A Ring Network Model

Jean-Raymond Abrial

September 2004
Purpose of this Lecture

- Learning more about modeling (in particular refinement)

- Learning some more modeling conventions

- Learning how to formalize an interesting structure: a ring

- Study a classical problem in distributed computing

- The example comes from the following paper:
- Each process is able to send messages to its right neighbor

- Each process is able to receive messages from its left neighbor
- Messages can be **buffered** in each process **before** being sent

- Messages can be **reordered** in each buffer
- After some (finite) time a **unique process becomes the leader**

- Constraint: each process executes the **same piece of code**
- Seems impossible

- Nothing makes one process different from the other

- The ring structure is homogeneous (no first, no last)

- The only difference between processes is their name

- But, it is not sufficient to make a difference between processes
- Giving more structure to names

- Names are natural numbers

- A special process is thus the one with the largest name

- How can a process know that it bears the largest name?
Process with Numbered Names
Initial Model

Constant: \( N, n \)

Variable: \( w \)

prp_1 : \( N \in F_1(N) \)

prp_2 : \( n \in N \)

inv_1 : \( w \in N \)

init \( \equiv \) begin
\[ \begin{align*}
    w &:= n \\
    \text{end}
\end{align*} \]

elect \( \equiv \) begin
\[ \begin{align*}
    w &:= \max(N) \\
    \text{end}
\end{align*} \]

- Exercise: Prove this initial model
<table>
<thead>
<tr>
<th>$\in$</th>
<th>set membership operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{N}$</td>
<td>set of Natural Numbers: ${0, 1, 2, 3, \ldots}$</td>
</tr>
<tr>
<td>$a \ldots b$</td>
<td>interval from $a$ to $b$: ${a, a + 1, \ldots, b}$</td>
</tr>
<tr>
<td>$S \rightarrow T$</td>
<td>set of total functions from $S$ to $T$</td>
</tr>
<tr>
<td>$S \mapsto T$</td>
<td>set of partial functions from $S$ to $T$</td>
</tr>
</tbody>
</table>
### Reminder of Conventions for Modeling (2)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>∪</td>
<td>set-theoretic union operator</td>
</tr>
<tr>
<td>↦</td>
<td>pair constructing operator</td>
</tr>
<tr>
<td>{...}</td>
<td>set defined in extension</td>
</tr>
<tr>
<td>∅</td>
<td>empty set</td>
</tr>
<tr>
<td>△</td>
<td>domain restriction operator</td>
</tr>
</tbody>
</table>
## More Conventions for Modeling

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_1(S)$</td>
<td>Non-empty set of finite subsets of $S$</td>
</tr>
<tr>
<td>$\mathbb{F}(S)$</td>
<td>Set of finite subsets of $S$</td>
</tr>
<tr>
<td>$\mathbb{P}_1(S)$</td>
<td>Non-empty set of subsets of $S$</td>
</tr>
<tr>
<td>$\mathbb{P}(S)$</td>
<td>Set of subsets of $S$</td>
</tr>
<tr>
<td>$\text{max}(S)$</td>
<td>Maximum of a non-empty finite set of numbers</td>
</tr>
</tbody>
</table>
Solution 1

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- Receiving a name (except its own), a process puts it in its buffer

- Processes also keep the names they receive

- When a process receives its own name then it tests for leadership

- Does it work? NO: because messages can be reordered in buffers
Solution 2

- Almost the same as solution 1

- Each process knows the number $n$ of different processes

- A process starts testing after receiving $n$ different names

- Does it work? **YES**, but is rather heavy

- Knowing the number of different processes is not always possible
Solution 3

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- A name is rejected if smaller than that of the receiving process

- Receiving a name (except its own), a process puts it in its buffer

- A process receiving its own name is the leader

- Does it work???
Basic Situation
Initial Situation
Accepting 4
Accepting 6
Rejecting 2
Accepting 6
Rejecting 1
Accepting 6
Accepting 6
Rejecting 3
Accepting 6
Accepting 5
Accepting 6. It is the **Winner**
- The \textit{nxt} function is a \textit{bijection}
- A bijection is a \textit{total function}
- The \textit{converse} of a bijection is also a \textit{total function}
- A bijection is not sufficient to define a single ring
- The interval between $a$ and $d$

\[ \text{itv}(a, d) = \{a, b, c, d\} \]
- The singleton interval

\[ itv(x, x) = \{ x \} \]
- The non-singleton interval

\[ itv(x, \text{nxt}(y)) = itv(x, y) \cup \{\text{nxt}(y)\} \]  

