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

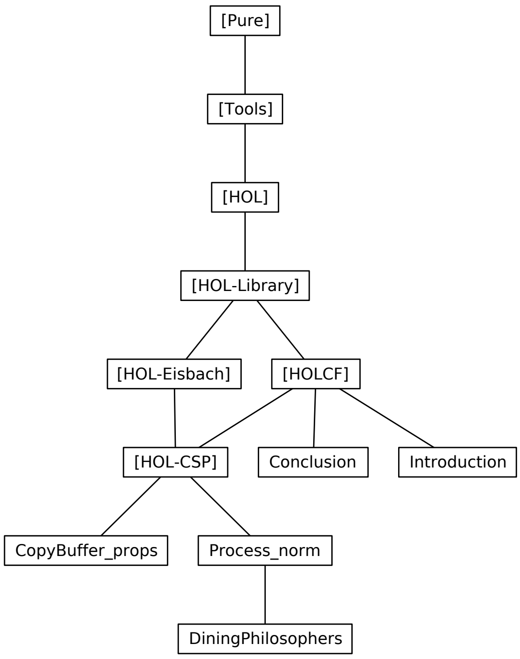

Figure 1.1: The overall architecture: HOLCF, HOL-CSP, and CSP_RefTk

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* τ υ $\equiv$ (μ X. (λ s. $\Box$ $e \in (\tau\ s) \rightarrow X\ (\upsilon\ s\ e)$))

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* (λX. (λs. $\Box$ $e \in (\tau\ s) \rightarrow X\ (\upsilon\ s\ e)$)) (**is** *cont ?f*)
proof −
  have *cont* (λX. *?f X s*) **for** *s* **by** (*simp add*:*cont-fun*)
  then show *?thesis* **by** *simp*
qed

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*$\cdot$(Λ X. *?P X*))
  defines *Q*: $Q \equiv P_{norm}[\![\tau_2,\upsilon_2]\!]$ (**is** $Q \equiv$ *fix*$\cdot$(Λ 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** $S \equiv \mathit{fix}\cdot(\Lambda\ X.\ ?S\ X)$)

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

proof −
  have *P-rec*: $P = ?P\ P$ **using** *fix-eq*[*of* $(\Lambda\ X.\ ?P\ X)$] P **by** *simp*
  have *Q-rec*: $Q = ?Q\ Q$ **using** *fix-eq*[*of* $(\Lambda\ X.\ ?Q\ X)$] Q **by** *simp*
  have *S-rec*: $S = ?S\ S$ **using** *fix-eq*[*of* $(\Lambda\ X.\ ?S\ X)$] S **by** *simp*
  have *dir1*: $\forall\ s_1\ s_2.\ (P\ s_1\ |||\ Q\ s_2) \sqsubseteq_{FD} S\ (s_1,\ s_2)$
  proof(*subst P*, *subst Q*,
    induct rule:*parallel-fix-ind-inc*[*of* $\lambda x\ y.\ \forall\ s_1\ s_2.\ (x\ s_1\ |||\ y\ s_2) \sqsubseteq_{FD} S\ (s_1,s_2)$])
   case *admissibility*
   then show *?case*
    by (*intro adm-all le-FD-adm*) (*simp-all add*: *cont2cont-fun monofunI*)
  next
   case (*base-fst y*)
   then show *?case* **by** (*metis app-strict BOT-leFD Sync-BOT Sync-commute*)
  next
   case (*base-snd x*)
   then show *?case* **by** *simp*
  next
   case (*step x*)
   then show *?case* (**is** $\forall\ s_1\ s_2.\ ?C\ s_1\ s_2$)
   proof(*intro allI*)
    fix $s_1\ s_2$
    show $?C\ s_1\ s_2$
     apply *simp*
     apply (*subst Mprefix-Sync-Mprefix-indep*[**where** $S = \{\}$, *simplified*])
     apply (*subst S-rec*, *simp add*: *Tr Up Mprefix-Un-distrib*)
     apply (*intro mono-Det-FD mono-Mprefix-FD*)
     using *step*(*3*)[*simplified*] *indep* **apply** *simp*
     using *step*(*2*)[*simplified*] *indep* **by** *fastforce*
   qed
  qed
  have *dir2*: $\forall\ s_1\ s_2.\ S\ (s_1,\ s_2) \sqsubseteq_{FD} (P\ s_1\ |||\ Q\ s_2)$
  proof(*subst S*, *induct rule*:*fix-ind-k*[*of* $\lambda x.\ \forall\ s_1\ s_2.\ x\ (s_1,s_2) \sqsubseteq_{FD} (P\ s_1\ |||\ Q\ s_2)$ *1*])
   case *admissibility*
   show *?case* **by** (*intro adm-all le-FD-adm*) (*simp-all add*: *cont-fun monofunI*)
  next
   case *base-k-steps*
   then show *?case* **by** *simp*
  next
   case *step*
   then show *?case* (**is** $\forall\ s_1\ s_2.\ ?C\ s_1\ s_2$)
   proof(*intro allI*)
    fix $s_1\ s_2$

have *P-rec-sym*:*Mprefix* (τ_1 s_1) ($\lambda e.$ P (υ_1 s_1 e)) = P s_1 **using** *P-rec* **by** *metis*
have *Q-rec-sym*:*Mprefix* (τ_2 s_2) ($\lambda e.$ Q (υ_2 s_2 e)) = Q s_2 **using** *Q-rec* **by** *metis*
show *?C* s_1 s_2
apply (*simp add*: *Tr Up Mprefix-Un-distrib*)
apply (*subst P-rec*, *subst Q-rec*, *subst Mprefix-Sync-Mprefix-indep*[**where** S={}, *simplified*])
apply (*intro mono-Det-FD mono-Mprefix-FD*)
apply (*subst Q-rec-sym*, *simp add*:*step*[*simplified*])
apply (*subst P-rec-sym*) **using** *step*[*simplified*] *indep* **by** *fastforce*
qed
qed
from *dir1 dir2* **show** *?thesis* **using** *FD-antisym* **by** *blast*
qed

2.3 Synchronous product lemma

lemma *dnorm-par*:
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*$\cdot$(Λ X. *?P* X))
defines *Q*: $Q \equiv P_{norm}[\![\tau_2,\upsilon_2]\!]$ (**is** $Q \equiv$ *fix*$\cdot$(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*$\cdot$(Λ X. *?S* X))

shows (P s_1 || Q s_2) = S (s_1,s_2)

proof −
have *P-rec*: P = *?P* P **using** *fix-eq*[*of* (Λ X. *?P* X)] P **by** *simp*
have *Q-rec*: Q = *?Q* Q **using** *fix-eq*[*of* (Λ X. *?Q* X)] Q **by** *simp*
have *S-rec*: S = *?S* S **using** *fix-eq*[*of* (Λ X. *?S* X)] S **by** *simp*
have *dir1*: $\forall$ s_1 s_2. (P s_1 || Q s_2) $\sqsubseteq_{FD}$ S (s_1, s_2)
proof(*subst P*, *subst Q*,
induct rule:*parallel-fix-ind*[*of* $\lambda x\ y.$ $\forall$ s_1 s_2. (x s_1 || y s_2) $\sqsubseteq_{FD}$ S (s_1,s_2)])
case *adm*:*1*
then show *?case*
by (*intro adm-all le-FD-adm*) (*simp-all add*: *cont2cont-fun monofunI*)
next
case *base*:*2*
then show *?case* **by** (*simp add*: *Sync-BOT*)
next
case *step*:(*3 x y*)
then show *?case* (**is** $\forall$ s_1 s_2. *?C* s_1 s_2)
proof(*intro allI*)
fix s_1 s_2
show *?C* s_1 s_2

