An Operational Semantics in Isabelle/HOL-CSP
Non Permanents Seminar

Benoît Ballenghien[1]

LMF - Paris-Saclay

December 17, 2024

Outline

What is CSP ?

CSP = Communicating Sequential Processes

- **What**: language to specify and verify patterns of interaction of concurrent systems
- **When**: appeared in the 80s, substantially evolved since
- **Who**: many people, but in particular Brookes, Hoare and Roscoe
- **Field**: process algebras, like CCS and LOTOS.

The Syntax of CSP at a Glance

P	::=	SKIP	can only terminate
	|	STOP	deadlock
	|	P □ P'	external choice: process follows contextual choices
	|	P ⊓ P'	internal choice: context follows process choices
	|	⊓ a∈A. P a	generalized internal choice
	|	P ⟦A⟧ P'	synchronization product
	|	P ; P'	sequential composition
	|	P \ A	hiding operator
	|	Renaming P g	renaming the event e in g e
	|	P △ P'	interruption
	|	P Θ a∈A. P' a	exception handler
	|	a → P	prefix
	|	□a∈A → P a	multi-prefix external choice
	|	μ X. f X	fixed point operator

The Syntax of CSP at a Glance

P	::=	SKIP	can only terminate
	|	STOP	deadlock
	|	P □ P'	external choice: process follows contextual choices
	|	P ⊓ P'	internal choice: context follows process choices
	|	⊓ a∈A. P a	generalized internal choice
	|	P ⟦A⟧ P'	synchronization product
	|	P ; P'	sequential composition
	|	P \ A	hiding operator
	|	Renaming P g	renaming the event e in g e
	|	P △ P'	interruption
	|	P Θ a∈A. P' a	exception handler
	|	a → P	prefix
	|	□a∈A → P a	multi-prefix external choice
	|	μ X. f X	fixed point operator

A special event ✓ denotes termination.

Denotational, Algebraic and Operational Facets

Denotational

$$\mathcal{F}\ (P\ [\![S]\!]\ Q) \equiv \ldots \qquad \mathcal{D}\ (P\ [\![S]\!]\ Q) \equiv \ldots$$

$$P \sqsubseteq_{FD} Q \equiv \mathcal{F}\ Q \subseteq \mathcal{F}\ P \wedge \mathcal{D}\ Q \subseteq \mathcal{D}\ P$$

Algebraic

$$\frac{A \subseteq S \qquad B \subseteq S}{\Box a{\in}A \to P\ a\ [\![S]\!]\ (\Box b{\in}B \to Q\ b) = \Box x{\in}A \cap B \to P\ x\ [\![S]\!]\ Q\ x}$$

Operational

$$\frac{e \notin S \qquad P \leadsto_e P'}{P\ [\![S]\!]\ Q \leadsto_e P'\ [\![S]\!]\ Q} \qquad \frac{P \leadsto_{\checkmark} P' \qquad Q \leadsto_{\tau} Q'}{P\ ;\ Q \leadsto_{\tau} Q'}$$

HOL-CSP in a Nutshell

The HOL-CSP [2] was developed by Safouan Taha, Burkhart Wolff, Lina Ye and more recently Benoît Ballenghien.

Outstanding points:

- process theory
- **no restriction on finiteness**
- **no restriction on types**: we do not construct process but 'a process (parameterized theory)
- HOLCF provides semantics for the least fixed point $\mathrm{P} \equiv \mu\ \mathrm{x.\ f\ x}$
- process refinement, especially FD-refinement $\mathrm{P} \sqsubseteq_{FD} \mathrm{Q}$, is crucial
- denotational construction, algebraic semantics proven

HOL-CSP in a Nutshell

The HOL-CSP [2] was developed by Safouan Taha, Burkhart Wolff, Lina Ye and more recently Benoît Ballenghien.

Outstanding points:

- process theory
- **no restriction on finiteness**
- **no restriction on types**: we do not construct process but 'a process (parameterized theory)
- HOLCF provides semantics for the least fixed point $P \equiv \mu$ x. f x
- process refinement, especially FD-refinement $P \sqsubseteq_{FD} Q$, is crucial
- denotational construction, algebraic semantics proven
- for now, operational semantics is not supported. Beyond the theoretical question, this would be useful for working with LTS, connecting model checkers (FDR) and certify their output, etc.

Overview of HOL-CSP

axiomatisation

HOL

Church's Higher-Order Logic

Overview of HOL-CSP

Isabelle

↓ axiomatisation

HOL — Church's Higher-Order Logic

↓ definitional extension

HOLCF — Scott's Logic for Computable Functions

Overview of HOL-CSP

↓ axiomatisation

HOL — Church's Higher-Order Logic

↓ definitional extension

HOLCF — Scott's Logic for Computable Functions

↓ definitional extension

HOL-CSP — Denotational and algebraic semantics of CSP

Overview of HOL-CSP

Isabelle

↓ axiomatisation

HOL — Church's Higher-Order Logic

↓ definitional extension

HOLCF — Scott's Logic for Computable Functions

↓ definitional extension

HOL-CSP — Denotational and algebraic semantics of CSP

⇣ definitional extension

HOL-CSP_OpSem — Operational semantics of CSP

Summary of the Denotational Construction

datatype 'a event = ev 'a | tick (‹✓›)

