The HOL-CSP Project: A Comprehensive Formal Theory of Concurrency

Burkhart Wolff, with contributions of
Safouan Taha, Benoit Ballenghien, Adrien Durier, Benjamin Puyobro,
University Paris-Saclay, LMF

Part I:

Objective: Comprehensiveness

Concurrency Theory
a plethora of concepts

Trace

Algebra

Parallel

Run

Process

Transition System

Thread

State

Event

Synchronised

Lifelock

Interleave

Non-Termination

State Explosion

Deadlock

Model-Checking

Process Refinement

How can we explain and study these concepts

- In a uniform, friendly way ?
- … based on common definitions of the concepts ?
- … on unified and a clear mathematical foundation ?

Part I:

Objective:
A Formal Development

Formalism and Theory

- A personal statement :

  I am a Formalist
  — I believe that only formal proofs based on formalised definitions and proofs give some form of scientific objectivity

  — above all, a proof SHOULD be reproducible like a physical experiment - independent from human subjectivity and social processes

  — This is not just a spleen, it is a epistemological position.

- The Philosophical Counter-Position: Intuitionism
  — (Brower): the truth of a mathematical statement is a subjective claim: a mathematical statement corresponds to a mental construction, and a mathematician can assert the truth of a statement only by verifying the validity of that construction by intuition.

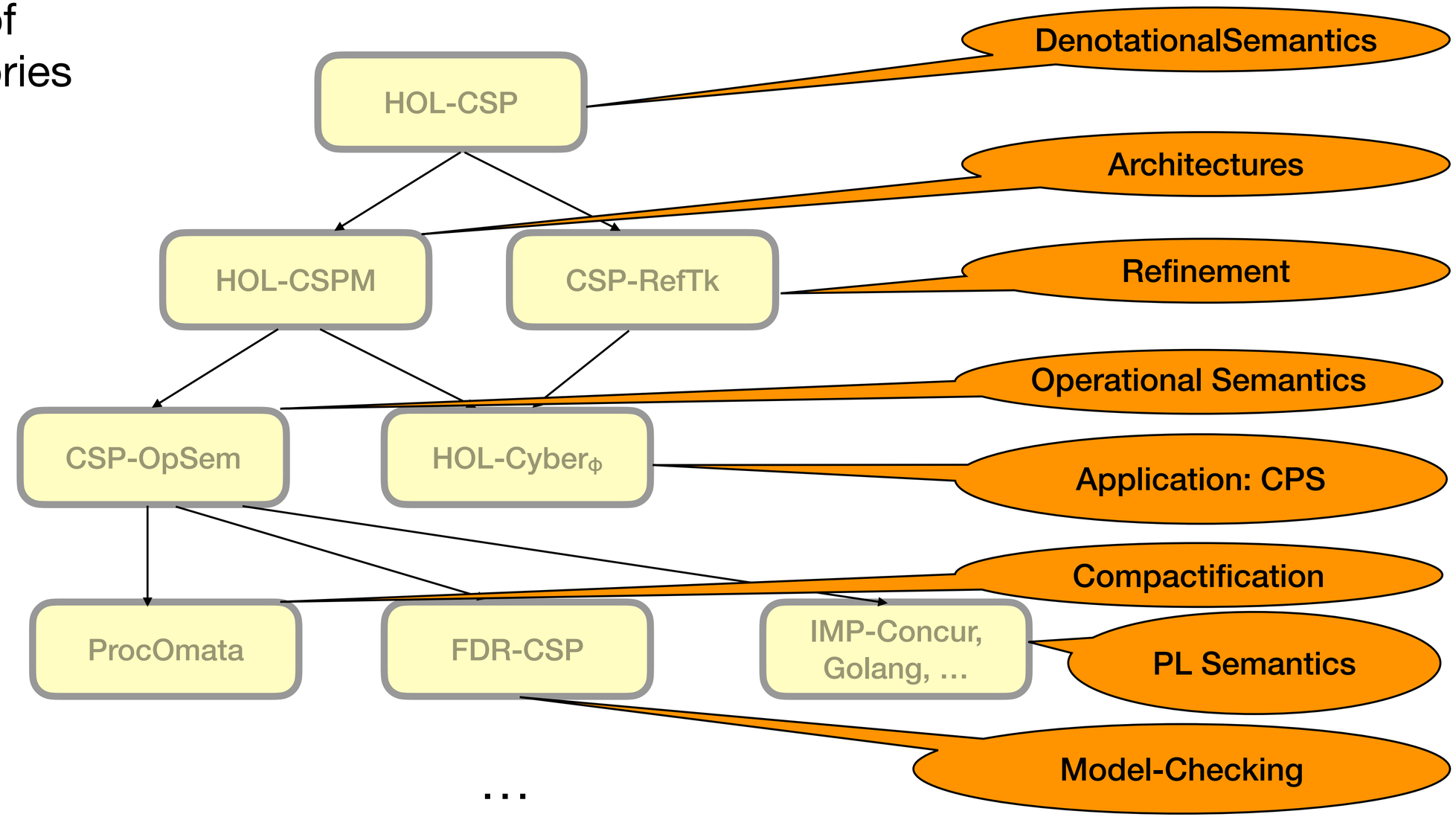
- Prominent Supporters: Einstein
  — „Der intuitive Geist ist ein heiliges Geschenk und der rationale Verstand ein treuer Diener.”

Part I:

Objective:
A Formal Development

HOL-CSP, CSP_M, ProcOmata, HOL-$Cyber_\Phi$

- Architecture of the CSP Theories



Part III:

Objective:
Theory of Concurrency
(with arbitrary data)

HOL-CSP

- A Theory in Isabelle/HOL for Communicating Sequential Processes
- Based on work of Hoare and Roscoe in the 80ies and 90ies, but still a “gold standard” for modelling concurrent systems
- HOL-CSP: a conservative embedding of the CSP theory following essentially the presentation in Bill Roscoe Book ”Theory and Practice of Concurrency”

Classic CSP

- The Core:
  A (finite) set of “events” Σ, $A \subseteq \Sigma$:

$$P ::= SKIP \mid STOP \mid a \rightarrow P \mid P \,\Box\, P' \mid P \sqcap P'$$

$$\mid P \,;\, P' \mid P \,[\![A]\!]\, P' \mid P \setminus A \mid \mu\, X.\, f(X)$$

- Some Syntactic Sugar:

  - $P \,||\, P' = P \,[\![\Sigma]\!]\, P'$
  - $P \,|||\, P' = P \,[\![\emptyset]\!]\, P'$
  - $\Box x \in A \rightarrow P(x) = a_1 \rightarrow P\ a_1 \,\Box \ldots \Box\ a_n \rightarrow P\ a_n$ with $A = \{a_1,\ldots,a_n\}$
  - $c!x \rightarrow P(x) = (c,\ a) \rightarrow P\ a$
  - $c?x \in X \rightarrow P(x) = \Box p \in \{(c,a) \mid a \in A\} \rightarrow P(snd\ p)$

Some Requirements on CSP

- The semantics should be compositional,

  (i.e. the semantic denotation should not depend on the context)