apply(*simp*)
apply(*subst Mprefix-Sync-Mprefix-subset*[**where** *S*=*UNIV*, *simplified*])
apply(*subst S-rec*, *simp add*: *Tr Up Mprefix-Un-distrib*)
by (*simp add*: *step mono-Mprefix-FD*)
qed
qed
have *dir2*: $\forall\ s_1\ s_2.\ S\ (s_1,\ s_2) \sqsubseteq_{FD} (P\ s_1 \mid\mid Q\ s_2)$
proof(*subst S*, *induct rule*:*fix-ind-k*[*of* $\lambda x.\ \forall\ s_1\ s_2.\ x\ (s_1,s_2) \sqsubseteq_{FD} (P\ s_1 \mid\mid Q\ s_2)$ *1*])
case *admissibility*
show *?case* **by** (*intro adm-all le-FD-adm*) (*simp-all add*: *cont-fun monofunI*)
next
case *base-k-steps*
then show *?case* **by** *simp*
next
case *step*
then show *?case* (**is** $\forall\ s_1\ s_2.$ *?C* $s_1\ s_2$)
proof(*intro allI*)
fix $s_1\ s_2$
have *P-rec-sym*:*Mprefix* $(\tau_1\ s_1)\ (\lambda e.\ P\ (\upsilon_1\ s_1\ e)) = P\ s_1$ **using** *P-rec* **by** *metis*
have *Q-rec-sym*:*Mprefix* $(\tau_2\ s_2)\ (\lambda e.\ Q\ (\upsilon_2\ s_2\ e)) = Q\ s_2$ **using** *Q-rec* **by** *metis*
show *?C* $s_1\ s_2$
apply(*simp add*: *Tr Up*)
apply(*subst P-rec*, *subst Q-rec*, *subst Mprefix-Sync-Mprefix-subset*[**where** *S*=*UNIV*, *simplified*])
apply(*rule mono-Mprefix-FD*)
using *step* **by** *auto*
qed
qed
from *dir1 dir2* **show** *?thesis* **using** *FD-antisym* **by** *blast*
qed

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*$\cdot(\Lambda\ X.\ ?P\ X)$)
shows $s \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0 \Longrightarrow$ *deadlock-free* $(P\ s)$
proof(*unfold deadlock-free-def DF-def*, *induct arbitrary*:*s rule*:*fix-ind*)
show *adm* $(\lambda a.\ \forall\, x.\ x \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0 \longrightarrow a \sqsubseteq_{FD} P\ x)$ **by** (*simp add*: *monofun-def*)

next
fix $s :: '\sigma$
show $s \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0 \Longrightarrow \bot \sqsubseteq_{FD} P\ s$ **by** *simp*
next
fix *s x* **assume** *1* : $\bigwedge s.\ s \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0 \Longrightarrow x \sqsubseteq_{FD} P\ s$
and *2* : $s \in \mathfrak{R}\ \tau\ \upsilon\ \sigma_0$
have *P-rec*: $P = ?P\ P$ **using** *fix-eq*[*of* $(\Lambda\ X.\ ?P\ X)$] *P* **by** *simp*

from *1 2* **show** $(\Lambda\ x.\ (\sqcap xa{\in}UNIV \rightarrow\ \ x))\cdot x \sqsubseteq_{FD} P\ s$
apply (*subst P-rec*, *rule-tac trans-FD*[*rotated*, *OF Mndetprefix-FD-Mprefix*])
apply *simp*
apply (*rule trans-FD*[*OF Mndetprefix-FD-subset*[*of* ‹τ *s*› ‹*UNIV*›]
mono-Mndetprefix-FD[*rule-format*, *OF 1*]])
using *non-reachable-sink*[*rule-format*, *OF 2*] **apply** *assumption*
by *blast* (*meson* $\mathfrak{R}$.*rstep*)
qed

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*
apply (*meson DF-COPY DF-Univ-freeness UNIV-not-empty image-is-empty sup-eq-bot-iff*)
apply (*rule deadlock-free-implies-lifelock-free*)
by (*metis DF-COPY DF-Univ-freeness UNIV-I empty-iff image-eqI le-sup-iff subset-empty*)

corollary df_{SKIPS}*-COPY*: $deadlock\text{-}free_{SKIPS}$ *COPY*
and lf_{SKIPS}*-COPY*: $lifelock\text{-}free_{SKIPS}$ *COPY*
and *nt-COPY*: *non-terminating COPY*
apply (*simp add*: *df-COPY* $deadlock\text{-}free\text{-}imp\text{-}deadlock\text{-}free_{SKIPS}$)
apply (*simp add*: *lf-COPY* $lifelock\text{-}free\text{-}imp\text{-}lifelock\text{-}free_{SKIPS}$)
using *lf-COPY lifelock-free-is-non-terminating* **by** *blast*

lemma *DF-SYSTEM*: *DF UNIV* $\sqsubseteq_{FD}$ *SYSTEM*
using *deadlock-free-def df-COPY impl-refines-spec′ trans-FD* **by** *blast*

corollary *df-SYSTEM*: *deadlock-free SYSTEM*
and *lf-SYSTEM*: *lifelock-free SYSTEM*
apply (*simp add*: *DF-SYSTEM deadlock-free-def*)

apply (*rule deadlock-free-implies-lifelock-free*)
by (*simp add*: *DF-SYSTEM deadlock-free-def*)

corollary df_{SKIPS}*-SYSTEM*: *deadlock-free*$_{SKIPS}$ *SYSTEM*
and lf_{SKIPS}*-SYSTEM*: *lifelock-free*$_{SKIPS}$ *SYSTEM*
and *nt-SYSTEM*: *non-terminating SYSTEM*
apply (*simp add*: *df-SYSTEM deadlock-free-imp-deadlock-free*$_{SKIPS}$)
apply (*simp add*: *lf-SYSTEM lifelock-free-imp-lifelock-free*$_{SKIPS}$)
using *lf-SYSTEM lifelock-free-is-non-terminating* **by** *blast*

end

3.2 Generalized Dining Philosophers

theory *DiningPhilosophers*
imports *Process-norm*
begin

3.2.1 Preliminary lemmas for proof automation

lemma *Suc-mod*: $n > 1 \Longrightarrow i \neq Suc\ i\ mod\ n$
by (*metis One-nat-def mod-Suc mod-if mod-mod-trivial n-not-Suc-n*)

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

lemma *l-suc*: $n > 1 \Longrightarrow \neg\ n \leq Suc\ 0$
by *simp*

lemma *minus-suc*: $n > 0 \Longrightarrow n - Suc\ 0 \neq n$
by *linarith*

lemma *numeral-4-eq-4*:$4 = Suc\ (Suc\ (Suc\ (Suc\ 0)))$
by *simp*

lemma *numeral-5-eq-5*:$5 = Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))$
by *simp*

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* = (μ *X*. (*picks i i* → (*picks i* (*i*−*1*) → (*putsdown i* (*i*−*1*) → (*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* :: *nat* ⇒ *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*)))

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*)
by (*subst upt-rec*, *auto simp add*:*numeral-2-eq-2*)+

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*))))
by (*simp add*:*RPHIL-def write0-def*, *subst fix-eq*, *simp*)

lemma *LPHIL0-rec*:
LPHIL0 = (*picks 0* (*N*−*1*) → (*picks 0 0* → (*putsdown 0 0* → (*putsdown 0* (*N*−*1*) → *LPHIL0*))))
by (*simp add*:*LPHIL0-def write0-def*, *subst fix-eq*, *simp*)

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*))))
by (*simp add*:*FORK-def write0-def*, *subst fix-eq*, *simp*)

3.2.3 Translation into normal form

lemma *N-pos*[*simp*]: *N* > *0*
using *N-g1 neq0-conv* **by** *blast*

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*} ∪ {*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*
  by (*auto simp add*:*fork-transitions-def split*:*if-splits*)

