

Synchronizing Processes on a Tree Network¹

Manuel Carro

manuel.carro@upm.es

Universidad Politécnica de Madrid &
IMDEA Software Institute

¹Example and several slides from J. R. Abrial book *Modeling in Event-B: system and software engineering*.

Goals	s. 3
Requirements	s. 5
Initial model	s. 10
First refinement	s. 22
Second refinement	s. 50
Third refinement	s. 54
Fourth refinement	s. 79

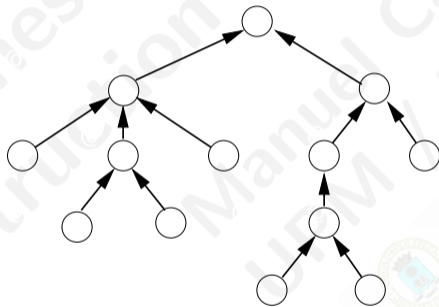
- We will formalize the solution to a problem in distributed computing.
 - Studied in: *W.H.J. Feijen and A.J.M. van Gasteren. On a Method of Multi-programming. Springer Verlag, 1999.*
- Using and **updating** functions.
- Formalize and prove properties on an interesting structure: a tree.
- Proofs more complex than those seen so far.

As usual:

- Define the informal requirements
- Define the refinement strategy
- Construct the various more and more concrete models

- Not a **transformational** system.
 - Not supposed to finish.
 - No final result.
- Not **reactive**.
 - No *external* world that reacts to system changes.
- **Distributed**.
 - Different *nodes* act **autonomously**.
 - With **limited** information access.
 - However, communication assumed to be reliable.
- Internal **concurrency**.
 - Every node has concurrent processes.
- Small model: just three events in the last refinement.
- However, proofs and reasoning are comparatively complex.

ENV 1 We have a fixed set of processes forming a tree

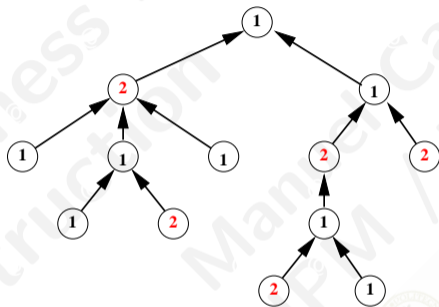


- Note: they do not need to form a tree from the beginning.
- A set of communicating processes can coordinate to form a tree.

- All processes are supposed to execute forever the same code.
- But processes must remain (somewhat) **synchronized**.
- For this, each process has (initially) one counter.

ENV 2	Each process has a counter, which is a natural number
--------------	---

- A process counter represents its “phase” (related to the work for which they have to synchronize).
- Difference between any two counters \leq one.
 - Each process is thus at most one phase ahead of the others



FUN 3 The difference between any two counters is at most one

- Reading the counters

FUN 4	Each process can read the counters of its neighbors only
--------------	--

(*Neighbors* to be understood as *connected by a link*)

- Modifying the counters

FUN 5	The counter of a process can be modified by this process only
--------------	---

- Construct abstract initial model dealing with **FUN 3** and **FUN 5**
- Improve design to (partially) take care of **FUN 4**
- Improve design to better take care of **FUN 4**
- (Simplify final design to obtain efficient implementation).

FUN 3	The difference between any two counters is at most one
--------------	--

FUN 4	Processes read counters of immediate neighbors only
--------------	---

FUN 5	A process can modify only its counter(s)
--------------	--

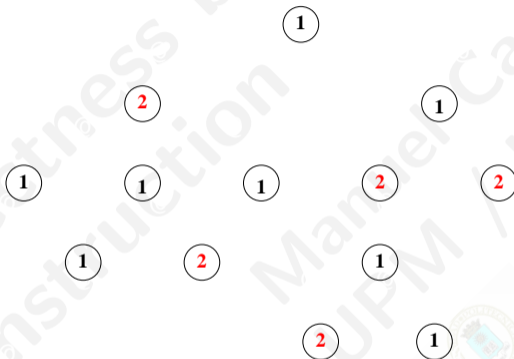
1. Initial model: all nodes access the state of all nodes.
2. First refinement: restrict access to a single node.
3. Second refinement: local check, upwards wave.
4. Third refinement: construct downwards wave.
5. Fourth refinement: remove upwards and downwards counters.

- Simplify situation: forget about tree
- We just define the counters and express the main property: **FUN 3**

FUN 3 The difference between any two counters is at most one

- The initial model is always far more abstract than the final system
- Other requirements are probably not fulfilled

Abstract situation



FUN 3 | The difference between any two counters is at most 1

Suggest constants, axioms, variables,
invariants for an initial model!

Correctness by
Construction
Manuel Carro
UPM iMDEA
software



carrier set: P

axm0_1: $\text{finite}(P)$

variable: c

inv0_1: $c \in P \rightarrow \mathbb{N}$

inv0_2: $\forall x, y \cdot \left(\begin{array}{l} x \in P \\ y \in P \\ \Rightarrow \\ c(x) \leq c(y) + 1 \end{array} \right)$

- ✓ Create project *synch_tree*
- ✓ Create context *c0* with set, axiom
- ✓ Create machine *m0* with variable, invariants.

Is that right?

- `inv0_2` may be surprising:

$$\mathcal{I}_0 : \forall x, y \cdot x \in P \wedge y \in P \Rightarrow c(x) \leq c(y) + 1$$

- Is it the same as $\mathcal{I}_1 : \forall i, j \cdot |c(i) - c(j)| \leq 1$?

Is that right?

- `inv0_2` may be surprising:

$$\mathcal{I}_0 : \forall x, y \cdot x \in P \wedge y \in P \Rightarrow c(x) \leq c(y) + 1$$

- Is it the same as $\mathcal{I}_1 : \forall i, j \cdot |c(i) - c(j)| \leq 1$?

Proof by double implication.

Let us choose two arbitrary nodes with counters a and b .

- If the invariant holds, then $a \leq b + 1$ and $b \leq a + 1$. From there, $a - b \leq 1$ and $b - a \leq 1$, therefore $|a - b| \leq 1$, and $\mathcal{I}_0 \Rightarrow \mathcal{I}_1$.
- If $|a - b| \leq 1$, then both $a - b \leq 1$ and $b - a \leq 1$. Then, `inv0_2` is implied by the intended invariant, and $\mathcal{I}_1 \Rightarrow \mathcal{I}_0$.

```
init  
  c := P × {0}
```

```
ascending  
  any n where  
    n ∈ P  
    ∀m · m ∈ P ⇒ c(n) ≤ c(m)  
  then  
    c(n) := c(n) + 1  
  end
```

- Note: **any** n is logically $\forall n \cdot n \in P \wedge \dots$
 - \forall can appear in guards.
 - **any** introduces \forall whose scope is the whole event.
- Intuition: *Any increment that respects difference among nodes can be done.*
 - Does not mean **all** increments are executed: non-determinism!
 - Not final state (there is none): action that (hopefully) respects invariant.

