7. Code-Erzeugung

Eingabe: Programm in Zwischensprache

Aufgaben:

Speicherabbildung	Definitionsmodul erweitern
Code-Auswahl	Instruktionssequenzen erzeugen
Registerzuteilung	Verwendung von Registern für Zwischenergebnisse

Ausgabe: Maschinenprogramm als Datenstruktur

Entwurf der Code-Erzeugung:

- Maschineneigenschaften untersuchen
- Speicherabbildung entwerfen
- Code-Sequenzen zu allen Zwischensprachoperatoren aufstellen; ggf. Alternativen

Implementierung:

Speicherabbildung:

Programm und Definitionsmodul durchlaufen,

Adressen und Speicherumfang berechnen

Code-Auswahl:

Generatoren für Musterersetzung in Bäumen

Registerzuteilung:

Algorithmen für unterschiedlich großen Kontext:

Ausdrucksbäume, Grundblöcke, Ablaufgraphen

Vorlesung Übersetzer WS 97/98 / Folie 59

Ziele:

- Übersicht zu Entwurf und Algorithmen

in der Vorlesung:

- Aufgaben abgrenzen und zuordnen

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7

Speicherabbildung

Ziel: Zu Speicherobjekten des Programms bestimmen:

Speicherklasse, Relativadresse darin, Umfang

Implementierung:

als Eigenschaften im Definitionsmodul berechnen.

Entwurf:

1. Speicherklassen:

Code-Speicher

globale Daten

Laufzeitkeller

Prozedurschachteln (activation records); siehe U-61

Halde für dynamisch allokierte Daten, Speicherbereinigung

Register für

Adressierung der Speicherbereiche

Funktionsergebnisse, ggf. Parameter

lokale Variable, Zwischenergebnisse (Registerzuteilung)

2. Abbildung der Grundtypen

Maschineneigenschaften:

Wertdarstellung

Verarbeitungsumfang, Speicherumfang, Ausrichtung

verfügbare Operationen

3. Abbildung zusammengesetzter Typen

Arrays, Verbunde, Mengen, Klassen, Funktionen, ...

Vorlesung Übersetzer WS 97/98 / Folie 60

Ziele:

- Programmzustand auf Maschinenzustand abbilden

in der Vorlesung:

- Speicherklassen erläutern
- Beispiele für Typabbildung

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7.2

Prozedurschachteln

Prozedurschachtel (activation record) enthält lokale Daten für einen Prozeduraufruf und Verkettung auf dem Laufzeitkeller.
Entwurf der Schachtelstruktur zusammen mit Aufruf-Code

Schachtelstruktur		**Aufruf-Code**
	Aufrufstelle	Prozedur-Code
1. Parameter	Parameter kellern	
n-ter Parameter		
stat. Vorgänger (*)	stat. Vorg. kellern	
Rückkehradresse	Unterprogrammsprung	
dynamischer Vorgänger (**)		dyn. Vorg. kellern
		Schachtelreg := Kellerpegel
lokale Variable		Kellerpegel für lok. Var. erhöhen
gesicherte Register		Register sichern
		...
		Prozedurrumpf
		...
		Register zurückladen
		lok. Variable freigeben
		Schachtelreg. := entkellern
		Rücksprung
	stat. Vorg. entkellern (*)	
	Parameter entkellern	

(*) Nur bei Sprachen mit geschachtelten Prozeduren.
(**) Relativadresse 0 der Schachtel

Vorlesung Übersetzer WS 97/98 / Folie 61

Ziele:

- Zusammenhang zwischen Schachtelstruktur und Aufruf-Code erkennen

in der Vorlesung:

- Verzeigerung von Schachteln, Register sichern, Register-Windowing erläutern

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7.2.2, 7.3.1

Verständnisfragen:

- Zeigen Sie eine Schachtelstruktur für Prozessoren mit Register-Windowing

Code-Abbildung Ablaufstrukturen

Ablaufstrukturen in Code-Sequenzen mit Marken und Sprüngen umsetzen. Je nach Zwischensprache evtl. schon in der Transformationsphase.