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* = (λ *s*. □ *e* ∈ (Tr_f *i s*) → $FORK_{norm}$ *i* (Up_f *i s e*))
  using *fix-eq*[*of* Λ *X*. (λ*s*. *Mprefix* (Tr_f *i s*) (λ*e*. *X* (Up_f *i s e*)))] $FORK_{norm}$*-def*
by *simp*

lemma *FORK-refines-FORK*$_{norm}$: $FORK_{norm}$ *i 0* $\sqsubseteq_{FD}$ *FORK i*
proof(*unfold FORK*$_{norm}$*-def*,
    induct rule:*fix-ind-k-skip*[**where** *k*=*2* **and** *f*=Λ *x*.(λ*s*. *Mprefix* (Tr_f *i s*) (λ*e*. *x* (Up_f *i s e*)))])
  case *lower-bound*
  then show *?case* **by** *simp*
next
  case *admissibility*
  then show *?case* **by** (*simp add*: *cont-fun monofunI*)
next
  case *base-k-steps*
  then show *?case* (**is** ∀ *j*<*2*. (*iterate j*·*?f*·⊥) *0* $\sqsubseteq_{FD}$ *FORK i*)
  proof −
    have *less-2*:⋀*j*. (*j*::*nat*) < *2* = (*j* = *0* ∨ *j* = *1*) **by** *linarith*
    moreover have (*iterate 0*·*?f*·⊥) *0* $\sqsubseteq_{FD}$ *FORK i* **by** *simp*
    moreover have (*iterate 1*·*?f*·⊥) *0* $\sqsubseteq_{FD}$ *FORK i*

by (*subst FORK-rec*)
(*simp add*: *write0-def fork-transitions-def Mprefix-Un-distrib mono-Mprefix-FD mono-Det-FD*)
ultimately show *?thesis* **by** *simp*
qed
next
case (*step x*)
then show *?case* (**is** (*iterate 2·?f·x*) *0* $\sqsubseteq_{FD}$ *FORK i*)
apply (*subst FORK-rec*)
by (*auto simp add*: *write0-def numeral-2-eq-2 fork-transitions-def* σ_{fork}*-update-def Mprefix-Un-distrib*
intro!: *mono-Det-FD mono-Mprefix-FD*)
qed

lemma *FORK*$_{norm}$*-refines-FORK*: *FORK i* $\sqsubseteq_{FD}$ *FORK*$_{norm}$ *i 0*
proof(*unfold FORK-def*,
induct rule:*fix-ind-k-skip*[**where** *k=1*])
show (*1*::*nat*) $\leq$ *1* **by** *simp*
next
show *adm* (λ*a*. *a* $\sqsubseteq_{FD}$ *FORK*$_{norm}$ *i 0*) **by** (*simp add*: *monofunI*)
next
case *base-k-steps*
show *?case* **by** *simp*
next
case (*step x*)
then show *?case* (**is** *iterate 1·?f·x* $\sqsubseteq_{FD}$ *FORK*$_{norm}$ *i 0*)
apply (*subst FORK*$_{norm}$*-rec*)
apply (*simp add*: *write0-def fork-transitions-def* σ_{fork}*-update-def Mprefix-Un-distrib*)
apply (*subst* (*1 2*) *FORK*$_{norm}$*-rec*)
by (*auto simp add*: *fork-transitions-def* σ_{fork}*-update-def Mprefix-Un-distrib intro*!: *mono-Mprefix-FD mono-Det-FD*)
qed

lemma *FORK*$_{norm}$*-is-FORK*: *FORK i* = *FORK*$_{norm}$ *i 0*
using *FORK-refines-FORK*$_{norm}$ *FORK*$_{norm}$*-refines-FORK FD-antisym* **by** *blast*

The all-forks process in normal form

type-synonym σ_{forks} = *nat list*

definition *forks-transitions*:: *nat* $\Rightarrow$ σ_{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*)
by (*simp add*:*forks-transitions-def*)

definition σ_{forks}*-update*:: σ_{forks} $\Rightarrow$ *dining-event* $\Rightarrow$ σ_{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*
unfolding σ_{forks}*-update-def*
by (*metis nat-less-le nat-neq-iff nth-take order-refl take-update-cancel take-update-swap*)

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

lemma $FORKs_{norm}$*-rec*: $FORKs_{norm}$ *n* = (λ *fs*. □ *e* ∈ (Tr_F *n fs*) → $FORKs_{norm}$ *n* (Up_F *fs e*))
using *fix-eq*[*of* Λ *X*. (λ*fs*. *Mprefix* (Tr_F *n fs*) (λ*e*. *X* (Up_F *fs e*)))] $FORKs_{norm}$*-def*
by *simp*

lemma $FORKs_{norm}$*-0*: $FORKs_{norm}$ *0 fs* = *STOP*
by (*subst* $FORKs_{norm}$*-rec*, *simp add*:*forks-transitions-def*)

lemma $FORKs_{norm}$*-1-dir1*: *length fs* > *0* ⟹ $FORKs_{norm}$ *1 fs* $\sqsubseteq_{FD}$ ($FORK_{norm}$ *0* (*fs*!*0*))
proof(*unfold* $FORKs_{norm}$*-def*,
induct arbitrary:*fs rule*:*fix-ind-k*[**where** *k*=*1*
and *f*=Λ *x*. (λ*fs*. *Mprefix* (Tr_F *1 fs*) (λ*e*. *x* (Up_F *fs e*)))])
case *admissibility*
then show *?case* **by** (*simp add*: *cont-fun monofunI*)
next
case *base-k-steps*
then show *?case* **by** *simp*
next
case (*step X*)
hence ($\bigcup$ *i*<*Suc 0*. Tr_f *i* (*fs* ! *i*)) = Tr_f *0* (*fs* ! *0*) **by** *auto*
with *step* **show** *?case*
apply (*subst* $FORK_{norm}$*-rec*, *simp add*:σ_{forks}*-update-def forks-transitions-def*)

apply (*intro mono-Mprefix-FD*)
by (*metis ev-id*$_{fork}$*x step.prems list-update-nonempty nth-list-update-eq*)
qed

lemma $FORKs_{norm}$*-1-dir2*: *length fs* > *0* ⟹ ($FORK_{norm}$ *0* (*fs*!*0*)) $\sqsubseteq_{FD}$ $FORKs_{norm}$ *1 fs*
proof(*unfold* $FORK_{norm}$*-def*,
induct arbitrary:*fs rule*:*fix-ind-k*[**where** *k*=*1*
and *f*=Λ *x*. (λ*s*. *Mprefix* (Tr_f *0 s*) (λ*e*. *x* (Up_f *0 s e*)))])
case *admissibility*
then show *?case* **by** (*simp add*: *cont-fun monofunI*)
next
case *base-k-steps*
then show *?case* **by** *simp*
next

case (*step X*)
have ($\bigcup$ *i<Suc 0. Tr_f i (fs ! i)*) = *Tr_f 0 (fs ! 0)* **by** *auto*
with *step* **show** *?case*
apply (*subst FORKs$_{norm}$-rec, simp add: σ_{forks}-update-def forks-transitions-def*)

apply (*intro mono-Mprefix-FD*)
by (*metis ev-id$_{fork}$x step.prems list-update-nonempty nth-list-update-eq*)
qed

lemma *FORKs$_{norm}$-1*: *length fs* > *0* $\Longrightarrow$ (*FORK$_{norm}$ 0 (fs!0)*) = *FORKs$_{norm}$ 1 fs*
using *FORKs$_{norm}$-1-dir1 FORKs$_{norm}$-1-dir2 FD-antisym* **by** *blast*

