Synchronizing Processes on a Tree Network[1]

Manuel Carro
manuel.carro@upm.es

IMDEA Software Institute &
Universidad Politécnica de Madrid

[1]Example and most slides borrowed from J. R. Abrial: see
http://wiki.event-b.org/index.php/Event-B_Language

Purpose of this lecture

- Learning a few more modeling conventions.
- Learning more about abstraction.
- Formalizing and proving on an interesting structure: a tree.
  - Will have an intermediate step to review functions, relations, data structures.
- Study a more complicated problem in distributed computing
- Example studied in: *W.H.J. Feijen and A.J.M. van Gasteren. On a Method of Multi-programming. Springer Verlag, 1999.*

As usual:

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

Prerequisites

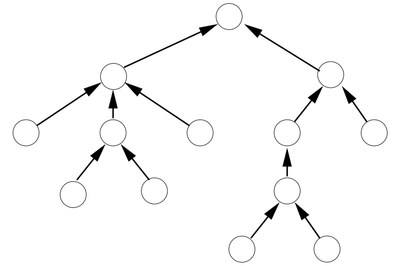
- Knowledge of first order logic, set theory, relations, and functions.
- Rodin (to discharge the proofs).
- Slides:
  - *Event B: Sets, Relations, Functions, Data Structures*
- Please go through them.
- I will review parts of it here, when needed.

Comparison with previous examples

- 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.
- Model small: just three events in the last refinement.
- However, proofs and reasoning involved.

Requirements

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.

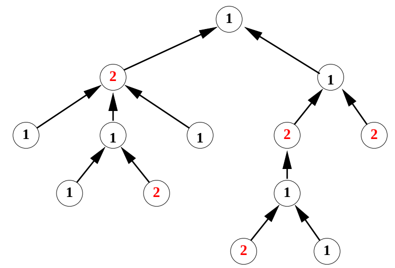
Requirements (Cont.)

- 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

Requirements (Cont.)



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

Requirements (Cont.)

- Reading the counters

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

- Modifying the counters

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

Refinement strategy

- 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)

Steps

1. Initial model: all nodes access to 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.

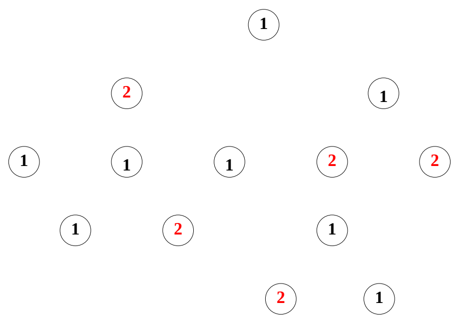
Initial model: the state

- 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 an initial model!

Initial model: the state

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 at first glance:

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

- Is it the same as $\forall i, j \cdot |c(i) - c(j)| \leq 1$?
- Disprove it or convince us!

Is that right?

- **inv0_2** may be surprising at first glance:

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

- Is it the same as $\forall i, j \cdot |c(i) - c(j)| \leq 1$?
- Disprove it or convince us!

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 $|a - b| \leq 1$.
- 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.

Initial model: events

```
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: it is logically $\exists n \cdot n \in P \land \cdots$
- Process counter incremented only when $\leq$ to all other counters.
- Intuition: *If I see I can increase without breaking difference constraint, I do it!*
- Non-determinism!
- A specification of what should happen.
- Not a final state (there is not one): a procedure that (hopefully) respects the invariant.

✓ *Add initialization, event*

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

Proof of invariant preservation

$$c \in P \rightarrow \mathbb{N} \qquad \textbf{inv0\_1}$$

$$\forall x, y \cdot \begin{pmatrix} x \in P \\ y \in P \\ \Rightarrow \\ c(x) \leq c(y) + 1 \end{pmatrix} \qquad \textbf{inv0\_2}$$

$$\begin{array}{l} n \in P \\ \forall m \cdot (m \in P \Rightarrow c(n) \leq c(m)) \end{array} \qquad \text{Guards of event ascending}$$

$$\vdash$$

$$\forall x, y \cdot \begin{pmatrix} x \in P \\ y \in P \\ \Rightarrow \\ (c \Leftarrow \{n \mapsto c(n)+1\})(x) \leq (c \Leftarrow \{n \mapsto c(n)+1\})(y) + 1 \end{pmatrix}$$

$$\Uparrow$$

Modified invariant **inv0_2**

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

Model so far

