XI. Synchronizing a Tree-shaped Network (October 2008)

In this chapter, we develop another distributed program example, where we shall encounter another interesting mathematical object: a tree. We shall thus learn how to formalize such a data structure and see how we can fruitfully reason about it using an induction rule. This example has been treated by many researchers: we have taken it from the following books [1], [2].

1 Introduction

In this example we have a network of nodes, which is slightly more complicated than in the previous chapter where we were dealing with a ring. Here we have a tree as indicated in figure 1.

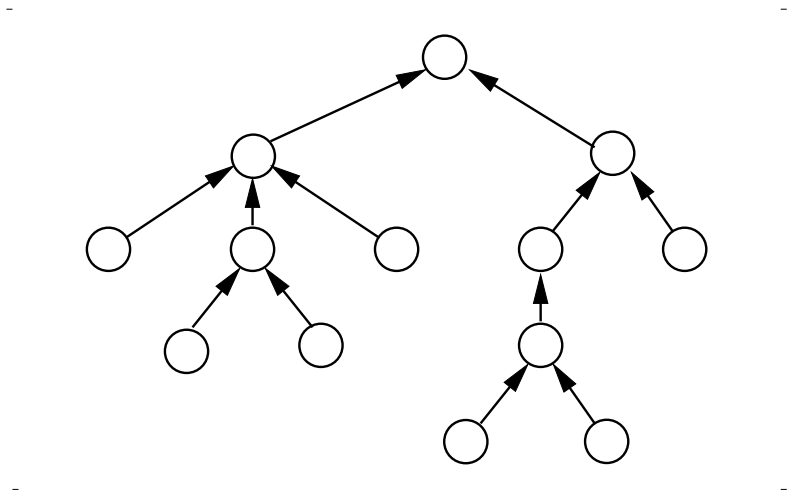


Fig. 1. A Tree-shaped Network

We have a set of nodes forming a tree	ENV-1

At each node of the tree, we have a process performing a certain task, which is the same for all processes (the exact nature of this task is not important). The constraint we want these processes to observe is that they remain *synchronized*. In other word, no one of them should be able to progress too much with regards to the others. In order to formalize this synchronization constraint, we assign a counter to each node of the tree. Intuitively, each counter represents the phase within which each process is currently running.

Each node has a counter, which is a natural number	FUN-1

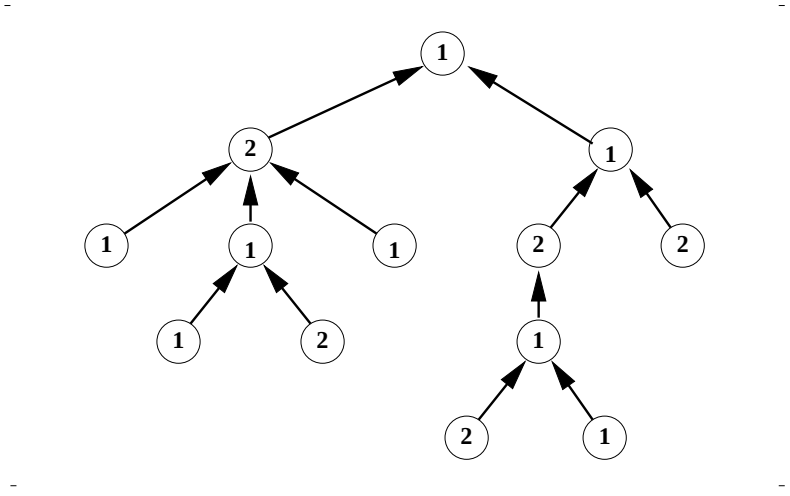


Fig. 2. A Tree with Counters at the Nodes

In order to express that each process is at most one phase ahead of the others, we simply state that the difference between any two of these counters is at most equal to 1. This is illustrated on figure 2.

The difference between any two counters is at most equal to 1	FUN-2

An additional constraint of our distributed algorithm states that each process can *read the counters of its immediate neighbors only*.

Each node can only read the counter of its immediate neighbors in the tree	FUN-3

Moreover, each process is *allowed to modify its own counter only*.

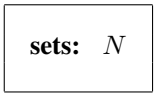
The counter of a node can be modified by this node only	FUN-4

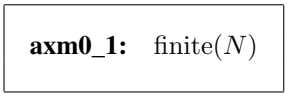
One of the most important aspect of our approach is that the local constraint as expressed in FUN-3 has not to be followed right from the beginning of the construction process. During the early phases, we feel free to magically have access everywhere from any node. During the refining phases however, we shall gradually strengthen the guards of the events so that this local constraint will be obeyed eventually. Again, this reveals an important distinction between a model and a program.

2 Initial Model

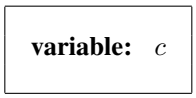
2.1 The State

We now proceed with the first formalization. The network is defined from a carrier set N of nodes. Our only property about this set is that it is finite. This is expressed below as axiom **axm0_1**.





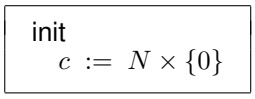
At this stage, we do not need to define the tree, the only state variable we have is the function c defining the counter value at each node (all initialized to 0). We now define two invariants involving c. First its basic invariant **inv0_1** stating that c is a total function from N to the set $\mathbb{N}$ natural numbers, and then another invariant, **inv0_2**, expressing the basic synchronizing requirement FUN-2: the difference between two node counters is at most one, so that each counter value is kept smaller than or equal to the value of any other counter plus one.

