CONCURRENT OBJECTS IN IDEALIZED CSP

Stephen Brookes
Department of Computer Science Carnegie Mellon University

July 1998
IDEALIZED CSP

communicating processes
+ call-by-name $\lambda$-calculus

• simply typed

$$\theta ::= \text{var}[\tau] \mid \text{chan}[\tau]$$
$$\mid \text{exp}[\tau] \mid \text{comm}$$
$$\mid \theta \rightarrow \theta' \mid \theta \times \theta'$$

$$\tau ::= \text{int} \mid \text{bool} \mid \text{unit}$$

• asynchronous communication

channels as unbounded buffers

• fair parallel execution

abstracts from network details
CONNECTIONS

• generalizes CSP
  – fairness
  – nested parallelism
  – dynamic process creation
  – asynchronous communication

• generalizes Idealized Algol
  – typed channels
  – communicating processes

• generalizes Kahn networks
  – non-determinism and fairness

• supports concurrent objects
  – parallel methods
  – shared or private state
SYNTAX

• Input

\[ \pi \vdash h : \text{chan}[\tau] \quad \pi \vdash X : \text{var}[\tau] \]

\[ \pi \vdash h ? X : \text{comm} \]

• Output

\[ \pi \vdash h : \text{chan}[\tau] \quad \pi \vdash E : \text{exp}[\tau] \]

\[ \pi \vdash h ! E : \text{comm} \]

• Parallel composition

\[ \pi \vdash P_1 : \text{comm} \quad \pi \vdash P_2 : \text{comm} \]

\[ \pi \vdash P_1 \parallel P_2 : \text{comm} \]

• Local declaration

\[ \pi \vdash D : \pi' \quad \pi, \pi' \vdash P : \text{comm} \]

\[ \pi \vdash \text{local } D \text{ in } P : \text{comm} \]
CATEGORY of WORLDS

Oles, Reynolds

• Objects: countable sets of states

\[ V_1 \times \cdots \times V_k \times H_1^* \times \cdots H_n^* \]

• Morphisms:

\[(f, Q) : W \rightarrow X\]

– function \( f \) from \( X \) to \( W \)
– equivalence relation \( Q \) on \( X \)
– each \( Q \)-class isomorphic to \( W \)

ADAPTATION

• channels as components of state
• communication as state change
EXPANSIONS

• The expansion morphism

\[ - \times V : W \to W \times V \]

is given by

\[ - \times V = (\text{fst} : W \times V \to W, \ Q) \]

\[
(w_0, v_0)Q(w_1, v_1) \iff v_0 = v_1
\]

• Used to model local variables and local channels

• Every morphism is an expansion, modulo isomorphism
SEMANTICS

• Types denote functors from worlds to domains, \([\theta] : W \rightarrow D\)

• Judgements \(\pi \vdash P : \theta\) denote natural transformations

\[\exists\theta : [P] : [[\pi]] \rightarrow [[\theta]]\]

i.e. when \(h : W \rightarrow X\),

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
[[\pi]]W \\
\downarrow \quad \downarrow \quad \downarrow \\
[[\pi]h]X \\
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
[[P]]W \\
\rightarrow \\
[[\theta]]W \\
\downarrow \quad \downarrow \quad \downarrow \\
[[\theta]h]X \\
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
[[P]]X \\
\rightarrow \\
[[\theta]]X \\
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

commutes.

Naturality enforces locality
COMMANDS

\[[\text{comm}]W = \mathcal{O}^\dagger((W \times W)^\infty)\]

- Commands denote closed trace sets
  \[\alpha\beta \in t \land w \in W \Rightarrow \alpha\langle w, w\rangle\beta \in t\]
  \[\alpha\langle w, w'\rangle\langle w', w''\rangle\beta \in t \Rightarrow \alpha\langle w, w''\rangle\beta \in t\]

- A trace \(\langle w_0, w'_0\rangle\langle w_1, w'_1\rangle \ldots \langle w_n, w'_n\rangle \ldots\)
  models a fair interaction

- A step \(\langle w_i, w'_i\rangle\) represents a
  finite sequence of atomic actions
CHANNELS

An “object-oriented” semantics:
• sender

\[ \text{give} : W \rightarrow (W \times V_\tau)\text{option} \]

• receiver

\[ \text{take} : V_\tau \rightarrow (W \rightarrow W) \]

satisfying

\[ \text{give}(\text{take} v w) = \]

\[ \text{case give } w \text{ of} \]

\[ \text{none} : \text{some}(w, v) \]

\[ \text{some}(w', v') : \text{some}(\text{take } v w', v') \]
PARALLEL COMPOSITION

Fair merge of traces

\[ [P_1 \parallel P_2]Wu = \{ \alpha \mid \exists \alpha_1 \in [P_1]Wu, \alpha_2 \in [P_2]Wu. (\alpha_1, \alpha_2, \alpha) \in \text{fairmerge}_{W \times W} \}^\dagger \]

where

\[
\text{fairmerge}_A = \text{both}^*_A \cdot \text{one}_A \cup \text{both}^\omega_A
\]

\[
\text{both}_A = \{ (\alpha, \beta, \alpha\beta), (\alpha, \beta, \beta\alpha) \mid \alpha, \beta \in A^+ \}
\]

\[
\text{one}_A = \{ (\alpha, \epsilon, \alpha), (\epsilon, \alpha, \alpha) \mid \alpha \in A^\infty \}
\]

fairmerge is natural
LOCAL CHANNELS

The traces of

\[ \text{local } h : \text{chan}[\tau] \text{ in } P \]

at \( W \) are projected from the traces of \( P \) at \( W \times V^*_\tau \), in which

- initially \( h = \epsilon \)
- contents of \( h \) never change across step boundaries

