To Be or Not To Be Created

Abstract Object Creation in Dynamic Logic

Wolfgang Ahrendt[1] Frank S. de Boer[2] Immo Grabe[3]

[1]Chalmers University, Göteborg, Sweden

[2]CWI, Amsterdam, The Netherlands

[3]Christian-Albrechts-University Kiel, Germany

Tallinn, 22/08/2012

Part I

Motivation and Outline

Modeling Object Creation in Program Logics

Object-oriented programming languages (like Java):

- high-level way of creating objects
- abstract away from memory allocation
- programmer has no access to non-created (pre-)objects

This abstraction not matched by program logics:

- constant domain assumption
- non-created objects included in quantification
- additional artifacts (ghost fields) to distinguish created objects
- *consistency conditions* on reachable states

Because of mismatch:

- logics loose full abstraction property
- additional complexity in formulas and proofs
- symbolic state bloated by *createdness* information

Approach Taken

- a logic that can only ‘talk about’ created objects
- problem:
  calculus cannot ‘substitute’ new objects into pre-conditions
- solution:
  non-standard substitution using meta-knowledge about ‘newness’
- carry over to symbolic execution paradigm

Simplifications for Presentation

- we examine object creation in simplified setting
- simplifications orthogonal to object creation
- scalable to full languages with abstract object creation

Part II

Syntax and Semantics

A Simple Object-Oriented While Language

- only one class: Object
- 3 types: Object, Integer, Boolean
- no methods
- variables (e.g. u, v, w) distinct from fields (e.g. x, y, z)

statements:

$s \quad ::= \quad \text{while } e \text{ do } s \text{ od} \mid s_1; s_2 \mid$
$\qquad u := e \mid e_1.x := e_2 \mid u := \text{new}$

expressions:

$e \quad ::= \quad u \mid e.x \mid \text{null} \mid e_1 = e_2 \mid op(e_1, ..., e_n)$

to separate issues object creation and aliasing:

- no native statement $\quad e.x := \text{new}$
- can be simulated by $\quad u := \text{new}; e.x := u \quad (u \text{ fresh})$

Hoare Logic

Hoare logic

- $\{\phi\}p\{\psi\}$
- backward calculus:

$$\frac{\{\phi\}p\{\psi^e_x\}}{\{\phi\}p; x := e\{\psi\}}$$

dynamic logic (DL) + updates

- DL extends FOL by modalities
  $[p]\psi$, $\langle p\rangle\psi$
- we add an "update" modality:
  $\{x := e\}\phi$
  (explicit substitution)
- forward calculus:

$$\frac{\Gamma \vdash \{x := e\}[p]\psi}{\Gamma \vdash [x := e; p]\psi}$$

The Logic: Dynamic Logic with Updates

- expressions e, $\{\mathcal{U}\}e$
- logical connectives $\wedge, \vee, \rightarrow, \neg$
- quantified formulas $\forall l.\phi$, $\exists l.\phi$
- modal formulas (base cases):
  $\langle s \rangle \phi$, $[s]\phi$, $\{\mathcal{U}\}\phi$
- $\mathcal{U}$ update of form:
  - $u := e$
  - $e_1.x := e_2$
  - $u :=$ new

Semantics

informal in this talk

- $[\![u := \text{new}]\!]_\sigma$: create new object and assign it to u

- $[\![e]\!]_\sigma \in$ set of objects existing in σ
- $[\![\forall o.\phi]\!]_\sigma$: ϕ holds for all objects existing in σ
- $[\![\exists o.\phi]\!]_\sigma$: ϕ holds for some object existing in σ

e, o of type Object

examples:

$\forall o.\langle u := \text{new}\rangle \neg(u = o)$ true in all states

$\langle u := \text{new}\rangle \forall o.\neg(u = o)$ false in all states

Part III

Calculus

Dynamic Logic Rules

$$\textit{split}\ \frac{\langle s_1\rangle\langle s_2\rangle\phi}{\langle s_1; s_2\rangle\phi} \qquad \textit{if}\ \frac{(e \rightarrow \langle s_1\rangle\phi) \wedge (\neg e \rightarrow \langle s_2\rangle\phi)}{\langle \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}\rangle\phi}$$

$$\textit{unwind}\ \frac{\langle \text{if } e \text{ then } \ s; \text{while } e \text{ do } s \text{ od } \ \text{ else } \ \text{skip } \ \text{ fi}\rangle\phi}{\langle \text{while } e \text{ do } s \text{ od}\rangle\phi}$$

$$\textit{assignVar}\ \frac{\{u := e\}\phi}{\langle u := e\rangle\phi} \qquad \textit{assignField}\ \frac{\{e_1.x := e_2\}\phi}{\langle e_1.x := e_2\rangle\phi}$$

$$\textit{createObj}\ \frac{\{u := \text{new}\}\phi}{\langle u := \text{new}\rangle\phi}$$

Update Application Rule

for certain formulas $\{\mathcal{U}\}\phi$, and expressions $\{\mathcal{U}\}e$,
the $\mathcal{U}$ can be *'applied'* (resolved) using rewrite relation $\rightsquigarrow$

following slides: *big-step* definition of $\rightsquigarrow$

Part IV

Update Application

Update Application: Standard Cases I

$$\frac{\neg\{\mathcal{U}\}\phi \rightsquigarrow \phi'}{\{\mathcal{U}\}(\neg\phi) \rightsquigarrow \phi'}$$

$$\frac{\{\mathcal{U}\}\phi_1 * \{\mathcal{U}\}\phi_2 \rightsquigarrow \phi'}{\{\mathcal{U}\}(\phi_1 * \phi_2) \rightsquigarrow \phi'} \quad \text{with } * \in \{\wedge, \vee, \rightarrow\}$$

$$\frac{op(\{\mathcal{U}\}e_1, ..., \{\mathcal{U}\}e_n) \rightsquigarrow e'}{\{\mathcal{U}\}op(e_1, ..., e_n) \rightsquigarrow e'}$$

$$\{\mathcal{U}\}\alpha \rightsquigarrow \alpha \quad \text{with } \alpha \in \{\mathsf{true}, \mathsf{false}, \mathsf{null}, l\}$$

this slide: $\mathcal{U}$ matches *all* updates

Update Application: Standard Cases II

$$\{u := e\}u \rightsquigarrow e$$

$$\{u := e\}v \rightsquigarrow v \quad u \not\equiv v$$

$$\frac{(\{u := e_1\}e_2).x \rightsquigarrow e'}{\{u := e_1\}(e_2.x) \rightsquigarrow e'}$$

$$\frac{((\{e.x := e_1\}e_2) = e \ ?\ e_1 \ :\ (\{e.x := e_1\}e_2).x) \rightsquigarrow e'}{\{e.x := e_1\}(e_2.x) \rightsquigarrow e'}$$

$$\frac{(\{e.x := e_1\}e_2).y \rightsquigarrow e'}{\{e.x := e_1\}(e_2.y) \rightsquigarrow e'} \quad x \not\equiv y$$

Update Application: Restricted Standard Cases

The standard rules for *quantifiers* and *equality* are restricted to non-creating updates $\mathcal{U}_{nc}$ of the forms ‘$u := e$’ , ‘$e_1.x := e_2$’ .
(‘$u :=$ new’ excluded from these rules.)

$$\frac{\forall l.\, \{\mathcal{U}_{nc}\}\phi \rightsquigarrow \phi'}{\{\mathcal{U}_{nc}\}(\forall l.\, \phi) \rightsquigarrow \phi'}$$

$$\frac{\exists l.\, \{\mathcal{U}_{nc}\}\phi \rightsquigarrow \phi'}{\{\mathcal{U}_{nc}\}(\exists l.\, \phi) \rightsquigarrow \phi'}$$

$$\frac{\{\mathcal{U}_{nc}\}e_1 = \{\mathcal{U}_{nc}\}e_2 \rightsquigarrow e'}{\{\mathcal{U}_{nc}\}(e_1 = e_2) \rightsquigarrow e'}$$

