

SPARSE – *Student PARSing* Environment

Michael A. Covington
Artificial Intelligence Center
The University of Georgia
Athens, Georgia 30602-7415
<http://www.ai.uga.edu/~mc>

Version 0.1, 1999 March 24

1 Introduction

SPARSE is a menu-driven software environment that runs under SICStus Prolog and allows students to create and test natural-language parsers without having to learn Prolog.

2 Prerequisites

To use SPARSE, you need to know:

- How to get to a UNIX command prompt on the AI Suns;
- How to create, edit, and manage files under UNIX;
- Just a bit of Prolog syntax, which will be explained below;
- How to construct generative grammars using context-free phrase-structure rules, optionally augmented by features.

See the *Sun Local Guide* for more information about the AI Center's computers.

```

% A sample grammar to use with SPARSE

rule(s, [np, vp]).
rule(np, [d, n]).
rule(vp, [v, optional(np)]).

word(d, the).
word(n, dog).
word(n, cat).
word(v, chased).
word(v, barked).

```

Figure 1: A sample grammar for use with SPARSE.

You can use SPARSE from the Sun console or from a Telnet or (preferably) TeraTerm connection. You should make your window relatively large (30 to 40 lines or more), because the output of SPARSE is sometimes lengthy. Note that TeraTerm allows you to scroll back to look at material that has scrolled off the top of the screen.

The most vital UNIX commands are:

- `ls -al` to see what files are in your current directory;
- `pico filename` to edit a file (or create one if it does not exist);
- `rm filename` to remove a file.

3 Creating grammars

3.1 The basics

Figure 1 shows a simple grammar written in the format required by SPARSE. Note that:

- Lines that begin with `%` are comments, ignored by the computer.

- Phrase-structure rules are encoded as clauses beginning with `rule`. For example:

$$S \rightarrow NP VP \quad \text{rule}(s, [np, vp]).$$

- Lexical insertion rules (rules that insert particular words) are encoded as clauses beginning with `word`. For example:

$$N \rightarrow \text{dog} \quad \text{word}(n, \text{dog}).$$

- The parentheses and brackets must be used in exactly the manner shown, and each clause must end with a period.
- The `rule` clauses must be together, followed by (or preceded by) the `word` clauses; do not alternate the two kinds of clauses in your file.
- All category labels and all words begin with lower-case letters. (In Prolog, upper-case letters are variables.)

Use the Pico, Emacs, or XEmacs editor to create files with your grammars on them. These files are actually parts of a Prolog program and should have names ending in `.pl`.

3.2 More about grammar rules

SPARSE uses a top-down parser, but it includes a loop checker so that left-recursive rules are permitted. For example, the rule

$$\text{rule}(np, [np, conj, np]).$$

does not cause a problem (although it does waste some time).

In grammar rules, you can mark optional elements with `optional(...)`. For example:

$$VP \rightarrow V (NP) \quad \text{rule}(vp, [v, \text{optional}(np)]).$$

The parser will try such a rule both with and without the optional element.

3.3 Arguments (features)

Arguments on category labels are permitted. For example:

```
rule(np(Number), [d(Number), n(Number)]).
```

```
word(n(singular), dog).
```

```
word(n(plural), dogs).
```

```
word(d(singular), a).
```

```
word(d(plural), two).
```

```
word(d(_), the).
```

These rules generate *a dog*, *two dogs*, *the dog*, and *the dogs*, but not **a dogs* or **two dog*.

Here `Number` is a variable; that's why it begins with a capital letter. The values `singular` and `plural` are constants. The symbol `_` matches anything; it allows the rule for *the* to match both `d(singular)` and `d(plural)`.

Note that now that you've given an argument to `n`, every `n` in the entire grammar must have an argument position; `n(singular)` will never match a plain `n`. The same goes for `d`. You can put more than one argument on a category (e.g., gender, number, and case), so long as you do it consistently, using `_` for "don't care" positions.

For more about arguments, see Covington (1994), chapter 3. All the techniques illustrated there can be used in SPARSE. A future version of SPARSE will include all the facilities of GULP (see Covington 1994 ch. 5 and references there).

4 Using SPARSE

4.1 Starting SPARSE

To start SPARSE from the UNIX command prompt, use the command:

```
aisun3% sparse
```

(naturally, this will work on any of the AI Suns, not just aisun3).

4.2 If SPARSE crashes...

Because SPARSE was written in some haste, it may not be perfectly reliable. If you see the Prolog prompt (?-), do this:

```
?- halt.  
aisun3%
```

There – you’re back at the UNIX prompt.

