TECHNISCHE UNIVERSITÄT DRESDEN

PROGRAMMIERUNG

ÜBUNG 11: C_1 UND ABSTRAKTE MASCHINE AM_1

Eric Kunze

eric.kunze@tu-dresden.de

TU Dresden, 29. Juni 2022

letzte Änderung:
29.06.2022, 15:29

INHALT

Implementierung von C_1 und abstrakte Maschine AM_1

C_1 **UND** AM_1

- **bisher:** Implementierung von C_0 mit AM_0
- **jetzt:** Erweiterung auf C_1 mit AM_1

C_1 **UND** AM_1

- **bisher:** Implementierung von C_0 mit AM_0
- **jetzt:** Erweiterung auf C_1 mit AM_1
  - Erweiterung um Funktionen *ohne* Rückgabewert
  - Einschränkungen von C_0 bleiben erhalten

C_1 **UND** AM_1

- **bisher:** Implementierung von C_0 mit AM_0
- **jetzt:** Erweiterung auf C_1 mit AM_1
  - Erweiterung um Funktionen *ohne* Rückgabewert
  - Einschränkungen von C_0 bleiben erhalten
- **Implementierung** durch
  - Syntax von C_1
  - Befehle und Semantik einer abstrakten Maschine AM_1
  - Übersetzer $C_1 \leftrightarrow AM_1$

ABSTRAKTE MASCHINE AM_1

Die AM_1 besteht aus

- einem Ein- und Ausgabeband,
- einem Datenkeller,
- einem *Laufzeitkeller*,
- einem Befehlszähler und
- einem *Referenzzeiger* (REF).

Im Vergleich zur AM_0 ist also aus dem Hauptspeicher ein *Laufzeitkeller* geworden und der *Referenzzeiger* ist hinzugekommen.

Den Zustand der AM_1 beschreiben wir daher nun mit einem 6-Tupel

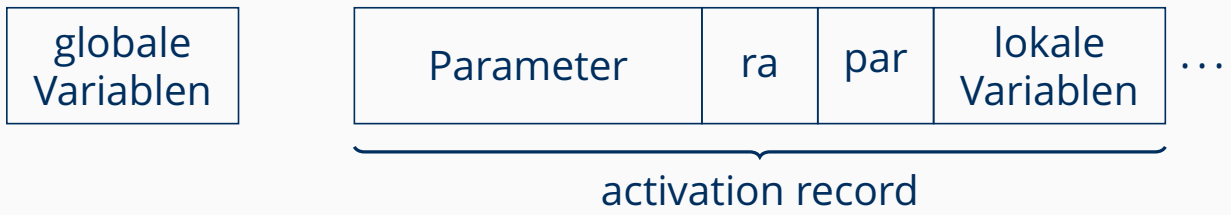
$$(m, d, h, r, inp, out) = (\text{BZ}, \text{DK}, \text{LZK}, \text{REF}, \text{Input}, \text{Output})$$

FUNKTIONSAUFRUFE & DER LAUFZEITKELLER

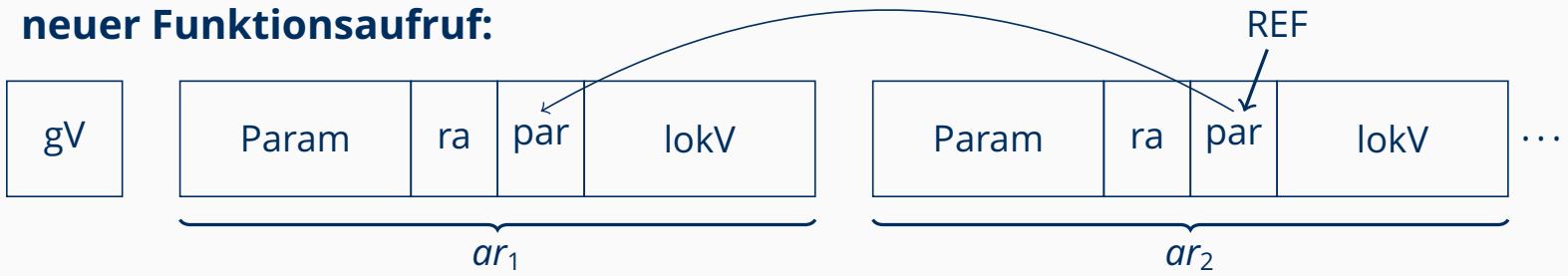
Wofür brauchen wir den REF?