Object Creating Updates: the Issue

note:

- '$\{\mathcal{U}\}\phi$' is the (explicit) weakest precondition $wp(\mathcal{U}, \phi)$

problem:

- result of $\{u := new\}\phi$, i.e., $wp(\{u := new\}, \phi)$, cannot talk about the new object because it does not exist in pre-state
- in particular: $\{u := new\}u \rightsquigarrow$?

basic approach:

- totally avoid '$\{u := new\}u$'
- observation: the only operations on objects are
  - de-referencing fields
  - test for equality
  - quantification
- in all cases, wp computation can employ meta knowledge

Object Creating Update Application: Field Access

$$\frac{(\{u := new\}e).x \rightsquigarrow e'}{\{u := new\}(e.x) \rightsquigarrow e'} \quad e \not\equiv u$$

$$\{u := new\}u.x \rightsquigarrow init_{T(x)}$$

$$init_{T(x)} \equiv \mathsf{null} \mid 0 \mid \mathsf{false}$$

Object Creating Update Application: Equality

$$\frac{(\{u := new\}e_1) = (\{u := new\}e_2) \rightsquigarrow e'}{\{u := new\}(e_1 = e_2) \rightsquigarrow e'}$$

$$e_1 \not\equiv u, \quad e_2 \not\equiv u$$

$$\{u := new\}(u = e) \rightsquigarrow \text{false}$$

$$e \not\equiv u$$

$$\{u := new\}(u = u) \rightsquigarrow \text{true}$$

Object Creating Update Application: Quantifiers

$$\frac{(\{u := new\}\phi(u)) \wedge \forall o.(\{u := new\}\phi(o)) \rightsquigarrow \phi'}{\{u := new\}\forall o.\phi(o) \rightsquigarrow \phi'}$$

$$\frac{(\{u := new\}\phi(u)) \vee \exists o.(\{u := new\}\phi(o)) \rightsquigarrow \phi'}{\{u := new\}\exists o.\phi(o) \rightsquigarrow \phi'}$$

Example Proof 1

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{*}{\text{false} \Longrightarrow}\,\mathit{closeFalse}
}{\Longrightarrow \neg\,\text{false}}\,\mathit{notRight}
}{\Longrightarrow \{u := \text{new}\}\neg(u = c)}\rightsquigarrow
}{\Longrightarrow \langle u := \text{new}\rangle\neg(u = c)}\,\mathit{assignVar}
}{\Longrightarrow \forall o.\langle u := \text{new}\rangle\neg(u = o)}\,\mathit{allRight}
$$

Example Proof 2

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{*}{\forall o.\neg \mathit{false} \Longrightarrow \mathit{true}}\,\mathit{closeTrue}
}{\neg \mathit{true}, \forall o.\neg \mathit{false} \Longrightarrow}\,\mathit{notLeft}
}{\neg \mathit{true} \wedge \forall o.\neg \mathit{false} \Longrightarrow}\,\mathit{andLeft}
}{\{u := \mathsf{new}\}\forall o.\neg(u = o) \Longrightarrow}\;\rightsquigarrow
}{\langle u := \mathsf{new}\rangle \forall o.\neg(u = o)) \Longrightarrow}\,\mathit{assignVar}
}{\Longrightarrow \neg\langle u := \mathsf{new}\rangle \forall o.\neg(u = o)}\,\mathit{notRight}
$$