✓ *Add initialization, event*

Note: \times is entered with **, **any** with pull-down menu, "Add event parameter".

Proof of invariant preservation

$$c \in P \rightarrow \mathbb{N}$$

inv0_1

$$\forall x, y \cdot \left(\begin{array}{l} x \in P \\ y \in P \\ \Rightarrow \\ c(x) \leq c(y) + 1 \end{array} \right)$$

inv0_2

$$n \in P$$

$$\forall m \cdot (m \in P \Rightarrow c(n) \leq c(m))$$

Guards of event
ascending

⊢

$$\forall x, y \cdot \left(\begin{array}{l} x \in P \\ y \in P \\ \Rightarrow \\ (c \triangleleft \{n \mapsto c(n) + 1\})(x) \leq (c \triangleleft \{n \mapsto c(n) + 1\})(y) + 1 \end{array} \right)$$

↑

Modified invariant **inv0_2**

In Rodin: automatic; if not, repeatedly apply lassoing, p0 or m0.

CONTEXT c0

SETS

P

AXIOMS

axm1: $finite(P)$

END

MACHINE m0

SEES c0

VARIABLES

c

INVARIANTS

inv1: $c \in P \rightarrow \mathbb{N}$

inv2: $\forall x, y \cdot x \in P \wedge y \in P \Rightarrow c(x) \leq 1 + c(y)$

EVENTS

Initialisation

begin

act1: $c := P \times \{0\}$

end

Event ascending ⟨ordinary⟩ $\hat{=}$

any

n

where

grd12: $n \in P$

grd11: $\forall m \cdot m \in P \Rightarrow c(n) \leq c(m)$

then

act11: $c(n) := c(n) + 1$

end

ascending

any n **where**

$n \in P$

$\forall m \cdot m \in P \Rightarrow c(n) \leq c(m)$

then

$c(n) := c(n) + 1$

end

What requirement is this event breaking?

ascending

any n **where**

$n \in P$

$\forall m \cdot m \in P \Rightarrow c(n) \leq c(m)$

then

$c(n) := c(n) + 1$

end

What requirement is this event breaking?

FUN 2

Each node can read the counters of its immediate neighbors only

1. Initial model: all nodes access the state of all nodes.
2. First refinement: restrict access to a single node.
3. Second refinement: local check, upwards wave.
4. Third refinement: construct downwards wave.
5. Fourth refinement: remove upwards and downwards counters.

First refinement: (partially) solving the problem

- Introduce a designated process r .
- Assume that counter of r always minimal

$$\forall m \cdot m \in P \Rightarrow c(r) \leq c(m)$$

- Rationale:

- We only synchronize with r — not compliant, but communication restricted.
- Helps ensure that difference between any two nodes \leq one.
 - If $\text{inv0_1}: \forall x, y \cdot x \in P \wedge y \in P \Rightarrow c(x) \leq c(y) + 1$, then $|c(r) - c(m)| \leq 1$ for any m .
 - If $c(r) \leq c(m)$, then $c(m) = c(r)$ or $c(m) = c(r) + 1$ for any m .
 - Then $|c(m) - c(n)| \leq 1$, for any m, n (will be proved).

- Treat this property as a new (temporary) invariant.

- ✓ Extend c_0 into c_1 (left pane, right click, "Extend"), add constant r , axiom $r \in P$
- ✓ Refine m_0 into m_1 (left pane, right click, "Refine"), add new invariant
- ✓ m_0 should "see" c_1

First refinement: proposal for the event refinement

We simplify the guard

```
(abstract-)ascending  
any  $n$  where  
   $n \in P$   
   $\forall m \cdot m \in P \Rightarrow c(n) \leq c(m)$   
then  
   $c(n) := c(n) + 1$   
end
```

```
(concrete-)ascending  
any  $n$  where  
   $n \in P$   
   $c(n) = c(r)$   
then  
   $c(n) := c(n) + 1$   
end
```

- Note: if $c(r)$ minimal, $c(n) < c(r)$ impossible; therefore $c(n) = c(r)$
✓ *Change "extended" to "not extended", change guard*
- We have then to prove guard strengthening.

$$c \in P \rightarrow \mathbb{N}$$

$$\forall x, y \cdot \left(\begin{array}{l} x \in P \\ y \in P \\ \Rightarrow \\ c(x) \leq c(y) + 1 \end{array} \right)$$

$$\forall m \cdot (m \in P \Rightarrow \boxed{c(r)} \leq c(m))$$

$$n \in P$$

$$\boxed{c(n) = c(r)}$$

⊢

$$n \in P$$

$$\forall m \cdot (m \in P \Rightarrow \boxed{c(n)} \leq c(m))$$

inv0_1

inv0_2

new invariant

Guards of **concrete**
event ascending

Guards of **abstract**
event ascending

✓ If necessary, switch to proving perspective, discharge proof with lasso + p0

Model so far

inv1 not discharged.

```
CONTEXT c1
EXTENDS c0
CONSTANTS
  r
AXIOMS
  axm1:  $r \in P$ 
END
```

```
MACHINE m1
REFINES m0
SEES c1
VARIABLES
  c
INVARIANTS
  inv1:  $\forall m \cdot m \in P \Rightarrow c(r) \leq c(m)$ 
EVENTS
  Initialisation (extended)
  begin
    act1:  $c := P \times \{0\}$ 
  end
  Event ascending (ordinary)  $\hat{=}$ 
  refines ascending
  any
    n
  where
    grd1:  $n \in P$ 
    grd2:  $c(r) = c(n)$ 
  then
    act1:  $c(n) := c(n) + 1$ 
  end
END
```

```
ascending
  any  $n$  where
     $n \in P$ 
     $c(n) = c(r)$ 
  then
     $c(n) := c(n) + 1$ 
  end
```

$$\forall m \cdot m \in P \Rightarrow c(r) \leq c(m)$$

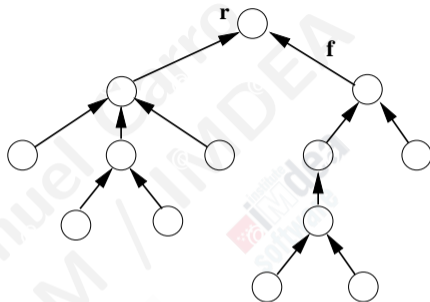
1. Prove that new “invariant” is preserved by the event
2. The guard of the event still does not fulfill requirement **FUN 4**.

FUN 4 Each node can read the counters of its immediate neighbors only

- Problem 1 solved in this refinement
- Problem 2 solved later

First refinement: defining the tree

- Tree: root r and “pointer” f from each node in $P \setminus \{r\}$ to every node's parent.
- Plus some additional properties and inference rules.
- Reminder: sets, relations, functions, specific data structures and their inference rules.
- Note: constructing a tree (\equiv root / leader + links) is a classical problem in distributed systems.
- Can also be tackled using Event B.



