

SS 2007
Informatik 2 - SECD

Dozent:
Michael Sperber

Skriptfassung in L^AT_EX von
Rouven Walter

Lizenz

Dieses Werk ist unter der folgenden Creative Commons-Lizenz lizenziert:
Namensnennung-NichtKommerziell-Weitergabe unter gleichen Bedingungen 2.0 Deutschland

Die vollständigen Lizenzbestimmungen sind einzusehen unter:

<http://creativecommons.org/licenses/by-nc-sa/2.0/de/>



Vorwort

Dieser Mitschrieb entstand während meiner Nachbearbeitung zur Informatik II Vorlesung im Sommersemester 2007 bei Prof. Dr. H. Klaeren an der Eberhards-Karl-Universität Tübingen. Ich erhebe keinen keinen Anspruch auf Vollständigkeit oder Richtigkeit. Bei Unklarheiten zum Inhalt empfehle ich daher, sich an die jeweiligen Dozenten/Tutoren zu wenden.

Über Verbesserungsvorschläge oder sonstige Anregungen, würde ich mich über eine Email an *RouvenWalter (AT) web.de* freuen.

Inhaltsverzeichnis

1	Auswertungsmodelle für Java-artige Programmiersprachen	4
1.1	λ -Kalkül	4
1.2	ISWIM: Substrat für andere Programmiersprachen	4
1.3	Definition: Schwache Kopfnormalform	4
1.4	Definition: Call-by-Value-Reduktion	4
1.5	Satz: Church/Rosser für ISWIM	5
1.6	Satz: Diamant	5
1.7	Satz: $eval_v$ ist eine partielle Funktion	5
1.8	Satz	5
1.9	Definition: Standard-Reduktion	5
1.10	Satz	6
1.11	Satz: $eval_v = eval_v^s$	6
1.12	\mathcal{L}_{ISWIM} übersetzen in SECD Instruktionen	6
1.13	Beispiel: Übersetzung $e \in \mathcal{L}_{ISWIM} \mapsto \hat{C}$	7
1.14	Ausführungsrelation für eine SECD Instruktion	7
2	Endrekursion	8
2.1	Beobachtung	8
2.2	\mathcal{L}_{ISWIM} übersetzen in SECD Instruktionen (unter Berücksichtigung der Endrekursion)	8
3	Speichermodell für die SECD Maschine	9
3.1	Einführung von Speicheradressen und dem Heap	9
3.2	Ausführungsrelationen für SECDH Instruktionen	9
3.3	Beobachtungen	10
3.4	Automatische Speicherverwaltung: Garbage Collection	10
3.5	Einfachste Lösungstechnik: Fixpunktiteration	10

1 Auswertungsmodelle für Java-artige Programmiersprachen

1.1 λ -Kalkül

Sprache \mathcal{L}_λ . V abzählbar, Variablen.

\mathcal{L}_λ . $V \subseteq \mathcal{L}_\lambda$, $\underbrace{(e_0 e_1)}_{\text{Applikation}} \in \mathcal{L}_\lambda$, $e_0 \in \mathcal{L}_\lambda$, $e_1 \in \mathcal{L}_\lambda$.

Abstraktion: $(\lambda x. e) \in \mathcal{L}_\lambda$, $x \in V$, $e \in \mathcal{L}_\lambda$. λ -Kalkül: Reduktionskalkül

β -Reduktion:

$(\lambda x. \underbrace{e}_{\text{Rumpf}}) f \xrightarrow{\beta} e \underbrace{[x \mapsto f]}_{\text{Substitution}} \rightarrow_{\beta}$ Erweiterung auf Subterme

λ -Kalkül kann abbilden:

- boolesche Werte und Verzweigungen
- Zahlen
- Rekursion

1.2 ISWIM: Substrat für andere Programmiersprachen

(The next 700 programming languages, P. J. Landin)

Funktionales ISWIM.

$\mathcal{L}_{ISWIM} = \mathcal{L}_\lambda \cup B \cup O$

$b \in B$: Basiswerte, $B = \mathbb{N}$.

$o^n \in \Sigma$.

$e = (o^n e_1 \dots e_n)$, Operatorenanwendung.

$(+ \ulcorner m \urcorner \ulcorner n \urcorner)$, wobei $\ulcorner m \urcorner$ Darstellung der Zahl $m \in \mathbb{N}$ in \mathcal{L}_{ISWIM} , $+^{(2)} \in \Sigma$.

$\beta \rightsquigarrow \beta_v$ Call-by-Value-Kalkül.