Summary of the Denotational Construction

datatype 'a event = ev 'a | tick (‹✓›)

type_synonym 'a trace = ‹'a event list›

type_synonym 'a refusal = ‹'a event set›

type_synonym 'a failure = ‹'a trace × 'a refusal›

type_synonym 'a divergence = ‹'a trace›

type_synonym 'a process_0 = ‹'a failure set × 'a divergence set›

Summary of the Denotational Construction

datatype 'a event = ev 'a | tick (‹✓›)

type_synonym 'a trace = ‹'a event list›

type_synonym 'a refusal = ‹'a event set›

type_synonym 'a failure = ‹'a trace × 'a refusal›

type_synonym 'a divergence = ‹'a trace›

type_synonym 'a process_0 = ‹'a failure set × 'a divergence set›

typedef 'a process = ‹{p :: 'a process_0 . is_process p}›

Summary of the Denotational Construction

datatype 'a event = ev 'a | tick (‹✓›)

type_synonym 'a trace = ‹'a event list›

type_synonym 'a refusal = ‹'a event set›

type_synonym 'a failure = ‹'a trace × 'a refusal›

type_synonym 'a divergence = ‹'a trace›

type_synonym 'a process_0 = ‹'a failure set × 'a divergence set›

typedef 'a process = ‹{p :: 'a process_0 . is_process p}›

instantiation process :: (type) pcpo

Summary of the Denotational Construction

```
datatype 'a event = ev 'a | tick (‹✓›)

type_synonym 'a trace = ‹'a event list›

type_synonym 'a refusal = ‹'a event set›

type_synonym 'a failure = ‹'a trace × 'a refusal›

type_synonym 'a divergence = ‹'a trace›

type_synonym 'a process₀ = ‹'a failure set × 'a divergence set›

typedef 'a process = ‹{p :: 'a process₀ . is_process p}›

instantiation process :: (type) pcpo


lift_definition Ndet :: ‹['a process,'a process] ⇒ 'a process› (infixl ‹⊓› 80)
  is ‹λP Q. (ℱ P ∪ ℱ Q , 𝒟 P ∪ 𝒟 Q)›
```

Summary of the Denotational Construction

datatype 'a event = ev 'a | tick (‹✓›)

type_synonym 'a trace = ‹'a event list›

type_synonym 'a refusal = ‹'a event set›

type_synonym 'a failure = ‹'a trace × 'a refusal›

type_synonym 'a divergence = ‹'a trace›

type_synonym 'a $process_0$ = ‹'a failure set × 'a divergence set›

typedef 'a process = ‹{p :: 'a $process_0$. is_process p}›

instantiation process :: (type) pcpo

lift_definition Ndet :: ‹['a process,'a process] ⇒ 'a process› (infixl ‹⊓› 80)
is ‹λP Q. ($\mathcal{F}$ P ∪ $\mathcal{F}$ Q , $\mathcal{D}$ P ∪ $\mathcal{D}$ Q)›

lemma prefix_Det_Ndet : ‹(a → P) □ (a → Q) = (a → P) ⊓ (a → Q)›

How do we want our Operational Semantics ?

HOL-CSP

How do we want our Operational Semantics ?

HOL-CSP

⇣

P^0 — Captures the set of events P can start with

How do we want our Operational Semantics ?

HOL-CSP

⇣

P^0 — Captures the set of events P can start with

⇣

P after e — Kind of inversion of $e \to P$

How do we want our Operational Semantics ?

HOL-CSP

↓

P^0 — Captures the set of events P can start with

↓

P after e — Kind of inversion of $e \rightarrow P$

↓

$P \leadsto_\tau Q$
$P \leadsto_e Q$
$P \leadsto_\checkmark Q$ — Small steps semantics

How do we want our Operational Semantics ?

HOL-CSP

↓

P^0 — Captures the set of events P can start with

↓

P after e — Kind of inversion of $e \to P$

↓

$P \leadsto_\tau Q$
$P \leadsto_e Q$
$P \leadsto_✓ Q$ — Small steps semantics

↓

$P \leadsto^* s\ Q$ — Big steps semantics

How do we want our Operational Semantics ?

HOL-CSP

P^0 — Captures the set of events P can start with

P after e — Kind of inversion of $e \to P$

$P \rightsquigarrow_\tau Q$
$P \rightsquigarrow_e Q$
$P \rightsquigarrow_\checkmark Q$ — Small steps semantics

$P \rightsquigarrow^* s\ Q$ — Big steps semantics

definitional extensions

The notion of initials

We need to capture the set of events a process P can start with.

$$\mathrm{P}^0 = \{\mathrm{e} \mid \exists \mathrm{s}.\ \mathrm{e} \cdot \mathrm{s} \in \mathcal{T}\ \mathrm{P}\}$$

The notion of initials

We need to capture the set of events a process P can start with.

$$\mathrm{P}^0 = \{\mathrm{e} \mid \exists \mathrm{s}.\ \mathrm{e} \cdot \mathrm{s} \in \mathcal{T}\ \mathrm{P}\}$$

$$\bot^0 = \mathrm{UNIV} \qquad (\mathrm{P}^0 = \emptyset) = (\mathrm{P} = \mathrm{STOP}) \qquad \mathrm{SKIP}^0 = \{\checkmark\}$$

$$(\mathrm{P} \sqcap \mathrm{Q})^0 = \mathrm{P}^0 \cup \mathrm{Q}^0 \qquad (\mathrm{P} \mathbin{\square} \mathrm{Q})^0 = \mathrm{P}^0 \cup \mathrm{Q}^0 \qquad (\mathrm{e} \rightarrow \mathrm{P})^0 = \{\mathrm{ev\ e}\}$$

Definition of After

When $\text{ev } e \in P^0$, we define:

- $\mathcal{F}\ (P \text{ after } e) \equiv \{(s, X) \mid (\text{ev } e \cdot s, X) \in \mathcal{F}\ P\}$
- $\mathcal{D}\ (P \text{ after } e) \equiv \{s \mid \text{ev } e \cdot s \in \mathcal{D}\ P\}$.

