Programmierung 2 (SS 2021)
Universität des Saarlandes
Fakultät MI
Compiler Design Lab

Musterlösung 13
Typüberprüfung & Codeerzeugung

Prof. Dr. Sebastian Hack
Julian Rosemann, B. Sc.
Thorsten KlöSSner, M. Sc.
Julia Wichlacz, M. Sc.
Maximilian Fickert, M. Sc.

Im Vorlesungskalender finden Sie Informationen über die Kapitel des Skripts, die parallel zur Vorlesung bearbeitet werden sollen bzw. dort besprochen werden. Die Übungsaufgaben dienen der Vertiefung des Wissens, das in der Vorlesung vermittelt wird und als Vorbereitung auf Minitests und Klausur.
Weitere Aufgaben zu den Themen finden Sie jeweils am Ende der Skriptkapitel.
Die Schwierigkeitsgrade sind durch Steine des 2048-Spiels gekennzeichnet, von 512 „leicht“ bis 2048 „schwer“. 4096 steht für Knobelaufgaben.

Aufgabe 13.0: Typüberprüfung

1. Betrachten sie das folgende C0pb-Programm und geben Sie die veränderte Typumgebung Γ' zu $\Gamma = \{\}$ nach der Ausführung der initialen Blockregel an.

```
{
    char c;
    char *pc;
    char **ppc;
    int i;
    int *pi;
    i = 42;
    ...
}
```

2. Werten sie die folgenden Ausdrücke unter der Typumgebung aus 1 aus:

   (a) `i - 42`

   (b) `*pc + i - 1337`

   (c) `**ppc - **&pc != 7198`

3. Werten sie erneut einen Blick auf 2a. Welchen Typ müsste *i* haben damit sie keine Typ-Inferenzregeln auf den Ausdruck anwenden können?

Lösung

1. $\{c \mapsto \texttt{char}, pc \mapsto \texttt{char*}, ppc \mapsto \texttt{char**}, i \mapsto \texttt{int}, pi \mapsto \texttt{int*}\}$

2. (a)

$$\text{BinArith}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{i} = \textbf{int}}{\Gamma \vdash \texttt{i} : \textbf{int}} \qquad \text{Const}\ \dfrac{}{\Gamma \vdash 42 : \textbf{int}}}{\Gamma \vdash \texttt{i - 42} : \textbf{int}}$$

   (b)

$$\text{BinArith}\ \frac{\text{Indir}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{pc} = \textbf{char*}}{\Gamma \vdash \texttt{pc} : \textbf{char*}}}{\Gamma \vdash \texttt{*pc} : \textbf{char}} \qquad \text{Var}\ \dfrac{\Gamma\ \texttt{i} = \textbf{int}}{\Gamma \vdash \texttt{i} : \textbf{int}}}{\text{BinArith}\ \dfrac{\Gamma \vdash \texttt{*pc + i} : \textbf{int} \qquad \text{Const}\ \dfrac{}{\Gamma \vdash 1337 : \textbf{int}}}{\Gamma \vdash \texttt{*pc + i - 1337} : \textbf{int}}}$$

   (c)

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma\ \texttt{ppc} = \textbf{char**}}{\Gamma \vdash \texttt{ppc} : \textbf{char**}}\,\text{Var}}{\Gamma \vdash \texttt{*ppc} : \textbf{char*}}\,\text{Indir}}{\Gamma \vdash \texttt{**ppc} : \textbf{char}}\,\text{Indir} \qquad \dfrac{\dfrac{\dfrac{\Gamma\ \texttt{pc} = \textbf{char*}}{\Gamma \vdash \texttt{pc} : \textbf{char*}}\,\text{Var}}{\Gamma \vdash \texttt{\&pc} : \textbf{char**}}\,\text{Addr}}{\Gamma \vdash \texttt{*\&pc} : \textbf{char*}}\,\text{Indir}}{\Gamma \vdash \texttt{**\&pc} : \textbf{char}}\,\text{Indir}}{\Gamma \vdash \texttt{**ppc - **\&pc} : \textbf{int}}\,\text{BinArith} \qquad \dfrac{}{\Gamma \vdash \texttt{7198} : \textbf{int}}\,\text{Const}$$
$$\dfrac{\Gamma \vdash \texttt{**ppc - **\&pc} : \textbf{int} \qquad \Gamma \vdash \texttt{7198} : \textbf{int}}{\Gamma \vdash \texttt{**ppc - **\&pc != 7198} : \textbf{int}}\,\text{BinComp}$$

.