$(+ \ulcorner m \urcorner \ulcorner n \urcorner) \rightarrow_{\sigma} \ulcorner m + n \urcorner$

1.3 Definition: Schwache Kopfnormalform

Unter den λ -Termen heißen die Abstraktionen auch *Werte* oder *schwache Kopfnormalform*.

\rightsquigarrow In ISWIM sind auch die $b \in B$ Werte.

1.4 Definition: Call-by-Value-Reduktion

$(\lambda x. e) w \xrightarrow{\beta_v} e[x \mapsto w]$, w Wert.

\rightarrow_{β_v} ist die Erweiterung auf Subterme (wenn Gesamtterm noch nicht in SKNF), die möglichst weit links innen stehen.

$V = \beta_v \cup \sigma$

\rightarrow_v Erweiterung auf beliebige Subterme
 \rightarrow_v^* transitive-reflexive-Hülle
 $=_v$ induzierte Äquivalenzrelation

$$eval_v(e) = \begin{cases} b, & \text{falls } e =_v b, b \in B \\ function, & \text{falls } e =_v \lambda x. e' \end{cases} \quad \text{Auswertungsfunktion.}$$

1.5 Satz: Church/Rosser für ISWIM

Falls $e =_v g$, dann gibt es Term h mit $e \rightarrow_v^* h, g \rightarrow_v^* h$

1.6 Satz: Diamant

Falls $h \rightarrow_v^* e, h \rightarrow_v^* g$, dann gibt es Term l mit $e \rightarrow_v^* l, g \rightarrow_v^* l$

1.7 Satz: $eval_v$ ist eine partielle Funktion

(Resultat, für Terme mit Resultat, ist eindeutig bestimmt)

$eval_v : \mathcal{L}_{ISWIM} \rightarrow B \cup \{function\}$ funktioniert auf Teilmenge.

„Term mit Loch“.

$$E = \underbrace{[]}_{\text{Loch}} |(w E)|(E w)|(o^n w \dots w E w \dots w)$$

$E[e]$ in E für $[]$ e einsetzen.

Beispiel:

$$\underbrace{((\lambda x. \lambda y. y)5)7}_{\text{Redex}}$$

$([]7)$ Kontext

1.8 Satz

Für alle e gibt es entweder $e = w$, w Wert, oder es gibt einen eindeutig bestimmten Auswertungskontext E mit $e = E[(w_1 w_2)]$ oder $e = E[(o^n w \dots w_n)]$

1.9 Definition: Standard-Reduktion

$$E[e] \mapsto_v E[e'], \text{ falls } e \rightarrow_v e'$$

1.10 Satz

$e \rightarrow_v^* g \Leftrightarrow e \mapsto_v^*$ für Wert w .

$w \rightarrow_v^* g$, insbesondere, falls $w \in B$, dann $e \rightarrow_v^* g \Leftrightarrow e \mapsto_v^* g$.

$$eval_v^s(e) = \begin{cases} b, & \text{falls } e \mapsto_v^* g \\ function, & \text{falls } e \mapsto_v^* \lambda x. e' \end{cases}$$

1.11 Satz: $eval_v = eval_v^s$

1.12 \mathcal{L}_{ISWIM} übersetzen in SECD Instruktionen

\rightsquigarrow maschinelle Auswertung

Pentium 4?

JVM Java Virtual Maschine

SECD-Maschine

\mathcal{L}_{ISWIM} wird übersetzt in:

$$\begin{aligned} \hat{C} &= \varepsilon \\ &| b \hat{C} \quad b \in B \\ &| x \hat{C} \quad x \in V \\ &| ap \hat{C} \\ &| prim_{o^n} \quad o^n \in \Sigma \\ &| \langle x, \hat{C}' \rangle \hat{C} \end{aligned}$$

Übersetzung $e \in \mathcal{L}_{ISWIM} \mapsto \hat{C}$

$$\begin{aligned} \llbracket b \rrbracket_{SECD} &= b \\ \llbracket x \rrbracket_{SECD} &= x \\ \llbracket (e_0 e_1) \rrbracket_{SECD} &= \llbracket e_0 \rrbracket_{SECD} \llbracket e_1 \rrbracket_{SECD} ap \\ \llbracket (o^n e_1 \dots e_n) \rrbracket_{SECD} &= \llbracket e_1 \rrbracket_{SECD} \dots \llbracket e_n \rrbracket_{SECD} prim_{o^n} \\ \llbracket (\lambda x. e) \rrbracket_{SECD} &= \langle x, \llbracket e \rrbracket_{SECD} \rangle \end{aligned}$$

