Kapitel 3
Korrektheit der Speicherabbildung

Wolf Zimmermann

Verifikation von Übersetzern

Inhalt

Ziele

- Kennenlernen der allgemeinen Vorgehensweise bei Transformation des Zustandsraums
- Korrektheitsnachweis der Speicherabbildung
- Bewusstsein für das Zusammenwirken mit der statischen Semantik

3.1 Einleitung

Vorgehen

- Zuordnung einer Semantik auf Basis des Speicherzustandsraums (Zustandsraum bis auf Befehlszeiger) der Zielmaschine
- Nachweis einer eingeschränkten 1-1-Simulation

Speicherzustandsraum der Dec-Alpha

- 64-Bit adressierter Byte-orientierter Speicher
- ☞ Ausrichtung 4 bei Wörtern
- Register R30 ist Kellerzeiger
- Register R29 ist Basisregister
- Statusregister für Ausnahmen

Partitionierung des Speichers (Aufgabe des Betriebssystem)

- Geschützter Bereich, der das Programm enthält
- Keller wächst von niedrigen zu hohen Adressen
- Halde wächst von hohen zu niedrigen Adressen
- Auswahl eines Registers (z.B R14) als Haldenzeiger, das bei Unterprogrammaufruf nicht zerstört wird.
- ⇒ Speicherüberlauf, wenn Anforderung zur Überlappung von Speicher und Halde führen würde

Bytes und Wörter

- Bytes sind Folgen von 8 Bits
- Wörter sind Folgen von 64-Bits
- Wörter können als ganze Zahlen $-2^{63} \leq z \leq 2^{63}-1$ interpretiert werden
- Wörter können als Gleitkommazahlen doppelter Genauigkeit interpretiert werden
- Dies sind auch Adressen
- Bytes müssen zu Wörtern zusammengesetzt werden können

spec VALUE_Q **extends** INT, FLOAT
sorts BIT, $\mathrm{BITSEQ}_i, i = 1,\ldots,64$
operations

O, L :		$\rightarrow$ BIT	
$mkbitseq_i$:	BIT^i	$\rightarrow \mathrm{BITSEQ}_i$	$i = 1,\ldots,64$
bit_i :	$\mathrm{BITSEQ}_i \times$ INT	$\rightarrow$?BIT	$i = 1,\ldots,64$
$sel_{i,j}^{k,j}$:	BITSEQ_i	$\rightarrow \mathrm{BITSEQ}_{j-k+1}$	$i = 1,\ldots,64, 1 \leq k < j \leq 64$
$(+\!+_i^j)$:	$\mathrm{BITSEQ}_i \times \mathrm{BITSEQ}_j$	$\rightarrow \mathrm{BITSEQ}_{i+j}$	$i,j \geq 1,\ i+j \leq 64$
$quad$:	BYTE^8	$\rightarrow$ QUAD	
$selbyte$:	QUAD $\times$ INT	$\rightarrow$?BYTE	
$quadtoint$:	QUAD	$\rightarrow$ INT	
$inttoquad$:	INT	$\rightarrow$?QUAD	
$quadtofloat$:	QUAD	$\rightarrow$ FLOAT	
$floattoquad$:	FLOAT	$\rightarrow$ QUAD	
$(+_I), (*_I), (-_I)$:	QUAD $\times$ QUAD	$\rightarrow$ QUAD	
$(+_F), (*_F), (-_F)$:	QUAD $\times$ QUAD	$\rightarrow$ QUAD	
$(/_F)$:	QUAD $\times$ QUAD	$\rightarrow$?QUAD	
$convert$:	QUAD $\times$ TYPE $\times$ TYPE	$\rightarrow$?QUAD	

BYTE	$\triangleq \mathrm{BITSEQ}_8$	$bit_i(n)$	selektiert n-tes Bit
QUAD	$\triangleq \mathrm{BITSEQ}_{64}$	$sel_{i,j}^{k,j}$	Teilfolge $b_k \cdots b_j$
$mkbyte$	$\triangleq mkbitseq_8$	$+\!+_i^j$	Anhängen zweier Bitfolgen
$mkquad$	$\triangleq mkbitseq_{64}$	$selbyte(n)$	selektiert n-tes Byte eines Worts
		$+_I$, $*_i$ etc.	Arithmetische Operationen auf ganzen Zahlen
		$+_F$, $*_F$ etc.	Arithmetische Operationen auf Gleitkommazahlen
		$convert(x,\ldots)$	Konversion in äquivalente Gleitkommazahl

Bytes und Wörter

Einige Axiome

$D(bit_i(b,n)) \Leftrightarrow n \geq 0 \wedge n < i$
$bit_i(mkbitseq_i(b_{i-1},\ldots,b_0),n) \doteq b_n,\ i = 1,\ldots,64,\ n = 0,\ldots,i-1$
$++_i^j(mkbitseq_i(b_{i-1},\ldots,b_0), mkbitseq(b'_{j-1},\ldots,b'_0) \doteq mkbitseq_i(b_{i-1},\ldots,b_0,b'_{j-1},\ldots,b'_0)$
$quad(b_7,\ldots,b_0) \triangleq b_7 ++_8^8 \cdots ++_{56}^8 b_0$

Satz 3.1 (Hardwarekorrektheit der arithmetischen Operationen)

Die Arithmetisch-Logische Einheit des DEC-Alpha-Prozesors implementiert eine $\textsc{Value}_\alpha$-Algebra $\mathfrak{V}$, so dass folgende Eigenschaften erfüllt sind:

$\mathfrak{V} \models x +_I y \doteq inttoquad(intplus(quadtoint(x), quadtoint(y)))$
$\mathfrak{V} \models x *_I y \doteq inttoquad(intmul(quadtoint(x), quadtoint(y)))$
$\mathfrak{V} \models x -_I y \doteq inttoquad(intminus(quadtoint(x), quadtoint(y)))$
$\mathfrak{V} \models x +_F y \doteq floattoquad(fltplus(quadtofloat(x), quadtofloat(y)))$
$\mathfrak{V} \models x *_F y \doteq floattoquad(fltmul(quadtofloat(x), quadtofloat(y)))$
$\mathfrak{V} \models x -_F y \doteq floattoquad(fltminus(quadtofloat(x), quadtofloat(y)))$
$\mathfrak{V} \models x /_F y \doteq floattoquad(fltdiv(quadtofloat(x), quadtofloat(y)))$
$\mathfrak{V} \models convert(x, inttype, flttype) \doteq floattoquad(inttofloat(quadtoint(x)))$

Beobachtung

Division und Rest bei ganzen Zahlen ist nicht vorhanden

⇒ Muss explizit programmiert werden

-3.5mm

Speicherzustandsraum

Speicherzustand

- 64-Bit-adressierter Byte-orientierte Speicher
- 32 64-Bit Ganzzahlregister (R31 ist immer 0)
- 32 64-Bit Gleitkommaregister (F31 ist immer 0)
- 1 64-Bit Gleitkommastatusregister (FPCR) (auch für Ausnahmen)

spec $\textsc{MemState}_\alpha$ **extends** $\textsc{Value}_\alpha$
sorts BYTELIST
operations

mem :	QUAD	→	BYTE
reg :	$BITSEQ_5$	→	QUAD
freg :	$BITSEQ_5$	→	QUAD
fpcr :		→	QUAD
inp :		→	BYTELIST
out :		→	BYTELIST

Einige nützliche Makros

- Lesen eines Wortes aus dem Speicher mit Adresse a:
  $Read(a) \triangleq quad(mem(a),\ldots,mem(a +_I 7))$
- Schreiben eines Wortes v in den Speicher an Adresse a (Big-Endian):

$$Store(a,v) \triangleq \begin{array}{ll} mem(a) & := selbyte(v,7) \\ mem(a+1) & := selbyte(v,6) \\ & \vdots \\ mem(a+7) & := selbyte(v,0) \end{array}$$

- Explizite Benennung von Registern:
  $R0 \triangleq reg(mkbitseq_5(O,O,O,O,O)),\ldots,R31 \triangleq reg(mkbitseq_5(L,L,L,L,L))$
- Explizite Benennung von Gleitkomma-Registern:
  $F0 \triangleq freg(mkbitseq_5(O,O,O,O,O)),\ldots,F31 \triangleq freg(mkbitseq_5(L,L,L,L,L))$

3.2 Transformation des Zustandsraums

Aufgaben

- Abbildung der ganzen Zahlen, Gleitkommazahlen und Werte
- Abbildung der Adressen
- Abbildung des Laufzeitkellers
- Abbildung des Speichers
- Abbildung der Ausnahme

Abbildung der Zwischenergebnisse

Diese Abbildung erfolgt durch Registerzuteilung

⇒ Aufgabe der Codeerzeugung

⇒ Zwischenergebnisse werden noch an AST-Knoten gebunden

Problem

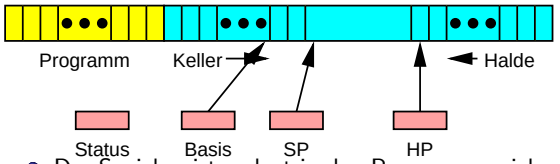
Divisionsoperatoren auf ganzen Zahlen existieren nicht

⇒ Muss bei Zwischencodeerzeugung als Funktionsaufruf implementiert werden

☞ Hier werden eine Operation $/_I$ und $\%_I$ definiert, so dass
$x/_I y \doteq inttoquad(intdiv(quadtoint(x), quadtoint(y)))$ gilt.
$x\%_I y \doteq inttoquad(mod(quadtoint(x), quadtoint(y)))$

Partitionierung des Speichers

Grundprinzip von Laufzeitsystemen



- Der Speicher ist zerlegt in den Programmspeicher und den Arbeitsspeicher des Programms
- Keine Adresse im Programm darf auf den Programmspeicher schreibend zugreifen

⇒ Korrektheit des Betriebssystem

- Der Arbeitsspeicher des Programms ist zerlegt in den Keller und die Halde, die von verschiedenen Enden aufeinander zuwachsen
- Der Kellerzeiger SP und der Haldenzeiger HP zeigen den aktuellen Pegelstand an.
- Das Statusregister gibt Ausnahmen an.
- Alle konstanten Adressen im Programm sind relativ zum Basisregister adressiert

Konsequenzen für die Speicherabbildung

Grundprinzip

- Der Laufzeitkeller wird auf den Keller abgebildet
- Globale Variablen werden ganz unten im Keller gespeichert
- Lokale Variablen, deren Typ ein Basistyp oder ein Verbund ist, werden direkt im Laufzeitkeller gespeichert
- Mit `new` erzeugte Objekte werden auf der Halde gespeichert
- Das Prozedurframe wird samt Parametern und lokalen Variablen bei Aufruf auf dem Keller angelegt und bei Verlassen wieder entfernt

⇒ Lokale Variablen befinden sich in diesem Frame

⇒ Lokale Variablen werden relativ zum Anfang des Frames adressiert

⇒ Basisregister enthält die Anfangsadresse des obersten Frames

- Bei Verbundobjekten werden die Verbundfelder ebenfalls relativ zum Verbundanfang adressiert

Anforderungen an die Speicherabbildung

- Die Relativadressen sind alle nicht-negativ
- Zwei Speicherbereiche lokaler Variablen dürfen sich nicht überlappen
- Zwei Speicherbereiche von Verbundfeldern dürfen sich nicht überlappen
- Zwei Speicherbereiche globaler Variablen dürfen sich nicht überlappen
- Die Ausrichtung einer Prozedur, Blocks bzw. Verbunds ist so bestimmt, dass sie maximal unter den Ausrichtungen der lokalen Variablen und Blöcke bzw. Verbundfeldern ist. Damit benötigt man die statischen Funktionen $size, align : \text{PROG} \times \text{OCC} \rightarrow ?\text{INT}$
- Zwei Prozedurframes auf dem Keller dürfen sich nicht überlappen
- Zwei Klassenobjekte auf der Halde dürfen sich nicht überlappen

Beobachtungen

- Wenn der Kellerzeiger oder Haldenzeiger bewegt wird, muss er um die entsprechende Größe (inkl. Ausrichtung) verschoben werden.

