting simple commands. The prepositional phrase net is not shown.

network is traversed.

- **POP arcs**, which signal the successful traversal of the network. If a network pops, the current node is assigned the next node in the parent network's network.

A sentence is accepted if the POP arc of the top-level network S is reached and there are no more remaining words.

The state of the parse is determined by a 3-tuple

(<current node>, <remaining words>, <return points>)

The current node is the node in the network being traversed. Remaining words are the words yet to be consumed in the parse process. Return points is a stack of return points generated from networks in which another network is pushed in the parse and to which the parse will return when the network is popped.

The expansion function must take the current state and apply the tests corresponding to the arcs emanating from the node of the current state. If a CAT arc is encountered and its condition met, a new state is generated where the current node is set to the next node in the network and the first word in the remaining words list is removed. If a PUSH arc is encountered, the new state is one with the next node in the current network pushed onto the stack of return points and the current node the first node in the network pushed. If a POP arc is encountered, the new state is one in which the current node is the top of the stack of return points and the return points stack is the stack after it is popped. A successful parse is indicated when the terminal node S2 is reached and there are no remaining words.

A trace of the stack during the parse of (1) is given in figure 6. The arcs used to generate successor states are also listed.

Stack of Parse States	Arc
1 ((S (pick up the red block)(nil)))	PUSH VP
2 ((VP (pick up the red block)(S1)))	CAT verb
3 ((VP1 (up the red block) (S1)) (S1 (up the red block) (nil)))	CAT particle POP
4 ((VP1 (the red block) (S1)) (S1 (up the red block) (nil)))	POP
5 ((S1 (the red block) (nil)) (S1 (up the red block) (nil)))	PUSH NP
6 ((NP (the red block) (S2)) (S1 (up the red block) (nil)))	CAT art
7 ((NP1 (red block) (S2)) (S1 (up the red block) (nil)))	CAT adj
8 ((NP1 (block) (S2)) (S1 (up the red block) (nil)))	CAT noun
9 ((NP2 nil S2) (S1 (up the red block) (nil)))	POP
10((S2 nil nil) (S1 (up the red block) (nil)))	POP

Figure 6
Parse of 'Pick up the red block'

4. From RTN to ATN

The RTN interpreter discussed above utilizes a strategy to control backtracking with which students were already familiar from the unit on search, enabling them to focus on how new parse states were generated by applying a grammar to a state. The conversion of the interpreter and grammar into ATNs required three modifications, which will be discussed in separate sections.

First, registers were associated with each network, and actions that created or updated them were added to the grammar. Feature tests were also added to the arcs connecting nodes in the networks. These are discussed in Section 5. Section 6 examines how the parse state was altered to include the stack of registers associated with that state in the parse. Section 7 discusses how information in lower level registers is returned to higher ones.

5. Registers and the ATN Grammar

The information required to build the desired representations is accumulated in registers that are associated with the various networks of the grammar. Registers contain slots that are filled as the networks are traversed. Registers associated with lower level networks in turn fill slots in higher level networks, resulting in a top-level representation of the sentence which indicates the action to be performed, describes all objects involved, and assigns

SREG	VREG	NREG
REGTYPE	REGTYPE	REGTYPE
ACTION	MAINV	REF
OBJ	PARTICLE	ADJS
DEST	STRUCTURE	HEAD
STRUCTURE	UPDATEFUN	UPDATEFUN
UPDATEFUN		

semantic roles to each object. The registers and their slots are listed in figure 7.

As discussed in section 3, the generation of a new parse state involves making a transition on one of three types of arcs. To convert the RTN grammar into an ATN one, we associate an action or actions to be performed when the arc is traversed. The three arcs have the following actions associated with them:

Figure 7
ATN Registers

- **CAT arcs** result in updates of the network register. The updates use **features** of the word's entry in the lexicon. **Feature tests** were added to category tests for traversing an arc.
- **PUSH arcs** result in a new register for the pushed network being generated.
- **POP arcs** result in information accumulated during a traverse of the network being returned to the register associated with the network from which it was pushed.