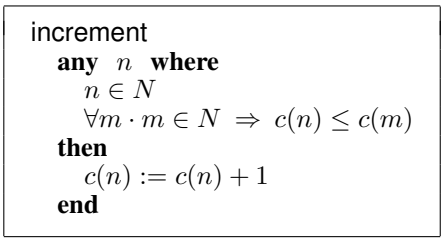


Invariant **inv0_2** seems a bit surprising at first glance. Does it state that the difference between the values of two counters is at most equal to 1? Let a and b be two such values. If $a \leq b$ then we have $b = a + d$ with $d \geq 0$, then we certainly have $a + d <= a + 1$ (that is, $d \leq 1$) since $b \leq a + 1$ according to **inv0_2**. In other words, in the case where $a \leq b$ then $b - a \in 0 \,..\, 1$. A similar reasoning would show that if $b \leq a$ then $a - b \in 0 \,..\, 1$.

2.2 The Events

Besides event init, our only other event, increment, makes explicit the conditions under which a node n can progress. It is obviously the case when its counter $c(n)$ is not greater than that of any other counter m, that is when $c(n) \leq c(m)$. In this case only, can the node increment its own counter without destroying the synchronizing invariant **inv0_2**. As can be seen (and as announced above), we have supposed that a given node n has free access to all other nodes in the tree: again, this is because we are here in an abstraction where every access is still possible. Note how we initialize the counters in event init to be all equal to 0.





2.3 The Proofs

The proofs of the statements expressing that event init establishes the invariants are trivial. We shall only informally develop the proof of the statement expressing the preservation of invariant **inv0_2** by event increment. Here is what we have to prove:

Invariant **inv0_1**	$c \in N \to \mathbb{N}$
Invariant **inv0_2**	$\forall x, y \cdot x \in N \ \wedge \ y \in N \ \Rightarrow \ c(x) \leq c(y) + 1$
Guards of	$n \in N$
event increment	$\forall m \cdot m \in N \ \Rightarrow \ c(n) \leq c(m)$
$\vdash$	$\vdash$
Mod. inv. **inv0_2**	$\forall x, y \cdot x \in N \ \wedge \ y \in N \ \Rightarrow \ (c \lhd \{n \mapsto c(n)+1\})(x) \ \leq$ $(c \lhd \{n \mapsto c(n)+1\})(y) + 1$

This statement can be simplified to the following by removing the universal quantification in the consequent and moving then both predicates $x \in N$ and $y \in N$ in the antecedent (rules ALL_R and IMP_R):

$$
\begin{array}{l}
c \in N \to \mathbb{N} \\
\forall x, y \cdot x \in N \wedge y \in N \ \Rightarrow \ c(x) \leq c(y) + 1 \\
n \in N \\
\forall m \cdot m \in N \ \Rightarrow \ c(n) \leq c(m) \\
x \in N \\
y \in N \\
\vdash \\
(c \lhd \{n \mapsto c(n)+1\})(x) \ \leq \ (c \lhd \{n \mapsto c(n)+1\})(y) + 1
\end{array}
$$

The proof now proceeds by cases (four of them, in fact) as is usual when one deals with the overriding operator $\lhd$:

(1) When x and y are both equal to n, then it results in the following goal, which is trivial:

$$c(n) + 1 \ \leq \ c(n) + 1 + 1$$

(2) When x is equal to n while y is not, then it results in the following goal, which is trivial according to the guarding condition, $\forall m \cdot m \in N \ \Rightarrow \ c(n) \leq c(m)$ of event ascending (instantiate m with y):

$$c(n) + 1 \ \leq \ c(y) + 1$$

(3) When x is not equal to n while y is, then it results in the following goal, which is trivial according to invariant **inv0_2** (instantiate x with x and y with n) :

$$c(x) \ \leq \ c(n) + 1$$

(4) When x and y are both not equal to n, then it results in the following goal, which is trivial according to invariant **inv0_2** (instantiate x with x and y with y):

$$c(x) \ \leq \ c(y) + 1$$

Note that such a proof can be done automatically by a prover.

3 First Refinement

3.1 The State

The problem with the guard of the event ascending proposed in the previous section (it is copied below) is that we have to compare the value $c(n)$ of the counter at node n, with that of *all* other counters $c(m)$.

increment
any n **where**
$n \in N$
$\forall m \cdot m \in N \ \Rightarrow \ c(n) \leq c(m)$
then
$c(n) := c(n) + 1$
end

In order to solve this problem (partially in this refinement), we have to introduce the tree structure of the network. This is what we are going to do in this section. The tree is defined by three constants: its root r, a set of leafs L and also its parent function f. We borrow the axiomatic definition of finite trees as well as the tree induction rule presented in section 7.7 of chapter 9.

constants: r L f

axm1_1 : $r \in N$

axm1_2 : $L \subseteq N$

axm1_3 : $f \in N \setminus \{r\} \twoheadrightarrow N \setminus L$

axm1_4 : $\forall T \cdot r \in T \ \wedge \ f^{-1}[T] \subseteq T \ \Rightarrow \ N \subseteq T$