$\rightarrow$ Funktionsaufrufe & Rücksprünge

Struktur des Laufzeitkellers:



neuer Funktionsaufruf:

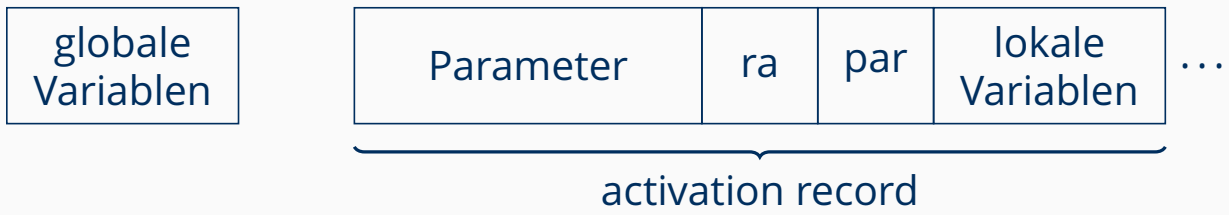


FUNKTIONSAUFRUFE & DER LAUFZEITKELLER

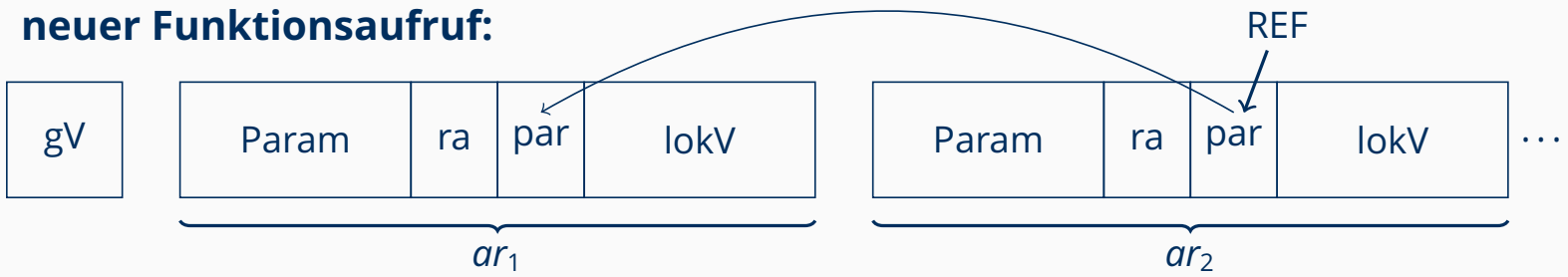
Wofür brauchen wir den REF?

$\rightarrow$ Funktionsaufrufe & Rücksprünge

Struktur des Laufzeitkellers:



neuer Funktionsaufruf:



Funktionsaufrufe übersetzen:

- Parameter `LOAD` & `PUSH`
- Funktion `CALL`

BEFEHLSSEMANTIK DER AM_1

$b \in \{\text{global}, \text{lokal}\}$
r ...aktueller REF

$$adr(r, b, o) = \begin{cases} r + o & \text{wenn } b = \text{lokal} \\ o & \text{wenn } b = \text{global} \end{cases}$$

BEFEHLSSEMANTIK DER AM_1

$b \in \{\text{global}, \text{lokal}\}$
r ...aktueller REF

$$adr(r, b, o) = \begin{cases} r + o & \text{wenn } b = \text{lokal} \\ o & \text{wenn } b = \text{global} \end{cases}$$

Befehl	**Auswirkungen**
LOAD(b,o)	Lädt den Inhalt von Adresse $adr(r, b, o)$ auf den Datenkeller, inkrementiere Befehlszähler
STORE(b,o)	Speichere oberstes Datenkellerelement an $adr(r, b, o)$, inkrementiere Befehlszähler
WRITE(b,o)	Schreibe Inhalt an Adresse $adr(r, b, o)$ auf das Ausgabeband, inkrementiere Befehlszähler
READ(b,o)	Lies oberstes Element vom Eingabeband, speichere an Adresse $adr(r, b, o)$, inkrementiere Befehlszähler