- On the other hand, communication inherently depends on contexts:

$$\Box x \in \{a,b,c\} \rightarrow P\ x \ \|\ b \rightarrow Q = b \rightarrow (P\ b \ \|\ Q)$$

$$\sqcap x \in \{a,b,c\} \rightarrow P\ x \ \|\ b \rightarrow Q = (b \rightarrow (P\ b \ \|\ Q)) \sqcap STOP$$

$$\Box x \in \{a,c\} \rightarrow P\ x \ \|\ b \rightarrow Q = STOP$$

$$\sqcap x \in \{a,c\} \rightarrow P\ x \ \|\ b \rightarrow Q = STOP$$

- Observation: Deterministic choice $\Box$ is ready to engage in “b”, and
  {a,b,c} is like a precondition on a read communication with the context “b → Q”

- Observation: Nondeterministic choice $\sqcap$ is like a non-deterministic write !
- Observation: Equational Reasoning on Processes would be a huge
  advantage for abstract reasoning (rather than tinkering on LTSs or, sigh, on automata)

Some Requirements on CSP

- Nondeterminism and potential futures …

$$(a \rightarrow P) \,\Box\, (a \rightarrow Q) = (a \rightarrow P) \sqcap (a \rightarrow Q)$$

$$= a \rightarrow P \sqcap Q$$

Some Requirements on CSP

- Processes are by nature infinite objects …

$$\mu X.\ a \rightarrow X \qquad a^{\infty}$$

but can be finite :

$$\mu X.\ (a \rightarrow (X \sqcap Skip)\ ;\ b \rightarrow Skip) \qquad a^n b^n$$

- handling termination differently like Buechi Automata
via a special symbol ✓ (called tick)

Denotational Semantics

- Recursion and infinite behaviour in the Scott-Stratchey setting

$$(\mu\, X.\ \Box x \in \{a,b\} \to X) = \Box x \in \{a,b\} \to (\mu\, X.\ \Box x \in \{a,b\} \to X)$$

$$\mu\, X.\ f(X) = f\ (\mu\, X.\ f\ X)$$

$$\bot$$

$$\Box x \in \{a,b\} \to \bot$$

$$\Box x \in \{a,b\} \to \Box x \in \{a,b\} \to \bot$$

$$\Box x \in \{a,b\} \to \Box x \in \{a,b\} \to \Box x \in \{a,b\} \to \bot$$

$$\ldots$$

$$\mathrm{LUB}(\lambda\, n.\ ((\lambda\, X.\ \Box x \in \{a,b\} \to X)^n\ \bot))$$

Denotational Semantics
Scott-Strachey Style

- the semantics of a fix-point is understood as a limit on an approximation process
- this implies the necessity of an order $\sqsubseteq$ and a domain D such that:
  - $\forall x \in D.\ \bot \sqsubseteq x$
  - $\forall x \in D.\ x \sqsubseteq x$
  - $\forall x \in D, \forall y \in D, \forall y \in D.\ x \sqsubseteq y \wedge y \sqsubseteq z \longrightarrow x \sqsubseteq z$
  - $\forall x \in D, \forall x \in y \in D.\ x \sqsubseteq y \wedge y \sqsubseteq z \longrightarrow x = y$
  - $\forall f \in \mathbb{N} \Rightarrow D.\ (\forall x,y.\ x \leq y \longrightarrow f\ x \sqsubseteq f\ z) \longrightarrow \mathrm{LUB}\ f \in D$
- … in other words, (D, $\sqsubseteq$) is a *complete partial order* (cpo)
- a function f is cont(inuous), if it reflects the LUB on any chain Y : $\forall x,y.\ x \leq y \longrightarrow Y\ x \sqsubseteq Y\ z$

$$\mathrm{LUB}(\bigcup_{i \in \mathbb{N}}.\ f\ (Y\ i)) = f(\mathrm{LUB}\ Y)$$

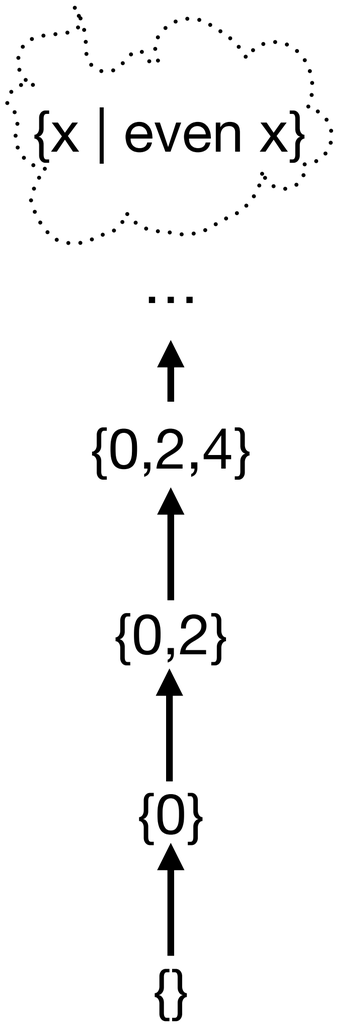
Denotational Semantics Scott-Strachey Style

- Key - Results:
  - a fix point can be defined by $\text{fix } f \equiv \text{LUB}(\lambda\, n.\ (f^n\ \bot))$

    (written $\mu\, X.\ f\ X$)
  - we have an unfolding property: $\text{cont } f \implies \text{fix } f = f(\text{fix } f)$
  - … and the fix point induction principle:

$$\text{cont } f \Longrightarrow \text{adm } P \Longrightarrow P\ \bot \Longrightarrow (\forall x.\ P\ x \Longrightarrow P\ (f\ x)) \Longrightarrow P\ (\mu\, X.\ f\ X)$$

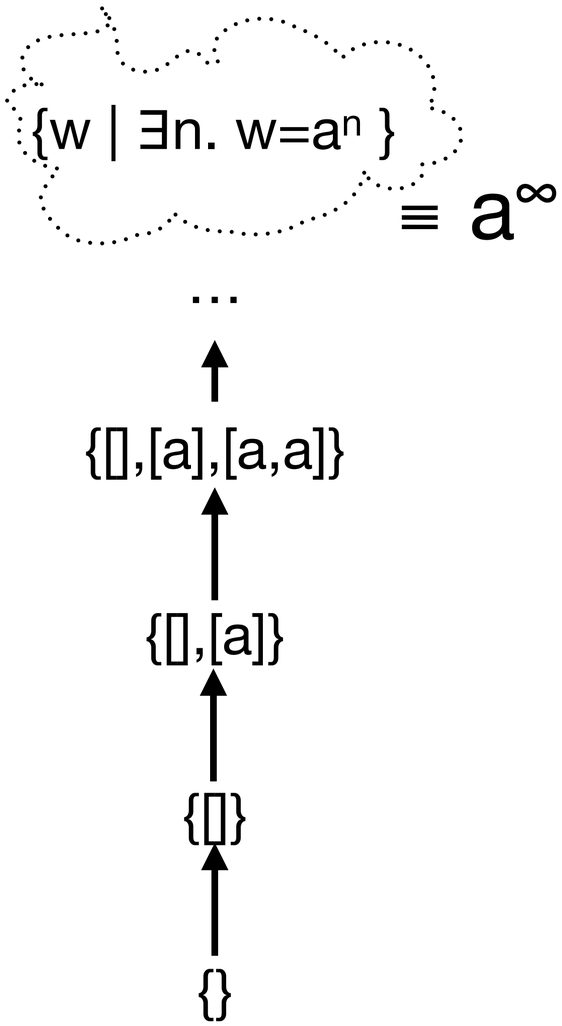
Denotational Semantics Scott-Strachey Style



- Example I: Inductively defined sets
  - $\bot$ is $\{\}$
  - $\sqsubseteq$ is $\subseteq$
  - LUB Y is $\bigcup_{i \in \mathbb{N}}. (Y\ i)$
  - Y n is $(\lambda X.\ \{0\} \cup ((+)2)\ `\ X)^n\ \{\}$

continuous !