```
CONTEXT c0
SETS
    P
AXIOMS
    axm1:  finite(P)
END
```

```
MACHINE m0
SEES c0
VARIABLES
    c
INVARIANTS
    inv1:  c ∈ P → ℕ
    inv2:  ∀x,y·x ∈ P ∧ y ∈ P ⇒ c(x) ≤ 1 + c(y)
EVENTS
Initialisation
    begin
        act1:  c := P × {0}
    end
Event ascending ⟨ordinary⟩ ≙
    any
        n
    where
        grd12:  n ∈ P
        grd11:  ∀m·m ∈ P ⇒ c(n) ≤ c(m)
    then
        act11:  c(n) := c(n) + 1
    end
```

Problem with the current event

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?

Problem with the current event

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

Steps

1. Initial model: all nodes access to 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.
- We suppose that the counter of r is 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.
  - Because: if for any m either $c(m) = c(r)$ or $c(m) = c(r) + 1$, then difference between any $m, n \leq 1$.
- Treat this property as a new (temporary) invariant.

✓ *Extend `c0` into `c1` (left pane, right click, "Extend"), add constant `r`, axiom `r` $\in$ PP*
✓ *Refine `m0` into `m1` (left pane, right click, "Refine"), add new invariant*
✓ *`m0` should "see" `c1`*

First refinement: proposal for the event refinement

We simplify the guard

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

Guard strengthening

$c \in P \to \mathbb{N}$ **inv0_1**

$\forall x, y \cdot \begin{pmatrix} x \in P \\ y \in P \\ \Rightarrow \\ c(x) \leq c(y) + 1 \end{pmatrix}$ **inv0_2**

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

$n \in P$ Guards of concrete

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

$\vdash$

$n \in P$ Guards of abstract

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

In Rodin: lasso + p0

✓ *Go to the proving perspective, discharge proof*

Model so far

`inv1` not discharged.

```
CONTEXT c1
EXTENDS c0
CONSTANTS
    r
AXIOMS
    axm1: r ∈ P
END
```

```
MACHINE m1
REFINES m0
SEES c1
VARIABLES
    c
INVARIANTS
    inv1: ∀m·m ∈ P ⇒ c(r) ≤ c(m)
EVENTS
Initialisation ⟨extended⟩
    begin
        act1: c := P × {0}
    end
Event ascending ⟨ordinary⟩ ≙
refines ascending
    any
        n
    where
        grd1: n ∈ P
        grd2: c(r) = c(n)
    then
        act1: c(n) := c(n) + 1
    end
END
```

Pending problems

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

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

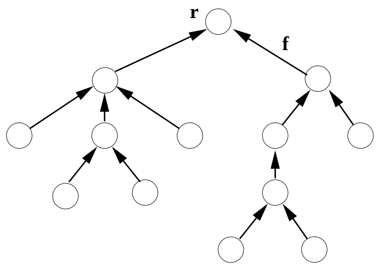
1. Prove that new "invariant" is preserved by the event.
2. The guard of the event still does not fulfill requirement FUN 4.

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.

Update model

✓ *Add to* `c1` *(note f is $\twoheadrightarrow$, written* `-->>`*)*

- Constant f.
- Axioms:

$$L \subseteq P$$
$$f \in P \setminus \{r\} \twoheadrightarrow P \setminus L$$
$$\forall S \cdot S \subseteq f^{-1}[S] \Rightarrow S = \varnothing$$

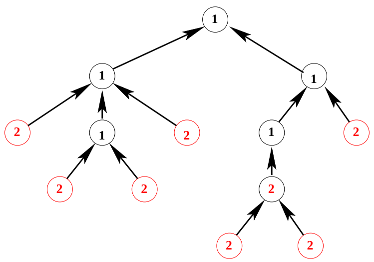
- f^{-1} is written `f~`.
- $\twoheadrightarrow$: f defined for all $P \setminus \{r\}$ and *arrives* to every element in $P \setminus L$.

Minimal counter at the root

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

$$\textbf{inv1\_1} : \forall m \cdot m \in P \setminus \{r\} \Rightarrow c(f(m)) \leq c(m)$$

✓ *Add it to* `m1`



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

- 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?

Minimal counter at the root

- Minimality of counter at the root

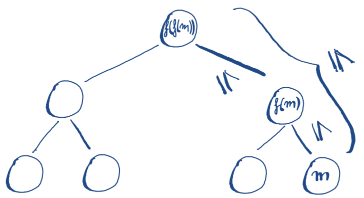
$$\forall m \cdot m \in P \Rightarrow c(r) \leq c(m)$$

  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?

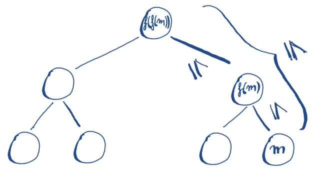
We need to *extend* the local property

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

to the whole tree.



Minimal counter at the root: induction



- Start with leaves $l \in L$.
- Prove that for any l, $c(f(l)) \leq c(l)$, then $c(f(f(l))) \leq c(f(l)) \leq c(l), \ldots$
- 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. $m = f(m')$, $c(m) \leq c(m')\ldots$
- And so on towards the leaves.

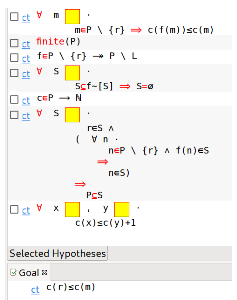
Minimal counter at the root: induction

- Induction: difficult for theorem provers to do on their own.
  - Needs to identify base case, property to use for induction — i.e., the *strategy*.
- Proving property for base case & inductive step within theorem provers' capabilities.
- In Rodin: needs adding induction scheme:
  ✓ *Add to* `c1`*:*
  $\forall S \cdot S \subseteq P \wedge r \in S \wedge (\forall n \cdot n \in P \setminus \{r\} \wedge f(n) \in S \Rightarrow n \in S) \Rightarrow P \subseteq S$
  ✓ *Tip:* `Ctrl-Enter` *breaks text in input box in separate lines.*
- Instantiating it with the property to prove expressed as a set:
  `{x | x ∈ P ∧ c(r) ≤ c(x)}` (next slide)

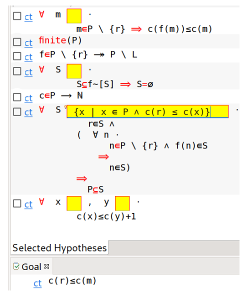
✓ *In* `m1`*: ensure you have* **inv1_1**: $\forall m \cdot m \in P \backslash \{r\} \Rightarrow c(f(m)) \leq c(m)$
✓ *Ensure* **thm1_1**: $\forall m \cdot m \in P \Rightarrow c(r) \leq c(m)$ *below invariant, marked as theorem*

Induction in Rodin: instantiation

- Double click in the unproved theorem (left pane).
- Switch to prover view, lasso.
- Locate induction axiom.
- Enter
  `{x | x ∈ P ∧ c(r) ≤ c(x)}`.
- Return and p0.
- The theorem should be proved now.



$\Longrightarrow$



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

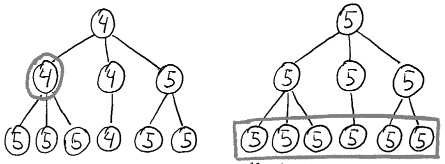
- Nodes with difference $\leq$ one from r.
- When can we update looking locally?

