

Rheinisch-Westfälische Technische Hochschule Aachen

Lehr- und Forschungsgebiet Informatik II

Prof. Dr. J. Giesl

Seminar "Verifikation von Programmen"

Java-Verifikation mit *Isabelle*

Achim Lücking

Wintersemester 2001/2002

Überblick

- Kurze Einführung in die Logik von *Isabelle*
- Formalisierung der Sprache Java in der Logik
 - Modellierung eines Java-Programms
 - Modellierung des Typsystems von Java
 - Betrachtung der Wohldefiniertheit von Java-Programmen
 - Beschreibung einer operationellen Semantik für Java
- Ziel: Nachweis der Typkorrektheit von Java
- Ausblick auf Möglichkeiten der Bytecode-Verifikation

Kurzeinführung in *Isabelle*

- Verwendung einer higher-order-logic (HOL)
- Kombination aus funktionaler und logikbasierter Programmierung
- Typsystem vergleichbar mit dem der funktionalen Sprache "Haskell"
- Möglichkeit von rekursiven Datentypen und Funktionen sowie induktiv definierten Mengen

Funktionen auf Listen

app :: $\alpha list \rightarrow \alpha list \rightarrow \alpha list$

length :: $\alpha list \rightarrow nat$

set :: $\alpha list \rightarrow \alpha set$

map :: $(\alpha \rightarrow \beta) \rightarrow \alpha list \rightarrow \beta list$

zip :: $\alpha list \rightarrow \beta list \rightarrow (\alpha \times \beta)list$

nodups :: $\alpha list \rightarrow bool$

nth :: $\alpha list \rightarrow nat \rightarrow \alpha$

Optionale Werte

datatype α *option* = None | Some α

consts the :: α *option* \rightarrow α

primrec the (Some x) = x

 the (None) = \square_{α}

- \square_{α} beliebiger Wert, abhängig von α
- vergleichbar mit Haskell-Datentyp *Maybe*

Mappings

types $\alpha \rightsquigarrow \beta = \alpha \rightarrow \beta$ *option*

- partielle Funktionen, zur Entscheidung, ob ein Eintrag definiert ist
- Verknüpfung von Namen/Adressen mit Informationen
- Verwendung z.B. für Symboltabellen und Heaps

Funktionen auf Mappings

- $\text{empty} :: \alpha \rightsquigarrow \beta$
 $\text{empty} \equiv \lambda k. \text{None}$
- $_(- \mapsto _) :: (\alpha \rightsquigarrow \beta) \rightarrow \alpha \rightarrow \beta \rightarrow (\alpha \rightsquigarrow \beta)$
 $m(x \mapsto y) \equiv \lambda k. \text{if } k = x \text{ then } (\text{Some } y) \text{ else } (m \ k)$
- $\text{map_of} :: (\alpha \rightsquigarrow \beta) \text{list} \rightarrow (\alpha \rightsquigarrow \beta)$
 $\text{map_of } [] = \text{empty}$
 $\text{map_of } ((x, y) \# l) = (\text{map_of } l)(x \mapsto y)$
- $_(-[\mapsto]_) :: (\alpha \rightsquigarrow \beta) \rightarrow \alpha \text{ list} \rightarrow \beta \text{ list} \rightarrow (\alpha \rightsquigarrow \beta)$
 $m([] [\mapsto] []) = m$
 $m(a \# as [\mapsto] b \# bs) = m(a \mapsto b)(as[\mapsto]bs)$

Formalisierung eines Java-Programms

- Programm:

types $prog = cdecl\ list$

- Klassendeklaration:

types $cdecl = cname \times cname\ option \times fdecl\ list \times mdecl\ list$

- Feld- und Methodendeklaration:

types $fdecl = vname \times ty$

types $mdecl = (mname \times ty\ list) \times ty \times mbody$

Methodenrumpfe

types $mbody = (vname \times ty)list \times stmt \times expr$

datatype $stmt$ = Skip
 | $expr$
 | $stmt; stmt$
 | $if (expr) stmt \text{ else } stmt$
 | $while (expr) stmt$

Methodenrumpfe

datatype *expr* = This
| *new cname*
| (*ty*) *expr*
| *val*
| *vname*
| *vname := expr*
| {*cname*}*expr.vname*
| {*cname*}*expr.vname := expr*
| *expr.mname*(*{ty list}**expr list*)

Typen und Werte

datatype <i>ty</i>	=	void	datatype <i>val</i>	=	Unit
		boolean			Bool <i>bool</i>
		int			Intg <i>int</i>
		NT			Null
		Class <i>cname</i>			Addr <i>loc</i>

- Beschränkung Boolean, Integer und Objektreferenzen
- jeder Typ hat einen Standardwert

Unterklassenrelation

- Menge von Klassenpaaren mit direkter Vererbung

$$\text{subclass } \Gamma \equiv \{(C, D) \mid \exists r (C, \text{Some } D, r) \in \text{set } \Gamma\}$$

