Abstract Interface Behavior of Object-Oriented Languages with Monitors

Martin Steffen

Christian-Albrechts University Kiel

Oslo

20 Februrary 2006

Structure

- introduction
- semantics
- interface description
  - lock ownership
  - data dependencies
  - control dependencies
- conclusion

Introduction

- considered so far
  - classes and instantiation
  - ⇒ heap
  - multithreading (vs. sequential/deterministic programs)
  - connectivity
- here: synchronization/monitors

Monitors

- shared (instance) state + concurrency ⇒ mutex
- sync. mechanism: monitors
- for instance in *Java*
- here
  - no synchronized blocks
  - no wait/signal[1]
  - no connectivity
- but:
  - re-entrant monitors (recursion)
- deliverable for task 1 (“compositionality and modularity: a semantic approach”), subtask 1.c (“basic features: libraries and synchronization protocols”), cf. [2, Sec. 7.2].

[1] In *Java*: wait and notify.

Why is this interesting?

- fundamental question:

  what is observable of an oo program?

- Now:

  Does the addition of monitors increase or decrease the discriminating power or not?

- intuitively: 2 plausible answers:
  - the observer sees less!
  - the observer sees more!

Why is this interesting?

- fundamental question:

  what is observable of an oo program?

- Now:

  > *Does the addition of monitors increase or decrease the discriminating power or not?*

- intuitively: 2 plausible answers:
  - the observer sees less!
  - the observer sees more!

Why is this interesting?

- fundamental question:

  what is observable of an oo program?

- Now:

  > *Does the addition of monitors increase or decrease the discriminating power or not?*

- intuitively: 2 plausible answers:
  - the observer sees less!
  - the observer sees more!

Road map

- incorporate *monitors* into the semantics
- characterization of the interface behavior
  - may and must approximation of lock-ownership
- design goals
  - (preferably) seamless extension of the calculus with an
  - eye to compositionality
  - ⇒ clean separation of concerns between

    assumptions vs. *commitments*

  - intuitively:
    - enabledness of input must depend only on the environment (= assumption)
    - enabledness of output must depend only on the component (= commitments)
  - interface *trace* must contain all relevant information relevant (and not part of the internal state(s))
  - cf. game theory

introduction

semantics

interface description
- lock ownership
- data dependencies
- control dependencies

conclusion

Syntax

- modest changes
- objects with locks
- extend object = class + fields (written $o[c, F]$ to "class + fields + lock"

$$o[c, F, n]$$

(lock n = reference to thread)

Syntax

$$
\begin{array}{rcll}
C & ::= & \mathbf{0} \mid C \parallel C \mid \nu(n{:}T).C \mid n[\![O]\!] \mid n[n,F,n] \mid n\langle t\rangle & \text{program} \\
O & ::= & F, M & \text{object} \\
M & ::= & l^u = m, \ldots, l^u = m, l^s = m, \ldots, l^s = m & \text{method suite} \\
F & ::= & l^u = f, \ldots, l^u = f & \text{fields} \\
m & ::= & \varsigma(n{:}T).\lambda(x{:}T, \ldots, x{:}T).t & \text{method} \\
f & ::= & \varsigma(n{:}T).\lambda().v \mid \varsigma(n{:}T).\lambda().\bot_n & \text{field} \\
t & ::= & v \mid \mathit{stop} \mid \mathit{let}\ x{:}T = e\ \mathit{in}\ t & \text{thread} \\
e & ::= & t \mid \text{if } v = v \text{ then } e \text{ else } e \mid \text{if } \mathit{undef}(v.l) \text{ then } e \text{ else } e & \text{expr.} \\
 & \mid & v.l(v, \ldots, v) \mid v.l := v \mid \mathit{currentthread} & \\
 & \mid & \mathit{new}\ n \mid \mathit{new}\langle t\rangle & \\
v & ::= & x \mid n & \text{values}
\end{array}
$$

