

From Static to Dynamic Process Types

Franz Puntigam

Vienna University of Technology
Vienna, Austria

`franz@complang.tuwien.ac.at`

`http://www.complang.tuwien.ac.at/franz/`

Static Process Types: Examples

- Static types of expressions in a process calculus translated to more conventional object-oriented languages
- Synchronization based on tokens

```
type Buffer is    token empty filled
                  put (e:E with empty -> filled)
                  get (with filled -> empty): E
```

```
copyElement (b: Buffer[empty filled -> filled.2]) do
    e:E = b.get()           -- b:Buffer[empty.2]
    b.put(e)                -- b:Buffer[empty filled]
    b.put(e)                -- b:Buffer[filled.2]
```

Static Process Types: Some Properties

```
class Buffer1 < Buffer is
  s:E                -- single buffer slot
  put(e:E with empty -> filled) do s=e
  get(with filled->empty): E do return s
  new(): Buffer1[empty] do null
```

- Simple non-blocking synchronization
- Supports constraints on invocation sequences
- Easy program analysis (no need of aliasing information)

Static Process Types: Open Problems

- State changes must be anticipated statically
- No state information on variables shared by several threads

```
type Window is      token displ icon
                    iconify (with displ -> icon)
                    display (with icon -> displ)
```

```
class WManager is
  win:      -- which type? State info changes over time
  button1() do win.iconify()
  button2() do win.display()
```

Dynamic Tokens in When-Clauses

- Pool of dynamically managed tokens per object

```
class BufferDyn50 is
```

```
  token sync empty filled      -- tokens used dynamically
  lst: List                    -- still no state information about list
  new(): BufferDyn50 -> sync empty.50 do lst = List.new()
  put(e:E) when sync empty -> sync filled do ...
  get(): E when sync filled -> sync empty do ...
```

```
class StaticAndDynamic is
```

```
  token t                      -- token used statically and dynamically
  beDynamic (with t->) when ->t do null
  beStatic (with ->t) when t-> do null
  new(): StaticAndDynamic[t] do null
```

Dynamic Typing

- Don't write down static types (including state information)
- Still needed:
 - with-clauses and when-clauses
 - tokens issued on object creation
- Availability of tokens checked at run time

```
class WindowImpl < Window is
  new(): WindowImpl[displ] do ...
class WManager is
  win:                                -- no type specified
  new(w): WManager do win=w          -- no state known
  button1() do win.iconify()         -- type-safe?
  button2() do win.display()
```

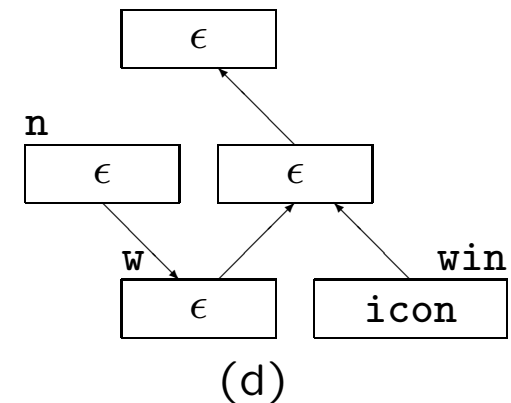
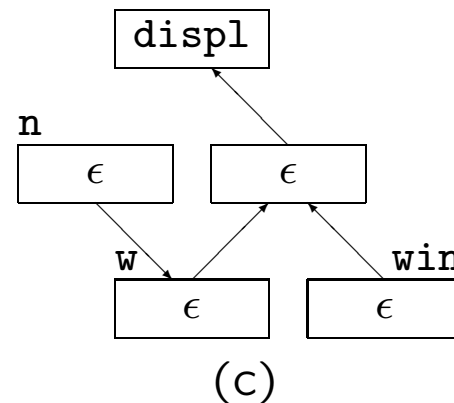
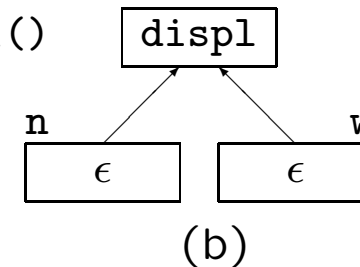
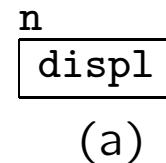
Flexible Dynamic Type Checking

- Flexible (= simple) approach:
With-clauses update a set of static tokens per object
- Flexible approach hides many synchronization mistakes
- Cause of the problem:
Static tokens belong to references, not referenced objects

Strict Dynamic Type Checking

- Add token set and link when splitting variable
- Take token from nearest set
- Add to local set
- Adapt link on return

```
n := WindowImpl.new()  
WManager.new(n).button1()
```



Why Strict Dynamic Type Checking

- Most likely to uncover synchronization problems
- Link structure can be created at compilation time
- Further work: static inference of state information
(= back to static checking with problems solved)

Race-Free Programs

- Easy to ensure race-free programs by analyzing tokens (missing tokens forbid simultaneous variable accesses)
- Both static and dynamic tokens considered
- On class by class basis – only local information needed

Conclusions

- Integration of abstract behavior specification (= restricted message acceptability) and synchronization
- Flexibility through dynamic tokens and dynamic typing
- Dynamic type safety and simpler (local) program analysis
- Hopefully static type safety achievable through state inference (future work based on presented concepts)