Definition of After

When ev e $\in P^0$, we define:

- $\mathcal{F}$ (P after e) $\equiv$ {(s, X) | (ev e · s, X) $\in \mathcal{F}$ P}
- $\mathcal{D}$ (P after e) $\equiv$ {s | ev e · s $\in \mathcal{D}$ P}.

and handle the case ev e $\notin P^0$ with a locale (parameterized theory).

```
locale After =  fixes Ψ :: ‹['a process, 'a] ⇒ 'a process› begin

lift_definition After :: ‹['a process, 'a] ⇒ 'a process› (infixl ‹after› 77)
  is ‹λP e.    if ev e ∈ P⁰
               then ({(s, X). (ev e # s, X) ∈ ℱ P}, {s. ev e # s ∈ 𝒟 P})
               else (ℱ (Ψ P e), 𝒟 (Ψ P e))›
```

Easy Properties of After

$\bot$ after e $= \bot$

STOP after e $= \Psi$ STOP e

a $\rightarrow$ P after e $=$ (if e $=$ a then P else Ψ (a $\rightarrow$ P) e)

P $\sqcap$ Q after e $=$
(if ev e $\in P^0 \cap Q^0$ then (P after e) $\sqcap$ (Q after e)
else if ev e $\in P^0$ then P after e
else if ev e $\in Q^0$ then Q after e else Ψ (P $\sqcap$ Q) e)

Hard Properties of After

$(P \; [\![S]\!] \; Q)$ after $e =$
(if $P = \bot \lor Q = \bot$ then $\bot$
else if $ev\ e \in P^0 \cap Q^0$
then if $e \in S$ then P after $e \; [\![S]\!] \; Q$ after e
else $(P$ after $e \; [\![S]\!] \; Q) \sqcap (P \; [\![S]\!] \; Q$ after $e)$
else if $ev\ e \in P^0 \land e \notin S$ then P after $e \; [\![S]\!] \; Q$
else if $ev\ e \in Q^0 \land e \notin S$ then $P \; [\![S]\!] \; Q$ after e
else $\Psi \; (P \; [\![S]\!] \; Q) \; e)$

Constraints on Transitions

We want that:

- the τ transition behaves like the FD-refinement ($\sqsubseteq_{FD}$)
- $P \leadsto_{e} Q$ (resp. $P \leadsto_{\checkmark} Q$) is impossible if ev $e \notin P^0$ (resp. $\checkmark \notin P^0$)
- event transitions should absorb τ transitions
- since $P \sqsubseteq_{FD} Q$ can be interpreted as "Q is more deterministic than P", Q should be at least as deterministic as P after e when P makes a transition via event e.

Locale for Transitions

```
locale OpSemTransitions = AfterExt Ψ Ω
     for Ψ :: ‹['a process, 'a] ⇒ 'a process› and Ω :: ‹'a process ⇒ 'a process› +
   fixes τ_trans :: ‹['a process, 'a process] ⇒ bool› (infixl ‹↝τ› 50)
assumes τ_trans_NdetL:                 ‹P ⊓ Q ↝τ P›
    and τ_trans_transitivity:          ‹P ↝τ Q ⟹ Q ↝τ R ⟹ P ↝τ R›
    and τ_trans_anti_mono_initials: ‹P ↝τ Q ⟹ Q⁰ ⊆ P⁰›
    and τ_trans_mono_AfterExt:       ‹e ∈ Q⁰ ⟹ P ↝τ Q ⟹ P after✓ e ↝τ Q after✓ e›
begin

abbreviation ev_trans :: ‹['a process, 'a, 'a process] ⇒ bool› (‹_ ↝_ _› [50, 3, 51] 50)
  where ‹P ↝e Q ≡ ev e ∈ P⁰ ∧ P after✓ ev e ↝τ Q›

abbreviation tick_trans :: ‹['a process, 'a process] ⇒ bool› (‹_ ↝✓ _› [50, 51] 50)
  where ‹P ↝✓ Q ≡ ✓ ∈ P⁰ ∧ P after✓ ✓ ↝τ Q›
```

Prove an Operational Rule : Methodology

The assumptions of the locale and the work done on After are already enough to derive some of the operational rules (for SKIP, e → P, □a∈A → P a, P ⊓ Q and μ x. f x we are exhaustive).

Prove an Operational Rule : Methodology

The assumptions of the locale and the work done on After are already enough to derive some of the operational rules (for SKIP, e $\rightarrow$ P, $\Box$a$\in$A $\rightarrow$ P a, P $\sqcap$ Q and μ x. f x we are exhaustive).