Denotational Semantics
Scott-Strachey Style



- Example II: Closer to the topic: Prefix-closed sets of traces
  - $\bot$ is $\{\}$
  - $\sqsubseteq$ is $\subseteq$
  - LUB Y is $\bigcup_{i \in \mathbb{N}.} (Y\ i)$
  - Y n is $(\lambda X.\ \{[]\} \cup ((\#)a) \grave{}\ X)^n\ \{\}$

continuous !

Denotational Semantics for CSP

- How to bring all this together in a domain D and an order ?
  - Non-Determinism and communication
  - Recursion and infinity
  - An algebraic structure on the domain
  - A notion of refinement

- The idea developed by Hoare, Roscoe and Brooks was:
  - … we take prefix-closed set of traces leading to a “state”
  - … and annotate the latter with the set of events a process can not engage in

Denotational Semantics for CSP

- The idea developed by Hoare, Roscoe and Brooks was:
  - … we take prefix-closed traces leading to a non-deterministic “state”
  - … and annotate these with the set of events a process can not engage in, the “refusals”

Recall :

$$(a \rightarrow SKIP) \,\Box\, (a \rightarrow STOP) = (a \rightarrow SKIP) \sqcap (a \rightarrow STOP)$$
$$= a \rightarrow (SKIP \sqcap STOP)$$

So the trace [a] should be annotated with a set of events called *refusals*

Denotational Semantics for CSP

- Now, all CSP Operators were defined as an operation on α *process*, and we have to prove that the domain properties are preserved

- We prove the "algebra" of CSP:

Note that this holds for an arbitrary type 'α definable in HOL

$$
\begin{array}{l}
(\forall\, y.\ c\ y \in S) \Longrightarrow c?x \to P\ x\ [\![S]\!]\ c?x \to Q\ x = c?x \to (P\ x\ [\![S]\!]\ Q\ x) \\
(\forall\, y.\ c\ y \in S) \Longrightarrow inj\ c \Longrightarrow c!a \to P\ [\![S]\!]\ c?x \to Q\ x = c!a \to (P\ [\![S]\!]\ Q\ a) \\
d\ a \notin S \Longrightarrow (\bigwedge y.\ c\ y \in S) \Longrightarrow d!a \to P\ [\![S]\!]\ c?x \to Q\ x = d!a \to (P\ [\![S]\!]\ c?x \to Q\ x) \\
d \in S \Longrightarrow (\bigwedge y.\ c\ y \notin S) \Longrightarrow d \to P\ [\![S]\!]\ c?x \to Q\ x = c?x \to (d \to P\ [\![S]\!]\ Q\ x) \\
d\ a \notin C \Longrightarrow c \in C \Longrightarrow c \to Q\ [\![C]\!]\ d!a \to P = d!a \to (c \to Q\ [\![C]\!]\ P) \\
\forall\, y.\ c\ y \notin B \Longrightarrow\ c?x \to P\ x \setminus B = c?x \to (P\ x \setminus B) \\
\forall\, y.\ c\ y \notin B \Longrightarrow c!a \to P \setminus B = c!a \to (P \setminus B) \\
c\ a \in B \Longrightarrow c!a \to P \setminus B = P \setminus B \qquad\qquad \textit{etc.}
\end{array}
$$

- … and we have a refinement order (more deterministic, more defined):

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

Denotational Semantics for CSP

- Failures and Process P:

$$
\begin{array}{lll}
\textit{datatype} & \alpha\ \textit{event} & = \textit{ev}\ \ \alpha \mid \textit{tick}\ (\langle\checkmark\rangle) \\
\textit{type\_synonym} & \alpha\ \textit{refusal} & = \alpha\ \ \textit{event set} \\
\textit{type\_synonym} & \alpha\ \ \textit{trace} & = \alpha\ \textit{event list} \\
\textit{type\_synonym} & \alpha\ \textit{failure} & = \alpha\ \textit{trace} \times \alpha\ \textit{refusal} \\
\textit{type\_synonym} & \alpha\ \textit{process} & = \ \alpha\ \textit{failure set} \times \alpha\ \textit{trace set}
\end{array}
$$

- Denotational Domain : the $\alpha\ process$ satisfying:

$$
\begin{array}{l}
([] \in \mathcal{T}\ P \wedge (\forall\, s\ X.\ (s, X) \in \mathcal{F}\ P \longrightarrow \textit{front-tickFree}\ s) \wedge \\
(\forall\, s\ t.\ s\ @\ t \in \mathcal{T}\ P \longrightarrow s \in \mathcal{T}\ P) \wedge (\forall\, s\ X\ Y.\ (s, Y) \in \mathcal{F}\ P \wedge X \subseteq Y \longrightarrow (s, X) \in \mathcal{F}\ P) \wedge \\
(\forall\, s\ X\ Y.\ (s, X) \in \mathcal{F}\ P \wedge (\forall\, c.\ c \in Y \longrightarrow s\ @\ [c] \notin \mathcal{T}\ P) \longrightarrow (s, X \cup Y) \in \mathcal{F}\ P) \wedge \\
(\forall\, s\ X.\ s\ @\ [\checkmark] \in \mathcal{T}\ P \longrightarrow (s, X - \{\checkmark\}) \in \mathcal{F}\ P) \wedge \\
(\forall\, s\ t.\ s \in \mathcal{D}\ P \wedge \textit{tickFree}\ s \wedge \textit{front-tickFree}\ t \longrightarrow s\ @\ t \in \mathcal{D}\ P) \wedge \\
(\forall\, s\ X.\ s \in \mathcal{D}\ P \longrightarrow (s, X) \in \mathcal{F}\ P) \wedge (\forall\, s.\ s\ @\ [\checkmark] \in \mathcal{D}\ P \longrightarrow s \in \mathcal{D}\ P))
\end{array}
$$

- With the projections for the traces $\mathcal{T}\ P$,failures $\mathcal{F}\ P$ and divergences $\mathcal{D}\ P$.

