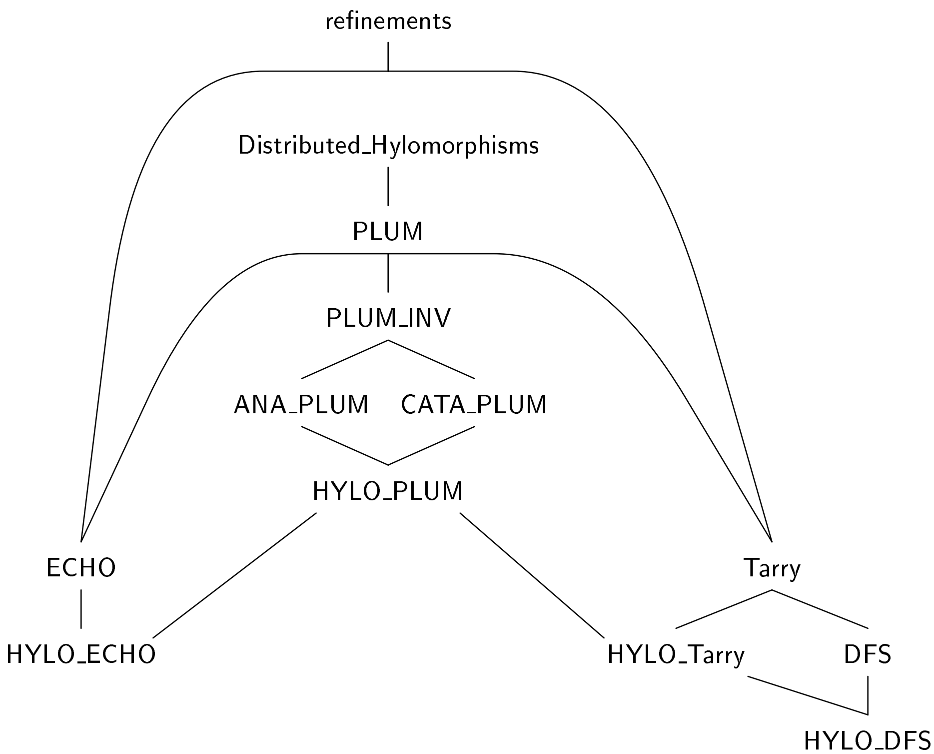
Proving distributed hylomorphisms



T. E. J. Vos and **S. D. Swierstra**
Informatica Instituut, Utrecht University
e-mail: {tanja, doaitse}@cs.uu.nl

UU-CS-2001-40

December 23, 2001

Contents

1 Introduction

This report presents detailed formal proofs of the correctness of distributed hylomorphisms with respect to their termination. The main objectives of the verification strategy are (a) to reduce proof effort and complexity by using the refinements framework from [VS01] and re-using as many results as possible, and (b) to write (or represent) comprehensible proofs by incrementally constructing invariants that are not pulled out of a hat.

2 Preliminaries, terminology and notation

Function application will be represented by a dot. In definitions we shall use $\stackrel{d}{=}$ meaning "is defined by".

The complement of a set W is denoted by W^{c}.

A relation R is *bitotal* on A and B (denoted by $\mathsf{bitotal}.R.A.B$), when for every element in A there exists at least one element on B to which it is related, and similarly for B.

A relation $\prec$ is *well-founded* over A, when it is not possible to construct an infinite sequence of decreasing values in A.

Universal quantification will be written like $(\forall x : P\ x : Q\ x)$ meaning for all x if P holds for x then also Q. If P is true for all x we just write $(\forall x :: Q\ x)$. Similar notation is used for existential quantification.

When referring to a theorem or definition we – when convenient for the reader – include the page number where the referred item can be found as a subscript.

Every definition and theorem is marked by the name it is identified with in the HOL theories that were constructed (see Section 11).

Preliminaries on states, actions, programs and specifications can be found in Appendix A.

3 A refinement relation

In [VS01] a refinement relation is formalized for UNITY programs, that defines $P \sqsubseteq Q$ to be true when program P can be refined to Q using any composition of guard strengthening and superposition program transformations. In the next two sections we will formalize this refinement relations. For a more thorough treatment the reader is referred to [VS01].

3.1 The formalisation

First we define action refinement. We say that action A_l is refined by action A_r, or A_r refines A_l, with respect to a set of variables V and a state-predicate J (denoted by $A_l \sqsubseteq_{V,J} A_r$), when:

- the conjunction of J with the guard of A_r is stronger then the guard of A_l.
- the results of A_l and A_r, both executed in the same state s where $J.s$ holds, on the variables in V are the same.

Definition 3.1 ACTION REFINEMENT *A_ref_DEF*

Let A_l and A_r be two actions from the universe `ACTION`, J be a state predicate, and V be a set of variables, then action refinement is defined as follows:

$$\begin{array}{lll} A_l \sqsubseteq_{V,J} A_r & = & \forall s\ ::\ \mathsf{guard\_of}.A_r.s \wedge J.s \Rightarrow \mathsf{guard\_of}.A_l.s \\ & & \wedge \\ & & \forall s,t,t'\ ::\ (\mathsf{compile}.A_l.s.t \wedge \mathsf{compile}.A_r.s.t' \wedge \mathsf{guard\_of}.A_r.s \wedge J.s) \Rightarrow t =_V t' \end{array}$$

Next, we define our relation of program refinement. P is refined by Q, or Q refines P, with respect to some relation $\mathcal{R}$ and state-predicate J, (denoted by $P \sqsubseteq_{\mathcal{R},J} Q$), if we can decompose the actions of program Q into $\mathbf{a}Q_1$ and $\mathbf{a}Q_2$, such that

- $\mathcal{R}$ is a bitotal relation on the two sets of actions $\mathbf{a}P$ and $\mathbf{a}Q_1$, i.e. for every action A_P in $\mathbf{a}P$ there exists at least one action in $\mathbf{a}Q_1$ to which $\mathbf{a}P$ is related by $\mathcal{R}$, and similarly for every action A_Q in $\mathbf{a}Q_1$ there exists at least one action in $\mathbf{a}P$ to which A_Q is related by $\mathcal{R}$.

- for all actions A_P of $\mathbf{a}P$ and A_Q of $\mathbf{a}Q_1$ that are related to each other by $\mathcal{R}$ (i.e. $A_P \; \mathcal{R} \; A_Q$ holds), we can prove that A_Q refines A_P with respect to the write variables of P and state-predicate J.

- the actions of Q that are in $\mathbf{a}Q_2$ refine skip with respect to the write variables of P and J.

Definition 3.2 PROGRAM REFINEMENT *P_ref_DEF*

Let P and Q be two UNITY programs, $\mathcal{R}$ be a relation, and J be a state predicate, then program refinement is defined as follows:

$$\begin{array}{lll} P \sqsubseteq_{\mathcal{R},J} Q & = \exists \mathbf{a}Q_1, \mathbf{a}Q_2 :: & \mathbf{a}Q = \mathbf{a}Q_1 \cup \mathbf{a}Q_2 \; \wedge \; \mathsf{bitotal}.\mathcal{R}.\mathbf{a}P.\mathbf{a}Q_1 \\ & & \wedge \\ & & \forall A_P \; A_Q \; : A_P \in \mathbf{a}P \; \wedge \; A_P \; \mathcal{R} \; A_Q \; : A_P \sqsubseteq_{\mathbf{w}P,J} A_Q \\ & & \wedge \\ & & \forall A_Q \; : A_Q \in \mathbf{a}Q_2 \; : \mathsf{skip} \sqsubseteq_{\mathbf{w}P,J} A_Q \end{array}$$

Note that $P \sqsubseteq_{\mathcal{R},J} Q$ does not say anything about Q inheriting properties or correctness from P. Nor does it say anything about the explicit program transformations that were (or could have been) applied to P in order to obtain Q.

3.2 Property preservation

Safety properties p unless q and $\circlearrowright p$, where p and q do not depend on the values of any superposed variables, are always preserved under refinement of two UNITY programs.

Theorem 3.7 unless PRESERVATION *P_ref_AND_SUPERPOSE_WRITE_PRESERVES_UNLESSe*

$$\frac{\begin{array}{c} P \sqsubseteq_{\mathcal{R},J} Q \wedge \mathsf{Unity}.P \wedge \mathsf{Unity}.Q \wedge (\;_Q\vdash \circlearrowright J_Q) \wedge (J_Q \Rightarrow J) \\ \exists W :: (\mathbf{w}Q = \mathbf{w}P \cup W) \wedge (p \; \mathcal{C} \; W^{\mathsf{c}}) \wedge (q \; \mathcal{C} \; W^{\mathsf{c}}) \end{array}}{_P\vdash p \; \mathsf{unless} \; q \Rightarrow \;_Q\vdash (J_Q \wedge p) \; \mathsf{unless} \; q}$$

Theorem 3.8 $\circlearrowright$ PRESERVATION *P_ref_AND_SUPERPOSE_WRITE_PRESERVES_STABLEe*

$$\frac{\begin{array}{c} P \sqsubseteq_{\mathcal{R},J} Q \wedge \mathsf{Unity}.P \wedge \mathsf{Unity}.Q \wedge (\;_Q\vdash \circlearrowright J_Q) \wedge (J_Q \Rightarrow J) \\ \exists W :: (\mathbf{w}Q = \mathbf{w}P \cup W) \wedge (p \; \mathcal{C} \; W^{\mathsf{c}}) \end{array}}{_P\vdash \circlearrowright p \Rightarrow \;_Q\vdash \circlearrowright (J_Q \wedge p)}$$

Progress properties $p \rightarrowtail q$ and $p \rightsquigarrow q$ are preserved under certain verification conditions stated in the theorems in Figure 1. Theorem 3.3 is the most general theorem, the other three are corollaries. Note that the Theorems in Figure 1 state property preservation in refinements independently from the specific program transformations that were applied. To read more about these theorems the reader is referred to [VS01].

4 The communication network

The communication networks are assumed to be connected centralised communication networks employing bi-directional asynchronous communication.

4.1 Centralised

A *centralised communication network* is modelled by the tuple ($\mathbb{P}$, neighs, $\mathsf{starter}$), where

$\mathbb{P}$ is a finite set of processes. Since we are talking about networks of processes, we assume that $\mathbb{P}$ at least has two processes.

neighs is a function that given some process $p \in \mathbb{P}$, gives the set of neighbors of p. In other words, for $p \in \mathbb{P}$, $\mathsf{neighs}.p$ is the set of processes that are connected to p by a bi-directional communication link. Obviously, the function neighs should satisfy: $\forall p \in \mathbb{P} : \; \mathsf{neighs}.p \subseteq \mathbb{P}$. We will only consider communication between distinct processes and not allow self-loops, thus neighs must also satisfy: $\forall p \in \mathbb{P}, \; q \in \mathsf{neighs}.p \; : p \neq q$. Since communication is bi-directional it holds that: $\forall p, q \in \mathbb{P} : (q \in \mathsf{neighs}.p) = (p \in \mathsf{neighs}.q)$.

Let $\prec$ be a well-founded relation over some set A, $M \in \texttt{State}{\rightarrow}A$, and P and Q be UNITY programs.

Theorem 3.3 *P_ref_SUPERPOSE_AND_WF_FUNC_PRESERVES_REACHe_GEN*
P_ref_SUPERPOSE_AND_WF_FUNC_PRESERVES_CONe_GEN

$$\frac{\begin{array}{c} P \sqsubseteq_{R,J} Q \wedge ({}_Q{\vdash}\circlearrowleft J_P \wedge J_Q) \wedge (J_P \wedge J_Q \Rightarrow J) \\ \exists W :: (\mathbf{w}Q = \mathbf{w}P \cup W) \wedge (J_P\ \mathcal{C}\ W^{\mathsf{c}}) \wedge (\mathbf{w}P \subseteq W^{\mathsf{c}}) \\ \forall A_Q : A_Q \in \mathbf{a}Q \wedge (\exists A_P :: (A_P \in \mathbf{a}P) \wedge (A_P\ \mathcal{R}\ A_Q)) : (\mathsf{guard\_of}.A_Q\ \mathcal{C}\ \mathbf{w}Q) \\ \forall A_P : A_P \in \mathbf{a}P : (J_P \wedge J_Q)\ {}_Q{\vdash}\ \mathsf{guard\_of}.A_P \rightarrowtail (\exists A_Q :: (A_P\ \mathcal{R}\ A_Q) \wedge \mathsf{guard\_of}.A_Q) \\ \exists M :: (M\ \mathcal{C}\ \mathbf{w}Q) \wedge (\forall k : k \in A :\ {}_Q{\vdash}\ (J_P \wedge J_Q \wedge M = k)\ \mathsf{unless}\ (M \prec k)) \\ \wedge \forall k\ A_P A_Q : k \in A \wedge A_P \in \mathbf{a}P \wedge A_P\ \mathcal{R}\ A_Q : \\ {}_Q{\vdash}\ (J_P \wedge J_Q \wedge \mathsf{guard\_of}.A_Q \wedge M = k)\ \mathsf{unless}\ (\neg(\mathsf{guard\_of}.A_P) \vee M \prec k) \end{array}}{((J_P\ {}_P{\vdash}\ p \rightarrowtail q) \Rightarrow (J_P \wedge J_Q\ {}_Q{\vdash}\ p \rightarrowtail q))\ \wedge\ ((J_P\ {}_P{\vdash}\ p \leadsto q) \Rightarrow (J_P \wedge J_Q\ {}_Q{\vdash}\ p \leadsto q))}$$

Theorem 3.4 *P_ref_SUPERPOSE_PRESERVES_REACHe_GEN*
P_ref_SUPERPOSE_PRESERVES_CONe_GEN

$$\frac{\begin{array}{c} P \sqsubseteq_{R,J} Q \wedge ({}_Q{\vdash}\circlearrowleft J_P \wedge J_Q) \wedge (J_P \wedge J_Q \Rightarrow J) \\ \exists W :: (\mathbf{w}Q = \mathbf{w}P \cup W) \wedge (J_P\ \mathcal{C}\ W^{\mathsf{c}}) \wedge (\mathbf{w}P \subseteq W^{\mathsf{c}}) \\ \forall A_Q : A_Q \in \mathbf{a}Q \wedge (\exists A_P :: (A_P \in \mathbf{a}P) \wedge (A_P\ \mathcal{R}\ A_Q)) : (\mathsf{guard\_of}.A_Q\ \mathcal{C}\ \mathbf{w}Q) \\ \forall A_P : A_P \in \mathbf{a}P : (J_P \wedge J_Q)\ {}_Q{\vdash}\ \mathsf{guard\_of}.A_P \rightarrowtail (\exists A_Q :: (A_P\ \mathcal{R}\ A_Q) \wedge \mathsf{guard\_of}.A_Q) \\ \forall A_P\ A_Q : A_P \in \mathbf{a}P \wedge A_P\ \mathcal{R}\ A_Q :\ {}_Q{\vdash}\ (J_P \wedge J_Q \wedge \mathsf{guard\_of}.A_Q)\ \mathsf{unless}\ \neg(\mathsf{guard\_of}.A_P) \end{array}}{((J_P\ {}_P{\vdash}\ p \rightarrowtail q) \Rightarrow (J_P \wedge J_Q\ {}_Q{\vdash}\ p \rightarrowtail q))\ \wedge\ ((J_P\ {}_P{\vdash}\ p \leadsto q) \Rightarrow (J_P \wedge J_Q\ {}_Q{\vdash}\ p \leadsto q))}$$

Theorem 3.5 *P_ref_SUPERPOSE_AND_WF_FUNC_PRESERVES_REACHe*
P_ref_SUPERPOSE_AND_WF_FUNC_PRESERVES_CONe

$$\frac{\begin{array}{c} P \sqsubseteq_{R,J} Q \wedge ({}_Q{\vdash}\circlearrowleft J_P \wedge J_Q) \wedge (J_P \wedge J_Q \Rightarrow J) \\ \exists W :: (\mathbf{w}Q = \mathbf{w}P \cup W) \wedge (J_P\ \mathcal{C}\ W^{\mathsf{c}}) \wedge (\mathbf{w}P \subseteq W^{\mathsf{c}}) \\ \forall A_P\ A_Q\ :\ A_P \in \mathbf{a}P\ \wedge\ A_P\ \mathcal{R}\ A_Q\ : (J_P \wedge J_Q)\ {}_Q{\vdash}\ \mathsf{guard\_of}.A_P \rightarrowtail \mathsf{guard\_of}.A_Q \\ \exists M :: (M\ \mathcal{C}\ \mathbf{w}Q) \wedge (\forall k : k \in A :\ {}_Q{\vdash}\ (J_P \wedge J_Q \wedge M = k)\ \mathsf{unless}\ (M \prec k)) \\ \wedge \forall k\ A_P A_Q : k \in A \wedge A_P \in \mathbf{a}P \wedge A_P\ \mathcal{R}\ A_Q : \\ {}_Q{\vdash}\ (J_P \wedge J_Q \wedge \mathsf{guard\_of}.A_Q \wedge M = k)\ \mathsf{unless}\ (\neg(\mathsf{guard\_of}.A_P) \vee M \prec k) \end{array}}{((J_P\ {}_P{\vdash}\ p \rightarrowtail q) \Rightarrow (J_P \wedge J_Q\ {}_Q{\vdash}\ p \rightarrowtail q))\ \wedge\ ((J_P\ {}_P{\vdash}\ p \leadsto q) \Rightarrow (J_P \wedge J_Q\ {}_Q{\vdash}\ p \leadsto q))}$$

Theorem 3.6 *P_ref_AND_SUPERPOSE_WRITE_PRESERVES_REACHe*
P_ref_AND_SUPERPOSE_WRITE_PRESERVES_CONe

$$\frac{\begin{array}{c} P \sqsubseteq_{R,J} Q \wedge ({}_Q{\vdash}\circlearrowleft J_P \wedge J_Q) \wedge (J_P \wedge J_Q \Rightarrow J) \\ \exists W :: (\mathbf{w}Q = \mathbf{w}P \cup W) \wedge (J_P\ \mathcal{C}\ W^{\mathsf{c}}) \wedge (\mathbf{w}P \subseteq W^{\mathsf{c}}) \\ \forall A_P\ A_Q\ :\ A_P \in \mathbf{a}P\ \wedge\ A_P\ \mathcal{R}\ A_Q\ : (J_P \wedge J_Q)\ {}_Q{\vdash}\ \mathsf{guard\_of}.A_P \rightarrowtail \mathsf{guard\_of}.A_Q \\ \forall A_P\ A_Q\ :\ A_P \in \mathbf{a}P\ \wedge\ A_P\ \mathcal{R}\ A_Q\ :\ {}_Q{\vdash}\ (J_P \wedge J_Q \wedge \mathsf{guard\_of}.A_Q)\ \mathsf{unless}\ \neg(\mathsf{guard\_of}.A_P) \end{array}}{((J_P\ {}_P{\vdash}\ p \rightarrowtail q) \Rightarrow (J_P \wedge J_Q\ {}_Q{\vdash}\ p \rightarrowtail q))\ \wedge\ ((J_P\ {}_P{\vdash}\ p \leadsto q) \Rightarrow (J_P \wedge J_Q\ {}_Q{\vdash}\ p \leadsto q))}$$

Figure 1: Preservation of $\rightarrowtail$ and $\leadsto$ properties.

starter is a process in $\mathbb{P}$ that distinguishes itself from all other processes (called the *followers*), in that it can spontaneously start the execution of its local algorithm (e.g. because it is triggered by some internal event). The *followers* can only start execution of their local algorithm after they have received a first message from some neighbour.

Definition 4.1 CENTRALISED COMMUNICATION NETWORK *Network_DEF*

$$\begin{array}{lcl}\mathsf{Network}.\mathbb{P}.\mathsf{neighs}.\mathsf{starter} & = & \mathsf{FINITE}.\mathbb{P} \wedge \mathsf{card}.\mathbb{P} > 1 \\ & \wedge & \mathsf{starter} \in \mathbb{P} \\ & \wedge & \forall p \in \mathbb{P} : \mathsf{neighs}.p \subseteq \mathbb{P} \\ & \wedge & \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p \ : p \neq q \\ & \wedge & \forall p, q \in \mathbb{P} \ : (q \in \mathsf{neighs}.p) = (p \in \mathsf{neighs}.q)\end{array}$$

4.2 Connected

A *connected network* is a network in which every pair of processes is connected by a path of communication links. Let us define the set of processes that are reachable from processes in a set S by following at most one communication link:

Definition 4.2 ACCUMULATE NEIGHBOURS *Neighs_DEF*

$$\mathsf{Neighs}.\mathsf{neighs}.S = \{q \mid \exists p \ :: \ p \in S \wedge q \in \mathsf{neighs}.p\} \cup S$$

If, for any $p \in \mathbb{P}$, there exists a number n such that the n-fold iterated application of the function Neighs.neighs on $\{p\}$ returns $\mathbb{P}$, then we can conclude that every pair of processes in $\mathbb{P}$ is connected by a path of communication links. Consequently, since starter $\in \mathbb{P}$, the following is a valid definition of connected networks:

Definition 4.3 CONNECTED NETWORK *Connected_Network*

$$\begin{array}{lcl}\mathsf{Connected\_Network}.\mathbb{P}.\mathsf{neighs}.\mathsf{starter} & = & \mathsf{Network}.\mathbb{P}.\mathsf{neighs}.\mathsf{starter} \\ & & \wedge \exists n \ :: \ \mathbb{P} = \mathsf{iterate}.n.(\mathsf{Neighs}.\mathsf{neighs}).\{\mathsf{starter}\}\end{array}$$

Since we only consider communication networks that have at least two processes we have the following property of connected networks:

Theorem 4.4 *Connected_Network_IMP_EXISTS_neigh*

$$\frac{\mathsf{Connected\_Network}.\mathbb{P}.\mathsf{neighs}.\mathsf{starter} \ \wedge \ p \in \mathbb{P}}{\exists q \ :: \ q \in \mathsf{neighs}.p}$$

4.3 Bi-directional asynchronous communication

The type of communication employed in a communication network is assumed to be asynchronous, i.e. send and receive operations work on buffered channels. To model asynchronous communication each algorithm on a communication network Network.$\mathbb{P}$.neighs.starter should have the following variables:

- nr_rec.$p.q$ that indicate the number of messages p has received from q via directed link (q, p).
- nr_sent.$p.q$ that indicate the number of messages p has sent to q via directed link (p, q).
- M.$p.q$ that represent the buffers that store messages in transit from p to q.

So if nr_rec, nr_sent, M are functions of type $\in \mathbb{P} \rightarrow \mathbb{P} \rightarrow \mathtt{Var}$, every algorithm needs the following variables:

Definition 4.5 *ASYNC_Vars*

ASYNC_Vars.$\mathbb{P}$.neighs

$= \{\mathsf{nr\_rec}.p.q \mid p \in \mathbb{P} \wedge q \in \mathsf{neighs}.p\} \cup \{\mathsf{nr\_sent}.p.q \mid p \in \mathbb{P} \wedge q \in \mathsf{neighs}.p\} \cup \{\mathsf{M}.p.q \mid p \in \mathbb{P} \wedge q \in \mathsf{neighs}.p\}$

Moreover, all algorithms should incorporate the following initial condition for these variables:

prog PLUM and ECHO

init $(\forall p \in \mathbb{P} : (p = starter) \neq (\mathsf{idle}.p)) \wedge (\mathsf{father}.starter = starter) \wedge init_\Pi$

assign

$[]_{q \in \mathsf{neighs}.p}$ **if** $\mathsf{idle}.p \wedge \mathsf{mit}.q.p$

then $\mathsf{receive}.p.q.\langle \mathbf{mes} \rangle \parallel \mathsf{father}.p := q \parallel \mathsf{idle}.p := \mathsf{false}$ (IDLE)

$[]$

$[]_{q \in \mathsf{neighs}.p}$ **if** $\neg\ \mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge collecting_\Pi.p$

then $\mathsf{receive}.p.q.\langle \mathbf{mes} \rangle$ (COL)

$[]$

$[]_{q \in \mathsf{neighs}.p}$ **if** $\neg\ \mathsf{idle}.p \wedge can\_propagate.p.q \wedge propagating_\Pi.p$

then $\mathsf{send}.p.q.\langle \mathbf{mes} \rangle$ (PROP)

$[]$

if $finished\_collecting\_and\_propagating.p \wedge \neg reported\_to\_father.p$

then $\mathsf{send}.p.(\mathsf{father}.p).\langle \mathbf{mes} \rangle$ (DONE)

Figure 2: The the local algorithm of process $p \in \mathbb{P}$ for $\Pi \in \{$PLUM, ECHO$\}$.

Definition 4.6 INITIALISE THE COMMUNICATION VARIABLES *ASYNC_Init*
$\mathsf{ASYNC\_Init}.\mathbb{P}.\mathsf{neighs}.s = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :: s.(\mathsf{nr\_rec}.p.q) = 0\ s.(\mathsf{nr\_sent}.p.q) = 0\ s.(\mathsf{M}.p.q) = []$

For this report it is sufficient to just state the functionality of the primitives (send, receive) and some additional operations (mit, nr_sent_to and nr_rec_from):

- $\mathsf{send}.p.q.m$ implements that a process p sends message m to q;
- $\mathsf{receive}.p.q.f.v$ makes sure that if there is a message in transit from q to p, process p receives a message from q, and the value of the received message is assigned to variable v after function f has been applied to it;
- $\mathsf{mit}.p.q$ the name is an acronym for message in transit, can be used to check for a message in transit from p to q;
- p nr_sent_to q enables processes to check how many messages they have already sent to a neighbour q (i.e. returns the value of variable $\mathsf{nr\_sent}.p.q$)
- p nr_rec_from q enables processes to check how many messages they have already received from a neighbour q (i.e. returns the value of variable $\mathsf{nr\_rec}.p.q$)

5 Distributed hylomorphisms

The class of distributed hylomorphisms from [Vos00] consists of 4 algorithms: PLUM, ECHO, TARRY and DFS. They are displayed in Figures 2 until 4 respectively. All four algorithms build a rooted spanning tree (using the father variable) in the connected network of processes and use this tree to let the required information (e.g. the values of which the sum has to be computed, or the feedback of the information

prog TARRY

init $(\forall p \in \mathbb{P} : (p = starter) \neq (\mathsf{idle}.p)) \wedge (\mathsf{father}.starter = starter)$
$\wedge \boxed{\forall p \in \mathbb{P} : (p = starter) \neq (\neg \mathsf{le\_rec}.p)}$

assign

$[]_{q \in \mathsf{neighs}.p}$ **if** $\mathsf{idle}.p \wedge \mathsf{mit}.q.p$

then $\mathsf{receive}.p.q.\langle \mathbf{mes} \rangle \parallel \mathsf{father}.p := q \parallel \mathsf{idle}.p := \mathsf{false} \parallel \boxed{\mathsf{le\_rec}.p := \mathsf{true}}$ (IDLE)

$[]$

$[]_{q \in \mathsf{neighs}.p}$ **if** $\neg\, \mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \boxed{collecting_{\text{TARRY}}.p}$

then $\mathsf{receive}.p.q.\langle \mathbf{mes} \rangle \parallel \boxed{\mathsf{le\_rec}.p := \mathsf{true}}$ (COL)

$[]$

$[]_{q \in \mathsf{neighs}.p}$ **if** $\neg\, \mathsf{idle}.p \wedge can\_propagate.p.q \wedge \boxed{propagating_{\text{TARRY}}.p}$

then $\mathsf{send}.p.q.\langle \mathbf{mes} \rangle \parallel \boxed{\mathsf{le\_rec}.p := \mathsf{false}}$ (PROP)

$[]$

if $finished\_collecting\_and\_propagating.p \wedge \neg reported\_to\_father.p$

then $\mathsf{send}.p.(\mathsf{father}.p).\langle \mathbf{mes} \rangle \parallel \boxed{\mathsf{le\_rec}.p := \mathsf{false}}$ (DONE)

Figure 3: The local algorithm of process $p \in \mathbb{P}$ of the TARRY algorithm.

that has to be propagated through the network) flow from the leaves to the root of the spanning tree. The similarities of the algorithms are captured by the characterisation of the following predicates:

$$rec\_from\_all\_neighs.p = \forall q \in \mathsf{neighs}.p \ : \mathsf{nr\_rec}.p.q = 1 \tag{1}$$

$$sent\_to\_all\_non\_fathers.p = \forall q \in \mathsf{neighs}.p : (q \neq \mathsf{father}.p) \Rightarrow (\mathsf{nr\_sent}.p.q = 1) \tag{2}$$

$$can\_propagate.p.q = (\mathsf{nr\_sent}.p.q = 0) \wedge (q \neq \mathsf{father}.p) \tag{3}$$

$$finished\_collecting\_and\_propagating.p = rec\_from\_all\_neighs.p \ \wedge \ sent\_to\_all\_non\_fathers.p \tag{4}$$

$$reported\_to\_father.p = (\mathsf{nr\_sent}.p.(\mathsf{father}.p) = 1) \tag{5}$$

$$sent\_to\_all\_neighs.p = \forall q \in \mathsf{neighs}.p \ : \mathsf{nr\_sent}.p.q = 1 \tag{6}$$

$$done.p = rec\_from\_all\_neighs.p \wedge sent\_to\_all\_neighs.p \tag{7}$$

The differences between the algorithms are in the communication protocols, i.e. when they are allowed to collect messages and propagate them.