For the remaining laws of a given operator, we have to add an additional assumption about τ transition e. g. we add P $\leadsto_\tau$ P' $\Longrightarrow$ P $\Box$ Q $\leadsto_\tau$ P' $\Box$ Q.
With this, we can derive the missing laws.

Prove an Operational Rule : Example

```
locale OpSemTransitionsSync = OpSemTransitions Ψ Ω ‹(⇝τ)›
    for Ψ :: ‹['a process, 'a] ⇒ 'a process›
    and Ω :: ‹'a process ⇒ 'a process›
    and τ_trans :: ‹['a process, 'a process] ⇒ bool› (infixl ‹⇝τ› 50) +
assumes τ_trans_SyncL : ‹P ⇝τ P' ⟹ P ⟦S⟧ Q ⇝τ P' ⟦S⟧ Q›
begin

lemma τ_trans_SyncR : ‹Q ⇝τ Q' ⟹ P ⟦S⟧ Q ⇝τ P ⟦S⟧ Q'›
  by (metis Sync_commute τ_trans_SyncL)

lemma ev_trans_SyncL : ‹e ∉ S ⟹ P ⇝e P' ⟹ P ⟦S⟧ Q ⇝e P' ⟦S⟧ Q›
  ...

lemma ev_trans_SyncR : ‹e ∉ S ⟹ Q ⇝e Q' ⟹ P ⟦S⟧ Q ⇝e P ⟦S⟧ Q'›
  by (metis Sync_commute ev_trans_SyncL)

lemma ev_trans_SyncLR : ‹e ∈ S ⟹ P ⇝e P' ⟹ Q ⇝e Q' ⟹ P ⟦S⟧ Q ⇝e P' ⟦S⟧ Q'›
  ...
```

Sample of Proven Rules

$$\frac{P \leadsto_\tau P'}{P \llbracket S \rrbracket Q \leadsto_\tau P' \llbracket S \rrbracket Q} \qquad \frac{e \notin S \qquad P \leadsto_e P'}{P \llbracket S \rrbracket Q \leadsto_e P' \llbracket S \rrbracket Q} \qquad \frac{P \leadsto_\checkmark P'}{P \llbracket S \rrbracket Q \leadsto_\tau SKIP \llbracket S \rrbracket Q}$$

$$\frac{Q \leadsto_\tau Q'}{P \llbracket S \rrbracket Q \leadsto_\tau P \llbracket S \rrbracket Q'} \qquad \frac{e \notin S \qquad Q \leadsto_e Q'}{P \llbracket S \rrbracket Q \leadsto_e P \llbracket S \rrbracket Q'} \qquad \frac{Q \leadsto_\checkmark Q'}{P \llbracket S \rrbracket Q \leadsto_\tau P \llbracket S \rrbracket SKIP}$$

$$\frac{e \in S \qquad P \leadsto_e P' \qquad Q \leadsto_e Q'}{P \llbracket S \rrbracket Q \leadsto_e P' \llbracket S \rrbracket Q'} \qquad \frac{}{SKIP \llbracket S \rrbracket SKIP \leadsto_\checkmark \Omega\ SKIP}$$

These rules are formally proven!

Interpretation of the locale

Our locale can actually be fully instantiated (interpreted in Isabelle's jargon) with two refinements:

- the FD-refinement $\sqsubseteq_{FD}$ (cf Jifeng He and Tony Hoare [1])
- the DT-refinement $\sqsubseteq_{DT}$

Interpretation of the locale

Our locale can actually be fully instantiated (interpreted in Isabelle's jargon) with two refinements:

- the FD-refinement $\sqsubseteq_{FD}$ (cf Jifeng He and Tony Hoare [1])
- the DT-refinement $\sqsubseteq_{DT}$

	basic	$\backslash$	$\Box$	$\rhd$	$\triangle$	;	$[\![\ ]\!]$	Throw	Renaming
$\sqsubseteq_{FD}$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\sqsubseteq_{DT}$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\sqsubseteq_{T}$	✓	✓	✓	✓	✓	✗	✗	✗	✗
$\sqsubseteq_{F}$	✓	✓	✗	✗	✗	✗	✗	✗	✗

Inductive Generalizations

Inductive generalizations of After_{tick} and small steps transitions.

$$\text{P after}_{\checkmark}\ \text{e} \longrightarrow \text{P after}_{\mathcal{T}}\ \text{s}$$

$$\begin{array}{c}\text{P} \leadsto_{\tau} \text{Q} \\ \text{P} \leadsto_{\text{e}} \text{Q} \\ \text{P} \leadsto_{\checkmark} \text{Q}\end{array} \longrightarrow \text{P} \leadsto^{*}\text{s Q}$$

Inductive Generalizations

Inductive generalizations of After_{tick} and small steps transitions.

$$P \text{ after}_{\checkmark} \text{ e} \longrightarrow P \text{ after}_{\mathcal{T}} \text{ s}$$

$$\begin{array}{c} P \leadsto_{\tau} Q \\ P \leadsto_{e} Q \\ P \leadsto_{\checkmark} Q \end{array} \longrightarrow P \leadsto^{*} s\ Q$$