Für jede Ablaufstruktur eine oder mehrere alternative Code-Sequenzen entwerfen, z. B.

```
while (B) S :   M1:  Code (B, false, M2)
                     Code (S)
                     goto M1
                M2:
```

oder

```
                     goto M2
                M1:  Code (S)
                M2:  Code (B, true, M1)
```

es bedeuten

Code (S): Code für `S` einsetzen

Code (B, true, M1): Code für Bedingung `B` so erzeugen, daß bei Ergebnis `true` auf die Marke `M1` gesprungen, sonst danach fortgesetzt wird

Vorlesung Übersetzer WS 97/98 / Folie 62

Ziele:

- Entwurfstechnik für Ablaufstrukturen erlernen

in der Vorlesung:

- weitere Beispiele dazu

nachlesen:

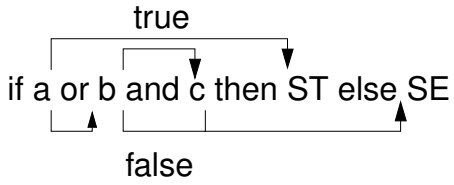
Kastens / Übersetzerbau, Abschnitt 7.3.2

Verständnisfragen:

- Warum ist die zweite while-Sequenz besser?

Kurzauswertung logischer Operatoren

Logische Ausdrücke in Sprungstrukturen übersetzen; falls Sprache erlaubt, nicht alle Operanden auszuwerten.



2 Code-Sequenzen für jeden Operator;Auswahl top-down:

Code (A and B, true, M):	Code (A, false, N) Code (B, true, M) N:
Code (A and B, false M):	Code (A, false, M) Code (B, false, M)
Code (A or B, true, M):	Code (A, true, M) Code (B, true M)
Code (A or B, false, M):	Code (A, true, N) Code (B, false, M) N:
Code (not A, X, M):	Code (A, not X, M)
Code (A < B, true, M):	Code (A, Ri); Code (B, Rj) cmp Ri, Rj bralt M
Code (A < B, false, M):	Code (A, Ri); Code (B, Rj) cmp Rj, Ri bralt M
Code für Blätter:	bedingter Sprung

Vorlesung Übersetzer WS 97/98 / Folie 63

Ziele:

- Spezielle Technik zur kontext-abhängigen Code-Auswahl kennenlernen

in der Vorlesung:

- Beispiel am Baum zeigen

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7.3.3

Verständnisfragen:

- Wie kann man die Technik durch eine AG beschreiben?
- Warum wird für den Operator *not* keine Instruktion erzeugt?
- Begründen Sie, ob die Technik für Pascal und C angewandt werden kann oder muß.

Code-Auswahl für Ausdrücke

- Benachbarte Operationen können evtl. zu einer Instruktion zusammengefaßt werden
- verschiedene Alternativen möglich
- Wertdeskriptoren zu Baumknoten beschreiben Unterbringung des Zwischenergebnisses, z. B.

R_i	Wert im Register R_i
R_i,c	Adresse gebildet aus Inhalt R_i + Konstante c
c	konstanter Wert c
(...)	Inhalt der in Klammern angegebenen Adresse

Übersetzungsmuster für Baumknoten, z. B.

addr = Variablenadresse, **cont** = Inhaltsoperation,
const = Konstanter Wert, **addradd** = Adresse + Wert

Operator	**Wertdeskriptoren**			**Code**
	Operand(en)		**Ergebnis**	
addr	R_i, c		-> R_i,c	./.
const	c		-> c	./.
const	c		-> R_i	load c, R_i
cont	R_i, c		-> (R_i, c)	./.
cont	R_i		-> (R_i)	./.
cont	R_i, c		-> R_j	load (R_i, c), R_j
cont	R_i		-> R_j	load (R_i), R_j
addradd	R_i	c	-> R_i, c	./.
addradd	R_i, c1	c2	-> R_i, c1 + c2	./.
addradd	R_i	R_j	-> R_k	add Ri, R_j, R_k
addradd	R_i, c	R_j	-> R_k, c	add R_i, R_j, R_k

Aufgabe: Baum mit Mustern passend überdecken, so daß möglichst wenige Instruktionen erzeugt werden.

