A Formal (proved) Approach to Discrete System Development

Modeling

J-R. Abrial

September 2004
Purpose of the Course

- Giving some insights about Formal Methods
- Showing that Formal Methods can be made practical
- Illustrating Formal Methods with examples
What you Will Learn

- By the end of the course you should be more comfortable with:

  - **Modeling** (versus programming)

  - **Abstraction** and **Refinement**

  - Some **mathematical techniques** (for data structures)

  - The idea of **proving** (what to prove)
The lectures

- 1st: Introduction by means of a small illustrating example

- 2nd: Introduction (cont’d) and another example

- 3rd: More complex examples
About the Examples

- All coming from the distributed programming area

- Other areas could have been chosen:

  - Sequential programming

  - Concurrent programming

  - Electronic circuit development

  - Complete systems (involving Software and Equipment)
Purpose of the First Lecture

- Learning about modeling

- Learning various conventions for modeling

- Learning some techniques for modeling

- Study a small example illustrating the introduced concepts
Formal Methods: a Great Confusion

- What are they used for?

- When are they to be used?

- Is UML a formal method?

- Are they needed when doing OO programming?

- What is their definition?
What About Other Engineering Disciplines

- Some mature engineering disciplines:
  - Avionics,
  - Space,
  - Civil engineering,
  - Mechanical engineering,
  - Train systems,
  - Ship building.

- Are there any equivalent approaches to Formal Methods?

- Yes, BLUE PRINTS
What is a Blue Print?

- A certain representation of the future system

- It is not a mock-up? (although mock-ups can be very useful too)

- The basis is lacking (you cannot “drive” the blue print of a car)

- Allows to reason about the future system during its design

- Is it important? (according to professionals) YES
Reasoning about the Future System?

- Defining and calculating its **behavior** (what it does)

- Incorporating **constraints** (what it must not do)

- Defining **architecture**

- Based on some **underlying theories**
  - strength of materials,
  - flight mechanics,
  - gravitation,
  - etc.
Techniques of “Blue Printing”

- Using pre-defined conventions (often computerized these days)

- Conventions should help facilitate reasoning

- Adding details on more accurate versions

- Postponing choices by having some open options

- Decomposing one blue print into several

- Reusing “old” blue prints (with slight changes)
Befestigung bei guten Platzverhältnissen

(1:20)

Befestigung bei schlechten Platzverhältnissen

(1:20)
What About BEFORE the Blue Print

- Define main objectives of future system
- Define requirements
- Study feasibility
What About AFTER the Blue Print

- Construct the system

- Perform functional tests

- Study how constraints are obeyed

- Organize maintenance
Definitions of Formal Methods (subjective)

- Formal methods are techniques for building and studying blue prints.

ADAPTED TO OUR DISCIPLINE

Our discipline is: design of hardware and software SYSTEMS.

- Such blue prints are now called models.

- Reminder:
  - Models allow to reason about a FUTURE system.
  - The basis is lacking (hence you cannot “execute” a model).
Conventions for Model Writing and Reasoning

- Reminder (cont’d):
  - Using pre-defined conventions
  - Conventions should help facilitate reasoning (more to come)

- Consequence: Using ordinary discrete mathematical conventions:
  - Classical Logic (Predicate Calculus)
  - Basic Set Theory (sets, relations and functions)

- But drawings can be useful too (i.e. UML related conventions)
Examples of Systems we are Interested to Develop

- a “classical” piece of software
- an electronic circuit
- a file transfer protocol
- an airline booking system
- a PC operating system
- a nuclear plant controller
- a SmartCard electronic purse
- a launch vehicle flight controller
- a mechanical press controller
- etc.
Characterizing such Systems (general)

- They are made of *many parts*

- They interact with a possibly *hostile environment*

- They involve *several executing agents*

- They require a *high degree of correctness*

- There construction spreads over *several years*

- Their specifications are subjected to *many changes*
- These systems operate in a *discrete* fashion

- Their dynamical behavior can be *abstracted* by:
  - A succession of *steady states*
  - Intermixed with *sudden jumps*

- The possibility of state changes is *enormous*

- Usually such systems *never halt*

- They are called *discrete transition systems*
What is our Unifying Underlying Theory?

