The HOL-CSP Refinement Toolkit

Safouan Taha   Burkhart Wolff   Lina Ye

March 19, 2025

Abstract

Recently, a modern version of Roscoes and Brookes [3] Failure-Divergence Semantics for CSP has been formalized in Isabelle [10].

We use this formal development called HOL-CSP2.0 to analyse a family of refinement notions, comprising classic and new ones. This analysis enables to derive a number of properties that allow to deepen the understanding of these notions, in particular with respect to specification decomposition principles for the case of infinite sets of events. The established relations between the refinement relations help to clarify some obscure points in the CSP literature, but also provide a weapon for shorter refinement proofs. Furthermore, we provide a framework for state-normalisation allowing to formally reason on parameterised process architectures.

As a result, we have a modern environment for formal proofs of concurrent systems that allow for the combination of general infinite processes with locally finite ones in a logically safe way. We demonstrate these verification-techniques for classical, generalised examples: The CopyBuffer for arbitrary data and the Dijkstra's Dining Philosopher Problem of arbitrary size.

If you consider to cite this work, please refer to [11].

Contents

Chapter 1

Context

1.1 Introduction

Communicating Sequential Processes CSP is a language to specify and verify patterns of interaction of concurrent systems. Together with CCS and LOTOS, it belongs to the family of *process algebras*. CSP's rich theory comprises denotational, operational and algebraic semantic facets and has influenced programming languages such as Limbo, Crystal, Clojure and most notably Golang [5]. CSP has been applied in industry as a tool for specifying and verifying the concurrent aspects of hardware systems, such as the T9000 transputer [1].

The theory of CSP, in particular the denotational Failure/Divergence Denotational Semantics, has been initially proposed in the book by Tony Hoare [6], but evolved substantially since [2, 3, 8].

Verification of CSP properties has been centered around the notion of *process refinement orderings*, most notably -$\sqsubseteq_{FD}$- and -$\sqsubseteq$-. The latter turns the denotational domain of CSP into a Scott cpo [9], which yields semantics for the fixed point operator $\mu x.\ f(x)$ provided that f is continuous with respect to -$\sqsubseteq$-. Since it is possible to express deadlock-freeness and livelock-freeness as a refinement problem, the verification of properties has been reduced traditionally to a model-checking problem for a finite set of events A.

We are interested in verification techniques for arbitrary event sets A or arbitrarily parameterized processes. Such processes can be used to model dense-timed processes, processes with dynamic thread creation, and processes with unbounded thread-local variables and buffers. Events may even be higher-order objects such as functions or again processes, paving the way for the modeling of re-programmable compute servers or dynamic distributed computing architectures. However, this adds substantial complexity to the process theory: when it comes to study the interplay of different denotational models, refinement-orderings, and side-conditions for continuity, paper-and-pencil proofs easily reach their limits of precision.

Several attempts have been undertaken to develop the formal theory of CSP in an interactive proof system, mostly in Isabelle/HOL [4, 12, 7]. This work is based on the most recent instance in this line, HOL-CSP 2.0, which has been published as AFP submission [10] and whose development is hosted at https://gitlri.lri.fr/burkhart.wolff/hol-csp2.0.

The present AFP Module is an add-on on this work and develops some support for

1. example of induction schemes (mutual fixed-point Induction, K-induction),
2. a theory of explicit state normalisation which allows for proofs over certain communicating networks of arbitrary size.

1.2 The Global Architecture of CSP_RefTk

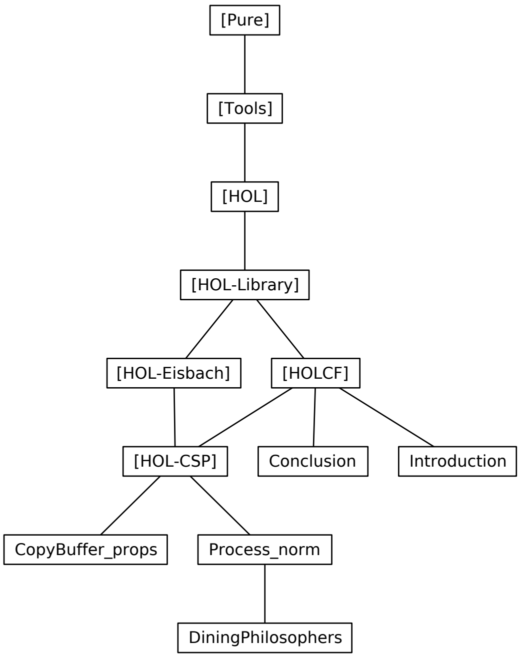

Figure 1.1: The overall architecture: HOLCF, HOL-CSP, and CSP_RefTk

The global architecture of CSP_RefTk is shown in Figure 1.1. The entire package resides on:

1. `HOL-Eisbach` from the Isabelle/HOL distribution,
2. `HOLCF` from the Isabelle/HOL distribution, and
3. `HOL-CSP` 2.0 from the Isabelle Archive of Formal Proofs.