⇒ Die Größe und Ausrichtung von Prozeduren und Verbunden muss so bestimmt werden, dass alle Objekte in das Frame passen

- Prozedurframe speichern zusätzlich noch Verweise auf den dynamischen Vorgänger und die Rücksprungadresse
- Die Ausrichtung von Wörtern ist auf der DEC-Alpha immer 4 (d.h. die letzen beiden Bits sind O)

Verwaltung der Relativadressen

spec RELTAB **extends** PROG
sorts RELTAB, ID, IDLIST, INT, BOOL, TYPETAB
operations

mktab :		→ RELTAB	erzeugt leere Tabelle
$addAddr_\alpha$:	RELTAB × ID × INT × INT × INT	→ RELTAB	neue Relativ Adresse
isDefined	RELTAB × ID	→ BOOL	
addr :	RELTAB × ID	→?QUAD	ermittelt Relativadresse
size :	RELTAB × ID	→?INT	ermittelt Größe
align :	RELTAB × ID	→?INT	ermittelt Ausrichtung
maxalign	RELTAB	→?INT	maximale Ausrichtung
minaddr	RELTAB	→?QUAD	minimale Adresse
maxaddr	RELTAB	→?QUAD	maximale Adresse
lastsize	RELTAB	→?INT	Größe des letzten Objekts
$deladdrs_\alpha$:	RELTAB × ID	→ RELTAB	beseitigt Bindung
$(++_r)$:	RELTAB × RELTAB	→ RELTAB	Anhängen von Bindungen
$(\backslash_r)$:	RELTAB × IDLIST	→ RELTAB	Entfernen von Bindungen
minaddr	RELTAB × IDLIST	→?QUAD	minimale Adresse der Objekte
mktypetab :		→ TYPETAB	Tabelle für Relativadressen von Typen
addType :	TYPETAB × ID × RELTAB	→ TYPETAB	neuer Typ
size :	TYPETAB × ID	→ INT	Größe der Objekte eines Typs
align :	TYPETAB × ID	→ INT	Ausrichtung der Objekte eines Typs
reltab :	TYPETAB × ID	→ RELTAB	Relativadressen der Verbundfelder

axioms Übung

- Mit der Relativadresse wird auch die Größe und die Ausrichtung eingetragen
- In Typtabelle werden nur Verbunde und Klassen mit den Relativadressen der Verbundfelder eingetragen.

Korrektheitsanforderungen an die Speicherabbildung

Anforderung 3.2 (Speicherabbildung)

Für jede statische RELTAB-Algebra $\mathfrak{A}$, alle Programme p und alle Tabellen $e \triangleq reladdr(p, o)$, $o' \triangleq parent(o)$, $e' \triangleq reladdr(p, o')$ mit $occ(p, o)$ **is** BLOCK, $occ(p, o)$ **is** PROCDECL, $occ(p, o)$ **is** STRUCT, $occ(p, o)$ **is** CLASS und $occ(p, o)$ **is** PROG gilt:

$\mathfrak{A} \models minaddr(e) \geq 0 \doteq true$
$\mathfrak{A} \models \forall x, y : \text{ID} \bullet \; addr(e, x) \leq_I addr(e, y) \doteq true \wedge$
$\qquad addr(e, y) <_I addr(e, x) + size(e, x) \doteq true \Rightarrow x \doteq y$
$\mathfrak{A} \models \forall x : \text{ID} \bullet isAligned(addr(e, x), align(e, x)) \doteq true$
$\mathfrak{A} \models align(p, o) \doteq maxalign(e)$
$\mathfrak{A} \models size(p, o) \leq maxaddr(e) + lastsize(e) \doteq \text{true}$
$\mathfrak{A} \models occ(p, o)$ **is** PROCDECL $\Rightarrow minaddr(e) \geq 16 \doteq true$
$\mathfrak{A} \models o \doteq first(prog) \Rightarrow minaddr(e) \geq 16 \doteq true$
$\mathfrak{A} \models occ(p, o')$ **is** BLOCK $\Rightarrow minaddr(e, locVar(p, o')) \triangleq maxaddr(e') + lastsize(e') + 1$
$\mathfrak{A} \models \neg occ(p, o)$ **is** PROCDECL $\wedge \neg occ(p, o')$ **is** BLOCK $\Rightarrow e' \doteq e$

Muss dann entsprechenden Teil des Übersetzers verifizieren, so dass Anforderung 3.2 garantiert erfüllt ist?

Beobachtung

Die Bedingungen in Anforderung 3.2 können statische überprüft werden

⇒ Ein verifizierender Übersetzer muss diese Information öffentlich zur Verfügung stellen.

⇒ Implementieren eines verifizierten Programmprüfers zur Überprüfung der Eigenschaften

⇒ Nach Durchführung der Transformation wird (verifiziert) überprüft, ob der Übersetzer auch tatsächlich die mitgeteilte Speicherabbildung verwendet hat.

Abbildung des Zustandsraums

Laufzeitkeller

Der Laufzeitkeller durch den Keller der DEC-Alpha implementiert:

- Jedes Frame hat einen Umfang

⇒ erhöhen des Kellerzeigers um diesen Umfang

- Im Frame wird auch der Zeiger auf den lokalen Vorgänger und die Rücksprungadresse gespeichert
- Nachweis, dass die Kellergesetze erfüllt sind
- Die aktuelle Umgebung *env* entspricht dem obersten Frame
- Die Basisadresse ist im Basisregister
- Für die Wahl des Kellerzeigers und des Basisregisters werden wegen der Interoperabilität mit den Betriebssystemen die Register R30 bzw. R29 gewählt.

Speicher

- Speicher des Programms wird auf den Speicher der DEC-Alpha abgebildet
- Partitionierung bedeutet, dass der Laufzeitkeller im Speicher abgelegt wird
- Haldenpegel wird in einem Register verwaltet, das bei Unterbrechungen (z.B. durch I/O) gerettet wird (z.B. Register R2)
- Register, das den Anfang angibt (Register R3)

Implementierung

Ziel

- Festlegen der Menge $\mathbb{A}_\alpha$ der Zustände einer $\textsc{State}_\alpha$-Algebra, die eine gültige Partitonierung des Speichers entsprechen
- Definieren einer Abbildung $\phi : \mathbb{A}_\alpha \to \mathbb{A}$, wobei $\mathbb{A}$ die Menge der Zustände der ASM der Sprache C-- ist aus Kapitel 2 ist

Vorgehen

- Definiere für jede $\textsc{State}_\alpha$-Algebra $\mathfrak{A} \in \mathbb{A}_\alpha$ eine $\textsc{State}_2$-Algebra $\mathfrak{B}$ mit Hilfe der Trägermengen und Funktionsinterpretationen aus $\mathfrak{A}$
- Definiere $\phi(\mathfrak{A}) \triangleq \mathfrak{B}$
- Nachweis der 1-1-Simulation (s. Abschnitt 3.3)

Signatur der Spezifikation Env_2

spec $\textsc{Env}_2$ **extends** $\textsc{Prog},\textsc{Value}$
sorts ENV, LOC, FRAME, FRAMESTACK, EXPRVALS
subsorts LOC $\sqsubseteq$ VALUE
operations

mkenv :		→ ENV
newbind :	ENV × ID × LOC	→ ENV
isUsed :	ENV × LOC	→ BOOL
bind :	ENV × ID	→?LOC
isBound :	ENV × ID	→ BOOL
delbind :	ENV × ID	→ ENV
$(++)$:	ENV × ENV	→ ENV
$(\backslash)$:	ENV × IDLIST	→ ENV
mkframe :	ENV × OCC × EXPRVALS	→ FRAME
mkstack :		→ FRAMESTACK
push :	FRAMESTACK × FRAME	→ FRAMESTACK
pop :	FRAMESTACK	→?FRAMESTACK
getenv :	FRAMESTACK	→?ENV
getpc :	FRAMESTACK	→?OCC
getval :	FRAMESTACK	→?EXPRVALS
novals :		→ EXPRVALS
addval :	OCC × VALUE × EXPRVALS	→ EXPRVALS
findval :	OCC × EXPRVALS	→?VALUE

Umgebungen im DEC-Alpha-Kontext

spec $\textsc{Env}_\alpha$ **extends** $\textsc{RelAddr},\textsc{Value}_\alpha$
sorts Env_α, ID, IDLIST, INT, BOOL
operations

$mkenv_\alpha$:		→ ENV_α	erzeugt leere Umgebung
$newbind_\alpha$:	$\text{ENV}_\alpha \times$ ID × QUAD	→ ENV_α	neue Zuordnung
$isUsed_\alpha$	$\text{ENV}_\alpha \times$ QUAD	→ BOOL	prüft Adresse
$isBound_\alpha$	$\text{ENV}_\alpha \times$ ID	→ BOOL	prüft Bindung
$bind_\alpha$:	$\text{ENV}_\alpha \times$ ID	→?QUAD	ermittelt Bindung
$size_\alpha$:	$\text{ENV}_\alpha \times$ ID	→?INT	ermittelt Größe
$align_\alpha$:	$\text{ENV}_\alpha \times$ ID	→?INT	ermittelt Ausrichtung
$delbind_\alpha$:	$\text{ENV}_\alpha \times$ ID	→ ENV_α	beseitigt Bindung
$(++_\alpha)$:	$\text{ENV}_\alpha \times \text{ENV}_\alpha$	→ ENV_α	Anhängen von Bindungen
$(\backslash_\alpha)$:	$\text{ENV}_\alpha \times$ IDLIST	→ ENV_α	Entfernen von Bindungen

axioms

$$D(newbind_\alpha(e,x,l)) \Leftrightarrow isUsed_\alpha(e,l) \doteq false$$
$$D(bind_\alpha(e,x)) \Leftrightarrow isBound_\alpha(e,x) \doteq true$$
$$bind_\alpha(newbind_\alpha(e,x,l),x) \doteq l$$
$$\neg x \doteq y \Rightarrow bind_\alpha(newbind_\alpha(e,y,l),x) \doteq bind_\alpha(e,x)$$
$$delbind_\alpha(mkenv_\alpha,x) \doteq mkenv_\alpha$$
$$delbind_\alpha(newbind_\alpha(e,x,l),x) \doteq e$$
$$\neg x \doteq y \Rightarrow delbind_\alpha(newbind_\alpha(e,y,l),x) \doteq delbind_\alpha(e,x)$$

Übrige Axiome sind zur Übung überlassen

Zustandsräume von C-- und DEC-Alpha

Zustandsraum von C--

spec STATE_2 **extends** ENV_2
sorts EXCEPTION, LISTOFVAL
operations

env :		→ ENV	
mem :	LOC	→?VALUE	
inp :		→ LISTOFVAL	
out :		→ LISTOFVAL	
prog :		→ PROG	
pc :		→ OCC	
exc :		→ EXCEPTION	
val :	OCC	→?VALUE	
loc :	OCC	→?LOC	
entered :	OCC	→?BOOL	
procstack :		→ FRAMESTACK	

Zustandsraum der DEC-Alpha bzgl. C--

spec STATE_α^M **extends** ENV_α
sorts BYTELIST
operations

mem_α :	QUAD	→ QUAD	
inp_α :		→ BYTELIST	
out_α :		→ BYTELIST	
prog :		→ PROG	
pc :		→ OCC	
fpcr :		→ QUAD	Statusregister
val :	OCC	→?QUAD	
loc :	OCC	→?QUAD	
entered :	OCC	→?BOOL	
UP :		→?QUAD	Anfang des Arbeitsspeichers
BP :		→?QUAD	Basisregister
SP :		→?QUAD	Kellerzeiger
HP :		→?QUAD	Haldenzeiger

Was ist nun zu tun?

Aufgabe

Angabe einer STATE_2-Algebra $\mathfrak{B} \triangleq \langle B, [\![\cdot]\!]_B$ unter Verwendung einer STATE_α^M-Algebra $\mathfrak{A} = \langle A, [\![\cdot]\!]_A \rangle$

- Interpretationen für alle Sorten von STATE_2
- Interpretationen für alle Funktionen von STATE_2
- Nachweis der Axiome von STATE_2

Vorgehen

- Intuitive Idee
- Sorten
- Interpretation der Funktionen auf diesen Sorten
- Nachweis der Axiome

☞ Ggf. Einführung weiterer Operationen zu ENV_α

Umgebungen

Intuitive Idee

- Adressen werden durch Wörter repräsentiert
- Programme werden in der Algebra $\mathfrak{B}$ genauso interpretiert wie $\mathfrak{A}$

Interpretationen

Sorten: $B_{\text{ENV}} \triangleq A_{\text{ENV}_\alpha}$, $B_{\text{LOC}} \triangleq A_{\text{QUAD}}$, $B_{\text{ID}} \triangleq A_{\text{ID}}$ und $B_{\text{IDLIST}} \triangleq A_{\text{IDLIST}}$

Operationen:

$[\![mkenv]\!]_B \triangleq [\![mkenv_\alpha]\!]_A$
$[\![newbind]\!]_B \triangleq [\![newbind_\alpha]\!]_A$
$[\![isUsed]\!]_B \triangleq [\![isUsed_\alpha]\!]_A$
$[\![bind]\!]_B \triangleq [\![bind_\alpha]\!]_A$
$[\![isBound]\!]_B \triangleq [\![isBound_\alpha]\!]_A$
$[\![++]\!]_B \triangleq [\![++_\alpha]\!]_A$
$[\![\backslash]\!]_B \triangleq [\![\backslash_\alpha]\!]_A$

$[\![intplus]\!]_B \triangleq [\![+_I]\!]_A$ usw.
$[\![n]\!] \triangleq$ Darstellung im 2-Komplement
$[\![coerce]\!]_B \triangleq [\![convert]\!]_A$
$[\![mkprog]\!]_B \triangleq [\![mkprog]\!]_A$ usw.
$[\![occ]\!]_B \triangleq [\![occ]\!]_A$ usw.

