Synchronization as a Special Case of Access Control

Franz Puntigam

Vienna University of Technology
Vienna, Austria

`franz@complang.tuwien.ac.at`

Motivation

- Concurrency too difficult for average (oo) programmer

  → Base synchronization on object state (as in SCOOP)

  → Specify accessibility instead of synchronization

  → Give static guarantees (race-freeness, continuity, . . .)

- Concurrent components are no “black boxes”

  → Express accessibility and synchronization in interfaces

- Concurrency dominates software architecture

  → Allow better factorization by “moving” synchronization

Specifying Access Constraints

```
class Window {
    public void iconify() [icon:true -> icon:false]
        { ...; icon=false; ... }
    public void uniconify() [icon:false -> icon:true]
        { ...; icon=true; ... }
    ...
    public Window(...) [-> !icon:false]
        { ... }
    ...
    private boolean icon = false;
    ...
}
```

Moving Tokens Around

```
void fooA (Window[icon:false -> icon:true] w) {
    w.iconify();
}

void fooB (Window[icon:false -> icon:boolean] w) {
    if (...) w.iconify();
}

Window[-> icon:true] fooC (Window[icon:false ->] w) {
    if (...) { w.iconify(); return w; }
    else { return null; }
}
```

Threads and Synchronization

```
void syncTest() {
   Window w = new Window(); // associate w with !icon:false
   t1(w);
   t2(w);
}
async t1 (Window[icon:false?true ->] u) {
   while(true)
      u.iconify();           // needs write-lock on u.icon
}
async t2 (Window[icon:true?false ->] v) {
   while(true)
      v.uniconify();         // needs write-lock on v.icon
}
```

Information in Tokens

$!v{:}\tau$	value in variable v is of type τ token replaceable by ?-tokens	exclusive access token unique
$v{:}\tau$	value in variable v is of type τ token not replaceable by ?-tokens	exclusive access sim. ?-tokens
$*v{:}\tau$	value in variable v is of type τ token not replaceable by ?-tokens	shared read-access sim. $*$-, ?-tokens
$v{:}\sigma?\tau$	type of value in v unknown v lockable if its value of type σ	no direct access sim. non-!-tokens

while locked: change value from one of σ to one of τ

Race-Free Programs

- Instance/class variable u is protected by v if each method
  - writing to u requires a token $v{:}\tau$ or $!v{:}\tau$;
  - reading from u requires a token $*v{:}\tau$ or $v{:}\tau$ or $!v{:}\tau$.
- Race-free program if each instance/class variable protected
- Easily ensured by static checking

Continuity

- For each token $v{:}\sigma?\tau$ repeatedly invoke a method annotated with [$v{:}\sigma$ -> $v{:}\tau$]
- Always cycles in ?-tokens: $v{:}\tau_0?\tau_1$, $v{:}\tau_1?\tau_2$, $\ldots$, $v{:}\tau_n?\tau_0$
- No ?-token must disappear
- Class/instance variables not associated with ?-tokens
- If v is in “stop mode”, each method invocation using $v{:}\sigma?\tau$ gets an exception instead of a lock

Deadlock Prevention

- Very simple approach:
  - Global ordering of variable names
  - Each u in required tokens $u{:}\sigma?\tau$ of a method precedes each v in required tokens $v{:}\tau$ and $*v{:}\tau$ of this method
- False positives because of inaccurate aliasing information

Conclusions

- Work in progress (early stage)
- Major goals in the area of component programming:
  - Think in terms of access control (= token availability) instead of synchronization
  - Internal synchronization visible in interfaces (?-tokens)
  - Access control for better factorization of concurrency
  - Subtyping considers concurrency
- Static guarantees desirable, but no major focus
- Model is more dynamic than it seems to be