Chapter 2

Normalisation of Deterministic CSP Processes

theory *Process-norm*

imports *HOL−CSP.CSP*

begin

2.1 Deterministic normal-forms with explicit state

abbreviation *P-dnorm* $\tau\ \upsilon \equiv (\mu\ X.\ (\lambda\ s.\ \square\ e \in (\tau\ s) \rightarrow X\ (\upsilon\ s\ e)))$

notation *P-dnorm* ($P_{norm}[\![$-,-$]\!]$ *60*)

lemma *dnorm-cont*[*simp*]:
  fixes τ::$'\sigma$::*type* $\Rightarrow$ $'event$::*type set* **and** υ::$'\sigma \Rightarrow 'event \Rightarrow '\sigma$
  shows *cont* $(\lambda X.\ (\lambda s.\ \square\ e \in (\tau\ s) \rightarrow X\ (\upsilon\ s\ e)))$ (**is** *cont ?f*)
⟨*proof*⟩

2.2 Interleaving product lemma

lemma *dnorm-inter*:
  fixes τ_1 ::$'\sigma_1$::*type* $\Rightarrow$ $'event$::*type set* **and** τ_2::$'\sigma_2$::*type* $\Rightarrow$ $'event$ *set*
    and υ_1 ::$'\sigma_1 \Rightarrow 'event \Rightarrow '\sigma_1$ **and** υ_2::$'\sigma_2 \Rightarrow 'event \Rightarrow '\sigma_2$
  defines *P*: $P \equiv P_{norm}[\![\tau_1,\upsilon_1]\!]$ (**is** $P \equiv fix\cdot(\Lambda\ X.\ ?P\ X)$)
  defines *Q*: $Q \equiv P_{norm}[\![\tau_2,\upsilon_2]\!]$ (**is** $Q \equiv fix\cdot(\Lambda\ X.\ ?Q\ X)$)

assumes *indep*: ‹$\forall\ s_1\ s_2.\ \tau_1\ s_1 \cap \tau_2\ s_2 = \{\}$›

defines *Tr*: $\tau \equiv (\lambda(s_1,s_2).\ \tau_1\ s_1 \cup \tau_2\ s_2)$
defines *Up*: $\upsilon \equiv (\lambda(s_1,s_2)\ e.\ \mathit{if}\ e \in \tau_1\ s_1\ \mathit{then}\ (\upsilon_1\ s_1\ e,s_2)$
$\mathit{else\ if}\ e \in \tau_2\ s_2\ \mathit{then}\ (s_1,\ \upsilon_2\ s_2\ e)\ \mathit{else}\ (s_1,s_2))$
defines *S*: $S \equiv P_{norm}[\![\tau,\upsilon]\!]$ (**is** $S \equiv fix\cdot(\Lambda\ X.\ ?S\ X)$)

shows $(P\ s_1\ |||\ Q\ s_2) = S\ (s_1{,}s_2)$

⟨*proof*⟩

2.3 Synchronous product lemma

lemma *dnorm-par*:
  fixes τ_1 ::'σ_1::*type* ⇒ '*event*::*type set* **and** τ_2::'σ_2::*type* ⇒ '*event set*
    and υ_1 ::'σ_1 ⇒ '*event* ⇒ 'σ_1 **and** υ_2::'σ_2 ⇒ '*event* ⇒ 'σ_2
  defines *P*: $P \equiv P_{norm}[\![\tau_1{,}\upsilon_1]\!]$ (**is** $P \equiv$ *fix*·(Λ *X*. *?P X*))
  defines *Q*: $Q \equiv P_{norm}[\![\tau_2{,}\upsilon_2]\!]$ (**is** $Q \equiv$ *fix*·(Λ *X*. *?Q X*))

defines *Tr*: $\tau \equiv (\lambda(s_1{,}s_2).\ \ \tau_1\ s_1 \cap \tau_2\ s_2)$
defines *Up*: $\upsilon \equiv (\lambda(s_1{,}s_2)\ e.\ (\upsilon_1\ s_1\ e,\ \upsilon_2\ s_2\ e))$
defines *S*: $S \equiv P_{norm}[\![\tau{,}\upsilon]\!]$ (**is** $S \equiv$ *fix*·(Λ *X*. *?S X*))

shows $(P\ s_1\ ||\ Q\ s_2) = S\ (s_1{,}s_2)$

⟨*proof*⟩

2.4 Consequences

inductive-set $\mathfrak{R}$ **for** τ ::'σ::*type* ⇒ '*event*::*type set*
  and υ ::'σ ⇒ '*event* ⇒ 'σ
  and σ_0 ::'σ
where *rbase*: $\sigma_0 \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0$
| *rstep*: $s \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0 \Longrightarrow e \in \tau\ s\ \Longrightarrow \upsilon\ s\ e \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0$