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(*rule FD-antisym*)
show *0* < *n* $\Longrightarrow$ *length fs* = *Suc n* $\Longrightarrow$
FORKs$_{norm}$ (Suc n) fs $\sqsubseteq_{FD}$ (*FORKs$_{norm}$ n (butlast fs)*|||*FORK$_{norm}$ n (fs!n)*)
proof(*subst FORKs$_{norm}$-def,*
induct arbitrary:fs
rule:fix-ind-k[**where** *k=1*
and *f=Λ x.* (*λfs. Mprefix (Tr$_F$ (Suc n) fs)* (*λe. x (Up$_F$ fs e)*))])
case *admissibility*
then show *?case* **by** (*simp add: cont-fun monofunI*)
next
case *base-k-steps*
then show *?case* **by** *simp*
next
case (*step X*)
have *indep*:∀ *s$_1$ s$_2$. Tr$_F$ n s$_1$* ∩ *Tr$_f$ n s$_2$* = {}
by (*auto simp add:forks-transitions-def fork-transitions-def*)
from *step* **show** *?case*
apply (*auto simp add:indep dnorm-inter FORKs$_{norm}$-def FORK$_{norm}$-def*)
apply (*subst fix-eq, simp add:forks-transitions-def Un-commute lessThan-Suc nth-butlast*)
proof(*rule mono-Mprefix-FD, safe, goal-cases*)
case (*1 e*)
from *1(4)* **have** *a:fork e* = *n*
by (*auto simp add:fork-transitions-def split:if-splits*)
show *?case*
using *1(1)*[*rule-format, of (Up$_F$ fs e)*]
apply (*simp add: 1 a butlast-list-update σ_{forks}-update-def*)
by (*metis 1(4) ev-id$_{fork}$x lessThan-iff less-not-refl*)
next
case (*2 e m*)

hence *a*:$e \notin Tr_f\ n\ (fs\ !\ n)$
using *ev-id*$_{fork}$*x* **by** *fastforce*
hence *c*:$Up_F\ fs\ e\ !\ n = fs\ !\ n$
by (*metis 2*(*4*) *ev-id*$_{fork}$*x* σ_{forks}*-update-def nth-list-update-neq*)
have *d*:Up_F (*butlast fs*) *e* = *butlast* (Up_F *fs e*)
apply(*simp add*:σ_{forks}*-update-def*)
by (*metis butlast-conv-take* σ_{forks}*-update-def forks-update-take length-list-update*)
from *2 a* **show** *?case*
using *2*(*1*)[*rule-format*, *of* (Up_F *fs e*)] *c d* σ_{forks}*-update-def* **by** *auto*
qed
qed
next
have *indep*:$\forall\ s_1\ s_2.\ Tr_F\ n\ s_1 \cap Tr_f\ n\ s_2 = \{\}$
by (*auto simp add*: *forks-transitions-def fork-transitions-def*)
show $0 < n \Longrightarrow$ *length fs* = *Suc n* $\Longrightarrow$
($FORKs_{norm}$ *n* (*butlast fs*)|||$FORK_{norm}$ *n* (*fs*!*n*)) $\sqsubseteq_{FD}$
$FORKs_{norm}$ (*Suc n*) *fs*
apply (*subst* $FORKs_{norm}$*-def*, *auto simp add*:*indep dnorm-inter* $FORK_{norm}$*-def*)
proof(*rule fix-ind*[**where**
$P=\lambda a.\ 0 < n \longrightarrow (\forall x.\ length\ x = Suc\ n \longrightarrow a\ (butlast\ x,\ x\ !\ n) \sqsubseteq_{FD}$
$FORKs_{norm}$ (*Suc n*) *x*),
rule-format], *simp-all*, *goal-cases*)
case *base*:*1*
then show *?case* **by** (*simp add*: *cont-fun monofunI*)
next
case *step*:(*2 X fs*)
then show *?case*
apply (*unfold* $FORKs_{norm}$*-def*, *subst fix-eq*, *simp add*:*forks-transitions-def*
Un-commute lessThan-Suc nth-butlast)
proof(*rule mono-Mprefix-FD*, *safe*, *goal-cases*)
case (*1 e*)
from *1*(*6*) **have** *a*:*fork e* = *n*
by (*auto simp add*:*fork-transitions-def split*:*if-splits*)
show *?case*
using *1*(*1*)[*rule-format*, *of* (Up_F *fs e*)]
apply (*simp add*: *1 a butlast-list-update* σ_{forks}*-update-def*)
using *a ev-id*$_{fork}$*x* **by** *blast*
next
case (*2 e m*)
have *a*:Up_F (*butlast fs*) *e* = *butlast* (Up_F *fs e*)
apply(*simp add*:σ_{forks}*-update-def*)
by (*metis butlast-conv-take* σ_{forks}*-update-def forks-update-take length-list-update*)
from *2* **show** *?case*
using *2*(*1*)[*rule-format*, *of* (Up_F *fs e*)] *a* σ_{forks}*-update-def* **by** *auto*
qed
qed
qed

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

proof (*induct n*, *simp*)
  case (*Suc n*)
  then show *?case*
    apply (*auto simp add*: $FORKs_{norm}$*-unfold* $FORK_{norm}$*-is-FORK*)
    apply (*metis Suc-le-D butlast-snoc replicate-Suc replicate-append-same*)
    by (*metis* $FORKs_{norm}$*-1 One-nat-def leI length-replicate less-Suc0 nth-replicate replicate-Suc*)
qed

corollary *FORKs-is-*$FORKs_{norm}$: $FORKs_{norm}$ *N* (*replicate N 0*) = *FORKs*
  using *ft N-pos* **by** *simp*

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* ∈ Tr_{rp} *i s* ⟹ *phil e* = *i*
  and *lphil0-phil*: *e* ∈ Tr_{lp} *s* ⟹ *phil e* = *0*
  by (*simp-all add*:*rphil-transitions-def lphil0-transitions-def split*:*if-splits*)

definition *rphil-state-update*:: id_{fork} ⇒ σ_{fork} ⇒ *dining-event* ⇒ σ_{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*:: σ_{fork} ⇒ *dining-event* ⇒ σ_{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} ⇒ σ_{fork} ⇒ *dining-event process*
  where $RPHIL_{norm}$ *i* = $P_{norm}[\![Tr_{rp}\ i, Up_{rp}\ i]\!]$

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

lemma $RPHIL_{norm}$*-rec*: $RPHIL_{norm}\ i = (\lambda\ s.\ \square\ e \in (Tr_{rp}\ i\ s) \rightarrow RPHIL_{norm}\ i\ (Up_{rp}\ i\ s\ e))$
using *fix-eq*[*of* Λ *X*. (λs. *Mprefix* (Tr_{rp} *i s*) (λe. *X* (Up_{rp} *i s e*)))] $RPHIL_{norm}$*-def*
by *simp*

lemma $LPHIL0_{norm}$*-rec*: $LPHIL0_{norm} = (\lambda\ s.\ \square\ e \in (Tr_{lp}\ s) \rightarrow LPHIL0_{norm}\ (Up_{lp}\ s\ e))$
using *fix-eq*[*of* Λ *X*. (λs. *Mprefix* (Tr_{lp} *s*) (λe. *X* (Up_{lp} *s e*)))] $LPHIL0_{norm}$*-def*
by *simp*

