Compilers — Homework 7
Parsing

Due Fri 23 Nov 2012, 20:00

For this assignment, you can work in groups of two or three people. Groups containing both Bachelor and Master's students are not allowed, except with the instructors permission. Every member of the group is responsible for understanding the work.

1 Parsing

Perl allows a statement to be conditionally evaluated by specifying a guard after the statement. For example, consider the following implementation of the factorial function:

```
sub fact {
  my $n = shift;
  return 1 if $n == 0;
  return &fact($n-1) * $n;
}
```

The `fact` function returns 1 if if the variable `$n` is zero; otherwise, it falls through to the second next statement, which calls `fact` recursively. The statement:

```
  return 1 if $n == 0;
```

is just another way to write:

```
  if ($n == 0) { return 1; }
```

Below is a simplified grammar for Perl statements. Expressions are represented as the terminal `exp`.

$$S ::= S\ \texttt{if}\ \texttt{exp}$$
$$S ::= \texttt{if}\ \texttt{exp}\ S$$
$$S ::= \texttt{if}\ \texttt{exp}\ S\ \texttt{else}\ S$$
$$S ::= \texttt{return}\ \texttt{exp}$$

(a) Why is this grammar *not* LL(1)?

(b) Construct the LR(0) state machine for this grammar.

(c) What states, if any, have a shift/reduce or reduce/reduce conflict?

(d) Step through the LR parsing algorithm using the input string:

```
    if exp return exp if exp else return exp
```

Show clearly what happens to the stack and to the automaton state as the string is parsed. If the parser reaches a state with a conflicting action (shift or reduce), show what happens for all possible choices. If the parser reaches a state where no action is possible, indicate so. For each successful parse, show the parse tree that results. Based on the results, how should any conflicts have been resolved? That is, if the parser reached a state with a shift/reduce conflict, should it have shifted or reduced? If the parser reached a state with a reduce/reduce conflict, with which rule should it have reduced?

2 Parsing JSON

JSON is a subset of JavaScript used for exchanging data on the Internet. A description of the JSON syntax can be found at `http://www.json.org/fatfree.html`.

(a) Based on the description of the syntax, what are the tokens in the language? Write regular expressions for these tokens.

(b) Write a context-free grammar for JSON using the tokens you defined above.

(c) Using PLY, implement a JSON scanner and parser. For full credit, your parser should have no shift/reduce or reduce/reduce conflicts. If it does, you need to change the grammar to correct this.

Create a Python script named `json.py`. This script should read JSON code from the standard input, parse the JSON code, and return a Python value. JSON objects should be translated into Python dictionaries. JSON arrays should be translated into Python lists. If the input is not legal JSON, your script should report an error and exit.

The script should then output the JSON object by calling `json.dumps` from Python's `json` module, and printing the result. You'll need Python 2.6 or later to use this module. For a given input, your parser should return a value equivalent to the value returned by `json.loads`. Your script should behave identically to `python -mjson.tool`.

Here is an example interaction with the script.

```
$ cat test.json
[1,2,3,{"a": 4, "b": 5}]
$ python -mjson.tool < test.json
[D
    1,
    2,
    3,
    {
        "a": 4,
        "b": 5
    }
]
$ python json.py < test.json
[D
    1,
    2,
    3,
    {
        "a": 4,
        "b": 5
    }
]
```