— Deadlock freeness
lemma *deadlock-free-dnorm-* :
  fixes τ ::'σ::*type* ⇒ '*event*::*type set*
    and υ ::'σ ⇒ '*event* ⇒ 'σ
    and σ_0 ::'σ
  assumes *non-reachable-sink*: $\forall\, s \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0.\ \tau\ s \neq \{\}$
  defines *P*: $P \equiv P_{norm}[\![\tau{,}\upsilon]\!]$ (**is** $P \equiv$ *fix*·(Λ *X*. *?P X*))
  shows $s \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0 \Longrightarrow$ *deadlock-free* (*P s*)
⟨*proof*⟩

lemmas *deadlock-free-dnorm* = *deadlock-free-dnorm-*[*rotated*, *OF rbase*, *rule-format*]

end

Chapter 3

Examples

3.1 CopyBuffer Refinement over an infinite alphabet

theory *CopyBuffer-props*
imports $HOL-CSP.CopyBuffer$ $HOL-CSP.CSP$
begin

3.1.1 The Copy-Buffer vs. reference processes

thm *DF-COPY*

3.1.2 ... and abstract consequences

corollary *df-COPY*: *deadlock-free COPY*
and *lf-COPY*: *lifelock-free COPY*
⟨*proof*⟩

corollary df_{SKIPS}*-COPY*: $deadlock\text{-}free_{SKIPS}$ *COPY*
and lf_{SKIPS}*-COPY*: $lifelock\text{-}free_{SKIPS}$ *COPY*
and *nt-COPY*: *non-terminating COPY*
⟨*proof*⟩

lemma *DF-SYSTEM*: *DF UNIV* $\sqsubseteq_{FD}$ *SYSTEM*
⟨*proof*⟩

corollary *df-SYSTEM*: *deadlock-free SYSTEM*
and *lf-SYSTEM*: *lifelock-free SYSTEM*
⟨*proof*⟩

corollary df_{SKIPS}*-SYSTEM*: $deadlock\text{-}free_{SKIPS}$ *SYSTEM*
and lf_{SKIPS}*-SYSTEM*: $lifelock\text{-}free_{SKIPS}$ *SYSTEM*
and *nt-SYSTEM*: *non-terminating SYSTEM*
⟨*proof*⟩

end

3.2 Generalized Dining Philosophers

theory *DiningPhilosophers*
imports *Process-norm*
begin

3.2.1 Preliminary lemmas for proof automation

lemma *Suc-mod*: $n > 1 \Longrightarrow i \neq Suc\ i\ mod\ n$
⟨*proof*⟩

lemmas *suc-mods* = *Suc-mod Suc-mod*[*symmetric*]

lemma *l-suc*: $n > 1 \Longrightarrow \neg\ n \leq Suc\ 0$
⟨*proof*⟩

lemma *minus-suc*: $n > 0 \Longrightarrow n - Suc\ 0 \neq n$
⟨*proof*⟩

lemma *numeral-4-eq-4*: $4 = Suc\ (Suc\ (Suc\ (Suc\ 0)))$
⟨*proof*⟩

lemma *numeral-5-eq-5*: $5 = Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))$
⟨*proof*⟩

3.2.2 The dining processes definition

locale *DiningPhilosophers* =

fixes *N*::*nat*
assumes *N-g1*[*simp*] : $N > 1$

begin

datatype *dining-event* = *picks* (*phil*:*nat*) (*fork*:*nat*)
| *putsdown* (*phil*:*nat*) (*fork*:*nat*)

definition *RPHIL*:: *nat* ⇒ *dining-event process*
where *RPHIL* $i = (\mu\ X.\ (picks\ i\ i \rightarrow (picks\ i\ (i-1) \rightarrow (putsdown\ i\ (i-1) \rightarrow (putsdown\ i\ i \rightarrow X)))))$

definition *LPHIL0*:: *dining-event process*
where *LPHIL0* $= (\mu\ X.\ (picks\ 0\ (N-1) \rightarrow (picks\ 0\ 0 \rightarrow (putsdown\ 0\ 0 \rightarrow (putsdown\ 0\ (N-1) \rightarrow X)))))$

definition *FORK* :: *nat* ⇒ *dining-event process*
where *FORK* $i = (\mu\ X.\ \ (picks\ i\ i \rightarrow (putsdown\ i\ i \rightarrow X))$

