SS 2007

Informatik 2 - SECD

Dozent:

Michael Sperber

Skriptfassung in LaTeX von

Rouven Walter

Lizenz

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

1.1 λ-Kalkül

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

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

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

λ-Kalkül kann abbilden:

- boolsche 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, $B = \mathbb{N}$.
$o^n \in \Sigma$.
$e = (o^n\ e_1 \ldots e_n)$, 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.
$(+\ \ulcorner m \urcorner\ \ulcorner n \urcorner) \quad \rightarrow_\sigma \quad \ulcorner m + n \urcorner$

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

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

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

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

$E[e]$ in E für [ ] e einsetzen.

Beispiel:

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

([ ]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 $e = E[(o^n\ w \ldots w_n)]$

1.9 Definition: Standard-Reduktion

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

1.10 Satz

$e \to_v^* g \quad \Leftrightarrow \quad e \mapsto_v^*$ für Wert w.
$w \to_v^* g$, insbesondere, falls $w \in B$, dann $e \to_v^* g \quad \Leftrightarrow \quad e \mapsto_v^* g$.

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

1.11 Satz: $eval_v = eval_v^s$

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

$\leadsto$ maschinelle Auswertung
Pentium 4?
JVM Java Virtual Mashine
SECD-Maschine

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

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

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

$$\begin{array}{rcl} [\![b]\!]_{SECD} & = & b \\ [\![x]\!]_{SECD} & = & x \\ [\![(e_0\ e_1)]\!]_{SECD} & = & [\![e_0]\!]_{SECD}\ [\![e_1]\!]_{SECD}\ ap \\ [\![(o^n\ e_1 \ldots e_n]\!]_{SECD} & = & [\![e_1]\!]_{SECD}\ \cdots\ [\![e_n]\!]_{SECD}\ prim_{o^n} \\ [\![(\lambda x.\, e)]\!]_{SECD} & = & \langle x,\ [\![e]\!]_{SECD} \rangle \end{array}$$

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

$$\begin{aligned} & [\![(\lambda x.\lambda y.\ (+\ x\ y))]\!] \\ = \;& \langle x, [\![\lambda y.\ (+\ x\ y)]\!]\rangle \\ = \;& \langle x, \langle y, [\![(+\ x\ y)]\!]\rangle\rangle \\ = \;& \langle x, \langle y, [\![x]\!]\ [\![y]\!]\ prim_+\rangle\rangle \\ = \;& \langle x, \langle y,\ x\ y\ prim_+\rangle\rangle \end{aligned}$$

1.14 Ausführungsrelation für eine SECD Instruktion

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

Ausführungsrelation:

$\langle \hat{S},\ \hat{E},\ \hat{C},\ \hat{D}\rangle$
$\mapsto_{SECD} \quad \langle \hat{S}',\ \hat{E}',\ \hat{C}',\ \hat{D}'\rangle$
$\langle \hat{S},\ \hat{E},\ x\hat{C},\ \hat{D}\rangle$
$\mapsto_{SECD} \quad \langle w\hat{S},\ \hat{E},\ \hat{C},\ \hat{D}\rangle \qquad ,\ w = \hat{E}(x)$
$\langle b_n \dots b_1\hat{S},\ \hat{E},\ prim_{o^n}\hat{C},\ \hat{D}\rangle$
$\mapsto_{SECD} \quad \langle b\hat{S},\ \hat{E},\ \hat{C},\ \hat{D}\rangle \qquad$, wobei $(o^n\ b_1 \dots b_n) \to_\sigma b$
$\langle \hat{S},\ \hat{E},\ \langle x,\ \hat{C}'\rangle\hat{C},\ \hat{D}\rangle$
$\mapsto_{SECD} \quad \langle\ \langle\langle x,\ \hat{C}'\rangle,\ \hat{E}\rangle\hat{S},\ \hat{E},\ \hat{C},\ \hat{D}\rangle$
$\langle w\langle\langle x,\ \hat{C}'\rangle,\ \hat{E}'\rangle\hat{S},\ \hat{E},\ ap\hat{C},\ \hat{D}\rangle$
$\mapsto_{SECD} \quad \langle\ \varepsilon,\ \hat{E}'[x \mapsto w],\ \hat{C}',\ \langle \hat{S},\ \hat{E},\ \hat{C},\ \hat{D}\rangle\rangle$
$\langle w,\ \hat{E},\ \varepsilon,\ \langle \hat{S}',\ \hat{E}',\ \hat{C}',\ \hat{D}'\rangle\rangle$
$\mapsto_{SECD} \quad \langle w\hat{S}',\ \hat{E}',\ \hat{C}',\ \hat{D}'\rangle$