- daraus abgeleitete Unterklassenrelation (C Unterklasse von D)

$$\Gamma \vdash C \preceq_C D \equiv (C, D) \in (\text{subclass } \Gamma)^*$$

- darauf basierende erweiterte Relation für Typen

$$\Gamma \vdash S \preceq T$$

Typrelationen

- Relation für Ausdrücke gleichen Typs

$$\frac{}{\Gamma \vdash T \preceq T}$$

- Relation für Nullpointer bei erwartetem Klassenreferenztyp

$$\frac{}{\Gamma \vdash \text{NT} \preceq \text{Class } C}$$

- Relation für Klassenvererbung

$$\frac{\Gamma \vdash C \preceq_C D}{\Gamma \vdash \text{Class } C \preceq \text{Class } D}$$

korrekte Typung

- Beispiel für While-Schleife:

$$\frac{E \vdash e :: \text{boolean} \quad E \vdash s \checkmark}{E \vdash \text{while } (e) \ s \ \checkmark}$$

- Feldzugriff:

$$\frac{E \vdash a :: \text{Class } D \quad \text{field } (\text{fst } E, D) \ fn = \text{Some}(C, fT)}{E \vdash \{C\}a.fn :: fT}$$

$\text{field} \equiv \text{map_of} \circ (\text{map}(\lambda((fn, fd), ft).(fn, (fd, ft)))) \circ \text{fields}$

Methodenaufruf

$$\frac{E \vdash e :: \text{Class } C \quad E \vdash ps[::]pTs \quad \text{max_spec (fst } E) C (mn, pTs) = \{((-), rT), fpTs\}}{E \vdash e.mn(\{fpTs\}ps) :: rT}$$

- max_spec liefert Menge "genauester" Methoden mit geforderter Signatur
- Menge muß einelementig sein, da sonst ungültige Überladung

Wohldefiniertes Programm

$$\begin{aligned} \text{wf_prog } \Gamma &\equiv (\text{Object}, (\text{None}, -, -)) \in \text{set } \Gamma \wedge \\ &\quad \forall c \in \text{set } \Gamma : \text{wf_cdecl } \Gamma \ c \wedge \\ &\quad \text{nodups}(\text{map fst } \Gamma) \end{aligned}$$

- Programm Γ besitzt Superklasse `Object`
- alle Klassendeklarationen sind wohldefiniert
- keine Klasse ist doppelt deklariert

Wohldefinierte Klassendeklaration

$$\begin{aligned} \text{wf_cdecl } \Gamma (C, sc, fs, ms) \equiv & \\ & (\forall (_, T) \in \text{set } fs : \text{is_type } \Gamma T) \wedge \\ & (\forall m \in \text{set } ms : \text{wf_mdecl } \Gamma C m) \wedge \\ & \text{nodups}(\text{map fst } fs) \wedge \text{nodups}(\text{map fst } ms) \wedge \\ & \text{case } sc \text{ of} \\ & \text{None} \quad \rightarrow C = \text{Object} \\ & | \text{Some } D \quad \rightarrow D \in \text{set}(\text{map fst } \Gamma) \wedge \\ & \quad \neg(\Gamma \vdash D \preceq_C C) \wedge \\ & \quad \forall (sg, T, _) \in \text{set } ms, \forall T' : \\ & \quad \text{method}(\Gamma, D)sg = \text{Some}(_, T', _) \\ & \quad \rightarrow \Gamma \vdash T \preceq T' \end{aligned}$$

Wohldefinierte Methodendeklaration

$$\text{wf_mhead } \Gamma \ (mn, pTs) \ rT \equiv (\forall T \in \text{set } pTs : \text{is_type } \Gamma \ T) \wedge \text{is_type } \Gamma \ rT$$

$$\text{is_type } \Gamma \ T \equiv \text{case } T \text{ of Class } C \rightarrow C \in \text{set}(\text{map fst } \Gamma) \mid _ \rightarrow \text{True}$$

$$\text{wf_method } \Gamma \ C \ ((mn, pTs), rT, (pns, lvars, blk, res)) \equiv$$

$$\text{length } pns = \text{length } pTs \wedge$$

$$\text{nodups } pns \wedge \text{nodups}(\text{map fst } lvars) \wedge$$

$$(\forall pn \in \text{set } pns : \text{map_of } lvars \ pn = \text{None}) \wedge$$

$$(\forall (vn, T) \in \text{set } lvars : \text{is_type } \Gamma \ T) \wedge$$

$$\text{let } E = (\Gamma, \text{map_of } lvars(pns[\mapsto]pTs)(\text{this} \mapsto \text{Class } C))$$

$$\text{in } E \vdash blk \ \checkmark \ \wedge \ (\exists T : E \vdash res :: T \wedge \Gamma \vdash T \preceq rT)$$