3. Ein beliebiger Zeiger-Typ ist ausreichend. Denn BinArith lässt sich nur auf zwei Integer-Typen anwenden. Wählen wir z.B. i als `int*` könnte man versuchen die Regeln wie folgt anzuwenden:

$$\dfrac{\dfrac{\lightning}{\Gamma \vdash \texttt{i} : \textbf{int/char}} \qquad \dfrac{}{\Gamma \vdash \texttt{42} : \textbf{int}}\,\text{Const}}{\Gamma \vdash \texttt{i - 42} : \textbf{int}}\,\text{BinArith}$$

Beim roten Blitz (Crash) gibt es jedoch keine Regel die `i` zu einem `int` oder `char` führen würde.

Aufgabe 13.1: Vorbereitung auf das letzte Projekt

Prüfen Sie mittels der Statischen Semantik von C0pb ob folgene Programmausschnitte Fehler aufweisen. Alle Teilprogramme sind in folgendem Kontext einzusetzen:

```
{
    int a;
    char c;
    void v;
    char* cp;
    int ** i;
    a = 4;
    c = '4';
    ...
}
```

1.
```
cp = &v;
while(cp)
    a=v;
```

2.
```
i = &cp;
**i = c+a;
cp =&v;
```

3.
```
cp = &c;
while(cp){
    int* a;
    a=c;
}
```

4.
```
cp = &v;
if(a)
    *cp=a;
else
    *cp=&a;
```

5.
```
c = a;
if(c)
    abort();
else
    *i = &cp;
```

Lösung

$\Gamma := \{a \mapsto \texttt{int}, c \mapsto \texttt{char}, v \mapsto \texttt{void}, cp \mapsto \texttt{char*}, i \mapsto \texttt{int**}\}$

1.

$$\text{While}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{cp} = \textbf{char*}}{\Gamma \vdash \texttt{cp} : \textbf{char*}} \qquad \text{Assign}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{a} = \textbf{int}}{\Gamma \vdash \texttt{a} : \textbf{int}} \qquad \text{Var}\ \dfrac{\Gamma\ \texttt{v} = \textbf{void}}{\Gamma \vdash \texttt{v} : \textbf{void}} \qquad \dfrac{\lightning}{\text{int} \leftrightarrow \text{void}}}{\Gamma \vdash \texttt{a = v;}}}{\Gamma \vdash \texttt{while(cp)a=v;}}$$

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{cp} = \textbf{char*}}{\Gamma \vdash \texttt{cp} : \textbf{char*}} \qquad \text{Addr}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{v} = \textbf{void}}{\Gamma \vdash \texttt{v} : \textbf{void}}}{\Gamma \vdash \texttt{\&v} : \textbf{void*}} \qquad \dfrac{}{\text{char*} \leftrightarrow \text{void*}}}{\Gamma \vdash \texttt{cp = \&v;}}$$

$$\text{Seq}\ \frac{\Gamma \vdash \texttt{cp = \&v;} \qquad \Gamma \vdash \texttt{while(cp) a=v;}}{\Gamma \vdash \texttt{cp = \&v;␣while(cp) a=v;}}$$

2.

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{cp} = \textbf{char*}}{\Gamma \vdash \texttt{cp} : \textbf{char*}} \qquad \text{Addr}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{v} = \textbf{void}}{\Gamma \vdash \texttt{v} : \textbf{void}}}{\Gamma \vdash \texttt{\&v} : \textbf{void*}} \qquad \dfrac{}{\text{void*} \leftrightarrow \text{char*}}}{\Gamma \vdash \texttt{cp = \&v;}}$$

$$\text{Assign}\ \frac{\text{Indir}\ \dfrac{\text{Indir}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{i} = \textbf{int**}}{\Gamma \vdash \texttt{i} : \textbf{int**}}}{\Gamma \vdash \texttt{*i} : \textbf{int*}}}{\Gamma \vdash \texttt{**i} : \textbf{int}} \qquad \text{BinArith}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{c} = \textbf{char}}{\Gamma \vdash \texttt{c} : \textbf{char}} \qquad \text{Var}\ \dfrac{\Gamma\ \texttt{a} = \textbf{int}}{\Gamma \vdash \texttt{a} : \textbf{int}}}{\Gamma \vdash \texttt{c+a} : \textbf{int}} \qquad \dfrac{}{\text{int} \leftrightarrow \text{int}}}{\Gamma \vdash \texttt{**i = c+a;}}$$

$$\text{Seq}\ \frac{\Gamma \vdash \texttt{**i = c+a;} \qquad \Gamma \vdash \texttt{cp = \&v;}}{\Gamma \vdash \texttt{**i = c+a;␣cp = \&v;}}$$

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{i} = \textbf{int**}}{\Gamma \vdash \texttt{i} : \textbf{int**}} \qquad \text{Addr}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{cp} = \textbf{char*}}{\Gamma \vdash \texttt{cp} : \textbf{char*}}}{\Gamma \vdash \texttt{\&cp} : \textbf{char**}} \qquad \dfrac{}{\text{int**} \leftrightarrow \text{char**}}}{\Gamma \vdash \texttt{i = \&cp;}}$$

$$\text{Seq}\ \frac{\Gamma \vdash \texttt{i = \&cp;} \qquad \Gamma \vdash \texttt{**i = c+a; cp = \&v;}}{\Gamma \vdash \texttt{i = \&cp;␣**i = c+a; cp = \&v;}}$$

3.