5.1 The PLUM algorithm

The PLUM algorithm allows a process to freely merge its propagating and collecting actions as long as it has not yet received messages from all its neighbours, and it has not yet sent to all its neighbours that are not its father. Consequently:

$$propagating_{\text{PLUM}}.p = \neg\ sent\_to\_all\_non\_fathers.p \tag{8}$$

$$collecting_{\text{PLUM}}.p = \neg\ rec\_from\_all\_neighs.p \tag{9}$$

prog DFS

init $\forall p \in \mathbb{P} : (p = starter) \neq (\mathsf{idle}.p) \wedge (\mathsf{father}.starter = starter)$
$\wedge\ \forall p \in \mathbb{P} : (p = starter) \neq (\neg \mathsf{le\_rec}.p)$

assign

$[]_{q \in \mathsf{neighs}.p}$ **if** $\mathsf{idle}.p \wedge \mathsf{mit}.q.p$ (IDLE)

then $\mathsf{receive}.p.q.\langle \mathbf{mes} \rangle \parallel \mathsf{father}.p{:=}q \parallel \mathsf{idle}.p{:=}\mathsf{false} \parallel \mathsf{le\_rec}.p{:=}\mathsf{true} \parallel \boxed{\mathsf{lp\_rec}.p{:=}q}$

$[]$

$[]_{q \in \mathsf{neighs}.p}$ **if** $\neg\ \mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \boxed{collecting_{\text{DFS}}.p}$ (COL)

then $\mathsf{receive}.p.q.\langle \mathbf{mes} \rangle \parallel \mathsf{le\_rec}.p{:=}\mathsf{true} \parallel \boxed{\mathsf{lp\_rec}.p{:=}q}$

$[]$

$[]_{q \in \mathsf{neighs}.p}$ **if** $\neg\ \mathsf{idle}.p \wedge can\_propagate.p.q \wedge \boxed{propagating_{\text{DFS}}.p} \wedge \boxed{q = \mathsf{lp\_rec}.p}$ (PROP_LP_REC)

then $\mathsf{send}.p.q.\langle \mathbf{mes} \rangle \parallel \mathsf{le\_rec}.p{:=}\mathsf{false}$

$[]$

$[]_{q \in \mathsf{neighs}.p}$ **if** $\neg\ \mathsf{idle}.p \wedge can\_propagate.p.q \wedge \boxed{propagating_{\text{DFS}}.p} \wedge \boxed{\neg(can\_propagate.p.(\mathsf{lp\_rec}.p))}$ (PROP_NOT_LP_REC)

then $\mathsf{send}.p.q.\langle \mathbf{mes} \rangle \parallel \mathsf{le\_rec}.p{:=}\mathsf{false}$

$[]$

if $finished\_collecting\_and\_propagating.p \wedge \neg reported\_to\_father.p$ (DONE)

then $\mathsf{send}.p.(\mathsf{father}.p).\langle \mathbf{mes} \rangle \parallel \mathsf{le\_rec}.p{:=}\mathsf{false}$

Figure 4: The local algorithm of process $p \in \mathbb{P}$ of the DFS algorithm.

5.2 The ECHO algorithm

In the ECHO algorithm, a non-*idle* process p can only receive a message, after p has sent messages to all its non-father-neighbours. So, the *propagating* activities must be completed before starting *collecting* from non-father-neighbours. Consequently:

$$propagating_{\text{ECHO}}.p = \neg\ sent\_to\_all\_non\_fathers.p \tag{10}$$

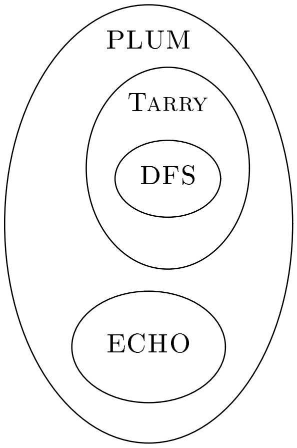
$$collecting_{\text{ECHO}}.p = \neg\ rec\_from\_all\_neighs.p\ \wedge\ \neg propagating_{\text{ECHO}}.p \tag{11}$$

5.3 The TARRY algorithm

In the TARRY algorithm, a non-*idle* process p can only propagate to a neighbour if the last event of p was a receive event; otherwise it has to wait until it receives something. So, the *propagating* and *collecting* activities alternate. From Figure 3 we can see that a boolean-typed variable $\mathsf{le\_rec}.p$ (i.e. last event was a receive) has been introduced for every process p. The assignments ($\mathsf{le\_rec}.p := \mathsf{true}$) and ($\mathsf{le\_rec}.p := \mathsf{false}$) in the **then** clauses of (COL) and (PROP) respectively, guarantee that the the value of $\mathsf{le\_rec}.p$ indicates whether the last event of p was a receive event. Consequently, we characterise the *collecting* and *propagating* predicates as follows:

$$propagating_{\text{TARRY}}.p = \neg\ sent\_to\_all\_non\_fathers.p\ \wedge\ (\mathsf{le\_rec}.p) \tag{12}$$

$$collecting_{\text{TARRY}}.p = \neg\ rec\_from\_all\_neighs.p\ \wedge\ \neg(\mathsf{le\_rec}.p) \tag{13}$$



$\forall A \in \{\text{IDLE, COL, PROP, DONE}\},\ p \in \mathbb{P},\ q \in \mathsf{neighs}.p$

$$\mathcal{R}\_\text{PLUM\_ECHO}.(A_{\text{PLUM}}.p.q).(A_{\text{ECHO}}.p.q)$$
$$\mathcal{R}\_\text{PLUM\_TARRY}.(A_{\text{PLUM}}.p.q).(A_{\text{TARRY}}.p.q)$$
$$\mathcal{R}\_\text{TARRY\_DFS}.(A_{\text{TARRY}}.p.q).(A_{\text{DFS}}.p.q)$$

(a) (b)

Figure 5: (a) refinement relation on PLUM, ECHO, TARRY, and DFS. (b) bitotal relations

5.4 The DFS algorithm

The characterisation of the *propagating* and *collecting* predicates for the DFS algorithm are identical to those of TARRY. The difference with TARRY is in the lesser freedom to choose a neighbour to send a message to in the propagating phase (see Figure 4). More specifically, for a non-idle process p in its propagating phase (i.e. there are still non-father-neighbours to which p has not yet sent) whose last event was receiving a message from some neighbour q: *if* p can propagate a message back to q, i.e. q is not p's father, and p has not yet sent to q, *then* p has to send a message back to this process q, *otherwise* it can act like in TARRY, and just pick any non-father-neighbour to which it has not yet sent a message (i.e. to which it can propagate). In order to be able to formalise and check these conditions each process in the DFS algorithm, remembers the identity of the sender of its last incoming message in the variable $\mathsf{lp\_rec}.p$ (last process of which p has received a message).

$$\textit{propagating}_{\text{DFS}}.p = \textit{propagating}_{\text{TARRY}}.p \tag{14}$$

$$\textit{collecting}_{\text{DFS}}.p = \textit{collecting}_{\text{TARRY}}.p \tag{15}$$

5.5 A refinement ordering on the distributed hylomorphisms

The algorithms in Figure 2 until 4 are ordered by our refinement relation as is visualised with venn-diagrams in Figure 5(a). The bitotal relations, with respect to which the different refinements are proved, are listed in Figure 5(b). Their definitions are straightforward, in that they relate all IDLE, COL, PROP and DONE actions of the original program to the corresponding actions in the refinement. For the relation between TARRY and DFS this results in $\text{PROP}_{\text{Tarry}}.p.q$ being related to both $\text{PROP\_LP\_REC}.p.q$ and $\text{PROP\_NOT\_LP\_REC}.p.q$. Although tedious, proving the bitotality of these relations and subsequently verifying the refinement ordering depicted in Figure 5 is reasonably easy. The resulting refinement theorems are listed below.

Theorem 5.1 *PLUM_refines_ECHO*

$$\forall J :: \text{PLUM} \sqsubseteq_{\mathcal{R}\_\text{PLUM\_ECHO},\, J} \text{ECHO}$$

Theorem 5.2 *PLUM_refines_Tarry*

$$\forall J :: \text{PLUM} \sqsubseteq_{\mathcal{R}\_\text{PLUM\_TARRY},\, J} \text{TARRY}$$

Theorem 5.3 *Tarry_refines_DFS*

$$\forall J :: \text{TARRY} \sqsubseteq_{\mathcal{R}\_\text{TARRY\_DFS},\, J} \text{DFS}$$

Theorem 6.2 VARIABLES IGNORED BY IDLE *Vars_IG_BY_IDLE*

$\{\mathsf{idle}.p, \mathsf{father}.p, \mathsf{M}.q.p, \mathsf{nr\_rec}.p.q, \mathsf{V}.p\}^{\mathsf{c}} \nleftarrow \textsc{idle}.p.q$

Theorem 6.3 VARIABLES IGNORED BY COL *Vars_IG_BY_COL*

$\{\mathsf{M}.q.p, \mathsf{nr\_rec}.p.q, \mathsf{V}.p\}^{\mathsf{c}} \nleftarrow \textsc{col}.p.q$

Theorem 6.4 VARIABLES IGNORED BY PROP *Vars_IG_BY_PROP*

$\{\mathsf{M}.p.q, \mathsf{nr\_sent}.p.q\}^{\mathsf{c}} \nleftarrow \textsc{prop}.p.q$

Theorem 6.5 VARIABLES IGNORED BY DONE *Vars_IG_BY_DONE*

$\{\mathsf{M}.p.q, \mathsf{nr\_sent}.p.q\}^{\mathsf{c}} \nleftarrow \textsc{done}.p.q$

Figure 6: Variables ignored by the actions from PLUM

Theorem 6.6 *guard_of_IDLE*

$\mathsf{guard\_of}.(\textsc{idle}.p.q) = \mathsf{idle}.p \wedge \mathsf{mit}.q.p$

Theorem 6.7 *guard_of_COL*

$\mathsf{guard\_of}.(\textsc{col}.p.q) = \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p$

Theorem 6.8 *guard_of_PROP*

$\mathsf{guard\_of}.(\textsc{prop}.p.q)$
$= \neg(\mathsf{idle}.p) \wedge (\mathsf{nr\_sent}.p.q = 0) \wedge (q \neq (\mathsf{father}.p)) \wedge \neg \mathit{sent\_to\_all\_non\_fathers}.p$

Theorem 6.9 *guard_of_DONE*

$\mathsf{guard\_of}.(\textsc{done}.p.q)$
$= \mathit{finished\_collecting\_and\_propagating}.p \wedge \neg \mathit{reported\_to\_father}.p \wedge (q = (\mathsf{father}.p))$

Figure 7: Guards of the actions from PLUM

6 The correctness of PLUM

The UNITY specification, stating termination of PLUM, reads:

Theorem 6.1 *HYLO_PLUM*

$$J_{\text{PLUM}} \ {}_{\text{PLUM}}\vdash \ \ \mathbf{ini}\text{PLUM} \rightsquigarrow \forall p : p \in \mathbb{P} : \mathit{done}.p$$

This specification is refined and decomposed – using the laws of the UNITY logic from Section A.4 and Appendices B and C – until it is expressed in one-step progress (i.e. ensures) and safety (i.e. ↻) properties that can be proved directly from the actions of the PLUM algorithm (see Figure 2).

6.1 Incremental, demand-driven construction of invariants

As already stated, we shall construct our invariant J_{PLUM} incrementally in a demand driven way *during* the process of refinement and decomposition. More specific, at the begin of the refinement and decomposition, the invariant J_{PLUM} is unspecified. Subsequently, at those points in the proof where an invariant is needed we propose a candidate cJ^i_{PLUM} for part of the invariant which suffices for that particular point in the proof. After decomposition, we gather all the candidates we have proposed during the refinement and decomposition of the initial specification, and from them deduce the minimal invariant J_{PLUM} that implies all the proposed candidates. To give a clear indication when a candidate for part of the invariant is proposed we shall mark this point by:

$cJ^i_{\text{PLUM}} = \ldots$

Once introduced it is assumed that J_{PLUM} implies the candidate, since this shall be ensured at the end of the decomposition. Similarly, we shall assume the stability of J_{PLUM} throughout the whole process of refinement and decomposition. Finally, we will call a candidate that is proposed for being part of the invariant, an *invariant-candidate*.

6.2 PLUM's variables and actions

During the verification, we shall assume that all of PLUM's variables are distinct. That is, e.g. for the idle variables it is assumed that:

$$\forall p, q \in \mathbb{P} : (\mathsf{idle}.p = \mathsf{idle}.q) = (p = q)$$

Similar properties are assumed for the V, father, nr_rec, nr_sent, and M variables. Moreover, we assume that the various kinds of variables are different, e.g. for the idle variables we assume:

$$\begin{array}{ll}\forall p, q, r \in \mathbb{P} : & (\mathsf{idle}.p \neq \mathsf{V}.q) \wedge (\mathsf{idle}.p \neq \mathsf{father}.q) \wedge (\mathsf{idle}.p \neq \mathsf{nr\_rec}.q.r) \\ & (\mathsf{idle}.p \neq \mathsf{nr\_sent}.q.r) \wedge (\mathsf{idle}.p \neq \mathsf{M}.q.r)\end{array}$$

Again similar properties are assumed for the V, father, nr_rec, nr_sent, and M variables. The exact definition capturing these properties of PLUM's is not presented here, since obviously it is very tedious and takes up a lot of space.

Theorems 6.2 through 6.5 indicate which variables are written by the various actions of the PLUM algorithm. (For the definition of $\not\leftarrow$ see A.5$_{53}$.) Since we assume the validity of distinct_PLUM_Vars, we know that if, for example, $(p \neq p')$, then action IDLE.$p.q$ does not write to the variables idle.p', father.p', M.$q.p'$, nr_rec.$p'.q$, and V.p'.

For ease of referring to the guards of the various actions of PLUM, Theorems 6.6 through 6.9 state them.

6.3 Presenting proofs of unless and ensures properties

During the refinement and decomposition of the specification, various one-step safety (i.e. unless) and progress (i.e. ensures) properties have to be verified. To enhance the readability of their proofs, this section shall introduce the proof format for the verification of these properties.

The proof obligations stating ensures-properties are introduced through an application of the $\rightsquigarrow$ INTRODUCTION (C.3$_{55}$) theorem. More specifically, applying this theorem results in proof obligations of the form:

$$\vdash (J_{\text{PLUM}} \wedge x) \ \mathsf{ensures} \ y$$

Rewriting with Definitions A.8$_{53}$ and A.12$_{54}$ gives us:

$$\forall A \in \mathbf{a}\text{PLUM}, s, t \in \texttt{State} : \ J_{\text{PLUM}}.s \wedge x.s \wedge \neg y.s \wedge \mathsf{compile}.A.s.t \Rightarrow (J_{\text{PLUM}}.t \wedge x.t) \vee y.t \ \Big\} \ \mathsf{unless} - \text{part}$$

$$\wedge$$

$$\exists A \in \mathbf{a}\text{PLUM} : \forall s, t \in \texttt{State} : \ J_{\text{PLUM}}.s \wedge x.s \wedge \neg y.s \wedge \mathsf{compile}.A.s.t \Rightarrow y.t \ \Big\} \ \mathsf{exists} - \text{part}$$

To prevent tedious rewriting with unless and ensures, and repeated discharging of the hypotheses at the left hand side of the implications, we introduce the proof-format displayed in Figure 8.

$\vdash (J_{\textsc{plum}} \wedge x)$ ensures y

unless-part.

IDLE.$p'.q'.s.t$

the proof that is displayed here, implicitly assumes the validity of

- $J_{\textsc{plum}}.s$ (and because of the assumed stability of $J_{\textsc{plum}}$ (Section 6.1) also $J_{\textsc{plum}}.t$)
- $x.s$
- $\neg y.s$
- compile.(IDLE.$p'.q'$).$s.t$

and aims to verify that $x.t \vee y.t$.

COL.$p'.q'.s.t$ dito, but then for COL

PROP.$p'.q'.s.t$ dito, but then for PROP

DONE.$p'.q'.s.t$ dito, but then for DONE

exists-part: directly after the colon we shall write that action A that is used to reduce the existential quantification.

Then, we present a proof that – under the implicit assumptions that $J_{\textsc{plum}}.s$, $x.s$, and $\neg y.s \wedge \mathsf{compile}.A.s.t$ – verifies that the action establishes the desired progress (i.e. $y.t$).

Figure 8: The proof-format for the verification of ensures -properties

Theorem 6.10 *not_evalb_sent_2_all_except_f*

$$\neg \mathit{sent\_to\_all\_non\_fathers}.p.s \exists q : q \in \mathsf{neighs}.p \ \wedge \ q \neq s.(\mathsf{father}.p) \ \wedge \ s.(\mathsf{nr\_sent}.p.q) \neq 1$$

Theorem 6.11 *not_evalb_rec_from_all_neighs*

$$\neg \mathit{rec\_from\_all\_neighs}.p.s = \exists q : q \in \mathsf{neighs}.p \ \wedge \ s.(\mathsf{nr\_rec}.p.q) \neq 1$$

Theorem 6.12 *finished_and_sent_2_f_IMP_sent_2_all_neighs*

$$\frac{\mathit{finished\_collecting\_and\_propagating}.p.s \wedge \mathit{reported\_to\_father}.p.s}{\mathit{sent\_to\_all\_neighs}.p.s}$$

Figure 9: Some useful theorems for arbitrary processes $p \in \mathbb{P}$ and states $s \in$ `State`

6.4 Some more theorems, notation and assumptions

Figure 9 displays some simple theorems that turn out to be useful during the verification, they all follow naturally from (1) through (7) on page 8.

During the whole process of verification, we shall assume that we have a connected centralised communication network. i.e. Connected_Network.$\mathbb{P}$.neighs.starter.

Moreover, during the process of decomposition:

$\vdash$ abbreviates $J_{\textsc{plum}\ \textsc{plum}}\vdash$.

6.5 Refinement and decomposition strategy

The global strategy applied to decompose the specification stating termination of distributed hylomorphisms, is inherent to the structure of distributed hylomorphisms:

$$\underbrace{\text{let the information flow from leaves to root of the RST}}_{\textbf{cata}} \circ \underbrace{\text{build an RST}}_{\textbf{ana}}$$

Distributed hylomorphisms build an RST by flooding messages to all processes in such a way that:

- when an idle process p receives its first message from q, it marks q as its father and opens its floodgate by becoming non-idle
- non-idle processes only flood (i.e. propagate) messages to non-father-neighbours.

Consequently, the shape of the rooted spanning tree is established by the father relation, once all processes have become non-idle. The construction of the tree, however, is finished only when

(1) every process has sent messages to all its neighbours that are not its father (i.e. it has sent messages to all of its non-father-neighbours)

(2) all messages meant in (1) are actually received (i.e. every process has received messages from all of its non-child-neighbours)

Requirement (1) is captured by the definition of *sent_to_all_non_fathers* (see (2) on page 8). Requirement (2) is, for some process $p \in \mathbb{P}$, characterised by the following definition:

Definition 6.13 RECEIVED FROM ALL NON-CHILDREN *rec_from_all_non_child*

$$rec\_from\_all\_non\_children.p = \forall q \in \mathsf{neighs}.p\ :\ (p \neq (\mathsf{father}.q)) \Rightarrow (\mathsf{nr\_rec}.p.q = 1)$$

this predicate states that process p has at least received messages from those neighbours of which p is not the father. Thus, in other words, p has at least received messages from all its non-child-neighbours.

Applying this global proof strategy to the initial specification results in the following **anamorphism**- and **catamorphism**-part:

$$\vdash \mathbf{ini}\mathrm{PLUM} \rightsquigarrow \forall p : p \in \mathbb{P} : done.p$$

$\Leftarrow$($\rightsquigarrow$ TRANSITIVITY (C.5_{55}))

$$\left.\begin{array}{ll}\vdash & \mathbf{ini}\mathrm{PLUM} \\ & \rightsquigarrow \\ & (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \\ & \wedge(\forall p \in \mathbb{P} : sent\_to\_all\_non\_fathers.p) \\ & \wedge(\forall p \in \mathbb{P} : rec\_from\_all\_non\_children.p)\end{array}\right\}\ \mathbf{anamorphism - part}$$

$$\wedge$$

$$\left.\begin{array}{ll}\vdash & (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \\ & \wedge(\forall p \in \mathbb{P} : sent\_to\_all\_non\_fathers.p) \\ & \wedge(\forall p \in \mathbb{P} : rec\_from\_all\_non\_children.p) \\ & \rightsquigarrow \\ & \forall p : p \in \mathbb{P} : done.p\end{array}\right\}\ \mathbf{catamorphism - part}$$

6.6 Verification of the anamorphism part

Decomposition of the **anamorphism**-part is straightforward and follows naturally from the discussion in the previous section: first prove that the shape of the RST is established by proving that all processes eventually become non-idle (**ana_1**); then prove that all processes end the construction of the RST by sending messages to all their non-father-neighbours (**ana_2**); finally prove that all messages sent in order to construct the RST are eventually received (**ana_3**).

$$\left.\begin{array}{ll}\vdash & \mathbf{ini}\mathrm{PLUM} \\ & \rightsquigarrow \\ & (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \\ & \wedge(\forall p \in \mathbb{P} : sent\_to\_all\_non\_fathers.p) \\ & \wedge(\forall p \in \mathbb{P} : rec\_from\_all\_non\_children.p)\end{array}\right\}\ \mathbf{anamorphism - part}$$

$\Leftarrow$($\rightsquigarrow$ ACCUMULATION (C.7_{56}), twice)

$$\left.\begin{array}{ll}\vdash & \mathbf{ini}\mathrm{PLUM} \\ & \rightsquigarrow \\ & \forall p \in \mathbb{P} : \neg\mathsf{idle}.p\end{array}\right\}\ \mathbf{ana\_1}$$

$\wedge$

$$\left.\begin{array}{ll}\vdash & \forall p \in \mathbb{P} : \neg\mathsf{idle}.p \\ & \rightsquigarrow \\ & \forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p\end{array}\right\}\textbf{ana\_2}$$

$\wedge$

$$\left.\begin{array}{ll}\vdash & (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \wedge (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) \\ & \rightsquigarrow \\ & (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \wedge (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p)\end{array}\right\}\textbf{ana\_3}$$

The verification of ana_1

Decomposition of **ana_1** proceeds by induction on the structure of the connected network underlying the PLUM algorithm. That is, we prove that when a process p is non-idle, then eventually all its neighbours will become non-idle. Consequently, from the connectivity of the network it can be deduced that since the *starter* is non-idle, eventually all processes will be non-idle.

$\vdash \textbf{ini}\mathrm{PLUM} \rightsquigarrow \forall p \in \mathbb{P} : \neg\mathsf{idle}.p \quad \}\,\textbf{ana\_1}$
$\Leftarrow$($\rightsquigarrow$ Substitution (C.2_{55}), using characterisation of initial condition PLUM)
$\vdash \forall p \in \{\mathit{starter}\} : \neg\mathsf{idle}.p \rightsquigarrow \forall p \in \mathbb{P} : \neg\mathsf{idle}.p$
$\Leftarrow$(rewrite with the definition of Connected_Network (4.3_6))
$\vdash \forall p \in \{\mathit{starter}\} : \neg\mathsf{idle}.p \rightsquigarrow \forall p \in \mathsf{iterate}.n.(\mathsf{Neighs.neighs}).\mathit{starter} : \neg\mathsf{idle}.p$
$\Leftarrow$($\rightsquigarrow$ Iterate (C.13_{56}))
$\forall L \subseteq \mathbb{P} : \vdash \forall p \in L : \neg\mathsf{idle}.p \rightsquigarrow \forall p \in \mathsf{Neighs.neighs}.L : \neg\mathsf{idle}.p$
$\Leftarrow$($\rightsquigarrow$ Substitution (C.2_{55}), prepare for $\rightsquigarrow$ Conjunction (C.11_{56}))
$\forall L \subseteq \mathbb{P} : \vdash \forall p \in L, \forall q \in \mathsf{neighs}.p : \neg\mathsf{idle}.p \wedge \neg\mathsf{idle}.p \rightsquigarrow \forall p \in L, \forall q \in \mathsf{neighs}.p : \neg\mathsf{idle}.q$
$\Leftarrow$($\rightsquigarrow$ Conjunction (C.11_{56}), three times)
$\forall L \subseteq \mathbb{P}, p \in L, q \in \mathsf{neighs}.p :\ (\vdash \neg\mathsf{idle}.p \rightsquigarrow \neg\mathsf{idle}.p) \ \wedge\ (\vdash \neg\mathsf{idle}.p \rightsquigarrow \neg\mathsf{idle}.q)$

The first conjunct can be proved using $\rightsquigarrow$ Reflexivity (C.4_{55}), and the stability of $\neg\mathsf{idle}.p$, stated below:

Theorem 6.14 *STABLEe_not_idle*

$\forall p \in \mathbb{P} : {}_{\mathrm{PLUM}}\vdash \circlearrowright \neg\mathsf{idle}.p$

We now proceed with the second conjunct. Since q is assumed to be an arbitrary neighbour of p, we have to make a distinction as to whether q is p's father or not.

$\Leftarrow$($\rightsquigarrow$ Case distinction (C.6_{55}))
$\forall L \subseteq \mathbb{P}, p \in L, q \in \mathsf{neighs}.p :$
$\underbrace{\vdash \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p) \rightsquigarrow \neg\mathsf{idle}.q}_{\textbf{ana\_1.1}} \ \wedge \underbrace{\vdash \neg\mathsf{idle}.p \wedge (q \neq \mathsf{father}.p) \rightsquigarrow \neg\mathsf{idle}.q}_{\textbf{ana\_1.2}}$

Examine the first conjunct **ana_1.1**, we need to verify that when a process p is non-idle, then eventually its father will be non-idle. When a process p is not idle, it has received a message from its father. Hence its father is not idle since otherwise it would not have been able to send a message to p. Therefore, the first conjunct should be provable from the invariant as follows: for arbitrary $p \in \mathbb{P}$ and $q \in \mathsf{neighs}.p$:

$\vdash \neg\mathsf{idle}.p \wedge q = \mathsf{father}.p \rightsquigarrow \neg\mathsf{idle}.q$
$\Leftarrow$($\rightsquigarrow$ Introduction (C.3_{55}))
$((J_{\mathrm{PLUM}} \wedge \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)) \Rightarrow \neg\mathsf{idle}.q) \ \wedge \vdash\ \circlearrowright (J_{\mathrm{PLUM}} \wedge \neg\mathsf{idle}.q)$

In order to establish this proof we introduce our first candidate for part of the invariant J_{PLUM}:

$$\boxed{cJ^1_{\mathrm{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \neg\mathsf{idle}.p \wedge q = \mathsf{father}.p \Rightarrow \neg\mathsf{idle}.q}$$

Obviously, when J_{PLUM} implies cJ^1_{PLUM}, the stability of J_{PLUM}, and the stability of $(\neg\mathsf{idle}.q)$ (stated in Theorem 6.14) establish **ana_1.1**.

The second conjunct **ana_1.2**, states that when a process p is non-idle, then eventually its non-father neighbours will be non-idle. Evidently, when p is non-idle, it shall eventually send a message to its non-father neighbour q; moreover, q shall eventually receive this message and, when not already non-idle, shall become non-idle. This is reflected in the following decomposition strategy: for arbitrary $p \in \mathbb{P}$ and $q \in \mathsf{neighs}.p$:

$$
\begin{array}{l}
\quad \vdash \neg\mathsf{idle}.p \wedge q \neq \mathsf{father}.p \rightsquigarrow \neg\mathsf{idle}.q \\
\Leftarrow (\rightsquigarrow \textsc{Transitivity}\ (\text{C}.5_{55})) \\
\quad \underbrace{\vdash \neg\mathsf{idle}.p \wedge q \neq \mathsf{father}.p \rightsquigarrow \mathsf{nr\_sent}.p.q = 1}_{\textbf{ana\_1.2.1}} \ \wedge\ \underbrace{\vdash \mathsf{nr\_sent}.p.q = 1 \rightsquigarrow \neg\mathsf{idle}.q}_{\textbf{ana\_1.2.2}}
\end{array}
$$

ana_1.2.1 can be proved using $\rightsquigarrow$ Introduction (C.3_{55}), leaving us with the proof obligations:

$$
\begin{array}{l}
\vdash\ \circlearrowright(J_{\text{PLUM}} \wedge \mathsf{nr\_sent}.p.q = 1) \\
\wedge \\
\vdash (J_{\text{PLUM}} \wedge \neg\mathsf{idle}.p \wedge q \neq \mathsf{father}.p)\ \mathsf{ensures}\ (\mathsf{nr\_sent}.p.q = 1)
\end{array}
$$

Stability of $(\mathsf{nr\_sent}.p.q = 1)$ can be proved separately from invariant J_{PLUM}, since, for all $p \in \mathbb{P}$ and $q \in \mathsf{neighs}.p$, the guards of PROP.$p.q$ and DONE.$p.q$ imply that $\mathsf{nr\_sent}.p.q = 0$. The proof is straightforward and the resulting theorem is presented below.