Denotational Semantics for CSP

- An intermediate Summary:
  - Non-Determinism and communication ✓
  - An algebraic structure on the domain ✓
  - A notion of refinement ✓
  - Recursion and infinity **ben, non.**

Denotational Semantics for CSP

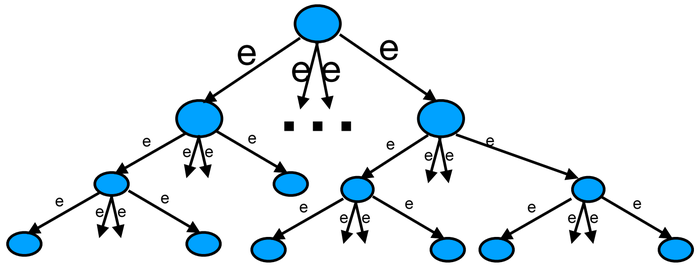
- The Problem :
  - The refinement order $\sqsubseteq_{FD}$ is too liberal to allow non-determinism to be complete …
  - Things like:

$$\mu X.\ (P\ X \ \Box\ X)$$

… can not be continuous, the deterministic choice “needs to know” in advance what events to accept and what to refuse

Denotational Semantics for CSP

- The solution (due to Roscoe and Brookes 86) :
  - a second order, written (here!) "$\sqsubseteq$", orders the elements in the domain as follows:



  - $\sqsubseteq$ relates process approximations "layer-wise"
    (in contrast to $\sqsubseteq_{FD}$)
  - all CSP operators are continuous wrt. to the $\sqsubseteq$ order
    with the exception of hiding P \ S (but hiding is continuous if S is finite)
  - this holds in particular for the combinations $\Box x \in A \rightarrow P\ x$ and $\sqcap x \in A \rightarrow P\ x$

HOL-CSP, CSP$_M$, ProcOmata, HOL-Cyber$_\Phi$

- Goal: Having a logically consisting basis to reason over processes operationally, consistent and symbolic

$$\frac{P \rightsquigarrow_e P' \qquad P' \rightsquigarrow_\tau P''}{P \rightsquigarrow_e P''} \qquad \frac{P \rightsquigarrow_\tau P' \qquad P' \rightsquigarrow_e P''}{P \rightsquigarrow_e P''}$$

$$\frac{P \rightsquigarrow_\checkmark P' \qquad P' \rightsquigarrow_\tau P''}{P \rightsquigarrow_\checkmark P''} \qquad \frac{P \rightsquigarrow_\tau P' \qquad P' \rightsquigarrow_\checkmark P''}{P \rightsquigarrow_\checkmark P''}$$

$$\frac{}{SKIP \rightsquigarrow_\checkmark \Omega\ SKIP} \qquad \frac{cont\ f \qquad P = (\mu\ x.\ f\ x)}{P \rightsquigarrow_\tau f\ P}$$

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

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

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

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

$$\frac{P \rightsquigarrow_\tau P'}{P \setminus B \rightsquigarrow_\tau P' \setminus B} \qquad \frac{P \rightsquigarrow_\checkmark P'}{P \setminus B \rightsquigarrow_\checkmark \Omega\ SKIP}$$

$$\frac{e \notin B \qquad P \rightsquigarrow_e P'}{P \setminus B \rightsquigarrow_e P' \setminus B} \qquad \frac{e \in B \qquad P \rightsquigarrow_e P'}{P \setminus B \rightsquigarrow_\tau P' \setminus B}$$

...

...

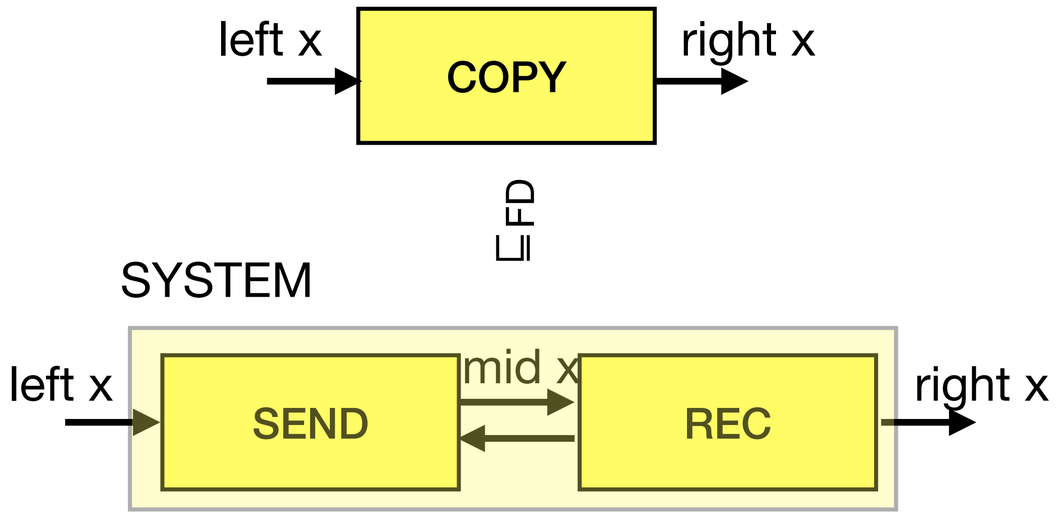
[ITP 2024]

HOL-CSP

- The theory HOL-CSP comprises
  - definitions of all operators
  - continuity proofs
  - proofs for algebraic laws
    (from ACI to synchronization laws)
  - How to model and reason with it?

Example for CSP Modeling and Reasoning: CopyBuffer

- A classic example:




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

definition COPY :: "'a channel process"
lemma impl_refines_spec : "COPY ⊑FD SYSTEM"
  unfolding SYSTEM_def COPY_def
  apply(rule fix_ind, auto intro: le_FD_adm simp:  cont_fun monofunI)
  apply(subst SEND_rec, subst REC_rec)
  apply (simp add: Sync_rules Hiding_rules mono_read_FD mono_write_FD)
  done
  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›
```

Denotational Semantics Scott-Strachey Style

- The sweet spot of domain theory:
  - cpo's for products and function spaces can be constructed automatically:

    $(D, \sqsubseteq_D),\ (E, \sqsubseteq_E) \implies (D \times E,\ \sqsubseteq_{D \times E})$

    $(D, \sqsubseteq_D),\ (E, \sqsubseteq_E) \implies (D \Rightarrow E,\ \sqsubseteq_{D \Rightarrow E})$
  - … actually : $(E, \sqsubseteq_E) \implies (D \Rightarrow E, \sqsubseteq_{D \Rightarrow E})$ suffices.
  - consequence: HOL-CSP processes can be parameterised by a state that they can carry around and modify. We get a notion of state for free! Example:

$$NG = \mu\ X.\ (\lambda\ A.\ (\sqcap a \in A \rightarrow (X\ (A - \{a\}))))$$

$$NG\ A = \sqcap a \in A \rightarrow (RNG\ (A - \{a\}))$$

$$NG\ \mathbb{N}$$

Some Examples in HOL-CSP

- Processes are infinite objects:

ones ≡ (μ X. 1 → X) zeros ≡ (μ X. 0 → X) oneszeros ≡ (μ X. □x∈{0,1} → X)

Property: ones ||| zeros = oneszeros (where P|||Q = P ⟦ {} ⟧ Q)

Proof: by anti-sym of = and fix point induction

- Fixpoints over process functions yield parameterizable processes:

digits ≡ μ X. (λx. ⌊x⌋ → X ((frac x) * 10)) implies: digits x = ⌊x⌋ → digits (frac x * 10)

Fact : digits π = 3 → 1 → 4 → 1 → 5 → 9 → …

- No problems with termination issues

Collatz n = (□x ∈ ({0, 1} ∩ {n}) → Skip)
□ (□x ∈ ((Even - {0}) ∩ {n}) → Collatz (x div 2))
□ (□x ∈ ((Odd - {1}) ∩ {n}) → Collatz (3 * x + 1))

KEY A = ⊓a ∈ A → KEY (A - {a})

HOL-CSP$_M$

- The CSP$_M$ is a language developed for FDR, a model-checker for CSP. It offers a number of “architectural operators”, i.e. generalisations of synchronisation P ⟦S⟧ P’, and sequencing P ⟦S⟧ .

  ⟦S⟧ x ∈ I. P(x), ||| x ∈ I. P(x), SEQ x ∈ I. P(x), …

  beyond the already known operators □x∈A → P(x) and ⊓x∈A → P (x)
  and the derived notations for “guarded read” and “non-deterministic write”

- These were handled as macros in FDR, but HOL-CSP$_M$ defined them as first-class citizens and develops a theory for it.

- HOL-CSP$_M$ treats them by proper algebraic rules.

Part IV:

Extensions of Theory of Concurrency

HOL-CSP$_M$

- Example for architectural composition: Dijkstra's Dining Philosophers …

  Let N be an uninterpreted constant:

```
datatype dining_event = picks ‹int × int› | putsdown ‹int × int››

definition RPHIL  ::
  ‹int ⇒ dining_event process› where
  ‹RPHIL i ≡ μ X. picks (i, i) → picks (i, ((i - 1) mod N)) →
                  putsdown( i, ((i - 1) mod N)) → putsdown (i, i) → X›

definition LPHIL0 :: ‹dining_event process› where
  ‹LPHIL0  ≡  μ X. picks (0, N-1) → picks (0, 0) →
                   putsdown (0, 0) → putsdown (0, N-1) → X›

definition FORK :: ‹int ⇒ dining_event process›
  where
  ‹FORK i ≡ μ X.  (picks(i,i) → putsdown(i,i) → X) □
             (picks (((i + 1) mod N), i) →
             putsdown ((i + 1) mod N, i) → X)››

PHILS ≡ LPHIL0 ||| (|||i ∈# mset [1..N-1]. RPHIL i)

FORKS ≡ ||| i ∈# mset [0..N-1]. FORK i

DINING ≡ FORKS || PHILS
```

Proc-Omata

- We had been intrigued to find a solution for architectures ranging over N. This resulted in a theory of Proc-Omata
- Proc-Omata are functional automata inside a process, so a process with an internal state + a transition function:

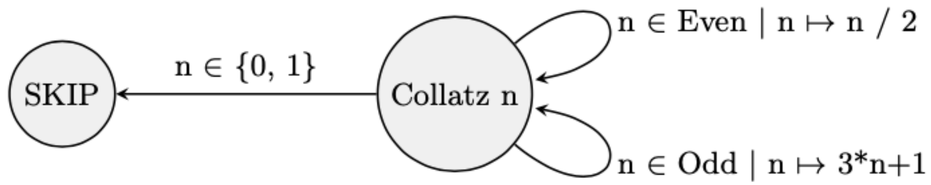


Fig. 1: The Collatz Function seen as Process

$$\begin{aligned}\text{Collatz} \equiv \mu\ \text{X}.\ (\lambda \text{n}.\quad & (\Box \text{x} \in \{0, 1\} \cap \{\text{n}\} \rightarrow \text{SKIP}) \\ & \Box\ (\Box \text{x} \in (\text{Even} - \{0\}) \cap \{\text{n}\} \rightarrow \text{X}\ (\text{x div } 2)) \\ & \Box\ (\Box \text{x} \in (\text{Odd} - \{1\}) \cap \{\text{n}\} \rightarrow \text{X}\ (3 * \text{x} + 1)))\end{aligned}$$

Proc-Omata

- Main theorem: Compactification of an unbounded series of Proc-Omata Processes.

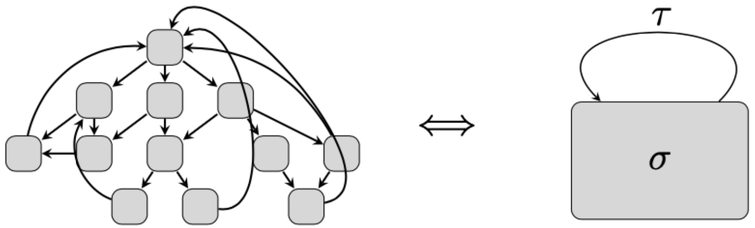


Fig. 2: Conversion of an LTS

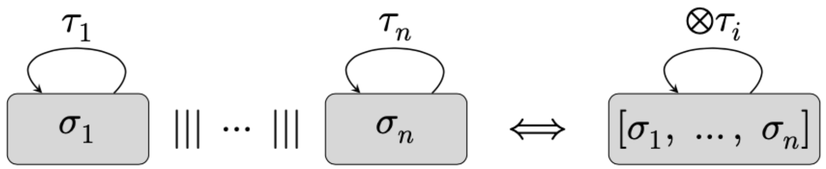


Fig. 3: Compactification

Over the compactified Proc-Omata, we can prove via an invariant over its state that, for example, DINING is deadlock free for arbitrary N.

Operational Semantics of CSP

- Goal: Having a logically consisting basis to reason over processes operationally, consistent with HOL-CSP and as a symbolic transition system.

$$\frac{P \rightsquigarrow_e P' \qquad P' \rightsquigarrow_\tau P''}{P \rightsquigarrow_e P''} \qquad \frac{P \rightsquigarrow_\tau P' \qquad P' \rightsquigarrow_e P''}{P \rightsquigarrow_e P''}$$

$$\frac{P \rightsquigarrow_\checkmark P' \qquad P' \rightsquigarrow_\tau P''}{P \rightsquigarrow_\checkmark P''} \qquad \frac{P \rightsquigarrow_\tau P' \qquad P' \rightsquigarrow_\checkmark P''}{P \rightsquigarrow_\checkmark P''}$$

$$\frac{}{SKIP \rightsquigarrow_\checkmark \Omega\ SKIP} \qquad \frac{cont\ f \qquad P = (\mu\ x.\ f\ x)}{P \rightsquigarrow_\tau f\ P}$$

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

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

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

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

$$\frac{P \rightsquigarrow_\tau P'}{P \setminus B \rightsquigarrow_\tau P' \setminus B} \qquad \frac{P \rightsquigarrow_\checkmark P'}{P \setminus B \rightsquigarrow_\checkmark \Omega\ SKIP}$$

$$\frac{e \notin B \qquad P \rightsquigarrow_e P'}{P \setminus B \rightsquigarrow_e P' \setminus B} \qquad \frac{e \in B \qquad P \rightsquigarrow_e P'}{P \setminus B \rightsquigarrow_\tau P' \setminus B}$$

[ITP 2024]

HOL-CSP and FDR

- Goal: Combining FDR4 and HOL-CSP (OpSem) in Isabelle, allowing to certify the checks of FDR by Proofs in CSP