$\Gamma' \coloneqq \Gamma[a \mapsto \texttt{int*}]$

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\Gamma'\ \texttt{a} = \textbf{int*}}{\Gamma' \vdash \texttt{a} : \textbf{int*}} \qquad \text{Var}\ \dfrac{\Gamma'\ \texttt{c} = \textbf{char}}{\Gamma' \vdash \texttt{c} : \textbf{char}} \qquad \dfrac{}{\text{int*} \leftrightarrow \text{char}}}{\Gamma' \vdash \texttt{a = c;}}$$

$$\text{While}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{cp} = \textbf{char*}}{\Gamma \vdash \texttt{cp} : \textbf{char*}} \qquad \text{Block}\ \dfrac{\Gamma[a \mapsto \texttt{int*}] \vdash \texttt{a = c;}}{\Gamma \vdash \texttt{\{int* a; a = c;\}}}}{\Gamma \vdash \texttt{while(cp)\{int* a; a = c;\}}}$$

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\Gamma\ \texttt{cp} = \textbf{char*}}{\Gamma \vdash \texttt{cp} : \textbf{char*}} \qquad \text{Addr}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{c} = \textbf{char}}{\Gamma \vdash \texttt{c} : \textbf{char}}}{\Gamma \vdash \texttt{\&c} : \textbf{char*}} \qquad \dfrac{}{\text{char*} \leftrightarrow \text{char*}}}{\Gamma \vdash \texttt{cp = \&c;}}$$

$$\text{Seq}\ \frac{\Gamma \vdash \texttt{cp = \&c;} \qquad \Gamma \vdash \texttt{while(cp)\{int* a; a = c;\}}}{\Gamma \vdash \texttt{cp = \&c;␣while(cp)\{int* a; a = c;\}}}$$

4.

$$\text{Assign}\ \frac{\text{Indir}\ \dfrac{\text{Var}\ \dfrac{\Gamma\ \texttt{cp} = \textbf{char*}}{\Gamma \vdash \texttt{cp} : \textbf{char*}}}{\Gamma \vdash \texttt{*cp} : \textbf{char}} \qquad \text{Var}\ \dfrac{\Gamma\ \texttt{a} = \textbf{int}}{\Gamma \vdash \texttt{a} : \textbf{int}} \qquad \dfrac{}{\text{char} \leftrightarrow \text{int}}}{\Gamma \vdash \texttt{*cp = a;}}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma\ \texttt{cp} = \textbf{char*}}\text{Var}}{\Gamma \vdash \texttt{cp} : \textbf{char*}}\text{Indir} \qquad \dfrac{\dfrac{}{\Gamma\ \texttt{a} = \textbf{int}}\text{Var}}{\Gamma \vdash \texttt{a} : \textbf{int}}\text{Addr} \qquad \dfrac{\lightning}{\text{char} \leftrightarrow \text{int*}}}{\Gamma \vdash \texttt{*cp = \&a;}}\text{Assign}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma\ \texttt{a} = \textbf{int}}\text{Var}}{\Gamma \vdash \texttt{a} : \textbf{int}} \qquad \Gamma \vdash \texttt{*cp = a;} \qquad \Gamma \vdash \texttt{*cp = \&a;}}{\Gamma \vdash \texttt{if (a) *cp = a; else *cp = \&a;}}\text{If}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma\ \texttt{cp} = \textbf{char*}}\text{Var}}{\Gamma \vdash \texttt{cp} : \textbf{char*}} \qquad \dfrac{\dfrac{}{\Gamma\ \texttt{v} = \textbf{void}}\text{Var}}{\Gamma \vdash \texttt{\&v} : \textbf{void*}}\text{Addr} \qquad \dfrac{}{\text{char*} \leftrightarrow \text{void*}}}{\Gamma \vdash \texttt{cp = \&v;}}\text{Assign}$$

$$\dfrac{\Gamma \vdash \texttt{cp = \&v;} \qquad \Gamma \vdash \texttt{if (a) *cp = a; else *cp = \&a;}}{\Gamma \vdash \texttt{cp = \&v;}\sqcup\texttt{if (a) *cp = a; else *cp = \&a;}}\text{Seq}$$