Vorlesung Übersetzer WS 97/98 / Folie 64

Ziele:

- Wertdeskriptor-Signaturen beschreiben Code-Sequenzen für Ausdrücke

in der Vorlesung:

- Beispiel U-65 erläutern
- Vergleich: überladene Operatoren

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7.3.4

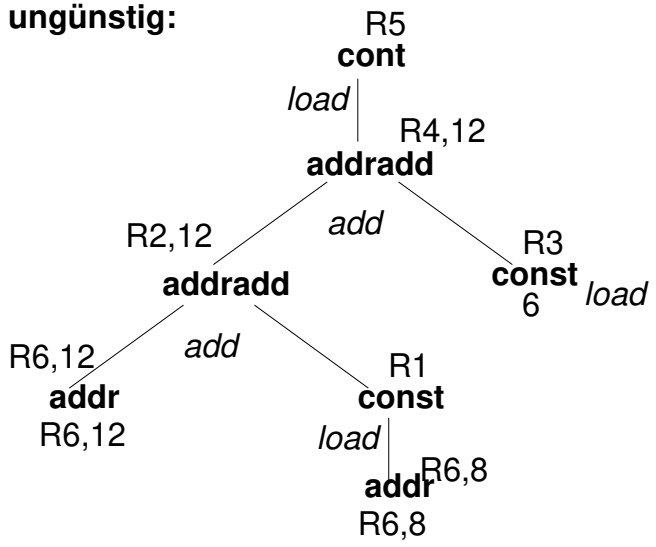
Verständnisfragen:

- Vergleichen Sie die Technik mit der Identifikation überladener Operatoren.

Beispiel Code-Auswahl

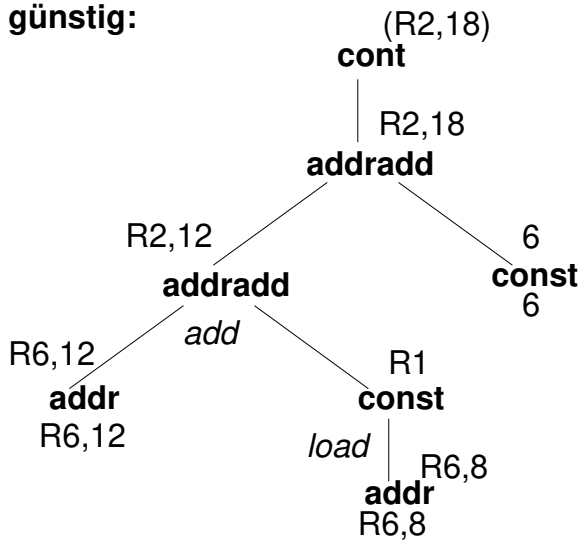
Baum zu `... := a[i].s`

ungünstig:



load (R6,8), R1
add R6,R1,R2
load 6,R3
add R2,R3,R4
load (R4,12),R5
store R5, ...

günstig:



load (R6,8), R1
add R6,R1,R2
store (R2,18),...

Vorlesung Übersetzer WS 97/98 / Folie 65

Ziele:

- Beispiel zu U-64
- Auswahl der Code-Muster nach Instruktionskosten

in der Vorlesung:

- Code-Muster den Baumknoten zuordnen

Musterersetzung in Bäumen

Methode: Baum bottom-up durchlaufen, anwendbare Muster auswählen, Entscheidung nach oben verzögern (vergl. dynamische Programmierung)

Spezifikation der Baummuster durch KFG-Produktionen plus Code; Wertdeskriptoren als Nichtterminale, z. B.

RegConst ::= addradd Reg Const

RegConst ::= addradd RegConst Reg add R_i, R_j, R_k

auch tiefere Baummuster, z. B.