□ (*picks* ((*i*+*1*) *mod N*) *i* → (*putsdown* ((*i*+*1*) *mod N*) *i* → *X*)))

abbreviation *foldPHILs n* ≡ *fold* (λ *i P*. *P* ||| *RPHIL i*) [*1*..< *n*] (*LPHIL0*)
abbreviation *foldFORKs n* ≡ *fold* (λ *i P*. *P* ||| *FORK i*) [*1*..< *n*] (*FORK 0*)

abbreviation *PHILs* ≡ *foldPHILs N*
abbreviation *FORKs* ≡ *foldFORKs N*

corollary *N* = *3* ⟹ *PHILs* = (*LPHIL0* ||| *RPHIL 1* ||| *RPHIL 2*)
⟨*proof*⟩

definition *DINING* :: *dining-event process*
where *DINING* = (*FORKs* || *PHILs*)

Unfolding rules

lemma *RPHIL-rec*:
RPHIL i = (*picks i i* → (*picks i* (*i*−*1*) → (*putsdown i* (*i*−*1*) → (*putsdown i i* → *RPHIL i*))))
⟨*proof*⟩

lemma *LPHIL0-rec*:
LPHIL0 = (*picks 0* (*N*−*1*) → (*picks 0 0* → (*putsdown 0 0* → (*putsdown 0* (*N*−*1*) → *LPHIL0*))))
⟨*proof*⟩

lemma *FORK-rec*: *FORK i* = ((*picks i i* → (*putsdown i i* → (*FORK i*)))
□ (*picks* ((*i*+*1*) *mod N*) *i* → (*putsdown* ((*i*+*1*) *mod N*) *i* → (*FORK i*))))
⟨*proof*⟩

3.2.3 Translation into normal form

lemma *N-pos*[*simp*]: *N* > *0*
⟨*proof*⟩

lemmas *N-pos-simps*[*simp*] = *suc-mods*[*OF N-g1*] *l-suc*[*OF N-g1*] *minus-suc*[*OF N-pos*]

The one-fork process

type-synonym id_{fork} = *nat*

type-synonym σ_{fork} = *nat*

definition *fork-transitions*:: $id_{fork} \Rightarrow \sigma_{fork} \Rightarrow$ *dining-event set* (Tr_f)
 where $Tr_f\ i\ s = (if\ s = 0\ \ then\ \{picks\ i\ i\} \cup \{picks\ ((i+1)\ mod\ N)\ i\}$
 $else\ if\ s = 1\ \ then\ \{putsdown\ i\ i\}$
 $else\ if\ s = 2\ \ then\ \{putsdown\ ((i+1)\ mod\ N)\ i\}$
 $else\ \ \{\})$

declare *Un-insert-right*[*simp del*] *Un-insert-left*[*simp del*]

lemma *ev-id*$_{fork}$*x*[*simp*]: $e \in Tr_f\ i\ s \Longrightarrow fork\ e = i$
 ⟨*proof*⟩

definition σ_{fork}*-update*:: $id_{fork} \Rightarrow \sigma_{fork} \Rightarrow$ *dining-event* $\Rightarrow \sigma_{fork}$ (Up_f)
 where $Up_f\ i\ s\ e = (\ if\ e = (picks\ i\ i)\ \ then\ 1$
 $else\ if\ e = (picks\ ((i+1)\ mod\ N)\ i)\ \ then\ 2$
 $else\ \ 0\ )$

definition $FORK_{norm}$:: $id_{fork} \Rightarrow \sigma_{fork} \Rightarrow$ *dining-event process*
 where $FORK_{norm}\ i = P_{norm}[\![Tr_f\ i\ , Up_f\ i]\!]$

lemma $FORK_{norm}$*-rec*: $FORK_{norm}\ i = (\lambda\ s.\ \square\ e \in (Tr_f\ i\ s) \rightarrow FORK_{norm}\ i\ (Up_f\ i\ s\ e))$
 ⟨*proof*⟩

lemma *FORK-refines-FORK*$_{norm}$: $FORK_{norm}\ i\ 0 \sqsubseteq_{FD} FORK\ i$
⟨*proof*⟩

lemma $FORK_{norm}$*-refines-FORK*: $FORK\ i \sqsubseteq_{FD} FORK_{norm}\ i\ 0$
⟨*proof*⟩