```
val system_lts =
  {event_conversion_table =
```

Table II: Dining Philosopers

model-size	iterations	events	transitions	elapsed time	cpu time	avg cpu time
N=3	7	20	46	0.5s	2.17s	47 ms
N=4	11	32	193	1.85s	12.08s	52 ms
N=5	14	50	876	9.26s	67.86s	77 ms
N=6	17	72	3274	47.78s	363.67s	110 ms
N=7	20	96	13670	248.44s	1899s	138 ms
N=8	23	124	51944	1336s	9694s	186 ms
N=9	26	160	194002	30142s	67157s	346 ms

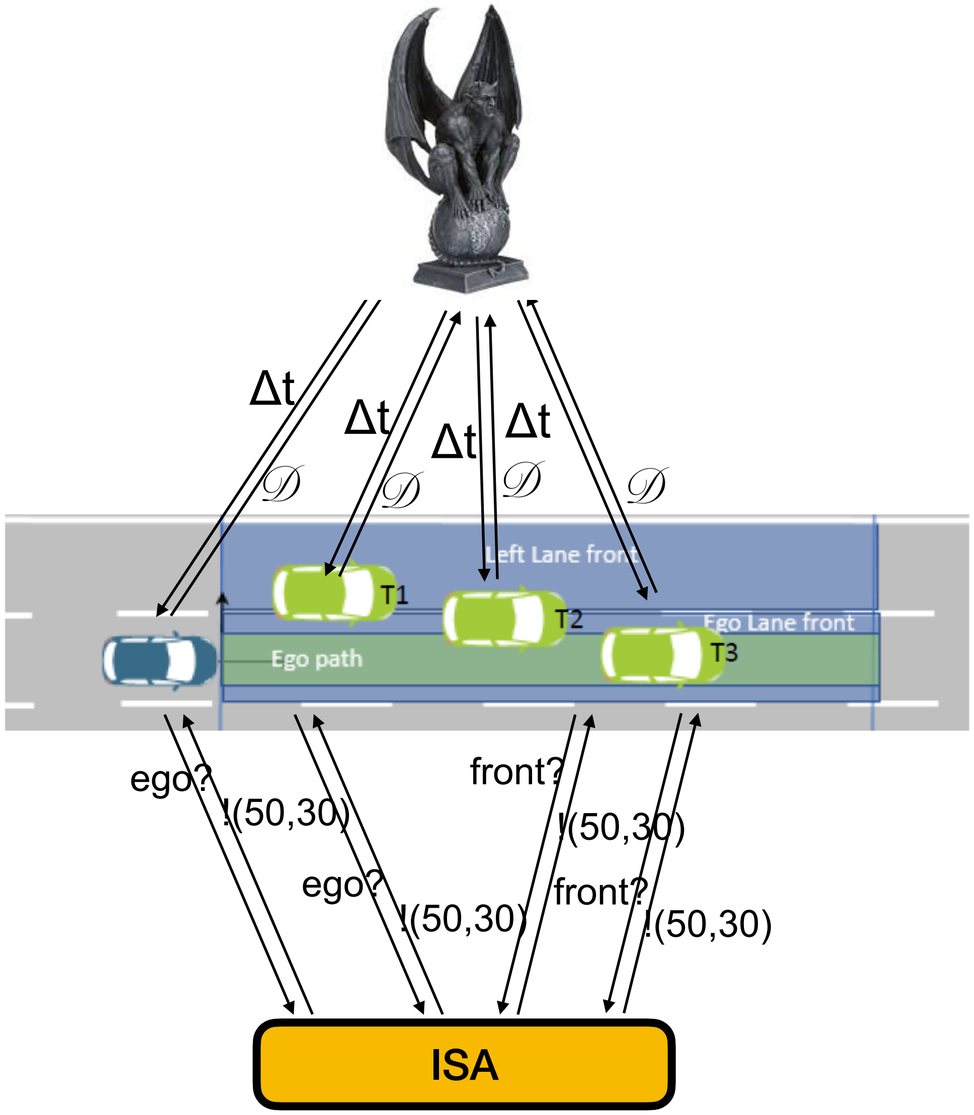
```
},
7370),
9729),
```

[CPP 2025 submitted]

HOL-CSP and HOL-Cyber$_\varphi$

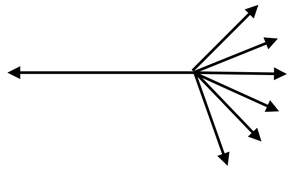
- A modeling approach for autonomous Agents in HOL-CSP: HOL-Cyber$_\varphi$

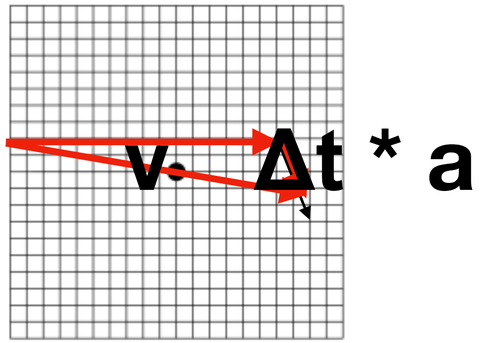
  - concurrent interaction of actors and environments
  - dense time ($\mathbb{R}_{\geq 0}$)
  - continuous environment with multi-dimensional observables underlying the laws of (Newtonian) physics
  - the discretising view of continuous observables (sampling) in systems
  - refinement between modeling notions and of models to executable code
  - formal analysis by test and proof


Δt
Δt
Δt
Δt
𝒟
𝒟
𝒟
𝒟
Left Lane front
T1
T2
Ego Lane front
T3
Ego path
ego?
!(50,30)
front?
!(50,30)
ego?
!(50,30)
front?
!(50,30)
ISA

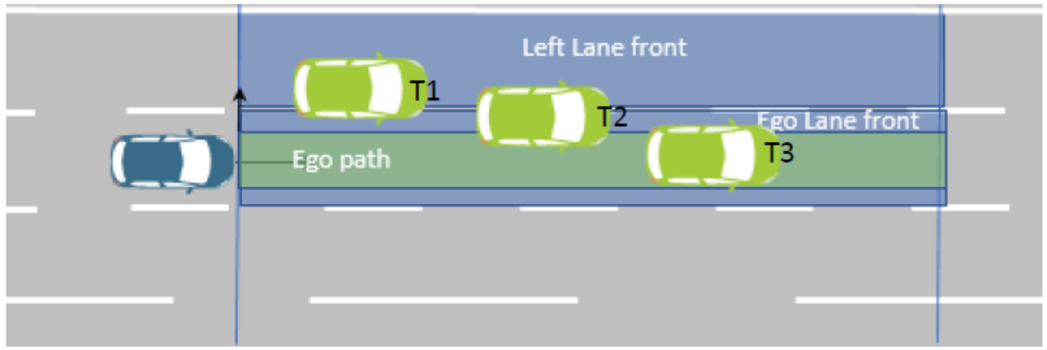
Autonomous Vehicles in HOL-CSP

- Vector-Spaces
  - for positions
  - accelerations
  - speeds



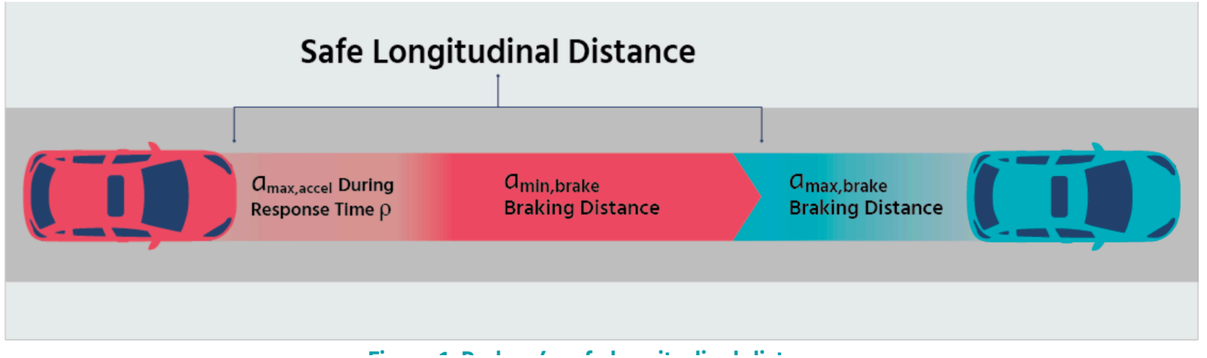


- … in Isabelle/HOL:
  - discrete
  - real/math
  - multi-dimensional
- we can therefore link more abstract to more computational models



The RSS Scenarios

- Two Cars, “current” set of assumptions
- Safety Property:
  - reaction time (>> Δt): ρ
  - minimal distance to be respected : $d_{\min}$
  - speeds for “rear” and “front” cars: v_r, v_f



$$d_{\min} = \left[v_r\,\rho + \frac{1}{2} a_{\max,\text{accel}}\,\rho^2 + \frac{(v_r + \rho\, a_{\max,\text{accel}})^2}{2a_{\min,\text{brake}}} - \frac{v_f^2}{2a_{\max,\text{brake}}} \right]_+$$

HOL-CSP and Programming Language Theory

- One might wonder what HOL-CSP theory has to do with Programming Language Theory, so, e.g. deductive verification
- We developed a PL-Front-End for HOL-CSP, called “IMP_Concur”.

  - **… and a semantics on a post-card:**