lemma *RPHIL-refines-*$RPHIL_{norm}$:
assumes *i-pos*: $i > 0$
shows $RPHIL_{norm}\ i\ 0 \sqsubseteq_{FD} RPHIL\ i$
proof(*unfold* $RPHIL_{norm}$*-def*,
induct rule:*fix-ind-k-skip*[**where** *k*=*4* **and** *f*=Λ *x*. (λs. *Mprefix* (Tr_{rp} *i s*) (λe. *x* (Up_{rp} *i s e*)))])
case *lower-bound*
then show *?case* (**is** $1 \leq 4$) **by** *simp*
next
case *admissibility*
then show *?case* (**is** *adm* ($\lambda a.\ a\ 0 \sqsubseteq_{FD} RPHIL\ i$))
by (*simp add*: *cont-fun monofunI*)
next
case *base-k-steps*
then show *?case* (**is** $\forall j<4.\ (iterate\ j\cdot ?f\cdot\bot)\ 0 \sqsubseteq_{FD} RPHIL\ i$)
proof −
have *less-2*:$\bigwedge j.\ (j::nat) < 4 = (j = 0 \lor j = 1 \lor j = 2 \lor j = 3)$ **by** *linarith*
moreover have $(iterate\ 0\cdot ?f\cdot\bot)\ 0 \sqsubseteq_{FD} RPHIL\ i$ **by** *simp*
moreover have $(iterate\ 1\cdot ?f\cdot\bot)\ 0 \sqsubseteq_{FD} RPHIL\ i$
by (*subst RPHIL-rec*) (*simp add*: *write0-def rphil-transitions-def mono-Mprefix-FD mono-Det-FD*)
moreover have $(iterate\ 2\cdot ?f\cdot\bot)\ 0 \sqsubseteq_{FD} RPHIL\ i$
by (*subst RPHIL-rec*)
(*auto simp add*: *numeral-2-eq-2 write0-def rphil-transitions-def rphil-state-update-def*
intro!: *mono-Mprefix-FD mono-Det-FD*)
moreover have $(iterate\ 3\cdot ?f\cdot\bot)\ 0 \sqsubseteq_{FD} RPHIL\ i$
by (*subst RPHIL-rec*) (*auto simp add*: *numeral-3-eq-3 write0-def rphil-transitions-def*
rphil-state-update-def minus-suc[*OF i-pos*]
intro!: *mono-Mprefix-FD mono-Det-FD*)
ultimately show *?thesis* **by** *simp*
qed
next
case (*step x*)
then show *?case* (**is** $(iterate\ 4\cdot ?f\cdot x)\ 0 \sqsubseteq_{FD} RPHIL\ i$)

apply (*subst RPHIL-rec*)
apply (*simp add*: *write0-def numeral-4-eq-4 rphil-transitions-def rphil-state-update-def*)
apply (*rule mono-Mprefix-FD*, *auto simp*:*minus-suc*[*OF i-pos*])+
using *minus-suc*[*OF i-pos*] **by** *auto*
qed

lemma *LPHIL0-refines-LPHIL0*$_{norm}$: *LPHIL0*$_{norm}$ *0* $\sqsubseteq_{FD}$ *LPHIL0*
proof(*unfold LPHIL0*$_{norm}$*-def*,
induct rule:*fix-ind-k-skip*[**where** *k*=*4* **and** *f*=Λ *x*. (λ*s*. *Mprefix* (*Tr*$_{lp}$ *s*) (λ*e*. *x* (*Up*$_{lp}$ *s e*)))])
show (*1*::*nat*) $\leq$ *4* **by** *simp*
next
show *adm* (λ*a*. *a 0* $\sqsubseteq_{FD}$ *LPHIL0*) **by** (*simp add*: *cont-fun monofunI*)
next
case *base-k-steps*
show *?case* (**is** $\forall$ *j*<*4*. (*iterate j*·*?f*·$\perp$) *0* $\sqsubseteq_{FD}$ *LPHIL0*)
proof −
have *less-2*:$\bigwedge$*j*. (*j*::*nat*) < *4* = (*j* = *0* $\vee$ *j* = *1* $\vee$ *j* = *2* $\vee$ *j* = *3*) **by** *linarith*
moreover have (*iterate 0*·*?f*·$\perp$) *0* $\sqsubseteq_{FD}$ *LPHIL0* **by** *simp*
moreover have (*iterate 1*·*?f*·$\perp$) *0* $\sqsubseteq_{FD}$ *LPHIL0*
by (*subst LPHIL0-rec*) (*simp add*: *write0-def lphil0-transitions-def mono-Mprefix-FD mono-Det-FD*)
moreover have (*iterate 2*·*?f*·$\perp$) *0* $\sqsubseteq_{FD}$ *LPHIL0*
by (*subst LPHIL0-rec*) (*auto simp add*: *numeral-2-eq-2 write0-def lphil0-transitions-def*

lphil0-state-update-def intro!: *mono-Mprefix-FD mono-Det-FD*)
moreover have (*iterate 3*·*?f*·$\perp$) *0* $\sqsubseteq_{FD}$ *LPHIL0*
by (*subst LPHIL0-rec*) (*auto simp add*: *numeral-3-eq-3 write0-def lphil0-transitions-def*

lphil0-state-update-def intro!: *mono-Mprefix-FD mono-Det-FD*)
ultimately show *?thesis* **by** *simp*
qed
next
case (*step x*)
then show *?case* (**is** (*iterate 4*·*?f*·*x*) *0* $\sqsubseteq_{FD}$ *LPHIL0*)
by (*subst LPHIL0-rec*) (*auto simp add*: *write0-def numeral-4-eq-4 lphil0-transitions-def*

lphil0-state-update-def intro!: *mono-Mprefix-FD mono-Det-FD*)
qed

lemma *RPHIL*$_{norm}$*-refines-RPHIL*:
assumes *i-pos*: *i* > *0*
shows *RPHIL i* $\sqsubseteq_{FD}$ *RPHIL*$_{norm}$ *i 0*
proof(*unfold RPHIL-def*, *induct rule*:*fix-ind-k-skip*[**where** *k*=*1*])
case *lower-bound*
then show *?case* (**is** *1* $\leq$ *1*) **by** *simp*
next
case *admissibility*

then show *?case* **by** (*simp add*: *monofunI*)
next
case *base-k-steps*
then show *?case* **by** *simp*
next
case (*step x*)**then show** *?case*
apply (*subst* $RPHIL_{norm}$*-rec*, *simp add*: *write0-def rphil-transitions-def rphil-state-update-def*)

apply (*rule mono-Mprefix-FD*, *simp*)
apply (*subst* $RPHIL_{norm}$*-rec*, *simp add*: *write0-def rphil-transitions-def rphil-state-update-def*)
apply (*rule mono-Mprefix-FD*, *simp add*:*minus-suc*[*OF i-pos*])
apply (*subst* $RPHIL_{norm}$*-rec*, *simp add*: *write0-def rphil-transitions-def rphil-state-update-def*)
apply (*rule mono-Mprefix-FD*, *simp add*:*minus-suc*[*OF i-pos*])
apply (*subst* $RPHIL_{norm}$*-rec*, *simp add*: *write0-def rphil-transitions-def rphil-state-update-def*)
apply (*rule mono-Mprefix-FD*, *simp add*:*minus-suc*[*OF i-pos*])
using *minus-suc*[*OF i-pos*] **by** *auto*
qed

lemma $LPHIL0_{norm}$*-refines-LPHIL0*: $LPHIL0 \sqsubseteq_{FD} LPHIL0_{norm}\ 0$
proof(*unfold LPHIL0-def*,
induct rule:*fix-ind-k-skip*[**where** *k=1*])
show $(1::nat) \leq 1$ **by** *simp*
next
show *adm* ($\lambda a.\ a \sqsubseteq_{FD} LPHIL0_{norm}\ 0$) **by** (*simp add*: *monofunI*)
next
case *base-k-steps*
show *?case* **by** *simp*
next
case (*step x*)
then show *?case* (**is** *iterate* $1 \cdot ?f \cdot x \sqsubseteq_{FD} LPHIL0_{norm}\ 0$)
apply (*subst* $LPHIL0_{norm}$*-rec*, *simp add*: *write0-def lphil0-transitions-def lphil0-state-update-def*)

apply (*rule mono-Mprefix-FD*, *simp*)
apply (*subst* $LPHIL0_{norm}$*-rec*, *simp add*: *write0-def lphil0-transitions-def lphil0-state-update-def*)

apply (*rule mono-Mprefix-FD*, *simp*)
apply (*subst* $LPHIL0_{norm}$*-rec*, *simp add*: *write0-def lphil0-transitions-def lphil0-state-update-def*)

apply (*rule mono-Mprefix-FD*, *simp*)
by (*subst* $LPHIL0_{norm}$*-rec*, *auto simp add*: *write0-def lphil0-transitions-def lphil0-state-update-def intro*!: *mono-Mprefix-FD mono-Det-FD*)
qed

lemma $RPHIL_{norm}$*-is-RPHIL*: $i > 0 \Longrightarrow RPHIL\ i = RPHIL_{norm}\ i\ 0$
using *RPHIL-refines-*$RPHIL_{norm}$ $RPHIL_{norm}$*-refines-RPHIL FD-antisym* **by** *blast*