lemma $FORK_{norm}$*-is-FORK*: $FORK\ i = FORK_{norm}\ i\ 0$
 ⟨*proof*⟩

The all-forks process in normal form

type-synonym σ_{forks} = *nat list*

definition *forks-transitions*:: $nat \Rightarrow \sigma_{forks} \Rightarrow$ *dining-event set* (Tr_F)
 where $Tr_F\ n\ fs = (\bigcup i{<}n.\ Tr_f\ i\ (fs!i))$

lemma *forks-transitions-take*: $Tr_F\ n\ fs = Tr_F\ n\ (take\ n\ fs)$
 ⟨*proof*⟩

definition σ_{forks}*-update*:: $\sigma_{forks} \Rightarrow$ *dining-event* $\Rightarrow \sigma_{forks}$ (Up_F)
 where $Up_F\ fs\ e = (let\ i{=}(fork\ e)\ in\ fs[i{:=}(Up_f\ i\ (fs!i)\ e)])$

lemma *forks-update-take*: $take\ n\ (Up_F\ fs\ e) = Up_F\ (take\ n\ fs)\ e$
 ⟨*proof*⟩

definition $FORKs_{norm}$:: $nat \Rightarrow \sigma_{forks} \Rightarrow$ *dining-event process*
where $FORKs_{norm}\ n = P_{norm}[\![Tr_F\ n\ , Up_F]\!]$

lemma $FORKs_{norm}$*-rec*: $FORKs_{norm}\ n = (\lambda\ fs.\ \Box\ e \in (Tr_F\ n\ fs) \rightarrow FORKs_{norm}\ n\ (Up_F\ fs\ e))$
⟨*proof*⟩

lemma $FORKs_{norm}$*-0*: $FORKs_{norm}\ 0\ fs = STOP$
⟨*proof*⟩

lemma $FORKs_{norm}$*-1-dir1*: $length\ fs > 0 \Longrightarrow FORKs_{norm}\ 1\ fs \sqsubseteq_{FD} (FORK_{norm}\ 0\ (fs!0))$
⟨*proof*⟩

lemma $FORKs_{norm}$*-1-dir2*: $length\ fs > 0 \Longrightarrow (FORK_{norm}\ 0\ (fs!0)) \sqsubseteq_{FD} FORKs_{norm}\ 1\ fs$
⟨*proof*⟩

lemma $FORKs_{norm}$*-1*: $length\ fs > 0 \Longrightarrow (FORK_{norm}\ 0\ (fs!0)) = FORKs_{norm}\ 1\ fs$
⟨*proof*⟩

lemma $FORKs_{norm}$*-unfold*:
$0 < n \Longrightarrow length\ fs = Suc\ n \Longrightarrow$
$FORKs_{norm}\ (Suc\ n)\ fs = (FORKs_{norm}\ n\ (butlast\ fs) ||| (FORK_{norm}\ n\ (fs!n)))$
⟨*proof*⟩

lemma *ft*: $0 < n \Longrightarrow FORKs_{norm}\ n\ (replicate\ n\ 0) = foldFORKs\ n$
⟨*proof*⟩

corollary *FORKs-is-*$FORKs_{norm}$: $FORKs_{norm}\ N\ (replicate\ N\ 0) = FORKs$
⟨*proof*⟩

The one-philosopher process in normal form:

type-synonym *phil-id* = *nat*
type-synonym *phil-state* = *nat*

definition *rphil-transitions*:: *phil-id* ⇒ *phil-state* ⇒ *dining-event set* (Tr_{rp})
where $Tr_{rp}\ i\ s = ($ *if* $s = 0$ *then* $\{picks\ i\ i\}$
else if $s = 1$ *then* $\{picks\ i\ (i-1)\}$
else if $s = 2$ *then* $\{putsdown\ i\ (i-1)\}$
else if $s = 3$ *then* $\{putsdown\ i\ i\}$
else $\{\})$

definition *lphil0-transitions*:: *phil-state* ⇒ *dining-event set* (Tr_{lp})

where $Tr_{lp}\ s = (\ if\ s = 0\ then\ \{picks\ 0\ (N-1)\}$
$else\ if\ s = 1\ then\ \{picks\ 0\ 0\}$
$else\ if\ s = 2\ then\ \{putsdown\ 0\ 0\}$
$else\ if\ s = 3\ then\ \{putsdown\ 0\ (N-1)\}$
$else\ \{\})$

corollary *rphil-phil*: $e \in Tr_{rp}\ i\ s \Longrightarrow phil\ e = i$
and *lphil0-phil*: $e \in Tr_{lp}\ s \Longrightarrow phil\ e = 0$
⟨*proof*⟩