In order to limit the number of comparisons which a node had to perform in the abstract version of event increment, the idea is to suppose that the value $c(r)$ of the counter at the root r of the tree is always smaller than or equal to that of any other counter. We have thus $c(r) \leq c(m)$ for *all* nodes m. In that case, it is sufficient to reduce the guard of event increment to a mere comparison of $c(n)$ with $c(r)$: when the equality holds, that is when we have $c(n) = c(r)$, then clearly $c(n) \leq c(m)$ holds for *all* nodes m and we can thus safely increment the counter of n. We could have stated the above rule - that is, $c(r) \leq c(m)$ for all nodes m - as an invariant but we choose to have the following more primitive invariant **inv1_1**:

inv1_1: $\forall m \cdot m \in N \setminus \{r\} \ \Rightarrow \ c(f(m)) \leq c(m)$

In order to maintain the property $c(r) \leq c(m)$ for *all* nodes m, it is sufficient to have the counter of the parent of each node m (except the root which has no parent) being kept smaller than or equal to that of its child m. In other words, $c(f(m)) \leq c(m)$ must hold for each node m except the root r: this is invariant **inv1_1**. As a consequence, the counters are incremented by *waves* moving up as illustrated in figure 3. Here is thus the theorem we want eventually to prove:

thm1_1: $\forall m \cdot m \in N \ \Rightarrow \ c(r) \leq c(m)$

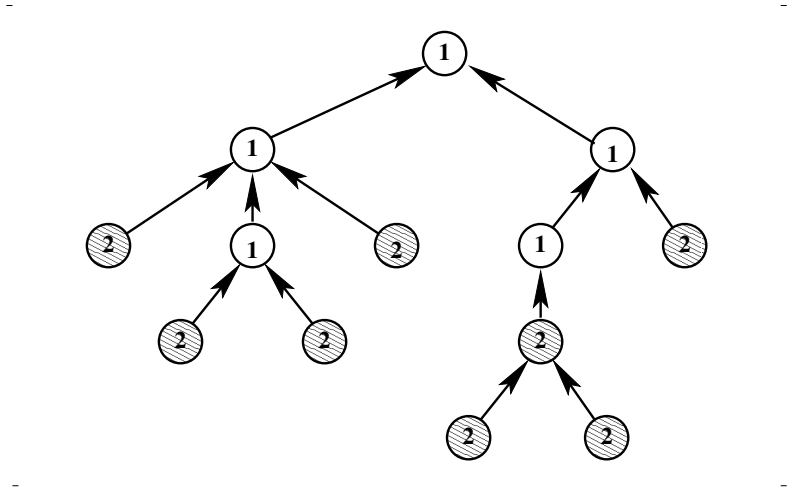


Fig. 3. An Ascending Wave

3.2 The Events

In order to maintain our new invariant **inv1_1**, we have to strengthen the guard of event ascending (a new name for abstract event increment) by ensuring that the counter of node n is distinct from that of each of its children:

init
$c := N \times \{0\}$

ascending
refines
increment
any n **where**
$n \in N$
$c(n) = c(r)$
$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$
then
$c(n) := c(n) + 1$
end

Note that the comparisons of $c(n)$ in node n are now limited to the counters $c(m)$ of the set of children, $f^{-1}[\{n\}]$, of node n, which is allowed according to requirement FUN-3 since node n and m are next to each other. But we also compare $c(n)$ with $c(r)$, which is not acceptable since nodes n and r are not in general next to each other. This problem will be solved in the next refinement.

3.3 The Proofs

In order to ease the proof, we introduce the following two theorems which are easily deductible from the invariants:

thm1_2: $\forall n \cdot n \in N \Rightarrow c(n) \in c(r) \mathinner{..} c(r) + 1$

thm1_3: $\forall n \cdot n \in N \setminus \{r\} \Rightarrow c(n) \in c(f(n)) \mathinner{..} c(f(n)) + 1$

The proof of **thm1_1** is done by tree induction under the assumption of **inv1_1**. Let us remind the tree induction rule IND_TREE, which was introduced in section 7.6 of chapter 9. We adapt it by instantiating V to N, t to r, p to f, and x to m yielding:

$$\frac{\mathsf{H} \vdash \mathsf{P}(r) \qquad \mathsf{H},\ m \in N \setminus \{r\},\ \mathsf{P}(f(m)) \vdash \mathsf{P}(m)}{\mathsf{H},\ m \in N \vdash \mathsf{P}(m)}\ \begin{array}{c}\mathsf{IND\_TREE}\\ (\mathbf{m}\ \underline{\mathsf{nfin}}\ \mathsf{H})\end{array}$$

Here $\mathsf{P}(m)$ is $c(r) \leq c(m)$. Next is the proof of **thm1_1** after applying rules ALL_R and IMP_R:

$$\begin{array}{l} \forall m \cdot m \in N \setminus \{r\} \Rightarrow c(f(m)) \leq c(m) \\ m \in N \\ \vdash \\ c(r) \leq c(m) \end{array} \quad \mathsf{IND\_TREE} \left\{ \begin{array}{ll} \begin{array}{l} \forall m \cdot m \in N \setminus \{r\} \Rightarrow c(f(m)) \leq c(m) \\ \vdash \\ c(r) \leq c(r) \end{array} & \mathsf{ARI} \\ \begin{array}{l} \forall m \cdot m \in N \setminus \{r\} \Rightarrow c(f(m)) \leq c(m) \\ m \in N \setminus \{r\} \\ c(r) \leq c(f(m)) \\ \vdash \\ c(r) \leq c(m) \end{array} & \mathsf{ALL\_L}\ \ldots \end{array} \right.$$