BEFEHLSSEMANTIK DER AM_1

Befehl	Auswirkungen
LOADI(o)	Ermittle Wert ($= b$) an Adresse $r + o$, Lade Inhalt von Adresse b auf Datenkeller, inkrementiere Befehlszähler
STOREI(o)	Ermittle Wert ($= b$) an Adresse $r+o$, nimm oberstes Datenkellerelement, speichere dieses an Adresse b, inkrementiere Befehlszähler
WRITEI(o)	Ermittle Wert ($= b$) an Adresse $r+o$, schreibe den Inhalt an Adresse b auf Ausgabeband, inkrementiere Befehlszähler
READI(o)	Ermittle Wert ($= b$) an Adresse $r + o$, lies das oberste Element vom Eingabeband, speichere es an Adresse b , inkrementiere Befehlszähler
LOADA(b,o)	Lege $adr(r, b, o)$ auf Datenkeller, inkrementiere Befehlszähler
PUSH	oberstes Element vom Datenkeller auf Laufzeitkeller, Befehlszähler inkrementieren
CALL adr	Befehlszählerwert inkrementieren und auf LZK legen, Befehlszähler auf adr setzen, REF auf LZK legen, REF auf Länge des LZK ändern
INIT n	n-mal 0 auf den Laufzeitkeller legen
RET n	im LZK alles nach REF-Zeiger löschen, oberstes Element des LZK als REF setzen, oberstes Element des LZK als Befehlszähler setzen, n Elemente von LZK löschen

MERKHILFEN

Übersetzen:

- `*x` wird mit `I`-Befehlen übersetzt (außer in Funktionsköpfen)
- `&x` wird mit `A`-Befehlen übersetzt
- `BEFEHL`(*global*, *o*) verhält sich wie in der AM_0
- `BEFEHL`(*lokal*, *o*) verhält sich ähnlich wie in der AM_0 mit *Adressberechnung* ($r + o$) vorher

MERKHILFEN

Übersetzen:

- `*x` wird mit `I`-Befehlen übersetzt (außer in Funktionsköpfen)
- `&x` wird mit `A`-Befehlen übersetzt
- `BEFEHL`(*global*, *o*) verhält sich wie in der AM_0
- `BEFEHL`(*lokal*, *o*) verhält sich ähnlich wie in der AM_0 mit *Adressberechnung* ($r + o$) vorher

Ablaufprotokolle:

- `I`-Befehle: Wert-an-Adresse-Prozess zweimal machen
- `A`-Befehle: Adresse direkt verarbeiten (nicht erst Wert auslesen)

Übungsblatt 11

AUFGABE 1 – TEIL (A)

Aufgabe. Gegeben ist folgender AM_1-Code:

```
1  INIT 1;
2  CALL 18;
3  INIT 0;
4  LOAD(lokal,-2);
5  LIT 0;
6  GT;
7  JMC 17;
8  LIT 2;
9  LOADI(-3);
10 MUL;
11 STOREI(-3);
12 LOAD(lokal,-2);
13 LIT 1;
14 SUB;
15 STORE(lokal,-2);
16 JMP 4;
17 RET 2;
18 INIT 0;
19 READ(global,1);
20 LOADA(global,1);
21 PUSH;
22 LOAD(global,1);
23 PUSH;
24 CALL 3;
25 WRITE(global,1);
26 JMP 0;
```

Führen Sie 12 Schritte der AM_1 auf der Konfiguration

$\sigma = (22, \varepsilon, 1:3:0:1, 3, \varepsilon, \varepsilon)$ aus.

AUFGABE 1 – TEIL (A)

Lösung.

