Parser

Théa Johnson
12307926

April 3, 2016

1 Defining the Types in OCaml

Before the parser could be created certain types had to be defined such that it could parse the text syntax of the behaviours and the constraints to these types. These types are regions, t, sesType, stackFrame, b and con.

The base types all either consist of strings or have no associated type and more complex types are built up from these.

In the cases where choices are offered these types consist of lists made up of tuples of the types.

2 Designing the Parser

The parser in this project is written using ocamllex (for the lexer) and menhir (for the parser). The first stage in building this was to design an text based version of the behaviours and constraints. This involved replacing all non standard keyboard characters with ones that can be typed easily. The lexer was then written to transform this syntax into a stream of tokens. This was then parsed by the parser file and, initially, output to the console to see that it was in fact parsing correctly.

2.1 Technologies Used

2.1.1 Menhir

2.1.2 Ocamllex

2.2 Text Syntax

The syntax given in the paper can be seen in (put figure in here). The text based syntax substitutions can be seen in the table given below.

Mathmatical Syntax	Text Based Syntax
α	[T][A-Z a-z 1-9]+
β	[B][A-Z a-z 1-9]+
ψ	[S][A-Z a-z 1-9]+
ρ	[R][A-Z a-z 1-9]+
TxT	pair (T;T)
$T \xrightarrow{\beta} T$	funct T -> T -[B][A-Z a-z 1-9]+
Ses^{ρ}	ses [R][A-Z a-z 1-9]+
cont...	

Behaviours and constraints of this syntax can then be parsed.

2.3 Parser

Ther parser is written in menhir. First tokens are declared for the syntax we will need to use and the Behaviour.ml file is opened. The start point of the grammar is the rule parse_behaviour. This consists of a behaviour followed by a constraint. The BNF forms of the syntax needed for each of the different types can be seen below.

Once a token sequence is encountered in the correct order it is mapped to the correct type.

2.4 Lexer

The lexer is written using ocamllex. This allows us to define multiple lexing rules in the same file. This has been taken advantage of to lex labels seperately to the rest of the file.

3 Grammars

3.1 Types

⟨*bType*⟩ ::= 'unit'
| ' bool '
| 'int'
| 'pair' '(' ⟨*bType*⟩ ';' ⟨*bType*⟩ ')'
| 'funct' ⟨*bType*⟩ '->' ⟨*bType*⟩ '-' ⟨*behaviourVariable*⟩
| 'ses' ⟨*region*⟩
| ⟨*TVar*⟩

3.2 Session Types

⟨*sessionType*⟩ ::= 'end'
| '!' ⟨*bType*⟩ ⟨*sessionType*⟩
| '?' ⟨*bType*⟩ ⟨*sessionType*⟩
| '!' ⟨*sessionType*⟩ ⟨*sessionType*⟩
| '?' ⟨*sessionType*⟩ ⟨*sessionType*⟩
| '(+)' [⟨*sesOpL*⟩] '('⟨*lable*⟩ ';' ⟨*sessionType*⟩')'
| '+' [⟨*sesOpL*⟩] [⟨*sesOpL*⟩]
| ⟨*SVar*⟩

⟨*sesOpL*⟩ ::= seperated_list(',',⟨*ses_opt_field*⟩)

⟨*ses_opt_field*⟩ := (⟨*lable*⟩ ';' ⟨*sessionType*⟩)

3.3 Behaviour

⟨*behaviour*⟩ ::= ⟨*behaviourVariable*⟩
| 'tau'
| ⟨*behaviour*⟩ ';' ⟨*behaviour*⟩
| 'chc' (⟨*behaviour*⟩,⟨*behaviour*⟩)
| 'rec' ⟨*behaviourVariable*⟩ (⟨*behaviour*⟩)
| 'spn' (⟨*behaviour*⟩)
| 'psh' (⟨*lable*⟩,⟨*sessionType*⟩)
| ⟨*region*⟩ '!' ⟨*behaviourVariable*⟩
| ⟨*region*⟩ '?' ⟨*behaviourVariable*⟩
| ⟨*region*⟩ '!' ⟨*region*⟩
| ⟨*region*⟩ '?' ⟨*lable*⟩
| ⟨*region*⟩ '!' ⟨*lable*⟩
| ⟨*region*⟩ '?' option[⟨*oplist*⟩]

⟨*oplist*⟩ ::= seperated_list(',',opt_feild)

⟨*opt_feild*⟩ ::= (⟨*lable*⟩ ';' ⟨*behavioiur*⟩)

3.4 Region Variables

⟨*regionVar*⟩ ::= ⟨*lable*⟩
| ⟨*region*⟩

3.5 Constraints

⟨*constr*⟩ ::= ⟨*bType*⟩ <⟨*bType*⟩
| ⟨*behaviour*⟩ <⟨*behaviour*⟩
| ⟨*region*⟩ ~⟨*regionVar*⟩
| ⟨*channel*⟩ ~⟨*sessionType*⟩
| ⟨*channelEnd*⟩ ~⟨*sessionType*⟩
| ⟨*contr*⟩ ',' ⟨*constr*⟩
| ϵ