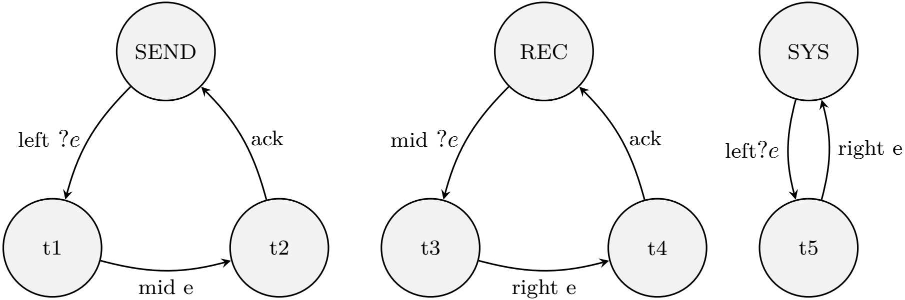
Reality checks (under reasonable assumptions) :

- $P \leadsto^{*} s\ Q$ if and only if $s \in \mathcal{T}\ P \wedge P \text{ after}_{\mathcal{T}}\ s \leadsto_{\tau} Q$
- $s \in \mathcal{T}\ P$ if and only if $\exists Q.\ P \leadsto^{*} s\ Q$.
- $s \in \mathcal{D}\ P$ if and only if $P \leadsto^{*} s\ \bot$.
- $(s, X) \in \mathcal{F}\ P$ if and only if $\exists Q.\ P \leadsto^{*} s\ Q \wedge X \in \mathcal{R}\ Q$.

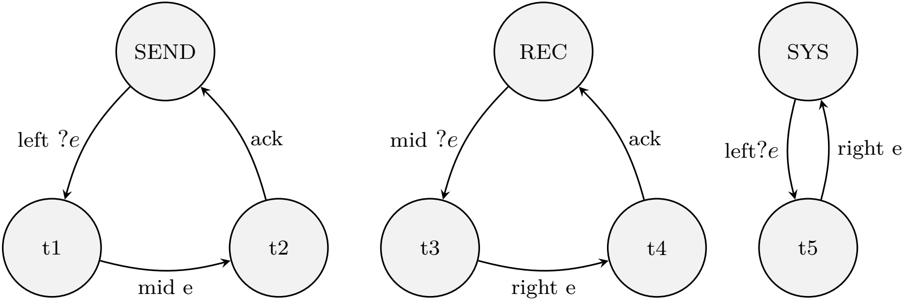
Copy Buffer : Definitions

```
datatype 'a channel = left 'a | right 'a | mid 'a | ack

definition SYN :: ‹'a channel set›
  where ‹SYN ≡ range mid ∪ {ack}›

definition COPY :: ‹'a channel process›
  where ‹COPY ≡ μ COPY. left?x → (right!x → COPY)›

definition SEND :: ‹'a channel process›
  where ‹SEND ≡ μ SEND. left?x → (mid!x → (ack → SEND))›

definition REC :: ‹'a channel process›
  where ‹REC  ≡ μ REC. mid?x → (right!x → (ack → REC))›

definition SYSTEM :: ‹'a channel process›
  where ‹SYSTEM ≡ SEND ⟦SYN⟧ REC \ SYN›
```

Copy Buffer : LTS



Copy Buffer : LTS

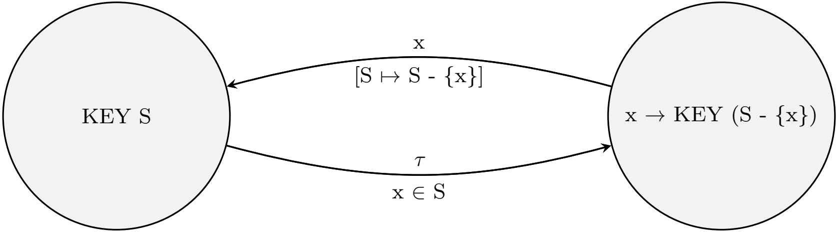


Note that t1 is a key for the term mid e $\rightarrow$ ack $\rightarrow$ SEND, t2 for ack $\rightarrow$ SEND, t3 for right e $\rightarrow$ ack $\rightarrow$ REC, t4 for ack $\rightarrow$ REC, and finally t5 for right e $\rightarrow$ SYS.

Key Generator

We define a non-deterministic key generator such that
KEY S ≡ ⊓x ∈ S → KEY (S - {x})

We prove for example:
[15, 18, 13, ...] ∈ $\mathcal{T}$ (KEY ℕ)



Comparison with Jifeng He and Tony Hoare

In 1993, Jifeng He and Tony Hoare [1] chose to define:

- $P \leadsto_{\tau} Q \equiv P \sqsubseteq_{FD} Q$
- $P \leadsto_{e} Q \equiv P \sqsubseteq_{FD} (e \rightarrow Q) \square P$

We prove these definitions to be equivalent to ours (even generalized in the locale).

Comparison with Jifeng He and Tony Hoare

In 1993, Jifeng He and Tony Hoare [1] chose to define:

- $P \leadsto_{\tau} Q \equiv P \sqsubseteq_{FD} Q$
- $P \leadsto_{e} Q \equiv P \sqsubseteq_{FD} (e \rightarrow Q) \square P$

We prove these definitions to be equivalent to ours (even generalized in the locale).

Advantages of our approach:

- deal with ✓
- direct access to the least deterministic process with After_{tick}
- After_{trace} induction proof technique
- prerequisite to execute processes symbolically.