We instantiate now m in the first assumption with m:

$$\ldots \quad \begin{array}{l} m \in N \setminus \{r\} \Rightarrow c(f(m)) \leq c(m) \\ m \in N \setminus \{r\} \\ c(r) \leq c(f(m)) \\ \vdash \\ c(r) \leq c(m) \end{array} \quad \mathsf{IMP\_L} \quad \begin{array}{l} c(f(m)) \leq c(m) \\ m \in N \setminus \{r\} \\ c(r) \leq c(f(m)) \\ \vdash \\ c(r) \leq c(m) \end{array} \quad \mathsf{ARI} \quad \begin{array}{l} c(f(m)) \leq c(m) \\ x \in N \setminus \{r\} \\ c(r) \leq c(f(m)) \\ c(r) \leq c(m) \\ \vdash \\ c(r) \leq c(m) \end{array} \quad \mathsf{HYP}$$

4 Second Refinement

It remains now for us to replace the test $c(r) = c(n)$ in the guard of the previous version of event ascending (copied below) by a more local test.

ascending
  any n **where**
    $n \in N$
    $c(n) = c(r)$
    $\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$
  then
    $c(n) := c(n) + 1$
  end

The problem is to have the nodes made informed that the value of the counter at the root is indeed equal to that of their local counter. This is clearly the case when an incrementing wave has reached the root, where then all nodes have the same value.

When this situation is reached, the idea is to have a second wave going down and gradually informing the nodes that the value of the c counter at the root r is the same as the value of all other c counters. For this, we need a second counter d at each node for holding that second descending wave. This is indicated in on figure 4 where the second counter d is the one indicated on the left of each node.

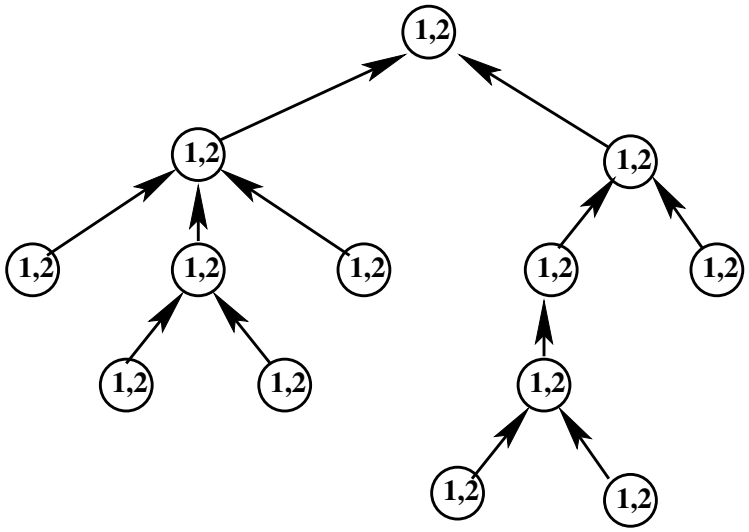


Fig. 4. Enlarging the State with a Second Counter

And now this second wave goes down as illustrated on figure 5. In this refinement, we shall just define the d counters (this is done in invariant **inv2_1** below) together with the basic property that the difference between two d counters does not exceed 1 (invariant **inv2_2**). Note that this property is of the same nature as that expressed in invariant **inv0_1** in the initial model for c counters. Formally:

variable: c, d

inv2_1:	$d \in N \rightarrow \mathbb{N}$
inv2_2:	$\forall x, y \cdot x \in N \wedge y \in N \Rightarrow d(x) \leq d(y) + 1$

We now have to extend event init in a straightforward fashion and also add a new event, descending, corresponding to the incrementation of the d counters when appropriate. Again, note the analogy with what we did in the initial model with the c counters.

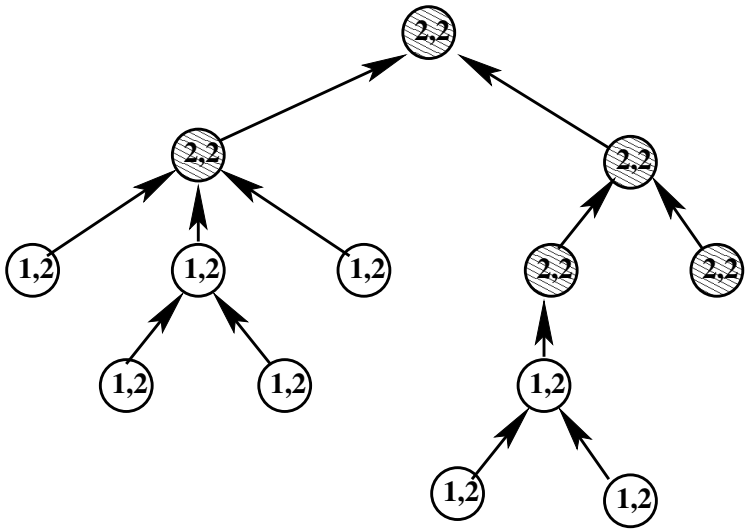


Fig. 5. The descending wave

init
$c := N \times \{0\}$
$d := N \times \{0\}$

descending
any n **where**
$n \in N$
$\forall m \cdot m \in N \Rightarrow d(n) \leq d(m)$
then
$d(n) := d(n) + 1$
end

Event ascending has not been touched in this refinement, hence we have not copied it above. We leave it to the reader to prove that this superposition refinement is correct. Don't forget to prove that the new event does not diverge and also that the system does not deadlock.

