Department of Information and Computing Sciences
Utrecht University

Languages and Compilers
INFOB3TC – Deeltoets 1

Johan Jeuring

Tuesday, 16 December 2008, 15 - 17, BBL 513

This is the exam for the course on Languages and Compilers.

- Write your **name** and **student number** on all submitted work. Also include the total number of separate sheets of paper.
- At each question we give the maximum score. The total amount of points you can get is 90.
- Results from questions may be used anywhere, even if you did not answer that particular question.
- Write readable, preferably do not write with pencil or red ink pens. Write concise and clear answers. Unreadable (or difficult to read) texts may be ignored when grading.
- You may answer the questions in Dutch or English.
- It is not allowed to consult any materials during the exam. Here are the types of some basic parser combinators:

```
-- Elementary parsers
symbol   ::  Eq s  =>  s -> Parser s s
satisfy  ::  (s -> Bool) -> Parser s s
token    ::  Eq s => [s] -> Parser s [s]
sptoken  ::  Eq s => String -> Parser Char String
succeed  ::  a -> Parser s a

-- Parser combinators
(<|>)  ::  Parser s a        -> Parser s a -> Parser s a
(<*>)  ::  Parser s (a -> b) -> Parser s a -> Parser s b
(<$>)  ::  (a -> b)          -> Parser s a -> Parser s b
```

```
-- EBNF parser combinators
option  ::  Parser s a -> a -> Parser s a
many    ::  Parser s a  -> Parser s [a]
many1   ::  Parser s a -> Parser s [a]

-- Applications of EBNF
listOf      ::  Parser s a -> Parser s b -> Parser s [a]
natural     ::  Parser Char Int
integer     ::  Parser Char Int
identifier  ::  Parser Char String
```

- Good luck!

Concepts

1 (5 points)**.** Explain in a few sentences the relation between the concepts *language*, *sentence*, and *grammar*.

2 (5 points)**.** Explain in a few sentences the relation between the concepts *derivation*, *parsing*, and *ambiguity*.

3 (10 points)**.** Describe a non-trivial context-free language that has not already been defined in the lecture notes, the labs, or the lectures. You may take any of the languages introduced in previous courses, such as SQL, propositional logic, predicate logic, set theory, recurrence relations, etc.

Define a grammar for your language, and analyse it. Is it left-recursive, ambiguous, ...? Can the language also be defined by means of a *regular* grammar?

Parser combinators

The following info is copied from the first lab exercise:

The concrete syntax of moves in so-called *Standard Algebraic Notation* (SAN) is given by the following grammar:

move	::= *piece disambiguation capture square promotion check*
	| `O-O-O` | `O-O`
piece	::= `N` | `B` | `R` | `Q` | `K` | ε
disambiguation	::= *file* | *rank* | *square* | ε
capture	::= `x` | ε
square	::= *file rank*
promotion	::= `=` *piece* | ε
check	::= `+` | `#` | ε
file	::= `a` | `b` | `c` | `d` | `e` | `f` | `g` | `h`
rank	::= `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8`

2

Terminals are written in typewriter font, nonterminals in italics. A regular move is a sequence indicating a piece to be moved, possible disambiguation between several pieces of the same category by giving as much information about the initial square of the move as necessary, an indication if the move is a capture, the target square of the move, information about a possible pawn promotion, and an annotation if the move results in check or checkmate.

There are two special moves, called *castling*, where the king and one of the rooks move at the same time. Queenside castling is indicated using `O-O-O` (composed of the letter `O`, not the number `0`, and dashes), kingside castling using `O-O`.

The piece that is moved is indicated using a single uppercase letter. Here, `N` stands for *knight*, `B` for *bishop*, `R` for *rook*, `Q` for *queen*, and `K` for *king*. If a *pawn* is moved, no letter is used.

In most cases, the rules of the game and/or the situation of the game board imply that giving the target square of the move uniquely determines the piece that is moved. If that is not the case, some information of the initial square of the move can be provided, by either giving the *file*, or the *rank*, or the complete *square*. The file is the 'column' in which a piece is located, indicated from left to right by a letter from `a` to `h`. The rank is the 'line' on which a piece is located, indicated from bottom to top by a number from `1` to `8`.

If a move captures an opponent's piece, this is indicated using an `x` between the possible disambiguation information and the target square.

If a pawn succeeds to move all across the board to the base rank of the opponent, it can be 'promoted' to another piece. This is indicated by the symbol = and the letter for the appropriate piece.

If a move results in check or checkmate, this is indicated by appending a `+` or a `#` symbol, respectively, to the move description.

No whitespace is allowed anywhere in a move!

4 (10 points)**.** (Same as in the lab exercise.) Define Haskell datatypes to describe the abstract syntax of a move. Call the datatype for a move `Move`. Note that you will probably have to declare many datatypes, but you may also decide to represent certain syntactic categories using type synonyms if you prefer.

5 (30 points)**.** (Different from the lab exercise.) Define a parser

```
parseMove :: Parser Char Move
```

that parses a single SAN move, using *parser combinators*. (So **not** using a parser generator as you did in the lab exercise.)

6 (10 points)**.** Actually, the language of moves is a regular language. Show that this is the case, by defining a regular grammar, or a DFA, or an NDFA, or .... To simplify an answer to this exercise, you may assume (but don't have to) that there is a just single piece instead of five, a single check-symbol, a single file, and a single rank.

Folds

A resistor is a basic resistor with a fixed (floating point) resistance, or consists of two resistors in sequence or in parallel. A datatype for resistors looks as follows:

```
data Resist = Resist:|:Resist
            | Resist:*:Resist
            | BasicR Float
```

7 (12 points)**.** Define the datatype `ResistAlgebra` and the fold on the datatype `Resist`. Don't forget to give the type of the fold-function.

8 (8 points)**.** Use the fold on `Resist` to calculate the resistance of a value of the datatype `Resist`. Remember that the resistance of two resistors in sequence is the sum of the respective resistances, and for resistors in parallel we have: $\frac{1}{r} = \frac{1}{r_1} + \frac{1}{r_2}$.

Meta question

9 (no points)**.** How many points out of 90 do you think you get for this exam?