```
ascending
   any  n  where
      n ∈ P
      c(r) = c(n)
      ∀m · m ∈ f⁻¹[{n}] ⇒ c(n) < 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"

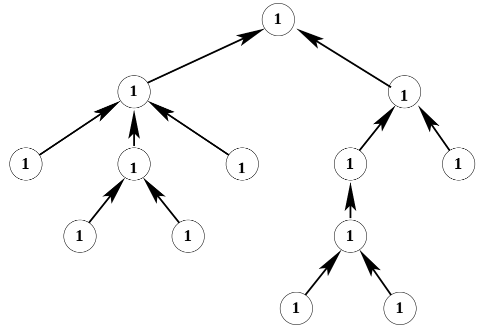


All of them can be updated

How it is expected to work

Update order restricted:

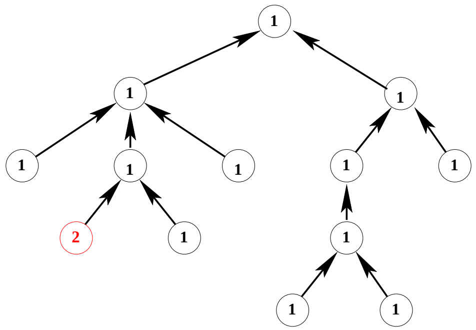
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

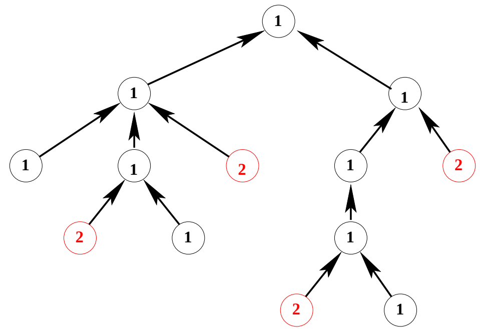
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