Invariant: we have a condition involving nodes in pairs and we need a condition that relates a single node r with all the others.

How can we model the relation node / parent node?

✓ Add to $c1$ (note f is \rightarrow , written $-->>$)

- Constant f ("father").
- Constant L ("leaves").
- Axioms:

$$L \subseteq P$$

$$f \in P \setminus \{r\} \rightarrow P \setminus L$$

$$\forall S \cdot S \subseteq f^{-1}[S] \Rightarrow S = \emptyset$$

- f^{-1} is written $f\sim$.
- \rightarrow : f defined for all $P \setminus \{r\}$ and *arrives* to every element in $P \setminus L$.

- Minimality of counter at the root

$$\forall m \cdot m \in P \Rightarrow c(r) \leq c(m)$$

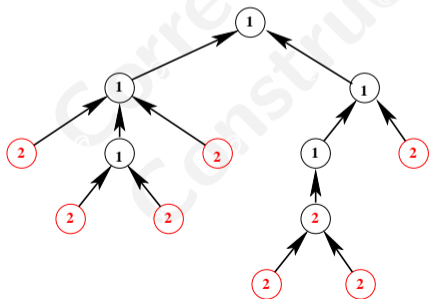
relates $c(r)$ with $c(m)$ for every m .

- Events change local values and consult neighbouring values.
- We can (easily) prove properties relating neighbouring nodes.
- How can we relate **local** properties with **global** properties?

- We define a weaker, local invariant first.
- The counter at every node is not greater than its descendants:

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

✓ Add it to m_1



Rationale (advancing the algorithm)

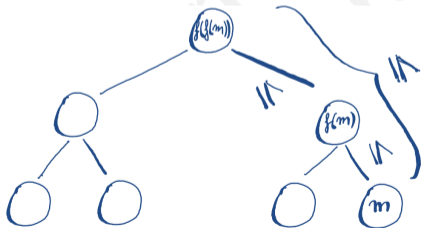
- Assume we can update the tree keeping a maximum difference between neighbors.
- Will be helpful to prove $c(r) \leq c(m)$.

Minimal counter at the root: induction

We need to *extend* the local property

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

to the whole tree.



- Start with leaves $l \in L$.
- Prove that for any l , $c(f(l)) \leq c(l)$, then $c(f(f(l))) \leq c(f(l)) \leq c(l), \dots$
- Work upwards towards root r .

OR

- Start with r .
- Prove that for all m s.t. $r = f(m)$, $c(r) \leq c(m)$.
 m is a child of r
- Then, for all m' s.t. $m = f(m')$, $c(m) \leq c(m')$...
- And so on towards the leaves.

- Induction: difficult for theorem provers to do on their own.
 - Needs to identify base case, property to use for induction.
- Then, proving property given base case & inductive step within theorem provers' capabilities.
- In Rodin: needs **adding** induction scheme:

✓ *Add to c1:*

$\forall S. S \subseteq P \wedge r \in S \wedge (\forall n. n \in P \setminus \{r\} \wedge f(n) \in S \Rightarrow n \in S) \Rightarrow P \subseteq S$

✓ *Tip: Ctrl-Enter breaks text in input box in separate lines.*

- **Instantiating** it with the property to prove, expressed as a set:
 $\{x \mid x \in P \wedge c(r) \leq c(x)\}$ (next slide)

✓ *In m1: ensure you have inv1_1: $\forall m. m \in P \setminus \{r\} \Rightarrow c(f(m)) \leq c(m)$*

✓ *Ensure thm1_1: $\forall m. m \in P \Rightarrow c(r) \leq c(m)$ below invariant, marked as theorem*

Induction in Rodin: instantiation

- Double click in the unproved theorem (left pane).
- Switch to prover view, lasso.
- Locate induction axiom.
- Enter $\{x \mid x \in P \wedge c(r) \leq c(x)\}$.
- Return and **p0**.
- The theorem should be proved now.

The screenshot shows the Rodin prover interface with two panes. The left pane displays the initial goal and hypotheses:

- Goal: $\forall m. m \in P \setminus \{r\} \Rightarrow c(f(m)) \leq c(m)$
- Hypotheses:
 - $\text{finite}(P)$
 - $f \in P \setminus \{r\} \rightarrow P \setminus L$
 - $\forall S. S \subseteq f^{-1}[S] \Rightarrow S = \emptyset$
 - $c \in P \rightarrow N$
 - $\forall S. \begin{aligned} & r \in S \wedge \\ & (\forall n. n \in P \setminus \{r\} \wedge f(n) \in S \Rightarrow n \in S) \\ & \Rightarrow P \subseteq S \end{aligned}$
 - $\forall x, y. c(x) \leq c(y) + 1$

The right pane shows the goal after instantiation, with a new hypothesis highlighted in yellow:

- Goal: $\forall m. m \in P \setminus \{r\} \Rightarrow c(f(m)) \leq c(m)$
- Hypotheses:
 - $\text{finite}(P)$
 - $f \in P \setminus \{r\} \rightarrow P \setminus L$
 - $\forall S. S \subseteq f^{-1}[S] \Rightarrow S = \emptyset$
 - $c \in P \rightarrow N$
 - $\forall S. \{x \mid x \in P \wedge c(r) \leq c(x)\}$ (highlighted)
 - $\forall S. \begin{aligned} & r \in S \wedge \\ & (\forall n. n \in P \setminus \{r\} \wedge f(n) \in S \Rightarrow n \in S) \\ & \Rightarrow P \subseteq S \end{aligned}$
 - $\forall x, y. c(x) \leq c(y) + 1$

The "Selected Hypotheses" and "Goal" sections at the bottom of both panes show the current state of the prover's internal state.

Invariant inv1_1 not yet proved. Requires order between parent and children $c(f(m)) \leq c(m)$ that **ascending** cannot guarantee: guard $c(r) = c(n)$ allows updates in arbitrary order. Will enforce through more local comparison.

