G-opgave
Oversættere
Vinter 2011/2012

Naja Wulff Mottelson (vsj465)
Søren Egede Pilgård (vpb984)
Mark Schor (vcl329)

21. december 2011

Indhold

1 Indledning

På nuværende hvor langt er vi nået med opgaven? Hvad virker, hvad virker ikke?

2 Lexer og parser

Umiddelbart er der ikke meget at bemærke ved vores arbejde med lexer og parser. Vi har indledningsvist tilføjet de manglende nøgleord (while og char) i lexeren, samt fjernet nøgleordet then, eftersom dette ikke er til stede i grammatikken for 100. Vi har herefter oprettet regulære udtryk i rule Token til at matche tilføjelserne (som ligeledes tæller ==- operatren, return-sætninger, krølleparenteser samt firkantede parenteser).

I arbejdet med parseren har vi genemgået den udleverede grammatik og indføjet de manglende termer som tokens og types i Parser.grm (hvor vi i samme omgang tilføjede præcedens og associativitet - her var hoveddelen af det vi gjorde at tilføje venstreassociativitet til ==-operatoren).

Ved kørsel af compile.sh modtager vi ingen fejlbeskeder eller notifikationer om shift/reduce konflikter. Eftersom vi antager at parsergeneratoren er korrekt implementeret, konkluderer vi derfor at den grammatik vi parser er entydig.

3 Typetjekker

Hoveddelen af det arbejde vi har lagt i typetjekkeren har været i form af udvidelser i funktionen Type.checkExp og Type.checkStat. Udover dette har vi dog indledningsvist tilføjet 100s indbyggede funktioner (balloc, walloc, getstring og putstring) i Type.checkProg, samt tilføjet typerne for 100s abstrakte syntaks (Char, CharRef, IntRef).

3.1 Tjek af udtryk

I Type.checkExp har vi tilføjet mønstre for de manglende 100-typer (CharConst, StringConst, Equal), samt tilføjet overlæsning at plus- og minusoperatorerne så at de er i stand til at tage både heltal, tegreferencer og heltalsreferencer som operander.

3.2 Tjek af sætninger

Idet 100 indeholder en del kontrolstrukturer som kan skabe usikkerhed om hvorvidt en funktion vil returnere, er dette ikke som sådan et trivielt problem. Under typetjekket kan man f. eks. møde en lignende funktion:

```
int foo(x)
    if x
        return bar
```

Funktionen foo er syntaktisk korrekt, men indeholder en typefejl eftersom det ikke er sikkert at return-sætningen vil blive nået. Der findes adskillige måder at tjekke for denne slags typefejl - en metode kunne f.eks. være at undersøge en funktions sidste linje og kontrollere at denne altid vil returnere. Denne løsning vil dog træffe en problematik ifbm. if-else-konstruktionen, da denne kan have return-sætninger i begge sine tilfælde. Således vil enhver funktion indeholdende en sådan if-else altid være sikker på at returnere, ligegyldigt hvad der står i dens sidste linje kode.

Vi implementerer tjekket ved at tilføje et yderligere output i Type.checkStat - en sandhedsværdi som angiver hvorvidt den matchede sætning indeholder en return-sætning. I tilfældene if og while returneres falsk, eftersom begge konstruktioner vil kræve en return-sætning i en følgende blok. I tilfældet if-else kontrollerer vi at begge tilfælde returnerer og sender denne sandhedsværdi videre. checkStats output kontrolleres senere i Type.checkFunDec, hvor en fejl kastes i tilfælde af manglende returns.

Som nævnt skal det dog også kontrolleres at return-sætningens og funktionens type er kongruente, til hvilket vi har overvejet forskellige løsninger. Den mest funktionelle løsning forekommer at være at lade Type.checkFunDec returnere funktionens returtype. Type.checkStat vil herfter kunne udføre sit vanlige tjek af tilstedeværelsen af return-sætninger, og herefter returnere enten NONE i tilfælde af manglende returns eller SOME af sætningens returtype, samt sætningens position. Sammenligningen af funktionens og return-sætningens typer (samt tjekket for manglende returns) vil herefter kunne foretages i Type.checkFunDec. Denne implementering bliver dog problematisk så snart forgreninger i koden medfører flere return-sætninger i samme blok. Type.checkStat ville i dette tilfælde være nødt til at returnere en liste indeholdende tupler af positioner og returtyper, som ville skulle gennemsøges i Type.checkFunDec for at udføre typesammenligningen.

Den måde vi har valgt at implementere tjekket på er (ligesom i algoritmen nævnt ovenfor) ved at sende en funktions returtype samt position med som output fra Type.checkFunDec, hvorefter kontrollen af typekongruensen foretages i Type.checkStat.

3.3 Tjek af venstreværdier

Vi har udvidet Type.checkLval med tilfældene S100.Deref samt S100.Lookup, så at det bliver muligt at typetjekke forsøg på at indeksere arrays (eller, snarere, repræsentationen af arrays i form af en reference). Essentielt kan to fejl opstå i denne situation: det indekserede array kan være udefineret, eller indekset kan være af anden type end heltal. I vores implementering har vi valgt at prioritere den første type fejl højest, så at en fejlbesked vedr. eksistensen af referencen vil blive kastet førend en om indeks.

4 Kodegenerering og oversætter

Vi har indsat de indbyggede funktioner (udover putint og getint) i Compiler.compile.
Eftersom det er muligt at have deklarationer i starten af blokke, er det nødvendigt at kunne udvide symboltabellen for at implementere dem i oversætteren. Vi har derfor introduceret funktionerne compileDecs og extend, som begge er hentet fra Type.sml - extend er dog en let modificeret version af Type.extend som også opretter en ny lokation.

4.1 Håndtering af referencer

Vi har valgt at udvide datatypen Location
Vi har skrevet funktionen ignoreChars, fordi vi abstraherer os fra at skulle håndtere char-typen i typetjekkeren.

5 Afprøvning

6 Opsummering