Semantics

1. operational semantics
2. remember the design-goals
3. two stages
   - internal semantics
     - closed system
     - spec. of the “virtual machine”
   - external semantics
     - interaction with environment via
     - message passing (calls/returns)

first attempt

- example: incoming call of unsynchronized method

$$\frac{\acute{\Xi} = \Xi + a \quad \acute{\Xi} \vdash \lfloor a \rfloor : T}{\dfrac{a = \nu(\Xi').\, n\langle call\ o_r.l(\vec{v})\rangle? \quad t_{blocked} = let\ x'{:}T' = block\ in\ t}{\begin{array}{l}\Xi \vdash C \parallel n\langle t_{blocked}\rangle \xrightarrow{a} \\ \acute{\Xi} \vdash C \parallel C(\Theta') \parallel n\langle let\ x{:}T = o_r.l(\vec{v})\ in\ return\ x; t_{blocked}\rangle\end{array}}}$$

first attempt

- example: incoming call of synchronized method
- assume: lock is free

$$\frac{\acute{\Xi} = \Xi + a \quad \acute{\Xi} \vdash \lfloor a \rfloor : T \qquad a = \nu(\Xi').\, n\langle call\ o_r.l(\vec{v})\rangle? \quad t_{blocked} = let\, x' {:} T' = block\, in\, t}{\Xi \vdash C \parallel o[c, F', \bot_{thread}] \parallel n\langle t_{blocked}\rangle \xrightarrow{a} \acute{\Xi} \vdash C \parallel C(\Theta') \parallel o[c, F', \mathbf{n}] \parallel n\langle let\, x{:}T = o_r.l(\vec{v})\, in\, return\, x; t_{blocked}\rangle}$$

first attempt

- example: incoming call of synchronized method
- assume: lock is free

$$\frac{a = \nu(\Xi').\, n\langle call\ o_r.l(\vec{v})\rangle? \quad t_{blocked} = let\ x'{:}T' = block\ in\ t \qquad \acute{\Xi} = \Xi + a \quad \acute{\Xi} \vdash \lfloor a \rfloor : T}{\Xi \vdash C \parallel o[c, F', \bot_{thread}] \parallel n\langle t_{blocked}\rangle \xrightarrow{a} \acute{\Xi} \vdash C \parallel C(\Theta') \parallel o[c, F', \mathbf{n}] \parallel n\langle let\ x{:}T = o_r.l(\vec{v})\ in\ return\ x;\, t_{blocked}\rangle}$$

- problem:
  - internal and external behavior not separated
  - whether the incoming call is possible: *dependent* on the component-internal state,[2] i.e.,
  - the history trace doesn't contain enough information to determine enabledness

[2] Note: for $t_{blocked}$, the problem is not there even if it looks the same.

“Non-atomic lock grabbing”

- handing over of call:
  - irrespective of availability of lock
  - i.e., no difference of external/intefaces rules for synchronized vs. non-synchronized methods!
  - component is input enabled

⇒ lock-grabbing (of comp. locks) is an internal step

- interface interaction: non-atomic lock-handling.

“Non-atomic lock grabbing”

- handing over of call:
  - irrespective of availability of lock
  - i.e., no difference of external/intefaces rules for synchronized vs. non-synchronized methods!
  - component is input enabled

$\Rightarrow$ lock-grabbing (of comp. locks) is an internal step

- interface interaction: non-atomic lock-handling.

$$\frac{\acute{\Xi} = \Xi + a \quad \acute{\Xi} \vdash \lfloor a \rfloor : T \qquad a = \nu(\Xi').\, n\langle call\ o_r.l(\vec{v})\rangle? \quad t_{blocked} = let\ x':T' = block\ in\ t}{\Xi \vdash C \parallel n\langle t_{blocked}\rangle \xrightarrow{a} \acute{\Xi} \vdash C \parallel C(\Theta') \parallel n\langle let\ x{:}T = o_r.l(\vec{v})\ in\ return\ x\,;\, t_{blocked}\rangle}$$

Internal steps

$$c[\![F,M]\!] \parallel o[c,F',\bot_{thread}] \parallel n\langle let\, x{:}T = o.l^s(\vec{v})\, in\, t\rangle \xrightarrow{\tau}$$
$$c[\![F,M]\!] \parallel o[c,F',\mathbf{n}] \parallel n\langle let\, x{:}T = M.l^s(o)(\vec{v})\, in\, release(\mathbf{o}); t\rangle$$

$$c[\![F,M]\!] \parallel o[c,F',\mathbf{n}] \parallel n\langle let\, x{:}T = o.l^s(\vec{v})\, in\, t\rangle \xrightarrow{\tau}$$
$$c[\![F,M]\!] \parallel o[c,F',\mathbf{n}] \parallel n\langle let\, x{:}T = M.l^s(o)(\vec{v})\, in\, t\rangle \qquad \text{CALL}^s_{i_2}$$