5.
$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma\ \texttt{i} = \textbf{int**}}\text{Var}}{\Gamma \vdash \texttt{i} : \textbf{int**}}\text{Indir}}{\Gamma \vdash \texttt{*i} : \textbf{int*}} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma\ \texttt{cp} = \textbf{char*}}\text{Var}}{\Gamma \vdash \texttt{cp} : \textbf{char*}}\text{Addr}}{\Gamma \vdash \texttt{\&cp} : \textbf{char**}} \qquad \dfrac{\lightning}{\text{int*} \leftrightarrow \text{char**}}}{\Gamma \vdash \texttt{*i = \&cp;}}\text{Assign}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma\ \texttt{a} = \textbf{int}}\text{Var}}{\Gamma \vdash \texttt{a} : \textbf{int}} \qquad \dfrac{}{\Gamma \vdash \texttt{abort();}}\text{Abort} \qquad \Gamma \vdash \texttt{*i = \&cp;}}{\Gamma \vdash \texttt{if (c) abort(); else *i = \&cp;}}\text{If}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma\ \texttt{c} = \textbf{char}}\text{Var}}{\Gamma \vdash \texttt{c} : \textbf{char}} \qquad \dfrac{\dfrac{}{\Gamma\ \texttt{a} = \textbf{int}}\text{Var}}{\Gamma \vdash \texttt{a} : \textbf{int}} \qquad \dfrac{}{\text{char} \leftrightarrow \text{int}}}{\Gamma \vdash \texttt{c = a;}}\text{Assign}$$

$$\dfrac{\Gamma \vdash \texttt{c = a;} \qquad \Gamma \vdash \texttt{if (c) abort(); else *i = \&cp;}}{\Gamma \vdash \texttt{c = a;}\sqcup\texttt{if (c) abort(); else *i = \&cp;}}\text{Seq}$$

Aufgabe 13.2: Codegenerierung: Der Anfang!

Generieren Sie MIPS Code für die folgenden C0 Programme. Bauen Sie dazu den Inferenzbaum aus den Regeln zur Codegenerierung auf und schreiben Sie anschlieSSend die Codesequenz auf. Nehmen Sie $\Omega = \{x \mapsto 0\}$ als Umgebung an.

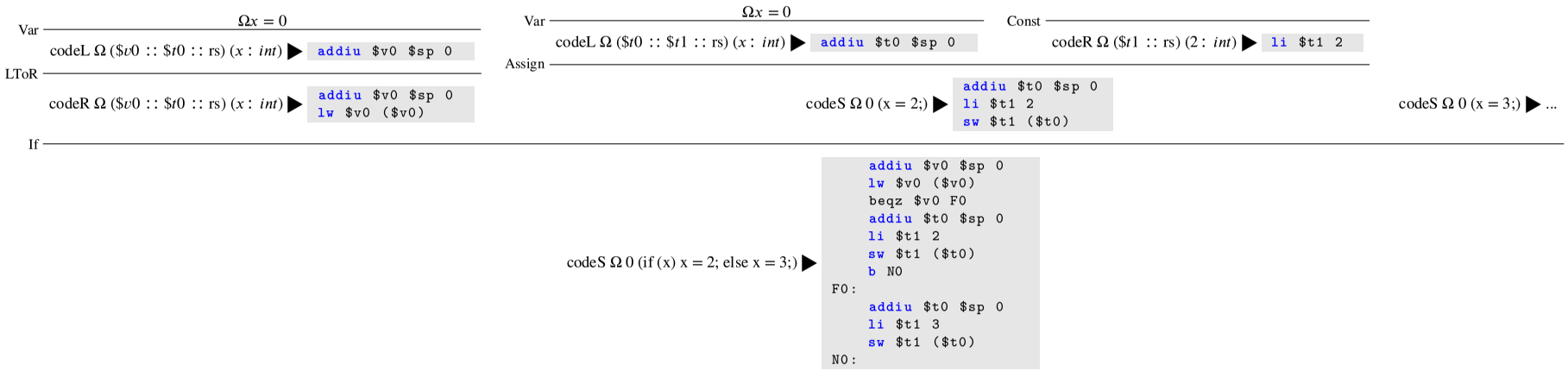
1. `(x-2) + 3`
2. `if (x) x = 2; else x = 3;`
3. `while (1) x = x + 1;`