Theorem 6.15 *STABLEe_nr_sent_is_1*

$$\forall p, q \in \mathbb{P} :\ {}_{\text{PLUM}}\vdash \circlearrowright(\mathsf{nr\_sent}.p.q = 1)$$

Consequently,

$$
\begin{array}{l}
\quad \vdash\ \circlearrowright(J_{\text{PLUM}} \wedge (\mathsf{nr\_sent}.p.q = 1)) \\
\Leftarrow (\circlearrowright \textsc{Conjunction}\ \text{A}.11_{53}) \\
\quad \vdash\ \circlearrowright J_{\text{PLUM}} \quad \wedge\ \vdash\ \circlearrowright(\mathsf{nr\_sent}.p.q = 1)
\end{array}
$$

Which is proved by the assumed stability of J_{PLUM}, and Theorem 6.15 from above.

The validation of the ensures -property is below:

$$\vdash (J_{\text{PLUM}} \wedge \neg\mathsf{idle}.p \wedge q \neq \mathsf{father}.p)\ \mathsf{ensures}\ (\mathsf{nr\_sent}.p.q = 1)$$

unless-part

IDLE.$p'.q'.s.t$

- if $p \neq p'$, then $\mathsf{idle}.p$ and $\mathsf{father}.p$ are not written by IDLE.$p'.q'.s.t$ and thus $s.(\mathsf{idle}.p) = t.(\mathsf{idle}.p)$ and $s.(\mathsf{father}.p) = t.(\mathsf{father}.p)$.
- if $p = p'$, then $(s = t)$ since the guard of IDLE.$p'.q'.s.t$ is disabled by $\neg s.(\mathsf{idle}.p)$. (see the explanation on the implicit assumptions implied by the presentation of ensures -properties from Section 6.3).

COL.$p'.q'.s.t$, PROP.$p'.q'.s.t$, DONE.$p'.q'.s.t$ do not write to the idle and father variables (Theorems 6.3_{11} through 6.5_{11}).

exists-part: PROP.$p.q.s.t$.

In order to verify that this action indeed sends a message to its neighbour q, we have to prove that its guard is enabled in state s. More specific (Theorem 6.8_{11}) this comes down to verifying that:

$$\neg s.(\mathsf{idle}.p) \wedge (s.(\mathsf{nr\_sent}.p.q) = 0) \wedge (q \neq s.(\mathsf{father}.p)) \wedge \neg \mathit{sent\_to\_all\_non\_fathers}.p.s$$

The implicit assumptions of ensures -proofs (Figure 8) tell us that $\neg s.(\mathsf{idle}.p)$, $(q \neq s.(\mathsf{father}.p))$, and $(s.(\mathsf{nr\_sent}.p.q) \neq 1)$, and hence Theorem 6.$10_{13}$ implies that $\neg \mathit{sent\_to\_all\_non\_fathers}.p.s$, the following proof obligation remains:

$$s.(\mathsf{nr\_sent}.p.q) = 0$$

In order to prove this, we need to propose an additional candidate for part of the invariant. Since, we have that $(s.(\mathsf{nr\_sent}.p.q) \neq 1)$, the invariant-part that suffices here, is a predicate stating that the number of messages a process has sent to a neighbour is always 0 or 1.

$$cJ^2_{\text{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \mathsf{nr\_sent}.p.q = 0 \;\vee\; \mathsf{nr\_sent}.p.q = 1$$

This ends the validation of **ana_1.2.1**.

Using Theorem 6.14_{15}, the assumed stability of J_{PLUM}, $\circlearrowright$ CONJUNCTION $\text{A}.11_{53}$, and $\leadsto$ INTRODUCTION $(\text{C}.3_{55})$, the proof obligation **ana_1.2.2** can be reduced to:

$$\vdash (J_{\text{PLUM}} \wedge \mathsf{nr\_sent}.p.q = 1) \;\mathsf{ensures}\; (\neg \mathsf{idle}.q)$$

unless-part

IDLE.$p'.q'.s.t$, COL.$p'.q'.s.t$ do not write to the nr_sent variables (Theorems 6.2_{11} and 6.3_{11}).

PROP.$p'.q'.s.t$

- If $(p \neq p')$ or $(q \neq q')$, the variable $\mathsf{nr\_sent}.p.q$ is not written.
- If $(p = p')$ and $(q = q')$, then $s = t$ because the guard of PROP.$p'.q'.s.t$ is disabled by the fact that $\mathsf{nr\_sent}.p'.q' = 1$ in state s.

DONE.$p'.q'.s.t$

- If $(p \neq p')$ or $(q \neq q')$ the variable $\mathsf{nr\_sent}.p.q$ is not written.
- Suppose $(p = p')$ and $(q = q')$.
  - If $q' \neq s.(\mathsf{father}.p')$ then the guard of DONE.$p'.q'.s.t$ is disabled and hence $s = t$.
  - Suppose $q' = s.(\mathsf{father}.p')$.
    - If $\neg$*finished_collecting_and_propagating.p.s*, then, from Theorem 6.9_{11}, we can deduce that the guard of DONE.$p'.q'.s.t$ is disabled, and hence that $s = t$.
    - If *finished_collecting_and_propagating.p.s*, then p has *sent_to_all_non_fathers* in state s $(4)_8$. Moreover, since we know that $\mathsf{nr\_sent}.p'.(\mathsf{father}.p') = 1$ in state s we have that (Theorem 6.12_{13}) *sent_to_all_neighs.p.s* and thus *done.p.s*. Consequently, the guard of DONE.$p'.q'.s.t$ is disabled and hence $s = t$.

exists-part: IDLE.$q.p.s.t$

In order to verify that process q indeed receives a message from its neighbour p, and becomes non-idle we have to prove that the guard of IDLE.$q.p.s.t$ is enabled in state s. Using Theorem 6.6_{11}, and the assumption that $s.(\mathsf{idle}.p)$ this comes down to verifying that:

$\mathsf{mit}.p.q.s$

The implicit assumptions and the already proposed invariant-candidates cJ^1_{PLUM} and cJ^2_{PLUM} do not give enough information to prove this. Consequently, we shall again have to construct some additional invariant-candidates. Intuitively, when a message is in transit from p to q this will always mean that $(\mathsf{nr\_rec}.q.p < \mathsf{nr\_sent}.p.q)$. Moreover, when a process p is idle this means that is has not yet received any message and hence all its nr_rec variables are 0. Proposing these as candidates for part of the invariant, enables us to prove the current exists-part. Since we have here that q is idle and $s.(\mathsf{nr\_sent}.p.q = 1)$, we can deduce that $(s.(\mathsf{nr\_rec}.q.p) < s.(\mathsf{nr\_sent}.p.q))$ and hence $\mathsf{mit}.p.q.s$.

$$cJ^3_{\text{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \mathsf{idle}.p \Rightarrow \mathsf{nr\_rec}.p.q = 0$$

$$cJ^4_{\text{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : (\mathsf{nr\_rec}.q.p < \mathsf{nr\_sent}.p.q) = \mathsf{mit}.p.q$$

This establishes the proof of **ana_1.2.2**, **ana_1.2**, and hence **ana_1**. For future reference the results are summarised in Figure 10.

The verification of ana_2

Proving that a non-idle process shall eventually send messages to all its non-father-neighbours can be proved by re-using **ana_1.2.1** (Theorem 6.16). The following derivation aims at bringing **ana_2** into the

Theorem 6.16 **ana_1.2.1** *not_idle_CON_sent_2_neighs_ex_f*

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : J_{\text{PLUM}} \ {}_{\text{PLUM}}\vdash \neg\mathsf{idle}.p \wedge (q \neq \mathsf{father}.p) \leadsto \mathsf{nr\_sent}.p.q = 1$

Theorem 6.17 **ana_1.2.2** *sent_to_q_CON_not_idle_q*

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : J_{\text{PLUM}} \ {}_{\text{PLUM}}\vdash \mathsf{nr\_sent}.p.q = 1 \leadsto \neg\mathsf{idle}.q$

Theorem 6.18 **ana_1.1** *not_idle_CON_idle_father*

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : J_{\text{PLUM}} \ {}_{\text{PLUM}}\vdash \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p) \leadsto \neg\mathsf{idle}.q$

Theorem 6.19 **ana_1.2** *not_idle_CON_not_idle_neighs*

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : J_{\text{PLUM}} \ {}_{\text{PLUM}}\vdash \neg\mathsf{idle}.p \wedge (q \neq \mathsf{father}.p) \leadsto \neg\mathsf{idle}.q$

Theorem 6.20 **ana_1** *Init_CON_all_not_idle*

$J_{\text{PLUM}} \ {}_{\text{PLUM}}\vdash \mathbf{ini}(\text{PLUM}.iA.h.\text{PROP\_mes}.\text{DONE\_mes}) \leadsto \forall p \in \mathbb{P} : \neg\mathsf{idle}.p$

Figure 10: Verification of **ana_1**

correct form for application of **ana_1.2.1**. (The notes ♫, with which some of the derivation steps are marked, can be ignored here. Their purpose will become clear later on.)

$\vdash \forall p \in \mathbb{P} : \neg\mathsf{idle}.p \leadsto \forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p \quad \}\ \mathbf{ana\_2}$

$\Leftarrow$($\leadsto$ SUBSTITUTION (C.2_{55}), $(2)_8$; prepare for $\leadsto$ CONJUNCTION (C.11_{56})) (♫)

$\vdash \quad \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \neg\mathsf{idle}.p$

$\leadsto$

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : (\neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)) \vee (\mathsf{nr\_sent}.p.q = 1)$

$\Leftarrow$($\leadsto$ CONJUNCTION (C.11_{56}), twice) (♫)

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \vdash \neg\mathsf{idle}.p \leadsto (\neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)) \vee (\mathsf{nr\_sent}.p.q = 1)$

$\Leftarrow$($\leadsto$ CASE DISTINCTION (C.6_{55}))

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :$

$\vdash \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p) \ \leadsto (\neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)) \vee (\mathsf{nr\_sent}.p.q = 1)$

$\wedge$

$\vdash \neg\mathsf{idle}.p \wedge (q \neq \mathsf{father}.p) \ \leadsto (\neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)) \vee (\mathsf{nr\_sent}.p.q = 1)$

$\Leftarrow$($\leadsto$ SUBSTITUTION (C.2_{55}) on the right hand side of both conjuncts)

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :$

$\vdash \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p) \ \leadsto \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)$

$\wedge$

$\vdash \neg\mathsf{idle}.p \wedge (q \neq \mathsf{father}.p) \ \leadsto (\mathsf{nr\_sent}.p.q = 1)$

$\Leftarrow$(Second conjunct is proved by Theorem 6.16_{18})

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \ \vdash \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p) \ \leadsto \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)$

$\Leftarrow$($\leadsto$ REFLEXIVITY (C.4_{55}), $\circlearrowleft$ CONJUNCTION A.11_{53}, and assumed stability of J_{PLUM})

$\vdash \ \circlearrowleft \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)$

This stability predicate is straightforward to prove since a non-idle process stays non-idle (Theorem 6.14_{15}) and does not write to its father variables.

Theorem 6.21 *STABLEe_not_idle_AND_q_IS_f_p*

$\forall p, q \in \mathbb{P} : \ {}_{\text{PLUM}}\vdash \circlearrowleft \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)$

For future reference we again summarise:

Theorem 6.22 ana_2 *not_idle_CON_not_propagating*

$$J_{\text{PLUM}} \ {}_{\text{PLUM}}\vdash \forall p \in \mathbb{P} : \neg\mathsf{idle}.p \leadsto \forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p$$

Theorem 6.23 *not_idle_AND_q_IS_f_p_CON_REFL*

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : J_{\text{PLUM}} \ {}_{\text{PLUM}}\vdash \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p) \ \leadsto \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)$$

Verification of ana_3

Proving **ana_3** comes down to verifying that when a message is sent, it shall eventually be received. In order to derive this proof obligation, we proceed as follows:

$$\left.\begin{array}{ll} \vdash & (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \wedge (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) \\ & \leadsto \\ & (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \wedge (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p) \end{array}\right\} \textbf{ana\_3}$$

$\Leftarrow$($\leadsto$ Substitution (C.2_{55}), $(2)_8$, and Definition 6.13_{14})

$$\begin{array}{ll} \vdash & \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \neg\mathsf{idle}.p \wedge ((q \neq \mathsf{father}.p) \Rightarrow (\mathsf{nr\_sent}.p.q = 1)) \\ & \leadsto \\ & \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \neg\mathsf{idle}.p \wedge ((q \neq \mathsf{father}.p) \Rightarrow (\mathsf{nr\_rec}.q.p = 1)) \end{array}$$

$\Leftarrow$($\leadsto$ Conjunction (C.11_{56}), twice)

$$\begin{array}{ll} \multicolumn{2}{l}{\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :} \\ \vdash & \neg\mathsf{idle}.p \wedge ((q \neq \mathsf{father}.p) \Rightarrow (\mathsf{nr\_sent}.p.q = 1)) \\ & \leadsto \\ & \neg\mathsf{idle}.p \wedge ((q \neq \mathsf{father}.p) \Rightarrow (\mathsf{nr\_rec}.q.p = 1)) \end{array}$$

= (logic)

$$\begin{array}{ll} \multicolumn{2}{l}{\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :} \\ \vdash & (\neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)) \ \vee \ (\neg\mathsf{idle}.p \wedge (\mathsf{nr\_sent}.p.q = 1)) \\ & \leadsto \\ & (\neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)) \ \vee \ (\neg\mathsf{idle}.p \wedge (\mathsf{nr\_rec}.q.p = 1)) \end{array}$$

$\Leftarrow$($\leadsto$ Disjunction (C.10_{56}))

$$\begin{array}{ll} \multicolumn{2}{l}{\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :} \\ \vdash & \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p) \leadsto \neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p) \\ \wedge & \\ \vdash & \neg\mathsf{idle}.p \wedge (\mathsf{nr\_sent}.p.q = 1) \leadsto \neg\mathsf{idle}.p \wedge (\mathsf{nr\_rec}.q.p = 1) \end{array}$$

$\Leftarrow$(First conjunct is proved by Theorem 6.23_{19})

$$\begin{array}{ll} \multicolumn{2}{l}{\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :} \\ \vdash & \neg\mathsf{idle}.p \wedge (\mathsf{nr\_sent}.p.q = 1) \leadsto \neg\mathsf{idle}.p \wedge (\mathsf{nr\_rec}.q.p = 1) \end{array}$$

$\Leftarrow$($\leadsto$ Conjunction (C.11_{56}))

$$\begin{array}{l} \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \\ \quad (\vdash \ \ \neg\mathsf{idle}.p \leadsto \neg\mathsf{idle}.p) \ \ \wedge \ (\vdash \ \ \mathsf{nr\_sent}.p.q = 1 \leadsto \mathsf{nr\_rec}.q.p = 1) \end{array}$$

$\Leftarrow$(First conjunct is proved using $\leadsto$ Reflexivity (C.4_{55}), and Theorem 6.14_{15})

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \vdash \mathsf{nr\_sent}.p.q = 1 \leadsto \mathsf{nr\_rec}.q.p = 1$$

So we have to prove that when a process p sends a message to a neighbour q, then q shall eventually receive this message. Since nothing is known about q, there are two possibilities:

q is non-idle In this case the execution of COL.$q.p$ shall ensure that p's message is eventually received.

q is idle This case is more subtle, since it is not ensured that execution of IDLE.$q.p$ shall receive p's message. In illustration, suppose another neighbour r ($r \neq p$) has also sent a message to the idle process q. If q decides to receive r's message before it receives the one from p, then q registers r as its father and becomes non-idle. Consequently, subsequent executions of q's IDLE-actions will behave like skip and therefore shall not be responsible for the receipt of p's message. In this case q's COL actions will ensure that p's message is eventually received.

This is reflected in the following proof:

$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \vdash \mathsf{nr\_sent}.p.q = 1 \leadsto \mathsf{nr\_rec}.q.p = 1$
$\Leftarrow$($\leadsto$ CASE DISTINCTION (C.6_{55}))

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p$$
$$\underbrace{\vdash \mathsf{nr\_sent}.p.q = 1 \land \neg \mathsf{idle}.q \leadsto \mathsf{nr\_rec}.q.p = 1}_{\textbf{ana\_3.1}}$$
$$\land$$
$$\underbrace{\vdash \mathsf{nr\_sent}.p.q = 1 \land \mathsf{idle}.q \leadsto \mathsf{nr\_rec}.q.p = 1}_{\textbf{ana\_3.2}}$$

As indicated, when q is non-idle (**ana_3.1**) the execution of COL.$q.p$ shall ensure that p's message is eventually received. Consequently, $\leadsto$ INTRODUCTION (C.3_{55}) is applied to **ana_3.1** giving us: for arbitrary $p \in \mathbb{P}$ and $q \in \mathsf{neighs}.p$

$$\vdash \circlearrowright J_{\text{PLUM}} \land \mathsf{nr\_rec}.q.p = 1$$
$$\land$$
$$\vdash (J_{\text{PLUM}} \land \mathsf{nr\_sent}.p.q = 1 \land \neg \mathsf{idle}.q) \text{ ensures } (\mathsf{nr\_rec}.q.p = 1)$$

Stability of ($\mathsf{nr\_rec}.q.p = 1$) cannot be proved separately from the stability of J_{PLUM}. The reason for this is that – unlike the guards of PROP.$p.q$ and DONE.$p.q$ that imply that ($\mathsf{nr\_sent}.p.q = 0$) and hence allow for the separate verification of $\circlearrowright(\mathsf{nr\_sent}.p.q = 1)$ – the guards of IDLE.$p.q$ and COL.$p.q$ actions do *not* imply that ($\mathsf{nr\_rec}.p.q = 0$). However, in combination with the proposed invariant-candidates they do. cJ^3_{PLUM} implies that when q is idle, $\mathsf{nr\_rec}.q.p = 0$. Therefore, when the guard of IDLE.$q.p$ (Definition 6.6_{11}) is enabled the validity J_{PLUM} implies $\mathsf{nr\_rec}.q.p = 0$. cJ^4_{PLUM}, together with cJ^2_{PLUM}, implies that when $\mathsf{mit}.q.p$ holds, $\mathsf{nr\_rec}.q.p = 0$. Therefore, when the guard of COL.$q.p$ (Definition 6.7_{11}) is enabled the validity J_{PLUM} implies $\mathsf{nr\_rec}.q.p = 0$. Consequently, we have the following theorem:

Theorem 6.24 *STABLEe_Invariant_AND_nr_rec_is_1*

$$\forall p, q \in \mathbb{P} : \ {}_{\text{PLUM}}\vdash \circlearrowright(J_{\text{PLUM}} \land \mathsf{nr\_rec}.p.q = 1)$$

The validation of the ensures -property is below:

$$\vdash (J_{\text{PLUM}} \land \mathsf{nr\_sent}.p.q = 1 \land \neg \mathsf{idle}.q) \text{ ensures } (\mathsf{nr\_rec}.q.p = 1)$$

unless-part

IDLE.$p'.q'.s.t$
- if ($p' = q$), then ($s = t$) since the guard of IDLE.$p'.q'.s.t$ is disabled by $\neg s.(\mathsf{idle}.q)$.
- if ($p' \neq q$) the variables $\mathsf{idle}.q$ and $\mathsf{nr\_sent}.p.q$ are not written

COL.$p'.q'.s.t$ does not write to idle and nr_sent variables (Theorem 6.3_{11}).

PROP.$p'.q'.s.t$
- If ($p \neq p'$) or ($q \neq q'$) the variable $\mathsf{nr\_sent}.p.q$ is not written. (idle variables are not written at all by PROP)
- If ($p = p'$) and ($q = q'$), then ($s = t$) since the guard of PROP.$p'.q'.s.t$ is disabled by the validity of ($s.(\mathsf{nr\_sent}.p'.q') = 1$).

DONE.$p'.q'.s.t$
- If ($p \neq p'$) or ($q \neq q'$) the variable $\mathsf{nr\_sent}.p.q$ is not written. (idle variables are not written at all by DONE)
- Suppose ($p = p'$) and ($q = q'$).
  - If $q' \neq s.(\mathsf{father}.p')$ then, from Theorem 6.9_{11}, we can deduce that the guard of DONE.$p'.q'.s.t$ is disabled and hence $s = t$.
  - Suppose $q' = s.(\mathsf{father}.p')$.
    - If $\neg$*finished_collecting_and_propagating.p.s*, then, from Theorem 6.9_{11}, we can deduce that the guard of DONE.$p'.q'.s.t$ is disabled and hence $s = t$.
    - If *finished_collecting_and_propagating.p.s*, then *sent_to_all_non_fathers.p.s* follows from $(4)_8$. Moreover, since p' has already sent to its father (i.e. ($s.(\mathsf{nr\_sent}.p'.(s.(\mathsf{father}.p'))) = 1$)) we have that (Theorem 6.12_{13}) *sent_to_all_neighs.p.s* and thus *done.p.s*. Consequently, the guard of DONE.$p'.q'.s.t$ is disabled and hence $s = t$.

exists-part: $\text{COL}.q.p.s.t$

In order to verify that process q indeed receives a message from its neighbour p, and establishes $t.(\mathsf{nr\_rec}.q.p) = 1$ we have to prove that the guard of $\text{COL}.q.p.s.t$ is enabled in state s, and $s.(\mathsf{nr\_rec}.q.p) = 0$. Since $s.(\mathsf{nr\_rec}.q.p) \neq 1$, Theorem 6.11_{13} gives us $\neg \mathit{rec\_from\_all\_neighs}.q$. Using Theorem 6.7_{11}, and the assumption that $\neg s.(\mathsf{idle}.p)$ the proof obligations that remain are:

$\mathsf{mit}.p.q.s \wedge s.(\mathsf{nr\_rec}.q.p) = 0$
$=$ (cJ^4_{PLUM}, and the assumption that $s.(\mathsf{nr\_sent}.p.q) = 1$)
$s.(\mathsf{nr\_rec}.q.p) < 1 \wedge s.(\mathsf{nr\_rec}.q.p) = 0$
$=$ (arithmetic)
$s.(\mathsf{nr\_rec}.q.p) = 0$

Again, looking at the assumptions and the already proposed invariant-candidates, we do not have enough information to prove this. Consequently, we introduce the following candidate, which obviously suffices in this case.

$$\boxed{cJ^5_{\text{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : (\mathsf{nr\_rec}.p.q = 0) \vee (\mathsf{nr\_rec}.p.q = 1)}$$

We hereby end the proof of **ana_3.1**.

Theorem 6.25 ana_3.1 *not_idle_AND_neigh_has_sent_CON_rec*

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \quad J_{\text{PLUM}} \;{}_{\text{PLUM}}\vdash \mathsf{nr\_sent}.p.q = 1 \wedge \neg\mathsf{idle}.q \rightsquigarrow \mathsf{nr\_rec}.q.p = 1$$

We continue with **ana_3.2** using the strategy delineated earlier on page 19.

$\quad \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \vdash \mathsf{nr\_sent}.p.q = 1 \wedge \mathsf{idle}.q \rightsquigarrow \mathsf{nr\_rec}.q.p = 1$
$\Leftarrow$($\rightsquigarrow$ TRANSITIVITY (C.5_{55}))
$\quad \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :$
$\quad \vdash \mathsf{nr\_sent}.p.q = 1 \wedge \mathsf{idle}.q \rightsquigarrow \mathsf{nr\_sent}.p.q = 1 \wedge \neg\mathsf{idle}.q \wedge (\exists r : \mathsf{nr\_rec}.q.r = 1)$
$\quad \wedge$
$\quad \vdash \mathsf{nr\_sent}.p.q = 1 \wedge \neg\mathsf{idle}.q \wedge (\exists r : \mathsf{nr\_rec}.q.r = 1) \rightsquigarrow \mathsf{nr\_rec}.q.p = 1$

Using $\rightsquigarrow$ SUBSTITUTION (C.2_{55}), the second conjunct can be reduced to, and hence proved by, Theorem 6.25_{21}. The first conjunct is proved by $\rightsquigarrow$ INTRODUCTION (C.3_{55}):

$\vdash \circlearrowright (J_{\text{PLUM}} \wedge \mathsf{nr\_sent}.p.q = 1 \wedge \neg\mathsf{idle}.q \wedge (\exists r : \mathsf{nr\_rec}.q.r = 1))$
$\wedge$
$\vdash (J_{\text{PLUM}} \wedge \mathsf{nr\_sent}.p.q = 1 \wedge \mathsf{idle}.q)$
$\quad$ ensures
$\quad (\mathsf{nr\_sent}.p.q = 1 \wedge \neg\mathsf{idle}.q \wedge (\exists r : \mathsf{nr\_rec}.q.r = 1))$

The stability requirement can be proved using $\circlearrowright$ CONJUNCTION A.11_{53}, Theorems 6.14_{15}, 6.15_{16}, and 6.24_{20}. The proof of the ensures -property is similar to that of **ana_3.1** on the understanding that $\text{IDLE}.q.p.s.t$ in instantiated in the exists-part instead of $\text{COL}.q.p.s.t$.

Theorem 6.26 ana_3.2 *idle_AND_neigh_has_sent_CON_rec*

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \quad J_{\text{PLUM}} \;{}_{\text{PLUM}}\vdash \mathsf{nr\_sent}.p.q = 1 \wedge \mathsf{idle}.q \rightsquigarrow \mathsf{nr\_rec}.q.p = 1$$

Theorem 6.27 ana_3 *not_propagating_and_not_idle_CON_not_idle_rec_from_all_non_child*

$$\begin{array}{ll} J_{\text{PLUM}} \;{}_{\text{PLUM}}\vdash & (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \wedge (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) \\ & \rightsquigarrow \\ & (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \wedge (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p) \end{array}$$

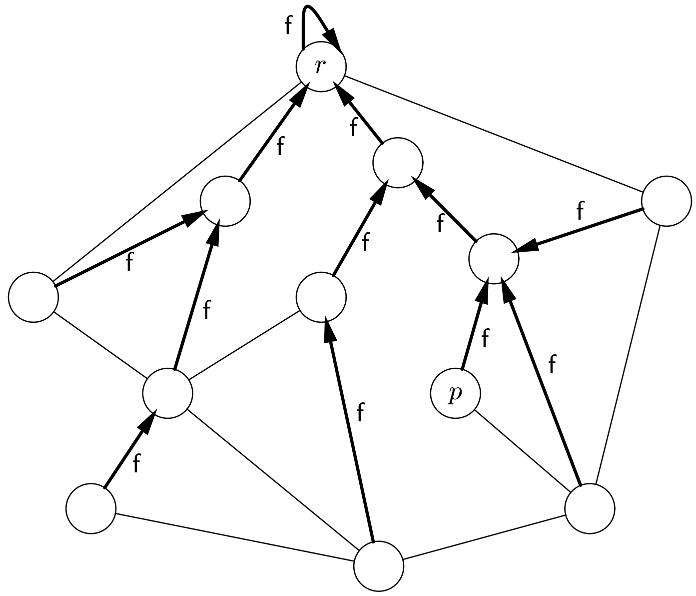


Figure 11: Rooted spanning tree; process p has depth 3.

6.7 Theory on rooted spanning trees

A *rooted spanning tree* of a connected communication network ($\mathbb{P}$, neighs) (see Figure 11) is a directed graph and consists of:

- a unique designated process r of the network which is considered to be the root of the tree, and hence has no outgoing edges to other processes in the network.
- a subset of communication links of the network, such that for all processes $p \in \mathbb{P}$ it holds that there is a unique path from p to r in the tree.