Beschreibung der operationellen Semantik

types $obj = cname \times (vname \rightsquigarrow val)$

$heap = loc \rightsquigarrow obj$

$locals = vname \rightsquigarrow val$

$state = heap \times locals$

$xstate = xcpt\ option \times state$

datatype $xcpt = \text{NullPointer} \mid \text{ClassCast} \mid \text{OutOfMemory}$

$prog \vdash xstate \quad -stmt \rightarrow \quad xstate$

$prog \vdash xstate \quad -expr \succ val \rightarrow \quad xstate$

$prog \vdash xstate \quad -expr\ list[\succ]val\ list \rightarrow \quad xstate$

Zustandübergänge

- Ausnahmen bleiben erhalten

$$\Gamma \vdash (\text{Some } xc, \sigma) - c \rightarrow (\text{Some } xc, \sigma)$$

- für Zustandsübergänge ohne Ausnahmen gilt:

$$\frac{\Gamma \vdash (\text{None}, \sigma) - c_1 \rightarrow s_1 \quad \Gamma \vdash (\text{None}, \sigma) - c_2 \rightarrow s_2}{\Gamma \vdash (\text{None}, \sigma) - c_1; c_2 \rightarrow s_2}$$

Beispiel: New C

$$\frac{h \text{ (the_Addr } a) = \text{None} \quad h' = h[a \mapsto \text{Some } (C, \text{init_vars } (\text{fields } (\Gamma, C)))]}{\Gamma \vdash (x, h, l) - \text{New } C \succ a \rightarrow (x, h', l)}$$

the_Addr(Addr l) = l

init_vars \equiv map_of \circ map($\lambda(n, T).((\text{fst } n), \text{default_val } T)$)

Invariante: Werte immer vom deklarierten Typ

$$prog, heap \vdash val :: \prec ty$$

$$prog, heap \vdash (\beta \rightsquigarrow val)[:: \prec](\beta \rightsquigarrow ty)$$

$$state :: \prec env$$

$$\begin{aligned} \text{obj_ty } o &\equiv \text{case } o \text{ of None} \rightarrow \text{None} \\ &\quad | \text{Some}(C, fm) \rightarrow \text{Some } C \end{aligned}$$

$$\begin{aligned} \Gamma, h \vdash v :: \preceq T &\equiv \exists S : \text{typeof } (\text{obj_ty } \circ h) v = \text{Some } S \\ &\quad \wedge \Gamma \vdash S \preceq T \end{aligned}$$

$$\begin{aligned} \Gamma, h \vdash vm[:: \preceq]Tm &\equiv \forall n T : Tm n = \text{Some } T \rightarrow \\ &\quad (\exists v : vm n = \text{Some } v \wedge \Gamma, h \vdash v :: \preceq T) \end{aligned}$$

Invariante: Objekt behält Typ

- erweiterter Heap:

$$h \sqsubseteq h' \equiv \forall a C fm : a = \text{Some}(C, fm) \rightarrow (\exists fm' : h' a = \text{Some}(C, fm'))$$

- Beweisziel der Invariante:

$$\begin{aligned} (\Gamma \vdash (x, h, l) - c \rightarrow (x', h', l')) &\quad \rightarrow h \sqsubseteq h' \wedge \\ (\Gamma \vdash (x, h, l) - e \succ v \rightarrow (x', h', l')) &\quad \rightarrow h \sqsubseteq h' \wedge \\ (\Gamma \vdash (x, h, l) - es[\gamma]vs \rightarrow (x', h', l')) &\quad \rightarrow h \sqsubseteq h' \end{aligned}$$

Hauptsatz zur Typkorrektheit

- für wohldefinierte Programme gibt es bei der Ausführung von typkorrekten Statements und der Auswertung von Ausdrücken eine Übereinstimmung der verwendeten mit den deklarierten Typen
- Teilaussage über Ausführung von Statements:

$$\begin{array}{l}
 \text{wf_prog } \Gamma \\
 \wedge \quad E = (\Gamma, lT) \\
 \wedge \quad E \vdash c \checkmark \\
 \wedge \quad \Gamma \vdash (-, h, l) - c \rightarrow (-, h', l') \\
 \wedge \quad (h, l) :: \preceq E \\
 \longrightarrow \quad (h', l') :: \preceq E \quad \wedge \quad h \trianglelefteq h'
 \end{array}$$

Bytecode-Verifikation

- Bytecode besteht aus assembler-ähnlichen Instruktionen
- Folge von Instruktionen kann ausgelesen werden
- Instruktionsfolge kann analog wie Quelltext modelliert werden
- Aufstellen einer operationellen Semantik für Instruktionsfolge

Mögliche Instruktionen der JVM

datatype *instr* = LS *load_and_store*
| CO *create_object*
| MO *manipulate_object*
| CH *check_object*
| MI *meth_inv*
| MR *meth_ret*
| OS *op_stack*
| BR *branch*