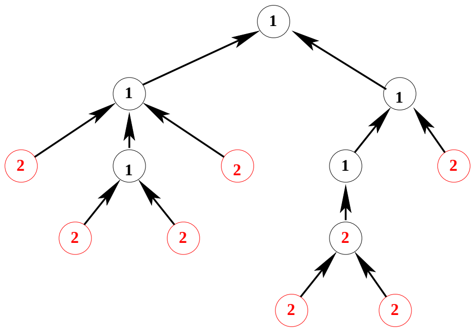
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

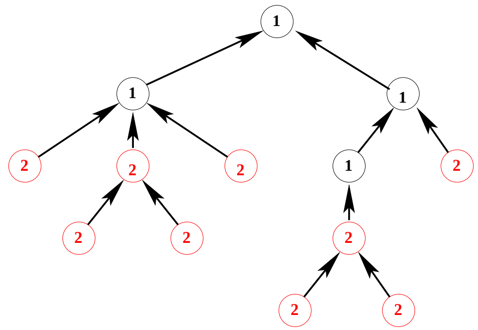
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

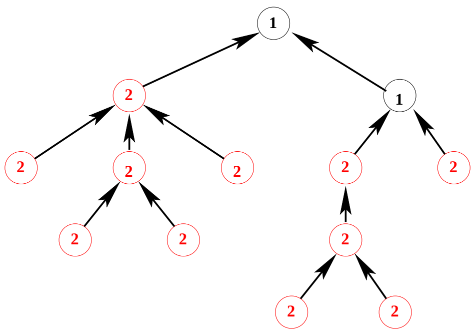
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

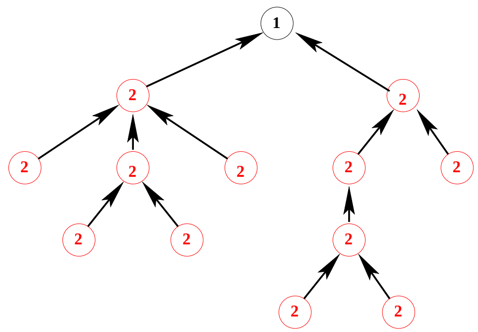
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

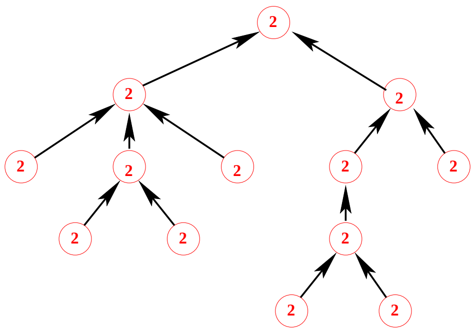
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

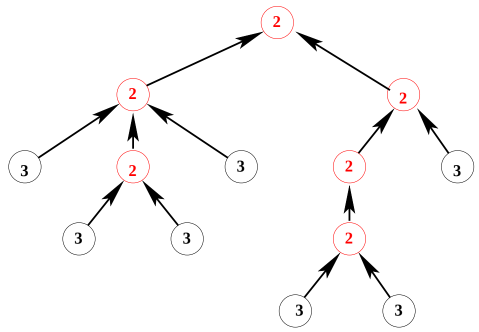
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

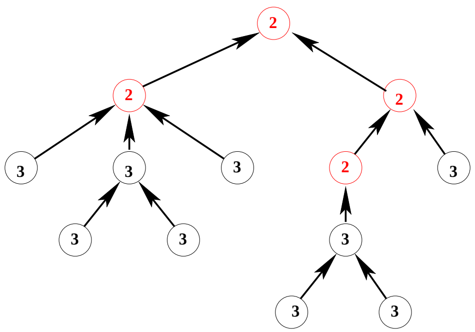
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

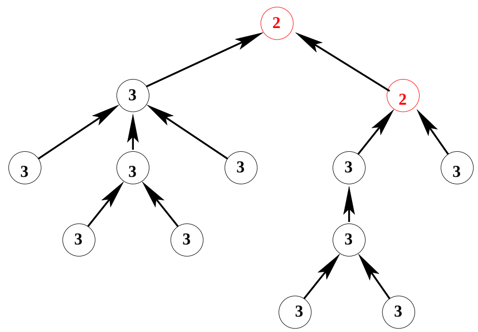
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



How it is expected to work

Update order restricted:

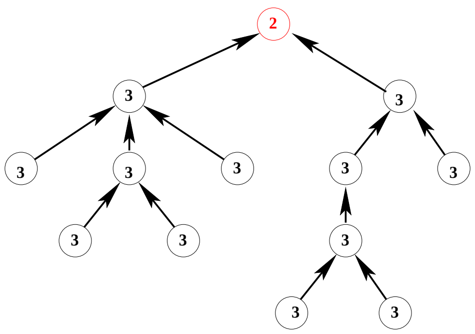
- **Before:** any node whose counter is equal to the root (the one with the minimum).
- **Now:** only those nodes whose counters are, in addition, smaller than all its descendants.
- Updates will go in waves towards the root.