- 2 internal rules for sync. methods
- note: re-entrancy, aux. syntax **release**

introduction

semantics

interface description

- lock ownership
- data dependencies
- control dependencies

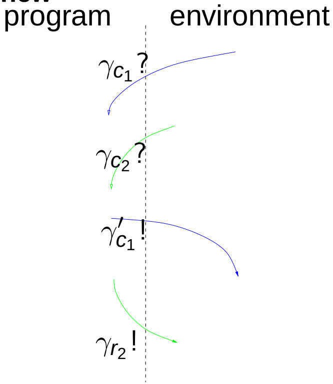
conclusion

Interface description: Task

- cf. Andreas' talk
- characterize possible interface behavior
- possible = adhering to the restriction of the language
  - well-typed
  - no violation of mutex
- rudimentary trace logic

Example (1)

- 2 calls, competing for the same (component) lock
- data dependence
  - o' received by the first call (of n_1)
  - returned by second thread n_1 afterwards
  - note: o' is **new**



- question: *is that trace possible?*

Example (1)

- 2 calls, competing for the same (component) lock
- data dependence
  - o' received by the first call (of n_1)
  - returned by second thread n_1 afterwards
  - note: o' is **new**

$$\gamma_{c_1}?\ \gamma_{c_2}?\ \gamma'_{c_1}!\ \gamma_{r_2}! \quad =$$

$$(\nu o':c) n_1\langle call\ o.l(o')\rangle?\ n_2\langle call\ o.l()\rangle?\ n_1\langle call\ \tilde{o}.l()\rangle!\ n_2\langle return(o')\rangle!$$

- question: *is that trace possible?*

Example (1)

$$\gamma_{c_1}?\ \gamma_{c_2}?\ \gamma'_{c_1}!\ \gamma_{r_2}! \quad =$$

$$(\nu o'{:}c)n_1\langle call\ o.l(o')\rangle?\ n_2\langle call\ o.l()\rangle?\ n_1\langle call\ \tilde{o}.l()\rangle!\ n_2\langle return(o')\rangle!$$

- question: *is that trace possible?*
- the answer is **no!**
- **data:** "n_1 before n_2"
- **monitors:**
  - the outgoing call of n_1 shows that n_1 must have the lock now

$\Rightarrow$ n_2 cannot have it now: $\Rightarrow$

"n_2 before n_1"

Example (1)

$$\gamma_{c_1}?\ \gamma_{c_2}?\ \gamma'_{c_1}!\ \gamma_{r_2}! \quad = \quad (\nu o'{:}c)n_1\langle call\ o.l(o')\rangle?\ n_2\langle call\ o.l()\rangle?\ n_1\langle call\ \tilde{o}.l()\rangle!\ n_2\langle return(o')\rangle!$$

- question: *is that trace possible?*

$\gamma_{c_1}?$ $\gamma_{c_2}?$

(2)

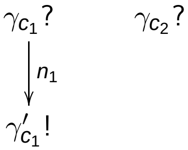
Note: *non-atomic* lock-grabbing $\Rightarrow$ **no order!**

Example (1)

$$\gamma_{c_1}? \ \gamma_{c_2}? \ \gamma'_{c_1}! \ \gamma_{r_2}! \quad = $$

$$(\nu o' {:} c) n_1 \langle call\ o.l(o') \rangle ? \ n_2 \langle call\ o.l() \rangle ? \ n_1 \langle call\ \tilde{o}.l() \rangle ! \ n_2 \langle return(o') \rangle !$$

- question: *is that trace possible?*



(3)

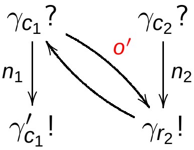
<u>Note:</u> *there is* ***no*** *order between events of* n_1 *and* n_2*!*

Example (1)

$$\gamma_{c_1}?\ \gamma_{c_2}?\ \gamma'_{c_1}!\ \gamma_{r_2}! \qquad = $$

$$(\nu o':c)n_1\langle call\ o.l(o')\rangle?\ n_2\langle call\ o.l()\rangle?\ n_1\langle call\ \tilde{o}.l()\rangle!\ n_2\langle return(o')\rangle!$$

- question: *is that trace possible?*



(4)

<u>Note:</u>

- data dependence because of o'

Conditions characterizing monitors

- apart from conditions concerning non-monitor features
  - well-typedness
  - freshness
  - (connectivity)
- 3 types of dependecies/precedences between events

1. ***mutual exclusion:***
   If a thread has taken the lock of a monitor, interactions of other threads with that monitor must either occur before the lock is taken, or after it has been released again.

2. ***data** dependencies:*
   no value (unless generated new) can be transmitted before it has been received.

3. ***control** dependecies:*
   within 1 thread, the events are linearly ordered.