More local comparison

- Nodes with difference \leq one from r .
- When can we update looking locally?

ascending

any n where

$n \in P$

$c(r) = c(n)$

$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$

then

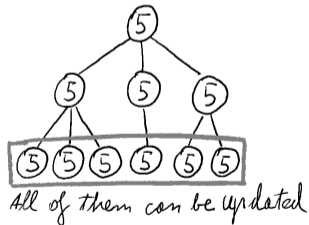
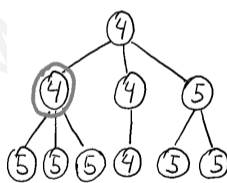
$c(n) := c(n) + 1$

end

Ensure $inv1_1$ is preserved: double click, prover view, lasso, $p0$ should do it.

$$c(n) = c(r) \wedge \forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(m) > c(n)$$

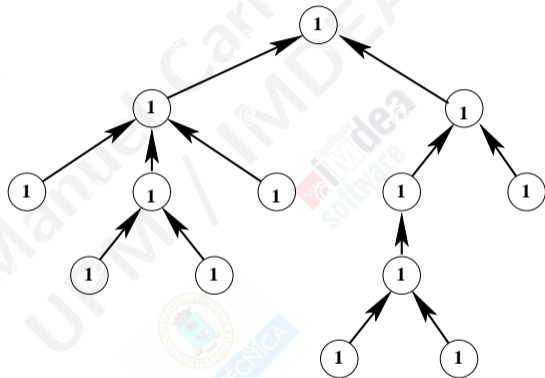
"There is a node that has not been updated and whose children have been updated"



How it is expected to work

Update order restricted:

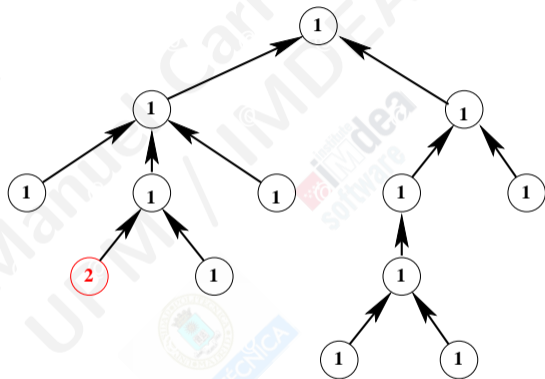
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

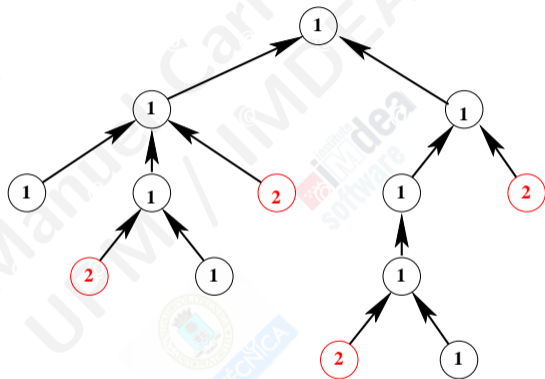
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

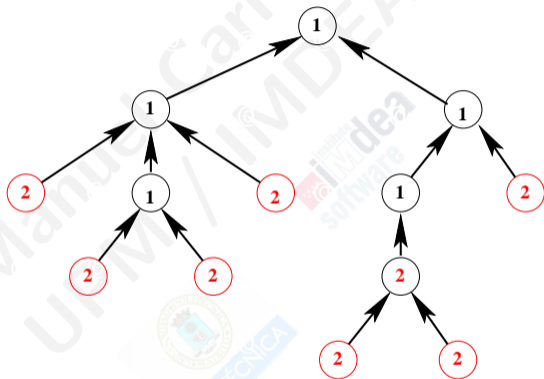
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

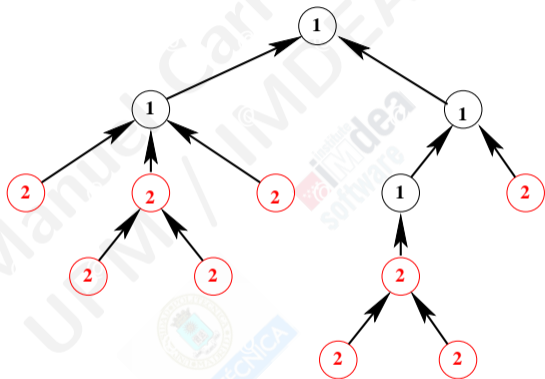
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

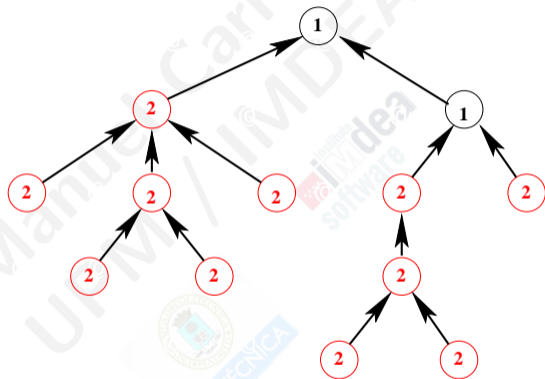
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

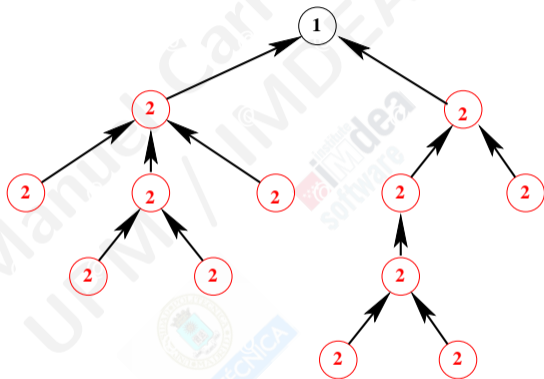
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

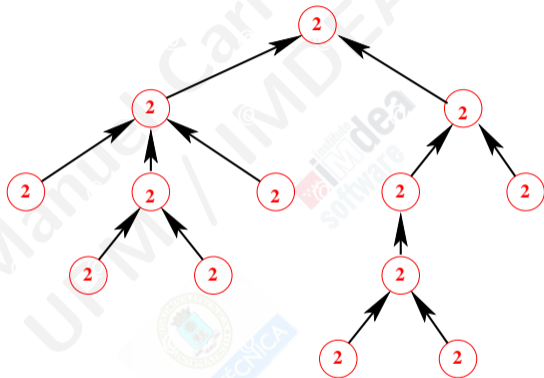
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

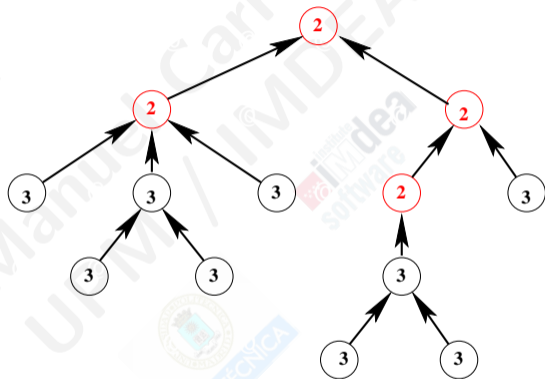
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

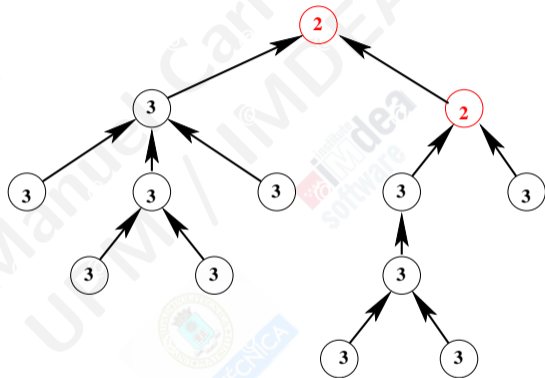
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

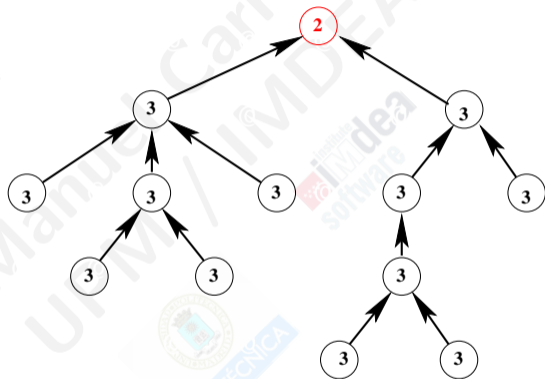
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



FUN 4	Each process can read the counters of its immediate neighbors only
--------------	--

- $\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$ uses only local comparisons.
- $c(r) = c(n)$ uses non-local comparisons.
- We will tackle that in the next refinement.