definition *rphil-state-update*:: $id_{fork} \Rightarrow \sigma_{fork} \Rightarrow dining\text{-}event \Rightarrow \sigma_{fork}\ (Up_{rp})$
where $Up_{rp}\ i\ s\ e = (\ if\ e = (picks\ i\ i)\ then\ 1$
$else\ if\ e = (picks\ i\ (i-1))\ then\ 2$
$else\ if\ e = (putsdown\ i\ (i-1))\ then\ 3$
$else\ 0\ )$

definition *lphil0-state-update*:: $\sigma_{fork} \Rightarrow dining\text{-}event \Rightarrow \sigma_{fork}\ (Up_{lp})$
where $Up_{lp}\ s\ e = (\ if\ e = (picks\ 0\ (N-1))\ then\ 1$
$else\ if\ e = (picks\ 0\ 0)\ then\ 2$
$else\ if\ e = (putsdown\ 0\ 0)\ then\ 3$
$else\ 0\ )$

definition $RPHIL_{norm}$:: $id_{fork} \Rightarrow \sigma_{fork} \Rightarrow dining\text{-}event\ process$
where $RPHIL_{norm}\ i = P_{norm}[\![Tr_{rp}\ i, Up_{rp}\ i]\!]$

definition $LPHIL0_{norm}$:: $\sigma_{fork} \Rightarrow dining\text{-}event\ process$
where $LPHIL0_{norm} = P_{norm}[\![Tr_{lp}, Up_{lp}]\!]$

lemma $RPHIL_{norm}$*-rec*: $RPHIL_{norm}\ i = (\lambda\ s.\ \square\ e \in (Tr_{rp}\ i\ s) \rightarrow RPHIL_{norm}\ i\ (Up_{rp}\ i\ s\ e))$
⟨*proof*⟩

lemma $LPHIL0_{norm}$*-rec*: $LPHIL0_{norm} = (\lambda\ s.\ \square\ e \in (Tr_{lp}\ s) \rightarrow LPHIL0_{norm}\ (Up_{lp}\ s\ e))$
⟨*proof*⟩

lemma *RPHIL-refines-*$RPHIL_{norm}$:
assumes *i-pos*: $i > 0$
shows $RPHIL_{norm}\ i\ 0 \sqsubseteq_{FD} RPHIL\ i$
⟨*proof*⟩

lemma *LPHIL0-refines-*$LPHIL0_{norm}$: $LPHIL0_{norm}\ 0 \sqsubseteq_{FD} LPHIL0$
⟨*proof*⟩

lemma $RPHIL_{norm}$*-refines-RPHIL*:
assumes *i-pos*: $i > 0$
shows $RPHIL\ i \sqsubseteq_{FD} RPHIL_{norm}\ i\ 0$

⟨*proof*⟩

lemma $LPHIL0_{norm}$*-refines-LPHIL0*: $LPHIL0 \sqsubseteq_{FD} LPHIL0_{norm}\ 0$
⟨*proof*⟩

lemma $RPHIL_{norm}$*-is-RPHIL*: $i > 0 \Longrightarrow RPHIL\ i = RPHIL_{norm}\ i\ 0$
⟨*proof*⟩

lemma $LPHIL0_{norm}$*-is-LPHIL0*: $LPHIL0 = LPHIL0_{norm}\ 0$
⟨*proof*⟩

3.2.4 The normal form for the global philosopher network

type-synonym $\sigma_{phils} = nat\ list$

definition *phils-transitions*:: $nat \Rightarrow \sigma_{phils} \Rightarrow dining\text{-}event\ set\ (Tr_P)$
where $Tr_P\ n\ ps = Tr_{lp}\ (ps!0) \cup (\bigcup i\in\{1..< n\}.\ Tr_{rp}\ i\ (ps!i))$

corollary *phils-phil*: $0 < n \Longrightarrow e \in Tr_P\ n\ s \Longrightarrow phil\ e < n$
⟨*proof*⟩

lemma *phils-transitions-take*: $0 < n \Longrightarrow Tr_P\ n\ ps = Tr_P\ n\ (take\ n\ ps)$
⟨*proof*⟩

definition σ_{phils}*-update*:: $\sigma_{phils} \Rightarrow dining\text{-}event \Rightarrow \sigma_{phils}\ (Up_P)$
where $Up_P\ ps\ e = (let\ i{=}(phil\ e)\ in\ if\ i = 0\ then\ ps[i{:=}(Up_{lp}\ (ps!i)\ e)]$
$else\ ps[i{:=}(Up_{rp}\ i\ (ps!i)\ e)])$