EXAMPLES

- local \( h \) in \( (h!0; P) = P \) if \( h \) not free in \( P \)

- local \( h \) in \( (h?x; P) = \text{while true do skip} \)
LAWS

• Symmetry

\[
\text{local } h_1 \text{ in local } h_2 \text{ in } P = \text{local } h_2 \text{ in local } h_1 \text{ in } P
\]

• Scope contraction

\[
\text{local } h \text{ in } (P_1 \parallel P_2) = (\text{local } h \text{ in } P_1) \parallel P_2
\]

if \( h \) not free in \( P_2 \)

... justifies graphical notation for networks of processes
LOCAL LAWS

● Local output

\[
\text{local } h = \rho \ \text{in } P_1 \parallel (h!v; P_2) \\
\quad = \text{local } h = \rho v \ \text{in } P_1 \parallel P_2
\]

if \( h! \) not free in \( P_1 \)

● Local input

\[
\text{local } h = v \rho \ \text{in } P_1 \parallel (h?x; P_2) \\
\quad = \text{local } h = \rho \ \text{in } P_1 \parallel (x:=v; P_2)
\]

if \( h? \) not free in \( P_1 \)

\[\text{... help when channels are uni-directional}\]
FAIRNESS LAWS

• Fair prefix

\[
\text{local } h \text{ in } (h?x; P) \parallel (Q_1; Q_2) = Q_1; \text{local } h \text{ in } (h?x; P) \parallel Q_2
\]
if \( h \) not free in \( Q_1 \)

• Cyclic synchronization

\[
\text{local } h_1, h_2 \text{ in } \begin{align*}
(P_1; h_1!\ast; h_2?\ast; Q_1) \parallel (P_2; h_2!\ast; h_1?\ast; Q_2)
\end{align*}
\text{local } h_1, h_2 \text{ in } (Q_1\parallel Q_2)
\]
if \( h_1, h_2 \) not free in \( P_1, P_2 \)

... require and reflect fair semantics
CLASSES AND OBJECTS

• Declarations as first-class citizens

\[ \pi \vdash D : \pi' \]

• Class is template for declaration:

\[
\text{class } C = \\
\text{private } \pi_1 \\
\text{public } \pi_2
\]

• Object instantiates template:

\[
\text{object } X : C = \text{private } D_1 \\
\text{public } D_2
\]

translates to

\[
\text{local } X.D_1 \text{ in } X.D_2
\]
BUFFER CLASSES

class $Buffer_1$ =

public

$put : \text{exp}[\tau] \rightarrow \text{comm}$

$get : \text{var}[\tau] \rightarrow \text{comm}$

class $Buffer_2 =$

$Buffer_1$ with private data : $\text{chan}[\tau]$

class $Buffer_3 =$

$Buffer_1$ with private data : $\text{var}[\tau]$

SUBCLASSES

$Buffer_2 \leq Buffer_1$

$Buffer_3 \leq Buffer_1$
A BUFFER OBJECT

object $B_1 : Buffer_2 =$

private

\[
\text{empty : chan[unit]} = [\ast]; \\
\text{data : chan[int]}
\]

public

\[
\text{put}(e) = (\text{empty}?\ast; \text{data}!e); \\
\text{get}(z) = (\text{data}?z; \text{empty}!\ast)
\]

PROPERTIES

• $B_1$ has class $Buffer_2$

• $Buffer_2 \leq Buffer_1$

• $B_1$ also has class $Buffer_1$

• $B_1$ behaves like a 1-place buffer
ANOTHER BUFFER

object $B_2 : Buffer_2 = $

private

$empty : chan[unit] = [\ast];$

$data : chan[int]$

public

$put(e) = (empty?\ast; data!(\neg e));$

$get(z) = \text{local } x : \text{var}[\text{int}] \text{ in}$

$(data?x; z:=(-x); empty!\ast)$

PROPERTIES

• Codes and decodes data

• Still behaves like 1-place buffer
object $B_3 : Buffer_3$ =

private

$empty : \text{var}[\text{bool}] = \text{true}$;
$full : \text{var}[\text{bool}] = \text{false}$;
$data : \text{var}[\tau]$

public

$put(e) =$
($\text{await } empty \text{ then } empty := \text{false};$
$data := e;$
$full := \text{false}$);

$get(x) =$
($\text{await } full \text{ then } full := \text{true};$
$x := data;$
$empty := \text{true}$)
EQUIVALENCES

• All three implementations of buffers are “equivalent”
  – no way to tell them apart

• Need to compare across paradigms
  – communicating processes
  – shared-variable

• Trace semantics can be used in both cases
  – all three buffer objects have same trace semantics
  – closure blurs granularity
CONCLUSIONS

• Idealized CSP supports a form of concurrent objects

• Trace semantics validates natural laws of equivalence
  – locality
  – fairness
  – synchronization patterns

• Can compare across paradigms

• Can abstract from granularity
SPECIFICATIONS

spec \texttt{BUFFER} =

\textbf{interface}

\texttt{empty, full : exp[bool]}

\textbf{with}

\{\texttt{empty}\}\,\texttt{put(v)}\{\texttt{full}\}

\{\texttt{full}\}\,\texttt{get(x)}\{\texttt{empty}\}

\texttt{put(v_1)||put(v_2)} =

(\texttt{put(v_1); put(v_2)}) \textbf{or} (\texttt{put(v_2); put(v_1)})

\{\texttt{empty}\}\,(\texttt{get(x_1)||get(x_2)}) =

(\texttt{get(x_1); get(x_2)}) \textbf{or} (\texttt{get(x_2); get(x_1)})

\{\texttt{empty}\}\,(\texttt{put(v)||get(x)}) =

\texttt{put(v); get(x)} = x := v