(if \( \text{nxt}(y) \neq x \))
- Constants: $nxt$, $itv$

\[
\begin{align*}
\text{prp}_3 : & \quad nxt \in N \rightarrow N \\
\text{prp}_4 : & \quad itv \in N \times N \rightarrow \mathbb{P}(N) \\
\text{prp}_5 : & \quad \forall x \cdot (x \in N \Rightarrow itv(x, x) = \{x\}) \\
\text{prp}_6 : & \quad \forall x, y \cdot \left( x \in N, y \in N, x \neq nxt(y) \Rightarrow itv(x, nxt(y)) = itv(x, y) \cup \{nxt(y)\} \right)
\end{align*}
\]
More Conventions for Modeling

<table>
<thead>
<tr>
<th>$S \mapsto T$</th>
<th>set of bijections from $S$ to $T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \times T$</td>
<td>Cartesian product of $S$ and $T$</td>
</tr>
</tbody>
</table>
- In a ring, the interval between \( \text{nxt}(x) \) and \( x \) is \( N \) (the entire set of nodes)
A Small Theory of Rings (2)

- Constants: $nxt$, $itv$

prp_{2} : \quad nxt \in N \rightarrow N

prp_{3} : \quad itv \in N \times N \rightarrow \mathbb{P}(N)

prp_{4} : \quad \forall x \cdot (x \in N \Rightarrow \text{itv}(x, x) = \{x\})

prp_{5} : \quad \forall x, y \cdot \left(\begin{array}{l}
x \in N \\
y \in N \\
x \neq nxt(y) \Rightarrow \\
\text{itv}(x, nxt(y)) = \text{itv}(x, y) \cup \{nxt(y)\}
\end{array}\right)

prp_{7} : \quad \forall x \cdot (x \in N \Rightarrow \text{itv}(nxt(x), x) = N)$
- Each name is at most in one position

- Each name is either rejected or accepted to the next position
- 6 is the maximum of the blue interval \{6, 3, 5, 2\}

- Because 6 has been accepted successfully from position 6 to position 2
The State

Variable: $pos$

\[
\begin{align*}
\text{inv}_2 & : \quad pos \in N \rightarrow N \\
\text{inv}_3 & : \quad \forall x \cdot (x \in \text{dom}(pos) \Rightarrow x = \max(itv(x, pos(x))))
\end{align*}
\]

- $pos$ yields the position of each node that has not yet been rejected

- $\text{inv}_3$ is the key invariant
# More Conventions for Modeling

<table>
<thead>
<tr>
<th><strong>dom</strong></th>
<th>domain of a function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ran</strong></td>
<td>range of a function</td>
</tr>
<tr>
<td><strong>≜</strong></td>
<td>domain restriction operator</td>
</tr>
<tr>
<td><strong>≜</strong></td>
<td>domain subtraction operator</td>
</tr>
<tr>
<td><strong>id(S)</strong></td>
<td>identity function built on the set $S$</td>
</tr>
</tbody>
</table>
- Several possible events (accept or reject) are enabled

- We must have a new convention for such events
Events (Reminder and Extension)

- An event is made of two parts: the guard and the action

- The guard explains when the event can occur
  - Made of several conditions

- The action explains how the variables are modified
  - Made of several simple or NON-DETERMINISTIC assignments

- Convention

\[
\text{when } < \text{guard} > \text{ then } < \text{action} > \text{ end}
\]
Non-deterministic Assignment

- Example (reject)

```
any < fresh_variables > where
< conditions >
then
< simple_assignments >
end
```

```
any x where
x ∈ dom(pos)
x < nxt(pos(x))
then
pos := \{x\} ⊆ pos
end
```
The Complete Event