1.13 Beispiel: Übersetzung $e \in \mathcal{L}_{ISWIM} \mapsto \hat{C}$

$$\begin{aligned}
 & \llbracket (\lambda x. \lambda y. (+ x y)) \rrbracket \\
 = & \langle x, \llbracket \lambda y. (+ x y) \rrbracket \rangle \\
 = & \langle x, \langle y, \llbracket (+ x y) \rrbracket \rangle \rangle \\
 = & \langle x, \langle y, \llbracket x \rrbracket \llbracket y \rrbracket \text{ prim}_+ \rangle \rangle \\
 = & \langle x, \langle y, x y \text{ prim}_+ \rangle \rangle
 \end{aligned}$$

1.14 Ausführungsrelation für eine SECD Instruktion

$$\begin{aligned}
 \underbrace{\hat{S}}_{\text{Stapel/Stack}} &= \varepsilon \mid \underbrace{\hat{w}}_{\text{Werte}} S \\
 \underbrace{\hat{E}}_{\text{Environment/Umgebung}} &= V \rightarrow \hat{w} \\
 \underbrace{\hat{D}}_{\text{Dump}} &= \varepsilon \mid \langle \hat{S}, \hat{E}, \hat{C}, \hat{D} \rangle \\
 \hat{w} &= b \mid \langle \langle x, \hat{C} \rangle, \underbrace{\hat{E}}_{\text{Closure}} \rangle
 \end{aligned}$$

Ausführungsrelation:

$$\begin{aligned}
 & \langle \hat{S}, \hat{E}, \hat{C}, \hat{D} \rangle \\
 \mapsto_{SECD} & \langle \hat{S}', \hat{E}', \hat{C}', \hat{D}' \rangle \\
 & \langle \hat{S}, \hat{E}, x\hat{C}, \hat{D} \rangle \\
 \mapsto_{SECD} & \langle w\hat{S}, \hat{E}, \hat{C}, \hat{D} \rangle, w = \hat{E}(x) \\
 & \langle b_n \dots b_1 \hat{S}, \hat{E}, \text{prim}_{o^n} \hat{C}, \hat{D} \rangle \\
 \mapsto_{SECD} & \langle b\hat{S}, \hat{E}, \hat{C}, \hat{D} \rangle, \text{wobei } (o^n b_1 \dots b_n) \rightarrow_\sigma b \\
 & \langle \hat{S}, \hat{E}, \langle x, \hat{C}' \rangle \hat{C}, \hat{D} \rangle \\
 \mapsto_{SECD} & \langle \langle \langle x, \hat{C}' \rangle, \hat{E} \rangle \hat{S}, \hat{E}, \hat{C}, \hat{D} \rangle \\
 & \langle w \langle \langle x, \hat{C}' \rangle, \hat{E}' \rangle \hat{S}, \hat{E}, \text{ap} \hat{C}, \hat{D} \rangle \\
 \mapsto_{SECD} & \langle \varepsilon, \hat{E}'[x \mapsto w], \hat{C}', \langle \hat{S}, \hat{E}, \hat{C}, \hat{D} \rangle \rangle \\
 & \langle w, \hat{E}, \varepsilon, \langle \hat{S}', \hat{E}', \hat{C}', \hat{D}' \rangle \rangle \\
 \mapsto_{SECD} & \langle w\hat{S}', \hat{E}', \hat{C}', \hat{D}' \rangle
 \end{aligned}$$

$e \in \mathcal{L}_{ISWIM}$ auswerten:

$$\begin{aligned}
 \langle \varepsilon, \emptyset, \llbracket e \rrbracket, \varepsilon \rangle & \mapsto_{SECD} \langle \underbrace{w}_{\text{Ergebnis}}, \hat{E}, \varepsilon, \varepsilon \rangle \\
 & \rightsquigarrow \text{eval}_{SECD} = \text{eval}_v
 \end{aligned}$$

2 Endrekursion

2.1 Beobachtung

$\langle \hat{V} \langle \langle x, \hat{C}' \rangle, \hat{E}', ap\hat{C}, \hat{D} \rangle \rangle$
 $\mapsto \langle \varepsilon, \hat{E}'[x \mapsto w], \hat{C}', \langle \hat{S}, \hat{E}, \hat{C}, \hat{D} \rangle \rangle$
 (Funktionsapplikation)

Beobachtung:

\hat{D} -Komponente wird bei jeder Funktionsapplikation größer, auch bei endrekursiven Aufrufen.
 \Rightarrow bei Maschinen mit endlichem Speicher reicht die Regel nicht aus, um Endrekursion oder Schleifen zu erklären.