CONTEXT c1

EXTENDS c0

CONSTANTS

r

f

L

AXIOMS

axm1: $r \in P$

axm3: $L \subseteq P$

Leaves

axm2: $f \in P \setminus \{r\} \rightarrow P \setminus L$

axm4: $\forall S \cdot S \subseteq f^{-1}[S] \Rightarrow S = \emptyset$

axm5:

$\forall S \cdot S \subseteq P \wedge$

$r \in S \wedge$

$(\forall n \cdot n \in P \setminus \{r\} \wedge f(n) \in S \Rightarrow n \in S)$

\Rightarrow

$P \subseteq S$

END

MACHINE m1

REFINES m0

SEES c1

VARIABLES

c

INVARIANTS

inv1: $\forall m \cdot m \in P \setminus \{r\} \Rightarrow c(f(m)) \leq c(m)$

inv2: (theorem) $\forall m \cdot m \in P \Rightarrow c(r) \leq c(m)$

EVENTS

Initialisation (extended)

begin

act1: $c := P \times \{0\}$

end

Event ascending (ordinary) $\hat{=}$

refines ascending

any

n

where

grd1: $n \in P$

grd2: $c(r) = c(n)$

grd3: $\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$

then

act1: $c(n) := c(n) + 1$

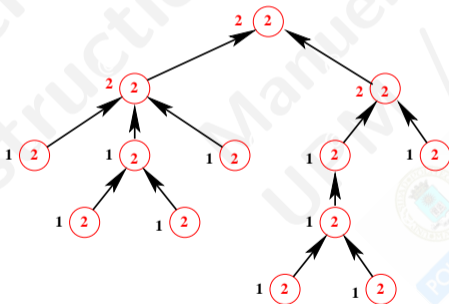
end

END

1. Initial model: all nodes access the state of all nodes.
2. First refinement: restrict access to a single node.
3. **Second refinement: local check, upwards wave.**
4. Third refinement: construct downwards wave.
5. Fourth refinement: remove upwards and downwards counters.

Second refinement

- Replace the guard $c(r) = c(n)$.
- Processes must be aware when this situation does occur.
- Add second counter $d(\cdot)$ to each node to capture value of $c(r)$.



carrier set: P

constants: r, f

variables: c, d

Invariant **inv2_2**
is as **inv0_2**

inv2_1: $d \in P \rightarrow \mathbb{N}$

inv2_2: $\forall x, y \cdot \left(\begin{array}{l} x \in P \\ y \in P \\ \Rightarrow \\ d(x) \leq d(y) + 1 \end{array} \right)$

d has an overall property similar to c :

$$\forall x, y \cdot x \in P \wedge y \in P \Rightarrow c(x) \leq c(y) + 1$$

- d will capture the value of $c(r)$.
- It will be updated in a downward wave.

✓ Refine $m1$ into $m2$

✓ Add variable d and invariants

This refinement captures:

- The existence of d .
- How its update can proceed not to break its invariant.

Event descending

any n where

$$n \in P$$

$$\forall m \cdot m \in P \Rightarrow d(n) \leq d(m)$$

then

$$d(n) := d(n) + 1$$

end

✓ Add event to $m2$

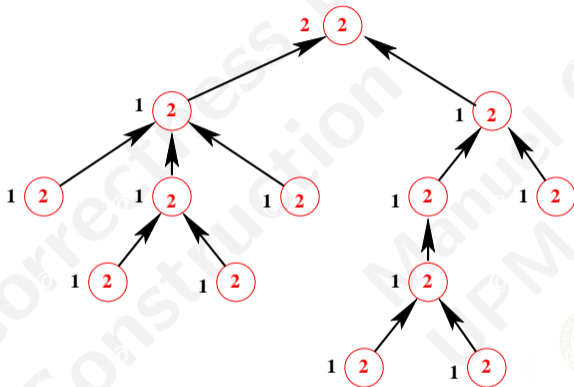
✓ Initialize d to 0 (copy the initialization of c)

1. Initial model: all nodes access the state of all nodes.
2. First refinement: restrict access to a single node.
3. Second refinement: local check, upwards wave.
4. Third refinement: construct downwards wave.
5. Fourth refinement: remove upwards and downwards counters.

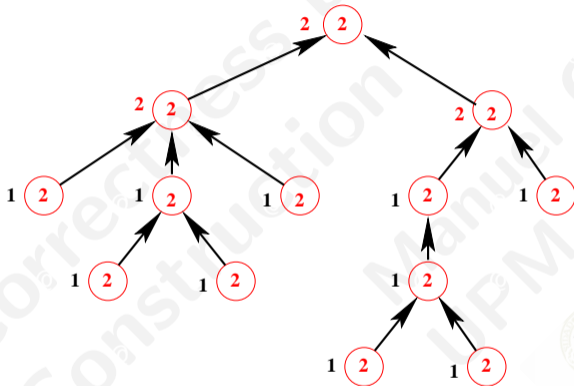
- We extend the invariant of counter d .
- We establish the relationship between both counters c and d .
 - This will allow us to refine event ascending
- We construct the descending wave (by refining event descending).
- **Remark:** this is the most difficult refinement.

