Group Project - FASTO

Anders Brandhof '190493' SGL135
Andreas Jørgensen '240594' SRV415
Casper Iversen '090691' JVP497
Søren Jensen '270792' PWS412

19. december 2015

Indhold

1 Opgaven

Vi skal i denne gruppe opgave færdiggøre en optimeret compiler til FASTO. FASTO er et simpelt første-ordens funktionelt sprog.

2 Design

Compileren er blevet lavet i sml. For at færdiggøre compileren har vi ændret i de følgende filer:

- Lexer.lex
- Parser.grm
- Interpreter.sml
- TypeChecker.sml
- CopyConstPropFold.sml
- CodeGen.sml

Vi vil i afsnittene gennemgå de vigtigste ændringer vi har lavet.

2.1 Lexer

I filen Lexer.lex har vi fået givet tokens og keywords. Vi har her skulle tilføje flere tokens og keywords til FASTO. Det har her været map, reduce, iota, true, false mm..
Vi har tilføjet keywords i funktionen "keyword (s, pos), dette har vi gjort på følgende måde:

```
fun keyword (s, pos) =
case s of
     "map"    => Parser.MAP pos
     "reduce" => Parser.REDUCE pos
     "true"   => Parser.BOOLVAL (true,pos)
     "false"  => Parser.BOOLVAL (false,pos)
```

Vi har kun beskrevet et par af de keywords vi har lavet, men det viser hvordan vi har gjort det.
Vi har også tilføjet tokens til Lexer.lex, dette har vi gjort på følgende måde:

```
  | '+'    { Parser.PLUS (getPos lexbuf) }
  | '-'    { Parser.MINUS (getPos lexbuf) }
  | "&&"   { Parser.AND (getPos lexbuf) }
  | "not"  { Parser.NOT (getPos lexbuf) }
  | _      { lexerError lexbuf "Illegal symbol in input" };
```

Vi har kun vist et udsnit af de tokens vi har lavet. Som det kan ses på eksemplet, bruger vi både ' ' og . Forskellen på dette er, at ' ' er en char og er en string. Grunden til vi bliver nød til at definere det som en string, er fordi det er sammensat af mere end en char, og derfor skal være en string for at blive genkendt.

2.2 Parser

I filen Parser.grm ser vi på hvordan vi bruger de tokens vi har lavet i Lexer.lex. Her er et lille udsnit af, hvordan vi har gjort dette:

```
%token <int*(int*int)> NUM
%token <bool*(int*int)> BOOLVAL
%token <char*(int*int)> CHARLIT
%token <(int*int)> AND OR
%token <(int*int)> NOT
```

Her ligger man mærke til, at vi har skrevet (int*int). Dette er for, at vi ved hvorhenne den pågældende token er blevet brugt. Dette gør det muligt for os at kunne returnere en relevant fejlbesked med lokationen i dokumentet.

Vi har i Parser.grm også skulle definere precedens af f.eks. +, -, · og /. Dette har vi gjort ud fra operatornes præcedens i matematikken.

Vi har også skulle tage stilling til, hvordan f.eks. DIVIDE skal bruge dens Exp, parameter etc.. Nedenunder har vi vist et udsnit af, hvordan vi har gjort dette:

```
| BOOLVAL          { Constant (BoolVal (#1 $1), #2 $1) }
| Exp TIMES  Exp { Times ($1, $3, $2) }
| Exp DIVIDE Exp { Divide ($1, $3, $2) }
```

2.3 TypeChecker

2.4 Interpreter

2.5 CopyConstPropFold

2.6 CodeGen

3 Tests