- Explication follows

\[
\text{reject} \; \equiv \; \begin{align*}
\text{when} \\
\exists x \cdot \left( x \in \text{dom}(\text{pos}) \; \land \; x < \text{nxt}(\text{pos}(x)) \right) \\
\text{then} \\
\text{any } x \text{ where} \\
\; \; x \in \text{dom}(\text{pos}) \\
\; \; x < \text{nxt}(\text{pos}(x)) \\
\text{then} \\
\text{pos} := \{x\} \trianglelefteq \text{pos} \\
\text{end} \\
\text{end}
\end{align*}
\]

rewritten simply as

\[
\text{reject} \; \equiv \; \begin{align*}
\text{any } x \text{ where} \\
\; \; x \in \text{dom}(\text{pos}) \\
\; \; x < \text{nxt}(\text{pos}(x)) \\
\text{then} \\
\text{pos} := \{x\} \trianglelefteq \text{pos} \\
\text{end}
\end{align*}
\]
First Refinement

init \equiv
begin
w \leftarrow n
pos \leftarrow \text{id}(N)
end

elect \equiv
any \ x \ where
x \in \text{dom}(pos)
x = \text{nxt}(pos(x))
then
w \leftarrow x
end

accept \equiv
any \ x \ where
x \in \text{dom}(pos)
nxt(pos(x)) < x
then
pos(x) \leftarrow \text{nxt}(pos(x))
end

reject \equiv
any \ x \ where
x \in \text{dom}(pos)
x < \text{nxt}(pos(x))
then
pos :\{x\} \triangleleft pos
end
$f(x) := E$ stands for $f := \{x\} \leftarrow f \cup \{x \mapsto E\}$

\[ \{x\} \leftarrow f \cup \{x \mapsto E\} \quad \text{also written} \quad f \leftarrow \{x \mapsto E\} \]

- $f$ overwritten by $\{x \mapsto E\}$
Further Studies

- Feasibility

- Before-after predicates for non-deterministic assignments

- Generalization of invariant preservation statement

- Generalization of refinement
- An event of the following form must be feasible

\[
\text{when } G(v) \\
\text{then} \\
\text{any } x \text{ where } C(x) \\
\text{then} \\
v := E(x, v) \\
\text{end} \\
\text{end}
\]

Statement to be proved

\[ G(v) \Rightarrow \exists x \cdot C(x) \]
Special Case

Such an event is automatically feasible

\[
\text{reject} \equiv \\
\text{when} \\
\exists x \cdot \left( \begin{array}{l}
\forall x \cdot (x \in \text{dom}(pos)) \\
x < \text{nxt}(pos(x))
\end{array} \right) \\
\text{then} \\
\text{any } x \text{ where} \\
x \in \text{dom}(pos) \\
x < \text{nxt}(pos(x)) \\
\text{then} \\
pos := \{x\} \triangleleft pos \\
\text{end}
\]

rewritten simply as

\[
\text{reject} \equiv \\
\text{any } x \text{ where} \\
x \in \text{dom}(pos) \\
x < \text{nxt}(pos(x)) \\
\text{then} \\
pos := \{x\} \triangleleft pos \\
\text{end}
\]
Before-After Predicates (Reminder)

- Assignments are substitutions

- We shall transform them into before-after predicates

- Given constants $c$, variables $v$, and an assignment of the form

  $$v := E(c, v)$$

- It can be mechanically transformed into the predicate

  $$v' = E(c, v)$$
B-A Predicate for Non-deterministic Assignment

- Non-deterministic assignment

\[
\text{any } x \text{ where } C(x) \text{ then } v := E(x, v) \text{ end}
\]

- Corresponding Before-after predicate

\[
\exists x \cdot \left( C(x) \land v' = E(x, v) \right)
\]
Invariant Preservation Statement (Reminder)

- Given constants $c$, properties $P(c)$, variables $v$, and invariant $I(c, v)$

- Given an event of the form

$$\text{when } G(c, v) \text{ then } v' = E(c, v) \text{ end}$$

- We have to prove

\[
\begin{align*}
P(c) \\
I(c, v) \\
G(c, v) \\
v' = E(c, v) \\
\Rightarrow \\
I(c, v')
\end{align*}
\]
Invariant Preservation Statement (Generalization)

- Given constants $c$, properties $P(c)$, variables $v$, and invariant $I(c, v)$

- Given an event of the form

\[
\text{when } G(c, v) \text{ then } Q(c, v, v') \text{ end}
\]