Korollar 3.3 (Korrektheit der Umgebungen)

$\mathfrak{B} \models D(newbind(e,x,l)) \Leftrightarrow isUsed(e,l) \doteq false$
$\mathfrak{B} \models D(bind(e,x)) \Leftrightarrow \mathfrak{B} \models isBound(e,x) \doteq true$
$\mathfrak{B} \models bind(newbind(e,x,l),x) \doteq l$
$\mathfrak{B} \models \neg x \doteq y \Rightarrow bind(newbind(e,y,l),x) \doteq bind(e,x)$
$\mathfrak{B} \models delbind(mkenv,x) \doteq mkenv$
$\mathfrak{B} \models delbind(newbind(e,x,l),x) \doteq e$
$\mathfrak{B} \models \neg x \doteq y \Rightarrow delbind(newbind(e,y,l),x) \doteq delbind(e,x)$
analog für weitere Axiome

Laufzeitkeller

Intuitive Idee

- Die Ausdruckszwischenergebnisse werden noch über eine geradzahlige Folge von Wörtern verwaltet

⇒ Funktionen *occtoquad* : PROG × OCC →?QUAD und *quadtoocc* : PROG × QUAD →?OCC

- Laufzeitkeller ist eine Liste solcher Frames

Interpretation

Sorten:
$B_{\text{FRAME}} \triangleq A_{FRAME} \times A_{\text{OCC}} \times A_{\text{BYTELIST}}$
$B_{\text{FRAMESTACK}} \triangleq A_{FRAME}^*$
$B_{\text{EXPRVALS}} \triangleq A_{\text{BYTELIST}}$

Operationssymbole:

$[\![novals]\!]_B \triangleq []$
$[\![addval]\!]_B(o,v,b) \triangleq [\![occtoquad]\!]_A(o)++v++b$
$[\![findval]\!]_B(o,b) \triangleq b[16i+8..16i+15]$
wobei i minimal, so dass $b[16i..16i+7] = [\![occtoquad]\!]_A(o)$
$[\![mkframe]\!]_B(e,o,b) \triangleq \langle e,o,b \rangle$
$[\![mkstack]\!]_B \triangleq []$
$[\![push]\!]_B(s,f) \triangleq f : s$
$[\![pop]\!]_B(s) \triangleq tail(s)$
$[\![getenv]\!]_B(s) \triangleq e$ wobei $head(s) = \langle e,o,b \rangle$
$[\![getpc]\!]_B(s) \triangleq o$ wobei $head(s) = \langle e,o,b \rangle$
$[\![getval]\!]_B(s) \triangleq o$ wobei $head(s) = \langle e,o,b \rangle$

Laufzeitkeller

Lemma 3.4 (Korrektheit der Laufzeitkeller)

$\mathfrak{B} \models pop(push(s, f)) \doteq s$
$\mathfrak{B} \models getenv(push(s, mkframe(e, o, v)) \doteq e$
$\mathfrak{B} \models getpc(push(s, mkframe(e, o, v)) \doteq$
$\mathfrak{B} \models getval(push(s, mkframe(e, o, v)) \doteq v$
$\mathfrak{B} \models findval(o, addval(o, x, v)) \doteq x$
$\mathfrak{B} \models \neg o \doteq o' \Rightarrow findval(o, addval(o', x, v)) \doteq findval(o, v)]$

Beweis

Wir zeigen $\mathfrak{B} \models findval(o, addval(o, x, v)) \doteq x$

$[\![findval]\!]_B(o, [\![addval]\!]_B(o, x, v))$
$= [\![findval]\!]_B([\![occtoquad]\!]_A(o){+}{+}x{+}{+}b)$
$= x$

Übung Restliche Axiome

Dynamische Funktionen

Intuitive Idee

- *procstack* und *env* können aus dem Laufzeitkeller im Speicher sowie den Relativadressen definiert werden
- Adressen der Form $floc(l, a)$ können direkt mit Hilfe der Relativadressen für Verbundfelder berechnet werden
- Der Speicher kann direkt implementiert werden.
- Die Ausnahme exc wird durch das Statusregister *fpcr* implementiert
- Zu speichernde Werte vom Typ BOOL werden als Wort gespeichert

$\Rightarrow$ Funktionen *booltoquad* : BOOL $\to$ QUAD und *quadtobool* : QUAD $\to$ BOOL

- Ein- und Ausgabeströme werden direkt aufeinander abgebildet

Interpretationen: $\mathfrak{B} = \phi(\mathfrak{A})$

$$
\begin{array}{lll}
[\![mem]\!]_B([\![l]\!]_A) & = [\![Read(l)]\!]_A & [\![pc]\!]_B = [\![pc]\!]_A \\
[\![inp]\!]_B & = [\![inp_\alpha]\!]_A & [\![head]\!]_B(b) = b[0..7] \\
[\![out]\!]_B & = [\![out_\alpha]\!]_A & [\![tail]\!]_B(b) = b[8..|b|-1] \\
[\![exc]\!]_B & = [\![fpcr]\!]_A & [\![divByZero]\!]_B = L0^9 L0^{53} \\
[\![entered]\!]_B & = [\![entered]\!]_A & [\![nullPtrDeref]\!]_B = L0^{63} \\
[\![loc]\!]_B(o) & = [\![loc]\!]_A(o) & [\![handle]\!]_B = [\![handle]\!]_A
\end{array}
$$

$$
[\![val]\!]_B(o) = \begin{cases} quadtobool([\![val]\!]_A(o)) & \text{falls } Pri(o) = booltype \\ [\![val]\!]_A(o) & \text{sonst} \end{cases}
$$

Laufzeitkeller und Umgebung

Intuitive Idee

- Laufzeitkeller im Speicher als Keller von Aktivierungsverbunden
- *BP* enthält die Basisadresse des obersten Aktivierungsverbundes
- An Adresse *BP* wird die Basisadresse des darunterliegenden Aktivierungsverbundes gespeichert
- An Adresse $BP + 8$ wird die Rücksprungadresse (Verweis auf Aufrufer) gespeichert
- An den folgenden Adressen werden die lokalen Variablen gespeichert
- Bei Aufruf werden anschließend die Ausdruckswerte gespeicherte, die noch benötigt werden.
- Unter dem Laufzeitkeller werden globale Variablen gespeichert.
- *env* ergibt sich aus den Relativadressen des obersten Aktivierungsverbunds

Umgebung

Für die Interpretation $[\![env]\!]_B$ bzw. $[\![globenv]\!]_B$ gilt:

$$
\begin{array}{ll}
[\![isBound]\!]_A([\![env]\!]_B, x) & = [\![isBound]\!]_A([\![RelAddrTab]\!]_A, x) \\
[\![bind_\alpha]\!]_A([\![env]\!]_B, x) & = [\![BP]\!]_A + [\![addr]\!]_A([\![RelAddrTab]\!]_A, x) \\
[\![floc]\!]_B(l, a) & = l + [\![addr]\!]_A([\![RelAddrTab]\!]_A, x)
\end{array}
$$

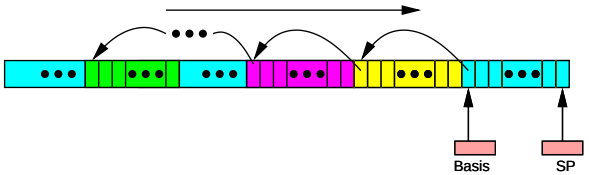
wobei $RelAddrTab \triangleq reladdr(prog, pc)$

$$
\begin{array}{ll}
[\![isBound]\!]_A([\![globenv]\!]_B, x) & = [\![isBound]\!]_A([\![GlobAddrTab]\!]_A, x) \\
[\![bind_\alpha]\!]_A([\![globenv]\!]_B, x) & = [\![UP]\!]_A + [\![addr]\!]_A([\![GlobAddrTab]\!]_A, x)
\end{array}
$$

wobei $GlobAddrTab \triangleq globaddr(prog)$

Laufzeitkeller

***i*-tes Frame von oben**



Das i-te Frame befindet sich an der folgenden Basisadresse BP_i: $BP_0 \triangleq BP$, $BP_{i+1} \triangleq Read(BP_i)$

Es gilt $get([\![procstack]\!]_B, i) = \langle e, o, v \rangle$, wobei

- $o = [\![quadtoocc(Read(\mathrm{BP}_i +_I quadtoint(8)))]\!]_B$
- e ist Umgebung mit folgenden Eigenschaften

  $$
  \begin{array}{ll}
  [\![isBound_\alpha]\!]_A(e, x) & = [\![isBound]\!]_A(RelAddrTab_i, x) \\
  [\![bind_\alpha]\!]_A(e, x) & = [\![BP_{i+1}]\!]_A + [\![addr]\!]_A(OldRelAddrTab_i, x)
  \end{array}
  $$

  wobei $RelAddrTab_i \triangleq [\![reladdr]\!]_A([\![prog]\!]_A, o)$
- Die Größe ist maximal bis zum Basisregister gegeben:
  $[\![BP_{i+1}]\!]_A + SizeLocals_i + |v| + 1 = [\![BP]\!]_A$
  wobei $SizeLocals_i \triangleq [\![maxaddr]\!]_A(RelAddrTab_i) + [\![lastsize]\!]_A(RelAddrTab_i)$
- $v \triangleq [[\![mem_\alpha]\!]_A([\![BP_{i+1}]\!]_A + SizeLocals + 1), \ldots, [\![mem_\alpha]\!]_A([\![BP_i]\!]_A - 1)]$

Diskussion

Ein Teil der Algebra impliziert, dass nicht alle Zustände so abgebildet werden können

⇒ Invarianten auf erreichbaren Zuständen erforderlich

- Zum Beweis dieser Invarianten werden einige weitere Invarianten benötigt
  - Aussagen über berechnete Adressen
  - Aussagen über Adressen im Speicher

State_α^M-Invarianten

- Es gibt ein n mit $BP_n = UP$ und $BP_i < BP_{i-1}$: $\mathfrak{A} \models BP_{i+1} <_I BP_i \doteq true$, $\mathfrak{A} \models BP_n \doteq UP$
- Der Aktivierungsverbund passt genau zwischen BP_{i+1} und BP_i: $\mathfrak{A} \models BP_{i+1} +_I SizeLocals_{i+1} +_I SizeRes_{i+1} \doteq BP_i$ wobei

$$\begin{array}{ll} SizeLocals_i & \triangleq FrameSize(RelAddrTab_i) \\ FrameSize(tab) & \triangleq maxaddr(tab) + lastsize(tab) \\ RelAddrTab_i & \triangleq reladdr(prog, CallPc_i) \\ CallPc_i & \triangleq occtoquad(Read(BP_i + 8)) \\ SizeRes_i & \triangleq inttoquad(length(exproccs(prog, CallPc_i)) * 16) \end{array}$$

- Zwischen UP und BP_{n-1} passen ein Wort die globalen Variablen: $\mathfrak{A} \models UP +_I SizeGlobals + 8 \doteq BP_{n-1}$ wobei

$$\begin{array}{ll} SizeGlobals & \triangleq FrameSize(GlobAddrTab) \\ GlobAddrTab & \triangleq reladdr(prog, ()) \end{array}$$

- Der Kellerzeiger SP folgt immer auf die lokalen Variablen:

$$\begin{array}{l} \mathfrak{A} \models \neg CT \textbf{ is } \text{BLOCK} \Rightarrow SP \doteq BP +_I SizeLocals \\ \mathfrak{A} \models CT \textbf{ is } \text{BLOCK} \wedge entered(pc) \doteq false \Rightarrow SP \doteq BP +_I SizeLocals \\ \mathfrak{A} \models CT \textbf{ is } \text{BLOCK} \wedge entered(pc) \doteq true \Rightarrow SP \doteq BP +_I SizeLocals' \end{array}$$

wobei

$$\begin{array}{ll} SizeLocals & \triangleq FrameSize(reladdr(prog, pc)) \\ SizeLocals' & \triangleq FrameSize(reladdr(prog, pc.1)) \end{array}$$

- Der Kellerzeiger ist immer kleiner gleich dem Haldenzeiger: $\mathfrak{A} \models SP \leq_I HP \doteq true$
- Die ausgerechneten Adressen befinden sich immer im Keller oder auf der Halde, wobei die Adressen globaler Variablen (inklusive Verbundfelder) im untersten Aktivierungsverbund sind.

$$\begin{array}{l} \mathfrak{A} \models loc(o) <_I UP \doteq true \Rightarrow loc(o) \doteq null \\ \mathfrak{A} \models loc(o) <_I BP +_I 16 \doteq true \wedge \neg IsField(o) \Rightarrow loc(o) +_I lastsize(GlobAddrTab) \leq_I BP_{n-1} \doteq true \\ \mathfrak{A} \models loc(o) <_I BP +_I 16 \doteq true \wedge IsField(o) \Rightarrow loc(o) +_I lastsize(RelAddrTab) \leq_I BP_{n-1} \doteq true \\ \mathfrak{A} \models loc(o) + lastsize(RelAddrTab) >_I SP \doteq true \Rightarrow loc(o) \geq_I HP \doteq true \end{array}$$

wobei $IsField(o) \triangleq occ(prog, o)$ **is** FIELD.