2.2 \mathcal{L}_{ISWIM} übersetzen in SECD Instruktionen (unter Berücksichtigung der Endrekursion)

$$\begin{aligned}
 \llbracket b \rrbracket &= b \\
 \llbracket x \rrbracket &= x \\
 \llbracket (e_0 e_1) \rrbracket &= \llbracket e_0 \rrbracket \llbracket e_1 \rrbracket ap \quad (\text{nicht endrekursiver Kontext}) \\
 \llbracket (o^n e_1 \dots e_n) \rrbracket &= \llbracket e_1 \rrbracket \dots \llbracket e_n \rrbracket prim_{o^n} \\
 \llbracket (\lambda x. e) \rrbracket &= \langle x, \llbracket e \rrbracket' \rangle
 \end{aligned}$$

endrekursive Aufrufe: Aufrufe ohne Kontext

endrekursiver Kontext:

$$\begin{aligned}
 \llbracket b \rrbracket' &= b \\
 \llbracket x \rrbracket' &= x \\
 \llbracket (e_0 e_1) \rrbracket' &= \llbracket e_0 \rrbracket \llbracket e_1 \rrbracket tailap \quad \text{wobei } \hat{C} = \dots | tailap \hat{C} \\
 \llbracket (o^n e_1 \dots e_n) \rrbracket' &= \llbracket e_1 \rrbracket \dots \llbracket e_n \rrbracket prim_{o^n} \\
 \llbracket (\lambda x. e) \rrbracket' &= \langle x, \llbracket e \rrbracket' \rangle
 \end{aligned}$$

$\langle \hat{V}\hat{S}, \hat{E}, \varepsilon, \langle \hat{S}', \hat{E}', \hat{C}', \hat{D} \rangle \rangle$
 $\mapsto \langle \hat{V}\hat{S}', \hat{E}', \hat{C}', \hat{D} \rangle$
 $\langle \hat{V} \langle \langle x, \hat{C}' \rangle, \hat{E}' \rangle \hat{S}, \hat{E}, tailap \hat{C}, \hat{D} \rangle$
 $\mapsto \langle \hat{S}, \hat{E}'[x \mapsto \hat{V}], \hat{C}', \hat{D} \rangle$

3 Speichermodell für die SECD Maschine

3.1 Einführung von Speicheradressen und dem Heap

(Speichermodell in Java ist exakt das gleiche wie in Scheme)

$\mathcal{L}_{ISWIM} = \dots$

$e = \underbrace{\dots}_{\text{wie gehabt}} \mid (:= x e) \quad (\text{in Java: } x = e;)$

$\hat{C} = \underbrace{\dots}_{\text{wie gehabt}} \mid :=_x$

$\llbracket (:= x e) \rrbracket = \llbracket e \rrbracket :=_x$

Vorher: Umgebung bildet Variablen auf Werte ab

Nachher: Umgebung bildet Variablen auf Speicheradressen

Zusätzlich: Heap bildet Speicheradressen auf Werte ab

$\hat{E} = V \mapsto \underbrace{\hat{A}}_{\text{Adresse}}$

$\hat{H} = \hat{A} \mapsto \hat{V} \mid \text{free}$

$\hat{V} = b \mid \langle \langle x, \hat{C} \rangle, \hat{E} \rangle \mid \text{void} \quad \text{wobei void Wert von Zuweisungsausdruck}$

3.2 Ausführungsrelationen für SECDH Instruktionen

$\langle \hat{S}, \hat{E}, b\hat{C}, \hat{D}, \hat{H} \rangle$

$\mapsto \langle b\hat{S}, \hat{E}, \hat{C}, \hat{D}, \hat{H} \rangle$

$\langle \hat{S}, \hat{E}, x\hat{C}, \hat{D}, \hat{H} \rangle$

$\mapsto \langle \hat{V}\hat{S}, \hat{E}, \hat{C}, \hat{D}, \hat{H} \rangle \quad \text{wobei } \hat{V} = \hat{H}(\hat{E}(x))$

$\langle b_n \dots b_1 \hat{S}, \hat{E}, \text{prim}_{o^n} \hat{C}, \hat{D}, \hat{H} \rangle$