5 Third Refinement

In this third refinement, we are going to perform two tasks: (1) we construct the descending wave of the d counters incrementation very much in the same way as we did it for the ascending wave of the c counters incrementation in section 3, and (2) we establish the relationship between the c and the d counters. Note that we do not extend the state. We have thus here a special case of superposition: the concrete state is exactly the same as the abstract one.

5.1 Refining event ascending

The first invariant we introduce, **inv3_1**, is very much of the same nature as invariant **inv1_1** introduced in the first refinement for the c counters (section 3.1), which we repeat below:

inv1_1: $\forall m \cdot m \in N \setminus \{r\} \Rightarrow c(f(m)) \leq c(m)$

The d counter at a node m is not greater than that of its parent node $f(m)$. It is what makes the wave descending:

inv3_1: $\forall m \cdot m \in N \setminus \{r\} \Rightarrow d(m) \leq d(f(m))$

With this invariant and the induction rule IND_TREE mentioned in section 5.3, we can easily prove the following theorem, which is of the same nature as **thm1_1** of section 5.3. It says that the d counters are are not greater than the value $d(r)$:

thm3_1: $\forall n \cdot n \in N \Rightarrow d(n) \leq d(r)$

It remains for us to establish the connection between $c(r)$ and $d(r)$. In fact the latter is never greater than the former. It is expressed in the following invariant:

inv3_2: $d(r) \leq c(r)$

Thanks to these invariants, we can refine event ascending as follows:

(abstract-)ascending
any n **where**
$n \in N$
$c(n) = c(r)$
$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$
then
$c(n) := c(n) + 1$
end

(concrete-)ascending
any n **where**
$n \in N$
$c(n) = d(n)$
$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$
then
$c(n) := c(n) + 1$
end

As can be seen, the only difference between the abstract and concrete versions of these events is the replacement of the abstract guard $c(n) = c(r)$ by the concrete one $c(n) = d(n)$. As a consequence, we have just to prove that the latter implies the former (guard strengthening), which is trivial:

concrete guard	$c(n) = d(n)$
according to **thm3_1**	$d(n) \leq d(r)$
invariant **inv3_2**	$d(r) \leq c(r)$
according to **thm1_1**	$c(r) \leq c(n)$
$\vdash$	$\vdash$
abstract guard	$c(n) = c(r)$

We have now reach our goal concerning event ascending: it is indeed accessing its neighbors' counters only as required by requirement FUN-3.

5.2 Refining event descending

We now turn our attention to event descending. In fact, the abstract descending event has to be split into two events: one dealing with any node n except the root r and another one dealing with the root node r. Here is the proposal for the first case together with the abstraction:

(abstract-)descending
any n **where**
$n \in N$
$\forall m \cdot m \in N \Rightarrow d(n) \leq d(m)$
then
$d(n) := d(n) + 1$
end

(concrete-)descending_1
any n **where**
$n \in N \setminus \{r\}$
$d(n) \neq d(f(n))$
then
$d(n) := d(n) + 1$
end

We notice that in the concrete version, the node n follows requirement FUN-3: it only accesses the value of counters d at n and at $f(n)$, the parent node of n. The only difference between the two versions concerns again the guards. The abstract guard $\forall m \cdot m \in N \Rightarrow d(n) \leq d(m)$ is replaced by the concrete one $d(n) \neq d(f(n))$. We have then just again to prove that the concrete guard implies the abstract one. It amounts to proving the following:

$$\begin{array}{l} n \in N \setminus \{r\} \\ d(n) \neq d(f(n)) \\ m \in N \\ \vdash \\ d(n) \leq d(m) \end{array}$$

For proving this statement, we can first prove the following simple theorems

thm3_2: $\forall n \cdot n \in N \setminus \{r\} \Rightarrow d(f(n)) \in d(n) \mathinner{..} d(n) + 1$

thm3_3: $\forall n \cdot n \in N \Rightarrow d(r) \in d(n) \mathinner{..} d(n) + 1$

From $d(n) \neq d(f(n))$ and **thm3_2**, we deduce $d(f(n)) = d(n) + 1$. By instantiating n in **thm3_3** successively with $f(n)$ and m, we obtain $d(r) \in d(f(n)) \mathinner{..} d(f(n))+1$ and $d(r) \in d(m) \mathinner{..} d(m)+1$. We are thus left to prove the following which is trivial (the assumptions yield: $d(n)+1 \leq d(r) \leq d(m)+1$):

$$\begin{array}{l} d(r) \in d(n) + 1 \mathinner{..} d(n) + 2 \\ d(r) \in d(m) \mathinner{..} d(m) + 1 \\ \vdash \\ d(n) \leq d(m) \end{array}$$