The top level of the ATN grammar used to parse imperatives having the action-object and action-object-destination structures is shown in figure 8. We illustrate the semantic actions taken by walking through the parse of 'pick up the red block.'

When the parse begins in the sentence network S, a SREG is generated. Two transitions are available from S, both of which push the traversal of the verb phrase register. Unique registers are generated for each of the transitions.² The VREG associated with the transition from S to S1 is assigned the structure action-obj, which indicates that the destination is implicit. The VREG associated with the transition S to S3 is assigned the structure action-obj-dest, which indicates that both the object and destination roles must be filled in order for the parse to be successful.

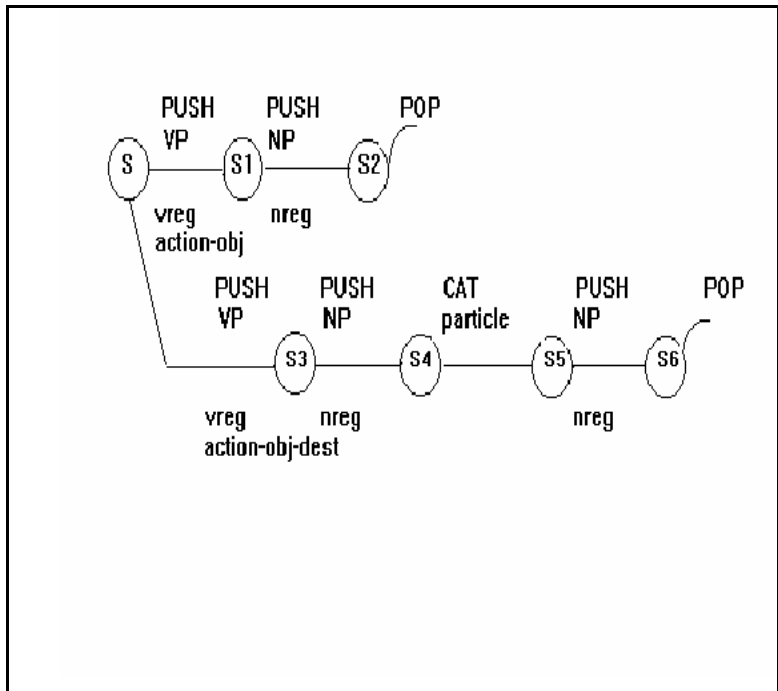


Figure 8
An ATN Grammar for simple imperatives. The semantic actions associated with each transition appear below the arcs.

The arc between VP and VP1 is a CAT arc that requires a verb. Two checks are made to determine if the transition can be made. First, the syntactic category of 'pick' is checked to determine whether it is a verb. Second, the sentence structure action-obj. Since the category VREG, is assigned the primitive problem sol

The same set of tests is made for the t checked and its feature list is checked to determine if it is compatible with the main verb. In this grammar, 'Pick up the red block' is allowed but not 'Pick on the red block'. The action associated with the arc fills the PARTICLE slot of VREG with 'up.'

The slots in the NP register are used to store the list of adjectives, the type of reference, and the noun used.

When the verb phrase and noun phrase networks are popped, data accumulated are passed back to the sentence register, SREG. The data in the noun registers are assigned semantic role slots the SREG. The action slot is filled by the action stored in VREG. This process is discussed in section 7 below.

A final check is made at the end of the parse to determine if the particle slot in the ACTION slot of the sentence register must be filled. The check is made at the end of the parse because the particle may appear before or after the noun phrase in a sentence with the structure action-object. For example, 'Put the red block on the blue block' could

be parsed as an action-object sentence, with `on' interpreted as a preposition. However, `to put' requires a particle in our grammar, and the lack of a particle causes a failure of this parse and a backtrack to a parse as a sentence having an action-object-destination structure.