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.



Neighborhood checking

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

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

Model so far

CONTEXT c1
EXTENDS c0
CONSTANTS
  r
  f
  L
AXIOMS
  axm1: $r \in P$
  axm3: $L \subseteq P$
    Leaves
  axm2: $f \in P \setminus \{r\} \twoheadrightarrow P \setminus L$
  axm4: $\forall S \cdot S \subseteq f^{-1}[S] \Rightarrow S = \varnothing$
  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

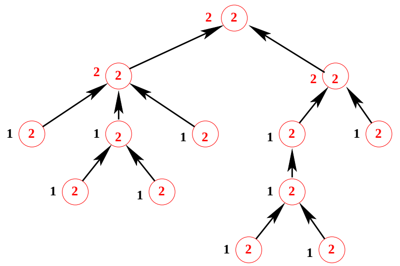
MACHINE m1
REFINES m0
SEES c1
VARIABLES
  c
INVARIANTS
  inv1: $\forall m \cdot m \in P \setminus \{r\} \Rightarrow c(f(m)) \leq c(m)$
  inv2: ⟨theorem⟩ $\forall m \cdot m \in P \Rightarrow c(r) \leq c(m)$
EVENTS
Initialisation ⟨extended⟩
  begin
    act1: $c := P \times \{0\}$
  end
Event ascending ⟨ordinary⟩ $\widehat{=}$
refines ascending
  any
    n
  where
    grd1: $n \in P$
    grd2: $c(r) = c(n)$
    grd3: $\forall m \cdot m \in f^{-1}[\{n\}] \Rightarrow c(n) < c(m)$
  then
    act1: $c(n) := c(n) + 1$
  end
END

Steps

1. Initial model: all nodes access to 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 $c(r)$.



Second refinement: the state

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: $\forall x, y \cdot \begin{pmatrix} x \in P \\ y \in P \\ \Rightarrow \\ d(x) \leq d(y) + 1 \end{pmatrix}$

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*

Updating d

This refinement captures:

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

```
Event descending
   any n where
      n ∈ P
      ∀m · m ∈ P ⇒ d(n) ≤ d(m)
   then
      d(n) := d(n) + 1
   end
```

✓ *Add event to m2*

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

Steps

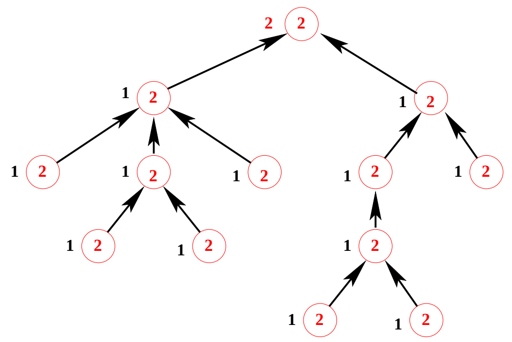
1. Initial model: all nodes access to the state of all nodes.
2. First refinement: restrict access to a single node.
3. Second refinement: local check, upwards wave.
4. $\boxed{\text{Third refinement: construct downwards wave.}}$
5. Fourth refinement: remove upwards and downwards counters.

Third refinement

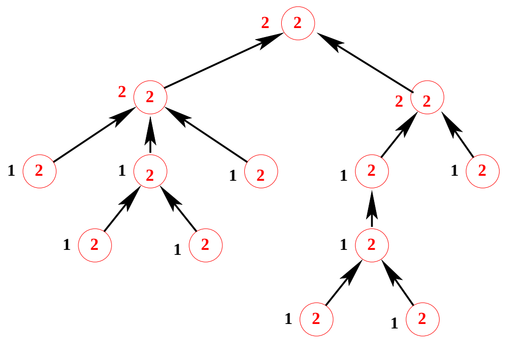
- We extend the invariant of counter d.
- We establish the relationship between both counters c and d.
  - This will allow us to refine event ascending
- We construct the descending wave (by refining event descending).
- Remark: this is the most difficult refinement.