$\mapsto \langle \hat{V}\hat{S}, \hat{E}, \hat{C}, \hat{D}, \hat{H} \rangle \quad \text{wobei } (o^n b_1 \dots b_n) \rightarrow_{\sigma} \hat{V}$

$\langle \hat{S}, \hat{E}, \langle x, \hat{C}' \rangle \hat{C}, \hat{D}, \hat{H} \rangle$

$\mapsto \langle \langle \langle x, \hat{C}' \rangle, \hat{E} \rangle \hat{S}, \hat{E}, \hat{C}, \hat{D}, \hat{H} \rangle \quad \text{Speicher wird nicht eingepackt}$

$\langle \hat{V} \langle \langle x, \hat{C}' \rangle, \hat{E}' \rangle \hat{S}, \hat{E}, \text{ap}\hat{C}, \hat{D}, \hat{H} \rangle$

$\mapsto \langle \varepsilon, \hat{E}'[x \mapsto \hat{A}], \hat{C}', \langle \hat{S}, \hat{E}, \hat{C}, \hat{D} \rangle, \hat{H}[\hat{A} \mapsto \hat{V}] \rangle \quad \text{wobei } \hat{H}(\hat{A}) = \text{free}$

$\langle \hat{V}, \hat{E}, \varepsilon, \langle \hat{S}', \hat{E}', \hat{C}', \hat{D} \rangle, \hat{H} \rangle$

$\mapsto \langle \hat{V}\hat{S}', \hat{E}', \hat{C}', \hat{D}, \hat{H} \rangle$

$\langle \hat{V}\hat{S}, \hat{E}, :=_x \hat{C}, \hat{D}, \hat{H} \rangle$

$\mapsto \langle \text{void}\hat{S}, \hat{E}, \hat{C}, \hat{D}, \hat{H}[\hat{E}(x) \mapsto \hat{V}] \rangle$

3.3 Beobachtungen

- Es gibt nur einen Heap
- linear gefädelt
- Heap wächst monoton

3.4 Automatische Speicherverwaltung: Garbage Collection

Schritt 1:

Ermittlung „lebendiger“ Speicheradressen, auf die das Programm noch zugreifen kann.

$$\begin{aligned}
 live_E(\hat{E}) &= \{\hat{A} \mid (x, \hat{A}) \in \hat{E}\} \\
 live_S(\varepsilon) &= \emptyset \quad \text{wobei } \varepsilon : \text{leerer Stack} \\
 live_S(b\hat{S}) &= live_S(\hat{S}) \\
 live_S(void\hat{S}) &= live_S(\hat{S}) \\
 live_S(\langle\langle x, \hat{C} \rangle, \hat{E} \rangle\hat{S}) &= live_E(\hat{E}) \cup live_S(\hat{S}) \\
 live_H(\Sigma, \hat{H}) &= \Sigma \cup \bigcup \{live_E(\hat{E}) \mid \hat{H}(\hat{A}) = \langle\langle x, \hat{C} \rangle, \hat{E} \rangle, \hat{A} \in live_H(\Sigma, \hat{H})\}
 \end{aligned}$$

wobei Σ Menge von Adressen, auf die von außen zugegriffen werden kann

Kleinste Lösung:

$$X = F(X)$$

$live_H(\Sigma, \hat{H})$ ist die kleinste Menge, die die obige Gleichung erfüllt.

$$\begin{aligned}
 \langle \hat{S}, \hat{E}, \hat{C}, \hat{D}, \hat{H}[\hat{A} \mapsto \hat{V}] \rangle \quad &\text{mit } \hat{A} \notin live_H(live_S(\hat{S}) \cup live_E(\hat{E}) \cup live_D(\hat{D}), \hat{H}) \\
 \mapsto \langle \hat{S}, \hat{E}, \hat{C}, \hat{D}, \hat{H} \rangle
 \end{aligned}$$

\rightsquigarrow Nichtdeterminismus

Garbage Collection

3.5 Einfachste Lösungstechnik: Fixpunktiteration

$$\begin{aligned}
 live_H(\Sigma, \hat{H}) &= \Sigma \cup F(live_H(\Sigma, \hat{H})) \\
 live_H^0(\Sigma, \hat{H}) &:= \Sigma \\
 live_H^{n+1}(\Sigma, \hat{H}) &:= \Sigma \cup F(live_H^n(\Sigma, \hat{H})) \\
 live_H(\Sigma, \hat{H}) &:= \bigcup_{i=0}^{\infty} live_H^i(\Sigma, \hat{H})
 \end{aligned}$$