Process Types for Active Objects

Franz Puntigam

Institut für Computersprachen
Technische Universität Wien

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

Contents

- Motivation
- Process Type Representation
- Type Equivalence and Subtyping
- Static Type Checking
- Improvements
- Conclusions

Types in Object-Oriented Languages

Object types are

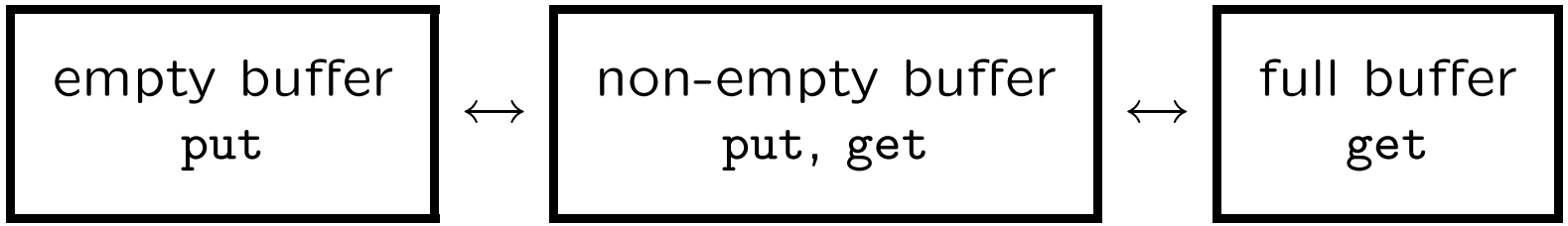
- object signatures
  $\longrightarrow$ specify immutable message sets
  $\longrightarrow$ static type checking can ensure: messages acceptable

- partial specifications of behavior
  $\longrightarrow$ may specify mutable message sets
  $\longrightarrow$ type consistency only dynamically checkable

Goal: static checking + types partially specifying behavior

Concurrency and Synchronization

Synchronization $\Leftrightarrow$ dynamic changes of message acceptability



Usual in statically typed concurrent languages:

- types specify immutable message sets
  $\longrightarrow$ static type checking possible
- acceptance of inappropriate messages delayed
  $\longrightarrow$ messages may never become acceptable

Example for Task Type in Ada

```
task type Buffer is
    entry put(E: in Element);
    entry get(E: out Element);
end Buffer;

task body Buffer is
    X: Element;
begin
    loop
        accept put(E: in Element) do X := E; end;
        accept get(E: out Element) do E := X; end;
    end loop;
end Buffer;
```

Specifying Mutable Message Sets

Process calculus:

$$\mathsf{Buffer}(\underline{u}) \stackrel{u}{=} u\triangleright\{\mathsf{put}(\underline{v})=u\triangleright\{\mathsf{get}(\underline{w})=w\triangleleft\mathsf{back}(v)|\mathsf{Buffer}(u)\}\}$$

Type = abstraction of object behavior:

- same structure as process expression
- only message acceptance considered
- object names replaced with types

$$\mathsf{BufferT} \stackrel{\mathrm{def}}{=} \{\mathsf{put}(\mathsf{ElemT})\rightarrow\{\mathsf{get}(\mathsf{BackT})\rightarrow\{\}|\mathsf{BufferT}\}\}$$
$$\mathsf{BackT} \stackrel{\mathrm{def}}{=} \{\mathsf{back}(\mathsf{ElemT})\rightarrow\{\}\}$$

Problems to be Addressed

- sent message sequence $\neq$ received message sequence

  $\longrightarrow$ keep ordering as far as necessary

- indeterministic object behavior

  $\longrightarrow$ basic types deterministic $\longrightarrow$ trace semantics

- several independent clients

  $\longrightarrow$ split types

- decidability of type equivalence and subtyping

  $\longrightarrow$ more restrictive than equivalence and containment of message sequence sets or these sets are regular