If SPARSE goes into an endless loop, hit Ctrl-C. If SPARSE crashes with a message such as

```
Prolog interruption (h for help)?
```

type a and press Enter to get to the Prolog prompt.

4.3 Sample session

The best way to tell you how to use SPARSE is to show you a sample session, annotated with some comments. In what follows, characters typed by the user are in *typewriter italics*, and my comments are in *small italics*. Here goes...

```
aisun6% sparse
```

That’s the UNIX command to start SPARSE.

```
restoring /usr/local/bin/sparse...
```

```
/usr/local/bin/sparse restored in 30 msec 20208 bytes
```

```
SPARSE - Student PARSing Environment 0.1, March 2000
```

```
L - Load a file of grammar rules  
D - Display the grammar rules that are in memory  
S - Parse a sentence  
C - Parse a constituent of any type  
G - Generate a sentence  
T - Turn tracing on or off  
Q - Quit
```

```
Your choice: l
```

That's the SPARSE main menu.

I typed l and hit Enter, to tell SPARSE to load a file.

I could equally well have typed uppercase L.

File to load: *samplegrammar.pl*

I tell it what file to load. The Prolog compiler reads the file:

```
consulting /home/faculty/mc/sparse-dir/samplegrammar.pl...
```

```
consulted /home/faculty/mc/sparse-dir/samplegrammar.pl in  
module user, 10 msec 1056 bytes
```

```
Press Enter to continue...
```

At this point I press Enter, of course.

Here's the main menu again:

SPARSE - Student PARSing Environment 0.1, March 2000

```
L - Load a file of grammar rules  
D - Display the grammar rules that are in memory  
S - Parse a sentence  
C - Parse a constituent of any type  
G - Generate a sentence  
T - Turn tracing on or off  
Q - Quit
```

Your choice: *d*

*I want to make sure it has actually read my grammar correctly,
so I tell it to display the rules that it loaded. Here they are:*

```
rule(s, [np, vp]).  
rule(np, [d, n]).  
rule(vp, [v, optional(np)]).
```

```
word(d, the).  
word(n, dog).  
word(n, cat).  
word(v, chased).  
word(v, barked).
```

Press Enter to continue...

SPARSE - Student PARSing Environment 0.1, March 2000

L - Load a file of grammar rules
D - Display the grammar rules that are in memory
S - Parse a sentence
C - Parse a constituent of any type
G - Generate a sentence
T - Turn tracing on or off
Q - Quit

Your choice: s

Let's tell it to parse a sentence.

Sentence to be parsed:

|: *The dog chased the cat.*

It doesn't matter whether I type punctuation or not.

All the words are converted to lower case before the parser sees them.

Parsing [the,dog,chased,the,cat] as s...

(All the words are in the lexicon.)

```
s
  np
    d - the
    n - dog
  vp
    v - chased
    np
      d - the
      n - cat
```

This is the tree structure, displayed as an indented outline.

Look for another alternative? (y/n) *n*

(If the sentence were syntactically ambiguous, there would be more than one way to parse it. We can tell the parser to backtrack and look for an alternative. In this case I chose not to.)

Press Enter to continue...

SPARSE - Student PARSing Environment 0.1, March 2000

- L - Load a file of grammar rules
- D - Display the grammar rules that are in memory
- S - Parse a sentence
- C - Parse a constituent of any type
- G - Generate a sentence
- T - Turn tracing on or off
- Q - Quit

Your choice: *s*

Sentence to be parsed:

|: *The dog chased the beaver.*

Parsing [the,dog,chased,the,beaver] as s...

Word "beaver" is not in the lexicon.

One of the most common reasons for a parse to fail is that the sentence contains a word that is not in the lexicon.

Accordingly, SPARSE checks that all the words are in the lexicon before doing anything else to a sentence you type in.

Parse failed.

Press Enter to continue...

SPARSE - Student PARSing Environment 0.1, March 2000

- L - Load a file of grammar rules
- D - Display the grammar rules that are in memory

- S - Parse a sentence
- C - Parse a constituent of any type
- G - Generate a sentence
- T - Turn tracing on or off
- Q - Quit

Your choice: s

Sentence to be parsed:

|: *The dog chased the.*

Here's one where the parse fails simply because the sentence isn't generated by the grammar.

Parsing [the,dog,chased,the] as s...
(All the words are in the lexicon.)

Parse failed.

Press Enter to continue...