Conclusion

Isabelle → HOL → HOLCF → HOL-CSP → HOL-CSP_OpSem

P^0 P after e   P after$_{\checkmark}$ e   P after$_{\mathcal{T}}$ e

$P \leadsto_{\tau} Q$ $P \leadsto_{e} Q$ $P \leadsto_{\checkmark} Q$ $P \leadsto^{*}s\ Q$

locale interpreted with $\sqsubseteq_{FD}$ and $\sqsubseteq_{DT}$

Conclusion

Isabelle → HOL → HOLCF → HOL-CSP → HOL-CSP_OpSem

P^0 P after e $P \text{ after}_{\checkmark} \text{ e}$ $P \text{ after}_{\mathcal{T}} \text{ e}$

$P \leadsto_{\tau} Q$ $P \leadsto_{e} Q$ $P \leadsto_{\checkmark} Q$ $P \leadsto^{*} s\ Q$

locale interpreted with $\sqsubseteq_{FD}$ and $\sqsubseteq_{DT}$

The operational rules are formally derived from a definitional extension of HOL-CSP where the bridge definitions for small steps semantics are equivalent to the choice made by Jifeng He and Tony Hoare.

The construction is therefore correct by design, and we can speak of completeness since all the classic laws of literature have been recovered..

Perspectives

- Connect HOL-CSP with FDR: certify the output
- Connect HOL-CSP with Interaction Trees
- Make process executable
- Generate scenarios for cyber-physical systems
- ...

Proven Laws I

$$\frac{\text{P} \leadsto_{\text{e}} \text{P'} \qquad \text{P'} \leadsto_{\tau} \text{P''}}{\text{P} \leadsto_{\text{e}} \text{P''}} \qquad \frac{\text{P} \leadsto_{\tau} \text{P'} \qquad \text{P'} \leadsto_{\text{e}} \text{P''}}{\text{P} \leadsto_{\text{e}} \text{P''}}$$

$$\frac{\text{P} \leadsto_{\checkmark} \text{P'} \qquad \text{P'} \leadsto_{\tau} \text{P''}}{\text{P} \leadsto_{\checkmark} \text{P''}} \qquad \frac{\text{P} \leadsto_{\tau} \text{P'} \qquad \text{P'} \leadsto_{\checkmark} \text{P''}}{\text{P} \leadsto_{\checkmark} \text{P''}}$$

ABSORPTION

$$\frac{}{\text{SKIP} \leadsto_{\checkmark} \Omega\ \text{SKIP}}\ \text{SKIP}$$

$$\frac{\text{cont f} \qquad \text{P} = (\mu\ \text{x. f x})}{\text{P} \leadsto_{\tau} \text{f P}}\ \text{FIXED POINT}$$

Proven Laws II

$$\frac{}{e \rightarrow P \rightsquigarrow_e P} \qquad \frac{e \in A}{\Box a{\in}A \rightarrow P\ a \rightsquigarrow_e P\ e}$$

PREFIX

$$\frac{}{P \sqcap Q \rightsquigarrow_\tau P} \qquad \frac{}{P \sqcap Q \rightsquigarrow_\tau Q} \qquad \frac{e \in A}{\sqcap\ a{\in}A.\ P\ a \rightsquigarrow_\tau P\ e}$$

INTERNAL CHOICE

$$\frac{P \rightsquigarrow_\tau P'}{P \Box Q \rightsquigarrow_\tau P' \Box Q} \qquad \frac{P \rightsquigarrow_e P'}{P \Box Q \rightsquigarrow_e P'} \qquad \frac{P \rightsquigarrow_\checkmark P'}{P \Box Q \rightsquigarrow_\checkmark \Omega\ SKIP}$$

$$\frac{Q \rightsquigarrow_\tau Q'}{P \Box Q \rightsquigarrow_\tau P \Box Q'} \qquad \frac{Q \rightsquigarrow_e Q'}{P \Box Q \rightsquigarrow_e Q'} \qquad \frac{Q \rightsquigarrow_\checkmark Q'}{P \Box Q \rightsquigarrow_\checkmark \Omega\ SKIP}$$

EXTERNAL CHOICE

Proven Laws III

$$\frac{}{P \rhd Q \rightsquigarrow_{\tau} Q} \qquad \frac{P \rightsquigarrow_{\tau} P'}{P \rhd Q \rightsquigarrow_{\tau} P' \rhd Q} \qquad \frac{P \rightsquigarrow_{e} P'}{P \rhd Q \rightsquigarrow_{e} P'} \qquad \frac{P \rightsquigarrow_{\checkmark} P'}{P \rhd Q \rightsquigarrow_{\checkmark} \Omega\ \text{SKIP}}$$

SLIDING CHOICE

$$\frac{P \rightsquigarrow_{\tau} P'}{P\ ;\ Q \rightsquigarrow_{\tau} P'\ ;\ Q} \qquad \frac{P \rightsquigarrow_{e} P'}{P\ ;\ Q \rightsquigarrow_{e} P'\ ;\ Q} \qquad \frac{P \rightsquigarrow_{\checkmark} P' \qquad Q \rightsquigarrow_{\tau} Q'}{P\ ;\ Q \rightsquigarrow_{\tau} Q'}$$