Invarianten für Adressen im Speicher bzw. als Wert

- Falls innerhalb eines Ausdrucks oder einer Zuweisung eine Adresse als Wert berechnet wurde, ist diese Adresse *null* oder verweist auf die Halde

$$\begin{array}{l} \mathfrak{A} \models InExpr(o) \wedge IsClassDes(o) \wedge val(o) <_I HP \doteq true \Rightarrow val(o) \doteq null \\ \mathfrak{A} \models InAssign(o) \wedge IsClassDes(o) \wedge val(o) <_I HP \doteq true \Rightarrow val(o) \doteq null \end{array}$$

wobei

$$\begin{array}{ll} InExpr(o) & \triangleq CT \textbf{ is } \text{EXPR} \wedge isPrefix(pc, o) \doteq true \\ IsClassDes(o) & \triangleq isClass(pri(prog, o)) \doteq true \wedge occ(prog, o) \textbf{ is } \text{DES} \\ InAssign(o) & \triangleq CT \textbf{ is } \text{ASSIGN} \wedge isPrefix(pc, o) \doteq true \end{array}$$

- Im Speicher enthaltene Adressen sind entweder *null* oder verweisen auf die Halde.

  Problem: Wie charakterisiert man welcher Wert im Speicher eine Adresse ist?
  - Typisierung des Speichers

  ⇒ Weitere dynamische Funktion *type* : QUAD →?TYPE.

  $\mathfrak{A} \models isClass(type(l)) \doteq true \wedge Read(l) <_I HP \doteq true \Rightarrow Read(l) \doteq null$

- Die Typen sind den Adressen korrekt zugeordnet.

$$\begin{array}{l} \mathfrak{A} \models isGlobal(prog, x) \doteq true \Rightarrow type(Glob(x)) \doteq gettype(deftab(prog, ()), x) \\ \mathfrak{A} \models isDefined(Proc_i, x) \doteq true \Rightarrow type(Loc_i(x)) \doteq gettype(Proc_i, x)) \\ \mathfrak{A} \models isDefined(deftab(prog, pc), x) \Rightarrow type(Loc_0(x)) \doteq gettype(deftab(prog, pc), x) \\ \mathfrak{A} \models isStruct(DefTab(type(l)), x) \Rightarrow type(l +_I FieldAddr(x)) \doteq gettype(DefTab(type(l)), x) \\ \mathfrak{A} \models isClass(type(l)) \doteq true \wedge \neg l \doteq null \Rightarrow type(Read(l)) \doteq dereftype(type(l)) \end{array}$$

wobei

$$\begin{array}{ll} Glob(x) & \triangleq UP +_I addr(GlobAddrTab, x) \\ Loc_i(x) & \triangleq BP_i +_I addr(RelAddrTab_i, x) \\ Proc_i & \triangleq deftab(prog, CallPc_{i-1}) \\ DefTab(A) & \triangleq identifyDef(deftab(prog, ()), A) \\ FieldAddr(x) & \triangleq addr(reladdr(DefTab(type(l))), x) \end{array}$$

Axiom: $dereftype(class(A)) \doteq struct(A)$

Lemma 3.5 (Benutzte Adressen)

Sei $l \in \mathcal{T}_{\text{LOC}}$. Falls $[\![SP]\!]_A \leq [\![l]\!]_B < [\![HP]\!]_A$ gilt, ist $[\![isUsed(l)]\!]_B = false$.

Zusammenfassung der State_α^M-Invarianten

$$\begin{array}{l} \mathfrak{A} \models BP_{i+1} <_I BP_i \doteq true \\ \mathfrak{A} \models BP_n \doteq UP \\ \mathfrak{A} \models BP_{i+1} +_I SizeLocals_{i+1} +_I SizeRes_{i+1} \doteq BP_i \\ \mathfrak{A} \models UP +_I SizeGlobals + 8 \doteq BP_{n-1} \\ \mathfrak{A} \models \neg CT \textbf{ is } \text{BLOCK} \Rightarrow SP \doteq BP +_I SizeLocals \\ \mathfrak{A} \models CT \textbf{ is } \text{BLOCK} \wedge entered(pc) \doteq false \Rightarrow SP \doteq BP +_I SizeLocals \\ \mathfrak{A} \models CT \textbf{ is } \text{BLOCK} \wedge entered(pc) \doteq true \Rightarrow SP \doteq BP +_I SizeLocals' \\ \mathfrak{A} \models SP \leq_I HP \doteq true \\ \mathfrak{A} \models loc(o) <_I UP \doteq true \Rightarrow loc(o) \doteq null \\ \mathfrak{A} \models loc(o) <_I BP +_I 16 \doteq true \wedge \neg IsField(o) \Rightarrow loc(o) +_I lastsize(GlobAddrTab) \leq_I BP_{n-1} \doteq true \\ \mathfrak{A} \models loc(o) <_I BP +_I 16 \doteq true \wedge IsField(o) \Rightarrow loc(o) +_I lastsize(RelAddrTab) \leq_I BP_{n-1} \doteq true \\ \mathfrak{A} \models loc(o) + lastsize(RelAddrTab) >_I SP \doteq true \Rightarrow loc(o) \geq_I HP \doteq true \\ \mathfrak{A} \models InExpr(o) \wedge IsClassDes(o) \wedge val(o) <_I HP \doteq true \Rightarrow val(o) \doteq null \\ \mathfrak{A} \models InAssign(o) \wedge IsClassDes(o) \wedge val(o) <_I HP \doteq true \Rightarrow val(o) \doteq null \\ \mathfrak{A} \models isClass(type(l)) \doteq true \wedge Read(l) <_I HP \doteq true \Rightarrow Read(l) \doteq null \\ \mathfrak{A} \models isGlobal(prog, x) \doteq true \Rightarrow type(Glob(x)) \doteq gettype(deftab(prog, ()), x) \\ \mathfrak{A} \models isDefined(Proc_i, x) \doteq true \Rightarrow type(Loc_i(x)) \doteq gettype(Proc_i, x) \\ \mathfrak{A} \models isDefined(deftab(prog, pc), x) \Rightarrow type(Loc_0(x)) \doteq gettype(deftab(prog, pc), x) \\ \mathfrak{A} \models isStruct(DefTab(type(l)), x) \Rightarrow type(l +_I FieldAddr(x)) \doteq gettype(DefTab(type(l)), x) \\ \mathfrak{A} \models isClass(type(l)) \doteq true \wedge \neg l \doteq null \Rightarrow type(Read(l)) \doteq dereftype(type(l)) \end{array}$$

$$\begin{array}{lll} CallPc_i & \triangleq occtoquad(Read(BP_i + 8)) & Glob(x) \triangleq UP +_I addr(GlobAddrTab, x) \\ RelAddrTab_i & \triangleq reladdr(prog, CallPc_i) & Loc_i(x) \triangleq BP_i +_I addr(RelAddrTab_i, x) \\ FrameSize(tab) & \triangleq maxaddr(tab) +_I lastsize(tab) & Proc_i \triangleq deftab(prog, CallPc_i) \\ SizeLocals_i & \triangleq FrameSize(RelAddrTab_i) & \\ SizeRes_i & \triangleq inttoquad(length(exproccs(prog, CallPc_i)) * 16) & \\ GlobAddrTab & \triangleq reladdr(prog, ()) & \\ SizeGlobals & \triangleq FrameSize(GlobAddrTab) & DefTab(A) \triangleq identifyDef(deftab(prog, ()), A) \\ SizeLocals & \triangleq FrameSize(reladdr(prog, pc)) & FieldAddr(x) \triangleq addr(reladdr(DefTab(type(l))), x) \\ SizeLocals' & \triangleq FrameSize(reladdr(prog, pc.1)) & \\ IsField(o) & \triangleq occ(prog, o) \textbf{ is } \text{FIELD} & \\ InExpr(o) & \triangleq CT \textbf{ is } \text{EXPR} \wedge isPrefix(pc, o) \doteq true & \\ IsClassDes(o) & \triangleq isClass(pri(prog, o)) \doteq true \wedge occ(prog, o) \textbf{ is } \text{DES} & \\ InAssign(o) & \triangleq CT \textbf{ is } \text{ASSIGN} \wedge isPrefix(pc, o) \doteq true & \end{array}$$

3.3 Verifikation der Speicherabbildung

Vorgehen

- Definition der Initialzustände auf DEC-Alpha-Semantik
- Nachweis, dass dieser Invariante erfüllt
- Nachweis, dass mit der Abbildung ϕ aus Abschnitt 3.2 $\phi(\mathfrak{I})$ eine INITSTATE_2-Algebra für jeden Initialzustand der DEC-Alpha-Semantik ist
- Definition der Zustandsübergangsregeln auf Basis von STATE_α^M-Algebren
- Nachweis, dass der Nachfolgezustand die Invarianten erfüllt
- Nachweis, dass $\phi(next_U(\mathfrak{A})) = next_{U'}(\phi(\mathfrak{A}))$

$$\begin{array}{ccc} \mathfrak{B} & \xrightarrow[next]{} & \mathfrak{B}' \\ \uparrow\phi & & \uparrow\phi \\ \mathfrak{A} & \xrightarrow[next]{} & \mathfrak{A}' \end{array}$$

Initialzustände

Initialzustände der C--Semantik

spec INITSTATE_2 **extends** STATE_2
axioms
$$\begin{array}{c} \neg D(mem(x)) \\ out \doteq [] \\ env \doteq mkenv \\ pc \doteq first(prog) \neg D(loc(o)) \\ \neg D(val(o)) \\ D(entered(o)) \Leftrightarrow occ(prog, o) \textbf{ is BLOCK} \\ entered(o) \doteq false \\ exc \doteq ok \\ procstack \doteq mkstack \end{array}$$

Initialzustände der C--Semantik bzgl. DEC-Alpha

spec INITSTATE_α **extends** STATE_α
axioms
$$\begin{array}{c} out \doteq [] \\ pc \doteq first(prog) \\ \neg D(loc(o)) \\ \neg D(val(o)) \\ D(entered(o)) \Leftrightarrow occ(prog, o) \textbf{ is BLOCK} \\ entered(o) \doteq false \\ fpcr \doteq 0^{64} \\ HP \doteq maxaddr \\ BP \doteq UP +_I SizeGlobals +_8 \\ SP \doteq BP +_I 16 \\ Read(BP) \doteq UP \\ type(UP +_I addr(GlobAddrTab, x)) \doteq gettype(deftab(prog, ()), x) \\ D(type(I)) \wedge isStruct(type(I)) \Rightarrow type(I + FieldAddr(x)) \doteq gettype(DefTab(type(I)), x) \\ D(type(I)) \Leftrightarrow I \doteq UP +_I addr(GlobAddrTab, x) \vee I \doteq I' +_I FieldAddr(x) \wedge isStruct(type(I')) \end{array}$$

Korrektheit

Lemma 3.6 (Initialzustände erfüllen Invarianten)

Jede INITSTATE_Q-Algebra $\mathfrak{I}$ erfüllt die Invarianten, d.h.
$\mathfrak{I} \models BP_1 \leq_I BP_0 \doteq true$
$\mathfrak{I} \models BP_1 \doteq UP$
$\mathfrak{I} \models UP + SizeGlobals + 8 \doteq BP_0$
$\mathfrak{I} \models SP \leq_I HP \doteq true$
$\mathfrak{I} \models \neg CT$ **is** BLOCK $\Rightarrow SP \doteq BP +_I SizeLocals$
$\mathfrak{I} \models CT$ **is** BLOCK $\wedge$ $entered(pc) \doteq false \Rightarrow SP \doteq BP +_I SizeLocals$
$\mathfrak{I} \models CT$ **is** BLOCK $\wedge$ $entered(pc) \doteq true \Rightarrow SP \doteq BP +_I SizeLocals'$
$\mathfrak{I} \models isGlobal(prog, x) \doteq true \Rightarrow type(Glob(x)) \doteq gettype(deftab(prog, ()), x)$
$\mathfrak{I} \models isStruct(DefTab(type(I)), x) \Rightarrow type(I +_I FieldAddr(x)) \doteq gettype(DefTab(type(I)), x)$

wobei
$$\begin{array}{ll} BP_0 & \triangleq BP \\ BP_{i+1} & \triangleq Read(BP_i) \\ FrameSize(tab) & \triangleq maxaddr(tab) + lastsize(tab) \\ GlobAddrTab & \triangleq reladdr(prog, ()) \\ SizeLocals & \triangleq FrameSize(RelAddrTab) \\ SizeGlobals & \triangleq FrameSize(GlobAddrTab) \\ SizeLocals' & \triangleq FrameSize(reladdr(prog, first(prog, pc))) \\ Glob(x) & \triangleq UP +_I addr(GlobAddrTab, x) \\ DefTab(A) & \triangleq identifyDef(deftab(prog, ()), A) \\ \\ FieldAddr(x) & \triangleq addr(reladdr(DefTab(type(I))), x) \end{array}$$

Bemerkung

Die anderen Invarianten enthalten undefinierte Terme ($loc(o)$) bzw. existieren nicht im Anfangszustand, weil noch keine Prozedur aufgerufen wurde.