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

$$\langle \varepsilon,\ \emptyset,\ [\![e]\!],\ \varepsilon\rangle \quad \mapsto_{SECD} \quad \langle\ \underbrace{w}_{\text{Ergebnis}},\ \hat{E},\ \varepsilon,\ \varepsilon\rangle$$

$$\rightsquigarrow eval_{SECD} = eval_v$$

2 Endrekursion

2.1 Beobachtung

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

$$
\begin{array}{rcl}
[\![b]\!] & = & b \\
[\![x]\!] & = & x \\
[\![(e_0\ e_1)]\!] & = & [\![e_0]\!]\ [\![e_1]\!]\ ap \qquad \text{(nicht endrekursiver Kontext)} \\
[\![(o^n\ e_1 \dots e_n]\!] & = & [\![e_1]\!]\ \dots\ [\![e_n]\!]\ prim_{o^n} \\
[\![(\lambda x.\ e)]\!] & = & \langle x,\ [\![e]\!]'\rangle
\end{array}
$$

endrekursive Aufrufe: Aufrufe ohne Kontext
endrekursiver Kontext:

$$
\begin{array}{rcl}
[\![b]\!]' & = & b \\
[\![x]\!]' & = & x \\
[\![(e_0\ e_1)]\!]' & = & [\![e_0]\!]\ [\![e_1]\!]\ tailap \qquad \text{wobei}\hat{C} = \dots \mid tailap\ \hat{C} \\
[\![(o^n\ e_1 \dots e_n]\!]' & = & [\![e_1]\!]\ \dots\ [\![e_n]\!]\ prim_{o^n} \\
[\![(\lambda x.\ e)]\!]' & = & \langle x,\ [\![e]\!]'\rangle
\end{array}
$$

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

3 Speichermodell für die SECD Maschine

3.1 Einführung von Speicheradressen und dem Heap

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

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

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

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

$[\![(:= \; x \; e)]\!] = [\![e]\!] :=_x$

Vorher: Umgebung bildet Variablen auf Werte ab
Nachher: Umgebung bildet Variablen auf Speicheradressen
Zusätzlich: Heap bildet Speicheradressen auf Werte ab

$$\begin{aligned}
\hat{E} &= V \mapsto \underbrace{\hat{A}}_{\text{Adresse}} \\
\hat{H} &= \hat{A} \mapsto \hat{V} \mid free \\
\hat{V} &= b \mid \langle\langle x, \; \hat{C}\rangle, \; \hat{E}\rangle \mid void \qquad \text{wobei void Wert von Zuweisungsausdruck}
\end{aligned}$$

3.2 Ausführungsrelationen für SECDH Instruktionen

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

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

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

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

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

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

$\langle \hat{V}\hat{S}, \; \hat{E}, \; :=_x \hat{C}, \; \hat{D}, \; \hat{H}\rangle$
$\mapsto \quad \langle void\hat{S}, \; \hat{E}, \; \hat{C}, \; \hat{D}, \; \hat{H}[\hat{E}(x) \mapsto \hat{V}]\rangle$

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.

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

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.

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

$\rightsquigarrow$ Nichtdeterminismus

Garbage Collection

3.5 Einfachste Lösungstechnik: Fixpunktiteration

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