6. The ATN Parse State

The parse state for a parse using an ATN grammar must include the registers for the networks being traversed. As a result the state consists of the 4-tuple

(`<current node>`, `<remaining words>`, `<return points>`, `<regstack>`)

with `regstack` being a stack of registers corresponding to the networks being traversed.

When a network is pushed, the register associated with the network is pushed onto the stack. Network registers are popped from the stack following network traversal. Figure 9 shows a trace of the parse of `pick up the red block' with the associated parse states.

7. Returning Data from Lower Level Registers

The action associated with a network's POP arc is the transfer of data collected in the traversal of the network to the register of the parent network from which the child network was pushed. The registers associated with both networks are in the `regstack`. The action, thus, consists of popping the register stack and removing the child network's register, popping the register stack again to get the parent's register, transferring data, and pushing the parent network's register back onto the register stack.

Each type or class of network register has a method for returning data to super-ordinate registers, registers of networks that can push the network. The class of the parent network's register is checked, and the appropriate actions taken.

The affect of the update can be illustrated by the representation of `Put the red block on the blue block down' in figure 3 above. The prepositional phrase `on the blue block' is interpreted to be a further description of the noun in the noun phrase from which it was pushed. The update function associated with prepositional phrase registers determines that the parent register is a noun phrase register and appends its data the noun phrase register's data.

8. Conclusion

The ATN interpreter and grammar presented have been used successfully in an introductory Artificial Intelligence course to illustrate how a natural language interface that makes calls to a problem solving system can be implemented. Students extended the grammar by adding additional commands that could be handled and succeeded in building calls to the problem solver.

The ATN met the authors' design goal of avoiding the ``black box'' syndrome, where the interpreter that applies a grammar is hidden from students' view. By treating parsing as a search problem and using a strategy for controlling search with which students were already familiar, students were able to understand and experiment with the process of building representations of sentences that captured their meaning.

Stack of ATN Parse States

```

1 ((S (pick up the red block)(nil)(SREG)))
2 ((VP (pick up the red block)(S1)(VREG SREG))
  (VP (pick up the red block)(S3)(VREG SREG)))
3 ((VP1 (up the red block) (S1)(VREG SREG))
  (VP (pick up the red block)(S3)(VREG SREG)))
4 ((VP1 (the red block) (S1)(VREG SREG))
  (VP (pick up the red block)(S3)(VREG SREG)))
5 ((S1 (the red block) (nil)(SREG))
  (VP (pick up the red block)(S3)(VREG SREG)))
6 ((NP (the red block) (S2)(NREG SREG))
  (VP (pick up the red block)(S3)(VREG SREG)))
7 ((NP1 (red block) (S2)(NREG SREG))
  (VP (pick up the red block)(S3)(VREG SREG)))
8 ((NP1 (block) (S2)(NREG SREG))
  (VP (pick up the red block)(S3)(VREG SREG)))
9 ((NP2 nil (S2)(NREG SREG))
  (VP (pick up the red block)(S3)(VREG SREG)))
10((S2 nil nil (SREG))
  (VP (pick up the red block)(S3)(VREG SREG)))

```

Figure 9

Trace of the parse of 'Pick up the red block'

References

- Allen, James. *Natural Language Understanding*. Reading, MA: Benjamin/Cummings Publishing, 1987.
- Allen, James. *Natural Language Understanding, Second Edition*. Redwood City, CA. Benjamin/Cummings Publishing, 1994.
- Harlan, Robert & Patrone, David. "Parsing as Search: An Easy to Understand RTN Parser," *SIGCSE Bulletin* 25,3. September, 1993. Also available as Bonaventure Undergraduate Robotics Laboratory Technical Report 1, <http://web.sbu.edu/cs/roboticsLab/>

¹ The restricted domain of a blocks world allows a simple mapping of syntactic roles to semantic roles. We map the syntactic verb and the direct and indirect objects to their semantic roles in the representations generated by the ATN. This makes our ATN grammar a semantic grammar.

² Separate registers are needed to avoid one register affecting another in the event that backtracking is required.