Beweis von Lemma 3.6

i. $\mathfrak{I} \models BP_1 <_I BP_0 \doteq true$

$$\begin{array}{lll} [\![BP_1]\!]_I & = [\![Read(BP_0)]\!]_I & \text{nach Definition } BP_1 \\ & = [\![UP]\!]_I & \text{Axiom } Read(BP) \doteq UP \\ & < [\![BP]\!]_I & \text{Axiom } BP \doteq UP +_I SizeGlobals +_I 8 \\ & = [\![BP_0]\!]_I & \text{nach Definition } BP_0 \end{array}$$

ii. $\mathfrak{I} \models BP_1 \doteq UP$: Schon in i. bewiesen

iii. $\mathfrak{I} \models UP + SizeGlobals + 8 \doteq BP_0$

$$\begin{array}{lll} [\![BP_0]\!]_I & = [\![BP]\!]_I & \text{nach Definition } BP_0 \\ & = [\![UP + SizeGlobals + 8]\!]_I & \text{Axiom } BP \doteq UP + SizeGlobals + 8 \end{array}$$

iv. $\mathfrak{I} \models SP \leq_I HP \doteq true$

$$\begin{array}{lll} [\![HP]\!]_I & = [\![maxaddr]\!]_I & \text{Axiom } HP \doteq maxaddr \\ & \geq [\![SP_I]\!] & \text{Axiom } a \leq_Q maxaddr \end{array}$$

v. $\mathfrak{I} \models \neg CT$ **is** BLOCK $\Rightarrow SP \doteq BP +_I SizeLocals$: Gilt weil $\mathfrak{I} \models CT$ **is** BLOCK

vi. $\mathfrak{I} \models CT$ **is** BLOCK $\wedge$ $entered(pc) \doteq false \Rightarrow SP \doteq BP +_I SizeLocals$.
Es gilt $\mathfrak{I} \models CT$ **is** BLOCK $\wedge$ $entered(pc) \doteq false$.

$$\begin{array}{lll} [\![SP]\!]_I & = [\![BP]\!]_I + 16 & \text{Axiom } SP \doteq BP +_I 16 \\ & = [\![BP]\!]_I + [\![minaddr(RelAddrTab)]\!]_I & \text{Anforderung } minaddr(e) \geq 16 \\ & = [\![BP]\!]_I + [\![SizeLocals]\!]_I & \text{keine lokalen Variablen bei Start} \end{array}$$

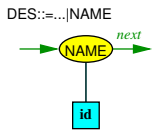
vii. $\mathfrak{I} \models CT$ **is** BLOCK $\wedge$ $entered(pc) \doteq true \Rightarrow SP \doteq BP +_I SizeLocals'$:
Gilt weil $\mathfrak{I} \models CT$ **is** BLOCK $\wedge$ $entered(pc) \doteq false$

viii. $\mathfrak{I} \models isGlobal(prog, x) \doteq true \Rightarrow type(Glob(x)) \doteq gettype(deftab(prog, ()), x)$. Gilt wegen des Axioms $type(UP +_I addr(GlobAddrTab, x)) \doteq gettype(deftab(prog, ()), x)$.

ix. $\mathfrak{I} \models isStruct(DefTab(type(I)), x) \Rightarrow type(I +_I FieldAddr(x)) \doteq gettype(DefTab(type(I)), x)$.
Gilt wegen des Axioms
$D(type(I)) \wedge isStruct(type(I)) \Rightarrow type(I + FieldAddr(x)) \doteq gettype(DefTab(type(I)), x)$.

Zugriffspfade: Variablenzugriff

Zustandsübergangsregel bei C--



if CT **is** NAME **then**
  $loc(pc) := Loc$
  $Coerce(val(pc), mem(Loc), pc)$
  $Proceed$

$Env \triangleq env{+}{+}globenv(prog)$
$Loc \triangleq bind(Env, id(pc, prog))$
$Pri(p) \triangleq pri(prog, p)$
$Post(p) \triangleq post(prog, p)$

$Coerce(src, dest, p) \triangleq dest := coerce(src, Pri(p), Post(p))$

Übergangsregel mit DEC-Alpha-Semantik

if CT **is** NAME **then**
  if $isGlobal(prog, Id)$ **then**
    $loc(pc) := Glob_\alpha$
  else $loc(pc) := Loc_\alpha$
  $Coerce_\alpha(val(pc), Read(Loc_\alpha), pc)$
  $Proceed$

$Id \triangleq id(pc, prog)$
$Glob_\alpha \triangleq UP +_I addr_\alpha(GlobAddrTab, Id)$
$Loc_\alpha \triangleq BP +_I addr_\alpha(RelAddrTab, Id)$
$Coerce_\alpha(src, dest, p) \triangleq dest := convert(src, Pri(p), Post(p))$

Lemma 3.7 (Korrektheit der Zugriffspfade)

Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE^M_α-Algebra mit $\mathfrak{A} \models CT$ **is** NAME und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B}' \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$, $\mathfrak{B}' \triangleq \phi(\mathfrak{A})$ gilt: $\mathfrak{B}' = \phi(\mathfrak{A}')$

Beweis

(1) $Update(\mathfrak{B}) = \{\ loc(pc) := Loc, pc := next(prog, pc)$
$val(pc) := coerce(mem(Loc), Pri(Pc), Post(Pc))\}$

(2) Falls $\mathfrak{A} \models IsGlobal(prog, Id) \doteq true$, ist
$Update(\mathfrak{A}) = \{\ loc(pc) := Glob_\alpha, pc := next(prog, pc)$
$val(pc) := convert(Read(Glob_\alpha), Pri(Pc), Post(Pc))\}$

(3) Falls $\mathfrak{A} \models IsGlobal(prog, Id) \doteq false$, ist
$Update(\mathfrak{A}) = \{\ loc(pc) := Loc_\alpha, pc := next(prog, pc)$
$val(pc) := convert(Read(Loc_\alpha), Pri(Pc), Post(Pc))\}$

(4) Falls $\mathfrak{A} \models isGlobal(prog, Id) \doteq true$ gilt, ist $[\![Loc]\!]_B = [\![Glob_\alpha]\!]_A$:

$[\![Loc]\!]_B$		$Loc \triangleq bind(Env, Id)$
	$= [\![bind(Env, Id)]\!]_B$	$Env \triangleq env{+}{+}globenv(prog)$
	$= [\![bind(env{+}{+}globenv(prog)), Id]\!]_B$	Eigenschaft von ENV
	$= [\![bind(globenv(prog)), Id)]\!]_B$	
	$= [\![bind]\!]_B([\![globenv(prog)]\!]_B, [\![Id]\!]_B)$	
	$= [\![bind_\alpha]\!]_A([\![globenv(prog)]\!]_B, [\![Id]\!]_B)$	$[\![bind]\!]_B \triangleq [\![bind_\alpha]\!]_A$
	$= [\![UP]\!]_A + [\![addr]\!]_A([\![GlobAddrTab]\!]_A, [\![Id_B]\!])$	Definition $[\![bind_\alpha]\!]$
	$= [\![UP]\!]_A + [\![addr]\!]_A([\![GlobAddrTab]\!]_A, [\![Id_A]\!])$	$[\![Id_B]\!] = [\![Id_A]\!]$
	$= [\![UP +_I addr(GlobAddrTab, Id)]\!]_A$	
	$= [\![Glob_\alpha]\!]$	Def. $Glob_\alpha$

Beweis (Forts.)

(5) Falls $\mathfrak{A} \models isGlobal(prog, Id) \doteq false$ gilt, ist $[\![Loc]\!]_B = [\![Loc_\alpha]\!]_A$:

$[\![Loc]\!]_B$		$Loc \triangleq bind(Env, Id)$
	$= [\![bind(Env, Id)]\!]_B$	$Env \triangleq env{+}{+}globenv(prog)$
	$= [\![bind(env{+}{+}globenv(prog)), Id]\!]_B$	Eigenschaft von ENV
	$= [\![bind(env), Id)]\!]_B$	
	$= [\![bind]\!]_B([\![env]\!]_B, [\![Id]\!]_B)$	
	$= [\![bind_\alpha]\!]_A([\![env]\!]_B, [\![Id]\!]_B)$	$[\![bind]\!]_B \triangleq [\![bind_\alpha]\!]_A$
	$= [\![BP]\!]_A + [\![addr]\!]_A([\![RelAddrTab]\!]_A, [\![Id_B]\!])$	Definition $[\![bind_\alpha]\!]$
	$= [\![BP]\!]_A + [\![addr]\!]_A([\![RelAddrTab]\!]_A, [\![Id_A]\!])$	$[\![Id_B]\!] = [\![Id_A]\!]$
	$= [\![BP +_I addr(RelAddrTab, Id)]\!]_A$	
	$= [\![Loc_\alpha]\!]$	Def. Loc_α

(6) Falls $o \neq [\![pc]\!]_B$ ist, ist $[\![loc]\!]_{B'}(o) = [\![loc]\!]_{A'}(o)$: Folgt aus $[\![pc]\!]_A = [\![pc]\!]_B$, und (1)–(3)

(7) $[\![loc]\!]_{B'}([\![pc_B]\!]) = [\![loc]\!]_{A'}([\![pc_B]\!])$:

$[\![loc]\!]_{B'}([\![pc]\!]_B)$	$= [\![Loc]\!]_B$	(1)
	$= \begin{cases} [\![Glob_\alpha]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = true \\ [\![Loc_\alpha]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = false \end{cases}$	(4), (5)
	$= [\![loc]\!]_{A'}([\![pc]\!]_A)$	(2), (3)
	$= [\![loc]\!]_{A'}([\![pc]\!]_B)$	$[\![pc]\!]_A = [\![pc]\!]_B$

(8) Falls $[\![y]\!]_A = [\![x]\!]_B$, dann ist $[\![convert(y, t_1, t_2)]\!]_A = [\![coerce(x, t_1, t_2)]\!]_B$:

$[\![coerce(x, t_1, t_2)]\!]_B$	$= [\![coerce]\!]_B([\![x]\!]_B, [\![t_1]\!]_B, [\![t_2]\!]_B)$	
	$= [\![coerce]\!]_B([\![y]\!]_A, [\![t_1]\!]_B, [\![t_2]\!]_B)$	Voraussetzung
	$= [\![coerce]\!]_B([\![y]\!]_A, [\![t_1]\!]_A, [\![t_2]\!]_A)$	$[\![t]\!]_B = [\![t]\!]_A$ für alle $t \in \mathcal{T}_{\text{TYPE}}$
	$= [\![convert]\!]_A([\![y]\!]_A, [\![t_1]\!]_A, [\![t_2]\!]_A)$	$[\![coerce]\!]_B = [\![convert]\!]_A$
	$= [\![convert(y, t_1, t_2)]\!]_A$	

Beweis (Forts.)