Lösung

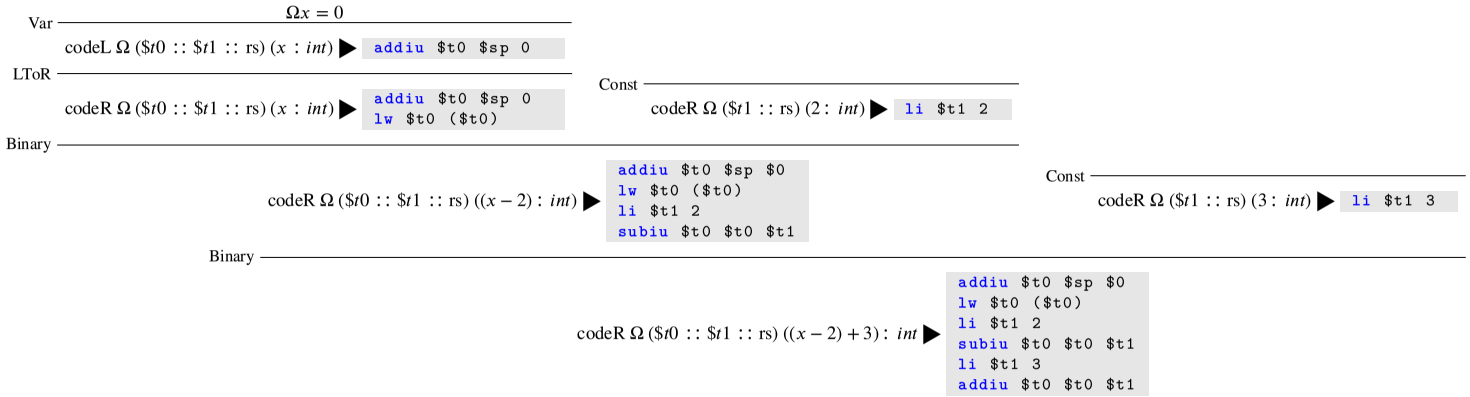
1. Lösung siehe nächste Seite
2. Lösung siehe nächste Seite
3. Mit der Regel `While` können wir den Code der Schleife generieren. Hierzu muss jedoch das `e = 1` zu einer Konstanten auswerten, was in diesem Fall trivial ist. Mittels `Assign` können wir dann die Zuweisung auflösen. Der finale Code ist dann:

```
 1    b T0
 2
 3  L0:
 4    addiu $v0 $sp 0
 5
 6    addiu $v1 $sp 0
 7    lw $v1 ($v1)
 8    li $v2 1
 9    addiu $v1 $v1 $v2
10
11    sw $v1 ($v0)
12
13  T0:
14    li $v0 1
15    bnez $v0 L0
```

2.


Var
$\Omega x = 0$
codeL Ω ($t0 :: $t0 :: rs) (x : int) ▶ addiu $v0 $sp 0
LToR
codeR Ω ($t0 :: $t0 :: rs) (x : int) ▶ addiu $v0 $sp 0
lw $v0 ($v0)
Var
$\Omega x = 0$
codeL Ω ($t0 :: $t1 :: rs) (x : int) ▶ addiu $t0 $sp 0
Const
codeR Ω ($t1 :: rs) (2 : int) ▶ li $t1 2
Assign
codeS Ω 0 (x = 2;) ▶ addiu $t0 $sp 0
li $t1 2
sw $t1 ($t0)
codeS Ω 0 (x = 3;) ▶ ...
If
codeS Ω 0 (if (x) x = 2; else x = 3;) ▶
addiu $v0 $sp 0
lw $v0 ($v0)
beqz $v0 F0
addiu $t0 $sp 0
li $t1 2
sw $t1 ($t0)
b N0
F0:
addiu $t0 $sp 0
li $t1 3
sw $t1 ($t0)
N0:

1.


Var
$\Omega x = 0$
codeL Ω ($t0 :: $t1 :: rs) (x : int) ▶ addiu $t0 $sp 0
LToR
codeR Ω ($t0 :: $t1 :: rs) (x : int) ▶ addiu $t0 $sp 0
lw $t0 ($t0)
Const
codeR Ω ($t1 :: rs) (2 : int) ▶ li $t1 2
Binary
codeR Ω ($t0 :: $t1 :: rs) ((x − 2) : int) ▶ addiu $t0 $sp $0
lw $t0 ($t0)
li $t1 2
subiu $t0 $t0 $t1
Const
codeR Ω ($t1 :: rs) (3 : int) ▶ li $t1 3
Binary
codeR Ω ($t0 :: $t1 :: rs) ((x − 2) + 3) : int ▶ addiu $t0 $sp $0
lw $t0 ($t0)
li $t1 2
subiu $t0 $t0 $t1
li $t1 3
addiu $t0 $t0 $t1