The tree is characterised by a process r and a function $\mathsf{f} \in \mathbb{P} \to \mathbb{P}$ (see Figure 11). To formalise the fact that the root is a process in the network, and has no outgoing edges to any other process, we define

$$(r \in \mathbb{P}) \wedge (\mathsf{f}.r = r)$$

Consequently, since the communication links in the tree have to be a subset of those in the network, f has to satisfy:

$$\forall p \in \mathbb{P} : (p \neq r) \Rightarrow (\mathsf{f}.p \in \mathsf{neighs}.p)$$

For ease of reference, when $q = \mathsf{f}.p$, we call q the *ancestor* or *father* of p, and similarly p the *descendant* or *child* of q. To specify that for every process $p \in \mathbb{P}$ there is a unique path from p to r in the tree, we define the depth of a process p, as follows:

Definition 6.28 *depth*

$$\mathsf{depth}.\mathsf{f}.r.p.k = (r = \mathsf{iterate}.k.\mathsf{f}.p) \wedge \forall m < k : \ (r \neq \mathsf{iterate}.m.\mathsf{f}.q)$$

In words, process p has depth k, if the shortest path from p to r in the tree has length k. Since f is a function, the existence of a unique path from p to r equals the existence of a shortest path from p to r in the tree. Consequently, the requirement that for every process $p \in \mathbb{P}$ there has to be a unique path from p to r in the tree can be characterised by:

$$\forall p \in \mathbb{P} : \ \exists k : \ \mathsf{depth}.\mathsf{f}.r.p.k$$

Summarising, we have the following definition of a rooted spanning tree of a connected network:

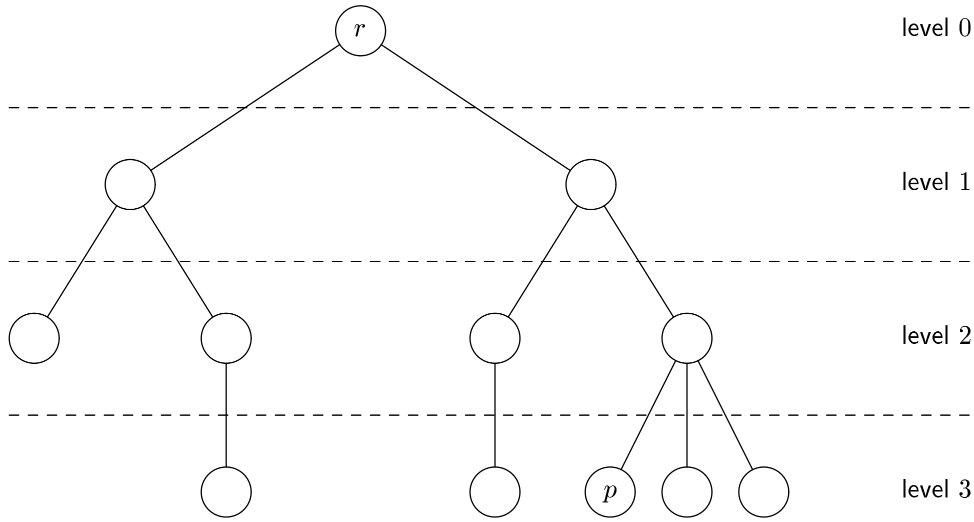


Figure 12: Processes categorised into levels.

Definition 6.29 Rooted Spanning Tree *RST*

$$\begin{array}{rcl} \mathsf{RST.f}.r.\mathbb{P}.\mathsf{neighs} & = & (r \in \mathbb{P}) \wedge (r = \mathsf{f}.r) \\ & & \forall p \in \mathbb{P} :\ (p \neq r) \Rightarrow (\mathsf{f}.p \in \mathsf{neighs}.p) \\ & & \forall p \in \mathbb{P} :\ \exists k :\ \mathsf{depth.f}.r.p.k \end{array}$$

Since every process in a rooted spanning tree has a unique depth, we can categorise processes into levels by using their depths. This is depicted in Figure 12. The set of processes at level k is defined as follows:

Theorem 6.30 *level*

$$\mathsf{level}.\mathbb{P}.\mathsf{f}.r.k = \{p \mid p \in \mathbb{P} \wedge \mathsf{depth.f}.r.p.k\}$$

When it is clear from the context which $\mathbb{P}$, f, and r are used, we shall abbreviate $\mathsf{level}.\mathbb{P}.\mathsf{f}.r.k$ by $\mathsf{level}.k$.

The height of a rooted spanning tree is defined to be the maximum of the depths of all processes in the underlying network:

Definition 6.31 Height of Tree *height*

$$\mathsf{height}.\mathbb{P}.\mathsf{f}.r.\mathsf{neighs}.h = (h = \mathsf{max}.\{k \mid p \in \mathbb{P} \wedge \mathsf{depth.f}.r.p.k\})$$

Again, when it is clear which $\mathbb{P}$, f, r, and neighs are used, we abbreviate $\mathsf{height}.\mathbb{P}.\mathsf{f}.r.\mathsf{neighs}.h$ by $\mathsf{height}.h$. The reader can check that the height of the rooted spanning tree in Figure 11 is 4. Moreover, it is not hard to see that:

Theorem 6.32 *RST_has_height*

$$\frac{\mathsf{Connected\_Network}.\mathbb{P}.\mathsf{neighs}.\mathit{starter} \wedge \mathsf{RST.f}.r.\mathbb{P}.\mathsf{neighs}}{\exists h : \mathsf{height}.\mathbb{P}.\mathsf{f}.r.\mathsf{neighs}.h}$$

6.8 Verification of the catamorphism part

$$\vdash \left.\begin{array}{l} (\forall p \in \mathbb{P} : \neg\mathsf{idle}.p) \\ \wedge\, (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) \\ \wedge\, (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p) \\ \rightsquigarrow \\ \forall p : p \in \mathbb{P} : \mathit{done}.p \end{array}\right\} \textbf{catamorphism} - \textbf{part}$$

First of all we need to construct the function $\mathsf{f} \in \mathbb{P} \to \mathbb{P}$, that characterises the rooted spanning tree. Obviously, the father variables were set as to define such a function. Consequently, we start by bringing this function f into the left hand side of $\leadsto$ as follows. In order to avoid confusion between the type of father and f we explicitly denote the state s in the last conjunct of the left hand side of $\leadsto$.

$$
\begin{array}{l}
\Leftarrow (\leadsto \textsc{Substitution } (\mathrm{C}.2_{55})) \\
\quad \vdash \quad \exists \mathsf{f} \in \mathbb{P} \to \mathbb{P} : \\
\qquad\qquad (\forall p \in \mathbb{P} : \neg \mathsf{idle}.p) \\
\qquad\qquad \wedge (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) \\
\qquad\qquad \wedge (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p) \\
\qquad\qquad \wedge (\forall p \in \mathbb{P} : (\lambda s.\ \mathsf{f}.p = (s \circ \mathsf{father}).p)) \\
\qquad\qquad \leadsto \\
\qquad\qquad \forall p : p \in \mathbb{P} : \mathit{done}.p \\
\Leftarrow (\leadsto \textsc{Disjunction } (\mathrm{C}.10_{56})) \\
\quad \forall \mathsf{f} \in \mathbb{P} \to \mathbb{P} : \\
\quad \vdash \quad (\forall p \in \mathbb{P} : \neg \mathsf{idle}.p) \\
\qquad\quad \wedge (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) \\
\qquad\quad \wedge (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p) \\
\qquad\quad \wedge (\forall p \in \mathbb{P} : (\lambda s.\ \mathsf{f}.p = (s \circ \mathsf{father}).p)) \\
\qquad\quad \leadsto \\
\qquad\quad \forall p : p \in \mathbb{P} : \mathit{done}.p
\end{array}
$$

Second, we have to prove that we have indeed built a rooted spanning tree. That is, we need to bring the conjunct $\mathsf{RST}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}$ into the left hand side of $\leadsto$. Using $\leadsto$ Substitution ($\mathrm{C}.2_{55}$) this means we have to prove that:

$$
\begin{array}{lll}
\forall s \in \mathtt{State} : & J_{\text{PLUM}}.s & \\
& \wedge\, \forall p \in \mathbb{P} : \neg\, s.(\mathsf{idle}.p) & \\
& \wedge\, \forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p.s & \\
& \wedge\, \forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p.s & \\
& \wedge\, \forall p \in \mathbb{P} : \mathsf{f}.p = (s \circ \mathsf{father}).p & \qquad (6.33) \\
& \Rightarrow & \\
& (\mathit{starter} = \mathsf{f}.\mathit{starter}) & \mathbf{P}_1 \\
& \forall p \in \mathbb{P} :\ (p \neq \mathit{starter}) \Rightarrow (\mathsf{f}.p \in \mathsf{neighs}.p) & \mathbf{P}_2 \\
& \forall p \in \mathbb{P} :\ \exists k :\ \mathsf{depth}.\mathsf{f}.\mathit{starter}.p.k & \mathbf{P}_3
\end{array}
$$

Evidently, in order to be able to prove this, we shall need to invent some new candidates for part of the invariant. The first invariant-candidate follows naturally from the proof obligation $\mathbf{P}_1$. Since, initially the *starter* is defined to be non-idle and $\mathsf{father}.\mathit{starter}$ equals[1] *starter*, the following is a valid (Theorem 6.21_{18}) invariant-candidate[2]:

$$\boxed{cJ^{6}_{\text{PLUM}} = (\lambda s.\ (s \circ \mathsf{father}).\mathit{starter} = \mathit{starter}\ \wedge\ \neg s.(\mathsf{idle}.\mathit{starter}))}$$

The next invariant-candidates are introduced as to establish proof obligation $\mathbf{P}_2$ and $\mathbf{P}_3$ respectively. Since processes only receive messages from their neighbours, and once non-idle never change the value of their father variable again, we propose:

$$\boxed{\begin{array}{ll} cJ^{7}_{\text{PLUM}} = (\lambda s.\ \forall p \in \mathbb{P} : & (p \neq \mathit{starter}) \wedge \neg s.(\mathsf{idle}.p) \\ & \Rightarrow ((s \circ \mathsf{father}).p \in \mathsf{neighs}.p)) \end{array}}$$

$$\boxed{cJ^{8}_{\text{PLUM}} = (\lambda s.\ \forall p \in \mathbb{P} :\ \neg s.(\mathsf{idle}.p) \Rightarrow \exists k :\ \mathsf{depth}.(s \circ \mathsf{father}).\mathit{starter}.p.k)}$$

It is not hard to see that these candidates are sufficient to prove 6.33.

Theorem 6.33 *all_not_idle_IMP_RST*

[1] Note that in order to be able to prove that this is invariant we need the initial condition: $\mathsf{father}.\mathit{starter} = \mathit{starter}$.

[2] Again we explicitly denote the state to avoid confusion.

For all $\mathsf{f} \in \mathbb{P} \to \mathbb{P}$, $s \in$ `State`:

$$\frac{J_{\text{PLUM}}.s \wedge (\forall p \in \mathbb{P} : \neg\, s.(\mathsf{idle}.p)) \wedge (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p.s) \quad (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p.s) \wedge (\forall p \in \mathbb{P} : \mathsf{f}.p = (s \circ \mathsf{father}).p)}{\mathsf{RST}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}}$$

For arbitrary $\mathsf{f} \in \mathbb{P} \to \mathbb{P}$, we now proceed with the catamorphism part as follows:

$$\begin{array}{ll} \vdash & (\forall p \in \mathbb{P} : \neg \mathsf{idle}.p) \\ & \wedge\, (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) \\ & \wedge\, (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p) \\ & \wedge\, (\forall p \in \mathbb{P} : (\lambda s.\ \mathsf{f}.p = (s \circ \mathsf{father}).p)) \\ & \leadsto \\ & \forall p : p \in \mathbb{P} : \mathit{done}.p \end{array}$$

$\Leftarrow$($\leadsto$ Substitution (C.2_{55}), using Theorem 6.33_{24})[3]

$$\begin{array}{ll} \vdash & (\forall p \in \mathbb{P} : \neg \mathsf{idle}.p) \\ & \wedge\, (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) \\ & \wedge\, (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p) \\ & \wedge\, (\forall p \in \mathbb{P} : (\lambda s.\ \mathsf{f}.p = (s \circ \mathsf{father}).p)) \\ & \wedge\, (\lambda s.\ \mathsf{RST}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}) \\ & \leadsto \\ & \forall p : p \in \mathbb{P} : \mathit{done}.p \end{array}$$

$\Leftarrow$($\leadsto$ Stable Shift (C.9_{56}))

$$\begin{array}{lll} (\forall p \in \mathbb{P} : \neg \mathsf{idle}.p) & & \\ \wedge\, (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) & & \\ \wedge\, (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p) & & \\ \wedge\, (\forall p \in \mathbb{P} : (\lambda s.\ \mathsf{f}.p = (s \circ \mathsf{father}).p)) & & \\ \wedge\, (\lambda s.\ \mathsf{RST}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}) & \vdash & \mathsf{true} \\ & & \leadsto \\ & & \forall p : p \in \mathbb{P} : \mathit{done}.p \end{array}$$

Before continuing with this proof obligation, it shall be clear that we need to do something about its readability. For this we introduce the following definition, which contains all conjuncts located at the left hand side of $\vdash$ (including J_{PLUM}, which is there implicitly (Section 6.3)). We call it J_{ana} since it refers to properties that were established during the anamorphism part.

Definition 6.34 *Invar_and_ANA*

$$\begin{array}{lll} J_{\mathsf{ana}} & = & J_{\text{PLUM}} \\ & & \wedge\, (\forall p \in \mathbb{P} : \neg \mathsf{idle}.p) \\ & & \wedge\, (\forall p \in \mathbb{P} : \mathit{sent\_to\_all\_non\_fathers}.p) \\ & & \wedge\, (\forall p \in \mathbb{P} : \mathit{rec\_from\_all\_non\_children}.p) \\ & & \wedge\, (\forall p \in \mathbb{P} : (\lambda s.\ \mathsf{f}.p = (s \circ \mathsf{father}).p)) \\ & & \wedge\, (\lambda s.\ \mathsf{RST}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}) \end{array}$$

Using $\circlearrowright$ Conjunction (A.11_{53}), 6.15_{16}, 6.24_{20}, 6.21_{18}, and the assumed validity of J_{PLUM}, we can derive:

Theorem 6.35 *STABLe_Invar_and_ANA*

$${}_{\text{PLUM}}\vdash \circlearrowright J_{\mathsf{ana}}$$

This reduces our current proof obligation to:

$$J_{\mathsf{ana}} \vdash \mathsf{true} \leadsto \forall p \in \mathbb{P} : \mathit{done}.p$$

[3] Note that RST is *not* a state-predicate. We have `State`-lifted it by enclosing it in between $(\lambda s.\ \ldots)$.

Now we can proceed with the proof strategy presented in Section 6.5; that is prove that the required information flows from the leaves to the root of the rooted spanning tree. In the case of proving termination this comes down to proving that when the leaves of the RST are *done*, then eventually all the processes will be *done*. From Theorem 6.32_{23} we can deduce the height h of the RST, and consequently we know that the leaves of the RST equal the processes at level h. Therefore we decompose our proof obligation as follows:

$$
\begin{array}{l}
\quad J_{\mathsf{ana}} \vdash \mathsf{true} \rightsquigarrow \forall p \in \mathbb{P} : \mathit{done.p} \\
\Leftarrow(\rightsquigarrow \textsc{Substitution}\ (\mathrm{C}.2_{55}),\ \text{Definition } 6.34_{25},\ \text{and Theorem } 6.32_{23}) \\
\quad J_{\mathsf{ana}} \vdash (\exists h.\mathsf{height}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}.h) \rightsquigarrow \forall p \in \mathbb{P} : \mathit{done.p} \\
\Leftarrow(\rightsquigarrow \textsc{Disjunction}\ (\mathrm{C}.10_{56})) \\
\quad \forall h :\ J_{\mathsf{ana}} \vdash \mathsf{height}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}.h \rightsquigarrow \forall p \in \mathbb{P} : \mathit{done.p} \\
\Leftarrow(\rightsquigarrow \textsc{Transitivity}\ (\mathrm{C}.5_{55})) \\
\quad \underbrace{\forall h :\ J_{\mathsf{ana}} \vdash \mathsf{height}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}.h \rightsquigarrow \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{done.p}}_{\textbf{cata\_1}} \\
\quad \wedge \\
\quad \underbrace{\forall h :\ J_{\mathsf{ana}} \vdash \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{done.p} \rightsquigarrow \forall p \in \mathbb{P} : \mathit{done.p}}_{\textbf{cata\_2}}
\end{array}
$$

Verification of cata_1

Since leaves have no descendants (i.e. children), and J_{ana} states that:

- all processes have received messages from all their non-child-neighbours
- all processes have sent messages to all their non-father-neighbours

we can prove that the leaves (i.e. the processes at level h in a RST of height h) have finished their collecting and propagating phases:

Theorem 6.36 *height_Invar_IMP_leaves_finished*

$$
\frac{J_{\mathsf{ana}} \wedge \mathsf{height}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}.h}{\forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{finished\_collecting\_and\_propagating.p}}
$$

Consequently, we can proceed with **cata_1** as follows:

$$
\begin{array}{l}
\quad \forall h :\ J_{\mathsf{ana}} \vdash \mathsf{height}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}.h \rightsquigarrow \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{done.p} \\
\Leftarrow(\rightsquigarrow \textsc{Substitution}\ (\mathrm{C}.2_{55}),\ \text{using Theorem } 6.36_{26}) \\
\quad \forall h :\ J_{\mathsf{ana}} \vdash \quad \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{finished\_collecting\_and\_propagating.p} \\
\qquad\qquad\qquad \rightsquigarrow \\
\qquad\qquad\qquad \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{done.p} \\
\Leftarrow(\rightsquigarrow \textsc{Conjunction}\ (\mathrm{C}.11_{56})) \\
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \\
\qquad J_{\mathsf{ana}} \vdash \mathit{finished\_collecting\_and\_propagating.p} \ \rightsquigarrow\ \mathit{done.p}
\end{array}
$$

Since the *rec_from_all_neighs* part of the *done* predicate (see $(7)_8$) was already established by the validity of *finished_collecting_and_propagating* (see $(4)_8$), we continue as follows:

$$
\begin{array}{l}
\Leftarrow((7)_8 \text{ and } (4)_8) \\
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \\
\qquad J_{\mathsf{ana}} \vdash \quad \mathit{rec\_from\_all\_neighs.p} \wedge \mathit{finished\_collecting\_and\_propagating.p} \\
\qquad\qquad \rightsquigarrow \\
\qquad\qquad \mathit{rec\_from\_all\_neighs.p} \wedge \mathit{sent\_to\_all\_neighs.p} \\
\Leftarrow(\rightsquigarrow \textsc{Conjunction}\ (\mathrm{C}.11_{56})) \\
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \\
\qquad J_{\mathsf{ana}} \vdash \mathit{rec\_from\_all\_neighs.p} \rightsquigarrow \mathit{rec\_from\_all\_neighs.p} \\
\quad \wedge \\
\qquad J_{\mathsf{ana}} \vdash \mathit{finished\_collecting\_and\_propagating.p} \rightsquigarrow \mathit{sent\_to\_all\_neighs.p}
\end{array}
$$

The first conjunct can easily be proved by $\rightsquigarrow$ REFLEXIVITY (C.4_{55}), $\circlearrowright$ CONJUNCTIVITY (A.11_{53}), and Theorem 6.24_{20}.

For the second conjunct, we argue as follows. When a *follower* process has finished its collecting and propagating phase, it is ready to sent its final message to its father after which it becomes *done* and hence has *sent_to_all_neighs*. However, when the *starter* has *finished_collecting_and_propagating*, and hence *sent_to_all_non_fathers*, it has already *sent_to_all_neighs*, since cJ^{6}_{PLUM} states that the father of the *starter* is the *starter* itself; and the definition of Network (Definition 4.1_{6}) defines that a process cannot be a neighbour of itself.

Theorem 6.37 *sent_2_all_except_f_starter_IMP_sent_2_all_neighs_starter*

$$\frac{J_{\text{PLUM}} \wedge \mathit{sent\_to\_all\_non\_fathers.starter}}{\mathit{sent\_to\_all\_neighs.starter}}$$

Consequently, we make the following case distinction: (note that this is a case distinction on the outermost level, *not* inside $\vdash$ using $\rightsquigarrow$ CASE DISTINCTION (C.6_{55}))

$$\begin{array}{l}
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h):\\
\quad\quad J_{\mathsf{ana}} \vdash \mathit{finished_collecting_and_propagating.p} \rightsquigarrow \mathit{sent_to_all_neighs.p}\\
\Leftarrow ((p = \mathit{starter}) \vee (p \neq \mathit{starter}))\\
\quad\quad J_{\mathsf{ana}} \vdash \mathit{finished_collecting_and_propagating.starter} \rightsquigarrow \mathit{sent_to_all_neighs.starter}\\
\quad \wedge\\
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h), p \neq \mathit{starter}:\\
\quad\quad J_{\mathsf{ana}} \vdash \mathit{finished_collecting_and_propagating.p} \rightsquigarrow \mathit{sent_to_all_neighs.p}
\end{array}$$

Evidently, the first conjunct can be proved by $\rightsquigarrow$ INTRODUCTION (C.3_{55}), using Theorem 6.37_{27}, Theorem 6.15_{16}, and (4_{8}). We carry on with the second conjunct by noticing that when a process has *finished_collecting_and_propagating*, it has already sent a message to its father or not.

$$\begin{array}{l}
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h), p \neq \mathit{starter}:\\
\quad\quad J_{\mathsf{ana}} \vdash \mathit{finished_collecting_and_propagating.p} \rightsquigarrow \mathit{sent_to_all_neighs.p}\\
\Leftarrow (\rightsquigarrow \text{CASE DISTINCTION (C.}6_{55}))\\
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h), p \neq \mathit{starter}:\\
\quad\quad J_{\mathsf{ana}} \vdash \quad \mathit{finished_collecting_and_propagating.p} \wedge \mathit{reported_to_father.p}\\
\quad\quad\quad\quad \rightsquigarrow\\
\quad\quad\quad\quad \mathit{sent_to_all_neighs.p}\\
\quad \wedge\\
\quad\quad J_{\mathsf{ana}} \vdash \quad \mathit{finished_collecting_and_propagating.p} \wedge \neg\mathit{reported_to_father.p}\\
\quad\quad\quad\quad \rightsquigarrow\\
\quad\quad\quad\quad \mathit{sent_to_all_neighs.p}
\end{array}$$

The first conjunct can again be easily proved by $\rightsquigarrow$ INTRODUCTION (C.3_{55}), using Theorem 6.12_{13}, and Theorem 6.15_{16}.

Progress stated in the second conjunct is ensured by the DONE action of process p. Consequently:

$$\begin{array}{l}
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h), p \neq \mathit{starter}:\\
\quad\quad J_{\mathsf{ana}} \vdash \quad \mathit{finished_collecting_and_propagating.p} \wedge \neg\mathit{reported_to_father.p}\\
\quad\quad\quad\quad \rightsquigarrow\\
\quad\quad\quad\quad \mathit{sent_to_all_neighs.p}\\
\Leftarrow (\rightsquigarrow \text{SUBSTITUTION (C.}2_{55}\text{), to recognise guard of DONE})\\
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h), p \neq \mathit{starter}:\\
\quad\quad J_{\mathsf{ana}} \vdash \quad \exists q \in \mathsf{neighs}.p: \quad \mathit{finished_collecting_and_propagating.p}\\
\quad\quad\quad\quad\quad\quad \wedge\neg\mathit{reported_to_father.p} \wedge (q = \mathsf{father}.p)\\
\quad\quad\quad\quad \rightsquigarrow\\
\quad\quad\quad\quad \exists q \in \mathsf{neighs}.p: \quad \mathit{sent_to_all_neighs.p}\\
\Leftarrow (\rightsquigarrow \text{DISJUNCTION (C.}10_{56}))\\
\quad \forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h), p \neq \mathit{starter}, q \in \mathsf{neighs}.p:
\end{array}$$

$$J_{\mathsf{ana}} \vdash \quad \begin{array}{l} \mathit{finished\_collecting\_and\_propagating.p} \wedge \neg \mathit{reported\_to\_father.p} \\ \wedge\ (q = \mathsf{father}.p) \\ \leadsto \\ \mathit{sent\_to\_all\_neighs.p} \end{array}$$

$\Leftarrow$($\leadsto$ INTRODUCTION (C.3_{55}), Theorem 6.15_{16})

$\forall h, p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter.h}), p \neq \mathit{starter}, q \in \mathsf{neighs}.p$:

$$\vdash \quad \begin{array}{l} J_{\mathsf{ana}} \wedge \mathit{finished\_collecting\_and\_propagating.p} \wedge \neg \mathit{reported\_to\_father.p} \\ \wedge\ (q = \mathsf{father}.p) \\ \ \mathsf{ensures} \\ \mathit{sent\_to\_all\_neighs.p} \end{array}$$

As the reader can verify, this ensures -property can be easily proved. This ends the verification of:

Theorem 6.38 *finished_collecting_and_propagating_CON_done*

$$\forall h:\ J_{\mathsf{ana}} \vdash \quad \begin{array}{l} \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter.h}) : \mathit{finished\_collecting\_and\_propagating.p} \\ \leadsto \\ \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter.h}) : \mathit{done.p} \end{array}$$

and consequently, of **cata_1**:

Theorem 6.39 cata_1 *height_h_CON_all_done_at_height_h*

$$\forall h:\ J_{\mathsf{ana}} \vdash \mathsf{height}.\mathbb{P}.\mathsf{f}.\mathit{starter}.\mathsf{neighs}.h \leadsto \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter.h}) : \mathit{done.p}$$

Verification of cata_2

The proof of **cata_2** proceeds by induction on h.

INDUCTION BASE: case 0

$$J_{\mathsf{ana}} \vdash \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.0) : \mathit{done.p} \leadsto \forall p \in \mathbb{P} : \mathit{done.p}$$

INDUCTION HYPOTHESIS:

$$\forall h:\ J_{\mathsf{ana}} \vdash \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter.h}) : \mathit{done.p} \leadsto \forall p \in \mathbb{P} : \mathit{done.p}$$

INDUCTION STEP: case $(h+1)$

$$J_{\mathsf{ana}} \vdash \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.(h+1)) : \mathit{done.p} \leadsto \forall p \in \mathbb{P} : \mathit{done.p}$$

proof of INDUCTION BASE

Since, the only process residing at $\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.0$ is the starter, and the *starter* can only be *done* when all other processes are *done*, the INDUCTION BASE can be proved by $\leadsto$ INTRODUCTION (C.3_{55}) as follows:

$J_{\mathsf{ana}} \vdash \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.0) : \mathit{done.p} \leadsto \forall p \in \mathbb{P} : \mathit{done.p}$

$\Leftarrow$($\leadsto$ SUBSTITUTION (C.2_{55}), Definition 6.30_{23})

$J_{\mathsf{ana}} \vdash \mathit{done.starter} \leadsto \forall p \in \mathbb{P} : \mathit{done.p}$

$\Leftarrow$($\leadsto$ INTRODUCTION (C.3_{55}))

$\vdash\ \circlearrowright(J_{\mathsf{ana}} \wedge \forall p \in \mathbb{P} : \mathit{done.p})$

$\wedge$

$\forall s \in \mathtt{State}.J_{\mathsf{ana}}.s \wedge \mathit{done.starter.s} \Rightarrow \forall p \in \mathbb{P} : \mathit{done.p.s}$

The stability predicate can be proved by $\circlearrowright$CONJUNCTION (A.11_{53}), using Theorem 6.15_{16}, Theorem 6.24_{20}, and 6.35_{25}. To prove the second conjunct, assume for arbitrary states s:

$\mathbf{A}_1$: $J_{\mathsf{ana}}.s$

$\mathbf{A}_2$: $\mathit{done.starter.s}$

$\mathbf{A}_3$: $p \in \mathbb{P}$

We prove $\mathit{done.p.s}$ by contradiction, by assuming that:

$\mathbf{A}_4 : \neg done.p.s$

and proving that $\neg done.starter.s$, which establishes false with $\mathbf{A}_2$.

The proof strategy will be the following. Since process p is not *done*, we know that is has not yet sent a message to its father. Consequently, p's father has not yet received a message from p, and hence cannot be done. Iterating this argument until the father of the process under consideration is the *starter*, will establish the proof.