(9) Falls $\mathfrak{A} \models isGlobal(prog, Id) \doteq true$, dann ist $[\![mem(Loc)]\!]_B = [\![Read(Glob_\alpha)]\!]_A$:

$[\![mem(Loc)]\!]_B$	$= [\![mem]\!]_B([\![Loc]\!]_B)$	
	$= [\![mem]\!]_B([\![Glob_\alpha]\!]_A)$	(4)
	$= [\![Read(Glob_\alpha)]\!]_A$	$[\![mem]\!]_B([\![I]\!]_A) = [\![Read(I)]\!]$

(10) Falls $\mathfrak{A} \models isGlobal(prog, Id) \doteq false$, dann ist $[\![mem(Loc)]\!]_B = [\![Read(Loc_\alpha)]\!]_A$:

$[\![mem(Loc)]\!]_B$	$= [\![mem]\!]_B([\![Loc]\!]_B)$	
	$= [\![mem]\!]_B([\![Loc_\alpha]\!]_A)$	(5)
	$= [\![Read(Loc_\alpha)]\!]_A$	$[\![mem]\!]_B([\![I]\!]_A) = [\![Read(I)]\!]$

(11) Falls $o \neq [\![pc]\!]_B$ ist, ist ist $[\![val]\!]_{B'}(o) = [\![val]\!]_{A'}(o)$: Folgt aus $[\![pc]\!]_A = [\![pc]\!]_B$, und (1)–(3)

(12) $[\![val]\!]_{B'}([\![pc_B]\!]) = [\![val]\!]_{A'}([\![pc_B]\!])$:

$[\![val]\!]_{B'}([\![pc]\!]_B)$		
	$= [\![coerce(mem(Loc), Pri(pc), Post(pc))]\!]_B$	(1)
	$= \begin{cases} [\![convert(Read(Glob_\alpha), Pri(pc), Post(pc))]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = true \\ [\![convert(Read(Loc_\alpha), Pri(pc), Post(pc))]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = false \end{cases}$	(8)–(10)
	$= [\![val]\!]_{A'}([\![pc]\!]_A)$	(2), (3)
	$= [\![val]\!]_{A'}([\![pc]\!]_B)$	$[\![pc]\!]_A = [\![pc]\!]_B$

(13) $[\![pc]\!]_{B'} = [\![pc]\!]_{A'}$:

$[\![pc_{B'}]\!]$	$= [\![next(prog, pc)]\!]_B$	(1)
	$= [\![next(prog, pc)]\!]_B$	$[\![pc]\!]_B = [\![pc]\!]_A$, $[\![f]\!]_B = [\![f]\!]_A$ für Funktionen auf Syntax
	$= [\![pc]\!]_{A'}$	(2), (3)

(14) Es gilt $\mathfrak{B}' = \phi(\mathfrak{A}')$: (1)–(3), (6)–(7), (11)–(12)und (13)

Beweis (Forts.)

(15) Falls $o \neq [\![pc]\!]_{A'}$ und $[\![loc]\!]_{A'}(o) \neq null$ ist, gilt $[\![loc]\!]_{A'(o)} \geq [\![UP]\!]_{A'}$: Folgt aus (2) und (3)

(16) Falls $[\![loc]\!]_{A'}([\![pc]\!]_A) \neq null$ ist, gilt $[\![loc]\!]_{A'}(pc_A) \geq [\![UP]\!]_{A'}$:

$$
\begin{array}{lll}
& [\![loc]\!]_{A'}([\![pc]\!]_A) & \\
= & \begin{cases} [\![Glob_\alpha]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = true \\ [\![Loc_\alpha]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = false \end{cases} & (2),\ (3) \\
= & \begin{cases} [\![UP]\!]_A + [\![addr(GlobAddrTab, Id)]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = true \\ [\![BP]\!]_A + [\![addr(RelAddrTab, Id)]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = false \end{cases} & \text{Defin. } Glob_\alpha,\ Loc_\alpha \\
\geq & \begin{cases} [\![UP]\!]_A + [\![addr(GlobAddrTab, Id)]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = true \\ [\![UP]\!]_A + [\![addr(RelAddrTab, Id)]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = true \end{cases} & \mathfrak{A} \models \text{INV} \\
\geq & \begin{cases} [\![UP]\!]_A + [\![minaddr(GlobAddrTab)]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = true \\ [\![UP]\!]_A + [\![minaddr(RelAddrTab)]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = false \end{cases} & \text{Axiom für } minaddr \\
\geq & [\![UP]\!]_A & \text{Anforderung 3.2} \\
= & [\![UP]\!]_{A'} & (2), (3)
\end{array}
$$

(17) Falls $o \neq [\![pc]\!]_A$, $[\![loc]\!]_{A'}(o) < [\![BP]\!]_{A'} + 16$ und $[\![IsField]\!]_{A'}(o) = false$ ist, dann ist $[\![loc]\!]_{A'}(o) + [\![lastsize(GlobAddrTab)]\!]_{A'} \leq [\![BP_{n-1}]\!]_{A'}$: Folgt aus (2) und (3)

(18) Falls $[\![isGlobal(prog, Id)]\!]_A \doteq false$ ist $[\![loc]\!]_{A'}([\![pc]\!]_A) \geq [\![BP]\!]_{A'} + 16$:

$$
\begin{array}{lll}
[\![loc]\!]_{A'}([\![pc]\!]_A) & = [\![Loc_\alpha]\!]_A & (3) \\
& = [\![BP]\!]_A + [\![addr(RelAddrTab, Id)]\!]_A & \text{Definition } Loc_\alpha \\
& \geq [\![BP]\!]_A + [\![minaddr(RelAddrTab]\!]_A & \text{Axiom } minaddr \\
& \geq [\![BP]\!]_A + 16 & \text{Anforderung 3.2} \\
& = [\![BP]\!]_{A'} + 16 & (2), (3)
\end{array}
$$

(19) Falls $[\![isGlobal(prog, Id)]\!]_A \doteq false$, $[\![loc]\!]_{A'}([\![pc]\!]_A) < [\![BP]\!]_{A'} + 16$ und $[\![IsField]\!]_{A'}([\![pc]\!]_A) = false$ ist, dann ist $[\![loc]\!]_{A'}([\![pc]\!]_A) + [\![lastsize(GlobAddrTab)]\!]_{A'} \leq [\![BP_{n-1}]\!]_{A'}$: folgt aus (18)

Beweis (Forts.)

(20) Falls $[\![isGlobal(prog, Id)]\!]_A \doteq true$ ist, ist $[\![loc]\!]_{A'}([\![pc]\!]_A) + [\![lastsize(GlobAddrTab)]\!]_A \leq [\![BP_{n-1}]\!]$:

$$
\begin{array}{lll}
& [\![loc]\!]_{A'}([\![pc]\!]_A) + [\![lastsize(GlobAddrTab)]\!]_A & \\
& [\![Glob_\alpha]\!]_A + [\![lastsize(GlobAddrTab)]\!]_A & (2) \\
= & [\![UP]\!]_A + [\![addr(GlobAddrTab, Id)]\!]_A + [\![lastsize(GlobAddrTab)]\!]_A & \text{Def. } Glob_\alpha \\
\leq & [\![UP]\!]_A + [\![maxaddr(GlobAddrTab)]\!]_A + [\![lastsize(GlobAddrTab)]\!]_A & \text{Axiom } maxaddr \\
= & [\![UP]\!]_A + [\![FrameSize(GlobAddrTab)]\!]_A & \text{Def. } FrameSize \\
= & [\![UP]\!]_A + [\![SizeGlobals]\!]_A & \text{Def. } SizeGlobals \\
= & [\![BP]\!]_A^{n-1} & \mathfrak{A} \models \text{INV} \\
= & [\![BP]\!]_{A'}^{n-1} & (2)
\end{array}
$$

(21) Falls $o \neq [\![pc]\!]_A$, $[\![loc]\!]_{A'}(o) < [\![BP]\!]_{A'} + 16$ und $[\![occ]\!]_{A'}([\![prog]\!]_{A'}, o) \in [\![\text{FIELD}]\!]_{A'}$, dann ist $[\![loc]\!]_{A'}(o) + [\![lastsize(GlobAddrTab)]\!]_{A'} \leq [\![BP_{n-1}]\!]_{A'}$: Folgt aus (2) und (3).

(22) $[\![occ]\!]_{A'}([\![prog]\!]_{A'}, [\![pc]\!]_A) \notin [\![\text{FIELD}]\!]_{A'}$: weil $[\![pc.0]\!]_A \in [\![\text{ID}]\!]_A$

(23) Falls $o \neq [\![pc]\!]_A$, $[\![loc]\!]_{A'}(o) + lastsize(RelAddrTab) > [\![SP]\!]_{A'}$, dann ist $[\![loc]\!]_{A'}(o) \geq [\![HP]\!]_{A'}$: Folgt aus (2) und (3).

(24) Falls $[\![isGlobal(prog, Id)]\!]_A = true$ ist, ist $[\![loc]\!]_{A'}([\![pc]\!]_{A'}) + [\![lastsizeRelAddrTab]\!]_{A'} \leq [\![SP]\!]_{A'}$:

$$
\begin{array}{lll}
& [\![loc]\!]_{A'}([\![pc]\!]_{A'}) + [\![lastsizeRelAddrTab]\!]_{A'} & \\
\leq & [\![BP_{n-1}]\!]_{A'} + [\![lastsizeRelAddrTab]\!]_{A'} & (20) \\
= & [\![BP_{n-1}]\!]_A + [\![lastsizeRelAddrTab]\!]_A & (2),\ (3) \\
< & [\![BP]\!]_A + [\![lastsizeRelAddrTab]\!]_A & \mathfrak{A} \models \text{INV} \\
\leq & [\![BP]\!]_A + [\![maxaddr]\!]_A(RelAddrTab) + [\![lastsizeRelAddrTab]\!]_A & Axiome \\
= & [\![BP]\!]_A + [\![SizeLocals]\!]_A & \text{Def. } SizeLocals \\
= & [\![SP]\!]_A & \mathfrak{A} \models \text{INV} \\
= & [\![SP]\!]_{A'} & (2)
\end{array}
$$

Beweis (Forts.)

(25) Falls $[\![isGlobal(prog, Id)]\!]_A = false$ ist, ist $[\![loc]\!]_{A'}([\![pc]\!]_{A'}) + [\![lastsizeRelAddrTab]\!]_{A'} \leq [\![SP]\!]_{A'}$:

$$
\begin{array}{lll}
& [\![loc]\!]_{A'}([\![pc]\!]_{A'}) + [\![lastsizeRelAddrTab]\!]_{A'} & \\
= & [\![Loc_\alpha]\!]_A & (3) \\
= & [\![BP]\!]_A + [\![addr(RelAddrTab, Id)]\!]_A + [\![lastsizeRelAddrTab]\!]_A & \text{Def. } Loc_\alpha \\
< & [\![BP]\!]_A + [\![maxaddr]\!]_A(RelAddrTab) + [\![lastsizeRelAddrTab]\!]_A & Axiome \\
\leq & [\![SP]\!]_{A'} & \text{analog (24)}
\end{array}
$$

(26) Aus $[\![o]\!]_{A'} \neq [\![pc]\!]_A$, $[\![InExpr(o)]\!]_{A'} = true$, $[\![IsClassDes(o)]\!]_{A'} = true$ und $[\![val(o)]\!]_{A'} < [\![HP]\!]'_A$ folgt $val(o) = null$: Folgt aus (2) und (3).

(27) Aus $[\![o]\!]_{A'} \neq [\![pc]\!]_A$, $[\![InAssign(o)]\!]_{A'} = true$, $[\![IsClassDes(o)]\!]_{A'} = true$ und $[\![val(o)]\!]_{A'} < [\![HP]\!]'_A$ folgt $val(o) = null$: Folgt aus (2) und (3).