lemma *phils-update-take*: $take\ n\ (Up_P\ ps\ e) = Up_P\ (take\ n\ ps)\ e$
⟨*proof*⟩

definition $PHILs_{norm}$:: $nat \Rightarrow \sigma_{phils} \Rightarrow dining\text{-}event\ process$
where $PHILs_{norm}\ n = P_{norm}[\![Tr_P\ n, Up_P]\!]$

lemma $PHILs_{norm}$*-rec*: $PHILs_{norm}\ n = (\lambda\ ps.\ \Box\ e \in (Tr_P\ n\ ps) \rightarrow PHILs_{norm}\ n\ (Up_P\ ps\ e))$
⟨*proof*⟩

lemma $PHILs_{norm}$*-1-dir1*: $length\ ps > 0 \Longrightarrow PHILs_{norm}\ 1\ ps \sqsubseteq_{FD} (LPHIL0_{norm}\ (ps!0))$
⟨*proof*⟩

lemma $PHILs_{norm}$*-1-dir2*: $length\ ps > 0 \Longrightarrow (LPHIL0_{norm}\ (ps!0)) \sqsubseteq_{FD} PHILs_{norm}\ 1\ ps$
⟨*proof*⟩

lemma $PHILs_{norm}$*-1*: $length\ ps > 0 \Longrightarrow PHILs_{norm}\ 1\ ps = (LPHIL0_{norm}\ (ps!0))$
⟨*proof*⟩

lemma $PHILs_{norm}$*-unfold*:
assumes *n-pos*: $0 < n$
shows $length\ ps = Suc\ n \Longrightarrow$
$PHILs_{norm}\ (Suc\ n)\ ps = (PHILs_{norm}\ n\ (butlast\ ps) ||| (RPHIL_{norm}\ n\ (ps!n)))$
⟨*proof*⟩

lemma *pt*: $0 < n \Longrightarrow PHILs_{norm}\ n\ (replicate\ n\ 0) = foldPHILs\ n$
⟨*proof*⟩

corollary *PHILs-is-PHILs*$_{norm}$: $PHILs_{norm}\ N\ (replicate\ N\ 0) = PHILs$
⟨*proof*⟩

3.2.5 The complete process system under normal form

definition *dining-transitions*:: $nat \Rightarrow \sigma_{phils} \times \sigma_{forks} \Rightarrow dining\text{-}event\ set\ (Tr_D)$
where $Tr_D\ n = (\lambda(ps,fs).\ (Tr_P\ n\ ps) \cap (Tr_F\ n\ fs))$

definition *dining-state-update*::
$\sigma_{phils} \times \sigma_{forks} \Rightarrow dining\text{-}event \Rightarrow \sigma_{phils} \times \sigma_{forks}\ (Up_D)$
where $Up_D = (\lambda(ps,fs)\ e.\ (Up_P\ ps\ e,\ Up_F\ fs\ e))$

definition $DINING_{norm}$:: $nat \Rightarrow \sigma_{phils} \times \sigma_{forks} \Rightarrow dining\text{-}event\ process$
where $DINING_{norm}\ n = P_{norm}[\![Tr_D\ n,\ Up_D]\!]$

lemma *ltsDining-rec*: $DINING_{norm}\ n = (\lambda\ s.\ \Box\ e \in (Tr_D\ n\ s) \to DINING_{norm}\ n\ (Up_D\ s\ e))$
⟨*proof*⟩

lemma *DINING-is-DINING*$_{norm}$: $DINING = DINING_{norm}\ N\ (replicate\ N\ 0,\ replicate\ N\ 0)$
⟨*proof*⟩

3.2.6 And finally: Philosophers may dine ! Always !

corollary *lphil-states*: $Up_{lp}\ r\ e = 0 \lor Up_{lp}\ r\ e = 1 \lor Up_{lp}\ r\ e = 2 \lor Up_{lp}\ r\ e = 3$
and *rphil-states*: $Up_{rp}\ i\ r\ e = 0 \lor Up_{rp}\ i\ r\ e = 1 \lor Up_{rp}\ i\ r\ e = 2 \lor Up_{rp}\ i\ r\ e = 3$
⟨*proof*⟩

lemma *dining-events*:
$e \in Tr_D\ N\ s \Longrightarrow$
$(\exists\, i \in \{1..<N\}.\ e = picks\ i\ i \lor e = picks\ i\ (i-1) \lor e = putsdown\ i\ i \lor e = putsdown\ i\ (i-1))$
$\lor\ (e = picks\ 0\ 0 \lor e = picks\ 0\ (N-1) \lor e = putsdown\ 0\ 0 \lor e = putsdown\ 0\ (N-1))$
⟨*proof*⟩