- Theory of discrete automaton (more to come)

- Automatons are characterized by:
  - a state
  - a number of transitions

- We want to study whether:
  - certain properties are maintained
  - certain goals can be achieved
Defining Automatons: State and Events

- We start from a number of given sets

- We define some constants in terms of these sets

- We define some variables in terms of these sets and constants

- We define some transitions on the variables (now called events)
An Example: File Transfer Protocol

- A file is to be transferred from a Sender to a Receiver

- On the Sender’s side the file is called $f$

- On the Receiver’s side the file is called $g$

- At the beginning of the protocol, $g$ is supposed to be empty

- At the end of the protocol, $g$ should be equal to $f$

- Studied in many places, in particular in the following book:
L. Lamport *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers* Addison-Wesley 1999
The Sender and the Receiver

Before

<table>
<thead>
<tr>
<th>SENDER</th>
<th>RECEIVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>g</td>
</tr>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

Channel

After

<table>
<thead>
<tr>
<th>SENDER</th>
<th>RECEIVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>g</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>
File transfer. The constant part of the state: $n$ and $f$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>c</td>
</tr>
</tbody>
</table>
File transfer. The variables of the state: $r$ and $g$
File transfer. The transition: receive
File transfer. The transition: receive

```
  -   -   -
   |   |   |
   f   f   f
   |   |   |
r  a  a  a
   |   |   |
   b  b  b
   |   |   |
n  c  c  c
```
File transfer. The transition: receive

<table>
<thead>
<tr>
<th></th>
<th>f</th>
<th>f</th>
<th>f</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>r</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>n</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td>n+1</td>
<td>r</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>g</th>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>g</td>
<td>b</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>g</td>
<td></td>
<td></td>
<td>g</td>
</tr>
</tbody>
</table>
File Transfer Protocol State (1)

- Set: \( D \)

- Constants: \( n, f \)

\[
\begin{align*}
\text{prp}_1 & : \quad n \in \mathbb{N} \\
\text{prp}_2 & : \quad f \in 1 \ldots n \rightarrow D
\end{align*}
\]

- Constants are characterized by their properties
## Conventions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>∈</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 .. 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>
</tbody>
</table>
File Transfer Protocol State (2)

- Variables: $r, g$

\[
\begin{align*}
\text{inv}_1 : & \quad r \in \mathbb{N} \\
\text{inv}_2 : & \quad g \in 1 \ldots n \mapsto D
\end{align*}
\]

- Variables are characterized by, so called, invariants

- Conventions

\[
\begin{array}{|c|c|}
\hline
S \mapsto T & \text{set of partial functions from } S \text{ to } T \\
\hline
\end{array}
\]
Events

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

- The guard explains when the event can occur
  - It is made of several conditions

- The action explains how the variables are modified
  - It is made of several simple assignments

- Convention

\[
\text{when } < \text{guard} > \quad \text{then } < \text{action} > \quad \text{end}
\]
File Transfer Protocol Event: receive

- An event is a transition which we can observe

\[
\text{receive } \equiv \\
\text{when } \\
\quad r \leq n \\
\text{then} \\
\quad g := g \cup \{r \mapsto f(r)\} \\
\quad r := r + 1 \\
\text{end}
\]

- The guard is made of one condition: \( r \leq n \)

- The action is made of two assignments: \[
\begin{cases} 
  g := g \cup \{r \mapsto f(r)\} \\
  r := r + 1 
\end{cases}
\]
## Conventions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
Initialization

- There exists a special initializing event

\[
\text{init} \equiv \\
\begin{align*}
\text{begin} \\
g &:= \emptyset \\
r &:= 1 \\
\text{end}
\end{align*}
\]

- Conventions

\[
\begin{array}{|c|c|}
\hline
\emptyset & \text{empty set} \\
\hline
\end{array}
\]
Summary of the File Transfer Protocol Model

Set: $D$  
Constants: $n, f$  
Variables: $r, g$

\begin{align*}
\text{prp}_1 : & \quad n \in \mathbb{N} \\
\text{prp}_2 : & \quad f \in 1 \ldots n \rightarrow D
\end{align*}

\begin{align*}
\text{inv}_1 : & \quad r \in \mathbb{N} \\
\text{inv}_2 : & \quad g \in 1 \ldots n \rightarrow D
\end{align*}

\begin{align*}
\text{init} & \equiv \\
\begin{aligned}
\text{begin} & \\
g & := \emptyset \\
r & := 1 \\
\text{end}
\end{aligned}
\end{align*}

\begin{align*}
\text{receive} & \equiv \\
\begin{aligned}
\text{when} & \quad r \leq n \\
\text{then} & \\
g & := g \cup \{r \mapsto f(r)\} \\
r & := r + 1 \\
\text{end}
\end{aligned}
\end{align*}
Structure of a Model