✓ *Refine $m2$ into $m3$*

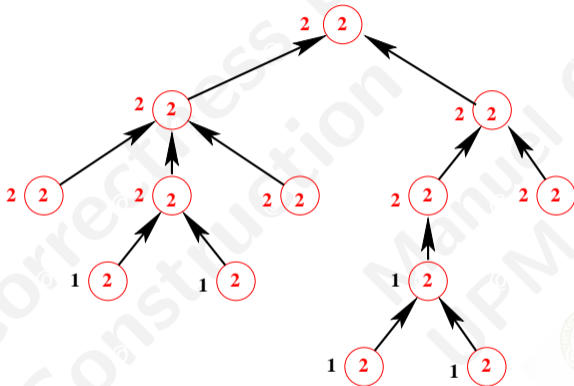
Idea behind third refinement



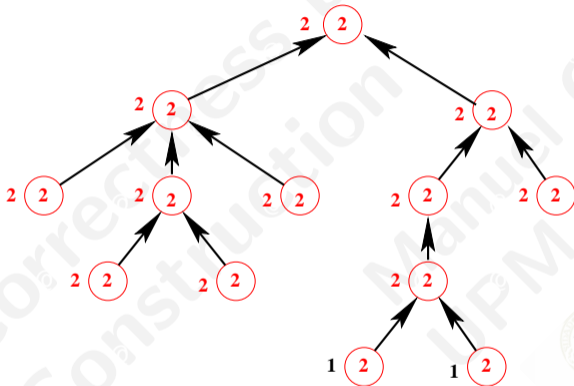
Idea behind third refinement



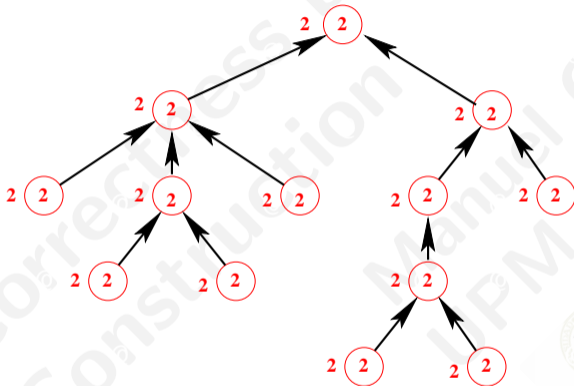
Idea behind third refinement



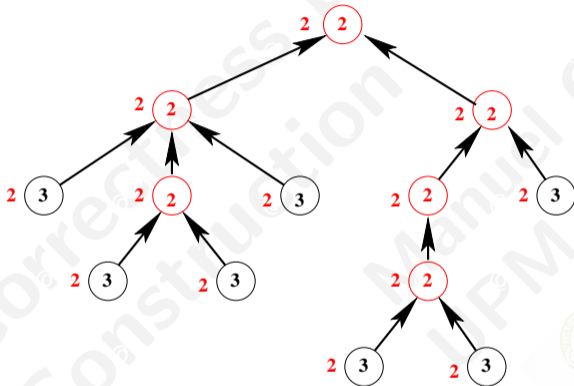
Idea behind third refinement



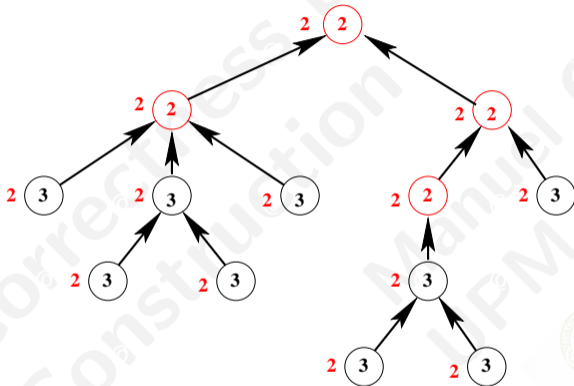
Idea behind third refinement



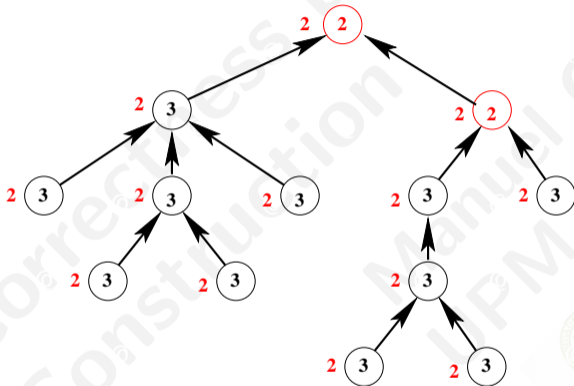
Idea behind third refinement



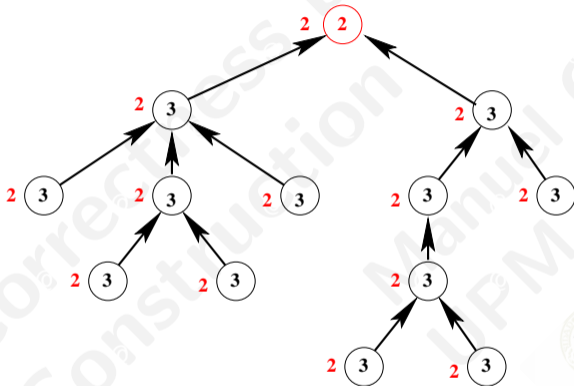
Idea behind third refinement



Idea behind third refinement



Idea behind third refinement



- Recall local condition for c :

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

Every node's counter is smaller than or equal to its children's.

- Local condition for d is similar:

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

Every node's counter is smaller than or equal to its parent (if it has a parent). This is what makes the wave descending.

- inv3_1 and tree induction proves that the root has the **highest** value of $d(\cdot)$:

$$\text{thm3_1} : \forall n \cdot n \in P \Rightarrow d(n) \leq d(r)$$

*(remember: root had the **smallest** value of $c(\cdot)$)*

✓ Add to $m3$:

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

$$\text{thm3_1} : \forall n \cdot n \in P \Rightarrow d(n) \leq d(r)$$

✓ Mark the latter as theorem

✓ Double click on the PO for THM

✓ Go to proving perspective; locate induction axiom

✓ Instantiate with $\{x \mid x \in P \wedge d(x) \leq d(r)\}$, invoke $p0$

✓ That should prove thm3_1

✓ inv3_1 cannot be proved yet - reasons similar to c .

We will deal with that later

Refining *ascending*

Event (abstract—)ascending

any n where

$$n \in P$$

$$c(n) = c(r)$$

$$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$$

then

$$c(n) := c(n) + 1$$

end

- Downward wave d will eventually propagate $d(r)$.

✓ *Change event guard in $m3$*

Event (concrete—)ascending

any n where

$$n \in P$$

$$c(n) = d(n)$$

$$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$$

then

$$c(n) := c(n) + 1$$

end

ascending: only **local** comparisons now!

Refining *ascending*

Event (abstract—)ascending

any n where

$$n \in P$$

$$c(n) = c(r)$$

$$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$$

then

$$c(n) := c(n) + 1$$

end

- Downward wave d will eventually propagate $d(r)$.

✓ *Change event guard in $m3$*

- Need to prove guard strengthening.

Event (concrete—)ascending

any n where

$$n \in P$$

$$c(n) = d(n)$$

$$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$$

then

$$c(n) := c(n) + 1$$

end

ascending: only **local** comparisons now!

Refining *ascending*

Event (abstract—)ascending

any n where

$$n \in P$$

$$c(n) = c(r)$$

$$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$$

then

$$c(n) := c(n) + 1$$

end

- Downward wave d will eventually propagate $d(r)$.

✓ *Change event guard in $m3$*

- Need to prove guard strengthening.
- We cannot.** c and d unrelated so far!