BZ	DK	LZK	REF	Inp	Out
22	ε	1:3:0:1	3	ε	ε
23	1	1:3:0:1	3	ε	ε
24	ε	1:3:0:1:1	3	ε	ε
3	ε	1:3:0:1:1:25:3	7	ε	ε
4	ε	1:3:0:1:1:25:3	7	ε	ε
5	1	1:3:0:1:1:25:3	7	ε	ε
6	0:1	1:3:0:1:1:25:3	7	ε	ε
7	1	1:3:0:1:1:25:3	7	ε	ε
8	ε	1:3:0:1:1:25:3	7	ε	ε
9	2	1:3:0:1:1:25:3	7	ε	ε
10	1:2	1:3:0:1:1:25:3	7	ε	ε
11	2	1:3:0:1:1:25:3	7	ε	ε
12	ε	2:3:0:1:1:25:3	7	ε	ε
13	1	2:3:0:1:1:25:3	7	ε	ε
14	1:1	2:3:0:1:1:25:3	7	ε	ε

AUFGABE 1 – TEIL (B)

Gegeben sei folgendes C_1-Programm:

```
1  #include <stdio.h>
2  int b;
3  void f(int a, int *b) {
4    int c;
5    c = a;
6    while (c > 0) {
7      *b = *b * 2;
8      c = c - 1;
9    }
10   }
11
12   void main () {
13     int a;
14     scanf("%i", &a);
15     b = 1;
16     f(a, &b);
17     printf("%d", b);
18   }
```

Übersetzen Sie das Programm mittels *trans* in ein AM_1-Programm mit baumstrukturierten Adressen. Geben Sie zunächst die Symboltabellen $tab_{\texttt{main}}$ und $tab_{\texttt{f}}$ zur Übersetzung der Statements in den Funktionen `main` bzw. `f` mittels *stseqtrans* an. Geben Sie keine weiteren Zwischenschritte an.

AUFGABE 1 – TEIL (B)

Symboltabellen:

$$\text{lokal-tab}_{\texttt{f}} = [\texttt{f}/(\text{proc}, 1), \texttt{a}/(\text{var}, \text{lokal}, -3), \texttt{b}/(\text{var-ref}, -2), \texttt{c}/(\text{var}, \text{lokal}, 1)]$$

$$\text{lokal-tab}_{\texttt{main}} = [\texttt{f}/(\text{proc}, 1), \texttt{b}/(\text{var}, \text{global}, 1), \texttt{a}/(\text{var}, \text{lokal}, 1)]$$

Übersetzung:

```
    INIT 1;
    CALL 2;
    JMP 0;

1   INIT 1;
    LOAD(lokal, -3);
    STORE(lokal, 1);
```

```
1.2.1   LOAD(lokal, 1);
        LIT 0;
        GT;
        JMC 1.2.2;
        LOADI(-2);
        LIT 2;
        MUL;
        STOREI(-2);
        LOAD(lokal, 1);
        LIT 1;
        SUB;
        STORE(lokal, 1);
        JMP 1.2.1;
1.2.2   RET 2;
```

```
2   INIT 1;
    READ(lokal, 1);
    LIT 1;
    STORE(global, 1);
    LOAD(lokal, 1);
    PUSH;
    LOADA(global, 1);
    PUSH;
    CALL 1;
    WRITE(global, 1);
    RET 0;
```

AUFGABE 2 – TEIL (A)

Gegeben sei folgendes Fragment eines C_1-Programms mit den Funktionen `g` und `f`:

```
1 while (y <= a)
2   if (y < *x)
3     g(&z, *x);
4 *x = y + 1;
```

Übersetzen Sie dieses Fragment mittels *stseqtrans* in AM_1-Code mit baumstrukturierten Adressen. Sie müssen keine Zwischenschritte angeben. Nehmen Sie dabei an, dass die `while`-Anweisung das erste Statement in `f` ist, und es sei

$$tab_{\text{f+lDecl}} = [\ \mathtt{g}/(\text{proc}, 1), \mathtt{f}/(\text{proc}, 2), \mathtt{a}/(\text{var}, \text{global}, 1), \mathtt{x}/(\text{var-ref}, -3), \mathtt{y}/(\text{var}, \text{lokal}, -2), \mathtt{z}/(\text{var}, \text{lokal}, 1)\ ].$$