```
datatype com = SKIP
fun Sem₀ :: "com ⇒ (σ ⇒ (sema_evs+glo_vars_ev) process) ⇒ σ ⇒ (sema_evs+glo_vars_ev) process"
  where "Sem₀ SKIP C                    = C"
       |"Sem₀ (x := E) C                = (λ σ. C (σ(x := E σ)))"
       |"Sem₀ (P ; Q) C                 = Sem₀ P (Sem₀ Q C)"
       |"Sem₀ (IF E THEN C1 ELSE C2) C = (λ σ. if E σ then Sem₀ C1 C σ else Sem₀ C2 C σ)"
       |"Sem₀ (WHILE E DO B) C          = (μ X. (λ σ. if E σ then Sem₀ B X σ else C σ))"
       |"Sem₀ (call F) C                = (λ σ. C (F σ))"
       |"Sem₀ (com.lock n) C            = (λ σ. Inl(sema_evs.lock n) → C σ)"
       |"Sem₀ (com.unlock n) C          = (λ σ. Inl(sema_evs.unlock n) → C σ)"
       |"Sem₀ (STORE E TO Xglo) C       = (λ σ. Inr(glo_vars_ev.update Xglo (E σ)) → C σ)"
       |"Sem₀ (LOAD Xloc FROM  Xglo) C   = (λ σ. (λid. Inr(glo_vars_ev.read Xglo id))?x → C(σ(Xloc:=x)))"
```