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

using *LPHIL0-refines-LPHIL0*$_{norm}$ *LPHIL0*$_{norm}$*-refines-LPHIL0 FD-antisym*
by *blast*

3.2.4 The normal form for the global philosopher network

type-synonym σ_{phils} = *nat list*

definition *phils-transitions*:: *nat* ⇒ σ_{phils} ⇒ *dining-event set* (Tr_P)
where Tr_P *n ps* = Tr_{lp} (*ps*!*0*) ∪ (⋃ *i*∈{*1*..< *n*}. Tr_{rp} *i* (*ps*!*i*))

corollary *phils-phil*: *0* < *n* ⟹ *e* ∈ Tr_P *n s* ⟹ *phil e* < *n*
by (*auto simp add*:*phils-transitions-def lphil0-phil rphil-phil*)

lemma *phils-transitions-take*: *0* < *n* ⟹ Tr_P *n ps* = Tr_P *n* (*take n ps*)
by (*auto simp add*:*phils-transitions-def*)

definition σ_{phils}*-update*:: σ_{phils} ⇒ *dining-event* ⇒ σ_{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*
by (*cases e*) (*simp-all add*: σ_{phils}*-update-def lphil0-state-update-def*
rphil-state-update-def take-update-swap)

definition *PHILs*$_{norm}$:: *nat* ⇒ σ_{phils} ⇒ *dining-event process*
where *PHILs*$_{norm}$ *n* = P_{norm}⟦Tr_P *n*,Up_P⟧

lemma *PHILs*$_{norm}$*-rec*: *PHILs*$_{norm}$ *n* = (λ *ps*. □ *e* ∈ (Tr_P *n ps*) → *PHILs*$_{norm}$ *n* (Up_P *ps e*))
using *fix-eq*[*of* Λ *X*. (λ*ps*. *Mprefix* (Tr_P *n ps*) (λ*e*. *X* (Up_P *ps e*)))] *PHILs*$_{norm}$*-def*
by *simp*

lemma *PHILs*$_{norm}$*-1-dir1*: *length ps* > *0* ⟹ *PHILs*$_{norm}$ *1 ps* $\sqsubseteq_{FD}$ (*LPHIL0*$_{norm}$ (*ps*!*0*))
proof(*unfold PHILs*$_{norm}$*-def*,
induct arbitrary:*ps*
rule:*fix-ind-k*[**where** *k*=*1*
and *f*=Λ *x*. (λ*ps*. *Mprefix* (Tr_P *1 ps*) (λ*e*. *x* (Up_P *ps e*)))])
case *admissibility*
then show *?case* **by** (*simp add*: *cont-fun monofunI*)
next
case *base-k-steps*
then show *?case* **by** *simp*
next
case (*step X*)
then show *?case*
apply (*subst LPHIL0*$_{norm}$*-rec*, *simp add*:σ_{phils}*-update-def phils-transitions-def*)

proof (*intro mono-Mprefix-FD*, *goal-cases*)

```
    case (1 e)
    with 1(2) show ?case
      using 1(1)[rule-format, of ps[0 := Up_lp (ps ! 0) e]]
      by (simp add:lphil0-transitions-def split:if-splits)
  qed
qed
```

lemma *PHILs*$_{norm}$*-1-dir2*: *length ps* $> 0 \Longrightarrow$ (*LPHIL0*$_{norm}$ (*ps*!*0*)) $\sqsubseteq_{FD}$ *PHILs*$_{norm}$ *1 ps*

```
proof(unfold LPHIL0_norm-def,
    induct arbitrary:ps rule:fix-ind-k[where k=1
      and f=Λ x. (λs. Mprefix (Tr_lp s) (λe. x (Up_lp s e)))])
  case admissibility
  then show ?case by (simp add: cont-fun monofunI)
next
  case base-k-steps
  then show ?case by simp
next
  case (step X)
  then show ?case
    apply (subst PHILs_norm-rec, simp add:σ_phils-update-def phils-transitions-def)

  proof (intro mono-Mprefix-FD, goal-cases)
    case (1 e)
    with 1(2) show ?case
      using 1(1)[rule-format, of ps[0 := Up_lp (ps ! 0) e]]
      by (simp add:lphil0-transitions-def split:if-splits)
  qed
qed
```

lemma *PHILs*$_{norm}$*-1*: *length ps* $> 0 \Longrightarrow$ *PHILs*$_{norm}$ *1 ps* = (*LPHIL0*$_{norm}$ (*ps*!*0*))
using *PHILs*$_{norm}$*-1-dir1 PHILs*$_{norm}$*-1-dir2 FD-antisym* **by** *blast*

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(rule FD-antisym)
 show length ps = Suc n ⟹ PHILs_norm (Suc n) ps ⊑_FD (PHILs_norm n (butlast ps))|||RPHIL_norm n (ps!n))
  proof(subst PHILs_norm-def,
      induct arbitrary:ps
      rule:fix-ind-k[where k=1
        and f=Λ x. (λps. Mprefix (Tr_P (Suc n) ps) (λe. x (Up_P ps e)))])
    case admissibility
    then show ?case by (simp add: cont-fun monofunI)
  next
    case base-k-steps
```

```
    then show ?case by simp
  next
    case (step X)
    have indep:∀ s1 s2. TrP n s1 ∩ Trrp n s2 = {}
      using phils-phil rphil-phil n-pos by blast
    from step have tra:(TrP (Suc n) ps) =(TrP n (butlast ps) ∪ Trrp n (ps ! n))
      by (auto simp add:n-pos phils-transitions-def nth-butlast Suc-leI
          atLeastLessThanSuc Un-commute Un-assoc)
    from step show ?case
      apply (auto simp add:indep dnorm-inter PHILsnorm-def RPHILnorm-def)
      apply (subst fix-eq, auto simp add:tra)
    proof(rule mono-Mprefix-FD, safe, goal-cases)
      case (1 e)
      hence c:UpP ps e ! n = ps ! n
        using 1(3) phils-phil σphils-update-def step n-pos
        by (cases phil e, auto) (metis exists-least-iff nth-list-update-neq)
      have d:UpP (butlast ps) e = butlast (UpP ps e)
        by (cases phil e, auto simp add:σphils-update-def butlast-list-update
            lphil0-state-update-def rphil-state-update-def)
      have e:length (UpP ps e) = Suc n
        by (metis (full-types) step(2) length-list-update σphils-update-def)
      from 1 show ?case
        using 1(1)[rule-format, of (UpP ps e)] c d e by auto
    next
      case (2 e)
      have e:length (UpP ps e) = Suc n
        by (metis (full-types) step(2) length-list-update σphils-update-def)
      from 2 show ?case
        using 2(1)[rule-format, of (UpP ps e), OF e] n-pos
        apply(auto simp add: butlast-list-update rphil-phil σphils-update-def)
        by (meson disjoint-iff-not-equal indep)
    qed
  qed
next
  have indep:∀ s1 s2. TrP n s1 ∩ Trrp n s2 = {}
    using phils-phil rphil-phil using n-pos by blast