AUFGABE 2 – TEIL (A)

Lösung.

```
2.1.1:   LOAD(lokal, -2); LOAD(global, 1); LE;
         JMC 2.1.2;
         LOAD(lokal, -2); LOADI(-3); LT;
         JMC 2.1.3.1;
         LOADA(lokal, 1); PUSH;
         LOADI(-3); PUSH;
         CALL 1;
2.1.3.1: JMP 2.1.1;
2.1.2:   LOAD(lokal, -2); LIT 1; ADD; STOREI(-3);
```

AUFGABE 2 – TEIL (B)

Gegeben sei folgender AM_1-Code:

```
1  INIT 1;
2  CALL 26;
3  JMP 0;
4  INIT 0;
5  LOAD(global,1);
6  LIT 1;
7  SUB;
8  STORE(global,1);
9  LOAD(global,1);
10 LIT 1;
11 GT;
12 JMC 23;
13 LOAD(lokal,-4);
14 PUSH;
15 LOAD(lokal,-2);
16 PUSH;
17 LOAD(lokal,-2);
18 LOAD(lokal,-3);
19 ADD;
20 PUSH;
21 CALL 4;
22 JMP 25;
23 LOAD(lokal,-2);
24 STOREI(-4);
25 RET 3;
26 INIT 1;
27 LIT 42;
28 STORE(lokal,1);
29 READ(global,1);
30 LOADA(lokal,1);
31 PUSH;
32 LIT 1;
33 PUSH;
34 LIT 1;
35 PUSH;
36 CALL 4;
37 WRITE(lokal,1);
38 RET 0;
```

Erstellen Sie ein Ablaufprotokoll der AM_1, indem Sie sie schrittweise ablaufen lassen, bis die Maschine terminiert. Die Startkonfiguration sei $\sigma = (21, \varepsilon, 2:3:0:42:4:1:1:37:3:4:1:2, 9, \varepsilon, \varepsilon)$.

AUFGABE 2 – TEIL (B)

Lösung.

Sind noch Fehler vorhanden?

	BZ		DK		LZK		REF		Inp		Out	
(	21	,	ε	,	2:3:0:42:4:1:1:37:3:4:1:2	,	9	,	ε	,	ε	)
(	4	,	ε	,	2:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	5	,	ε	,	2:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	6	,	2	,	2:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	7	,	1:2	,	2:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	8	,	1	,	2:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	9	,	ε	,	1:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	10	,	1	,	1:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	11	,	1:1	,	1:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	12	,	0	,	1:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	23	,	ε	,	1:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	24	,	2	,	1:3:0:42:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	25	,	ε	,	1:3:0: 2:4:1:1:37:3:4:1:2:22:9	,	14	,	ε	,	ε	)
(	22	,	ε	,	1:3:0: 2:4:1:1:37:3	,	9	,	ε	,	ε	)
(	25	,	ε	,	1:3:0: 2:4:1:1:37:3	,	9	,	ε	,	ε	)
(	37	,	ε	,	1:3:0: 2	,	3	,	ε	,	ε	)
(	38	,	ε	,	1:3:0: 2	,	3	,	ε	,	2	)
(	3	,	ε	,	1	,	0	,	ε	,	2	)
(	0	,	ε	,	1	,	0	,	ε	,	2	)

Ein weiteres Beispiel aus der Aufgabensammlung

AUFGABE AGS 15.16 – TEIL (A)

Aufgabe.

```
1 #include <stdio.h>
2 int x, y;
3 void f(...) {...}
4 void g(int a, int *b) {
5   int c;
6   c = 3;
7   if (c == *b) while (a > 0) f(&a, b);
8 }
9 void main () {...}
```

Übersetzen Sie die Sequenz der Statements im Rumpf von g in entsprechenden AM_1-Code mit baumstrukturierten Adressen (mittels *stseqtrans*). Sie brauchen keine Zwischenschritte anzugeben. Geben Sie zunächst die benötigte Symboltabelle tab_g an.