✓ *Refine m2 into m3*

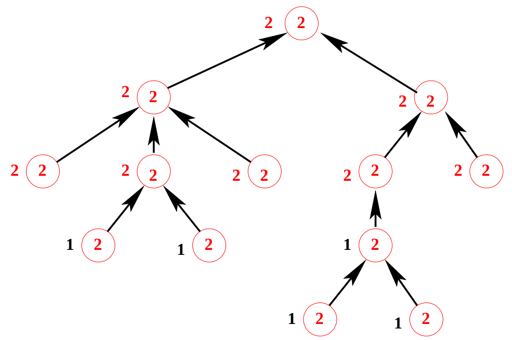
Idea behind third refinement



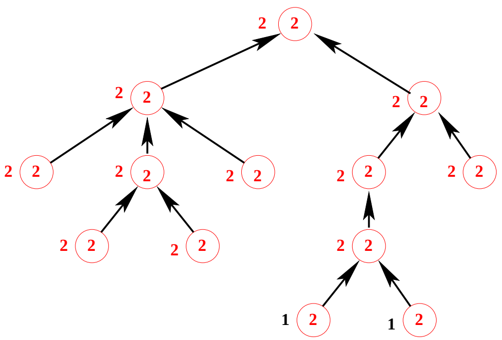
Idea behind third refinement



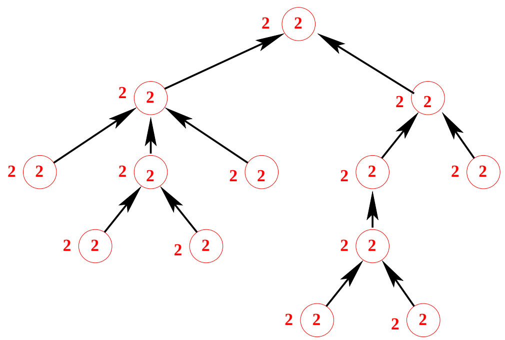
Idea behind third refinement



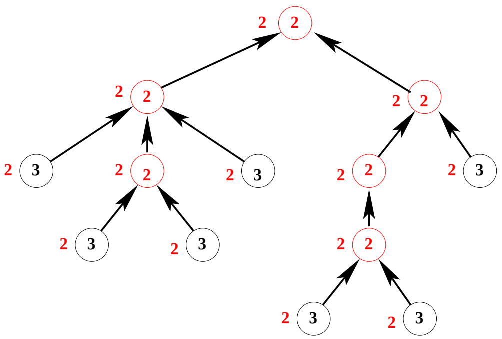
Idea behind third refinement



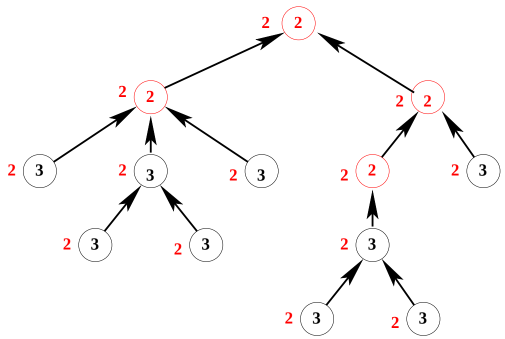
Idea behind third refinement



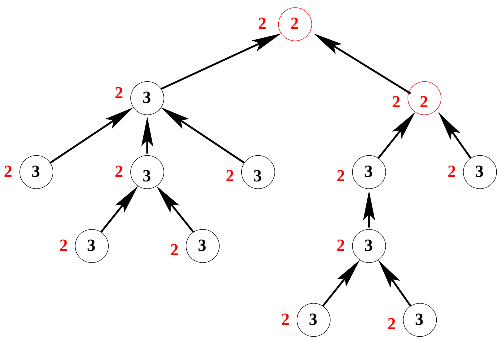
Idea behind third refinement



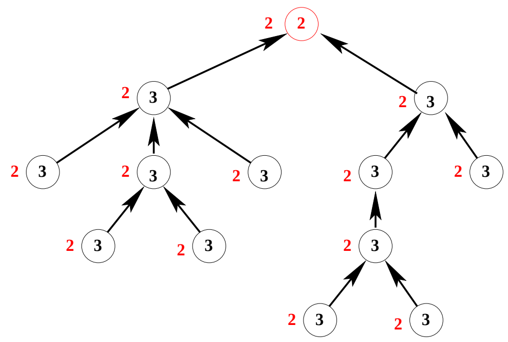
Idea behind third refinement



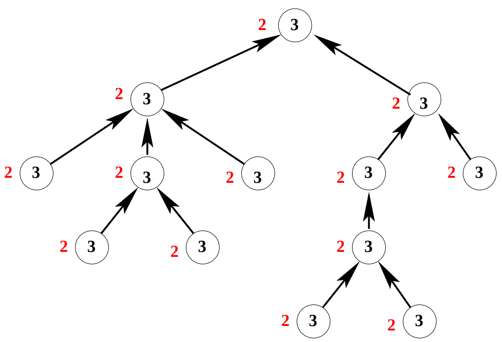
Idea behind third refinement



Idea behind third refinement



Idea behind third refinement



State and invariants

- Recall local condition for c:

$$\textbf{inv1\_1} : \forall m \cdot m \in P \setminus \{r\} \Rightarrow c(f(m)) \leq c(m)$$