(28) Falls $[\![IsClassDes(pc)]\!]_A = true$ und $[\![isGlobal(prog, Id)]\!]_A = true$ ist, ist $[\![isClass(type(Glob_\alpha))]\!]_A = true$

$$
\begin{array}{lll}
& [\![isClass(type(Glob_\alpha))]\!]_A & \\
= & [\![isClass(type(UP +_l addr(GlobAddrTab, Id)))]\!]_A & \text{Def. } Glob_\alpha \\
= & [\![isClass(type(Glob(x)))]\!]_A & \text{Def. } Glob \\
= & [\![isClass(gettype(deftab(prog, ())), Id))]\!]_A & \mathfrak{A} \models \text{INV und Voraussetzung} \\
= & [\![isClass(pri(prog, pc))]\!]_A & \text{Attributierung} \\
= & true & \text{Voraussetzung}
\end{array}
$$

(29) Falls $[\![IsClassDes(pc)]\!]_A = true$ und $[\![isDefined(deftab(prog, pc), Id)]\!]_A = false$ ist, ist $[\![isClass(type(Loc_\alpha))]\!]_A = true$

$$
\begin{array}{lll}
& [\![isClass(type(Loc_\alpha))]\!]_A & \\
= & [\![isClass(type(BP +_l addr(RelAddrTab, Id)))]\!]_A & \text{Def. } Loc_\alpha \\
= & [\![isClass(type(Loc_0(x)))]\!]_A & \text{Def. } Loc_0,\ BP_0 \\
= & [\![isClass(gettype(Proc_0, Id))]\!]_A & \mathfrak{A} \models \text{INV und Voraussetzung} \\
= & [\![isClass(pri(prog, pc))]\!]_A & \text{Attributierung} \\
= & true & \text{Voraussetzung}
\end{array}
$$

Beweis (Forts.)

(30) Falls $[\![isGlobal(prog, Id)]\!]_A = true$ und $[\![Read(Glob_\alpha)]\!]_A < [\![HP]\!]_A$ ist, gilt $[\![Read(Glob_\alpha)]\!]_A = null$:

$$
\begin{array}{ll}
[\![isClass(type(Glob_\alpha))]\!]_A = true & (28) \\
[\![Read(Glob_\alpha)]\!]_A = null & \mathfrak{A} \models \text{INV}
\end{array}
$$

(31) Falls $[\![isGlobal(prog, Id)]\!]_A = false$ und $[\![Read(Loc_\alpha)]\!]_A < [\![HP]\!]_A$ ist, gilt $[\![Read(Loc_\alpha)]\!]_A = null$:

$$
\begin{array}{ll}
[\![isDefined(deftab(prog, pc), Id)]\!]_A = true & \text{Attributierung} \\
[\![isClass(type(Loc_\alpha))]\!]_A = true & (29) \\
[\![Read(Glob_\alpha)]\!]_A = null & \mathfrak{A} \models \text{INV}
\end{array}
$$

(32) Sei $[\![o]\!]_{A'} = [\![pc]\!]_A$, $[\![isGlobal(prog, Id)]\!]_A = true$ und $[\![val(o)]\!]_{A'} < [\![HP]\!]_{A'}$. Dann gilt: $[\![Read(Glob_\alpha)]\!]_A < [\![HP]\!]_A$: $[\![Read(Glob_\alpha)]\!]_A = [\![val(o)]\!]_{A'}$ (2); $< [\![HP]\!]_{A'}$ Voraussetzung; $= [\![HP]\!]_A$ (2)

(33) Sei $[\![o]\!]_{A'} = [\![pc]\!]_A$, $[\![isGlobal(prog, Id)]\!]_A = false$ und $[\![val(o)]\!]_{A'} < [\![HP]\!]_{A'}$. Dann gilt: $[\![Read(Loc_\alpha)]\!]_A < [\![HP]\!]_A$: $[\![Read(Loc_\alpha)]\!]_A = [\![val(o)]\!]_{A'}$ (3); $< [\![HP]\!]_{A'}$ Voraussetzung; $= [\![HP]\!]_A$ (3)

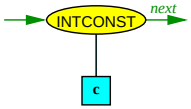
(34) Sei $[\![o]\!]_{A'} = [\![pc]\!]_A$, $[\![IsClassDes(pc)]\!]_{A'} = true$ und $[\![val(o)]\!]_{A'} < [\![HP]\!]_{A'}$. Dann ist $[\![val(o)]\!]_{A'} = null$:

$$
[\![val(o)]\!]_{A'} = \begin{cases} [\![Read(Glob_\alpha)]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = true \\ [\![Read(Loc_\alpha)]\!]_A & \text{falls } [\![isGlobal(prog, Id)]\!]_A = false \\ = null & \end{cases} \quad \begin{array}{l} (2),\ (3) \\ (30) - (33) \end{array}
$$

(35) $\mathfrak{A}' \models \text{INV}$: (2)–(3), (15)–(16), (17), (19), (20)–(21), (22), (23)–(25), (26), (27), (34)

Konstanten

Semantik auf C--



if CT **is** CONST **then**
$Coerce(val(pc), const(pc, prog), pc)$
$Proceed$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** CONST **then**
$Coerce_\alpha(val(pc), const(pc, prog), pc)$
$Proceed$

Lemma 3.8 (Korrektheit der Konstantenzugriffe)

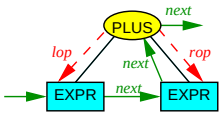
Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** CONST und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Binäre Ausdrücke

Zustandsübergangsregel bei C--



if CT **is** PLUS **then**
$Coerce(val(pc), Sum, pc)$
$Proceed$

$Lop \triangleq val(lop(pc, prog))$
$Rop \triangleq val(rop(pc, prog))$
$Sum \triangleq Lop +_I Rop$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** PLUS **then**
$Coerce_\alpha(val(pc), Sum_\alpha, pc)$
$Proceed$

$Sum_\alpha \triangleq intplus(Lop, Rop)$

Lemma 3.9 (Korrektheit der Addition)

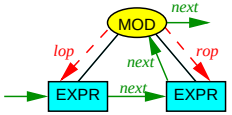
Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** PLUS und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Binäre Ausdrücke: Divisionsoperatoren

Zustandsübergangsregel bei C--



if CT **is** MOD **then**
  if $Rop \doteq 0$ **then**
    $exc := divByZero$
    $pc := handle(divByZero, pc)$
  else $Coerce(val(pc), Rem, pc)$
    $Proceed$

$Rem \triangleq mod(Lop, Rop)$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** MOD **then**
  if $Rop \doteq 0$ **then**
    $fpcr := L0^9 LO^{53}$
    $pc := handle(L0^9 LO^{53}, pc)$
  else $Coerce_\alpha(val(pc), Rem_\alpha, pc)$
    $Proceed$

$Rem_\alpha \triangleq Lop\%_I Rop$

Lemma 3.10 (Korrektheit der Restberechnung)

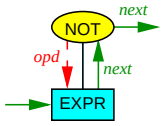
Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** MOD und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Unäre Ausdrücke

Zustandsübergangsregel bei C--



if CT **is** NOT **then**
$val(pc) := not(Opd)$
$Proceed$

$Opd \triangleq val(opd(pc, prog))$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** NOT **then**
$val(pc) := not_\alpha(Opd)$
$Proceed$

Lemma 3.11 (Korrektheit der Negation)

Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** NOT und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Zuweisung atomarer Werte

Zustandsübergangsregel bei C--

if CT **is** ASSIGN $\wedge$ $isAtomic(Pri(lhs))$ **then**
  $mem(loc(Lhs)) := val(Rhs)$
  $Proceed$

$Lhs \triangleq lhs(prog, pc)$
$Rhs \triangleq lhs(prog, pc)$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** ASSIGN $\wedge$ $isAtomic(Pri(lhs))$ **then**
  $Store(loc(Lhs), val(Rhs))$
  $Proceed$

$$Store(a, v) \triangleq \begin{array}{ll} mem(a) & := selbyte(v, 7) \\ & \vdots \\ mem(a+7) & := selbyte(v, 0) \end{array}$$

Lemma 3.12 (Korrektheit der Zuweisung atomarer Werte)

Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** ASSIGN $\wedge$ $isAtomic(Pri(lhs))$ und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Lese-Anweisung

Zustandsübergangsregel bei C--

if CT **is** READ **then**
  $mem(loc(Dest)) := head(inp)$
  $inp := tail(inp)$
  $Proceed$

$Dest \triangleq dest(opd(pc, prog))$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** READ **then**
  $Store(loc(Dest), head_\alpha(inp_\alpha))$
  $inp_\alpha := tail_\alpha(inp_\alpha)$
  $Proceed$

Lemma 3.13 (Korrektheit der Leseanweisung)

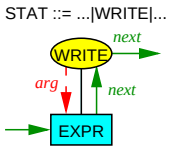
Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** READ und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Schreibe-Anweisung

Zustandsübergangsregel bei C--



if CT **is** WRITE **then**
  $out := out.val(Arg)$
  $Proceed$

$Arg \triangleq arg(opd(pc, prog))$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** WRITE **then**
  $out_\alpha := out_\alpha +\!\!+_\alpha val(Arg)$
  $Proceed$

Lemma 3.14 (Korrektheit der Schreibeanweisung)

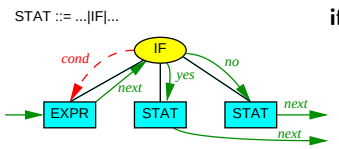
Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** WRITE und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Verzweigung

Zustandsübergangsregel bei C--



if CT **is** IF **then**
  if $val(Cond) \doteq true$ **then**
    $pc := yes(pc, prog)$
  else $pc := no(pc, prog)$

$Cond \triangleq cond(opd(pc, prog))$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** IF **then**
  if $quadtobool(val(Cond)) \doteq true$ **then**
    $pc := yes(pc, prog)$
  else $pc := no(pc, prog)$

Lemma 3.15 (Korrektheit der Verzweigung)

Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** IF und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung Schleifen behandelt man analog

Break-Anweisung

Zustandsübergangsregel bei C--

STAT ::= ...|BREAK|...

if *CT* **is** BREAK **then**
  pc := *break*(*pc*, *prog*)

Zustandsübergangsregel in DEC-Alpha-Semantik

if *CT* **is** BREAK **then**
  pc := *break*(*pc*, *prog*)

Lemma 3.16 (Korrektheit der Break-Anweisung)

Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE^M_α-Algebra mit $\mathfrak{A} \models$ *CT* **is** BREAK und $\mathfrak{A} \models$ *Inv* erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models$ *Inv* und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

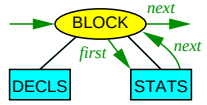
Beweis

Übung Continue behandelt man analog

Block

Zustandsübergangsregeln bei C--

STAT ::= ...|BLOCK|...



if *CT* **is** BLOCK $\wedge$ *entered*(*pc*) $\doteq$ *false* **then**
  AllocateVars(*locVars*(*prog*, *pc*))
  entered(*pc*) := *true*
  pc := *first*(*prog*, *pc*)
if *CT* **is** BLOCK $\wedge$ *entered*(*pc*) $\doteq$ *true* **then**
  DeAllocateVars(*locVars*(*prog*, *pc*))
  entered(*pc*) := *false*
  Proceed

AllocateVars(*ids*) $\triangleq$ **choose** *e* : ENV • *Good*(*e*, *ids*) **do** *env* := *e*++*env*
DeAllocateVars(*ids*) $\triangleq$ *env* := *env* \ *ids*

Zustandsübergangsregeln in DEC-Alpha-Semantik

if *CT* **is** BLOCK $\wedge$ *entered*(*pc*) $\doteq$ *false* **then**
  if *SP* $+_I$ *inttoquad*(*SizeLocals*) $\geq_I$ *HP* **then**
    fpcr := $L0^{63}$
    pc := 2
  else *SP* := *SP* $+_I$ *inttoquad*(*SizeLocalsBlock*)
    entered(*pc*) := *true*
    TypeAddrs(*locVars*(*prog*, *pc*), *RelAddrTab*′, *Deftab*′, *BP*)
    pc := *first*(*prog*, *pc*)

if *CT* **is** BLOCK $\wedge$ *entered*(*pc*) $\doteq$ *true* **then**
  SP := *SP* $-_I$ *inttoquad*(*SizeLocalsBlock*)
  UntypeAddrs(*SP*, *inttoquad*(*SizeLocalsBlock*))
  entered(*pc*) := *false*
  Proceed

RelAddrTab′ $\triangleq$ *reladdr*(*prog*, *first*(*prog*, *pc*))
FrameSize(*tab*) $\triangleq$ *maxaddr*(*tab*) + *lastsize*(*tab*)
SizeLocalsBlock $\triangleq$ *FrameSize*(*RelAddrTab*′) $-_I$ *minaddr*(*RelAddrTab*′, *locVars*(*prog*, *pc*)) $+_I$ 1
Deftab′ $\triangleq$ *deftab*(*prog*, *first*(*prog*, *pc*))
TypeAddrs(*v*, *r*, *d*, *b*) $\triangleq$ **forall** *x* : ID • *x* $\in$ *v* **do**
  type(*b* $+_I$ *addr*(*r*, *x*)) := *gettype*(*d*, *x*)
  if *isStruct*(*gettype*(*d*, *x*)) **then**
    TypeFields(*gettype*(*d*, *x*), *GlobAddrTab*, *b* $+_I$ *addr*(*r*, *x*))
TypeFields(*t*, *tab*, *b*) $\triangleq$ *TypeAddrs*(*fields*(*t*), *getaddrs*(*tab*, *t*), *getfielddefs*(*tab*, *t*), *b*)
UntypeAddrs(*u*, *n*) $\triangleq$ **forall** *a* : QUAD • *u* $-_I$ *n* $\leq_I$ *a* $\doteq$ *true* $\wedge$ *a* $<_I$ *u* $\doteq$ *true* **do**
  type(*a*) := *undef*

Korrektheit der Block-Anweisung

Lemma 3.17 (Korrektheit der Blockausführung)

Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE^M_α-Algebra mit $\mathfrak{A} \models$ *CT* **is** BLOCK und $\mathfrak{A} \models$ *Inv* erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models$ *Inv* und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$. oder $[\![fpcr]\!]_{A'} = [\![LO^{63}]\!]_{A'}$

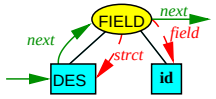
Beweis

Übung

Verbund– oder Klassenfeldzugriff

Zustandsübergangsregel bei C--

Des::=...|Field|...