SPARSE - Student PARSing Environment 0.1, March 2000

- L - Load a file of grammar rules
- D - Display the grammar rules that are in memory
- S - Parse a sentence
- C - Parse a constituent of any type
- G - Generate a sentence
- T - Turn tracing on or off
- Q - Quit

Your choice: c

Let's parse something other than an S.

Kind of constituent to parse: np

Phrase to be parsed:

|: *the cat*

Parsing [the,cat] as np...
(All the words are in the lexicon.)

np
 d - the
 n - cat

Sure enough, "the cat" is an NP.

Look for another alternative? (y/n) *n*
Press Enter to continue...

SPARSE - Student PARSing Environment 0.1, March 2000

L - Load a file of grammar rules
D - Display the grammar rules that are in memory
S - Parse a sentence
C - Parse a constituent of any type
G - Generate a sentence
T - Turn tracing on or off
Q - Quit

Your choice: *g*

We'll ask the parser to generate a sentence.

This is a good way to make sure the grammar generates something, even if it doesn't generate what you want.

s
 np
 d - the
 n - dog
 vp
 v - chased
 np
 d - the
 n - dog

the dog chased the dog

Look for another alternative? (y/n) *y*

s

np

d - the

n - dog

vp

v - chased

np

d - the

n - cat

the dog chased the cat

Look for another alternative? (y/n) *y*

s

np

d - the

n - dog

vp

v - chased

the dog chased

Look for another alternative? (y/n) *n*

(We'd better stop sometime!)

Press Enter to continue...

SPARSE - Student PARSing Environment 0.1, March 2000

L - Load a file of grammar rules

D - Display the grammar rules that are in memory

S - Parse a sentence

C - Parse a constituent of any type
G - Generate a sentence
T - Turn tracing on or off
Q - Quit

Your choice: *t*

Tracing has been turned on

Tracing allows you to see the steps the parser goes through when parsing a sentence. Tracing can be very helpful if your parser isn't working and you can't tell why.

Press Enter to continue...

SPARSE - Student PARSing Environment 0.1, March 2000

L - Load a file of grammar rules
D - Display the grammar rules that are in memory
S - Parse a sentence
C - Parse a constituent of any type
G - Generate a sentence
T - Turn tracing on or off
Q - Quit

Your choice: *s*

Sentence to be parsed:

|: *the dog chased the cat*

Parsing [the,dog,chased,the,cat] as s...

(All the words are in the lexicon.)

<0> Trying s as [np,vp]

<1> Trying np as [d,n]

<2> Accepted the as d

<2> Accepted dog as n

<1> Accepted np as [[d,[the]], [n,[dog]]]

<1> Trying vp as [v,optional(np)]

<2> Accepted chased as v

```
<2> Trying np as [d,n]
  <3> Accepted the as d
  <3> Accepted cat as n
  <2> Accepted np as [[d,[the]], [n,[cat]]]
<1> Accepted vp as [[v,[chased]], [np,[d,[the]], [n,[cat]]]]
<0> Accepted s as
[[np,[d,[the]], [n,[dog]]], [vp,[v,[chased]], [np,[d,[the]], [n,[cat]]]]]
```

s

np

d - the

n - dog

vp

v - chased

np

d - the

n - cat

Look for another alternative? (y/n) *n*

Press Enter to continue...

SPARSE - Student PARSing Environment 0.1, March 2000

```
L - Load a file of grammar rules
D - Display the grammar rules that are in memory
S - Parse a sentence
C - Parse a constituent of any type
G - Generate a sentence
T - Turn tracing on or off
Q - Quit
```

Your choice: *t*

Tracing has been turned off

(That's how you turn it off again.)

Press Enter to continue...

SPARSE - Student PARSing Environment 0.1, March 2000

L - Load a file of grammar rules
D - Display the grammar rules that are in memory
S - Parse a sentence
C - Parse a constituent of any type
G - Generate a sentence
T - Turn tracing on or off
Q - Quit

Your choice: *q*

aisun6%

There! Back home to UNIX at last!

4.4 Getting a transcript of your session

The UNIX `script` command will record, to a text file, a copy of everything sent to your terminal. For example:

```
aisun4% script xxxxx.txt
```

```
aisun4% sparse
```

```
.
```

```
...a long and intricate interaction with SPARSE occurs here...
```

```
.
```

```
aisun4% exit
```

```
Script written to xxxxx.txt
```

You can then edit the script file and extract the information of interest to you.

References

Covington, Michael A. (1994) *Natural Language Processing for Prolog Programmers*. Englewood Cliffs, N.J.: Prentice-Hall.