Void ::= assign RegConst addradd Reg Const

store (R_i, c1),(R_j, c2)

Kosten für Auswahl: Anzahl Instruktionen

Methode „Graham, Glenville“, Generatoren : GG, GGSS:

Baum in Präfix-Form,

Parsing mit mehrdeutiger KFG

Entscheidung nach Kosten

Methode „Bottom-up Rewrite“, Generator BURG:

Muster zur Generierungszeit analysieren

Baumautomat generieren

Auswahlentscheidungen werden als Zustände von Knoten codiert

Vorlesung Übersetzer WS 97/98 / Folie 66

Ziele:

- Systematik zur Code-Auswahl für Ausdrucksbäume

in der Vorlesung:

- Methoden am Beispiel erläutern

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7.4.3

Verständnisfragen:

- Begründen Sie: BURS-Methode liefert bessere Ergebnisse als Parsing-Methode, weil Entscheidungen später und globaler getroffen werden.

Registerzuteilung

Registerverwendungen:

Zwischenergebnis in Ausdrücken

mehrfach benutzte Ausdrucksergebnisse

Variable

Prozedurparameter

Funktionsergebnis

Kellerpegel, Schachtelzeiger, Haldenzeiger, ...

Anzahl der Register (in jeder Klasse) ist beschränkt

gute Registernutzung reduziert

- Zahl der Speicherzugriffe
- Code zum Zwischenspeichern (spill code)

verschiedene Zuteilungsverfahren für unterschiedlich großen Kontext:

- Ausdrucksbäume (Sethi, Ullman)
- Grundblöcke (Belady)
- Ablaufstrukturen (Graphfärbung)

nützliche Technik: Verschieben der Registerzuteilung in spätere Phase durch Zuteilung **symbolischer Register** aus unbegrenztem Vorrat

Vorlesung Übersetzer WS 97/98 / Folie 67

Ziele:

- Übersicht zur Aufgabe Registerzuteilung

in der Vorlesung:

- Registerverwendungen erläutern

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7.5

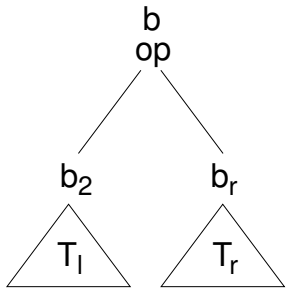
Registerzuteilung für Ausdrucksbäume

Voraussetzung für Optimalität: [Sethi, Ullman '70]

optimaler Code kann „baumartig“ zusammengesetzt werden

(nicht bei mehreren Registerklassen oder Registerpaaren)

Prinzip: bei 2-stelligen Operatoren erst den Teilbaum mit höherem Registerbedarf auswerten



Ann.: Erg. v. T_l und T_r in Registern

Reihenfolge	Reg.bedarf b =
T_l T_r op	$\max(b_l, b_r + 1)$
T_r T_l op	$\max(b_r, b_l + 1)$

b minimieren

falls b > regmax (Anzahl verfügbarer Register):

dann gilt $b_l = b_r$ = regmax;

Spill-Code: T_r; **store R_r, h;** T_l; **load h, R_r**; op R_r, R_l

für T_l sind alle Register frei

load h, R_r entfällt, wenn

Speicheroperand h in op h, R_l möglich ist.

Implementierung z. B. durch Attributierung:

1. Phase: Bedarf, Ausw.reihenf. = synth. Attribute (aufwärts)
2. Phase: Registerzuteilung = inh. Attribute (abwärts)
3. Phase: Code-Erz. in Auswertungsreihenfolge (aufwärts)

Vorlesung Übersetzer WS 97/98 / Folie 68

Ziele:

- Zusammenhang zwischen Registerbedarf und Auswertungsreihenfolge erkennen

in der Vorlesung:

- am Beispiel zeigen; Grenzen des Verfahrens

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7.5.3

Übungsaufgaben:

- Verfahren für mehrere Registerklassen anwenden (Aufgabe 30)

Verständnisfragen:

- In einem Ausdrucksbaum werden an 2 Knoten Spill-Code erzeugt. Wo liegen diese Knoten?
- Beschreiben Sie das Verfahren durch eine AG.

Grundblöcke in 2 Durchgängen (Belady)

1. Durchgang:

Bestimmung der **Lebensdauer** der Werte, d.h. von Berechnung bis letzter Benutzung (mehrfache Benutzung ist möglich!).