definition *inv-dining ps fs* $\equiv$
$(\forall i.\ Suc\ i < N \longrightarrow ((fs!(Suc\ i) = 1) \longleftrightarrow ps!Suc\ i \neq 0)) \wedge (fs!(N-1) = 2 \longleftrightarrow ps!0 \neq 0)$
$\wedge (\forall i < N - 1.\ fs!i = 2 \longleftrightarrow ps!Suc\ i = 2) \wedge (fs!0 = 1 \longleftrightarrow ps!0 = 2)$
$\wedge (\forall i < N.\ fs!i = 0 \vee fs!i = 1 \vee fs!i = 2)$
$\wedge (\forall i < N.\ ps!i = 0 \vee ps!i = 1 \vee ps!i = 2 \vee ps!i = 3)$
$\wedge\ length\ fs = N \wedge length\ ps = N$

lemma *inv-DINING*: $s \in \mathfrak{R}\ (Tr_D\ N)\ Up_D\ (replicate\ N\ 0,\ replicate\ N\ 0) \Longrightarrow inv\text{-}dining\ (fst\ s)\ (snd\ s)$
$\langle proof \rangle$

lemma *inv-implies-DF*: $inv\text{-}dining\ ps\ fs \Longrightarrow Tr_D\ N\ (ps, fs) \neq \{\}$
$\langle proof \rangle$

corollary *deadlock-free-DINING*: *deadlock-free DINING*
$\langle proof \rangle$

corollary $deadlock\text{-}free_{SKIPS}\text{-}DINING$: $deadlock\text{-}free_{SKIPS}\ DINING$
$\langle proof \rangle$

end

end

Chapter 4

Conclusion

We presented a formalisation of the most comprehensive semantic model for CSP, a 'classical' language for the specification and analysis of concurrent systems studied in a rich body of literature. For this purpose, we ported [12] to a modern version of Isabelle, restructured the proofs, and extended the resulting theory of the language substantially. The result HOL-CSP 2 has been submitted to the Isabelle AFP [10], thus a fairly sustainable format accessible to other researchers and tools.

We developed a novel set of deadlock - and livelock inference proof principles based on classical and denotational characterizations. In particular, we formally investigated the relations between different refinement notions in the presence of deadlock - and livelock; an area where traditional CSP literature skates over the nitty-gritty details. Finally, we demonstrated how to exploit these results for deadlock/livelock analysis of protocols.

We put a large body of abstract CSP laws and induction principles together to form concrete verification technologies for generalized classical problems, which have been considered so far from the perspective of data-independence or structural parametricity. The underlying novel principle of "trading rich structure against rich state" allows one to convert processes into classical transition systems for which established invariant techniques become applicable.

Future applications of HOL-CSP 2 could comprise a combination with model checkers, where our theory with its derived rules can be used to certify the output of a model-checker over CSP. In our experience, labelled transition systems generated by model checkers may be used to steer inductions or to construct the normalized processes $P_{norm}[\![\tau,\upsilon]\!]$ automatically, thus combining efficient finite reasoning over finite sub-systems with globally infinite systems in a logically safe way.

Bibliography

[1] G. Barrett. Model checking in practice: the t9000 virtual channel processor. *IEEE Transactions on Software Engineering*, 21(2):69–78, Feb 1995.

[2] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[3] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 281–305, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[4] A. J. Camilleri. A higher order logic mechanization of the csp failure-divergence semantics. In G. Birtwistle, editor, *IV Higher Order Workshop, Banff 1990*, pages 123–150, London, 1991. Springer London.

[5] A. Donovan and B. Kernighan. *The Go Programming Language*. Addison-Wesley Professional Computing Series. Pearson Education, 2015.

[6] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[7] Y. Isobe and M. Roggenbach. Csp-prover: a proof tool for the verification of scalable concurrent systems. *Information and Media Technologies*, 5(1):32–39, 2010.

[8] A. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1998.

[9] D. Scott. Continuous lattices. In F. W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, pages 97–136, Berlin, Heidelberg, 1972. Springer.

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

[11] S. Taha, L. Ye, and B. Wolff. Philosophers may Dine - Definitively! In C. A. Furia, editor, *Integrated Formal Methods (iFM)*, number 12546 in Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2020.

[12] H. Tej and B. Wolff. A corrected failure divergence model for CSP in Isabelle/HOL. In J. S. Fitzgerald, C. B. Jones, and P. Lucas, editors, *Formal Methods Europe (FME)*, volume 1313 of *Lecture Notes in Computer Science*, pages 318–337, Heidelberg, 1997. Springer-Verlag.