(applyUpd step in Example Proof 2)

$$\dfrac{\dfrac{\dfrac{}{\{u := new\}(u = u) \rightsquigarrow true}}{\{u := new\}\neg(u = u) \rightsquigarrow \neg true} \quad \dfrac{\dfrac{\dfrac{}{\{u := new\}(u = o) \rightsquigarrow false}}{\{u := new\}\neg(u = o) \rightsquigarrow \neg false}}{\forall o.\{u := new\}\neg(u = o) \rightsquigarrow \forall o.\neg false}}{\{u := new\}\neg(u = u) \wedge \forall o.\{u := new\}\neg(u = o) \rightsquigarrow \neg true \wedge \forall o.\neg false}$$
$$\{u := new\}\forall o.\neg(u = o) \rightsquigarrow \neg true \wedge \forall o.\neg false$$

Part V

Abstract Object Creation in Symbolic Execution

KeY-style Symbolic Execution

up to here, backwards reasoning only

KeY approach:
forward symbolic execution using update parallelisation

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{u < v \Longrightarrow u < v}\,close}{u < v \Longrightarrow \{w := u \,|\, u := v \,|\, v := u\} v < u}\,applyUpd}{u < v \Longrightarrow \{w := u \,|\, u := v\}\{v := w\} v < u}\,mergeUpd}{u < v \Longrightarrow \{w := u \,|\, u := v\}\langle v := w\rangle v < u}\,assignVar}{u < v \Longrightarrow \{w := u\}\{u := v\}\langle v := w\rangle v < u}\,mergeUpd,assignVar}{u < v \Longrightarrow \{w := u\}\langle u := v; v := w\rangle v < u}\,split,assignVar}{u < v \Longrightarrow \langle w := u; u := v; v := w\rangle v < u}\,split,assignVar
$$

Problem: Parallelising Object Creating Updates

no natural way of merging $\{u := new\}$ with other updates

consider the two formulas (one true, one false):

$$\langle u := \text{new}; v := u \rangle (u = v) \qquad \langle u := \text{new}; v := \text{new} \rangle (u = v)$$

symbolic execution generates:

$$\{u := \text{new}\}\{v := u\}(u = v) \qquad \{u := \text{new}\}\{v := \text{new}\}(u = v)$$

merging updates, both result in:

$$\{u := \text{new} \mid v := \text{new}\}(u = v)$$

cannot be true and false

Solution

- not merge object creation with other updates
- split $\{u := \text{new}\}$ into creation and (mergable) assignment to u

new object creation rule:

$$\textit{createObj}\ \frac{\{a := \text{new}\}\{u := a\}\phi}{\langle u := \text{new}\rangle\phi}$$

a a fresh program variable

facilitate merging of all non-creating updates by shifting creation

$$\textit{shiftCreation}\ \frac{\{u := \text{new}\}\{\mathcal{U}_{nc}\}\phi}{\{\mathcal{U}_{nc}\}\{u := \text{new}\}\phi}$$

u not appearing in (non-creating) $\mathcal{U}_{nc}$

Symbolic Execution Proof

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{*}{\Longrightarrow \neg \mathit{false}}\ \mathit{notRight},\ \mathit{closeFalse}
}{\Longrightarrow \{a := \mathsf{new}\}\neg(v = a)}\ \mathit{applyUpd}
}{\Longrightarrow \{a := \mathsf{new}\}\{u := v \mid v := a \mid w := u\}\neg(w = v)}\ \mathit{applyUpd}
}{\Longrightarrow \{a := \mathsf{new}\}\{u := v \mid v := a\}\langle w := u\rangle\neg(w = v)}\ \mathit{assignVar},\mathit{mergeUpd}
}{\Longrightarrow \{a := \mathsf{new}\}\{u := v\}\{v := a\}\langle w := u\rangle\neg(w = v)}\ \mathit{mergeUpd}
}{\Longrightarrow \{u := v\}\{a := \mathsf{new}\}\{v := a\}\langle w := u\rangle\neg(w = v)}\ \mathit{shiftCreation}
}{\Longrightarrow \{u := v\}\langle v := \mathsf{new}\rangle\langle w := u\rangle\neg(w = v)}\ \mathit{createObj}
}{\Longrightarrow \langle u := v; v := \mathsf{new}; w := u\rangle\neg(w = v)}\ \mathit{split},\mathit{assignVar},\mathit{split}
$$