- We have to prove

\[
\begin{align*}
P(c) \\
I(c, v) \\
G(c, v) \\
Q(c, v, v') \\
\Rightarrow \\
I(c, v')
\end{align*}
\]
Simplification

when
  \[ G(c, v) \]
then
  \[ \exists x \cdot \left( \begin{array}{l}
  C(x, c, v) \\
  v' = E(x, c, v)
\end{array} \right) \]
end

Could be generated by a tool

\[
P(c) \\
I(c, v) \\
G(c, v) \\
\exists x \cdot \left( \begin{array}{l}
  C(x, c, v) \\
  v' = E(x, c, v)
\end{array} \right) \\
\Rightarrow \\
I(c, v')
\]

simplifies to
(if \( x \) is fresh)

\[
P(c) \\
I(c, v) \\
G(c, v) \\
C(x, c, v) \\
\Rightarrow \\
I(c, E(x, c, v))
\]
Correct Refinement Proof: Generalization

- Given constants $c$ and properties $P(c)$
- Given an abstraction with variables $v$ and invariant $I(c, v)$
- Given a refinement with variables $w$ and abstraction pred. $J(c, v, w)$
- Given an abstract event and refined event of the forms

```
when
  \[ G(c, v) \]
then
  \[ Q(c, v, v') \]
end

```

```
when
  \[ H(c, w) \]
then
  \[ R(c, w, w') \]
end
```
State and Event Refinement (Deterministic Case)

$I(v)$  \quad $G(c,v)$  \quad Abstract Event  \quad $I(v')$

$v$  \quad $v'=E(c,v)$

$J(c,v,w)$

$J(c,v',w')$

$w$  \quad $w'=F(c,w)$

$H(c,w)$
State and Event Reflt. (Non-deterministic Case)

I(v) \rightarrow G(c,v) \\
\rightarrow \text{Abstract Event} \\
J(c,v,w) \\
\rightarrow \text{Concrete Event} \\
J(c,v',w') \\
R(c,w,w') \\
Q(c,v,v')
- One has to prove:

\[
\begin{align*}
P(c) \\
I(c, v) \\
J(c, v, w) \\
H(c, w) \\
R(c, w, w') \\
\Rightarrow \\
G(c, v) \\
\exists v'. \left( \begin{array}{cc} 
Q(c, v, v') \\
J(c, v', w') 
\end{array} \right)
\end{align*}
\]

- Could be generated by a tool
Correct Refinement Proof: Special Case (1)

- Given constants $c$ and properties $P(c)$
- Given an abstraction with variables $v$ and invariant $I(c, v)$
- Given a refinement with variables $w$ and abstraction pred. $J(c, v, w)$
- Given an abstract event and refined event of the forms

```
  when
  G(c, v)
  then
    any x where
    A(c, x, v)
    then
    v := E(x, c, v)
  end
  end

  when
  H(c, w)
  then
    any y where
    B(c, y, w)
    then
    w := F(y, c, w)
  end
  end
```
With the Before-after Predicates

when $G(c, v)$
then
$\exists x \cdot \left( A(c, x, v) \land v' = E(x, c, v) \right)$
end

when $H(c, w)$
then
$\exists y \cdot \left( B(c, y, w) \land w' = F(y, c, w) \right)$
end
- One has to prove the following (could be generated by a tool):

\[
P(c) \quad I(c, v) \quad J(c, v, w) \quad H(c, w) \\
\exists y \cdot \left( \begin{array}{c}
B(c, y, w) \\
w' = F(y, c, w)
\end{array} \right) \\
\Rightarrow \\
G(c, v) \\
\exists v' \cdot \left( \begin{array}{c}
\exists x \cdot \left( A(c, x, v) \\
v' = E(x, c, v)
\end{array} \right) \\
J(c, v', w')
\right)
\]
Correct Refinement Proof: Providing a “witness”

- If one provides a witness $W(c, w, y)$ for $x$

\[
\begin{align*}
&P(c) \\
&I(c, v) \\
&J(c, v, w) \\
&H(c, w) \\
&B(c, y, w) \\
\Rightarrow \\
&G(c, v) \\
&\exists x \cdot \left( \begin{array}{c}
A(c, x, v) \\
J(c, E(x, c, v),) \\
F(y, c, w))
\end{array} \right)
\end{align*}
\]
Correct Refinement Proof: Special Case (2)