However, in order to apply this strategy, we shall have to introduce two new invariant-candidates since, as the reader can verify, the ones introduced until now do not suffice. We propose:

$$\boxed{cJ^9_{\text{PLUM}} = \forall p, q \in \mathbb{P} : \neg(\mathsf{idle}.p) \wedge \neg done.p \wedge (q = \mathsf{father}.p) \Rightarrow \mathsf{nr\_sent}.p.q = 0}$$

So we can deduce that when a process p is not done, it has not yet sent a message to its father. Furthermore, we propose the invariant-candidate that states that the number of messages a process q has received from p is always less than or equal to the number of messages p has sent to q:

$$\boxed{cJ^{10}_{\text{PLUM}} = \forall p, q \in \mathbb{P} : \mathsf{nr\_rec}.q.p \leq \mathsf{nr\_sent}.p.q}$$

So we can deduce that when p has not yet sent a message to some neighbour q, q has not yet received a message from p. When a process q still has neighbours p from which it has not received a message (i.e. it holds that $\mathsf{nr\_rec}.q.p = 0$), we can prove (using cJ^5_{PLUM}) that q has not *rec_from_all_neighs* and hence is not *done*. Consequently, equipped with the new invariant-candidates proposed above, we can now prove that when p is not *done*, neither is its father:

Theorem 6.40 *not_done_IMP_f_not_done*

For all states $s \in \mathtt{State}$:

$$\frac{J_{\text{PLUM}}.s \wedge p \in \mathbb{P} \wedge \neg s.(\mathsf{idle}.p) \wedge \neg done.p.s \wedge (q = (s \circ \mathsf{father}).p)}{\neg done.q.s}$$

Subsequently, by induction we can prove that:

Theorem 6.41 *not_done_IMP_iterate_f_not_done*

For all states $s \in \mathtt{State}$:

$$\frac{J_{\text{PLUM}}.s \wedge p \in \mathbb{P} \wedge \neg s.(\mathsf{idle}.p) \wedge \neg done.p.s}{\forall m, q : (q = \mathsf{iterate}.m.(s \circ \mathsf{father}).p) \Rightarrow \neg done.q.s}$$

Consequently, using invariant-part cJ^8_{PLUM} we can prove that:

Theorem 6.42 *not_done_IMP_starter_not_done*

For all states $s \in \mathtt{State}$:

$$\frac{J_{\text{PLUM}}.s \wedge p \in \mathbb{P} \wedge \neg s.(\mathsf{idle}.p) \wedge \neg done.p.s}{\neg done.starter.s}$$

Assumptions $\mathbf{A}_1$, $\mathbf{A}_3$, $\mathbf{A}_4$, Theorem 6.42_{29}, and the characterisation of J_{ana} (Definition 6.34_{25}) now establish that $\neg done.starter.s$.

end of proof Induction Base

proof of Induction Step

$$J_{\mathsf{ana}} \vdash \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.starter.(h+1)) : done.p \leadsto \forall p \in \mathbb{P} : done.p$$
$\Leftarrow$($\leadsto$ Transitivity ($C.5_{55}$), and Induction Hypothesis)
$$J_{\mathsf{ana}} \vdash \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.starter.(h+1)) : done.p \leadsto \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.starter.h) : done.p$$

The intuitive idea behind the proof strategy for this last proof obligation is the following: because processes at level $(h+1)$ are done, these have sent messages to their fathers who all reside at level h; eventually

$cJ^1_{\text{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \neg\mathsf{idle}.p \wedge q = \mathsf{father}.p \Rightarrow \neg\mathsf{idle}.q$

$cJ^2_{\text{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \mathsf{nr\_sent}.p.q = 0 \;\vee\; \mathsf{nr\_sent}.p.q = 1$

$cJ^3_{\text{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \mathsf{idle}.p \Rightarrow \mathsf{nr\_rec}.p.q = 0$

$cJ^4_{\text{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : (\mathsf{nr\_rec}.q.p < \mathsf{nr\_sent}.p.q) = \mathsf{mit}.p.q$

$cJ^5_{\text{PLUM}} = \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : (\mathsf{nr\_rec}.p.q = 0) \vee (\mathsf{nr\_rec}.p.q = 1)$

$cJ^6_{\text{PLUM}} = (\lambda s.\ (s \circ \mathsf{father}).\mathit{starter} = \mathit{starter} \;\wedge\; \neg s.(\mathsf{idle}.\mathit{starter}))$

$cJ^7_{\text{PLUM}} = (\lambda s.\ \forall p \in \mathbb{P} :\ (p \neq \mathit{starter}) \wedge \neg s.(\mathsf{idle}.p) \Rightarrow ((s \circ \mathsf{father}).p \in \mathsf{neighs}.p))$

$cJ^8_{\text{PLUM}} = (\lambda s.\ \forall p \in \mathbb{P} :\ \neg s.(\mathsf{idle}.p) \Rightarrow \exists k :\ \mathsf{depth}.(s \circ \mathsf{father}).\mathit{starter}.p.k)$

$cJ^9_{\text{PLUM}} = \forall p, q \in \mathbb{P} : \neg(\mathsf{idle}.p) \wedge \neg \mathit{done}.p \wedge (q = \mathsf{father}.p) \Rightarrow \mathsf{nr\_sent}.p.q = 0)$

$cJ^{10}_{\text{PLUM}} = \forall p, q \in \mathbb{P} : \mathsf{nr\_rec}.q.p \leq \mathsf{nr\_sent}.p.q$

Figure 13: Invariant-candidates proposed during refinement and decomposition

all processes at $\mathsf{level}\ h$ shall receive these messages and (since already having *sent_to_all_non_fathers* and *rec_from_all_non_children* (J_{ana})) will have *finished_collecting_and_propagating*; consequently, all processes at $\mathsf{level}\ h$ will eventually send a message to their father and become *done*.

$J_{\mathsf{ana}} \vdash \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.(h+1)) : \mathit{done}.p \rightsquigarrow \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{done}.p$

$\Leftarrow$($\rightsquigarrow$ TRANSITIVITY (C.5_{55}))

$J_{\mathsf{ana}} \vdash \quad \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.(h+1)) : \mathit{done}.p$
$\rightsquigarrow$
$\forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{finished\_collecting\_and\_propagating}.p$

$\wedge$

$J_{\mathsf{ana}} \vdash \quad \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{finished\_collecting\_and\_propagating}.p$
$\rightsquigarrow$
$\forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{done}.p$

$\Leftarrow$(The second conjunct is proved by Theorem 6.38_{28})

$J_{\mathsf{ana}} \vdash \quad \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.(h+1)) : \mathit{done}.p$
$\rightsquigarrow$
$\forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.h) : \mathit{finished\_collecting\_and\_propagating}.p$

$\Leftarrow$($\rightsquigarrow$ SUBSTITUTION (C.2_{55}), and $(4)_8$, $(7)_8$, 6.34_{25}, 6.30_{23})

$J_{\mathsf{ana}} \vdash \quad \forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.(h+1)), q \in \mathsf{neighs}.p : \mathsf{nr\_sent}.p.q = 1 \wedge \neg\mathsf{idle}.q$
$\rightsquigarrow$
$\forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.(h+1)), q \in \mathsf{neighs}.p : \mathsf{nr\_rec}.q.p = 1$

$\Leftarrow$($\rightsquigarrow$ CONJUNCTION (C.11_{56}), twice)

$\forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.(h+1)), q \in \mathsf{neighs}.p :$
$J_{\mathsf{ana}} \vdash \mathsf{nr\_sent}.p.q = 1 \wedge \neg\mathsf{idle}.q \rightsquigarrow \mathsf{nr\_rec}.q.p = 1$

$\Leftarrow$($\rightsquigarrow$ STABLE STRENGTHENING (C.8_{56}), Definition 6.34_{25}, and Theorem 6.35_{25})

$\forall p \in (\mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.(h+1)), q \in \mathsf{neighs}.p :$
$J_{\text{PLUM}} \vdash \mathsf{nr\_sent}.p.q = 1 \wedge \neg\mathsf{idle}.q \rightsquigarrow \mathsf{nr\_rec}.q.p = 1$

Since $p \in \mathsf{level}.\mathbb{P}.\mathsf{f}.\mathit{starter}.(h+1)$, implies $p \in \mathbb{P}$, Theorem 6.25_{21} establishes the INDUCTION STEP.

end of proof INDUCTION STEP

Definition 6.43 PLUM'S INVARIANT *Invariant_DEF*

$J_{\text{PLUM}} =$

$$
\begin{array}{lr}
\quad \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \neg\mathsf{idle}.p \wedge q = \mathsf{father}.p \Rightarrow \neg\mathsf{idle}.q & cJ^1_{\text{PLUM}} \\
\wedge\ \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \mathsf{nr\_sent}.p.q = 0 \ \vee\ \mathsf{nr\_sent}.p.q = 1 & cJ^2_{\text{PLUM}} \\
\wedge\ \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \mathsf{idle}.p \Rightarrow \mathsf{nr\_rec}.p.q = 0 & cJ^3_{\text{PLUM}} \\
\wedge\ \forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : (\mathsf{nr\_rec}.q.p < \mathsf{nr\_sent}.p.q) = \mathsf{mit}.p.q & cJ^4_{\text{PLUM}} \\
\wedge\ \mathsf{father}.starter = starter \ \wedge\ \neg(\mathsf{idle}.starter) & cJ^6_{\text{PLUM}} \\
\wedge\ \forall p \in \mathbb{P} :\ (p \neq starter) \wedge \neg(\mathsf{idle}.p) \Rightarrow (\mathsf{father}.p \in \mathsf{neighs}.p) & cJ^7_{\text{PLUM}} \\
\wedge\ (\lambda s.\ \forall p \in \mathbb{P} :\ \neg s.(\mathsf{idle}.p) \Rightarrow \exists k :\ \mathsf{depth}.(s \circ \mathsf{father}).starter.p.k) & cJ^8_{\text{PLUM}} \\
\wedge\ \forall p, q \in \mathbb{P} : \neg(\mathsf{idle}.p) \wedge \neg done.p \wedge (q = \mathsf{father}.p) \Rightarrow \mathsf{nr\_sent}.p.q = 0 & cJ^9_{\text{PLUM}} \\
\wedge\ \forall p, q \in \mathbb{P} : \mathsf{nr\_rec}.q.p \leq \mathsf{nr\_sent}.p.q & cJ^{10}_{\text{PLUM}} \\
\wedge\ \forall p, q \in \mathbb{P} : \mathsf{M}.p.q = [] \vee (\exists x : \mathsf{M}.p.q = [x]) & cJ^{11}_{\text{PLUM}} \\
\wedge\ \forall p, q \in \mathbb{P} : \mathsf{idle}.p \Rightarrow \mathsf{nr\_sent}.p.q = 0 & cJ^{12}_{\text{PLUM}}
\end{array}
$$

Theorem 6.44 *STABLEe_Invariant*

$$_{\text{PLUM}}\vdash \circlearrowleft J_{\text{PLUM}}$$

Theorem 6.45 *INVe_Invariant*

$$_{\text{PLUM}}\vdash \ \Box J_{\text{PLUM}}$$

Figure 14: PLUM's invariant

6.9 Construction of the invariant

As indicated in Section 6.1 the invariant J_{PLUM} is constructed such that it implies all the candidates that were proposed during the process of refinement and decomposition. All the proposed candidates are collected in Figure 13. Finding the minimal invariant is now like a nice puzzle. In order to solve this puzzle, we shall start by analysing the different candidates. The first thing we notice is that:

$$cJ^2_{\text{PLUM}} \wedge cJ^{10}_{\text{PLUM}} \ \Rightarrow\ cJ^5_{\text{PLUM}}$$

Consequently, aiming for minimality, cJ^5_{PLUM} can be dropped. Subsequently, we shall start verifying the stability of the conjunction of the remaining candidates. That is, we verify that:

$$\vdash\ \circlearrowleft cJ^1_{\text{PLUM}} \ \wedge\ cJ^2_{\text{PLUM}} \ \wedge\ cJ^3_{\text{PLUM}} \ \wedge\ cJ^4_{\text{PLUM}} \ \wedge\ cJ^6_{\text{PLUM}} \wedge\ cJ^7_{\text{PLUM}} \ \wedge\ cJ^8_{\text{PLUM}} \ \wedge\ cJ^9_{\text{PLUM}} \ \wedge cJ^{10}_{\text{PLUM}}$$

During these verification activities, two more invariant-candidates had to be proposed. One, – cJ^{11}_{PLUM} below – had to be introduced to prove the stability of cJ^4_{PLUM}; and another – cJ^{12}_{PLUM} below – was needed in order to prove the stability of cJ^8_{PLUM} and cJ^9_{PLUM}. Since, the verification activities are straightforward we shall not describe them here, and just state the two invariant-candidates:

$$\boxed{cJ^{11}_{\text{PLUM}} = \forall p, q \in \mathbb{P} : \mathsf{M}.p.q = [] \vee (\exists x : \mathsf{M}.p.q = [x])}$$

Stating that, on every communication channel there is no message in transit , or precisely one.

$$\boxed{cJ^{12}_{\text{PLUM}} = \forall p, q \in \mathbb{P} : \mathsf{idle}.p \Rightarrow \mathsf{nr\_sent}.p.q = 0}$$

Stating that idle processes have not yet sent messages to their neighbours.

Finally, we construct our invariant consisting of the conjunction of: cJ^1_{PLUM} through cJ^{12}_{PLUM} with the exception of cJ^5_{PLUM}. The resulting definition, together with the theorems stating stability and invariance

Theorem 7.3 *guard_of_IDLE_ECHO*

$\mathsf{guard\_of}.(\textsc{idle}_{\textsc{echo}}.p.q) = \mathsf{guard\_of}.(\textsc{idle}.p.q)$

Theorem 7.4 *guard_of_COL_ECHO*

$\mathsf{guard\_of}.(\textsc{col}_{\textsc{echo}}.p.q) = \mathsf{guard\_of}.(\textsc{col}.p.q) \wedge \mathit{sent\_to\_all\_non\_fathers}.p$

Theorem 7.5 *guard_of_PROP_ECHO*

$\mathsf{guard\_of}.(\textsc{prop}_{\textsc{echo}}.p.q) = \mathsf{guard\_of}.(\textsc{prop}.p.q)$

Theorem 7.6 *guard_of_DONE_ECHO*

$\mathsf{guard\_of}.(\textsc{done}_{\textsc{echo}}.p.q) = \mathsf{guard\_of}.(\textsc{done}.p.q)$

Figure 15: Guards of the actions from ECHO

in PLUM are in Figure 14_{31}. In the characterisation of J_{PLUM} (Definition 6.43_{31}), all logical operators, except for those in cJ^{8}_{PLUM}, are overloaded to denote their `State`-lifted versions.

7 Using refinements to derive termination of ECHO

This section shall describe how termination of the ECHO algorithm is proved using the refinements framework from [VS01] summarized in Section 3, and the already proved fact that:

$$\forall J :: \text{PLUM} \sqsubseteq_{\mathcal{R}\_\text{PLUM\_ECHO},\, J} \text{ECHO}$$

The UNITY specification reads:

Theorem 7.1 *HYLO_ECHO*

$$J_{\text{PLUM}} \wedge J_{\text{ECHO}} \;\; {}_{\text{ECHO}}\vdash \textbf{ini}\text{ECHO} \rightsquigarrow \forall p : p \in \mathbb{P} : \mathit{done}.p$$

where invariant J_{ECHO} captures additional safety properties for ECHO (if any). Again, J_{ECHO} shall, if necessary, be constructed incrementally in a demand-driven way following the conventions described in Section 6.1.

Using $\circlearrowright$ Preservation Theorem 3.8_{4}, it is straightforward to derive that J_{PLUM} is also (Theorem 6.44_{31}) a stable predicate in ECHO.

Theorem 7.2 *STABLEe_Invariant_in_ECHO*

$${}_{\text{ECHO}}\vdash \circlearrowright J_{\text{PLUM}}$$

The stability of: ${}_{\text{ECHO}}\vdash \circlearrowright J_{\text{PLUM}} \wedge J_{\text{ECHO}}$ will be implicitly assumed throughout the verification process, and verified when the precise characterisation of J_{ECHO} has been established. For ease of reference, Figure 15 displays theorems about the guards of ECHO's actions. For readability we introduce the notational convention that $\vdash$ abbreviates $J_{\text{PLUM}} \wedge J_{\text{ECHO}} \;\; {}_{\text{ECHO}}\vdash$.

Termination of ECHO is proved using the property preserving Theorem 3.6_{5}.

$${}_{\text{ECHO}}\vdash \textbf{ini}\text{ECHO} \rightsquigarrow \forall p : p \in \mathbb{P} : \mathit{done}.p$$

$\Leftarrow$(Theorem 3.6_{5}, 6.1_{11}, 5.1_{10})

$$\exists W :: (\mathbf{w}\mathrm{ECHO} = \mathbf{w}\mathrm{PLUM} \cup W) \wedge (J_{\text{PLUM}}\ \mathcal{C}\ W^{\mathsf{c}}) \wedge (\mathbf{w}\mathrm{PLUM} \subseteq W^{\mathsf{c}})$$
$$\wedge$$
$$\forall A_P\ A_E\ :\ A_P \in \mathbf{a}\mathrm{PLUM}\ \wedge\ A_P\ \mathcal{R}_{\text{-PLUM\_ECHO}}\ A_E\ :\ {}_{\text{ECHO}}\vdash \mathsf{guard\_of}.A_P \rightarrowtail \mathsf{guard\_of}.A_E$$
$$\wedge$$
$$\forall A_P\ A_E\ :\ A_P \in \mathbf{a}\mathrm{PLUM}\ \wedge\ A_P\ \mathcal{R}_{\text{-PLUM\_ECHO}}\ A_E\ :\ {}_{\text{ECHO}}\vdash (J_{\text{PLUM}} \wedge J_{\text{ECHO}} \wedge \mathsf{guard\_of}.A_E)\ \mathsf{unless}\ \neg(\mathsf{guard\_of}.A_P)$$

Since no variables are superimposed on PLUM in order to construct ECHO, the first conjunct can be proved by instantiation with $\emptyset$. Subsequently, using:

- the characterisation of $\mathcal{R}_{\text{-PLUM\_ECHO}}$ (Figure 5_{10})
- the Theorems from Figure 15_{32}, stating that the guards of the $\text{IDLE}_{\text{ECHO}}$, $\text{PROP}_{\text{ECHO}}$, and $\text{DONE}_{\text{ECHO}}$ actions are equal to those of PLUM
- anti-reflexivity of unless (Theorem A.10_{53})
- reflexivity of $\rightarrowtail$ (Theorem B.4_{54})
- the implicit assumption stating stability of $(J_{\text{PLUM}} \wedge J_{\text{ECHO}})$

we can reduce the second and the third conjunct to:

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :$$
$$\left.{}_{\text{ECHO}}\vdash \mathsf{guard\_of}.\text{COL}.p.q \rightarrowtail \mathsf{guard\_of}.\text{COL}_{\text{ECHO}}.p.q\right\}\ \mathbf{reach-part}$$
$$\wedge$$
$$\left.{}_{\text{ECHO}}\vdash J_{\text{PLUM}} \wedge J_{\text{ECHO}} \wedge \mathsf{guard\_of}.\text{COL}_{\text{ECHO}}.p.q\ \mathsf{unless}\ \neg\mathsf{guard\_of}.\text{COL}.p.q\right\}\ \mathbf{unless-part}$$

The **unless**-part is not hard to verify and will be left up to the enthusiastic reader. In order to prove it, the current conjuncts from J_{PLUM} suffice, and hence no additional safety properties have to be added to J_{ECHO}.

The proof of the **reach-part** proceeds by rewriting with Theorem 6.7_{11} and 7.4_{32}:

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :$$
$${}_{\text{ECHO}}\vdash\ \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \rightarrowtail \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \wedge \mathit{sent\_to\_all\_non\_fathers}.p$$

$\Leftarrow$($\rightarrowtail$ Case distinction (B.6_{55}))

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :$$
$${}_{\text{ECHO}}\vdash\ \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \wedge \mathit{sent\_to\_all\_non\_fathers}.p \rightarrowtail \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \wedge \mathit{sent\_to\_all\_non\_fathers}.p$$
$$\wedge$$
$${}_{\text{ECHO}}\vdash\ \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \wedge \neg\mathit{sent\_to\_all\_non\_fathers}.p \rightarrowtail \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \wedge \mathit{sent\_to\_all\_non\_fathers}.p$$

$\Leftarrow$($\rightarrowtail$ Reflexivity (B.4_{54}) proves the first conjunct)

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :$$
$${}_{\text{ECHO}}\vdash\ \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \wedge \neg\mathit{sent\_to\_all\_non\_fathers}.p \rightarrowtail \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \wedge \mathit{sent\_to\_all\_non\_fathers}.p$$

$\Leftarrow$($\rightarrowtail$ Substitution (B.2_{54}), to bring into correct form for $\rightarrowtail$ PSP (B.8_{55}))

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p :$$
$${}_{\text{ECHO}}\vdash\ (\neg\mathsf{idle}.p \wedge \neg\mathit{sent\_to\_all\_non\_fathers}.p) \wedge (\neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p) \rightarrowtail (\mathit{sent\_to\_all\_non\_fathers}.p \wedge (\neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p)) \vee (\neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \wedge \mathit{sent\_to\_all\_non\_fathers}.p)$$

$\Leftarrow$($\rightarrowtail$ PSP (B.8_{55}))

$$\left.\begin{array}{ll}\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : & \\ \quad {}_{\text{ECHO}}\vdash & J_{\text{PLUM}} \wedge J_{\text{ECHO}} \\ & \wedge \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \\ & \ \mathsf{unless} \\ & \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \\ & \wedge\ \mathit{sent\_to\_all\_non\_fathers}.p \end{array}\right\} \textbf{PSP} - \textbf{unless}$$

$$\wedge$$

$$\left.\begin{array}{ll}\forall p \in \mathbb{P} : & \\ \quad {}_{\text{ECHO}}\vdash & \neg\mathsf{idle}.p \wedge \neg \mathit{sent\_to\_all\_non\_fathers}.p \\ & \rightarrowtail \\ & \mathit{sent\_to\_all\_non\_fathers}.p \end{array}\right\} \textbf{PSP} - \textbf{reach}$$

The proof of the **PSP-unless**-part is not complicated, again the characterisation of J_{PLUM} suffices, and hence no additional safety properties have to be added to J_{ECHO}. Note, that at this point J_{ECHO} can be substituted by true.

We shall proceed with the **PSP-reach**-part. If we look at it closely, we can see that it resembles **ana_2**, a proof obligation we encountered during the verification of termination of PLUM (see pages 15, 17). Obviously, if we can transform the **PSP-reach**-part into a **ana_2**, we can re-use the proof-strategy used to prove **ana_2** in the context of PLUM, to prove the **PSP-reach**-part in the context of ECHO. Since, **ana_2**'s proof-strategy uses conjunctivity of $\rightsquigarrow$ (theorem C.11$_{56}$), and $\rightarrowtail$ does not have this property, we first replace $\rightarrowtail$ by $\rightsquigarrow$:

$\Leftarrow$($\rightsquigarrow$ Convergence Implies Progress (C.1$_{55}$))

$$\begin{array}{ll}\forall p \in \mathbb{P} : & \\ \quad {}_{\text{ECHO}}\vdash & \neg\mathsf{idle}.p \wedge \neg \mathit{sent\_to\_all\_non\_fathers}.p \\ & \rightsquigarrow \\ & \mathit{sent\_to\_all\_non\_fathers}.p \end{array}$$

Then, we apply a $\rightsquigarrow$ Substitution (C.2$_{55}$) step similar to the ♫-marked-substitution step made on page 18 to obtain:

$$\begin{array}{ll}\forall p \in \mathbb{P} : & \\ \quad {}_{\text{ECHO}}\vdash & \forall q \in \mathsf{neighs}.p : \neg\mathsf{idle}.p \\ & \rightarrowtail \\ & \forall q \in \mathsf{neighs}.p : (\neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)) \vee (\mathsf{nr\_sent}.p.q = 1) \end{array}$$

Subsequently, we apply a conjunction step similar to the ♫-marked-conjunction step made on page 18. Now, our proof obligation has become equal to that of **ana_2** only now in the context of ECHO:

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \ {}_{\text{ECHO}}\vdash \neg\mathsf{idle}.p \rightsquigarrow (\neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p)) \vee (\mathsf{nr\_sent}.p.q = 1)$$

Consequently, the same proof strategy applies. Inspecting **ana_2**'s proof strategy on page 18 this comes down to proving:

Theorem 7.7 *STABLEe_not_idle_AND_q_IS_f_p_in_ECHO*

$$\forall p, q \in \mathbb{P} : \ {}_{\text{ECHO}}\vdash \circlearrowright(\neg\mathsf{idle}.p \wedge (q = \mathsf{father}.p))$$

which is straightforward, using the stability preserving Theorem 3.8$_4$. Moreover, we need an ECHO equivalent for Theorem 6.16$_{18}$ (i.e. **ana_1.2.1**, page 16). Again, the proof-strategy of **ana_1.2.1** can be re-used. Returning to page 16, we can see this comes down to proving the following two properties. First,

Theorem 7.8 *STABLEe_nr_sent_is_1_in_ECHO*

$$\forall p, q \in \mathbb{P} : \ {}_{\text{ECHO}}\vdash \circlearrowright(\mathsf{nr\_sent}.p.q = 1)$$

which again is easy using stability preserving Theorem 3.8$_4$. Second,

$${}_{\text{ECHO}}\vdash (J_{\text{PLUM}} \wedge J_{\text{ECHO}} \wedge \neg\mathsf{idle}.p \wedge q \neq \mathsf{father}.p)\ \mathsf{ensures}\ (\mathsf{nr\_sent}.p.q = 1)$$

This last proof obligation can be proved similarly to that of the ensures-part of **ana_1.2.1** (see page 16), and doing so, the unless-part of the ensures-part of **ana_1.2.1** can be inherited by using unless-preserving Theorem 3.7_4.

This ends the verification of the **reach-part**. Since, no additional safety properties have to be proved for ECHO, we can define J_{ECHO} to be true.

Definition 7.9 *Invariant_ECHO*

$J_{\mathrm{ECHO}} = \mathsf{true}$

since true is trivially stable, this ends verification of termination of ECHO. Although the definition for J_{ECHO} might appear superfluous, we decided to include it for two reasons. The first one being preservation of consistency throughout this report. The second reason is that by explicitly defining J_{ECHO} to be true, it immediately becomes clear that PLUM and ECHO have the same safety properties.

8 Using refinements to derive termination of Tarry

This section shall describe how termination of the Tarry algorithm is proved using the refinements framework from Section 3, and the already proven fact that:

$$\forall J :: \mathrm{PLUM} \sqsubseteq_{\mathcal{R}\_\mathrm{PLUM\_TARRY},\, J} \mathrm{TARRY}$$

The UNITY specification reads:

Theorem 8.1 *HYLO_Tarry*

$$J_{\mathrm{PLUM}} \wedge J_{\mathrm{TARRY}} \;\; {}_{\mathrm{TARRY}}\!\vdash \mathbf{ini}\mathrm{TARRY} \leadsto \forall p : p \in \mathbb{P} : \mathit{done}.p$$

where invariant J_{TARRY} captures additional safety properties for Tarry. Again, J_{TARRY} shall be constructed incrementally in a demand-driven way following the conventions described in Section 6.1.

Using ↻Preservation Theorem 3.8_4, it is straightforward to derive that J_{PLUM} is also (Theorem 6.44_{31}) a stable predicate in Tarry.

Theorem 8.2 *STABLEe_Invariant_in_Tarry*

$${}_{\mathrm{TARRY}}\!\vdash \circlearrowright J_{\mathrm{PLUM}}$$

The stability of: ${}_{\mathrm{TARRY}}\!\vdash \circlearrowright (J_{\mathrm{PLUM}} \wedge J_{\mathrm{TARRY}})$ will be implicitly assumed throughout the verification process, and verified when the precise characterisation of J_{TARRY} has been established. For ease of reference, Figure 16 displays theorems about the guards of Tarry's actions. For readability we, again, introduce the notational convention that $\vdash$ abbreviates $J_{\mathrm{PLUM}} \wedge J_{\mathrm{TARRY}} \;\; {}_{\mathrm{TARRY}}\!\vdash$.