Lebensdauer dargestellt als Intervallgraph

Schnitt im Graph = Anzahl belegter Register

maximaler Schnitt im Graph = Registerbedarf des Blockes

Register im Graph zuordnen

Bei Registermangel Werte zum Zwischenspeichern auswählen, Kriterien:

- Wert, der schon im Speicher steht
- Wert, der am spätesten wiederverwendet wird

2. Durchgang:

Register im Code zuteilen

(Ausführungsreihenfolge wird nicht verändert.)

Verfahren ursprünglich als **Seitentauschverfahren** für Speicherverwaltungen von **Belady** 1966 vorgestellt.

Vorlesung Übersetzer WS 97/98 / Folie 69

Ziele:

- Lebensdauer und Registerbedarf durch Intervallgraph modellieren

in der Vorlesung:

- Verfahren am Beispiel U-70 erläutern

nachlesen:

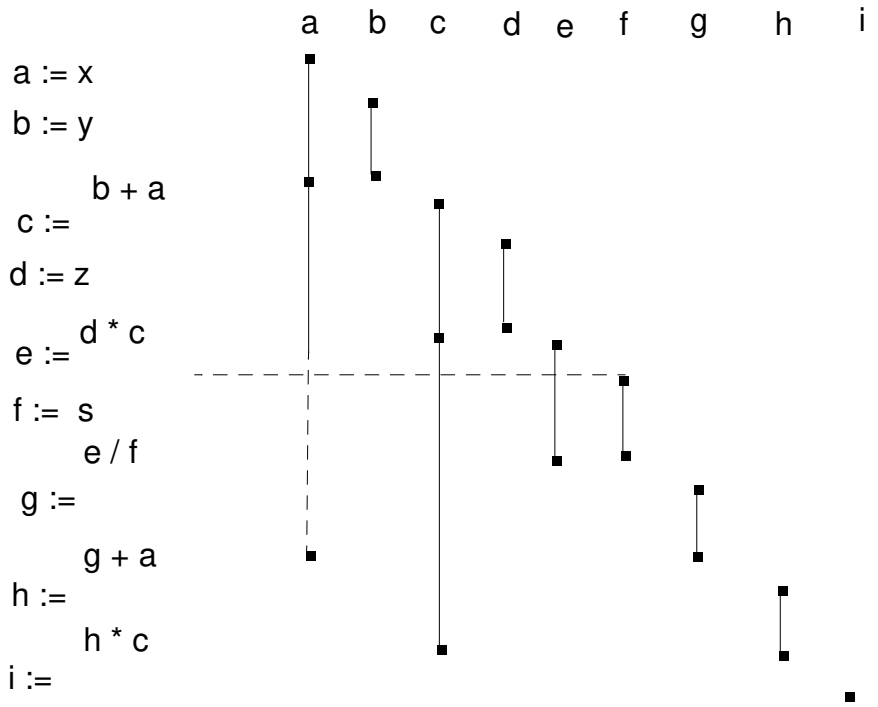
Kastens / Übersetzerbau, Abschnitt 7.5.2

Verständnisfragen:

- Begründen Sie die Auswahlkriterien bei Registermangel.
- Zeigen Sie die Entsprechungen zum Seitentauschverfahren.

Beispiel für Belady-Verfahren

Lebensdauer der Werte eines Grundblockes



Registerzuteilungen:

		a	b	c	d	e	f	g	h	i
4 Reg:	(a)	d1	d2	d2	d3	d3	d4	d3	d3	d3
3 Reg:	(b)			*			d2			
3 Reg:	(c)	*					d1			

* zwischenspeichern

Vorlesung Übersetzer WS 97/98 / Folie 70

Ziele:

- Beispiel für U-69

in der Vorlesung:

- Beispiel erläutern

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7.5.2, Abb. 7.5-3

Übungsaufgaben:

- Verfahren an einem Beispiel anwenden (Aufgabe 31)

Verständnisfragen:

- Begründen Sie die Alternativen (b) und (c).