We now have to consider the second case for the descending event. Here is the proposal:

(abstract-)descending
any n **where**
$n \in N$
$\forall m \cdot m \in N \Rightarrow d(n) \leq d(m)$
then
$d(n) := d(n) + 1$
end

(concrete-)descending_2
when
$d(r) \neq c(r)$
then
$d(r) := d(r) + 1$
end

Again, the concrete version follows requirement FUN-3 as node r only accesses the c and d counters of this node. We have a case here where the abstract event is introduced by an **any** construct whereas the concrete one is introduced by a **when** construct. We have to provide a witness, which is clearly r for n in the abstraction. As a consequence, the two actions are now the same and the only statement to prove is again the strengthening of the guard. After some simplification, it amounts to proving the following:

$$\begin{array}{l} d(r) \neq c(r) \\ m \in N \\ \Rightarrow \\ d(r) \leq d(m) \end{array}$$

Now suppose that we have the following theorem:

thm3_4: $\forall n \cdot n \in N \ \Rightarrow \ c(r) \ \in \ d(n) \mathinner{..} d(n) + 1$

We can instantiate this theorem first with r yielding $c(r) \in d(r) \mathinner{..} d(r) + 1$. But we have $d(r) \neq c(r)$. As a consequence, we have $c(r) = d(r) + 1$. We now instantiate **thm3_4** with m yielding $c(r) \in d(m) \mathinner{..} d(m) + 1$. We are thus left to prove the following, which is trivial:

$$\begin{array}{l} d(r) + 1 \ \in \ d(m) \mathinner{..} d(m) + 1 \\ \vdash \\ d(r) \leq d(m) \end{array}$$

5.3 Proving Theorem thm3_4

It now remains for us to prove **thm3_4**. For this we have to introduce a new invariant, which is the following:

inv3_3: $\forall n \cdot n \in N \ \Rightarrow \ c(n) \ \in \ d(n) \mathinner{..} d(n) + 1$

Note that invariant **inv3_2**, stating that $d(r)$ is not greater than $c(r)$, can be transformed into a mere theorem as it is clearly implied by invariant **inv3_3**. Theorem **thm3_4** can now be proved from invariant **inv3_3** and theorems **thm3_1** and **thm1_2**. Here is what we have to prove:

Theorem **thm3_1**	$\forall n \cdot n \in N \ \Rightarrow \ d(n) \leq d(r)$
Invariant **inv3_3**	$\forall n \cdot n \in N \ \Rightarrow \ c(n) \ \in \ d(n) \mathinner{..} d(n) + 1$
Theorem **thm1_2**	$\forall n \cdot n \in N \ \Rightarrow \ c(n) \ \in \ c(r) \mathinner{..} c(r) + 1$
$\vdash$	$\vdash$
Theorem **thm3_4**	$\forall n \cdot n \in N \ \Rightarrow \ c(r) \ \in \ d(n) \mathinner{..} d(n) + 1$

By properly instantiating and simplifying the previous statement, we obtain the following which is trivially true (the two first antecedents yields $d(n) \ \leq \ d(r) \ \leq \ c(r)$, whereas the last two yields $c(r) \ \leq \ c(n) \ \leq \ d(n) + 1$):

Theorem **thm3_1** instantiated with n	$d(n) \leq d(r)$
Invariant **inv3_3** instantiated with r	$c(r) \in d(r) \mathinner{..} d(r)+1$
Theorem **thm1_2** instantiated with n	$c(n) \in c(r) \mathinner{..} c(r)+1$
Invariant **inv3_3** instantiated with n	$c(n) \in d(n) \mathinner{..} d(n)+1$
$\vdash$	$\vdash$
Theorem **thm3_4**	$c(r) \in d(n) \mathinner{..} d(n)+1$

5.4 Proving Preservation of Invariant inv3_3

It remains now for us to prove that the new invariant **inv3_3** is indeed preserved by the proposed events. Here is what we have to prove for event ascending to preserve invariant **inv3_3**:

Invariant **inv3_3**	$\forall m \cdot m \in N \Rightarrow c(m) \in d(m) \mathinner{..} d(m)+1$
Guard of ascending	$n \in N$
	$c(n) = d(n)$
	$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$
$\vdash$	$\vdash$
Mod. inv. **inv3_3**	$\forall n \cdot m \in N \Rightarrow (c \lhd \{n \mapsto c(n)+1\})(m) \in d(m) \mathinner{..} d(m)+1$

This can be rearranged like this:

$$
\begin{array}{l}
\forall m \cdot m \in N \Rightarrow c(m) \in d(m) \mathinner{..} d(m)+1 \\
n \in N \\
c(n) = d(n) \\
m \in N \\
\vdash \\
c \lhd \{n \mapsto c(n)+1\})(m) \in d(m) \mathinner{..} d(m)+1
\end{array}
$$

As usual in the presence of the overriding operator $\lhd$, we have to do a proof by case. First case: $m = n$, and second case: $m \neq n$. The first case leads to the following which is trivial:

$$
\begin{array}{l}
c(n) = d(n) \\
n \in N \\
\vdash \\
c(n)+1 \in d(n) \mathinner{..} d(n)+1
\end{array}
$$