Aufgabe 13.3: Codegenerierung: Swap

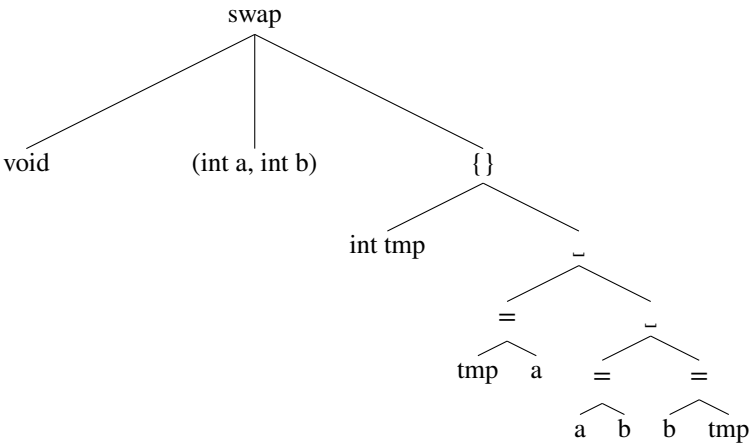
Betrachten Sie folgendes Programm:

```
void almost_swap(int a, int b) {
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

1. Zeichnen Sie den abstrakten Syntaxbaum des Programms.
2. Übersetzen Sie das Programm in MIPS-Code. Benutzen Sie dafür die Regeln aus dem Skript. (Abschnitt 10.2 Syntaxgesteuerte Code-Erzeugung)

Lösung

1.



2. Im Folgenden wird der entstehende Inferenzbaum zur Übersichtlichkeit in mehrere Teile unterteilt:
Da wir eine Funktion übersetzen wollen, müssen wir mit der Regel `Function` beginnen:

$$\text{Function}\ \frac{\texttt{codeS}\ \ \{\}\ \ 0\ \ \texttt{\{ int a; int b; int tmp; ... \}} \blacktriangleright c, S}{\texttt{codeFunc (void swap(int a, int b) \{ int tmp; ... \})} \blacktriangleright}$$

```
swap:
    subiu $sp $sp (S + 4)
    sw $ra S($sp)
    sw $a0 0($sp)
    sw $a1 4($sp)
    c
swap_end:
    lw $ra S($sp)
    addiu $sp $sp (S + 4)
    jr $ra
```

Da die Anweisung unseres `codeS` Ausdrucks ein Block ist, wollen wir auch die gleichnamige Regel `Block` verwenden:

$$\text{Seq}\ \frac{\text{Seq}\ \dfrac{\text{Block}\ \dfrac{\texttt{codeS}\ \ \{\}\ \ 0\ \ (\{\ \texttt{int a; int b; int tmp; ...}\ \})\ \blacktriangleright\ c_1, S_1}{\texttt{codeS}\ \ \{a \mapsto 0, b \mapsto 4, tmp \mapsto 8\}\ \ 12\ \ (\texttt{tmp = a;}\_\ldots)\ \blacktriangleright\ c_1, S_1}\quad \texttt{codeS}\ \{a \mapsto 0, ...\}\ 12\ (\texttt{tmp = a;})\ \blacktriangleright\ c_2, S_2 \texttt{codeS}\ \{a \mapsto 0, ...\}\ 12\ (\texttt{a=b;\_b=tmp})\ \blacktriangleright\ c_3, S_3}{\text{codeS}\ \{a \mapsto 0, ...\}\ 12\ (\texttt{a = b;})\ \blacktriangleright\ c_4, S_4 \text{codeS}\ \{a \mapsto 0, ...\}\ 12\ (\texttt{b = tmp;})\ \blacktriangleright\ c_5, S_5}$$

Dabei ist $S_1 = max(S_2, S_3)$ und $S_3 = max(S_4, S_5)$. AuSSerdem sind c_1 und c_3 wie folgt definiert:

c_2
c_3

Code von c_1

c_4
c_5

Code von c_3

c_2
c_4
c_5

$\Rightarrow$ entstandener Code von c_1

Nun wollen wir also c_2, c_4 und c_5 herleiten, um unseren Code zu erzeugen. Beginnen wir mit c_2:

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\{a \mapsto 0, ...\}tmp = 8}{\text{codeL}\ \{a \mapsto 0, ...\}\ (\$t0 :: rs)\ (tmp : int)\ \blacktriangleright\ \texttt{addiu \$t0 \$sp 8}}\quad \dfrac{\dfrac{\{a \mapsto 0, ...\}a = 0}{\text{codeL}\ \{a \mapsto 0, ...\}\ (\$t1 :: rs')\ (a : int)\ \blacktriangleright\ \texttt{addiu \$t1 \$sp 0}}\ \text{Var}}{\text{codeR}\ \{a \mapsto 0, ...\}\ (\$t1 :: rs')\ (a : int)\ \blacktriangleright\ \texttt{addiu \$t1 \$sp 0}\ \texttt{lw \$t1 (\$t1)}}\ \text{LToR}}{\text{codeS}\ \{a \mapsto 0, b \mapsto 4, tmp \mapsto 8\}\ 12\ (\texttt{tmp = a;})\ \blacktriangleright\ \texttt{addiu \$t0 \$sp 8}\ \texttt{addiu \$t1 \$sp 0}\ \texttt{lw \$t1 (\$t1)}\ \texttt{sw \$t1 (\$t0)}, 12}$$

c_4 entsteht ganz analog:

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\{a \mapsto 0, ...\}a = 0}{\text{codeL}\ \{a \mapsto 0, ...\}\ (\$t0 :: rs)\ (a : int)\ \blacktriangleright\ \texttt{addiu \$t0 \$sp 0}}\quad \dfrac{\dfrac{\{a \mapsto 0, ...\}b = 4}{\text{codeL}\ \{a \mapsto 0, ...\}\ (\$t1 :: rs')\ (b : int)\ \blacktriangleright\ \texttt{addiu \$t1 \$sp 4}}\ \text{Var}}{\text{codeR}\ \{a \mapsto 0, ...\}\ (\$t1 :: rs')\ (b : int)\ \blacktriangleright\ \texttt{addiu \$t1 \$sp 4}\ \texttt{lw \$t1 (\$t1)}}\ \text{LToR}}{\text{codeS}\ \{a \mapsto 0, b \mapsto 4, tmp \mapsto 8\}\ 12\ (\texttt{a = b;})\ \blacktriangleright\ \texttt{addiu \$t0 \$sp 0}\ \texttt{addiu \$t1 \$sp 4}\ \texttt{lw \$t1 (\$t1)}\ \texttt{sw \$t1 (\$t0)}, 12}$$

Und c_5 auch:

$$\text{Assign}\ \frac{\text{Var}\ \dfrac{\{a \mapsto 0, ...\}b = 4}{\text{codeL}\ \{a \mapsto 0, ...\}\ (\$t0 :: rs)\ (b : int)\ \blacktriangleright\ \texttt{addiu \$t0 \$sp 4}}\quad \dfrac{\dfrac{\{a \mapsto 0, ...\}tmp = 8}{\text{codeL}\ \{a \mapsto 0, ...\}\ (\$t1 :: rs')\ (tmp : int)\ \blacktriangleright\ \texttt{addiu \$t1 \$sp 8}}\ \text{Var}}{\text{codeR}\ \{a \mapsto 0, ...\}\ (\$t1 :: rs')\ (tmp : int)\ \blacktriangleright\ \texttt{addiu \$t1 \$sp 8}\ \texttt{lw \$t1 (\$t1)}}\ \text{LToR}}{\text{codeS}\ \{a \mapsto 0, b \mapsto 4, tmp \mapsto 8\}\ 12\ (\texttt{b = tmp;})\ \blacktriangleright\ \texttt{addiu \$t0 \$sp 4}\ \texttt{addiu \$t1 \$sp 8}\ \texttt{lw \$t1 (\$t1)}\ \texttt{sw \$t1 (\$t0)}, 12}$$

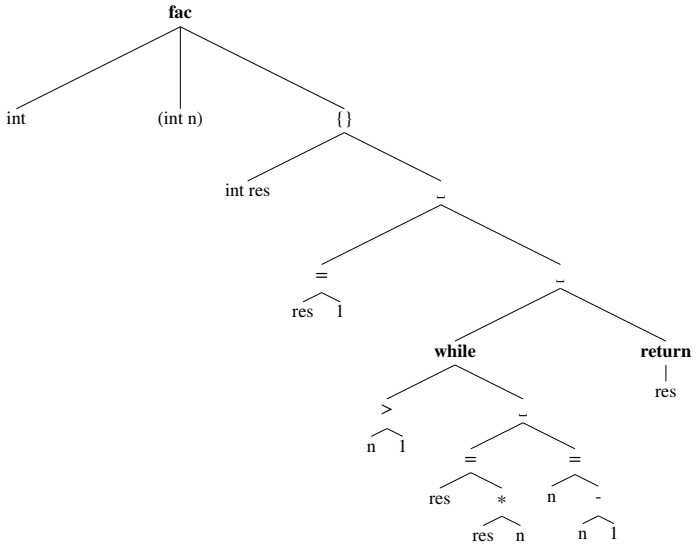
Damit gilt $S = 12$ und wir können unser c zusammenbauen. Unser endgültiger Code sieht also wie folgt aus:

```
swap:
    subiu $sp $sp 16
    sw $ra 12($sp)
    sw $a0 0($sp)
    sw $a1 4($sp)
    addiu $t0 $sp 8
    addiu $t1 $sp 0
    lw $t1 ($t1)
    sw $t1 ($t0)
    addiu $t0 $sp 0
    addiu $t1 $sp 4
    lw $t1 ($t1)
    sw $t1 ($t0)
    addiu $t0 $sp 4
    addiu $t1 $sp 8
    lw $t1 ($t1)
    sw $t1 ($t0)
swap_end:
    lw $ra 12($sp)
    addiu $sp $sp 16
    jr $ra
```

Aufgabe 13.4: Codegenerierung: While

Übersetzen Sie folgendes Programm mit dem angegebenen Syntaxbaum in MIPS-Code. Benutzen Sie dafür die Regeln aus dem Skript.


```
int fac(int n) {
    int res;
    res = 1;
    while (n > 1) {
        res = res * n;
        n = n - 1;
    }
    return res;
}
```



Lösung

```
1     .global fac
2 #function prolog
3 fac:
4     subiu    $sp $sp 12   # offset = 1x LocalVar +1x Arg + ra =12
5     sw       $ra 8($sp)   # ra
6     sw       $a0 ($sp)    # int n
7     addiu    $t0 $sp 4    # L-Auswertung res
8     li       $t1 1        # make 1
9     sw       $t1 ($t0)    # store in res
10    b while_end           # jump while cond
11
12 while_body:
13    #res=res*n
14    addiu    $t0 $sp 4    # get address res (L-Auswertung)
15    addiu    $t1 $sp 4    # get address res (R-Auswertung)
16    lw       $t1 ($t1)    # load res
17    addiu    $t2 $sp 0    # get adress n
18    lw       $t2 ($t2)    # load n
19    mul      $t3 $t1 $t2  # mult res * n
20    sw       $t3 ($t0)    # store in res
21    #n=n-1
22    addiu    $t0 $sp 0    # get adress n (L-Auswertung)
23    addiu    $t1 $sp 0    # get adress n (R-Auswertung)
24    lw       $t1 ($t1)    # load n
25    li       $t2 1        # make 1
26    sub      $t3 $t1 $t2  # n-1
27    sw       $t3 ($t0)    # store in n
28
29 while_end:
30    addiu    $t0 $sp 0    # get adress n
31    lw       $t0 ($t0)    # load n
32    li       $t1 1        # make 1
33    sgt      $t0 $t0 $t1  # n > 1
34    bnez     $t0 while_body  # if true do body again
35    #end while
36    #return res
37    addiu    $t0 $sp 4    # get adress res
38    lw       $t0 ($t0)    # load res
39    move     $v0 $t0      # move res to $v0 as return
40    #end return res
41
42 fac_end:
43 #function epilog
44    lw       $ra 8($sp)   # reset ra to org. val
45    addiu    $sp $sp 12   # reset sp offset
46    jr       $ra          # jr
```

Aufgabe 13.5: Unterprogrammaufrufe

Fügen Sie der Codegenerierung von C0 Unterprogrammaufrufe hinzu. Definieren Sie dazu eine Inferenzregel im Stil der Inferenzregeln die im Kapitel 10.2 des Skripts zu finden sind.

Lösung

$$\text{Function Call}\ \frac{\begin{array}{c}\text{codeR}\ \Omega\ (r_1 :: r_2 :: \ldots :: r_{n+1} :: rs)\ e_0 = c_0 \\ \text{codeR}\ \Omega\ (r_2 :: r_3 :: \ldots :: r_{n+1} :: rs)\ e_1 = c_1 \\ \vdots \\ \text{codeR}\ \Omega\ (r_{n+1} :: rs)\ e_n = c_n\end{array}}{\text{codeR}\ \Omega\ (r_0 :: r_1 :: \ldots :: r_{n+1} :: rs)\ (\texttt{f}(e_0, e_1, \ldots, e_n)) = \begin{array}{l} c_0 \\ c_1 \\ \vdots \\ c_n \\ \texttt{subiu \$sp \$sp } ((m+1)\cdot 4) \\ \texttt{sw \$t0 0(\$sp)} \\ \texttt{sw \$t1 4(\$sp)} \\ \vdots \\ \texttt{sw \$t}m\ 4m\texttt{(\$sp)} \\ \texttt{move \$a0 } r_1 \\ \texttt{move \$a1 } r_2 \\ \vdots \\ \texttt{move \$a}n\ r_{n+1} \\ \texttt{jal } f \\ \texttt{lw \$t0 0(\$sp)} \\ \texttt{lw \$t1 4(\$sp)} \\ \vdots \\ \texttt{lw \$t}m\ 4m\texttt{(\$sp)} \\ \texttt{addiu \$sp \$sp } ((m+1)\cdot 4) \\ \texttt{move } r_0\ \texttt{\$v0}\end{array}$$

Aufgabe 13.6: For Schleife

Ergänzen Sie die C0 Codegenerierung um das Konstrukt der For-Schleife.
Abstrakte Syntax: `for` $(s_{init}, e, s_{inc})\ s_{body}$

Lösung

$$\text{For}\ \frac{\begin{array}{c}\text{codeS}\ \Omega\ \delta\ (s_{init}) = c_0, S_0 \quad \text{codeR}\ \Omega\ (r :: rs)\ e = c_1 \\ \text{codeS}\ \Omega\ \delta\ (s_{inc}) = c_2, S_2 \quad \text{codeS}\ \Omega\ \delta\ (s_{body}) = c_3, S_3 \quad n \text{ frisch}\end{array}}{\text{codeS}\ \Omega\ \delta\ (\texttt{for}\ (s_{init}, e, s_{inc})\ s_{body}) = \begin{array}{l} \quad c_0 \\ \quad \texttt{b } \texttt{T}n \\ \texttt{L}n\texttt{:} \\ \quad c_3 \\ \quad c_2 \\ \texttt{T}n\texttt{:} \\ \quad c_1 \\ \quad \texttt{bnez } r\ \texttt{L}n\end{array}, \max(S_0, S_2, S_3)}$$