StrctLoc $\triangleq$ *loc*(*strct*(*prog*, *pc*))
ClassLoc $\triangleq$ *mem*(*StrctLoc*)
FldLoc(*l*) $\triangleq$ *floc*(*l*, *field*(*prog*, *pc*))
Strct(*d*, *v*, *o*) $\triangleq$ **if** *isAtomic*(*Pri*(*pc*)) **then** *Proceed*

if *CT* **is** FIELD **then**
  if *isClass*(*Pri*(*strct*(*prog*, *pc*))) **then**
    if *loc*(*pc*) = *null* **then**
      exc := *nullPointerDeref*
      pc := *handle*(*nullPointerDeref*, *pc*)
    else *loc*(*pc*) := *FldLoc*(*ClassLoc*)
      Strct(*val*(*pc*), *mem*(*FldLoc*(*ClassLoc*)), *pc*)
      Proceed
  else
    loc(*pc*) := *FldLoc*(*StrctLoc*)
    Coerce(*val*(*pc*), *mem*(*FldLoc*(*StrctLoc*)), *pc*)
    Proceed

Zustandsübergangsregel in DEC-Alpha-Semantik

if *CT* **is** FIELD **then**
  if *isClass*(*Pri*(*strct*(*prog*, *pc*))) **then**
    if *loc*(*pc*) = *null* **then**
      fpcr := $L0^{63}$
      pc := *handle*(*nullPointerDeref*, *pc*)
    else *loc*(*pc*) := $FldLoc_\alpha(ClassLoc_\alpha)$
      $Strct_\alpha$(*val*(*pc*), *Read*(*FldLoc*(*ClassLoc*))), *pc*)
      Proceed
  else *loc*(*pc*) := $FldLoc_\alpha(StrctLoc)$
    $Strct_\alpha$(*val*(*pc*), *mem*($FldLoc_\alpha$(*StrctLoc*)), *pc*)
    Proceed

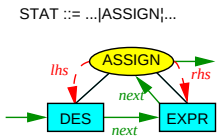
$ClassLoc_\alpha$ $\triangleq$ *Read*(*StrctLoc*)
$FldLoc_\alpha(l)$ $\triangleq$ *l* $+_I$ *reladdr*(*RelAddrTab*, *Id*)
$Strct_\alpha(d, v, o)$ $\triangleq$ **if** *isAtomic*(*Pri*(*pc*)) **then** $Coerce_\alpha(v, l, o)$

Lemma 3.18 (Korrektheit der Feldzugriffe)

Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE^M_α-Algebra mit $\mathfrak{A} \models$ *CT* **is** FIELD und $\mathfrak{A} \models$ *Inv* erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models$ *Inv* und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Zuweisung nicht-atomarer Werte

Zustandsübergangsregel bei C--



if CT **is** ASSIGN $\wedge$ $isAtomic(Pri(Lhs)) \doteq false$ **then**
  $StructCopy(Lhs, Rhs)$
  $Proceed$

$Lhs \triangleq lhs(prog, pc)$
$Rhs \triangleq rhs(prog, pc)$

$StructCopy(dest, src) \triangleq$ **forall** x : ID • $isField(Pri(dest), x)$ **do**
  if $isAtomic(gettype(fields(Pri(dest), x))$ **then**
    $mem(FldLoc(loc(dest))) := mem(FldLoc(loc(src)))$
  else $StructCopy(FldLoc(loc(dest)), FldLoc(loc(src)))$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** ASSIGN $\wedge$ $isAtomic(Pri(Lhs)) \doteq false$ **then**
  $StructCopy_\alpha(Lhs, Rhs)$
  $Proceed$

$StructCopy_\alpha(dest, src) \triangleq$ **forall** x : ID • $isField(Pri(dest), x)$ **do**
  if $isAtomic(gettype(fields(Pri(dest), x)))$ **then**
    $Store(FldLoc(loc(dest)), Read(FldLoc(loc(src))))$
  else $StructCopy_\alpha(FldLoc(loc(dest)), FldLoc(loc(src)))$

Zuweisung nicht-atomarer Werte

Lemma 3.19 (Korrektheit der Zuweisung nichtatomarer Werte)

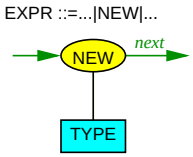
Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** ASSIGN $\wedge$ $isAtomic(Pri(lhs)) \doteq false$ und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Erzeugung von Objekten

Zustandsübergangsregel bei C--



if CT **is** NEW **then**
  choose l : LOC • $IsUsed(l) \doteq false$ **do**
    $loc(pc) := l$
  $Proceed$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** NEW **then**
  if $HP -_l FieldsSize <_l SP \doteq true$ **then**
    $fpcr := L0^{63}$
    $pc := 2$
  else $HP := HP -_l FieldsSize$
    $TypeFields(Pri(pc), GlobAddrTab, HP -_l FieldsSize)$
    $pc := break(pc, prog)$
    $Proceed$

$FieldsSize \triangleq FrameSize(reladdr(prog, pc.0))$

Lemma 3.20 (Korrektheit der New-Anweisung)

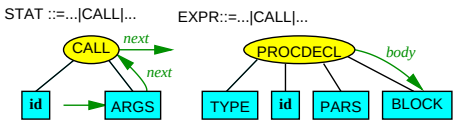
Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** NEW und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Prozeduraufruf

Zustandsübergangsregel bei C--



if CT **is** CALL **then**
  $Pass(pars(CurProc))$
  $procstack := push(procsstack, Frame)$
  $pc := body(CurProc)$

$Frame \triangleq mkframe(env, pc, save(exproccs(pc)))$
$CurProc \triangleq identifyDef(deftab(prog, pc), occ(prog, pc.0))$
$Par(i) \triangleq get(ids, i)$
$Arg(i) \triangleq arg(prog, pc, i)$

$Pass(ids) \triangleq$ **choose** e : ENV • $Good(e, ids)$ **do**
  $env := e$
  forall i : INT • $0 \leq i < length(ids)$ **do**
    if $isAtomic(Post(Arg(i))) \doteq true$ **then**
      $Coerce(mem(bind(e, Par(i))), val(Arg(i)), Arg(i))$
    else $StructCopy(bind(e, x), Arg(i))$

Prozeduraufruf: Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** CALL **then**
  if $SP +_I Incr \geq_I HP \doteq true$ **then**
    $fpcr := L0^{63}$
    $pc := 2$
  else $Pass_\alpha(pars(CurProc))$
    $Save(exprprocs(pc)$
    $SP := SP +_I Incr$
    $pc := body(CurProc)$
    $Store(SP +_I ExprSize, BP)$
    $Store(Rel(SP +_I ExprSize, 8), occtoquad(pc))$
    $TypeAddrs($

$ExprSize \triangleq inttoquad(length(exprprocs(pc) * 16)$
$ParsSize \triangleq FrameSize(reladdr(prog, pc.0)$
$Rel(a, i) \triangleq a + inttoquad(i)$
$Incr \triangleq Rel(ExprSize, ParsSize + 16)$
$RelAddrTab \triangleq reladdr(prog, CurProc)$
$Occ(i) \triangleq occtoquad(get(occs, i))$
$Addr(i) \triangleq Rel(SP +_I ExprSize, addr(CurProc, Par(i)))$

$Save(occs) \triangleq$ **forall** i : INT $\bullet 0 \leq i < length(occs)$ **do**
  $Store(Rel(SP, i * 16), OCC(i))$
  $Store(Rel(SP, i * 16 + 8), val(OCC(i)))$

$Pass_\alpha(ids) \triangleq$ **forall** i : INT $\bullet 0 \leq i < length(ids)$ **do**
  $type(Addr(i)) := Post(Arg(i))$
  if $isAtomic(Post(Arg(i))) \doteq true$ **then**
    $Coerce_\alpha(Addr(i), val(Arg(i)), Arg(i))$
  else $StructCopy(Addr(i), Arg(i))$
    $TypeFields(Post(Arg(i)), GlobAddrTab, Addr(i))$

Prozeduraufrufe

Lemma 3.21 (Korrektheit der Prozeduraufrufe)

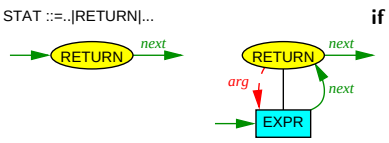
Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** CALL und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Prozedurrückkehr

Zustandsübergangsregel bei `C--`



$OldPc \triangleq getpc(procstack)$
$OldEnv \triangleq getenv(procstack)$
$OldVals \triangleq getval(procstack)$
$Arg \triangleq arg(prog, pc)$

if CT **is** RETURN $\vee \neg D(CT)$ **then**
  $pc := next(prog, OldPc)$
  $env := OldEnv$
  $procstack := pop(procstack)$
  forall o : OCC $\bullet$ $isElem(o, exproccs(prog, OldPc))$ **do**
    $val(o) := findval(o, OldVals)$
  if $isFunction(CurProc)$ **then**
    if $isAtomic(rettype(CurProc)) \doteq true$ **then**
      $Coerce(val(pc), val(Arg), OldPc)$
    else choose l : LOC $\bullet$ $IsUsed(l) \doteq false$ **do**
      $StructCopy(l, Arg)$
      $loc(OldPc) := l$

Zustandsübergangsregel in DEC-Alpha-Semantik

if CT **is** RETURN $\vee \neg D(CT)$ **then**
  $pc := next(prog, OldPc_\alpha)$
  $BP := Read(bp)$
  forall i : INT $\bullet 0 \leq i < exprprocs(OldPc_\alpha)$ **do**
    $val(RestOcc(i)) := RestVal(i)$
  $UntypeAddrs(SP, SP -_I BP)$
  if $isFunction(CurProc)$ **then**
    if $isAtomic(rettype(CurProc)) \doteq true$ **then**
      $Coerce_\alpha(val(pc), val(Arg), OldPc_\alpha)$
      $SP := NewSP$
    else
      $StructCopy(NewSP, Arg)$
      $loc(OldPc) := NewSP$
      $SP := Rel(NewSP, FrameSize(reladdr(prog, rettype)))$

$OldPc_\alpha \triangleq quadtoocc(Read(Rel(BP, 8)))$
$NewSP \triangleq BP - ExprSize$
$RestOcc(i) \triangleq Read(Rel(NewSP, i * 16))$
$RestVal(i) \triangleq Read(Rel(NewSP, i * 16 + 8))$
$Arg \triangleq arg(prog, pc)$

Prozedurrückkeht

Lemma 3.22 (Korrektheit der Prozedurrückkehr)

Sei $\mathfrak{A} = \langle A_S, [\![\cdot]\!]_A \rangle$ eine STATE_α^M-Algebra mit $\mathfrak{A} \models CT$ **is** RET $\vee$ $\neg D(CT)$ und $\mathfrak{A} \models Inv$ erfülle die Invarianten aus Abschnitt 3.2. Weiter sei $\mathfrak{A}' \triangleq next_{Update(\mathfrak{A})}(\mathfrak{A})$ Nachfolgezustand von $\mathfrak{A}$. Dann gilt auch $\mathfrak{A}' \models Inv$ und für $\mathfrak{B} \triangleq next_{Update(\mathfrak{B})}(\mathfrak{B})$: $\mathfrak{B}' = \phi(\mathfrak{B})$.

Beweis

Übung

Satz 3.23 (Korrektheit der DEC-Alpha-Semantik)

Die Semantik auf der DEC-Alphabasis ist eine eingeschränkte 1-1-Simulation der Semantik von C-- aus Kapitel 2