✓ *Relate c and d : $inv3_2 : d(r) \leq c(r)$*

- If needed: proving perspective, lasso + **p0** proves strengthening.

Event (concrete—)ascending

any n where

$$n \in P$$

$$c(n) = d(n)$$

$$\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) \neq c(m)$$

then

$$c(n) := c(n) + 1$$

end

ascending: only **local** comparisons now!

- A different case.
- Two situations raise a change of d :
 1. For a non-root node: parent's d change.
 2. For the root node: $c(r)$ changes.
- Different guards.
- We will prepare the events to be edited.

✓ *Change (concrete) descending event to non-extended*

✓ *Left click on circle to left of name to select*

Ctrl-C to copy, Ctrl-V to paste

✓ *Rename first event as `descending_nr`.*

✓ *Rename second event as `descending_r`.*

Refining *descending*: the non-root case

Event (abstract—)descending

any n where

$$n \in P$$

$$\forall m \cdot m \in N \Rightarrow d(n) \leq d(m)$$

then

$$d(n) := d(n) + 1$$

end

Event (concrete—)descending

any n where

$$n \in P \setminus \{r\}$$

$$d(n) \neq d(f(n))$$

then

$$d(n) := d(n) + 1$$

end

✓ *Update guards*

(Note: Rodin ≥ 3.6 seems to prove strengthening automatically; previous versions needed additional steps [in next slide])

Note: the steps below do not seem to be necessary in Rodin 3.6 with the Atelier B provers installed. Strengthening is proven automatically.

$$n \in P \setminus \{r\}, d(n) = d(f(n)), m \in P \vdash d(n) \leq d(m)$$

We need some magic mushrooms to help the provers:

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

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

thm3_2 downward wave, parent is at most one more than children (when it has just been increased)

thm3_3 special case for root (the first one to be increased)

Refining *descending* (Cont. — the root case.)

Event (abstract—)descending

any n where

$n \in P$

$\forall m \cdot m \in P \Rightarrow d(n) \leq d(m)$

then

$d(n) := d(n) + 1$

end

Event (concrete—)descending

refines

descending

when

$d(r) \neq c(r)$

with

$n: n = r$

then

$d(r) := d(r) + 1$

end

✓ *Click on circle left of param. n , delete*

- Parameter n disappeared!
- Substitute (*witness*) for GRD, SIM.
- We are particularizing for r .

- Similar to gluing invariant!
- Note **with** label: specific Rodin idiom.
- Need to prove
 $d(r) \neq c(r), m \in P \vdash d(r) \leq d(m)$

Finishing proofs

The technique in this slide was necessary for Rodin versions previous to 3.6. For Rodin 3.6 onwards, it seems that it is not necessary. Skip to the next slide.

I needed two more magic pills:

inv3_3 : $\forall n \cdot n \in P \Rightarrow c(n) \in d(n)..d(n) + 1$ To prove GRD

thm3_4 : $\forall n \cdot n \in P \Rightarrow c(r) \in d(n)..d(n) + 1$ To prove inv3_3

Plus, if not added before:

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

thm3_3 : $\forall n \cdot n \in P \Rightarrow d(r) \in d(n)..d(n) + 1$

After this, the invariant can be proved with a combination of several steps:

- Apply lasso.
- Instantiate $\forall n \cdot c(r) \in d(n)..d(n) + 1$ (which relates c and d) with n .
- Remove \in in goal
($c(n) \in d(n) + 1..d(n) + 1 + 1$) to create inequalities.
- Do **PO** in $c(n) \leq d(n) + 1 + 1$ goal.
- Note that only possibility to prove is $d(n) = c(n)$.
- Do case distinction with $d(n) = c(n)$,
- Apply **ML** to the subgoals.

Finishing proofs

This strategy is necessary with Rodin 3.6 and 3.7 and, apparently, 3.9.

An additional invariant is necessary to prove GRD of descending_r:

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

After adding it, GRD is immediately proven. However, the invariant remains unproven. It can be proved with the following steps:

- Apply lasso.
- Remove \in in goal
 $c(n) \in d(n) + 1..d(n) + 1 + 1$ to transform it into inequalities that can be proven separately.
- Use **ml** or **p0** for the goal
- For $d(n) + 1 \leq c(n)$, do case distinction:
 - Either with $d(n) = c(n)$, or
 - with $d(n) + 1 = c(n)$and **ML** to the subgoals.

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

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

inv3_3: $\forall n \cdot (n \in P \Rightarrow c(n) \in d(n) .. d(n) + 1)$

thm3_1: $\forall m \cdot (m \in P \Rightarrow d(m) \leq d(r))$

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

thm3_3: $\forall n \cdot (n \in P \Rightarrow d(r) \in d(n) .. d(n) + 1)$

thm3_4: $\forall n \cdot (n \in P \Rightarrow c(r) \in d(n) .. d(n) + 1)$

Third refinement: events

```
Event descending_r  
  when  
     $d(r) \neq c(r)$   
  with  
    n:  $n = r$   
  then  
     $d(r) := d(r) + 1$   
  end
```

```
Event descending_nr  
  any n where  
     $n \in P \setminus \{r\}$   
     $d(n) \neq d(f(n))$   
  then  
     $d(n) := d(n) + 1$   
  end
```

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

1. Initial model: all nodes access the state of all nodes.
2. First refinement: restrict access to a single node.
3. Second refinement: local check, upwards wave.
4. Third refinement: construct downwards wave.
5. Fourth refinement: remove upwards and downwards counters.

- The difference among counters is at most one.
 - That has been proven by construction.
- In the guards, we only care whether they are equal or not.
- For this, we only need parity!

$$a, b \in \mathbb{N} \wedge |a - b| \leq 1 \Rightarrow (a = b \Leftrightarrow \text{parity}(a) = \text{parity}(b))$$

- We will prove that this is a valid refinement.

✓ *Extend context $c1$ into $c2$*

✓ *Refine $m3$ into $m4$*

✓ *$m4$ should see $c2$*

Formalizing parity

- We replace the counters by their parities
- we add the constant *parity*

carrier set: P

constants: r, f, \textit{parity}

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

axm4_2: $\textit{parity}(0) = 0$

axm4_2: $\forall x. (x \in \mathbb{N} \Rightarrow \textit{parity}(x + 1) = 1 - \textit{parity}(x))$

✓ Add *parity* and axioms to *c2*. Note: *parity* is a function!

✓ Need some clicking (*dom* to \mathbb{N} + *ML*) to prove *WD*

The definitions that replace $c(\cdot)$ and $d(\cdot)$

- We replace c and d by p and q

variables: p, q

inv4.1: $p \in P \rightarrow \{0, 1\}$

inv4.2: $q \in P \rightarrow \{0, 1\}$

inv4.3: $\forall n. (n \in P \Rightarrow p(n) = \text{parity}(c(n)))$

inv4.4: $\forall n. (n \in P \Rightarrow q(n) = \text{parity}(d(n)))$

- ✓ Do it in m_4 . Note the gluing invariants! p and q really syntactic sugar.
- ✓ Remove variables c and d . Not accessed / updated in this refinement!
- ✓ Initialize p and q , remove initializations for c and d .