The second case is solved by instantiating m in the universal quantification with m, leading to the following which is trivial:

$$
\begin{array}{l}
c(m) \in d(m) \mathinner{..} d(m)+1 \\
m \in N \\
\vdash \\
c(m) \in d(m) \mathinner{..} d(m)+1
\end{array}
$$

We now prove that invariant **inv3_3** is preserved by event descending_1. It amounts to proving the following:

Invariant **inv3_3**	$\forall m \cdot m \in N \Rightarrow c(m) \in d(m) \mathinner{..} d(m)+1$
Guard of descending_1	$n \in N \setminus \{r\}$
	$d(n) \neq d(f(n))$
$\vdash$	$\vdash$
Mod. inv. **inv3_3**	$\forall m \cdot m \in N \Rightarrow c(m) \in (d \lhd \{n \mapsto d(n)+1\})(m) \mathinner{..}$ $(d \lhd \{n \mapsto d(n)+1\})(m)+1$

This can be rearranged like this:

$$\begin{array}{l}
\forall m \cdot m \in N \ \Rightarrow \ c(m) \ \in \ d(m) \mathinner{..} d(m)+1 \\
n \in N \setminus \{r\} \\
d(n) \neq d(f(n)) \\
m \in N \\
\vdash \\
c(m) \ \in \ (d \lhd \{n \mapsto d(n)+1\})(m) \mathinner{..} (d \lhd \{n \mapsto d(n)+1\})(m)+1
\end{array}$$

We do a proof by cases. First case $m = n$. Second case $m \neq n$. Here is what the first case yields:

$$\begin{array}{l}
n \in N \setminus \{r\} \\
d(n) \neq d(f(n)) \\
\Rightarrow \\
c(n) \ \in \ d(n)+1 \mathinner{..} d(n)+2
\end{array}$$

By instantiating **thm3_2** with n, we obtain $d(f(n)) \ \in \ d(n) \mathinner{..} d(n)+1$, which together with $d(n) \neq d(f(n))$ yields $d(f(n)) = d(n)+1$. Instantiating then **thm3_1** with $f(n)$ yields $d(f(n)) \leq d(r)$, that is $d(n)+1 \leq d(r)$. We are thus left to prove the following where we have added some theorems and invariant instantiations. This is trivially true (the three first antecedents yield $d(n)+1 \ \leq \ d(r) \ \leq \ c(r) \ \leq \ c(n)$, whereas the last antecedent yields $c(n) \ \leq d(n+1)$, therefore we have $c(n) = d(n)+1$):

	$d(n)+1 \leq d(r)$
inv3_3 instantiated with r	$c(r) \ \in \ d(r) \mathinner{..} d(r)+1$
thm1_2 instantiated with n	$c(n) \ \in \ c(r) \mathinner{..} c(r)+1$
inv3_3 instantiated with n	$c(n) \ \in \ d(n) \mathinner{..} d(n)+1$
	$\vdash$
	$c(n) \ \in \ d(n)+1 \mathinner{..} d(n)+2$

The second case ($m \neq n$) yields the following which is trivially true:

$$\begin{array}{l}
\forall m \cdot m \in N \ \Rightarrow \ c(m) \ \in \ d(m) \mathinner{..} d(m)+1 \\
n \in N \setminus \{r\} \\
d(n) \neq d(f(n)) \\
m \in N \\
\vdash \\
c(m) \ \in \ d(m) \mathinner{..} d(m)+1
\end{array}$$

It remains finally for us to prove that invariant **inv3_3** is preserved by event descending_2. It amounts to proving:

Invariant **inv3_3**	$\forall m \cdot m \in N \ \Rightarrow \ c(m) \ \in \ d(m) \mathinner{..} d(m)+1$
Guard of descending_2	$d(r) \neq c(r)$
$\vdash$	$\vdash$
Mod. inv. **inv3_3**	$\forall m \cdot m \in N \ \Rightarrow \ c(m) \ \in \ (d \lhd \{r \mapsto d(r)+1\})(m) \mathinner{..} (d \lhd \{r \mapsto d(r)+1\})(m)+1$

This can be rearranged as follows:

$$
\begin{array}{l}
\forall m \cdot m \in N \ \Rightarrow \ c(m) \ \in \ d(m) \mathinner{..} d(m)+1 \\
d(r) \neq c(r) \\
m \in N \\
\vdash \\
c(m) \ \in \ (d \lhd \{r \mapsto d(r)+1\})(m) \mathinner{..} (d \lhd \{r \mapsto d(r)+1\})(m)+1
\end{array}
$$

The proof proceeds again by cases. The first case ($m = r$), leads to the following which is trivially true:

Theorem **thm3_1** instantiated with r

$$
\begin{array}{l}
d(r) \neq c(r) \\
c(r) \ \in \ d(r) \mathinner{..} d(r)+1 \\
\vdash \\
c(r) \ \in \ d(r)+1 \mathinner{..} d(r)+1+1
\end{array}
$$

The second case ($m \neq r$), leads to the following which is also trivially true:

Invariant **inv3_3** instantiated with m

$$
\begin{array}{l}
c(m) \ \in \ d(m) \mathinner{..} d(m)+1 \\
\vdash \\
c(m) \ \in \ d(m) \mathinner{..} (d(m)+1
\end{array}
$$