SEQUENTIAL COMPOSITION

Proven Laws IV

$$\frac{P \leadsto_{\tau} P'}{P \setminus B \leadsto_{\tau} P' \setminus B}$$

$$\frac{P \leadsto_{\checkmark} P'}{P \setminus B \leadsto_{\checkmark} \Omega\ SKIP}$$

$$\frac{e \notin B \qquad P \leadsto_{e} P'}{P \setminus B \leadsto_{e} P' \setminus B}$$

$$\frac{e \in B \qquad P \leadsto_{e} P'}{P \setminus B \leadsto_{\tau} P' \setminus B}$$

HIDING

$$\frac{P \leadsto_{\tau} P'}{P \llbracket S \rrbracket Q \leadsto_{\tau} P' \llbracket S \rrbracket Q}$$

$$\frac{e \notin S \qquad P \leadsto_{e} P'}{P \llbracket S \rrbracket Q \leadsto_{e} P' \llbracket S \rrbracket Q}$$

$$\frac{P \leadsto_{\checkmark} P'}{P \llbracket S \rrbracket Q \leadsto_{\tau} SKIP \llbracket S \rrbracket Q}$$

$$\frac{Q \leadsto_{\tau} Q'}{P \llbracket S \rrbracket Q \leadsto_{\tau} P \llbracket S \rrbracket Q'}$$

$$\frac{e \notin S \qquad Q \leadsto_{e} Q'}{P \llbracket S \rrbracket Q \leadsto_{e} P \llbracket S \rrbracket Q'}$$

$$\frac{Q \leadsto_{\checkmark} Q'}{P \llbracket S \rrbracket Q \leadsto_{\tau} P \llbracket S \rrbracket SKIP}$$

$$\frac{e \in S \qquad P \leadsto_{e} P' \qquad Q \leadsto_{e} Q'}{P \llbracket S \rrbracket Q \leadsto_{e} P' \llbracket S \rrbracket Q'}$$

$$\frac{}{SKIP \llbracket S \rrbracket SKIP \leadsto_{\checkmark} \Omega\ SKIP}$$

SYNCHRONIZATION

Proven Laws V

$$\frac{P \rightsquigarrow_{\tau} P'}{P \triangle Q \rightsquigarrow_{\tau} P' \triangle Q} \qquad \frac{P \rightsquigarrow_{e} P'}{P \triangle Q \rightsquigarrow_{e} P' \triangle Q} \qquad \frac{P \rightsquigarrow_{\checkmark} P'}{P \triangle Q \rightsquigarrow_{\checkmark} \Omega\ SKIP}$$

$$\frac{Q \rightsquigarrow_{\tau} Q'}{P \triangle Q \rightsquigarrow_{\tau} P \triangle Q'} \qquad \frac{Q \rightsquigarrow_{e} Q'}{P \triangle Q \rightsquigarrow_{e} Q'} \qquad \frac{Q \rightsquigarrow_{\checkmark} Q'}{P \triangle Q \rightsquigarrow_{\checkmark} \Omega\ SKIP}$$

INTERRUPT

$$\frac{P \rightsquigarrow_{\tau} P'}{P\ \Theta\ a{\in}A.\ Q\ a \rightsquigarrow_{\tau} P'\ \Theta\ a{\in}A.\ Q\ a} \qquad \frac{P \rightsquigarrow_{\checkmark} P'}{P\ \Theta\ a{\in}A.\ Q\ a \rightsquigarrow_{\checkmark} \Omega\ SKIP}$$

$$\frac{e \notin A \qquad P \rightsquigarrow_{e} P'}{P\ \Theta\ a{\in}A.\ Q\ a \rightsquigarrow_{e} P'\ \Theta\ a{\in}A.\ Q\ a} \qquad \frac{e \in A \qquad P \rightsquigarrow_{e} P'}{P\ \Theta\ a{\in}A.\ Q\ a \rightsquigarrow_{e} Q\ e}$$

THROW

Proven Laws for Renaming

$$\frac{P\ {}_{\alpha}\!\leadsto_{\tau}\ P'}{\text{Renaming } P\ f\ {}_{\beta}\!\leadsto_{\tau}\ \text{Renaming } P'\ f}$$

$$\frac{f\ a = b \qquad P\ {}_{\alpha}\!\leadsto_{a}\ P'}{\text{Renaming } P\ f\ {}_{\beta}\!\leadsto_{b}\ \text{Renaming } P'\ f} \qquad \frac{P\ {}_{\alpha}\!\leadsto_{\checkmark}\ P'}{\text{Renaming } P\ f\ {}_{\beta}\!\leadsto_{\checkmark}\ \Omega_{\beta}\ \text{SKIP}}$$

RENAMING

Minor Differences with Roscoe

For the termination of the synchronization.

- Roscoe's version:

$$\frac{P \rightsquigarrow_{\checkmark} P'}{P \ [\![S]\!] \ Q \rightsquigarrow_{\tau} \Omega' \ [\![S]\!] \ Q} \qquad \frac{Q \rightsquigarrow_{\checkmark} Q'}{P \ [\![S]\!] \ Q \rightsquigarrow_{\tau} P \ [\![S]\!] \ \Omega'} \qquad \Omega' \ [\![S]\!] \ \Omega' \rightsquigarrow_{\checkmark} \Omega'$$

Minor Differences with Roscoe

For the termination of the synchronization.