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

- Local condition for d is similar:

$$\textbf{inv3\_1} : \forall m \cdot m \in P \setminus \{r\} \Rightarrow d(m) \leq d(f(m))$$

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 $d(\cdot)$:

$$\textbf{thm3\_1} : \forall n \cdot n \in P \Rightarrow d(n) \leq d(r)$$

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

Proving theorem and invariant

✓ *Add to m3:*

$$inv3\_1: \quad \forall m \cdot m \in P \setminus \{r\} \Rightarrow d(m) \leq d(f(m))$$

$$thm3\_1: \quad \forall n \cdot n \in P \Rightarrow d(n) \leq d(r)$$

✓ *Mark the latter as theorem*
✓ *Double click on the PO for THM*
✓ *Go to proving perspective; locate induction axiom*
✓ *Instantiate with $\{x | x \in P \wedge d(x) \leq d(r)\}$, 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 ∈ P
        c(n) = c(r)
        ∀m · m ∈ f⁻¹[{n}] ⇒ c(n) < c(m)
    then
        c(n) := c(n) + 1
    end
```

```
Event (concrete−)ascending
    any n where
        n ∈ P
        c(n) = d(n)
        ∀m · m ∈ f⁻¹[{n}] ⇒ c(n) < c(m)
    then
        c(n) := c(n) + 1
    end
```

- Downward wave d will eventually propagate $d(r)$.
  ✓ *Change event guard in m3*

ascending: only local comparisons now!

Refining *ascending*


```
Event (abstract−)ascending
    any n where
        n ∈ P
        c(n) = c(r)
        ∀m · m ∈ f⁻¹[{n}] ⇒ c(n) < c(m)
    then
        c(n) := c(n) + 1
    end
```

```
Event (concrete−)ascending
    any n where
        n ∈ P
        c(n) = d(n)
        ∀m · m ∈ f⁻¹[{n}] ⇒ c(n) < 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.

ascending: only local comparisons now!

Refining *ascending*


```
Event (abstract−)ascending
    any n where
        n ∈ P
        c(n) = c(r)
        ∀m · m ∈ f⁻¹[{n}] ⇒ c(n) < c(m)
    then
        c(n) := c(n) + 1
    end
```

```
Event (concrete−)ascending
    any n where
        n ∈ P
        c(n) = d(n)
        ∀m · m ∈ f⁻¹[{n}] ⇒ c(n) < 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** : $d(r) \leq c(r)$
- Now: proving perspective, lasso + p0 proves strengthening.

ascending: only local comparisons now!

Refining *descending*

- 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 show here non-root case.

```
Event (abstract−)descending
    any n where
        n ∈ P
        ∀m · m ∈ N ⇒ d(n) ≤ d(m)
    then
        d(n) := d(n) + 1
    end
```

```
Event (concrete−)descending
    any n where
        n ∈ P\{r}
        d(n) ≠ d(f(n))
    then
        d(n) := d(n) + 1
    end
```

✓ *Change (concrete) descending event to non-extended*
✓ *Update guards*
Guard strengthening needs to be proved.

Proving guard strengthening

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

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

We need some magic mushrooms to help the provers:

thm3_2 : $\forall n \cdot n \in P\backslash\{r\} \Rightarrow d(f(n)) \in d(n)..d(n)+1$
thm3_3 : $\forall n \cdot n \in P \Rightarrow d(r) \in d(n)..d(n)+1$

thm3_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 ∈ P
        ∀m · m ∈ P ⇒ d(n) ≤ d(m)
    then
        d(n) := d(n) + 1
    end
```

✓ *Rename descending to descending_nr*
✓ *Copy event "descending_r":*
Left click on circle left of name to select
`Ctrl-C` *to copy,* `Ctrl-V` *to paste*

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

```
Event (concrete−)descending
    refines
        descending
    when
        d(r) ≠ c(r)
    with
        n: n = r
    then
        d(r) := d(r) + 1
    end
```

✓ *Change name to "descending_r"*

- 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

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 : $\forall n \cdot n \in P\backslash\{r\} \Rightarrow d(f(n)) \in d(n)..d(n)+1$
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 P0 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 P0 to one goal, ML (any force) to the other.