Termination of Tarry is proved using property preserving Theorem 3.5_5. The reason for using this theorem is that Theorem 3.6_5 – which is easier and hence preferable – cannot be used since its application results in the following, not provable, proof obligation:

$${}_{\mathrm{TARRY}}\!\vdash J_{\mathrm{PLUM}} \wedge J_{\mathrm{TARRY}} \wedge \mathsf{guard\_of}.(\mathrm{PROP}_{\mathrm{TARRY}}.p.q) \;\mathsf{unless}\; \neg\mathsf{guard\_of}.(\mathrm{PROP}.p.q)$$

The reason why this cannot be proved is because, during the execution of Tarry, it is possible that the guard of $\mathrm{PROP}_{\mathrm{TARRY}}.p.q$ is falsified while the guard of $\mathrm{PROP}.p.q$ still holds. For the sake of clarity, we shall elucidate this below. We rewrite the unless-property from above, using Definition $\mathrm{A}.8_{53}$, Theorem 6.8_{11} and Theorem 8.5_{36}. (Note that we have omitted compile):

$$
\begin{array}{l}
\forall A \in \mathbf{a}\mathrm{TARRY}, s, t \in \mathtt{State} : \\
J_{\mathrm{PLUM}}.s \;\wedge\; J_{\mathrm{TARRY}}.s \wedge \neg s.(\mathsf{idle}.p) \wedge \neg \mathit{sent\_to\_all\_non\_fathers.p.s} \wedge \mathit{can\_propagate.p.q.s} \wedge s.(\mathsf{le\_rec}.p) \wedge\; A.s.t \\
\Rightarrow \\
\quad J_{\mathrm{PLUM}}.t \wedge J_{\mathrm{TARRY}}.t \wedge \neg t.(\mathsf{idle}.p) \wedge \neg \mathit{sent\_to\_all\_non\_fathers.p.t} \wedge \mathit{can\_propagate.p.q.t} \wedge t.(\mathsf{le\_rec}.p)) \\
\;\vee \\
\quad t.(\mathsf{idle}.p) \vee \mathit{sent\_to\_all\_non\_fathers.p.t} \vee \neg \mathit{can\_propagate.p.q.t}
\end{array}
$$

Theorem 8.3 *guard_of_IDLE_Tarry*

$\mathsf{guard\_of}.(\text{IDLE}_{\text{TARRY}}.p.q) = \mathsf{guard\_of}.(\text{IDLE}.p.q)$

Theorem 8.4 *guard_of_COL_Tarry*

$\mathsf{guard\_of}.(\text{COL}_{\text{TARRY}}.p.q) = \mathsf{guard\_of}.(\text{COL}.p.q) \wedge \neg\mathsf{le\_rec}.p$

Theorem 8.5 *guard_of_PROP_Tarry*

$\mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.p.q) = \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \mathsf{le\_rec}.p$

Theorem 8.6 *guard_of_DONE_Tarry*

$\mathsf{guard\_of}.(\text{DONE}_{\text{TARRY}}.p.q) = \mathsf{guard\_of}.(\text{DONE}.p.q)$

Figure 16: Guards of the actions from TARRY

We have to prove this for arbitrary actions of TARRY. Consider the propagating action $\text{PROP}_{\text{TARRY}}.p.q'$, with $(q \neq q')$. Assume for arbitrary states s and t that:
$\mathbf{A}_1$: $J_{\text{PLUM}}.s \wedge J_{\text{TARRY}}.s$
$\mathbf{A}_2$: $\neg s.(\mathsf{idle}.p) \wedge \neg \mathit{sent\_to\_all\_non\_fathers}.p.s \wedge \mathit{can\_propagate}.p.q.s \wedge s.(\mathsf{le\_rec}.p)$
$\mathbf{A}_3$: $\text{PROP}_{\text{TARRY}}.p.q'.s.t$
$\mathbf{A}_4$: $(q \neq q')$
If p cannot propagate to q' in state s, then $s = t$ and there is no problem in the sense that the conclusion of the implication stated above can be proved. However, suppose p can propagate to q' (i.e. $\mathit{can\_propagate}.p.q'.s$). Then the guard of $\text{PROP}_{\text{TARRY}}.p.q'.s.t$ is enabled and execution of this action establishes: $\neg t.(\mathsf{le\_rec}.p)$. Consequently, the guard of $\text{PROP}_{\text{TARRY}}.p.q$ is disabled in state t, and in order to prove the conclusion of the implication we have to prove that the guard of $\text{PROP}.p.q$ is also disabled in state t. That is, we have to prove one of:

$$t.(\mathsf{idle}.p) \vee \mathit{sent\_to\_all\_non\_fathers}.p.t \vee \neg \mathit{can\_propagate}.p.q.t$$

However,

- $t.(\mathsf{idle}.p)$ cannot be proved, since from $\mathbf{A}_2$ we know that p is non-idle in state s, and since PROP-actions do not write to idle-variables we know that p is still non-idle in state t.
- $\neg \mathit{can\_propagate}.p.q.t$ cannot be proved, since from $\mathbf{A}_2$ we know that, in state s, p can propagate to q ($\mathit{can\_propagate}.p.q.s$), and since $(q \neq q')$ we know that p can still propagate to q in state t (i.e. $\mathit{can\_propagate}.p.q.t$).
- $\mathit{sent\_to\_all\_non\_fathers}.p.t$ is not necessarily valid. It can hold in state t, but it might as well be the case that is does not.

Consequently, we cannot prove the unless-property from above. What we need is a function which is non-increasing with respect to some well-founded relation, and which decreases when a message is sent. Since then, we can ensure that this kind of premature falsification of the guard of $\text{PROP}_{\text{TARRY}}.p.q$, while the guard of $\text{PROP}.p.q$ still holds, cannot happen infinitely often.

As an aside: The guards of IDLE and DONE actions in TARRY are equal to those of PLUM (Theorems 8.3_{36} and 8.6_{36}). Consequently, for these actions, a unless-property similar to the one above can if necessary be proved using unless ANTI-REFLEXIVITY $\text{A}.10_{53}$.

For the COL-actions, the construction of a non-increasing function is *not* required, since we can, if necessary, prove that when the guard of $\text{COL}_{\text{TARRY}}.p.q$ (Theorem 8.4_{36}) is falsified, then so is the guard of $\text{COL}.p.q$. This is because, intuitively, TARRY has the additional invariant that there is always at most one message in transit. Therefore, if some action $\text{COL}_{\text{TARRY}}.p.q'$ $(q \neq q')$ receives the message that is in transit from q' to p and as a consequence falsifies the guard of $\text{COL}_{\text{TARRY}}.p.q$ by setting $\mathsf{le\_rec}.p$ to true, then we can prove that afterward there are no messages at all in transit and hence that the guard of $\text{COL}.p.q$ cannot be true.

So, since the least complicated property preservation Theorem (3.6_5) cannot be used to derive termination of TARRY, we move on to the second least complicated one, i.e. 3.5_5. Since the bitotal relation defined on the actions of PLUM and TARRY is one-to-one, this one turns out to be sufficient.

$$_{\text{TARRY}}\vdash \mathbf{ini}(\text{TARRY}.iA.h.\text{PROP\_mes}.\text{DONE\_mes}) \rightsquigarrow \forall p : p \in \mathbb{P} : done.p$$

$\Leftarrow$(Theorem 3.5_5, 6.1_{11}, 5.2_{10})
For some well-founded relation $\prec$:

$$\exists W :: (\mathbf{w}\text{TARRY} = \mathbf{w}\text{PLUM} \cup W) \wedge (J_{\text{PLUM}}\ \mathcal{C}\ W^{c}) \wedge (\mathbf{w}\text{PLUM} \subseteq W^{c})$$

$\wedge$

$$\left.\begin{array}{l} \forall A_P\ A_T\ :\ A_P \in \mathbf{a}\text{PLUM}\ \wedge\ A_P\ \mathcal{R}_{\text{-PLUM\_TARRY}}\ A_T\ : \\ \quad {}_{\text{TARRY}}\vdash \mathsf{guard\_of}.A_P \rightarrowtail \mathsf{guard\_of}.A_T \end{array}\right\}\ \mathbf{reach-part}$$

$\wedge$

$$\left.\begin{array}{l} \exists M :: (M\ \mathcal{C}\ \mathbf{w}\text{TARRY}) \\ \quad \wedge \\ \quad \forall k ::\ {}_{\text{TARRY}}\vdash (J_{\text{PLUM}} \wedge J_{\text{TARRY}} \wedge M = k)\ \mathsf{unless}\ (M \prec k) \\ \quad \wedge \\ \quad \forall k\ A_P\ A_T : A_P \in \mathbf{a}\text{PLUM} \wedge A_P\ \mathcal{R}_{\text{-PLUM\_TARRY}}\ A_T : \\ \qquad {}_{\text{TARRY}}\vdash \quad (J_{\text{PLUM}} \wedge J_{\text{TARRY}} \wedge \mathsf{guard\_of}.A_T \wedge M = k) \\ \qquad\qquad \mathsf{unless} \\ \qquad\qquad (\neg(\mathsf{guard\_of}.A_P) \vee M \prec k) \end{array}\right\}\ \mathbf{unless-part}$$

Since, $\mathsf{le\_rec}.p$ variables are superimposed on PLUM in order to obtain TARRY, the first conjunct is instantiated with the set $\{\mathsf{le\_rec}.p \mid p \in \mathbb{P}\}$. Proving that J_{PLUM} is confined by the complement of this set is tedious but straightforward, since the variables le_rec do not appear in it.

Verification of the **unless-part** involves the construction of a function over the variables of TARRY, that is non-increasing with respect to some well-founded relation $\prec$. From the discussion above, we can deduce that we need a function that decreases when a message is sent. However, it turns out that the verification of the **reach-part** involves an application of $\rightarrowtail$ BOUNDED PROGRESS ($B.10_{55}$) that needs a function that decreases not only when a message is sent, but also when a message is received. Consequently, we shall continue with the construction of a function over the variables of TARRY, that is non-increasing with respect to some well-founded relation $\prec$, and that decreases when a message is sent as well as received. Obviously, this function can then be used for both purposes.

8.1 Construction of a non-increasing function

Constructing a non-increasing function that decreases when a message is sent, and when a message is received is not complicated. Observe the following:

- the sending of a message is always accompanied by incrementing a nr_sent variable
- similarly, receiving a message is always accompanied by incrementing a nr_rec variable
- from J_{PLUM} it follows that at most one message is sent over each directed communication link
- consequently, at most one message is received over each directed communication link
- consequently, the total amount of messages sent and received has an upper-bound, that equals twice the cardinality of the set of directed communication links

From these observations a non-increasing function is constructed as follows. First, we define the upper-bound on the total amount of messages sent *and* received.

Definition 8.7 *MAX_MAIL*

$$\mathsf{MAX\_MAIL} = 2\ \times\ \mathsf{card}.(\mathsf{links}.\mathbb{P}.\mathsf{neighs})$$

Next, we define the total amount of messages that a process $p \in \mathbb{P}$ has sent, and respectively received, in some state s.

Definition 8.8 NUMBER OF MESSAGES SENT BY PROCESSES p *NR_SENT*

$$\mathsf{NR\_SENT}.p.s = \sum_{q \in \mathsf{neighs}.p} s.(\mathsf{nr\_sent}.p.q)$$

Theorem 8.12 *rec_from_all_p_EQ_NR_REC_EQ_CARD_p*

$$\forall p \in \mathbb{P}, s \in \texttt{State} : \quad \frac{J_{\text{PLUM}}.s}{\mathit{rec\_from\_all\_neighs}.p = (\mathsf{NR\_REC}.p.s = \mathsf{card}.(\mathsf{neighs}.p))}$$

Theorem 8.13 *sent_2_all_p_EQ_NR_SENT_EQ_CARD_p*

$$\forall p \in \mathbb{P}, s \in \texttt{State} : \quad \frac{J_{\text{PLUM}}.s}{\mathit{sent\_to\_all\_neighs}.p = (\mathsf{NR\_SENT}.p.s = \mathsf{card}.(\mathsf{neighs}.p))}$$

Theorem 8.14 *NR_REC_leq_CARD*

$$\forall p \in \mathbb{P}, s \in \texttt{State} : \quad \frac{J_{\text{PLUM}}.s}{\mathsf{NR\_REC}.p.s \leq \mathsf{card}.(\mathsf{neighs}.p)}$$

Theorem 8.15 *NR_REC_SUC_NR_SENT_IMP_not_sent_2_all*

$$\forall p \in \mathbb{P}, s \in \texttt{State} : \quad \frac{J_{\text{PLUM}}.s \wedge (\mathsf{NR\_REC}.p.s = \mathsf{NR\_SENT}.p.s + 1)}{\neg \mathit{sent\_to\_all\_neighs}.p}$$

Theorem 8.16 *sent_2_all_except_f_IMP_SUC_NR_SENT_EQ_CARD*

$$\forall p \in \mathbb{P}, s \in \texttt{State} : \quad \frac{J_{\text{PLUM}}.s \wedge \mathit{sent\_to\_all\_non\_fathers}.p.s \wedge \neg \mathit{sent\_to\_all\_neighs}.p.s}{\mathsf{NR\_SENT}.p.s + 1 = \mathsf{card}.(\mathsf{neighs}.p)}$$

Figure 17: Some properties of NR_REC and NR_SENT

Definition 8.9 NUMBER OF MESSAGES RECEIVED BY PROCESSES p *NR_REC*

$$\mathsf{NR\_REC}.p.s = \sum_{q \in \mathsf{neighs}.p} s.(\mathsf{nr\_rec}.p.q)$$

The total amount of messages that are sent, and respectively received, in the whole network of processes can be defined as follows:

Definition 8.10 TOTAL NUMBER OF MESSAGES SENT IN THE NETWORK *TOTAL_NR_SENT*

$$\mathsf{TOTAL\_NR\_SENT}.s = \sum_{p \in \mathbb{P}} \mathsf{NR\_SENT}.p.s$$

Definition 8.11 TOTAL NUMBER OF MESSAGES RECEIVED IN THE NETWORK *TOTAL_NR_REC*

$$\mathsf{TOTAL\_NR\_REC}.s = \sum_{p \in \mathbb{P}} \mathsf{NR\_REC}.p.s$$

Finally, we define our non-increasing function as follows:

Definition 8.17 NON-INCREASING FUNCTION OVER THE VARIABLES OF TARRY *Y_DEF*

$$Y.s = \mathsf{MAX\_MAIL} - (\mathsf{TOTAL\_NR\_SENT}.s + \mathsf{TOTAL\_NR\_REC}.s)$$

The value of Y only depends on the variables nr_rec and nr_sent. Since these are write variables of TARRY is it easy to verify that:

Theorem 8.18 *CONF_Y_Write_Vars_Tarry*

$$Y \ \mathcal{C} \ \mathbf{w}\text{TARRY}$$

The following lemma states that whenever a message is sent or received – because the guard of one of TARRY's actions is enabled – the value of Y decreases.

Lemma 8.19 *A_DECR_Y*

For arbitrary processes $p \in \mathbb{P}$, $q \in \mathsf{neighs}.p$, and actions A;
$A \in \{\text{IDLE}_{\text{Tarry}}, \text{COL}_{\text{Tarry}}, \text{PROP}_{\text{Tarry}}, \text{DONE}_{\text{Tarry}}\}$:

$$\forall k :: \frac{J_{\text{PLUM}}.s \wedge A.p.q.s.t \wedge \mathsf{guard\_of}.(A.p.q).s \wedge (Y.s = k)}{Y.t < k}$$

Using this lemma, it is straightforward to prove that, during the execution of TARRY, Y is non-increasing with respect to the well-founded relation $<$ on numerals.

Theorem 8.20 *DECREASING_DECR_FUNCTION*

For arbitrary characterisations of J_{Tarry}:

$$\forall k :: \ {}_{\text{Tarry}}\vdash (J_{\text{PLUM}} \wedge J_{\text{Tarry}} \wedge Y = k) \ \mathsf{unless}\ (Y < k)$$

Verification of the unless-part

Return to page 37 for the **unless-part**. Instantiating this proof obligation with Y, and rewriting with Theorems 8.18_{38} and 8.20_{39} results in the following proof obligation:

$$\begin{array}{l} \forall k\ A_P\ A_T : A_P \in \mathbf{a}\text{PLUM} \wedge A_P\ \mathcal{R}_{\text{-PLUM\_TARRY}}\ A_T : \\ \quad {}_{\text{Tarry}}\vdash (J_{\text{PLUM}} \wedge J_{\text{Tarry}} \wedge \mathsf{guard\_of}.A_T \wedge Y = k)\ \mathsf{unless}\ (\neg(\mathsf{guard\_of}.A_P) \vee Y < k) \end{array}$$

Proving this is straightforward using the characterisation of $\mathcal{R}_{\text{-PLUM\_Tarry}}$ from Figure 5_{10}, and Lemma 8.19_{39}. Note that, since Y is constructed as to decrease when a message is sent as well as when a message is received, we do not have to use the proof strategy delineated in the aside on page 36 for the COL actions. Consequently, constructing a non-increasing function that decreases upon the sending as well as upon receiving of a message is not only more efficient since it is re-usable in the proof of the **reach-part**, it also simplifies the verification of the **unless-part**.

Verification of the reach-part

We shall now continue with the **reach-part**, which is re-displayed below for convenience.

$$\begin{array}{l} \forall A_P\ A_T\ :\ A_P \in \mathbf{a}\text{PLUM}\ \wedge\ A_P\ \mathcal{R}_{\text{-PLUM\_TARRY}}\ A_T\ : \\ \quad {}_{\text{Tarry}}\vdash \mathsf{guard\_of}.A_P \rightarrowtail \mathsf{guard\_of}.A_T \end{array}$$

Subsequently, using:

- the characterisation of $\mathcal{R}_{\text{-PLUM\_Tarry}}$ (Figure 5_{10})
- Theorems 8.3_{36} and 8.6_{36}, stating that the guards of the $\text{IDLE}_{\text{Tarry}}$, and $\text{DONE}_{\text{Tarry}}$ actions are equal to those of PLUM
- reflexivity of $\rightarrowtail$ (Theorem $\text{B}.4_{54}$)
- the implicit assumption stating stability of $(J_{\text{PLUM}} \wedge J_{\text{Tarry}})$

we reduce the **reach-part** for arbitrary $p \in \mathbb{P}$ and $q \in \mathsf{neighs}.p$, as follows:

$$\begin{array}{l} {}_{\text{Tarry}}\vdash \mathsf{guard\_of}.(\text{COL}.p.q) \rightarrowtail \mathsf{guard\_of}.(\text{COL}_{\text{Tarry}}.p.q)\ \big\}\ \mathbf{reach} - \text{COL} - \mathbf{part} \\ \wedge \\ {}_{\text{Tarry}}\vdash \mathsf{guard\_of}.(\text{PROP}.p.q) \rightarrowtail \mathsf{guard\_of}.(\text{PROP}_{\text{Tarry}}.p.q)\ \big\}\ \mathbf{reach} - \text{PROP} - \mathbf{part} \end{array}$$

Verification of reach-COL-part

Rewriting with the characterisations of the guards (Theorem 6.7_{11} and 8.4_{36}) gives:

$$\begin{array}{ll} {}_{\text{Tarry}}\vdash & \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \\ & \rightarrowtail \\ & \neg\mathsf{idle}.p \wedge \mathsf{mit}.q.p \wedge \neg \mathit{rec\_from\_all\_neighs}.p \wedge \neg\mathsf{le\_rec}.p \end{array}$$

Due to the alternating sending and receiving of messages, which is inherent to TARRY, we know that it must be provable that there is always at most one message in transit during the execution of TARRY's algorithm. This means that if there is a message in transit, it is the only one, and hence the event last executed by all processes was a send-event and thus not a receive-event. Consequently, the above proof obligation must be provable from the invariant, by using $\rightarrowtail$ INTRODUCTION (B.3_{54}). In order to establish this we propose the following invariant-candidate:

$cJ^1_{\text{TARRY}} = (\exists p \in \mathbb{P}, q \in \mathsf{neighs}.p : mit.p.q) \Rightarrow (\forall p \in \mathbb{P} : \neg \mathsf{le\_rec}.p)$

which, evidently, suffices to establish the **reach**-COL-**part**.

Verification of reach-PROP-part

Rewriting with the characterisations of $\text{PROP}_{\text{TARRY}}$'s the guard (8.$5_{36}$) gives:

$$_{\text{TARRY}}\vdash \quad \mathsf{guard\_of}.(\text{PROP}.p.q) \rightarrowtail \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \mathsf{le\_rec}.p$$

If p's last event was a receive event this is easy to prove:

$\Leftarrow$($\rightarrowtail$ CASE DISTINCTION (B.6_{55}), p's last event was a receive event or not)

$$_{\text{TARRY}}\vdash \quad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \mathsf{le\_rec}.p \rightarrowtail \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \mathsf{le\_rec}.p$$

$\wedge$

$$_{\text{TARRY}}\vdash \quad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \rightarrowtail \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \mathsf{le\_rec}.p$$

$\Leftarrow$($\rightarrowtail$ REFLEXIVITY (B.4_{54}), proves the first conjunct)

$$_{\text{TARRY}}\vdash \quad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \rightarrowtail \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \mathsf{le\_rec}.p$$

To explain the proof-strategy that is used to verify the conjunct from above, we refer to Figure 18_{41}. The p and q in this figure correspond to the p and q in the current proof-obligation, x, y, z, and w are arbitrary processes. We already indicated that, during an execution of TARRY's algorithm, there is always at most one message in transit. This message is indicated with a $\bullet$ in Figure 18. In Figure 18(b), this message is in transit from w to z, and hence from invariant-candidate cJ^1_{TARRY} we can infer that $\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p$ In 18(a) this message has just been received by x, and hence we can infer that $\mathsf{le\_rec}.x$. In order to establish our current proof obligation, we need to invent a proof strategy that enables us to prove that this message shall eventually reach p such that the latter can set $\mathsf{le\_rec}.p$ to true. Suppose that $\mathsf{guard\_of}.(\text{PROP}.p.q)$ holds, and that the last event of p was *not* a receive event. Using Theorem 6.8_{11}):

$$\neg\mathsf{idle}.p \wedge cp.p.q \wedge \neg sent\_to\_all\_non\_fathers.p \wedge \neg\mathsf{le\_rec}.p \quad (☼)$$

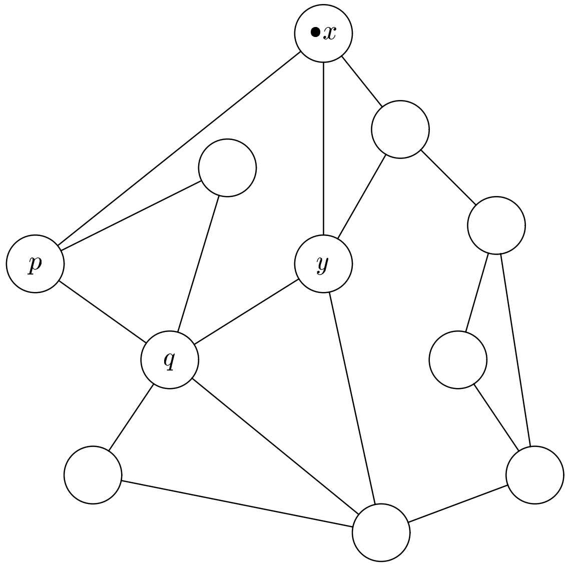
If the current situation is that of Figure 18(a), then x has just received the message, and hence $\mathsf{le\_rec}.x$ holds. Since, we have assumed that $\neg\mathsf{le\_rec}.p$, we know that $(x \neq p)$. There are now two possibilities: either $\text{PROP}_{\text{TARRY}}.x.y$ or action $\text{DONE}_{\text{TARRY}}.x.y$ is enabled (y is arbitrary) and will execute. Consequently, we know that a message will be sent and hence that Y will decrease. Since $(x \neq p)$, we know that (☼) still holds, and subsequently, we have arrived in a situation similar to that of Figure 18(b).

If the current situation is that of Figure 18(b), then either $\text{IDLE}_{\text{TARRY}}.z.w$ or action $\text{COL}_{\text{TARRY}}.z.w$ is enabled. If $(z = p)$, then we know that $\mathsf{le\_rec}.p$ will become true, and hence we are ready. If $(z \neq p)$, then we know that, since the message will be received by z, again Y shall decrease. Since $(z \neq p)$, we know that (☼) still holds, and subsequently, we have arrived again in a situation similar to that of Figure 18(a).

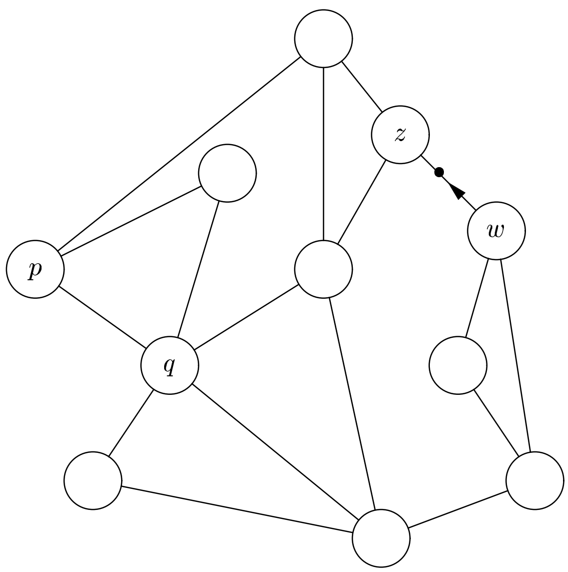
Since we have already proved that Y is a non-increasing function with respect to the well-founded relation $<$, we know that we cannot infinitely proceed from the situation in Figure 18(a) to the situation in Figure 18(b). Therefore, we shall eventually end in Figure 18(b) where $(z = p)$, and hence $\mathsf{le\_rec}.p$ will be set to true.

$$_{\text{TARRY}}\vdash \quad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \rightarrowtail \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \mathsf{le\_rec}.p$$

$\Leftarrow$($\rightarrowtail$ BOUNDED PROGRESS (B.10_{55}), using Y)



(a)



(b)

Figure 18: Possible situations when $\mathsf{guard\_of}.(\textsc{prop}.p.q) \wedge \neg\mathsf{le\_rec}.p$ holds

$$\begin{array}{ll} {}_{\text{TARRY}}\vdash & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \\ & \rightarrowtail \\ & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}$$

$\Leftarrow$($\rightarrowtail$ CASE DISTINCTION (B.6_{55}): situation of Figure 18(a), or 18(b))

$$\begin{array}{ll} {}_{\text{TARRY}}\vdash & \left.\begin{array}{l} \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\exists x \in \mathbb{P} : \mathsf{le\_rec}.x) \\ \rightarrowtail \\ \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}\right\} \mathbf{18(a)} \\ \wedge & \\ {}_{\text{TARRY}}\vdash & \left.\begin{array}{l} \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p) \\ \rightarrowtail \\ \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}\right\} \mathbf{18(b)} \end{array}$$

Verification of 18(a)

We shall proceed with proof-obligation **18(a)**, using the proof-strategy explained above. That is, we shall need to decompose the proof-obligation in such a way that we can use $\rightarrowtail$ INTRODUCTION (B.3_{54}) to prove that either $\text{PROP}_{\text{TARRY}}.x.y$ or $\text{DONE}_{\text{TARRY}}.x.y$ will decrease Y. First, we shall identify process x (from Figure 18(a)) in the left hand side of $\rightarrowtail$ as follows:

$$\begin{array}{ll} {}_{\text{TARRY}}\vdash & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\exists x \in \mathbb{P} : \mathsf{le\_rec}.x) \\ & \rightarrowtail \\ & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}$$

$\Leftarrow$ ($\rightarrowtail$ SUBSTITION (B.2_{54}), $\rightarrowtail$ DISJUNCTION (B.9_{55}),
and $(x \neq p)$ since $(\neg\mathsf{le\_rec}.p \wedge \mathsf{le\_rec}.x)$)

$\forall x \in \mathbb{P}, (x \neq p)$:

$$\begin{array}{ll} {}_{\text{TARRY}}\vdash & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge \mathsf{le\_rec}.x \\ & \rightarrowtail \\ & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}$$

Whether $\text{PROP}_{\text{TARRY}}.x.y$ or $\text{DONE}_{\text{TARRY}}.x.y$ is the action that will decrease Y, depends on whether x has *sent_to_all_non_fathers*, or not. Therefore, we proceed making the following case distinction:

$\Leftarrow$($\rightarrowtail$ CASE DISTINCTION (B.6_{55}))

$\forall x \in \mathbb{P}, (x \neq p)$:

$$\begin{array}{ll} {}_{\text{TARRY}}\vdash & \left.\begin{array}{l} \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \\ \wedge\ \mathsf{le\_rec}.x \wedge \neg\mathit{sent\_to\_all\_non\_fathers}.x \\ \rightarrowtail \\ \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}\right\} \begin{array}{l}\mathbf{18(a)} \\ -\text{PROP}\end{array} \\ \wedge & \\ {}_{\text{TARRY}}\vdash & \left.\begin{array}{l} \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \\ \wedge\ \mathsf{le\_rec}.x \wedge \mathit{sent\_to\_all\_non\_fathers}.x \\ \rightarrowtail \\ \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}\right\} \begin{array}{l}\mathbf{18(a)} \\ -\text{DONE}\end{array} \end{array}$$

Verification of 18(a)-PROP

The proof strategy for **18(a)**-PROP shall consists of using $\rightarrowtail$ INTRODUCTION (B.3_{54}), and proving that, for some y, $\text{PROP}_{\text{TARRY}}.x.y$ ensures that the value of Y decreases. Consequently, we have to substitute the left hand side $\rightarrowtail$ in such a way that it implies the existence of an y such that the guard of $\text{PROP}_{\text{TARRY}}.x.y$ holds. In order to be able to do this it suffices to prove that for arbitrary states s:

$$\begin{array}{l} J_{\text{PLUM}}.s \wedge J_{\text{TARRY}}.s \wedge s.(\mathsf{le\_rec}.x) \wedge \neg\mathit{sent\_to\_all\_non\_fathers}.x.s \\ \Rightarrow \\ \exists y \in \mathsf{neighs}.x : \neg\mathsf{idle}.x \wedge cp.x.y.s \wedge \neg\mathit{sent\_to\_all\_non\_fathers}.x.s \wedge s.(\mathsf{le\_rec}.x) \end{array}$$

Using Theorem 6.10_{13}, and cJ^2_{PLUM} from J_{PLUM}, it is straightforward to prove that:

Theorem 8.21 *not_sent_2_all_except_f_IMP_cp*

$$\forall p \in \mathbb{P} : \frac{J_{\text{PLUM}}.s \wedge \neg \mathit{sent\_to\_all\_non\_fathers.p.s}}{\exists q \in \mathsf{neighs}.p : \mathit{cp.p.q.s}}$$

Consequently, it remains to prove that x is non-idle. Since the fact that x has not *sent_to_all_non_fathers* is *not* sufficient to deduce this, we need a new invariant-candidate for J_{TARRY}. Evidently, the one that suffices here is:

$cJ^2_{\text{TARRY}} = \forall p \in \mathbb{P} : \mathsf{le\_rec}.p \Rightarrow \neg \mathsf{idle}.p$

Subsequently, **18(a)**-PROP is established as follows:

$\Leftarrow$($\rightarrowtail$ SUBSTITUTION (B.2$_{54}$), cJ^2_{TARRY}, and Theorems 8.5$_{36}$ and 8.21$_{43}$)
$\forall x \in \mathbb{P}, (x \neq p)$:
$_{\text{TARRY}}\vdash$ $\exists y \in \mathsf{neighs}.x$:
$\mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg \mathsf{le\_rec}.p \wedge (Y = k) \wedge \mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.x.y)$
$\rightarrowtail$
$\mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg \mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p))$
$\Leftarrow$($\rightarrowtail$ DISJUNCTION (B.9$_{55}$), $\rightarrowtail$ INTRODUCTION (B.3$_{54}$))
$\forall x \in \mathbb{P}, (x \neq p), y \in \mathsf{neighs}.x$:
$_{\text{TARRY}}\vdash$ $J_{\text{PLUM}} \wedge J_{\text{TARRY}}$
$\wedge\ \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg \mathsf{le\_rec}.p \wedge (Y = k) \wedge \mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.x.y)$
ensures
$\mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg \mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p))$

Proving this ensures-property is straightforward using Lemma 8.19$_{39}$.