Type Representation

$\mathsf{FBuffer}\langle \mathsf{empty}^7, \mathsf{full}^3 \rangle$	finite buffer, 7 empty and 3 full slots
$\mathsf{IBuffer}\langle \mathsf{full}^3 \rangle$	infinite buffer, at least 3 full slots

$$\begin{aligned}
\mathsf{FBuffer} &\stackrel{\text{def}}{=} \{\mathsf{put}(\mathsf{Elem}\langle\rangle)\langle \mathsf{empty}\rightarrow\mathsf{full}\rangle, \\
&\qquad \mathsf{get}(\mathsf{Back}\langle \mathsf{reply}\rangle)\langle \mathsf{full}\rightarrow\mathsf{empty}\rangle\} \\
\mathsf{IBuffer} &\stackrel{\text{def}}{=} \{\mathsf{put}(\mathsf{Elem}\langle\rangle)\langle \rightarrow\mathsf{full}\rangle,\ \mathsf{get}(\mathsf{Back}\langle \mathsf{reply}\rangle)\langle \mathsf{full}\rightarrow\rangle\} \\
\mathsf{Back} &\stackrel{\text{def}}{=} \{\mathsf{back}(\mathsf{Elem}\langle\rangle)\langle \mathsf{reply}\rightarrow\rangle\}
\end{aligned}$$

Further type expressions: $\sigma||\tau$ $\quad$ $\mathbf{rec}(\underline{t})\,\tau$ $\quad$ t

Basic Operations on Types

Type Updating:

remove/add tokens from/to the type state

Type Splitting:

$$\{\mu_1,\ldots,\mu_m\}\langle \mathsf{p}_1,\ldots,\mathsf{p}_k\rangle \| \{\nu_1,\ldots,\nu_n\}\langle \mathsf{q}_1,\ldots,\mathsf{q}_l\rangle \equiv \{\mu_1,\ldots,\mu_m,\nu_1,\ldots,\nu_n\}\langle \mathsf{p}_1,\ldots,\mathsf{p}_k,\mathsf{q}_1,\ldots,\mathsf{q}_l\rangle$$

if $\mathsf{p}_1,\ldots,\mathsf{p}_k \in \mathsf{rel}(\mu_1,\ldots,\mu_m)$
and $\mathsf{q}_1,\ldots,\mathsf{q}_l \in \mathsf{rel}(\nu_1,\ldots,\nu_n)$

Example: $\mathsf{FBuffer}\langle \mathsf{empty}^3, \mathsf{full}^1\rangle \| \mathsf{FBuffer}\langle \mathsf{empty}^4, \mathsf{full}^2\rangle \equiv \mathsf{FBuffer}\langle \mathsf{empty}^7, \mathsf{full}^3\rangle$

Type Equivalence and Subtyping

Equivalence Relation $\equiv$

- associativity, commutativity, neutral element $\{\}\langle\rangle$ of $\|$
- split/combine types
- ignore irrelevant tokens and redundant descriptors
- fold/unfold recursive types
- rename bound type parameters

Subtyping Relation $\leq$

$$\frac{\sigma \equiv \tau \| \rho}{\sigma \leq \tau}$$

Properties of Equivalence, Subtyping

- Decidable in cubic time
- Soundness and completeness:

$$\mathsf{seq}(\sigma) = \mathsf{seq}(\tau) \;\Leftarrow\; \sigma \equiv \tau \;\Leftrightarrow\; \forall \rho : \mathsf{seq}(\sigma{\uparrow}\rho) = \mathsf{seq}(\tau{\uparrow}\rho)$$
$$\mathsf{seq}(\tau) \subseteq \mathsf{seq}(\sigma) \;\Leftarrow\; \sigma \leq \tau \;\Leftrightarrow\; \forall \rho : \mathsf{seq}(\tau{\uparrow}\rho) \subseteq \mathsf{seq}(\sigma{\uparrow}\rho)$$

- $\mathsf{seq}(\sigma \| \tau)$ contains at least all arbitrary interleavings of sequences in $\mathsf{seq}(\sigma)$ with those in $\mathsf{seq}(\tau)$
- If σ deterministic, $\sigma \leq \tau$, and σ' and τ' constructed by updating σ and τ according to the same message, then $\sigma' \leq \tau'$ holds.

