Concurrent Object-Oriented Programming

Bertrand Meyer, Piotr Nienaltowski
Lecture 11:
More about synchronization in SCOOP
Outline

- Refresher on type system
- Handling false traitors
- Lock passing and how to use it
Refresher: type system

- Current processor
  \[ x : \mathcal{X} \]

- Some processor (top)
  \[ x : \text{separate } \mathcal{X} \]

- Processor tag
  \[ \alpha \in \{\bullet, \top, \bot, \langle p \rangle, \langle a.\text{handler} \rangle\} \]

- Attached/detachable
  \[ \gamma \in \{!, \} \]

- No processor (bottom)
  \[ \text{Void} \]

\[ \Gamma \vdash x :: (\gamma, \alpha, \mathcal{C}) \]
Examples

\( x: X \quad -- \ x :: (!, \bullet, X) \)

\( y: \text{separate} \ Y \quad -- \ y :: (!, \tau, Y) \)

\( z: \ ? \text{separate} \ Z \quad -- \ z :: (?, \tau, Z) \)

- Expanded types are attached and non-separate
  \( i: \text{INTEGER} \quad -- \ i :: (!, \bullet, \text{INTEGER}) \)

- **Void** is detachable
  -- **Void** :: (?, \perp, \text{NONE})

- **Current** is attached and non-separate
  -- **Current** :: (!, \bullet, C_{\text{Current}})
Subtyping rules

- Since you don’t like Greek letters, I’ll keep it informal
- \( TT_2 \leq TT_1 \) means “\( TT_2 \) is a subtype of \( TT_1 \)”
- Conformance on class types like in Eiffel, essentially based on inheritance
  \[ D \leq_{Eiffel} C \iff (\gamma, \alpha, D) \leq (\gamma, \alpha, C) \]
- Attached \( \leq \) detachable
  \[ (!, \alpha, C) \leq (? , \alpha, C) \]
- Any processor tag \( \leq T \)
  \[ (\gamma, \alpha, C) \leq (\gamma, \top, C) \]
- In particular, non-separate \( \leq T \)
  \[ (\gamma, \cdot, C) \leq (\gamma, \top, C) \]
- \( \bot \leq \) any processor tag
  \[ (\gamma, \bot, C) \leq (\gamma, \alpha, C) \]
Using the type system

- We can rely on standard type rules
- Enriched types give us additional guarantees
- Assignment rule: source conforms to target

\[
x :: \text{TT}_x, \quad e :: \text{TT}_e, \quad \text{TT}_e \leq \text{TT}_x
\]

[Assign] \[x := e\]

- No need for special validity rules for separate
Type combinators

\[ x : T_x \]
\[ f (fa : T_{fa}) : T_{res} \]

- Actual argument
  \[ a :: T_x \otimes T_{fa} \]

- Result type
  \[ x.f (a) :: T_x \ast T_{res} \]
**False traitors**

```
meet_friend (person: separate PERSON) is
  local
    a_friend: PERSON
  do
    a_friend := person.friend  -- Invalid assignment.
    visit_locally (a_friend)
  end
```
meet_friend (person: separate PERSON) is

  local

    a_friend: PERSON

  do

    a_friend ?= person.friend -- Valid assignment attempt.

    if a_friend /= void then visit_locally (a_friend) end

end
Assignment attempt

- Like in Eiffel but also "downcasts" processor tags
  - "deep downcast" over expanded attributes

- Eiffel standard has replaced `?=` with `object test`

```java
if {a_friend: PERSON} person.friend then
  visit_locally (a_friend)
end
```

- Use assignment attempt with `scoop2scoopl`
  - **Attention**: conversion does not work with assignment attempts
A synchronization problem in SCOOP

... x.f

x.g(y)  x.f  Px  g(y: separate Y)

...  y.f

... y.f

end
Solution (original SCOOP)

- Make $x$ wait until $y$ becomes available
- May be **dangerous**. Why?

```c
r (x: separate X; y: separate Y) is
do
  x.f
  x.g (y)
  y.f
  ...
  value := x.some_query
end
```

- "Business Card principle" for dealing with tricky cases
- Not flexible
Solution (current SCOOP)

- **Lock passing** occurs when client passes locked actual argument to separate call that wants to lock that argument.

\[ r (x: \text{separate } X; y: \text{separate } Y) \text{ is} \]

\[
do
  \quad x.f
  \quad x.g (y) \quad \text{-- x gets access to y.}
    \quad \text{-- Current waits until call has terminated.}
  \quad y.f
  \quad \ldots
  \quad \text{value} := x.\text{some_query} \quad \text{-- No deadlock here.}
end
\]

- **x gets exclusive access on y immediately**
- **Client has to wait until separate call terminated**
Solution (current SCOOP)

- In fact, client has to pass all its locks

- Necessary conditions for lock passing
  - **Current** has lock on supplier \( x \) and argument \( y \).
  - \( x \) is handled by a different processor than \( y \) and **Current**; \( y \) and **Current** may be on the same processor.
  - **Current** calls feature \( f \) on \( x \) with argument \( y \); formal argument of \( f \) corresponding to \( y \) is separate and attached.

- If feature \( f \) takes detachable formal argument, no lock passing occurs.

- Original SCOOP semantics can be simulated using object test.
Why do we need lock passing?

- Avoiding deadlocks

\[ s (x: \texttt{separate } X) \text{ is} \]
\[
\text{do}
\]
\[
\text{value} := x.f (\texttt{Current}) \quad \text{-- Deadlocks if no lock passing.}
\]
\[
\text{end}
\]

- Expressiveness and precision: do what the programmer wants.

- Novel approach to reasoning about asynchronous calls
  - Hoare-style rules
  - separate objects locked by client are treated as non-separate
In practice

- Lock passing fully implemented
- Detachable and attached types are not yet supported by Eiffel compilers
- Scoop2scoopl does not accept '?' annotation
- Lock passing applied by default