Lock ownership

- question:

  given interaction of thread n, is the lock of object o available

- first attempt:

  “after call $n\langle call\ o.l()\rangle?$, thread n owns the lock of o.”

- alas: not true!

- *complication*: non-atomic lock-grabbing
- handing-over call $\Rightarrow$ not necessarily obtaining lock

Lock ownership: non-atomic lock grabbing

- delayed observation:

  "after $n\langle call\ o.l()\rangle?$, *thread n may own lock of component object o. "*

- and later:

  after $n\langle call\ o.l()\rangle?\ n\langle call\ o'.l()\rangle!$, *thread n must own lock of o.*

- 2 approximations *per thread*:
  - *potential* lock-ownership: "may", written: $\Diamond_n \mathbf{o}$
  - *necessary* lock-ownership: "must", written: $\Box_n \mathbf{o}$

Lock-ownership: May-approximation

- given the trace t projected to one thread
- from the component-perspective[3]

  after s, the thread may own the lock of o:

$$\boxed{\Xi \vdash s : \Diamond_o}$$

[3] dually for the environment.

Lock-ownership: May-approximation

$$\frac{\vdash s_2 : \mathit{balanced} \qquad s_2 \neq \epsilon \qquad \Xi \vdash s_1 : \Diamond o}{\Xi \vdash s_1\, s_2 : \Diamond o}\ \text{M-}\Diamond$$

$$\frac{\mathit{receiver}(s_1\gamma_c) = o}{\Xi \vdash s_1\ \gamma_c? : \Diamond o}\ \text{M-I}\Diamond_1 \qquad \frac{\mathit{receiver}(s_1\gamma_c) \neq o \quad \Xi \vdash s_1 : \Diamond o}{\Xi \vdash s_1\ \gamma_c? : \Diamond o}\ \text{M-I}\Diamond_2$$

$$\frac{\Xi \vdash s_1\ : \Diamond o}{\Xi \vdash s_1\ \gamma_c! : \Diamond o}\ \text{M-O}\Diamond$$

Lock-ownership: Must-approximation

- similar system as in the may case
- based on the may-system[3]
- again from the component-perspective

  after s, the thread must own the lock of o:

$$\Xi \vdash s : \Box_o$$

[3] but no mutual recursion

Lock-ownership: Must-approximation

$$\frac{\Xi \vdash t : \Box o}{\Xi \vdash t\gamma_c? : \Box o}\ \text{M-I}\Box_1 \qquad \frac{\Xi \vdash t\gamma_r?\gamma_r'! : \Diamond o \qquad \Xi \vdash t : \Box o}{\Xi \vdash t\gamma_r? : \Box o}\ \text{M-I}\Box_2$$

$$\frac{\Xi \vdash t : \Diamond o}{\Xi \vdash t\gamma_c! : \Box o}\ \text{M-O}\Box_1 \qquad \frac{\Xi \vdash t : \Box o}{\Xi \vdash t\gamma_r! : \Box o}\ \text{M-O}\Box_2$$

Illustration

Example

$$t = \gamma_c? = (\nu\Xi)n\langle call\ o_r.l(o)\rangle? \ .$$

then

$$\Xi \vdash t : \Diamond_{o_r} \quad \text{and} \quad \Xi \vdash t : \neg\Diamond o$$

Note: $\Diamond$ is a *local* interpretation.

Example

$$t = \gamma_c?\gamma_r! = (\nu\Xi)n\langle call\ o_r.l()\rangle?\ n\langle return()\rangle! \ .$$

Then:

$$\Xi \vdash \gamma_c? : \Diamond_n o_r \quad \text{but} \quad \Xi \nvdash \gamma_c? : \Box_n o_r$$

and

$$\Xi \vdash t : \neg\Diamond_{o_r}$$

Mutual exclusion

- here: again for component locks
- "*global*" perspective: not just one thread
- *mutex* precedence edges for event *a* after *r* wrt. component object *o*.

$$\boxed{M_{\ominus}(ra, o)}$$

- auxiliary definitions:
  - "after may": $\acute{\Diamond}(t, o)$
  - "before must": $\grave{\Box}(t, o)$
- edges: $\vdash \mathbf{a_1} \rightarrow^{\mathbf{m}} \mathbf{a_2}$
- distinction for *a* between
  - incoming communication
    - no condition for incoming returns
    - incoming calls
  - outgoing communication: 2 conditions
    - *a* before other threads have taken the lock
    - after