  show length ps = Suc n ⟹ (PHILsnorm n (butlast ps)|||RPHILnorm n (ps!n))
⊑FD PHILsnorm (Suc n) ps
   apply (subst PHILsnorm-def, auto simp add:indep dnorm-inter RPHILnorm-def)
  proof(rule fix-ind[where
        P=λa. ∀ x. length x = Suc n ⟶ a (butlast x, x ! n) ⊑FD PHILsnorm (Suc
n) x, rule-format],
      simp-all, goal-cases base step)
    case base
    then show ?case by (simp add: cont-fun monofunI)
  next
    case (step X ps)
    hence tra:(TrP (Suc n) ps) =(TrP n (butlast ps) ∪ Trrp n (ps ! n))
```

by (*auto simp add*:*n-pos phils-transitions-def nth-butlast*
Suc-leI atLeastLessThanSuc Un-commute Un-assoc)
from *step* **show** *?case*
apply (*auto simp add*:*indep dnorm-inter* $PHILs_{norm}$-*def* $RPHIL_{norm}$-*def*)
apply (*subst fix-eq*, *auto simp add*:*tra*)
proof(*rule mono-Mprefix-FD*, *safe*, *goal-cases*)
case (*1 e*)
hence *c*:Up_P *ps e* ! *n* = *ps* ! *n*
using *1*(*3*) *phils-phil* σ_{phils}-*update-def step n-pos*
by (*cases phil e*, *auto*) (*metis exists-least-iff nth-list-update-neq*)
have *d*:Up_P (*butlast ps*) *e* = *butlast* (Up_P *ps e*)
by (*cases phil e*, *auto simp add*:σ_{phils}-*update-def butlast-list-update*
lphil0-state-update-def rphil-state-update-def)
have *e*:*length* (Up_P *ps e*) = *Suc n*
by (*metis* (*full-types*) *step*(*3*) *length-list-update* σ_{phils}-*update-def*)
from *1* **show** *?case*
using *1*(*2*)[*rule-format*, *of* (Up_P *ps e*), *OF e*] *c d* **by** *auto*
next
case (*2 e*)
have *e*:*length* (Up_P *ps e*) = *Suc n*
by (*metis* (*full-types*) *2*(*3*) *length-list-update* σ_{phils}-*update-def*)
from *2* **show** *?case*
using *2*(*2*)[*rule-format*, *of* (Up_P *ps e*), *OF e*] *n-pos*
apply(*auto simp add*: *butlast-list-update rphil-phil* σ_{phils}-*update-def*)
by (*meson disjoint-iff-not-equal indep*)
qed
qed
qed

lemma *pt*: $0 < n \Longrightarrow PHILs_{norm}\ n\ (replicate\ n\ 0) = foldPHILs\ n$
proof (*induct n*, *simp*)
case (*Suc n*)
then show *?case*
apply (*auto simp add*: $PHILs_{norm}$-*unfold* $LPHIL0_{norm}$-*is-LPHIL0*)
apply (*metis Suc-le-eq butlast.simps*(*2*) *butlast-snoc* $RPHIL_{norm}$-*is-RPHIL*
nat-neq-iff replicate-append-same replicate-empty)
by (*metis One-nat-def leI length-replicate less-Suc0* $PHILs_{norm}$-*1 nth-Cons-0*
replicate-Suc)
qed

corollary *PHILs-is-*$PHILs_{norm}$: $PHILs_{norm}$ *N* (*replicate N 0*) = *PHILs*
using *pt N-pos* **by** *simp*

3.2.5 The complete process system under normal form

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

lemma *ltsDining-rec*: $DINING_{norm}\ n = (\lambda\ s.\ \square\ e \in (Tr_D\ n\ s) \rightarrow DINING_{norm}\ n\ (Up_D\ s\ e))$
using *fix-eq*[*of* $\Lambda\ X.\ (\lambda s.\ Mprefix\ (Tr_D\ n\ s)\ (\lambda e.\ X\ (Up_D\ s\ e)))$] $DINING_{norm}$*-def*
by *simp*

lemma *DINING-is-*$DINING_{norm}$: $DINING = DINING_{norm}\ N\ (replicate\ N\ 0,\ replicate\ N\ 0)$
proof –
have $DINING_{norm}\ N\ (replicate\ N\ 0,\ replicate\ N\ 0) = (PHILs_{norm}\ N\ (replicate\ N\ 0)\ ||\ FORKs_{norm}\ N\ (replicate\ N\ 0))$
unfolding $DINING_{norm}$*-def* $PHILs_{norm}$*-def* $FORKs_{norm}$*-def* *dining-transitions-def*

dining-state-update-def dnorm-par **by** *simp*
thus *?thesis*
using *PHILs-is-*$PHILs_{norm}$ *FORKs-is-*$FORKs_{norm}$ *DINING-def*
by (*simp add*: *Sync-commute*)
qed

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*: $Up_{rp}\ i\ r\ e = 0 \lor Up_{rp}\ i\ r\ e = 1 \lor Up_{rp}\ i\ r\ e = 2 \lor Up_{rp}\ i\ r\ e = 3$
unfolding *lphil0-state-update-def rphil-state-update-def* **by** *auto*

lemma *dining-events*:
$e \in Tr_D\ N\ s \Longrightarrow$
$(\exists\, i{\in}\{1..{<}N\}.\ e = picks\ i\ i \lor e = picks\ i\ (i{-}1) \lor e = putsdown\ i\ i \lor e = putsdown\ i\ (i{-}1))$
$\lor (e = picks\ 0\ 0 \lor e = picks\ 0\ (N{-}1) \lor e = putsdown\ 0\ 0 \lor e = putsdown\ 0\ (N{-}1))$
by (*auto simp add*:*dining-transitions-def phils-transitions-def rphil-transitions-def*

lphil0-transitions-def split:*prod.splits if-splits*)

definition *inv-dining ps fs* $\equiv$
$(\forall\, i.\ Suc\ i < N \longrightarrow ((fs!(Suc\ i) = 1) \longleftrightarrow ps!Suc\ i \neq 0)) \land (fs!(N{-}1) = 2 \longleftrightarrow ps!0 \neq 0)$
$\land (\forall\, i < N - 1.\ fs!i = 2 \longleftrightarrow ps!Suc\ i = 2) \land (fs!0 = 1 \longleftrightarrow ps!0 = 2)$
$\land (\forall\, i < N.\ fs!i = 0 \lor fs!i = 1 \lor fs!i = 2)$

$\wedge$ ($\forall$ *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-dining (*fst s*) (*snd s*)
proof(*induct rule*:$\mathfrak{R}$.*induct*)
  case *rbase*
  show *?case*
    by (*simp add*: *inv-dining-def*)
next
  case (*rstep s e*)
  from *rstep*(*2*,*3*) **show** *?case*
   apply(*auto simp add*:*dining-transitions-def phils-transitions-def forks-transitions-def*
     lphil0-transitions-def rphil-transitions-def fork-transitions-def
     lphil0-state-update-def rphil-state-update-def σ_{fork}*-update-def*
     dining-state-update-def σ_{phils}*-update-def* σ_{forks}*-update-def*
     split:*if-splits prod.split*)
    unfolding *inv-dining-def*
  proof(*goal-cases*)
    case (*1 ps fs*)
    then show *?case*
      by (*simp add*:*nth-list-update*) *force*
  next
    case (*2 ps fs*)
    then show *?case*
      by (*auto simp add*:*nth-list-update*)
  next
    case (*3 ps fs*)
    then show *?case*
      using *N-pos-simps*(*3*) **by** *force*
  next
    case (*4 ps fs*)
    then show *?case*
      by (*simp add*:*nth-list-update*) *force*
  next
    case (*5 ps fs*)
    then show *?case*
      using *N-g1* **by** *linarith*
  next
    case (*6 ps fs*)
    then show *?case*
      by (*auto simp add*:*nth-list-update*)
  next
    case (*7 ps fs i*)
    then show *?case*
      apply (*simp add*:*nth-list-update*, *intro impI conjI*, *simp-all*)
      by *auto*[*1*] (*metis N-pos Suc-pred less-antisym*, *metis zero-neq-numeral*)
  next
    case (*8 ps fs i*)