Verification of 18(a)-DONE

The proof strategy for **18(a)**-DONE is similar to that of **18(a)**-PROP. That is, we use $\rightarrowtail$ INTRODUCTION (B.3$_{54}$) and prove that $\text{DONE}_{\text{TARRY}}.x.y$ ensures that the value of Y decreases. Again we have to substitute the left hand side $\rightarrowtail$ in such a way that it implies the guard of $\text{DONE}_{\text{TARRY}}.x.y$. However, since the guard of $\text{DONE}_{\text{TARRY}}$ is never enabled for the *starter*, we first have to prove that $(x \neq \mathit{starter})$. In order to do this we prove **18(a)**-DONE for the case when $(x = \mathit{starter})$ and $(x \neq \mathit{starter})$.

Verification of 18(a)-DONE when $x = \mathit{starter}$

We have to prove that, when $(\mathit{starter} \neq p)$,

$_{\text{TARRY}}\vdash$ $\mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg \mathsf{le\_rec}.p \wedge (Y = k)$
$\wedge\ \mathsf{le\_rec}.\mathit{starter} \wedge \mathit{sent\_to\_all\_non\_fathers.starter}$
$\rightarrowtail$
$\mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg \mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p))$

Since the guard of $\text{DONE}_{\text{TARRY}}$ is never enabled for the *starter*, the only possible way to proceed here is: use $\rightarrowtail$ INTRODUCTION (B.3$_{54}$), and subsequently prove that the left hand side of the $\rightarrowtail$ in conjunction with J_{PLUM} and J_{TARRY} evaluates to false. So assume, for some state s, it holds that:
$\mathbf{A}_1$: $J_{\text{PLUM}}.s \wedge J_{\text{TARRY}}.s$
$\mathbf{A}_2$: $\mathsf{guard\_of}.(\text{PROP}.p.q).s \wedge \neg s.(\mathsf{le\_rec}.p)$
$\mathbf{A}_3$: $s.(\mathsf{le\_rec}.\mathit{starter}) \wedge \mathit{sent\_to\_all\_non\_fathers.starter.s}$
We shall now try to reach a contradiction. From $\mathbf{A}_2$, we can, using (2)$_8$ through (7)$_8$ and 6.8$_{11}$, deduce that:
$\mathbf{A}_4$: $\neg \mathit{done.p.s}$
As a result, from Theorem 6.42$_{29}$ together with assumptions $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_4$, we can infer that:
$\mathbf{A}_5$: $\neg \mathit{done.starter.s}$
From Theorem 6.37$_{27}$ and assumption $\mathbf{A}_3$, we can derive that:
$\mathbf{A}_6$: *sent_to_all_neighs.starter*
Since the *starter*'s last event was a receive event, we can argue, due to the alternating send and receive behaviour of TARRY, that the *starter* has *rec_from_all_neighs*, and consequently ($\mathbf{A}_6$ and (7)$_8$) is *done*.

Obviously, this establishes the desired contradiction with assumption $\mathbf{A}_5$. In order to be able to prove that the *starter* is indeed done, we need to introduce a new invariant-candidate for Tarry. Since initially, the $\mathsf{le\_rec}$ variable of the *starter* is set to true, we state the following candidate:

$$cJ^3_{\text{TARRY}} \;=\; \begin{array}{l} \mathsf{le\_rec}.\mathit{starter} \Rightarrow (\mathsf{NR\_SENT}.\mathit{starter} = \mathsf{NR\_REC}.\mathit{starter}) \\ \wedge \\ \neg\mathsf{le\_rec}.\mathit{starter} \Rightarrow (\mathsf{NR\_SENT}.\mathit{starter} = \mathsf{NR\_REC}.\mathit{starter} + 1) \end{array}$$

Using 8.12_{38} and 8.13_{38}, this candidate suffices to prove – under the assumptions stated above – that the *starter* is *done*.

Verification of 18(a)-DONE when $x \neq$ *starter*

Now we know that x is not the *starter*, we have to substitute the left hand side of $\rightarrowtail$ in such a way that it implies the guard of $\text{DONE}_{\text{Tarry}}.x.y$. According to Theorems 8.6_{36}, 6.9_{11} and $(4)_8$, it suffices to prove that for arbitrary states s:

$$\begin{array}{l} J_{\text{PLUM}}.s \wedge J_{\text{Tarry}}.s \wedge s.(\mathsf{le\_rec}.x) \wedge \mathit{sent\_to\_all\_non\_fathers}.x.s \\ \Rightarrow \\ \quad \exists y \in \mathsf{neighs}.x : \quad \begin{array}{l} \mathit{rec\_from\_all\_neighs}.x.s \\ \wedge\ \mathit{sent\_to\_all\_non\_fathers}.x.s \\ \wedge\ \neg \mathit{sent\_to\_all\_neighs}.x.s \\ \wedge\ (y = (\mathsf{father}.x)) \end{array} \end{array}$$

Similar to the line of reasoning above, we introduce the following invariant-candidate for this purpose:

$$cJ^4_{\text{TARRY}} \;= \forall p \in \mathbb{P} : \; \begin{array}{l} \mathsf{le\_rec}.p \Rightarrow (\mathsf{NR\_REC}.p = \mathsf{NR\_SENT}.p + 1) \\ \wedge \\ \neg\mathsf{le\_rec}.p \Rightarrow (\mathsf{NR\_REC}.p = \mathsf{NR\_SENT}.p) \end{array}$$

Subsequently, 18(a)-DONE for the case that ($x \neq$ *starter*) is established as follows:

$\Leftarrow$($\rightarrowtail$ Substitution (B.2_{54}), cJ^4_{TARRY}, 6.9_{11}, 8.6_{36}, 8.12_{38}, and 8.16_{38})

$$\begin{array}{l} \forall x \in \mathbb{P}, (x \neq p), (x \neq \mathit{starter}) : \\ \quad {}_{\text{Tarry}}\vdash \quad \exists y \in \mathsf{neighs}.x : \\ \qquad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge \mathsf{guard\_of}.(\text{DONE}_{\text{Tarry}}.x.y) \\ \qquad \rightarrowtail \\ \qquad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}$$

$\Leftarrow$($\rightarrowtail$ Disjunction (B.2_{54}), $\rightarrowtail$ Introduction (B.3_{54}))

$$\begin{array}{l} \forall x \in \mathbb{P}, (x \neq p), (x \neq \mathit{starter}) : \\ \quad {}_{\text{Tarry}}\vdash \quad J_{\text{PLUM}} \wedge J_{\text{Tarry}} \\ \qquad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge \mathsf{guard\_of}.(\text{DONE}_{\text{Tarry}}.x.y) \\ \qquad \mathsf{ensures} \\ \qquad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}$$

Proving this $\mathsf{ensures}$-property is straightforward using Lemma 8.19_{39}.

Verification of 18(b)

For convenience, the proof obligation tackled in in this section is re-displayed below (from page 42):

$$\begin{array}{l} {}_{\text{Tarry}}\vdash \quad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p) \\ \qquad \rightarrowtail \\ \qquad \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \end{array}$$

Here we shall employ the proof-strategy explained on page 40. That is, we shall need to decompose the proof-obligation in such a way that we can use $\rightarrowtail$ Introduction (B.3_{54}) to prove that either $\text{IDLE}_{\text{Tarry}}$

or $\text{COL}_{\text{TARRY}}$ decreases Y or establishes $\mathsf{le\_rec}.p$. From Figure 18(b), we know that in this situation, there is a message in transit somewhere in the network. Moreover, using cJ^4_{PLUM}, cJ^2_{PLUM}, cJ^2_{PLUM} and $(1)_8$, it is not hard to prove that:

Theorem 8.22 *mit_IMP_not_rec_from_all_neighs*

$$\forall p \in \mathbb{P}, q \in \mathsf{neighs}.p : \frac{J_{\text{PLUM}}.s \wedge \mathsf{mit}.q.p.s}{\neg rec\_from\_all\_neighs.p.s}$$

Consequently, we can substitute the left hand side of $\rightarrowtail$ as follows: (we use the names z and w since these correspond to Figure 18(b))

$\Leftarrow$($\rightarrowtail$ Substitution (B.2_{54}), cJ^1_{TARRY}, Theorem 8.22_{45})

$$
\begin{array}{ll}
{}_{\text{TARRY}}\vdash & \exists z \in \mathbb{P}, w \in \mathsf{neighs}.z : \\
& \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p) \\
& \wedge\ \mathsf{mit}.w.z \wedge \neg rec\_from\_all\_neighs.z \\
& \rightarrowtail \\
& \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p))
\end{array}
$$

$\Leftarrow$($\rightarrowtail$ Disjunction (B.9_{55}))

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall z \in \mathbb{P}, w \in \mathsf{neighs}.z :} \\
{}_{\text{TARRY}}\vdash & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p) \\
& \wedge\ \mathsf{mit}.w.z \wedge \neg rec\_from\_all\_neighs.z \\
& \rightarrowtail \\
& \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p))
\end{array}
$$

If $(z = p)$, the proof obligation from above can be proved using $\rightarrowtail$ Introduction (B.3_{54}), since execution of $\text{COL}_{\text{TARRY}}.p.w$ will ensure that $\mathsf{le\_rec}.p$ is set to true.

Suppose $(z \neq p)$. Whether $\text{IDLE}_{\text{TARRY}}.z.w$ or $\text{COL}_{\text{TARRY}}.z.w$ is the action that will decrease Y, depends on whether z is idle or not. Therefore, we proceed as follows:

$\Leftarrow$($\rightarrowtail$ Case Distinction (B.6_{55}))

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall z \in \mathbb{P}, w \in \mathsf{neighs}.z, (z \neq p) :} \\
{}_{\text{TARRY}}\vdash & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p) \\
& \wedge\ \mathsf{mit}.w.z \wedge \neg rec\_from\_all\_neighs.z \wedge \mathsf{idle}.z \\
& \rightarrowtail \\
& \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \\
\wedge & \\
{}_{\text{TARRY}}\vdash & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p) \\
& \wedge\ \mathsf{mit}.w.z \wedge \neg rec\_from\_all\_neighs.z \wedge \neg\mathsf{idle}.z \\
& \rightarrowtail \\
& \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p))
\end{array}
$$

$\Leftarrow$($\rightarrowtail$ Substitution (B.2_{54}) on both conjuncts, using 6.6_{11}, 6.7_{11}, 8.3_{36}, 8.4_{36})

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall z \in \mathbb{P}, w \in \mathsf{neighs}.z, (z \neq p) :} \\
{}_{\text{TARRY}}\vdash & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p) \\
& \wedge\ \mathsf{guard\_of}.(\text{IDLE}_{\text{TARRY}}.z.w) \\
& \rightarrowtail \\
& \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p)) \\
\wedge & \\
{}_{\text{TARRY}}\vdash & \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k) \wedge (\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p) \\
& \wedge\ \mathsf{guard\_of}.(\text{COL}_{\text{TARRY}}.z.w) \\
& \rightarrowtail \\
& \mathsf{guard\_of}.(\text{PROP}.p.q) \wedge ((\neg\mathsf{le\_rec}.p \wedge (Y < k)) \vee (\mathsf{le\_rec}.p))
\end{array}
$$

Both conjuncts can be proved using $\rightarrowtail$ Introduction (B.3_{54}), and Lemma 8.19_{39}.

This ends the verification of **18(b)**, and hence of the **reach-PROP-part** (page 40), and consequently of the termination of Tarry. The one thing that remains to be done, is constructing Tarry's additional invariant. Gathering all the candidates introduced (i.e. cJ^1_{TARRY} through cJ^4_{TARRY}), analysing them, and verifying the stability of their conjunction results in the need to introduce yet three more invariant-candidates.

Definition 8.23 Tarry's additional invariant *Invariant_Tarry_Part*

$$
\begin{array}{lll}
J_{\text{Tarry}} = & & \\
& (\exists p \in \mathbb{P}, q \in \mathsf{neighs}.p : mit.p.q) = (\forall p \in \mathbb{P} : \neg\mathsf{le\_rec}.p) & cJ^1_{\text{TARRY}}, cJ^6_{\text{TARRY}} \\
\wedge & \forall p \in \mathbb{P} : \mathsf{le\_rec}.p \ \Rightarrow \neg\mathsf{idle}.p & cJ^2_{\text{TARRY}} \\
\wedge & \mathsf{le\_rec}.\mathit{starter} \Rightarrow (\mathsf{NR\_SENT}.\mathit{starter} = \mathsf{NR\_REC}.\mathit{starter}) & \\
& \wedge\ \neg\mathsf{le\_rec}.\mathit{starter} \Rightarrow (\mathsf{NR\_SENT}.\mathit{starter} = \mathsf{NR\_REC}.\mathit{starter} + 1) & cJ^3_{\text{TARRY}} \\
\wedge & \forall p \in \mathbb{P} : \ \mathsf{le\_rec}.p \Rightarrow (\mathsf{NR\_REC}.p = \mathsf{NR\_SENT}.p + 1) & \\
& \qquad\qquad \wedge\ \neg\mathsf{le\_rec}.p \Rightarrow (\mathsf{NR\_REC}.p = \mathsf{NR\_SENT}.p) & cJ^4_{\text{TARRY}} \\
\wedge & \forall p, x \in \mathbb{P}, q \in \mathsf{neighs}.p, y \in \mathsf{neighs}.x : & \\
& \quad \mathsf{mit}.p.q \wedge \mathsf{mit}.x.y \Rightarrow (p = x) \wedge (q = y) & cJ^5_{\text{TARRY}} \\
\wedge & \forall p, q \in \mathbb{P} : \mathsf{le\_rec}.p \wedge \mathsf{le\_rec}.q \Rightarrow (p = q) & cJ^7_{\text{TARRY}}
\end{array}
$$

Theorem 8.24 *STABLEe_Invariant_Tarry*

$$_{\text{Tarry}}\vdash \circlearrowright J_{\text{PLUM}} \wedge J_{\text{Tarry}}$$

Theorem 8.25 *INVe_Invariant_Tarry*

$$_{\text{Tarry}}\vdash \ \Box J_{\text{PLUM}} \wedge J_{\text{Tarry}}$$

Figure 19: Tarry's invariant

Again, since the verification activities are not all that exciting, we shall just state the required candidates. The first one comes as no surprise and states that, if there is a message in transit it is the only one:

$$\boxed{cJ^5_{\text{TARRY}} = \ \forall p, x \in \mathbb{P}, q \in \mathsf{neighs}.p, y \in \mathsf{neighs}.x : \ \mathsf{mit}.p.q \wedge \mathsf{mit}.x.y \Rightarrow (p = x) \wedge (q = y)}$$

The second and the third one together state that if there is no message in transit, then there is exactly one process that has received a message:

$$\boxed{cJ^6_{\text{TARRY}} = \neg(\exists p \in \mathbb{P}, q \in \mathsf{neighs}.p : \mathsf{mit}.p.q) \Rightarrow (\exists p \in \mathbb{P} : \mathsf{le\_rec}.p)}$$

$$\boxed{cJ^7_{\text{TARRY}} = \forall p, q \in \mathbb{P} : \mathsf{le\_rec}.p \wedge \mathsf{le\_rec}.q \Rightarrow (p = q)}$$

Since, cJ^1_{TARRY} and cJ^6_{TARRY} can be coalesced into one candidate using equality, we have derived a characterisation of J_{Tarry} that is displayed in Figure 19.

9 Using refinements to derive termination of DFS

This section shall describe how termination of the DFS algorithm is proved using the refinements framework from [VS01], and the already proven fact that:

$$\forall J :: \text{Tarry} \sqsubseteq_{\mathcal{R}\_\text{Tarry\_dfs},\ J} \text{DFS}$$

The UNITY specification reads:

Theorem 9.1 *HYLO_DFS*

$$J_{\text{PLUM}} \wedge J_{\text{Tarry}} \wedge J_{\text{DFS}} \ _{\text{DFS}}\vdash \textbf{ini}\text{DFS} \rightsquigarrow \forall p : p \in \mathbb{P} : \mathit{done}.p$$

where invariant J_{DFS} captures additional safety properties for DFS (if any). Using $\circlearrowright$ Preservation Theorem 3.8_4, it is straightforward to derive:

Theorem 9.3 *guard_of_IDLE_DFS*

$\mathsf{guard\_of}.(\text{IDLE}_{\text{DFS}}.p.q) = \mathsf{guard\_of}.(\text{IDLE}_{\text{TARRY}}.p.q)$

Theorem 9.4 *guard_of_COL_DFS*

$\mathsf{guard\_of}.(\text{COL}_{\text{DFS}}.p.q) = \mathsf{guard\_of}.(\text{COL}_{\text{TARRY}}.p.q)$

Theorem 9.5 *guard_of_PROP_lp_rec_DFS*

$\mathsf{guard\_of}.(\text{PROP\_LP\_REC}.p.q) = \mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.p.q) \wedge q = \mathsf{lp\_rec}.p$

Theorem 9.6 *guard_of_PROP_not_lp_rec_DFS*

$\mathsf{guard\_of}.(\text{PROP\_NOT\_LP\_REC}.p.q) = \mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.p.q) \wedge \neg cp.p.(\mathsf{lp\_rec}.p)$

Theorem 9.7 *guard_of_DONE_DFS*

$\mathsf{guard\_of}.(\text{DONE}_{\text{DFS}}.p.q) = \mathsf{guard\_of}.(\text{DONE}_{\text{TARRY}}.p.q)$

Figure 20: Guards of the actions from DFS

Theorem 9.2 *STABLEe_Invariant_in_DFS*

$_{\text{DFS}}\vdash \circlearrowleft (J_{\text{PLUM}} \wedge J_{\text{TARRY}})$

The stability of: $_{\text{DFS}}\vdash \circlearrowleft (J_{\text{PLUM}} \wedge J_{\text{TARRY}} \wedge J_{\text{DFS}})$ will be implicitly assumed throughout the verification process. For ease of reference, Figure 20 displays theorems about the guards of DFS's actions. And again, for readability we introduce the notational convention that: $\vdash$ abbreviates $J_{\text{PLUM}} \wedge J_{\text{TARRY}} \wedge J_{\text{DFS}}\ {}_{\text{DFS}}\vdash$.

Termination of DFS is proved using property preserving Theorem 3.3_5. The reasons for using this Theorem are twofold. First, since every PROP action in TARRY is bitotally related to two actions in DFS (namely PROP_LP_REC and PROP_NOT_LP_REC), we need to be able to pick one of those DFS PROP-actions when proving that the guards of TARRY's PROP-actions eventually implies the guards of related DFS's PROP-actions. Consequently, we cannot use preservation theorems 3.6_5 or 3.5_5. The second reason for using 3.3_5 is *not* because 3.4_5 cannot be used, but because it reduces proof effort. As we have seen during TARRY's verification, Lemma 8.19_{39} was very useful when proving unless and ensures properties that involved Y. A similar lemma can easily be proved for the actions of DFS, and hence verification of unless and ensures properties involving Y in the context of DFS will be simple too.

Lemma 9.8 *A_DECR_Y*

For arbitrary processes $p \in \mathbb{P}$, $q \in \mathsf{neighs}.p$, and actions $A \in \{\text{IDLE}_{\text{DFS}}, \text{COL}_{\text{DFS}}, \text{PROP\_LP\_REC}, \text{PROP\_NOT\_LP\_REC}, \text{DONE}_{\text{DFS}}\}$:

$$\forall k :: \frac{J_{\text{PLUM}}.s \wedge A.p.q.s.t \wedge \mathsf{guard\_of}.(A.p.q).s \wedge (Y.s = k)}{Y.t < k}$$

Therefore, we decided to use 3.3_5, although a function that is non-increasing with respect to some well-founded relation is *not* needed in order to be able to prove that falsification of the guards of DFS's PROP-actions go hand in hand with the falsification of the guards of TARRY's PROP-actions.

As a result, the initial specification stating termination of DFS is decomposed as follows:

$_{\text{DFS}}\vdash \textbf{ini}\text{DFS} \rightsquigarrow \forall p : p \in \mathbb{P} : done.p$

$\Leftarrow$(Theorem 3.4_5, 8.1_{35}, 5.3_{10})
For some well-founded relation $\prec$:

$$\exists W :: (\mathbf{w}\mathrm{DFS} = \mathbf{w}\textsc{Tarry} \cup W) \wedge ((J_{\textsc{plum}} \wedge J_{\textsc{Tarry}})\ \mathcal{C}\ W^{\mathsf{c}}) \wedge (\mathbf{w}\textsc{Tarry} \subseteq W^{\mathsf{c}})$$

$\wedge$

$$\forall A_D : A_D \in \mathbf{a}\mathrm{DFS} \wedge (\exists A_T :: A_T \in \mathbf{a}\textsc{Tarry} \wedge (A_T\ \mathcal{R}\_{\textsc{Tarry\_dfs}}\ A_D)) : (\mathsf{guard\_of}.A_D\ \mathcal{C}\ \mathbf{w}\mathrm{DFS})$$

$\wedge$

$$\left.\begin{array}{l} \forall A_T\ A_D\ :\ A_T \in \mathbf{a}\textsc{Tarry} \\ {}_{\textsc{dfs}}\vdash\ \ \mathsf{guard\_of}.A_T \\ \qquad \rightarrowtail \\ \qquad (\exists A_D :: (A_T\ \mathcal{R}\_{\textsc{Tarry\_dfs}}\ A_D) \wedge \mathsf{guard\_of}.A_D) \end{array}\right\} \textbf{reach} - \textbf{part}$$

$\wedge$

$$\left.\begin{array}{l} \exists M :: (M\ \mathcal{C}\ \mathbf{w}\mathrm{DFS}) \\ \quad \wedge \\ \quad \forall k ::\ {}_{\textsc{dfs}}\vdash (J_{\textsc{plum}} \wedge J_{\textsc{Tarry}} \wedge J_{\textsc{dfs}} \wedge M = k)\ \mathsf{unless}\ (M \prec k) \\ \quad \wedge \\ \quad \forall k\ A_T\ A_D : A_T \in \mathbf{a}\textsc{Tarry} \wedge A_T\ \mathcal{R}\_{\textsc{Tarry\_dfs}}\ A_D : \\ \qquad {}_{\textsc{dfs}}\vdash\ \ (J_{\textsc{plum}} \wedge J_{\textsc{Tarry}} \wedge J_{\textsc{dfs}} \wedge \mathsf{guard\_of}.A_D \wedge M = k) \\ \qquad\qquad \mathsf{unless} \\ \qquad\qquad (\neg(\mathsf{guard\_of}.A_T) \vee M \prec k) \end{array}\right\} \textbf{unless} - \textbf{part}$$

Since, $\mathsf{lp\_rec}.p$ variables are superimposed on TARRY in order to obtain DFS, the first conjunct is instantiated with the set $\{\mathsf{lp\_rec}.p \mid p \in \mathbb{P}\}$. Proving that $J_{\textsc{plum}}$ and $J_{\textsc{Tarry}}$ are confined by the complement of this set is tedious but straightforward, since the variables $\mathsf{le\_rec}$ do not appear in it. Similarly, proving that the guards of the actions in DFS are confined by DFS's write variables (i.e. the second conjunct) is not complicated.

The **unless-part** is now easy to prove by instantiating with Y (Definition 8.17_{38}):

- proving that Y is confined by the write variables of DFS is easy using Theorem 8.18_{38} and monotonicity of confinement $\mathrm{A}.2_{52}$
- proving that Y is non-increasing in DFS, can be proved using unless PRESERVATION (Theorem 3.7_4), and Theorem 8.20_{39}.
- proving that falsification of the guards of DFS's actions go hand in hand with the falsification of the guards of related TARRY's actions is easy using Lemma 9.8_{47}.

For the **reach-part**, the IDLE, COL, and DONE cases can be proved using $\rightarrowtail$ INTRODUCTION ($\mathrm{B}.3_{54}$). As a consequence, we are left with the PROP case:

$$\begin{array}{l} {}_{\textsc{dfs}}\vdash\ \ \mathsf{guard\_of}.(\textsc{prop}_{\textsc{Tarry}}.p.q) \\ \qquad \rightarrowtail \\ \qquad (\exists A_D :: (\textsc{prop}_{\textsc{Tarry}}.p.q\ \mathcal{R}\_{\textsc{Tarry\_dfs}}\ A_D) \wedge \mathsf{guard\_of}.A_D) \end{array}$$

This case states that: from a situation in which $\mathsf{guard\_of}.(\textsc{prop}.p.q)$ holds, we will eventually reach a situation in which either the guard of action PROP_LP_REC.$p.q$ or PROP_NOT_LP_REC.$p.q$ holds. To explain the proof-strategy that is used to verify this proof obligation, we refer to Figure 21. The p and q in the picture correspond to the p and q in the proof obligation, z is an arbitrary process. In Figure 21 we are in the situation that the guard of $\textsc{prop}_{\textsc{Tarry}}.p.q$ holds, that is (Theorem 8.5_{36}):

$$\mathsf{guard\_of}.(\textsc{prop}.p.q) \wedge \mathsf{le\_rec}.p$$

Process p has just received the message, and therefore is the only process that can do something. There are now two possibilities:

$q = \mathsf{lp\_rec}.p$ In this case the guard of PROP_LP_REC.$p.q$ holds and we are done.

$q \neq \mathsf{lp\_rec}.p$ In this case the guard of PROP_LP_REC.$p.q$ cannot hold. Again there are two possibilities:

$\neg\ cp.p.(\mathsf{lp\_rec}.p)$, that is p is not allowed to propagate a message to the process it has received its last message from. In this case, p can pick any non-father-neighbour to which it has not yet sent a message. Evidently, we can pick q, and as a consequence, the guard of PROP_NOT_LP_REC.$p.q$ is enabled.

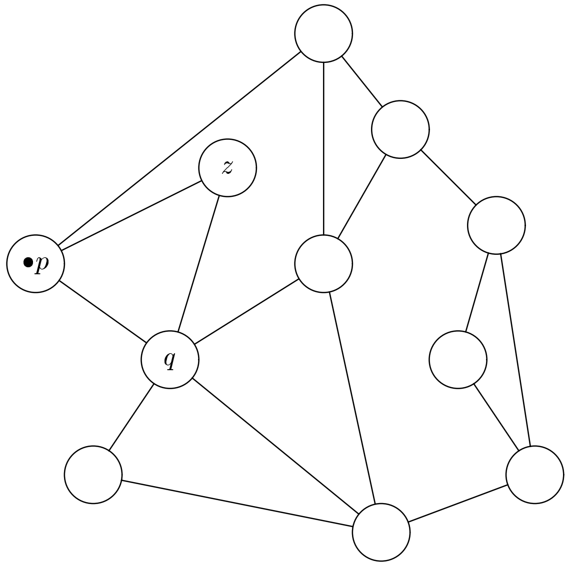


Figure 21: Situation when $\mathsf{guard\_of}.(\text{PROP}.p.q) \land \mathsf{le\_rec}.p$ holds

$cp.p.(\mathsf{lp\_rec}.p)$ In this case p *has to* send a message to the process it has received its last message from, and since this is not q, neither the guard of PROP_LP_REC.$p.q$ nor PROP_NOT_LP_REC.$p.q$ holds. If z (from Figure 21) is equal to $\mathsf{lp\_rec}.p$, then the guard of PROP_LP_REC.$p.z$ is enabled and consequently p shall send a message to z. Since $(z \neq q)$, we know that afterward the following holds:

$$\mathsf{guard\_of}.(\text{PROP}.p.q) \land \neg\mathsf{le\_rec}.p$$

Now we find ourself in the situation in Figure 18_{41}(b), from which we can transfer to situation in Figure 18_{41}(a) or Figure 21. Again, a well-foundedness argument, using $\rightarrowtail$ BOUNDED PROGRESS (B.10_{55}), shall enable us to prove that we cannot infinitely go back and forth between these situations, and therefore that eventually the guard of PROP_LP_REC.$p.q$ or PROP_NOT_LP_REC.$p.q$ will be enabled.