- Given constants $c$ and properties $P(c)$
- Given an abstraction with variables $v$ and invariant $I(c, v)$
- Given a refinement with variables $w$ and abstraction pred. $J(c, v, w)$
- Given an abstract event and refined event of the forms

```
when $G(c, v)$ then
  $v := E(x, c, v)$
end
```

```
when $H(c, w)$ then
  any $y$ where $B(c, y, w)$ then
    $w := F(y, c, w)$
  end
end
```
With the Before-after Predicates

when
\[ G(c, v) \]
then
\[ v' = E(x, c, v) \]
end

when
\[ H(c, w) \]
then
\[ \exists y \cdot \left( B(c, y, w) \wedge w' = F(y, c, w) \right) \]
end
Correct Refinement Proof

- One has to prove the following (could be generated by a tool):

\[
\begin{align*}
P(c) \\
I(c, v) \\
J(c, v, w) \\
H(c, w) \\
\exists y \cdot \begin{pmatrix} B(c, y, w) \\
w' = F(y, c, w) \end{pmatrix} \\
\Rightarrow \\
G(c, v) \\
\exists v'. \begin{pmatrix} v' = E(x, c, v) \\
J(c, v', w') \end{pmatrix}
\end{align*}
\]
The Events (Reminder)

Init \( \equiv \)

\[
\text{begin} \\
w := n \\
pos := \text{id}(N) \\
\text{end}
\]

Elect \( \equiv \)

\[
\text{any } x \hspace{1em} \text{where} \\
x \in \text{dom}(pos) \\
x = \text{nxt}(pos(x)) \\
\text{then} \\
w := x \\
\text{end}
\]

Accept \( \equiv \)

\[
\text{any } x \hspace{1em} \text{where} \\
x \in \text{dom}(pos) \\
nxt(pos(x)) < x \\
\text{then} \\
pos(x) := \text{nxt}(pos(x)) \\
\text{end}
\]

Reject \( \equiv \)

\[
\text{any } x \hspace{1em} \text{where} \\
x \in \text{dom}(pos) \\
x < \text{nxt}(pos(x)) \\
\text{then} \\
pos := \{x\} \triangleleft pos \\
\text{end}
\]
Refinement Proof for event elect

(abs\_elect) \equiv
begin
  w' = \max(N)
end

(ref\_elect) \equiv
when
  \exists x \cdot \left( x \in \text{dom}(pos_1) \wedge x = \text{nxt}(pos_1(x)) \right)
then
  \exists x \cdot \left( x \in \text{dom}(pos_1) \wedge x = \text{nxt}(pos_1(x)) \wedge w'_1 = x \right)
end
Refinement Proof for event elect (cont’d)

\[ \forall x \cdot \left( x \in N \Rightarrow \left( itv(nxt(x), x) = N \right) \right) \]

\[ pos \in N \mapsto N \]

\[ \forall x \cdot \left( x \in \text{dom}(pos) \Rightarrow \left( x = \max(itv(x, pos(x))) \right) \right) \]

\[ x \in \text{dom}(pos_1) \]

\[ pos = pos_1 \]

\[ x = \text{nxt}(pos_1(x)) \]

\[ \Rightarrow \]

\[ \max(N) = x \]

pos_1 \in N \mapsto N

\[ \forall x \cdot \left( x \in N \Rightarrow \left( itv(nxt(x), x) = N \right) \right) \]

x = \max(itv(x, pos_1(x)))

x \in \text{dom}(pos_1)

x = \text{nxt}(pos_1(x))

\[ \Rightarrow \]

\[ \max(N) = x \]

- Applying equality pos = pos_1

- Instantiating the second universal quantification with x
Refinement Proof for event elect (cont’d)

\[ pos_1 \in N \rightarrow N \]
\[ \forall x \cdot \left( \begin{array}{c} x \in N \\ itv(nxt(x), x) = N \end{array} \right) \]
\[ x = \max(itv(x, pos_1(x))) \]
\[ x \in \text{dom}(pos_1) \]
\[ x = nxt(pos_1(x)) \]
\[ \Rightarrow \]
\[ \max(N) = x \]

- Applying equalities
- Instantiating universal quantification with \( pos_1(x) \)
- One has to prove \( pos_1(x) \in N \)
- This proof could be done by a tool
Exercises