```
    then show ?case
      apply (simp add:nth-list-update, intro impI conjI allI, simp-all)
      by (metis 8(1) zero-neq-one)+
  next
    case (9 ps fs i)
    then show ?case
      apply (simp add:nth-list-update, intro impI conjI allI, simp-all)
      by (metis N-pos Suc-pred less-antisym) (metis n-not-Suc-n numeral-2-eq-2)
  next
    case (10 ps fs i)
    then show ?case
      apply (simp add:nth-list-update, intro impI conjI allI, simp-all)
      by (metis 10(1) 10(5) One-nat-def n-not-Suc-n numeral-2-eq-2)+
  qed
qed

lemma inv-implies-DF:inv-dining ps fs ⟹ Tr_D N (ps, fs) ≠ {}
  unfolding inv-dining-def
  apply(simp add:dining-transitions-def phils-transitions-def forks-transitions-def
      lphil0-transitions-def
      split: if-splits prod.splits)
proof(elim conjE, intro conjI impI, goal-cases)
  case 1
  hence putsdown 0 (N − Suc 0) ∈ (⋃i<N. Tr_f i (fs ! i))
    by (auto simp add:fork-transitions-def)
  then show ?case
    by blast
next
  case 2
  hence putsdown 0 0 ∈ (⋃i<N. Tr_f i (fs ! i))
    by (auto simp add:fork-transitions-def)
  then show ?case
    by (simp add:fork-transitions-def) force
next
  case 3
  hence a:fs!0 = 0 ⟹ picks 0 0 ∈ (⋃i<N. Tr_f i (fs ! i))
    by (auto simp add:fork-transitions-def)
  from 3 have b1:ps!1 = 2 ⟹ putsdown 1 0 ∈ (⋃x∈{Suc 0..<N}. Tr_rp x (ps
! x))
    using N-g1 by (auto simp add:rphil-transitions-def)
  from 3 have b2:fs!0 = 2 ⟹ putsdown 1 0 ∈ Tr_f 0 (fs ! 0)
    using N-g1 by (auto simp add:fork-transitions-def) fastforce
  from 3 have c:fs!0 ≠ 0 ⟹ ps!1 = 2
    by (metis N-pos N-pos-simps(3) One-nat-def diff-is-0-eq neq0-conv)
  from 3 have d:fs!0 ≠ 0 ⟹ fs!0 = 2
    using N-pos by meson
  then show ?case
    apply(cases fs!0 = 0)
    using a apply (simp add: fork-transitions-def Un-insert-left)
```

```
    using b1[OF c] b2[OF d] N-pos by blast
next
  case 4
  then show ?case
    using 4(5)[rule-format, of 0, OF N-pos] apply(elim disjE)
  proof(goal-cases)
    case 41:1
    then show ?case
      using 4(5)[rule-format, of 1, OF N-g1] apply(elim disjE)
    proof(goal-cases)
      case 411:1
      from 411 have a0: ps!1 = 0
        by (metis N-g1 One-nat-def neq0-conv)
      from 411 have a1: picks 1 1 ∈ (⋃i<N. Tr_f i (fs ! i))
        apply (auto simp add:fork-transitions-def)
        by (metis (mono-tags, lifting) N-g1 Int-Collect One-nat-def lessThan-iff)
      from 411 have a2: ps!1 = 0 ⟹ picks 1 1 ∈ (⋃i∈{Suc 0..<N}. Tr_rp i (ps
! i))
        apply (auto simp add:rphil-transitions-def)
        using N-g1 by linarith
      from 411 show ?case
        using a0 a1 a2 by blast
    next
      case 412:2
      hence ps!1 = 1 ∨ ps!1 = 3
        by (metis N-g1 One-nat-def less-numeral-extra(3) zero-less-diff)
      with 412 show ?case
      proof(elim disjE, goal-cases)
        case 4121:1
        from 4121 have b1: picks 1 0 ∈ (⋃i<N. Tr_f i (fs ! i))
          apply (auto simp add:fork-transitions-def)
            by (metis (full-types) Int-Collect N-g1 N-pos One-nat-def lessThan-iff
mod-less)
        from 4121 have b2: picks 1 0 ∈ (⋃i∈{Suc 0..<N}. Tr_rp i (ps ! i))
          apply (auto simp add:rphil-transitions-def)
          using N-g1 by linarith
        from 4121 show ?case
          using b1 b2 by blast
      next
        case 4122:2
        from 4122 have b3: putsdown 1 1 ∈ (⋃i<N. Tr_f i (fs ! i))
          apply (auto simp add:fork-transitions-def)
          using N-g1 by linarith
        from 4122 have b4: putsdown 1 1 ∈ (⋃i∈{Suc 0..<N}. Tr_rp i (ps ! i))
          apply (auto simp add:rphil-transitions-def)
          using N-g1 by linarith
        then show ?case
          using b3 b4 by blast
      qed
```

next
case *413*:*3*
then show *?case*
proof(*cases N* = *2*)
case *True*
with *413* **show** *?thesis* **by** *simp*
next
case *False*
from *False 413* **have** *c0*: *ps*!*2* = *2*
by (*metis N-g1 Suc-1 Suc-diff-1 nat-neq-iff not-gr0 zero-less-diff*)
from *False 413* **have** *c1*: $putsdown\ 2\ 1 \in (\bigcup i{<}N.\ Tr_f\ i\ (fs\ !\ i))$
apply (*auto simp add*:*fork-transitions-def*)
using *N-g1* **apply** *linarith*
using *N-g1* **by** *auto*
from *False 413* **have** *c2*: $ps!2 = 2 \Longrightarrow putsdown\ 2\ 1 \in (\bigcup i{\in}\{Suc\ 0..{<}N\}.\ Tr_{rp}\ i\ (ps\ !\ i))$
apply (*auto simp add*:*rphil-transitions-def*)
using *N-g1* **by** *linarith*
from *413 False* **show** *?thesis*
using *c0 c1 c2* **by** *blast*
qed
qed
next
case *42*:*2*
then show *?case* **by** *blast*
next
case *43*:*3*
from *43* **have** *d0*: *ps*!*1* = *2*
by (*metis One-nat-def gr0I*)
from *43* **have** *d1*: $putsdown\ 1\ 0 \in (\bigcup i{<}N.\ Tr_f\ i\ (fs\ !\ i))$
by (*auto simp add*:*fork-transitions-def*)
from *43* **have** *d2*: $ps!1 = 2 \Longrightarrow putsdown\ 1\ 0 \in (\bigcup i{\in}\{Suc\ 0..{<}N\}.\ Tr_{rp}\ i\ (ps\ !\ i))$
apply (*auto simp add*:*rphil-transitions-def*)
using *N-g1* **by** *linarith*
from *43* **show** *?case*
using *d0 d1 d2* **by** *blast*
qed
next
case *5*
then show *?case*
using *5*(*6*)[*rule-format*, *of 0*] **by** *simp*
qed

corollary *deadlock-free-DINING*: *deadlock-free DINING*
unfolding *DINING-is-DINING*$_{norm}$ *DINING*$_{norm}$*-def*
using *inv-DINING inv-implies-DF* **by** (*subst deadlock-free-dnorm*) *auto*

corollary *deadlock-free*$_{SKIPS}$*-DINING*: *deadlock-free*$_{SKIPS}$ *DINING*
by (*simp add*: *deadlock-free-DINING deadlock-free-imp-deadlock-free*$_{SKIPS}$)

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.