Consequently, when we use non-increasing function Y again for this well-foundedness argument, the proof of DFS's **reach**-PROP-**part** shall resemble that of TARRY's (see page 40). Therefore we shall only present the begin of the proof, which is slightly different from TARRY.

$$
\begin{array}{ll}
{}_{\text{DFS}}\vdash & \mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.p.q) \\
 & \rightarrowtail \\
 & (\exists A_D :: (\text{PROP}_{\text{TARRY}}.p.q\ \mathcal{R}\_{\text{TARRY\_DFS}}\ A_D) \land \mathsf{guard\_of}.A_D) \\
\Leftarrow(\rightarrowtail & \text{CASE DISTINCTION (B.}6_{55})) \\
{}_{\text{DFS}}\vdash & \mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.p.q) \land q = \mathsf{lp\_rec}.p \\
 & \rightarrowtail \\
 & (\exists A_D :: (\text{PROP}_{\text{TARRY}}.p.q\ \mathcal{R}\_{\text{TARRY\_DFS}}\ A_D) \land \mathsf{guard\_of}.A_D) \\
\land & \\
{}_{\text{DFS}}\vdash & \mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.p.q) \land q \neq \mathsf{lp\_rec}.p \\
 & \rightarrowtail \\
 & (\exists A_D :: (\text{PROP}_{\text{TARRY}}.p.q\ \mathcal{R}\_{\text{TARRY\_DFS}}\ A_D) \land \mathsf{guard\_of}.A_D)
\end{array}
$$

$\Leftarrow$($\rightarrowtail$ INTRODUCTION (B.3_{54}), and 9.5_{47} proves first conjunct,
$\rightarrowtail$ CASE DISTINCTION (B.6_{55}) on second conjunct)

$$
\begin{array}{ll}
{}_{\text{DFS}}\vdash & \mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.p.q) \land q \neq \mathsf{lp\_rec}.p \land \neg cp.p.(\mathsf{lp\_rec}.p) \\
 & \rightarrowtail \\
 & (\exists A_D :: (\text{PROP}_{\text{TARRY}}.p.q\ \mathcal{R}\_{\text{TARRY\_DFS}}\ A_D) \land \mathsf{guard\_of}.A_D) \\
\land &
\end{array}
$$

$_{\text{DFS}}\vdash$ $\mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.p.q) \wedge q \neq \mathsf{lp\_rec}.p \wedge cp.p.(\mathsf{lp\_rec}.p)$
$\rightarrowtail$
$(\exists A_D :: (\text{PROP}_{\text{TARRY}}.p.q\ \mathcal{R}\_{\text{TARRY\_DFS}}\ A_D) \wedge \mathsf{guard\_of}.A_D)$

$\Leftarrow$($\rightarrowtail$ INTRODUCTION (B.3$_{54}$), and 9.6$_{47}$ proves first conjunct,
$\rightarrowtail$ TRANSITIVITY (B.5$_{55}$) on second conjunct)

$_{\text{DFS}}\vdash$ $\mathsf{guard\_of}.(\text{PROP}_{\text{TARRY}}.p.q) \wedge q \neq \mathsf{lp\_rec}.p \wedge cp.p.(\mathsf{lp\_rec}.p)$
$\rightarrowtail$
$\mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p$
$\wedge$
$_{\text{DFS}}\vdash$ $\mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p$
$\rightarrowtail$
$(\exists A_D :: (\text{PROP}_{\text{TARRY}}.p.q\ \mathcal{R}\_{\text{TARRY\_DFS}}\ A_D) \wedge \mathsf{guard\_of}.A_D)$

$\Leftarrow$($\rightarrowtail$ INTRODUCTION (B.3$_{54}$), PROP_LP_REC.p.($\mathsf{lp\_rec}.p$) establishes $\neg$ $\mathsf{le\_rec}.p$,
$\rightarrowtail$ BOUNDED PROGRESS (B.10$_{55}$) on second conjunct)

$_{\text{DFS}}\vdash$ $\mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y = k)$
$\rightarrowtail$
$(\mathsf{guard\_of}.(\text{PROP}.p.q) \wedge \neg\mathsf{le\_rec}.p \wedge (Y < k))$
$\vee$
$(\exists A_D :: (\text{PROP}_{\text{TARRY}}.p.q\ \mathcal{R}\_{\text{TARRY\_DFS}}\ A_D) \wedge \mathsf{guard\_of}.A_D)$

From here, the proof is similar to that of TARRY (starting at page 40), and hence is not repeated. We end the verification of DFS's termination by observing that the verification of DFS did not need any more safety properties, and thus that J_{DFS} can defined to be true.

Definition 9.9 *Invariant_DFS*

$J_{\text{DFS}} = \mathsf{true}$

10 Concluding remarks

Although this is a tough report to read (as well as write), we think we have succeeded in presenting intuitive and structured proofs of the correctness of distributed hylomorphisms with respect to their termination. Due to the incremental, demand-driven construction of the invariant, the latter is not "pulled out of a hat" [Cho95], and the purpose of its various conjuncts are well motivated. Moreover, since, various property preservation theorems are necessary throughout the verification process, this report also serves as an illustration of the usage and effectiveness of the refinement framework from [VS01].

11 HOL theories

All results in this report have been verified with HOL [GM93]. The approach used to verify the distributed hylomorphisms is reflected in the resulting hierarchy of HOL theories, which is depicted in Figure 22.

network is the theory about centralised and decentralised connected networks described in Section 4.

RST constitutes the theory about rooted spanning trees described in Section 6.7.

communication contains the theory about asynchronous communication from Section 4.

Distributed_Hylomorphisms embodies definitions (1)$_8$ through (7)$_8$.

PLUM formalises the PLUM algorithm.

PLUM_INV defines and proves the invariant of PLUM.

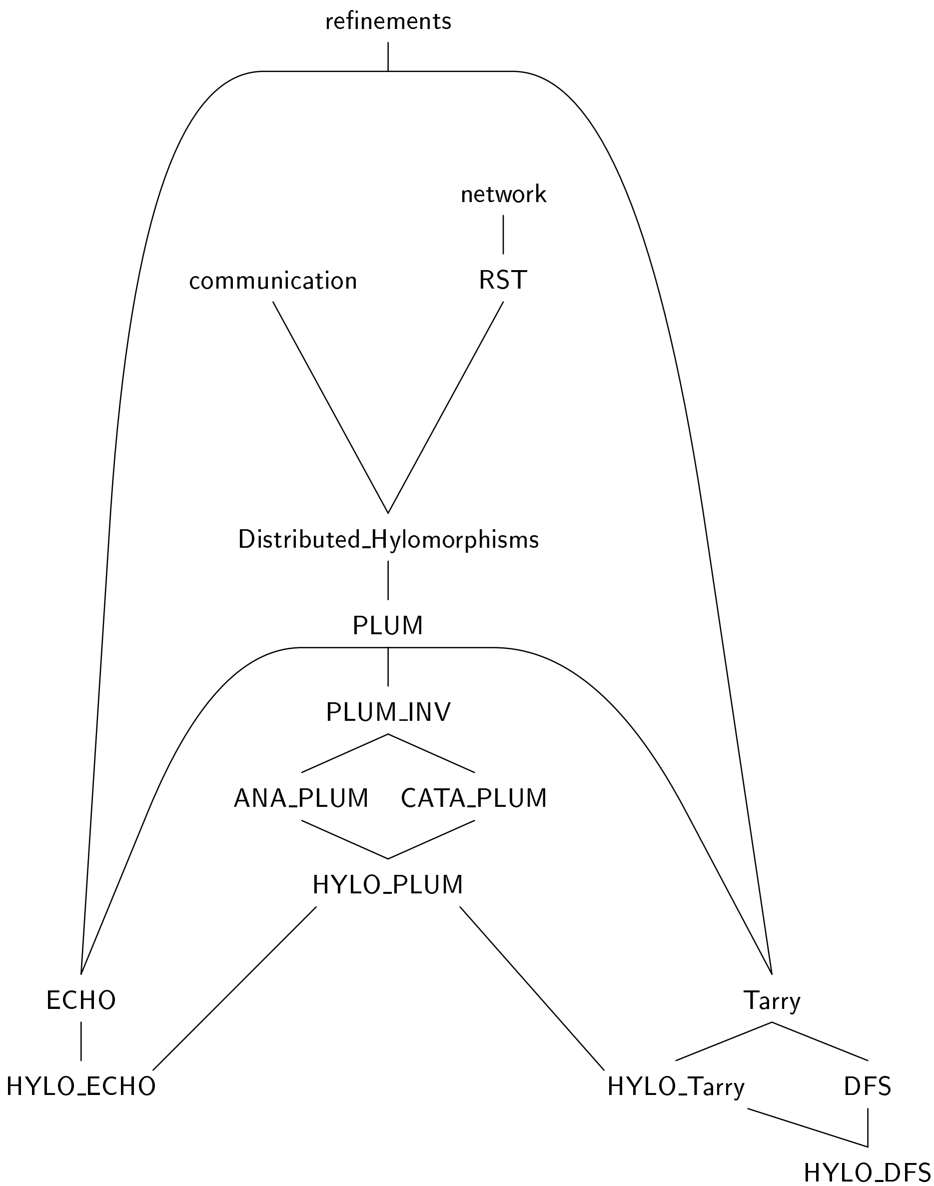


Figure 22: Theory hierarchy

ANA_PLUM contains the proof of the anamorphism part of the distributed hylomorphism, i.e. the construction of a rooted spanning tree.

CATA_PLUM contains the proof of the catamorphism part, i.e. using the rooted spanning tree to establish the desired result.

HYLO_PLUM combines the anamorphism and catamorphism part to prove termination of PLUM.

ECHO, Tarry, DFS formalise ECHO, TARRY, and DFS respectively, and contain Theorems 5.1 through 5.3.

HYLO_ECHO, HYLO_Tarry, HYLO_DFS prove termination of ECHO, TARRY, and DFS respectively by using the refinement framework.

The resulting theories can be obtained by sending an email to one of the authors.

A Preliminaries: states, actions, programs and specifications

A.1 Variables, values, states

We assume we have a universe `Var` of program variables and a universe `Val` of values that these variables can take. Program states will be modelled as functions that are elements of `Var`→`Val`, and the set of all program states will be denoted by `State`. A state-predicate is an element of `State`→`bool`. We say that a state-predicate p is *confined by* a set of variables $V \subseteq$ `Var` if p does not restrict the value of any variable outside V. Let us write $s =_V t$, if all variables in V have the same values in state s and t (i.e. $\forall v : v \in V : s.v = t.v$). Now we can formally define predicate confinement as follows:

Definition A.1 CONFINEMENT *CONF_DEF*

$$p\,\mathcal{C}\,V \stackrel{d}{=} \forall s,t : s =_V t : p.s = p.t$$

The confinement operator is monotonic in its second argument.

Theorem A.2 $\mathcal{C}$ MONOTONICITY *CONF_MONO*

$$\forall f :: V \subseteq W \wedge (f\,\mathcal{C}\,V) \Rightarrow (f\,\mathcal{C}\,W)$$

A.2 Actions

Actions can be (multiple) assignments or guarded (if-then) actions. Simultaneous execution of assignments is modelled by the operator $\|$. For example, $x, y := 1, 2 \,\|\, w, z := 3, 4$ equals $x, y, z, w := 1, 2, 3, 4$.

All actions is this report are assumed to be well-formed, meaning that their guard is a state-predicate, and the amount of variables at the left hand side of the := is equal to the amount of values at the right hand side.

We will assume a deep embedding of actions, i.e. the abstract syntax of actions is defined by a recursive data type `ACTION`, and their semantics is defined by a recursive function, e.g. compile, of type `ACTION`→(`State`→`State`→`Bool`). As a consequence, we are able to obtain and reason about various components of actions. For example, we assume that we have functions guard_of and assign_vars that given an action returns its guard and the set of variables it assigns to respectively. Examples of these functions:

guard_of(if $x > 0 \wedge y < 10$ then $x := x + 1 \,\|\, y := y - 1$) $= x > 0 \wedge y < 10$
assign_vars(if $x > 0 \wedge y < 10$ then $x := x + 1 \,\|\, y := y - 1$) $= \{x, y\}$

Moreover, we have functions is_assign and is_guard that enable us to check the type of an action.

An action that is always ready to make a transition is called *always enabled.*

Definition A.3 ALWAYS ENABLED ACTION *ALWAYS_ENABLED*

$$\Box_{\mathsf{En}} A \stackrel{d}{=} \forall s :: (\exists t :: \mathsf{compile}.A.s.t)$$

Multiple assignments and guarded if-then actions are always enabled. Note that this means that a guarded action with a false guard behaves like skip, i.e. the action that does not change the value of any variable.

Theorem A.10 ANTI-REFLEXIVITY *UNLESS_ANTI_REFL*

$$_P\vdash\; p \ \mathsf{unless}\ \neg p$$

Theorem A.11 $\circlearrowright$ CONJUNCTION

$$\frac{_P\vdash (\circlearrowright p) \ \wedge \ _P\vdash (\circlearrowright q)}{_P\vdash \circlearrowright (p \wedge q)}$$

Figure 23: Some theorems about unless and $\circlearrowright$

Definition A.4 SKIP ACTION *SKIP_DEF*

For any action A, $\mathsf{skip} \stackrel{d}{=} \mathsf{if\ false\ then}\ A$

A set of variables is V *ignored-by* an action A, denoted by $V \not\leftarrow A$, if executing A's executable in any state does not change the values of these variables. Variables in V^{c} *may* however be written by A.

Definition A.5 VARIABLES IGNORED-BY ACTION *dIG_BY_DEF*

$$V \not\leftarrow A \ \stackrel{d}{=}\ \forall s,t : \mathsf{compile}.A.s.t : s =_V t$$

A set of variables V is said to be *invisible-to* an action A, denoted by $V \not\twoheadrightarrow A$, if the values of the variables in V do not influence the result of A's executable, hence A only depends on the variables outside V.

Definition A.6 VARIABLES INVISIBLE-TO ACTION *dINVI_DEF*

$$V \not\twoheadrightarrow A \ \stackrel{d}{=}\ \forall s,t,s',t' : s =_{V^{\mathsf{c}}} s' \wedge t =_{V^{\mathsf{c}}} t' \wedge s' =_V t' \wedge \mathsf{compile}.A.s.t : \mathsf{compile}.A.s'.t'$$

A.3 Programs

UNITY programs P are modelled by a quadruple $(\mathbf{a}P, \mathbf{ini}P, \mathbf{r}P, \mathbf{w}P)$; $\mathbf{a}P$, is the set of actions separated by the symbol $[]$; $\mathbf{ini}P$ is the initial condition of the program; $\mathbf{r}P$ is the set of read variables; and $\mathbf{w}P$ the set of write variables.

A program execution of such a program is infinite, in each step an action is selected nondeterministically and executed. Selection is weakly fair, meaning that every action is selected infinitely often.

A.4 Specifications

As usual, reasoning about actions is done by means of Hoare triples [Hoa69]. If p and q are state-predicates, and A is an action, then $\{p\}\ A\ \{q\}$ means that if A is executed in any state satisfying p, it will end in a state satisfying q:

Definition A.7 HOARE TRIPLE *HOAe_DEF*

$$\{p\}\ A\ \{q\} \ \stackrel{d}{=}\ \forall s,t : p.s \wedge \mathsf{compile}.A.s.t : q.t$$

To reason about programs we will use the UNITY specification and proof logic from [CM89] augmented by [Pra95]. Safety properties can be specified by the following operators:

Definition A.8 UNLESS (SAFETY PROPERTY) *UNLESSe*

$$_P\vdash p\ \mathsf{unless}\ q \ \stackrel{d}{=}\ \forall A : A \in \mathbf{a}P : \{p \wedge \neg q\}\ A\ \{p \vee q\}$$

Definition A.9 STABLE PREDICATE *STABLEe*

$$_P\vdash \circlearrowright p \ \stackrel{d}{=}\ _P\vdash p\ \mathsf{unless\ false}$$

In Figure 23 some theorems about unless and $\circlearrowright$ are listed that we will need later in this report.
One-step progress properties are specified by:

Definition A.12 ENSURES (PROGRESS PROPERTY) *ENSURESe*

$${}_P\vdash p \ \mathsf{ensures}\ q \ \stackrel{d}{=}\ ({}_P\vdash p \ \mathsf{unless}\ q) \ \wedge\ (\exists A : A \in \mathbf{a}P : \{p \wedge \neg q\}\ A\ \{q\})$$

To specify general progress properties we will use Prasetya's [Pra95] reach ($\rightarrowtail$) and convergence ($\rightsquigarrow$) operators. The $\rightarrowtail$-operator is defined as the least disjunctive and transitive closure of $\mathsf{ensures}$:

Definition A.13 REACH OPERATOR *REACHe*
$(\lambda p, q.\ J\ {}_P\vdash p \rightarrowtail q)$ is defined as the smallest relation $\rightarrow$ satisfying:

Lifting $\dfrac{p\,\mathcal{C}\,\mathbf{w}P \ \wedge\ q\,\mathcal{C}\,\mathbf{w}P \ \wedge\ ({}_P\vdash\circlearrowright J) \ \wedge\ ({}_P\vdash J \wedge p \ \mathsf{ensures}\ q)}{p \rightarrow q}$

Transitivity $\dfrac{p \rightarrow q \wedge q \rightarrow r}{p \rightarrow r}$

Disjunctivity $\dfrac{\forall i : W.i : p_i \rightarrow q}{(\exists i : W.i : p_i) \rightarrow q}$

where $W \in \alpha{\rightarrow}\mathtt{Val}$ characterises a non-empty set.

Many properties about $\rightarrowtail$ can be found in [Pra95], the properties we need in this report are listed in Appendix B.

The $\rightsquigarrow$-operator defines a restricted form of self-stabilisation, a notion first introduced by Dijkstra in [Dij74]. Roughly speaking, a self-stabilising program is a program which is capable of recovering from arbitrary transient failures of the environment in which the program is executing. Obviously such programs are very useful, although the requirement to allow *arbitrary* failures may be too strong. A more restricted form of self-stabilisation, called convergence, allows a program to recover only from certain failures. In [Pra95], a convergence operator is defined in terms of $\rightarrowtail$:

Definition A.14 CONVERGENCE *CONe*

$$J\ {}_P\vdash p \rightsquigarrow q \ \triangleq\ q\,\mathcal{C}\,\mathbf{w}P \ \wedge\ (\exists q' :: (J\ {}_P\vdash p \rightarrowtail q' \wedge q) \ \wedge\ ({}_P\vdash\circlearrowright(J \wedge q' \wedge q)))$$

Again some properties taken from [Pra95] are listed in Appendix C. Most properties are analogous to those of $\rightarrowtail$. There is, however, one property that is satisfied by $\rightsquigarrow$ but not by $\rightarrowtail$ nor $\mapsto$, viz. CONJUNCTIVITY.

B Laws of $\rightarrowtail$

Theorem B.1 $\rightarrowtail$ STABLE BACKGROUND AND CONFINEMENT *REACHe_IMP_STABLE* *REACHe_IMP_CONF*

$$P : \ \dfrac{J \vdash p \rightarrowtail q}{\circlearrowright J \ \wedge\ p, q\,\mathcal{C}\,\mathbf{w}P}$$

Theorem B.2 $\rightarrowtail$ SUBSTITUTION *REACHe_SUBST*

$$P, J : \ \dfrac{p, s\,\mathcal{C}\,\mathbf{w}P \ \wedge\ [J \wedge p \Rightarrow q] \ \wedge\ (q \rightarrowtail r) \ \wedge\ [J \wedge r \Rightarrow s]}{p \rightarrowtail s}$$

Theorem B.3 $\rightarrowtail$ INTRODUCTION *REACHe_ENS_LIFT,REACHe_IMP_LIFT*

$$P, J : \ \dfrac{p, q\,\mathcal{C}\,\mathbf{w}P \ \wedge (\circlearrowright J) \ \wedge\ ([J \wedge p \Rightarrow q] \ \vee\ (J \wedge p \ \mathsf{ensures}\ q))}{p \rightarrowtail q}$$

Theorem B.4 $\rightarrowtail$ REFLEXIVITY *REACHe_REFL*

$$P, J : \ \dfrac{p\,\mathcal{C}\,\mathbf{w}P \ \wedge\ (\circlearrowright J)}{p \rightarrowtail p}$$

Theorem B.5 $\rightarrowtail$ TRANSITIVITY *REACHe_TRANS*

$$P, J : \frac{(p \rightarrowtail q) \ \wedge \ (q \rightarrowtail r)}{p \rightarrowtail r}$$

Theorem B.6 $\rightarrowtail$ CASE DISTINCTION *REACHe_DISJ_CASES*

$$P, J : \frac{(p \wedge \neg r \rightarrowtail q) \ \wedge \ (p \wedge r \rightarrowtail q)}{p \rightarrowtail q}$$

Theorem B.7 $\rightarrowtail$ CANCELLATION *REACHe_CANCEL*

$$P, J : \frac{q \, \mathcal{C} \, \mathbf{w}P \ \wedge \ (p \rightarrowtail q \vee r) \ \wedge \ (r \rightarrowtail s)}{p \rightarrowtail q \vee s}$$

Theorem B.8 $\rightarrowtail$ PROGRESS SAFETY PROGRESS (PSP) *REACHe_PSP*

$$P, J : \frac{r, s \, \mathcal{C} \, \mathbf{w}P \ \wedge \ (r \wedge J \ \mathsf{unless} \ s) \wedge \ (p \rightarrowtail q)}{p \wedge r \rightarrowtail (q \wedge r) \vee s}$$

Theorem B.9 $\rightarrowtail$ DISJUNCTION *REACHe_GEN_DISJe*

$$P, J : \frac{(\forall i : i \in W : p.i \rightarrowtail q.i)}{(\exists i : i \in W : p.i) \rightarrowtail (\exists i : i \in W : q.i)} \quad \text{if } W \neq \emptyset$$

Theorem B.10 $\rightarrowtail$ BOUNDED PROGRESS *REACHe_WF_INDUCT*

For a well-founded relation $\prec$ over some set W, and metric $M \in \mathtt{State} \rightarrow W$:

$$P, J : \frac{q \, \mathcal{C} \, \mathbf{w}P \ \wedge \ (\forall m \in W : p \wedge (M = m) \rightarrowtail (p \wedge (M \prec m)) \vee q)}{p \rightarrowtail q}$$

C Laws of $\rightsquigarrow$

Theorem C.1 CONVERGENCE IMPLIES PROGRESS *CONe_IMP_REACHe*

$$P, J : \frac{p \rightsquigarrow q}{p \rightarrowtail q}$$

Theorem C.2 $\rightsquigarrow$ SUBSTITUTION *CONe_SUBST*

$$P, J : \frac{p, s \, \mathcal{C} \, \mathbf{w}P \wedge [J \wedge p \Rightarrow q] \wedge \ (q \rightsquigarrow r) \ \wedge [J \wedge r \Rightarrow s]}{p \rightsquigarrow s}$$

Theorem C.3 $\rightsquigarrow$ INTRODUCTION *CONe_ENSURES_LIFT, CONe_IMP_LIFT*

$$P, J : \frac{p, q \, \mathcal{C} \, \mathbf{w}P \ \wedge \ (\circlearrowright J) \ \wedge \ (\circlearrowright (J \wedge q)) \ \wedge \ ([J \wedge p \Rightarrow q] \ \vee \ (p \wedge J \ \mathsf{ensures} \ q))}{p \rightsquigarrow q}$$

Theorem C.4 $\rightsquigarrow$ REFLEXIVITY *CONe_REFL*

$$P, J : \frac{p \, \mathcal{C} \, \mathbf{w}P \ \wedge \ (\circlearrowright J) \ \wedge \ (\circlearrowright (J \wedge p))}{p \rightsquigarrow p}$$

Theorem C.5 $\rightsquigarrow$ TRANSITIVITY *CONe_TRANS*

$$P, J : \frac{(p \rightsquigarrow q) \ \wedge \ (q \rightsquigarrow r)}{p \rightsquigarrow r}$$

Theorem C.6 $\rightsquigarrow$ CASE DISTINCTION *CONe_DISJ_CASES*

$$P, J : \frac{(p \wedge \neg r \rightsquigarrow q) \ \wedge \ (p \wedge r \rightsquigarrow q)}{p \rightsquigarrow q}$$

Theorem C.7 ACCUMULATION *CON_SPIRAL*

$$P, J : \frac{(p \rightsquigarrow q) \ \wedge \ (q \rightsquigarrow r)}{p \rightsquigarrow q \wedge r}$$

Theorem C.8 $\rightsquigarrow$ STABLE STRENGTHENING *CONe_STAB_MONO_GEN*

$$P : \frac{q \,\mathcal{C}\, \mathbf{w}P \ \wedge \ (\circlearrowleft (J_1 \wedge J_2)) \ \wedge \ J_1 \vdash p \rightsquigarrow q}{(J_1 \wedge J_2) \vdash p \rightsquigarrow q}$$

Theorem C.9 $\rightsquigarrow$ STABLE SHIFT *CONe_STABLE_SHIFT*

$$P : \frac{p' \,\mathcal{C}\, \mathbf{w}P \wedge (\circlearrowleft J) \ \wedge \ (J \wedge p' \vdash p \rightsquigarrow q)}{J \vdash p' \wedge p \rightsquigarrow q}$$

Theorem C.10 $\rightsquigarrow$ DISJUNCTION *CONe_GEN_DISJ*

$$P, J : \frac{(\forall i : i \in W : p.i \rightsquigarrow q.i)}{(\exists i : i \in W : p.i) \rightsquigarrow (\exists i : i \in W : q.i)} \quad \text{if } W \neq \emptyset$$

Theorem C.11 $\rightsquigarrow$ CONJUNCTION *CONe_CONJ*

For all *non-empty* and *finite* sets W:

$$P, J : \frac{(\forall i : i \in W : p.i \rightsquigarrow q.i)}{(\forall i : i \in W : p.i) \rightsquigarrow (\forall i : i \in W : q.i)}$$

Theorem C.12 $\rightsquigarrow$ BOUNDED PROGRESS *CONe_WF_INDUCT*

For a well-founded relation $\prec$ over some set A, and metric $M \in \texttt{State} \rightarrow A$:

$$P, J : \frac{(q \rightsquigarrow q) \ \wedge \ (\forall m \in A : p \wedge (M = m) \rightsquigarrow (p \wedge (M \prec m)) \vee q)}{p \rightsquigarrow q}$$

Theorem C.13 $\rightsquigarrow$ ITERATION *Iterate_thm_CONe*

For arbitrary sets W,

$$P, J, L : \frac{\begin{array}{c}(\circlearrowleft ((\forall x : x \in L : Q.x) \wedge J)) \wedge (\forall x : x \in L : Q.x \,\mathcal{C}\, \mathbf{w}P) \\ L \subseteq W \Rightarrow ((f.L) \subseteq W \ \wedge \ (\forall x : x \in L : Q.x) \rightsquigarrow (\forall x : x \in f.L : Q.x))\end{array}}{\forall n\ L : L \subseteq W \Rightarrow (\forall x : x \in L : Q.x) \rightsquigarrow (\forall x : x \in \mathsf{iterate}.n.f.L : Q.x)}$$

References

[Cho95] C.-T. Chou. Using operational intuition about events and causality in assertional proofs. Technical Report 950013, UCLA, Feb. 1995.

[CM89] K.M. Chandy and J. Misra. *Parallel Program Design.* Addison-Wesley, Austin, Texas, May 1989.

[Dij74] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communication of the ACM*, 17(11):643–644, 1974.

[GM93] M.J.C. Gordon and T.F. Melham. *Introduction to HOL.* CUP, 1993.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computers programs. *Commun. Ass. Comput. Mach.*, 12:576–583, 1969.

[Pra95] Wishnu Prasetya. *Mechanically Supported Design of Self-stabilizing Algorithms.* PhD thesis, UU, Oct 1995.

[Vos00] T.E.J. Vos. *UNITY in Diversity, A stratified approach to the verification of distributed algorithms.* PhD thesis, Utrecht University, 2000.

[VS01] T.E.J. Vos and S.D. Swierstra. Program refinement in UNITY. Technical Report UU-CS-2001-41, Utrecht University, 2001.