Static Type Checking Algorithm

Servers: ensure for each object of type τ:

- τ is deterministic
- object can accept all message sequences specified by τ

Clients: ensure for each variable v of type τ:

- v is initialized with an instance of a subtype of τ
- τ specifies each message sent to v
- τ is split appropriately where v occurs as argument

Result: no "message-not-understood error" at run time

Static Type System

- type safety is decidable
- type-safe processes cannot be reduced to *error*
  $\longrightarrow$ a process $\ldots|u\triangleright\{\ldots\}|u\triangleleft x(\ldots)|\ldots$ is reducible
- separate compilation
- component replacement
- the class of type-safe processes is large

Dynamic Type Information

Types as Parameters:

$$\begin{aligned}\mathsf{DBuffer} &\stackrel{\text{def}}{=} \{\mathsf{put}(\underline{\mathsf{t}};\mathsf{t})\langle \mathsf{empty}\rightarrow\mathsf{full}\rangle,\\ &\quad\ \mathsf{get}(\mathsf{Back}\langle\mathsf{reply}\rangle)\langle\mathsf{full}\rightarrow\mathsf{empty}\rangle,\\ &\quad\ \mathsf{dup}\langle\mathsf{empty},\mathsf{full}\rightarrow\mathsf{full}^2\rangle\}\\ \mathsf{Back} &\stackrel{\text{def}}{=} \{\mathsf{back}(\underline{\mathsf{t}};\mathsf{t})\langle\mathsf{reply}\rightarrow\rangle\}\end{aligned}$$

Type Comparison: **if** $\mathsf{s}\leq\tau$ **then**$(\underline{\mathsf{t}})$ P **else** Q

within P: $\mathsf{s}\equiv\tau\|\mathsf{t}$

Type Combination: **if** $\mathsf{u}=\mathsf{v}$ **then** P **else** Q

outside of P: u:DBuffer$\langle\mathsf{empty}\rangle$ and v:DBuffer$\langle\mathsf{full}\rangle$

within P: u:DBuffer$\langle\mathsf{empty},\mathsf{full}\rangle$

Type Renewal

$$(\underline{\mathrm{Dup}} : (\underline{u}{:}\mathrm{DBuffer}\langle \mathrm{full}, \mathrm{empty}\rangle, \underline{v}{:}\{\}\langle\rangle) \overset{v}{:} \{\}\langle\rangle)$$

$$(\ \mathrm{Dup}(\underline{u}, \underline{v}) \overset{v}{=} u \triangleleft \mathrm{get}(v) \mid$$
$$v \triangleright \{\mathrm{back}(\underline{t}; \underline{w}) = u \triangleleft \mathrm{put}(t; w) \mid u \triangleleft \mathrm{put}(\{\}\langle\rangle; w)\}$$
$$\ldots\ )$$

Static type checking:

for each occurrence of a self reference as argument:

renew its type with formal parameter type

type of self reference v:	$\{\}\langle\rangle$
formal parameter type:	$\mathrm{Back}\langle \mathrm{reply}\rangle$
renewed type:	$\{\}\langle\rangle \| \mathrm{Back}\langle \mathrm{reply}\rangle$

Synchronized Process Types

Unsynchronized process types:

- acceptability positively depends on token availability
- type splitting without restrictions
- $\mathsf{x}_1^m, \mathsf{x}_2^n, \mathsf{x}_3^m, \mathsf{x}_4^n$ not specifiable

Synchronized process types:

- acceptability can depend on non-availability of tokens
- type splitting restricted
- $\mathsf{x}_1^m, \mathsf{x}_2^n, \mathsf{x}_3^m, \mathsf{x}_4^n$ specifiable

Deadlock Prevention

Optional message:

- can be sent if acceptable
- we can lose types specifying only optional messages

Obligatory message:

- must be sent if acceptable
- we must not lose types specifying optional messages
- static type checking ensures: no deadlocks
- antisymmetric relation on tokens $\longrightarrow$ no cyclic waiting
- can have priority over optional messages

State Inference

Explicit state specification:

- programmer provides static type part and type state
- code is readable and supports separate compilation

State inference:

- programmer provides only static type part
- compiler infers states usually in reasonable time
- exponential complexity for: **if** u=v **then** P **else** Q
- not yet usable together with genericity

Conclusions

- Process types specify synchronization conditions
- Static type system ensures message acceptability
- Coordination of clients without static aliasing information
- Support of subtyping, genericity, dynamic type information

◦ Support of arrays not satisfactory

◦ Difficult to add process types to C++, Java, Ada