- Transform events accept and reject

- Determine the before-after predicates for accept and reject

- State the refinement statements for accept and reject

- Prove these refinement statements

- State and prove the additional proof of refinement
  (refinement does not stop earlier than abstraction)
Some Ideas for a Refinement

- We introduce two buffers in and out at each node

- We remove the variable \textit{pos}

- We introduce a new event \textit{send}
Basic Situation
Initial Situation
Sending 4
Sending 6
Accepting 4
Sending 2
Accepting 6
Rejecting 2
Sending 6
Sending 3
Rejecting 1
Exercises

- Introduce *in* and *out* buffers

- Write the gluing invariant between *in*, *out* and *pos*

- Refine events *elect*, *accept*, and *reject*

- Introduce a new event *send*

- Prove your refinement
What we Have Learned in this Lecture

- A few more mathematical conventions
- The non-deterministic assignment
- More on refinement proofs
- How to formalize a ring
### Reminder of Conventions for Modeling (1)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \in )</td>
<td>set membership operator</td>
</tr>
<tr>
<td>( \mathbb{N} )</td>
<td>set of Natural Numbers: ( {0, 1, 2, 3, \ldots } )</td>
</tr>
<tr>
<td>( a \ldots b )</td>
<td>interval from ( a ) to ( b ): ( {a, a + 1, \ldots, b} )</td>
</tr>
<tr>
<td>( S \rightarrow T )</td>
<td>set of total functions from ( S ) to ( T )</td>
</tr>
<tr>
<td>( S \mapsto T )</td>
<td>set of partial functions from ( S ) to ( T )</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>( \cup )</td>
<td>set-theoretic union operator</td>
</tr>
<tr>
<td>( \mapsto )</td>
<td>pair constructing operator</td>
</tr>
<tr>
<td>{ \ldots }</td>
<td>set defined in extension</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>empty set</td>
</tr>
<tr>
<td>( \triangleleft )</td>
<td>domain restriction operator</td>
</tr>
<tr>
<td>( F_1(S) )</td>
<td>Non-empty set of finite subsets of ( S )</td>
</tr>
<tr>
<td>( F(S) )</td>
<td>Set of finite subsets of ( S )</td>
</tr>
<tr>
<td>( P_1(S) )</td>
<td>Non-empty set of subsets of ( S )</td>
</tr>
<tr>
<td>( P(S) )</td>
<td>Set of subsets of ( S )</td>
</tr>
<tr>
<td>( \max(S) )</td>
<td>Maximum of a non-empty finite set of numbers</td>
</tr>
</tbody>
</table>
### Reminder of Conventions for Modeling (4)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \leftrightarrow T$</td>
<td>set of bijections from $S$ to $T$</td>
</tr>
<tr>
<td>$S \times T$</td>
<td>Cartesian product of $S$ and $T$</td>
</tr>
<tr>
<td>$f \triangleleft g$</td>
<td>overwriting operator for functions</td>
</tr>
<tr>
<td><strong>dom</strong></td>
<td>domain of a function</td>
</tr>
<tr>
<td><strong>ran</strong></td>
<td>range of a function</td>
</tr>
<tr>
<td><strong>Δ</strong></td>
<td>domain restriction operator</td>
</tr>
<tr>
<td><strong>(\succ)</strong></td>
<td>domain subtraction operator</td>
</tr>
<tr>
<td><strong>id(S)</strong></td>
<td>identity function built on the set (S)</td>
</tr>
</tbody>
</table>
Non-deterministic Assignment

any \( < \text{variable} > \) where
\( < \text{condition} > \)
\( \ldots \)
then
\( < \text{variable} > \):= \( < \text{expression} > \)
\( \ldots \)
end
Set: $N$ \hspace{2cm} Constants: $\text{nxt, itv}$

\[ \text{nxt} \in N \implies N \]

\[ \text{itv} \in N \times N \rightarrow \mathbb{P}(N) \]

\[ \forall x \cdot (x \in N \implies \text{itv}(x, x) = \{x\}) \]

\[ \forall x, y \cdot \begin{cases} x \in N \\ y \in N \\ x \neq \text{nxt}(y) \end{cases} \implies \begin{cases} \text{itv}(x, \text{nxt}(y)) = \text{itv}(x, y) \cup \{\text{nxt}(y)\} \end{cases} \]

\[ \forall x \cdot (x \in N \implies \text{itv}(\text{nxt}(x), x) = N) \]