Part VI

Object Creation vs. Object Activation

Abstract Object Creation Proof

reconsider proof from above

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\text{false} \Longrightarrow}\,\mathit{closeFalse}}{\Longrightarrow \neg\,\text{false}}\,\mathit{notRight}}{\Longrightarrow \{u := \text{new}\}\neg(u = c)}\,\rightsquigarrow}{\Longrightarrow \langle u := \text{new}\rangle\neg(u = c)}\,\mathit{assignVar}}{\Longrightarrow \forall o.\langle u := \text{new}\rangle\neg(u = o)}\,\mathit{allRight}$$

Object Activation Proof

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{c.\mathrm{cre},\ \mathrm{obj}(\mathrm{next}) = c \Longrightarrow c.\mathrm{cre}}\ \mathit{close}}{c.\mathrm{cre},\ \mathrm{obj}(\mathrm{next}) = c \Longrightarrow \mathrm{obj}(\mathrm{next}).\mathrm{cre}}\ \mathit{equality}}{\neg\mathrm{obj}(\mathrm{next}).\mathrm{cre},\ c.\mathrm{cre},\ \mathrm{obj}(\mathrm{next}) = c \Longrightarrow}\ \mathit{notLeft}}{(\mathrm{obj}(\mathrm{next}).\mathrm{cre} \leftrightarrow \mathrm{next} < \mathrm{next}),\ c.\mathrm{cre},\ \mathrm{obj}(\mathrm{next}) = c \Longrightarrow}\ \mathit{(\approx 2\ rules)}}{\forall n.(\mathrm{obj}(n).\mathrm{cre} \leftrightarrow n < \mathrm{next}),\ c.\mathrm{cre},\ \mathrm{obj}(\mathrm{next}) = c \Longrightarrow}\ \mathit{allLeft}}{c.\mathrm{cre},\ \mathrm{obj}(\mathrm{next}) = c \Longrightarrow}\ \mathit{inReachableState}}{c.\mathrm{cre} \Longrightarrow \neg(\mathrm{obj}(\mathrm{next}) = c)}\ \mathit{notRight}}{c.\mathrm{cre} \Longrightarrow \{u := \mathrm{obj}(\mathrm{next});\ u.\mathrm{cre} := \mathrm{true};\ \mathrm{next} := \mathrm{next}+1\}\neg(u = c)}\ \mathit{applyUpd}}{\dfrac{\dfrac{c.\mathrm{cre} \Longrightarrow \langle u := \mathrm{new}\rangle\neg(u = c)}{\Longrightarrow c.\mathrm{cre} \rightarrow \langle u := \mathrm{new}\rangle\neg(u = c)}\ \mathit{impRight}}{\Longrightarrow \forall o.\,(o.\mathrm{cre} \rightarrow \langle u := \mathrm{new}\rangle\neg(u = o))}\ \mathit{allRight}}\ \mathit{createObj}
$$

Part VII

Reflections

Reflections

- abstraction level of logic matches programming language
- changes to standard treatment very local
  - additional update type,
    not mergable with others, but shiftable to the front
  - update application differs only in few cases
- formulas and proofs are simpler
- symbolic state representation (updates):
  - not diluted by createdness bookkeeping
  - separates out
    1. newly created objects (shifted forward)
    2. symbolic value of fields and variables

Part VIII

Ongoing Work

Ongoing Work

- current work (with Stijn de Gouw + Richard Bubel):
  - improved implementation + case studies
  - scale to standard language features
  - using contracts in verification