- Roscoe's version:

$$\frac{P \rightsquigarrow_{\checkmark} P'}{P \llbracket S \rrbracket Q \rightsquigarrow_{\tau} \Omega' \llbracket S \rrbracket Q} \qquad \frac{Q \rightsquigarrow_{\checkmark} Q'}{P \llbracket S \rrbracket Q \rightsquigarrow_{\tau} P \llbracket S \rrbracket \Omega'} \qquad \Omega' \llbracket S \rrbracket \Omega' \rightsquigarrow_{\checkmark} \Omega'$$

- Our version:

$$\frac{P \rightsquigarrow_{\checkmark} P'}{P \llbracket S \rrbracket Q \rightsquigarrow_{\tau} \text{SKIP} \llbracket S \rrbracket Q} \qquad \frac{Q \rightsquigarrow_{\checkmark} Q'}{P \llbracket S \rrbracket Q \rightsquigarrow_{\tau} P \llbracket S \rrbracket \text{SKIP}}$$

$$\frac{}{\text{SKIP} \llbracket S \rrbracket \text{SKIP} \rightsquigarrow_{\checkmark} \Omega\ \text{SKIP}}$$

Difficulties for generating automatically a LTS I

```
ML ‹
fun mk_instantiated_OpSem_rules ct exempted =
    let val exempted_match =
            List.find (fn (x, _, _) => Thm.term_of ct = Thm.term_of x) exempted
     in case exempted_match
          of SOME (_, _, thms) => thms
          |  NONE =>
             let val dest2 = dest_last_two_args
                 val dest3 = dest_last_three_args
                 val dest_fp = Thm.dest_arg o Thm.dest_arg
              in case Thm.term_of ct
                   of Const (const_name ‹Pcpo.pcpo_class.bottom›,
                             Type (type_name ‹Process.process›, _))    => [@{thm
BOT_OpSem_rule}]
                      (* ⊥ alone is not sufficient, ‹⊥ :: 'a :: pcpo› would be recognized *)
                   |  Const (const_name ‹STOP›,        _)              => []
                   |  Const (const_name ‹SKIP›,        _)              => [@{thm
SKIP_OpSem_rule}]
                  |  Const (const_name ‹write0›,      _) $ _ $ _       => mk_write0_instantiated
(dest2 ct)
```

Difficulties for generating automatically a LTS II

```
                  | Const (const_name ‹Mprefix›,     _) $ _ $ _      =>
mk_Mprefix_instantiated     (dest2 ct)
                  | Const (const_name ‹Mndetprefix›, _) $ _ $ _      =>
mk_Mndetprefix_instantiated (dest2 ct)
                  | Const (const_name ‹read›,        _) $ _ $ _ $ _  => mk_read_instantiated
(dest3 ct)
                  | Const (const_name ‹write›,       _) $ _ $ _ $ _  => mk_write_instantiated
(dest3 ct)
                    (* TODO : specify the type like we do for ⊥ *)
                  | Const (const_name ‹Cfun.cfun.Rep_cfun›, _)  $
                    Const (const_name ‹Fix.fix›, _) $
                    (Const (const_name ‹Cfun.cfun.Abs_cfun›, _) $ _) =>
mk_fix_point_instantiated   (dest_fp ct)
                  | Const (const_name ‹Ndet›,        _) $ _ $ _      => mk_Ndet_instantiated
(dest2 ct)
                  | Const (const_name ‹GlobalNdet›,  _) $ _ $ _      =>
mk_GlobalNdet_instantiated  (dest2 ct)
                  | Const (const_name ‹Det›,         _) $ _ $ _      => mk_Det_instantiated
(dest2 ct)
                  | Const (const_name ‹Sliding›,     _) $ _ $ _      =>
mk_Sliding_instantiated     (dest2 ct)
```

Difficulties for generating automatically a LTS III

```
             | Const (const_name ‹Seq›,        _) $ _ $ _       => mk_Seq_instantiated
(dest2 ct)
             | Const (const_name ‹Hiding›,     _) $ _ $ _       =>
mk_Hiding_instantiated     (dest2 ct)
             | Const (const_name ‹Sync›,       _) $ _ $ S $ _  =>
               let val (clhs, cS, crhs) = dest3 ct
                in case S of Const (const_name ‹top›, _) => mk_Par_instantiated   (clhs,
crhs)
                       | Const (const_name ‹bot›, _) => mk_Inter_instantiated (clhs, crhs)
                         | _ => mk_Sync_instantiated (clhs, cS, crhs)
               end
             | Const (const_name ‹Interrupt›,  _) $ _ $ _       =>
mk_Interrupt_instantiated  (dest2 ct)
             | Const (const_name ‹Throw›,      _) $ _ $ _ $ _  =>
mk_Throw_instantiated      (dest3 ct)
             | _ => raise CTERM ("Operator of cterm not recognized for generation of
instantiated rules.", [ct])
         end
     end
›
```

[1] H. Jifeng and C. Hoare. From algebra to operational semantics. *Information Processing Letters*, 45(2):75–80, 1993.

[2] S. Taha, L. Ye, and B. Wolff. HOL-CSP Version 2.0. *Archive of Formal Proofs*, Apr. 2019. ISSN 2150-914x. URL http://isa-afp.org/entries/HOL-CSP.html.