Third refinement: invariants

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

```
ascending
  any n where
    n ∈ P
    c(n) = d(n)
    ∀m · ( m ∈ f⁻¹[{n}] ⇒ c(n) ≠ c(m) )
  then
    c(n) := c(n) + 1
  end
```

```
descending_1
  any n where
    n ∈ P \ {r}
    d(n) ≠ d(f(n))
  then
    d(n) := d(n) + 1
  end
```

```
descending_2
  when
    d(r) ≠ c(r)
  then
    d(r) := d(r) + 1
  end
```

Third refinement: events

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

```
Event descending_nr
  any n where
    n ∈ P\{r}
    d(n) ≠ d(f(n))
  then
    d(n) := d(n) + 1
  end
```

```
Event ascending
  any n where
    n ∈ P
    c(n) = d(n)
    ∀m · m ∈ f⁻¹[{n}] ⇒ c(n) < c(m)
  then
    c(n) := c(n) + 1
  end
```

Steps

1. Initial model: all nodes access to 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.

Observation

- 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!

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

- 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: $r, f, parity$

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

axm4_2: $parity(0) = 0$

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

✓ *Add parity and axioms to c2. Note: parity is a function!*
✓ *Need some clicking 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: $\forall n \,.\, (n \in P \;\Rightarrow\; p(n) = parity(c(n)))$

inv4_4: $\forall n \,.\, (n \in P \;\Rightarrow\; q(n) = parity(d(n)))$

✓ *Do it in m4. Note the gluing invariants! p and q really syntactic sugar.*

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

A note on the guard of ascending

Ascending's guard was:

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

A direct translation would be

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

- This would be wrong:
  - $p(n) < p(m)$ does not imply $c(n) < c(m)$.
  - If $p(n) < p(m)$, it could also happen $c(n) > c(m)$!
- However, we know that $|c(n) - c(m)| \leq 1$.
  - So, $c(n) < c(m) \equiv c(n) \neq c(m)$.
  - We could have used it throughout the model.
- Then, $p(n) \neq p(m)$ can (and should) be used.

Proving remaining POs (in `ascending`)

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

- Needs additional property

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

- We could make it axiom, but it can be proven as theorem (better!).
- Proving it is not difficult.

  WD: P0 takes care of it.

  THM: A couple of simple rewritings + distinction by cases work.

- $\Longleftrightarrow$: rewrite in two implications.
- par(x) = par(y) $\Rightarrow x = y$: ah with possible values of x.
- Prove ah with ML.
- Goal $y = y + 1$: do dc with par(y) $= 0$.
- P0 works for both branches.

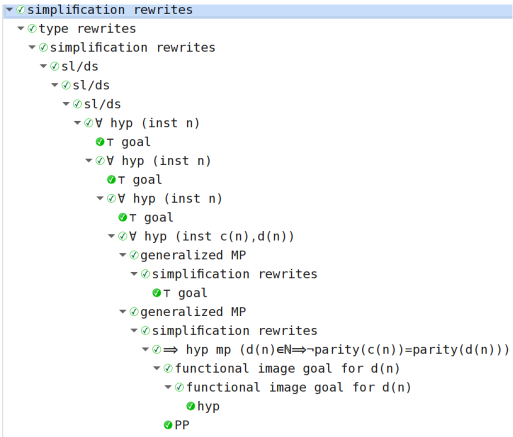
Proving remaining POs (in `ascending`)

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

- With theorem

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

- Instantiate with $c(n)$, $d(n)$.
- Instantiate defs. of $p(n)$, $q(n)$.
- Invoke P0.
- See recording linked from course web page.



Proving POs (in `ascending`)

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

Two paths that work:

1. Add two new THM:

$$\forall n \cdot n \in P \setminus \{r\} \quad \Rightarrow \quad c(n) \in c(f(n))..c(f(n)) + 1$$
$$\forall n \cdot n \in P \quad \Rightarrow \quad c(n) \in c(r)..c(r) + 1$$

   Then introduce the hypothesis $n = f(m)$ (which comes from $m \in f^{-1}[n]$) and use ML. See recording at course web.

2. Introduce $n = f(m)$. Work more by hand to deduce that $c(f(n)) \leq c(n) \leq c(f(n)) + 1$. Deduce that $p(m) \neq p(f(m))$. Deduce the relationship $parity(c(m)) = parity(c(f(m)) \Leftrightarrow c(m) = c(f(m))$. Launch a theorem prover. See recording.