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;
    ...
}

Places'08: Synchronization as a Special Case of Access Control
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; }
}
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:τ value in variable v is of type τ exclusive access
token replaceable by ?-tokens token unique

v:τ value in variable v is of type τ exclusive access
token not replaceable by ?-tokens sim. ?-tokens

* v:τ value in variable v is of type τ shared read-access
token not replaceable by ?-tokens sim. *-, ?-tokens

v:σ?τ type of value in v unknown no direct access
v lockable if its value of type σ 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 $\nu:\sigma?\tau$ repeatedly invoke a method annotated with $[\nu:\sigma \rightarrow \nu:\tau]$

• Always cycles in $?\text{-tokens}$: $\nu:\tau_0?\tau_1, \nu:\tau_1?\tau_2, \ldots, \nu:\tau_n?\tau_0$

• No $?\text{-token}$ must disappear

• Class-instance variables not associated with $?\text{-tokens}$

• If $\nu$ is in “stop mode”, each method invocation using $\nu:\sigma?\tau$ 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