New events: counters replaced by parity

```
ascending
  any  $n$  where
     $n \in P$ 
     $p(n) = q(n)$ 
     $\forall m \cdot (m \in f^{-1}[\{n\}] \Rightarrow p(m) \neq p(n))$ 
  then
     $p(n) := 1 - p(n)$ 
  end
```

```
descending_1
  any  $n$  where
     $n \in P \setminus \{r\}$ 
     $q(n) \neq q(f(n))$ 
  then
     $q(n) := 1 - q(n)$ 
  end
```

```
descending_2
  when
     $p(r) \neq q(r)$ 
  then
     $q(r) := 1 - q(r)$ 
  end
```

GRD of $q(n) = p(n)$

- The essence of the pending GRD proof is $\dots, q(n) = p(n) \vdash c(n) = d(n)$.
- Depends on proving $\text{parity}(a) = \text{parity}(b) \Rightarrow a = b$.
- Holds in specific cases (if $|a - b| \leq 1$).
- But theorem provers unable to apply / deduce that property.
- Needs to be stated explicitly:

$$\forall x, y \cdot y \in \mathbb{N} \wedge x \in y..y + 1 \Rightarrow (\text{parity}(x) = \text{parity}(y) \Leftrightarrow x = y)$$

- We could make it axiom, but it can be proven as theorem (better!).

- We need to deal with Well Definedness and the theorem itself.
 - WD: Removing **dom** in goal + **P0** takes care of it (if **WD** is to be discharged).
 - THM: Adding hypothesis + case distinction works. See below.
- \iff : split in two implications. One is proven.
- For the other: introduce **ah** with possible values of x : $x = y \vee x = y + 1$ (because $x \in y..y + 1$ among the hypotheses).
- Prove new hypothesis with **ml**.
- For the pending $x = y$ goal, bring hypotheses with lasso.
- New goal: $y = y + 1$. We need to find contradiction in hypotheses.
- One hypothesis is $parity(y + 1) = parity(y)$, which is false.
- Use **dc** with $parity(y) = 0$. This causes two instantiations that make proving inconsistency easier.
- **P0** works for both branches.

GRD of $q(n) = p(n)$

- Do lasso.
- Instantiate theorem

$$\forall x, y \cdot y \in \mathbb{N} \wedge x \in y..y + 1 \Rightarrow \\ (\text{parity}(x) = \text{parity}(y) \Leftrightarrow x = y)$$

with $c(n)$, $d(n)$.

(Bring it from hypotheses if not among selected hypotheses).

- **Note:** instantiate the right variable with the right value!
- Invoke **P0** for the branches remaining to be proven.

```
✓ simplification rewrites : c(n)=d(n)
  ✓ type rewrites : c(n)=d(n)
    ✓ simplification rewrites : c(n)=d(n)
      ✓ sl/ds : c(n)=d(n)
        ✓ sl/ds : c(n)=d(n)
          ✓ ∀ hyp (inst c(n),d(n)) : c(n)=d(n)
            ✓ generalized MP : (n∈dom(d)∧d∈P → Z)∧(n∈dom(c)∧c∈P → Z)
              ✓ simplification rewrites : (T∧T)∧(T∧T)
                ✓ T goal : T
            ✓ generalized MP : c(n)=d(n)
              ✓ simplification rewrites : c(n)=d(n)
                ✓ PP : c(n)=d(n)
```

GRD of $\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow p(n) \neq p(m)$

Idea: we have to prove that if $p(m) \neq p(f(m))$, then $c(n) \neq c(f(m))$. We have a theorem that says $\text{parity}(x) = \text{parity}(y) \Leftrightarrow x = y$ when $x \in y..y + 1$. So we need $c(n) \in c(f(n))..c(f(n)) + 1$ to apply it. We add it as a **theorem**, which is immediately proven.

1. Add a new THM: $\forall n \cdot n \in P \setminus \{r\} \Rightarrow c(n) \in c(f(n))..c(f(n)) + 1$
2. Click on the PO for the undischarged GRD.
3. Introduce the hypothesis $n = f(m)$ (which comes from $m \in f^{-1}[\{n\}]$) with **ah** and use **ML** repeatedly.
4. If some subgoal is not proven, bring all the available hypotheses from the Search window and use ML.

- In my case, GRD for $q(n) \neq q(f(n))$ in **descending_nr** remains to be proven.
- It should imply $d(n) \neq d(f(n))$.
- Similar to the previous case.
- Add a symmetrical theorem
 $\forall n \cdot n \in P \setminus \{r\} \Rightarrow d(f(n)) \in d(n)..d(n) + 1$
- It is immediately proven and it also automatically discharges the pending GRD proof.

- With Rodin 3.8 it may be the case that the invariant

$$\forall n \cdot n \in P \Rightarrow p(n) = \text{parity}(c(n))$$

is unproven.

- The goal to be proven is

$$1 - p(n) = \text{parity}(c(n) + 1)$$

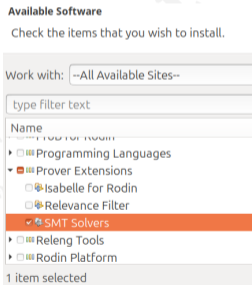
which is immediate from the definition of *parity* and *c(n)*.

- Just apply lasso, instantiate the definition of *parity* with *c(n)*, and use **ML** or **PO**

At this point, all the POs should be discharged.

Less Manual Work?

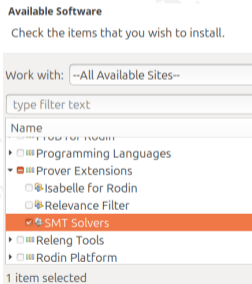
- **Atelier B provers**: developed for (Event-)B, integrated with Rodin.
- Not the most powerful provers.
- Additional provers: Install Software → Work with “– All Available Sites –” → Prover Extensions → SMT Solvers.



- Can often discharge proofs without / with less manual intervention.

Less Manual Work?

- **Atelier B provers**: developed for (Event-)B, integrated with Rodin.
- Not the most powerful provers.
- Additional provers: Install Software → Work with “– All Available Sites –” → Prover Extensions → SMT Solvers.



- Can often discharge proofs without / with less manual intervention.

- Why not using them before?
 - SMT solvers external → stability not guaranteed?
 - If using SMT Solvers, examples requiring interaction likely too complex for a first contact.
- Install and try the SMT solvers in the examples in this section of the course using less additional theorems / invariants.
- Plugin feeds SMT solvers with “Selected Hypothesis”. Possible heuristics:
 - Bring hypotheses to the “Selected” set with lasso, or
 - Select all hypotheses in “Search” tab, add them to “Selected”.