Registerzuteilung durch Graphfärbung

Anwendung auf **Ablaufstrukturen**

Berechnung und Benutzung der Werte in verschiedenen Grundblöcken.

Zunächst symbolische Register zuteilen; dann

Unverträglichkeitsgraph aufstellen:

Knoten: symbolische Register

Kante {a, b}: Lebensdauer von a und b überlappt

Lebensdauern werden mit Datenflußanalyse ermittelt.

„Färbung“ mit realen Registernummern:

Kante {a, b} im Graph:

a und b in verschiedenen Registern

Graphfärbung NP-vollständig

heuristisches Verfahren zur Färbung mit k Farben (Registern):

sukzessive Knoten mit Grad < k streichen

falls am Ende Graph leer:

Graph ist k-färbbar

Knoten in umgekehrter Reihenfolge färben

sonst

Graph ist so nicht k-färbbar

Knoten zum Zwischenspeichern auswählen (spill code)

Verfahren erneut anwenden

Vorlesung Übersetzer WS 97/98 / Folie 71

Ziele:

- Überlappende Lebensdauern durch Unverträglichkeitsgraph modellieren
- allgemeines Prinzip

in der Vorlesung:

- Modellierung am Beispiel U-72 erläutern
- Heuristisches Verfahren zeigen

nachlesen:

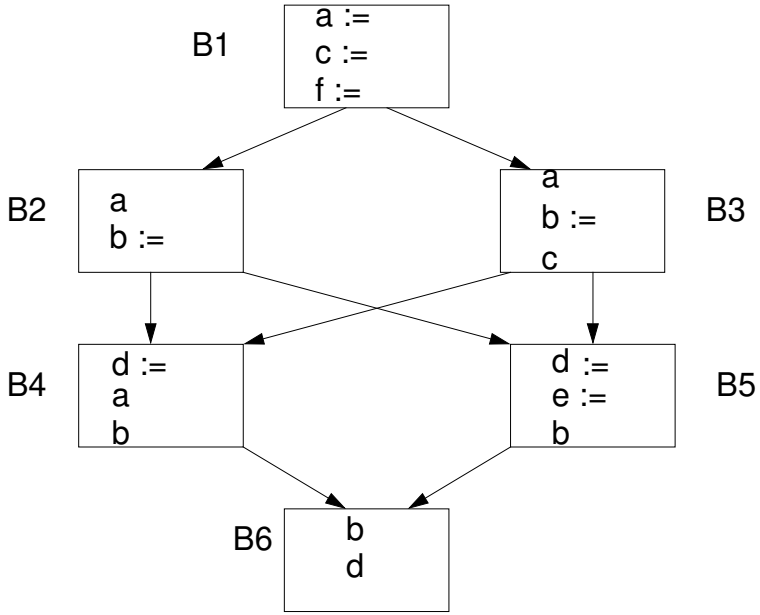
Kastens / Übersetzerbau, Abschnitt 7.5.4

Verständnisfragen:

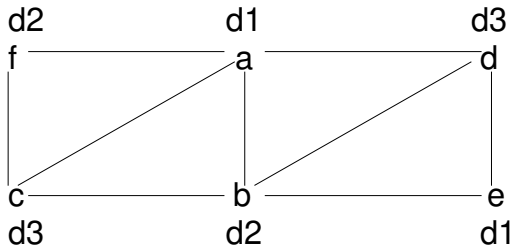
- Warum benötigt man DFA um die überlappenden Lebensdauern festzustellen? Man könnte doch jeden Block isoliert untersuchen. Zeigen Sie ein Beispiel, in dem dies falsche Ergebnisse liefert.
- Geben Sie einen k-färbbaren Graphen an, für den die Heuristik versagt.

Beispiel: Graphfärbung

Ablaufgraph mit Benutzung von Werten



Unverträglichkeitsgraph



Vorlesung Übersetzer WS 97/98 / Folie 72

Ziele:

- Beispiel für U-71

in der Vorlesung:

- Beispiel erläutern

nachlesen:

Kastens / Übersetzerbau, Abschnitt 7.5.4, Abb. 7.5-6

Übungsaufgaben:

- Verfahren an einem Beispiel anwenden