6 Fourth Refinements

The idea of the next refinement comes from a careful observation of the three events we have obtained in the previous section. Let us copy them here again (note that we have changed the guard of event ascending to an equivalent one):

ascending
any n **where**
$n \in N$
$c(n) = d(n)$
$\forall m \cdot \left(\begin{array}{l} m \in N \\ n = f(m) \\ \Rightarrow \\ c(f(m)) \neq c(m) \end{array} \right)$
then
$c(n) := c(n)+1$
end

descending_1
any n **where**
$n \in N \setminus \{r\}$
$d(n) \neq d(f(n))$
then
$d(n) := d(n)+1$
end

descending_2
when
$d(r) \neq c(r)$
then
$d(r) := d(r)+1$
end

In event ascending, we observe a comparison of $c(n)$ and $d(n)$, and also a comparison of $c(m)$ and $c(f(m))$. In event descending_1 we observe that the guard contains a comparison of $d(n)$ and $d(f(n))$ and in event descending_2, we observe that the guard contains a comparison of $d(r)$ and $c(r)$. Moreover, the counters c or d are incremented in all events.

We have the impression to encounter a situation which is very similar to the one already encountered in the file transfer protocol example in chapters 4 and 6. Should this impression be confirmed then we could replace the values of the counters c and d simply by their parities. It is certainly an interesting transformation since then we have no more risk that such counters become too big. But in order to be able to do this refinement, we must be sure that the *difference between the compared values is not greater than one*. In the following table, we state that it is indeed the case:

Comparison	Justifying Theorem
$c(n)$ and $d(n)$	**inv3_3:** $\forall n \cdot n \in N \Rightarrow c(n) \in d(n) \mathinner{..} d(n)+1$
$c(m)$ and $c(f(m))$	**thm1_3:** $\forall n \cdot n \in N \setminus \{r\} \Rightarrow c(n) \in c(f(n)) \mathinner{..} c(f(n))+1$
$d(n)$ and $d(f(n))$	**thm3_2:** $\forall n \cdot n \in N \setminus \{r\} \Rightarrow d(f(n)) \in d(n) \mathinner{..} d(n)+1$
$d(r)$ and $c(r)$	**thm3_4:** $\forall n \cdot n \in N \Rightarrow c(r) \in d(n) \mathinner{..} d(n)+1$

The fourth refinement is now mere routine. We copy here the properties and basic theorem concerning parities:

constants: $r, f, parity$

axm4_1: $parity \in \mathbb{N} \to \{0,1\}$

axm4_2: $parity(0) = 0$

axm4_3: $\forall x \cdot x \in \mathbb{N} \Rightarrow parity(x+1) = 1 - parity(x)$

thm4_1: $\forall x, y \cdot x \in \mathbb{N} \land y \in \mathbb{N} \land x \in y \mathinner{..} y+1 \land parity(x) = parity(y) \Rightarrow x = y$

And we define the parities $p(n)$ and $q(n)$ of $c(n)$ and $d(n)$:

variables: p, q

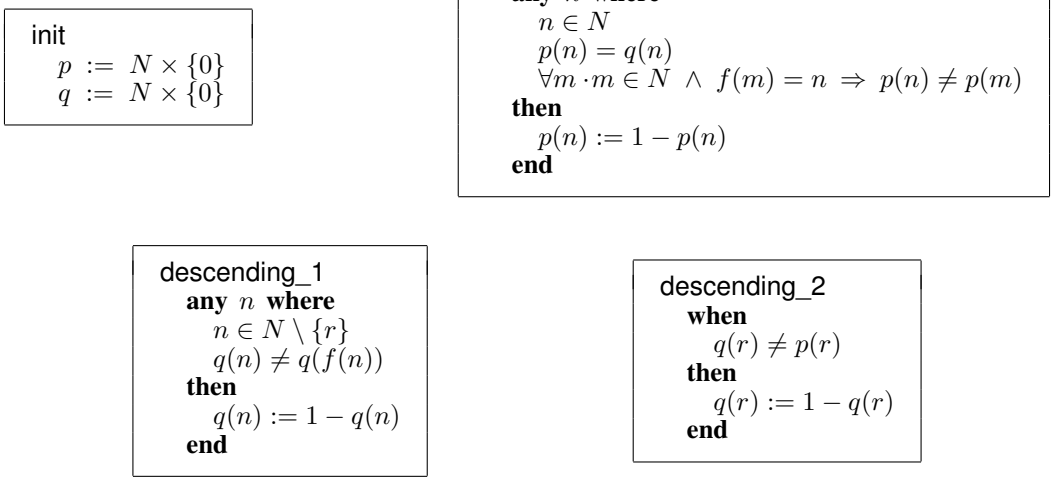
inv4_1: $p \in N \to \{0,1\}$

inv4_2: $q \in N \to \{0,1\}$

inv4_3: $\forall n \cdot (n \in N \Rightarrow p(n) = parity(c(n)))$

inv4_4: $\forall n \cdot (n \in N \Rightarrow q(n) = parity(d(n)))$

The final events are as follows:



References

1. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers 1996.
2. W.H.J. Feijen and A.J.M. van Gasteren. *On a Method of Multi-programming* Springer 1999.