```
definition Thread1 where
λσ. 1 + 2);
'd'' > 3) THEN lock 4 ELSE lock 3;
σ ''e'' > 3) DO lock 4 ;
σ ''d'' + 1) TO ''b'';
FROM  ''b'';
```

Part V:

A Formal Theory of Concurrency:

Conclusions so far.

Formalism and Theory

- Practical objections:

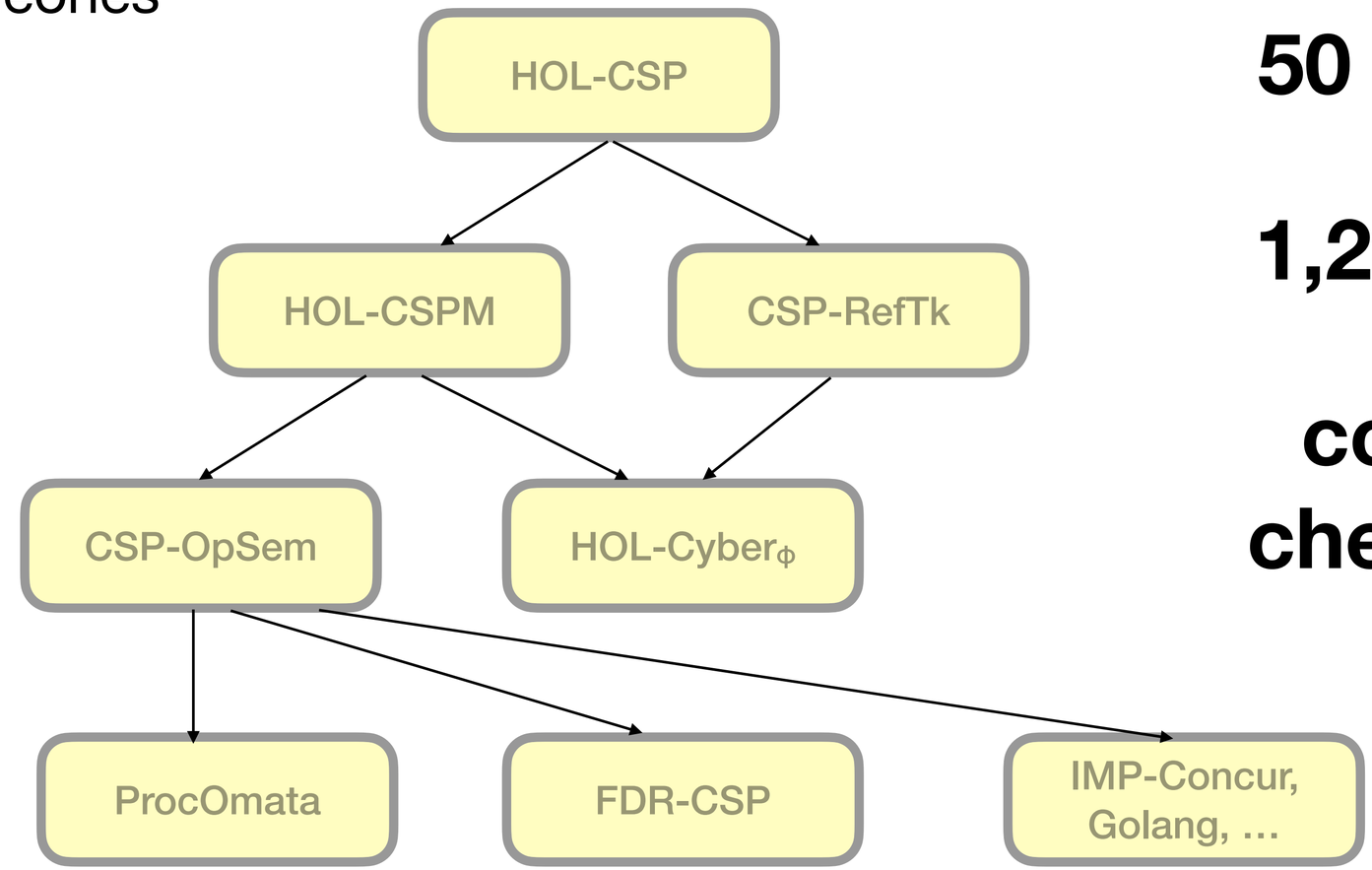
  — a formalisation of a true mathematical proof is impractical
  (a certain proof would take 10^9 steps in the “Bourbaki-Method”)

  — proof languages are unreadable by readers not familiar with a particular ITP

  — it takes too long to make a leading edge mathematical proof
  (Peter Schulze 1.0, before saying the contrary after the
  Liquid Tensor Experiment)

HOL-CSP, CSP_M, ProcOmata, HOL-$Cyber_\Phi$

- Architecture of the CSP Theories



50 k Loc

1,2 kthm

continuous check on AFP

www.isa-afp.org/entries/HOL-CSP.html

Formalism and Theory

- On the downside, there is still so much to do …
- On the upside, there are quite important advantages
  of a Formal Development of Concurrency:
    - Proofs become an machine-readable artefact like code, which democratizes proofs to a certain extent,
        - which can be stored, run, versioned …
        - People can do unforeseen optimisations without having understood the full Monty
        - Proof Architecture extensions profit from a clear
          consistency criterion and an impact-analysis of changes
        - Proof-architectures can be shared like OTS-Components
          (we reused things like: a model for json-files,
          but also Jacobian matrices, or a theory on LTS's …)
        - Developing libraries forces to reconsider generalisation and reusability
    - Documentation and reusability becomes important (the ITP game is therefore quite different from ATPs)

Thank you

HOL-CSP and CPS

- concurrent interaction of actors and environments
- dense time ($\mathbb{R}_{\geq 0}$)
- continuous environment multi-dimensional observables underlying the laws of (Newtonian) physics
- the discretising view of continuous observables in systems (sampling)
- refinement between modeling notions and of models to executable code
- formal analysis by (refinement) proof (and in future: by test)

A Bluffers Guide to Isabelle Proofs

- The idea of a "Formal Proof" (in contrast to a paper-and-pencil proof) has its roots in the 20th century
- Result of a Crisis: Too many false "proofs", therefore quest for logical foundations of Math, Physics and Engineering
- … semi-automated procedures construct most of formal proofs.
- Since we use as method to analyse RSS formal proof … we'd like to elaborate this a bit.
- Details: Online course on interactive theorem proving: (www.lri.fr/~wolff/teach-material/2020-2021/M2-CSMR/index.html)

Formalism and Theory

- A personal statement :

I am a Formalist

— I believe that only formal proofs based on formalised definitions can give certainty and some form of scientific objectivity

— above all, a proof SHOULD be reproducible like a physical experiment - independent from human subjectivity and social processes

— Intuition is a human necessity, but treacherous.

— David Hilbert: "One must be able to say at all times --instead of points, straight lines, and planes-- tables, chairs, and beer mugs."

- This is not just a spleen, it is a epistemological position.

Formalism and Theory

- I acknowledge that a integrated formal theory of whatever faces substantial problems:
  - Capturing the SIZE of mathematical knowledge (say, a 3rd year Math-student in Analysis, Topology, Probability, …)
  - Capturing the SIZE of larger proof architectures (FLT, 4Colors, Feit-Thomson…)
  - Assuring logical and technical consistency and its maintenance
  - Assuring readability and accessibility of formal theories within a community of researchers and just: users
  - … over a long period of time (Isabelle/Coq : since 1987)

Formalism and Theory

- On the upside, there are quite important advantages:
    - Proofs become an machine-readable artefact like code, which democratises proofs to a certain extent,
        - which can be stored, run, versioned …
        - People can do unforeseen optimisations without having understood the full Monty
        - Proof Architecture extensions profit from a clear consistency criterion and an impact-analysis of changes
        - Proof-architectures can be shared like OTS-Components (we reused things like: a model for json-files, but also Jacobian matrices, or a theory on LTS's …)
        - Developing libraries forces to reconsider generalisation and reusability
    - Documentation and reusability becomes important (the ITP game is therefore quite different from the ATP game)

Formalism and Theory

- The Philosophical Counter-Position: Intuitionism
  — (Brower): the truth of a mathematical statement is a subjective claim: a mathematical statement corresponds to a mental construction, and a mathematician can assert the truth of a statement only by verifying the validity of that construction by intuition.

- Supporters: Einstein
  — „Der intuitive Geist ist ein heiliges Geschenk und der rationale Verstand ein treuer Diener."

  But:
  — „Seit sich die Mathematiker über meine Relativitätstheorie gebeugt haben, verstehe ich sie selber nicht mehr …"

Formalism and Theory

- More vulgar variants of the intuitionistic position:

  — it is empirically evident that a mathematical proof just holds independent of a prover
  — it is the mathematical community that decides anyway … the social process among the “savants” is sufficient to assure truth
  — formalisation is just an “implementation” of a proof, a “mechanisation” (english language)
  — did you prove your prover ?
  — Goedel proved that you can’t prove everything

- Practical objections:
  — a formalisation of a true mathematical proof is impractical (a certain proof would take 10^9 steps in the “Bourbaki-Method”)
  — proof languages are unreadable by readers not familiar with a particular ITP
  — it takes too long to make a leading edge mathematical proof (Peter Schulze 1.0)

Formalism and Theory

- I acknowledge that a integrated formal theory of whatever faces substantial problems:
  - Capturing the SIZE of mathematical knowledge (say, a 3rd year Math-student in Analysis, Topology, Probability, …)
  - Capturing the SIZE of larger proof architectures (FLT, 4Colors, Feit-Thomson…)
  - Assuring logical and technical consistency and its maintenance
  - Assuring readability and accessibility of formal theories within a community of researchers and just: users
  - … over a long period of time (Isabelle/Coq : since 1987)