CAU

Mutual exclusion

$$
\begin{aligned}
M_{\ominus}(r\gamma_c?, o) &= \acute{\Diamond}_{\neq n}(r, o) \rightharpoonup \gamma_c? \\
M_{\ominus}(r\gamma_r?, o) &= \{\} \\
\\
M_{\ominus}(r\gamma!, o) &= \gamma! \rightharpoonup \grave{\Box}_{\neq n}(r, o), \\
&\quad\; \acute{\Diamond}_{\neq n}(r, o) \rightharpoonup \grave{\Box}_{n}(r\gamma!, o)
\end{aligned}
$$

data dependence

- jugment

$$\boxed{\vdash_{\Theta} r : \gamma? \rightarrow^{d} o}$$

  if $o \in names(\gamma)$ and $r'\gamma?$ is a prefix of r.

- "o is potentially data-dependent on event/label $\gamma?$ of trace r"
- <u>note</u>: it's only potential dependence

$$\begin{array}{rcll} D_{\Theta}(r\gamma!) & = & \{\vec{\gamma}? \rightarrow \gamma!\} & \text{where } \vdash_{\Theta} \vec{\gamma}? \rightarrow^{d} fn(\gamma!) \cap \Delta(r) \\ D_{\Theta}(r\gamma?) & = & \{\} . & \end{array}$$

For Δ, the definitions are applied dually.

control dependencies

- precedence nr. 3
- trivial

$\Rightarrow$ the events within each trace are linearly ordered

- notation

$$\vdash \mathbf{a}' \rightarrow^{\mathbf{c}} \mathbf{a}$$

putting it together: legal traces

- formal system to characterize interface behavior
- *non-branching* :-)
- judgment:

$$\Xi; G \vdash r \rhd s : trace$$

- "after r and with assumption/commitment-contexts Ξ and G, the trace s is possible"
- context G:
  - precedence graphs
  - cleanly separated into G_Δ and G_Θ
  - 3 reasons for precedence:
    1. $\rightarrow^{\mathbf{m}}$
    2. $\rightarrow^{\mathbf{d}}$
    3. $\rightarrow^{\mathbf{c}}$
- G must remain acyclic: $\vdash \mathbf{G}\ ok$

putting it together: legal traces

$$\dfrac{\begin{array}{c}\Xi \vdash r \rhd o_s \xrightarrow{a} o_r \quad \acute{\Xi} = \Xi + a \quad \acute{\Xi} \vdash a : ok \\ \acute{G}_\ominus = G_\ominus \cup G_\ominus(ra, o_r) \quad \acute{G}_\Delta = G_\Delta \cup G_\Delta(ra, o_s) \quad \vdash \acute{G}_\Delta : ok \\ a = \nu(\Xi').\ n\langle call\ o_r.l(\vec{v})\rangle? \quad \acute{\Xi}; \acute{G} \vdash r\ \mathbf{a} \rhd s : trace\end{array}}{\Xi; G \vdash r \rhd a\ s : trace} \text{ L-CALLI}$$

Results

- Soundness of the abstraction
- in particular: soundness of may and must:

Lemma (Soundness of lock ownership)

1. $\Xi \vdash C \stackrel{t}{\Longrightarrow} \acute{\Xi} \vdash \acute{C}$ *and* $\Xi \vdash t : \Box_n o$*, then thread* n *has the lock of* o *in* $\acute{C}$*.*
2. *If* $\Xi \vdash C \stackrel{t}{\Longrightarrow}$ *and* $\Xi \vdash t : \Diamond_n o$ *and there does not exist an* $n' \neq n$ *s.t.* $\Xi \vdash t : \Box_{n'} o$*, then* $\Xi \vdash C \stackrel{t}{\Longrightarrow} \acute{\Xi} \vdash \acute{C}$ *for some* $\acute{\Xi} \vdash \acute{C}$ *s.t. the thread* n *has the lock of* o *in* $\acute{C}$*.*

Future work

- combination with cross-border instantiation/connectivity[3]
- thread coordination:[4]
  - wait
  - signal
- “cleaner” characterization:
  - non-determinism is theoretically (and practically) unpleasant
  - better: “real” strongest post-condition
  - “event-structures”?

[3] conceptually not too complicated, technically tricky.

[4] no ideas yet

References I

[1] E. Ábrahám, A. Grüner, and M. Steffen.
Abstract interface behavior of object-oriented languages with monitors.
Jan. 2006.
Submitted as conference contribution.

[2] Mobi-j II. Formal methods for components and objects.
A continuation proposal for cooperation between research groups in bilateral research program NWO/DFG, May 2004.