AUFGABE AGS 15.16 – TEIL (A)

Lösung.

$$\begin{aligned} tab_{\mathrm{g}} = [\ & \mathtt{f}/(proc, 1), \mathtt{g}/(proc, 2), \\ & \mathtt{x}/(var, global, 1), \mathtt{y}/(var, global, 2), \\ & \mathtt{a}/(var, lokal, -3), \mathtt{b}(var - ref, -2), \mathtt{c}/(var, lokal, 1)\ ] \end{aligned}$$

AUFGABE AGS 15.16 – TEIL (A)

Lösung.

$$tab_{\mathrm{g}} = [\ \mathtt{f}/(\mathit{proc}, 1), \mathtt{g}/(\mathit{proc}, 2),$$
$$\mathtt{x}/(\mathit{var}, \mathit{global}, 1), \mathtt{y}/(\mathit{var}, \mathit{global}, 2),$$
$$\mathtt{a}/(\mathit{var}, \mathit{lokal}, -3), \mathtt{b}(\mathit{var} - \mathit{ref}, -2), \mathtt{c}/(\mathit{var}, \mathit{lokal}, 1)\ ]$$

```
            LIT 3; STORE(lokal,1);
            LOAD(lokal,1); LOADI (-2); EQ; JMC 2.2.1;
2.2.2.1:    LOAD(lokal,-3); LIT 0; GT; JMC 2.2.2.2;
            LOADA(lokal,-3); PUSH;
            LOAD(lokal,-2); PUSH; CALL 1;
            JMP 2.2.2.1;
2.2.2.2:    2.2.1:
```

AUFGABE AGS 15.18 – TEIL (B)

Aufgabe.

```
1  INIT 1;
2  CALL 13;
3  INIT 0;
4  LOADI (-2);
5  LIT 2;
6  GT;
7  JMC 12;
8  LOADI (-2);
9  LIT 2;
10 DIV;
11 STOREI (-2);
12 RET 1;
13 INIT 0;
14 READ(global , 1);
15 LOADA(global , 1)
       ;
16 PUSH;
17 CALL 3;
18 WRITE(global , 1)
       ;
19 JMP 0;
```

Erstellen Sie ein Ablaufprotokoll der AM_1, indem Sie sie schrittweise ablaufen lassen, bis die Maschine terminiert. Die Anfangskonfiguration sei $(14, \varepsilon, 0 : 0 : 1, 3, 4, \varepsilon)$. Sie müssen nur Zellen ausfüllen, deren Wert sich im Vergleich zur letzten Zeile geändert hat.

AUFGABE AGS 15.18 – TEIL (B)

Lösung.

	BZ		DK		LZK		REF		Inp		Out	
(	14	,	ε	,	0:0:1	,	3	,	4	,	ε	)
(	15	,	ε	,	4:0:1	,	3	,	ε	,	ε	)
(	16	,	1	,	4:0:1	,	3	,	ε	,	ε	)
(	17	,	ε	,	4:0:1:1	,	3	,	ε	,	ε	)
(	3	,	ε	,	4:0:1:1:18:3	,	6	,	ε	,	ε	)
(	4	,	ε	,	4:0:1:1:18:3	,	6	,	ε	,	ε	)
(	5	,	4	,	4:0:1:1:18:3	,	6	,	ε	,	ε	)
(	6	,	2:4	,	4:0:1:1:18:3	,	6	,	ε	,	ε	)
(	7	,	1	,	4:0:1:1:18:3	,	6	,	ε	,	ε	)
(	8	,	ε	,	4:0:1:1:18:3	,	6	,	ε	,	ε	)
(	9	,	4	,	4:0:1:1:18:3	,	6	,	ε	,	ε	)
(	10	,	2:4	,	4:0:1:1:18:3	,	6	,	ε	,	ε	)
(	11	,	2	,	4:0:1:1:18:3	,	6	,	ε	,	ε	)
(	12	,	ε	,	2:0:1:1:18:3	,	6	,	ε	,	ε	)
(	18	,	ε	,	2:0:1	,	3	,	ε	,	ε	)
(	19	,	ε	,	2:0:1	,	3	,	ε	,	2	)
(	0	,	ε	,	2:0:1	,	3	,	ε	,	2	)