- List of Sets (identifiers)

- List of Constants (identifiers)

- List of Properties (predicates built on sets and constants)

- List of Variables (identifiers)

- List of Invariants (predicates built on sets, constants, and variables)

- List of Events (next slide)
Shape of an Event

\[
\begin{align*}
\langle \text{name} \rangle & \equiv \\
\text{when} & \\
\langle \text{guard} \rangle & \\
\ldots & \\
\text{then} & \\
\langle \text{variable} \rangle & \equiv \langle \text{expression} \rangle \\
\ldots & \\
\text{end} & 
\end{align*}
\]

- \( \langle \text{guards} \rangle \) are predicates built on sets, constants, and variables

- \( \langle \text{expressions} \rangle \) are terms built on sets, constants, and variables
Something we would like to Prove

When \( r = n + 1 \) then the protocol is finished: \( g = f \)
Reasoning: Theorems

\[ \text{thm}_1 : \quad r = n + 1 \implies g = f \]

- Theorems should be **logically deduced** from invariants

- But we **cannot prove** \text{thm}_1 from our properties and invariants

\[
\begin{align*}
\text{prp}_1 : & \quad n \in \mathbb{N} \\
\text{prp}_2 : & \quad f \in 1 \ldots n \rightarrow D \\
\text{inv}_1 : & \quad r \in \mathbb{N} \\
\text{inv}_2 : & \quad g \in 1 \ldots n \rightarrow D
\end{align*}
\]
Reasoning: Invariants

- We add the following two invariants (more to come here):

\[ \text{inv}_3 : \quad r \in 1 \ldots n + 1 \]
\[ \text{inv}_4 : \quad g = (1 \ldots r - 1) \triangle f \]

- Conventions

| $\triangle$ | domain restriction operator |
Observing Invariants

\[ inv_3 : \quad r \in 1 \ldots n + 1 \]

\[ inv_4 : \quad g = (1 \ldots r - 1) \triangleleft f \]
- Our theorem is then easy to prove

\[
\begin{align*}
\text{prp}_1 : & \quad n \in \mathbb{N} \\
\text{prp}_2 : & \quad f \in 1..n \to D \\
\text{inv}_3 : & \quad r \in 1..n + 1 \\
\text{inv}_4 : & \quad g = (1..r - 1) \triangle f \\
\text{thm}_1 : & \quad r = n + 1 \implies g = f
\end{align*}
\]
Another Summary of the Protocol Model

Set: \( D \)  
Constants: \( n, f \)  
Variables: \( r, g \)

\begin{align*}
\text{prp}_1 & : \quad n \in \mathbb{N} \\
\text{prp}_2 & : \quad f \in 1..n \rightarrow D
\end{align*}

\begin{align*}
\text{inv}_3 & : \quad r \in 1..n + 1 \\
\text{inv}_4 & : \quad g = 1..r - 1 \triangleleft f
\end{align*}

\begin{align*}
\text{init} & \equiv \\
& \begin{align*}
\text{begin} \\
g & := \emptyset \\
r & := 1 \\
\text{end}
\end{align*}
\end{align*}

\begin{align*}
\text{receive} & \equiv \\
& \begin{align*}
\text{when} \\
r & \leq n \\
\text{then} \\
g & := g \cup \{r \mapsto f(r)\} \\
r & := r + 1 \\
\text{end}
\end{align*}
\end{align*}
More Modeling Conventions

- The assignment:

\[ g := g \cup \{ r \mapsto f(r) \} \]

- can be re-written

\[ g(r) := f(r) \]

- Because \( r \) is not in the domain of \( g \) (more to come) since we have

\[ \text{inv}_4 : g = 1 \ldots r - 1 \triangleleft f \]
A Better Summary of the Protocol Model

Set: $D$  
Constants: $n, f$  
Variables: $r, g$

prp\_1: $n \in \mathbb{N}$
prp\_2: $f \in 1..n \rightarrow D$

inv\_3: $r \in 1..n + 1$
inv\_4: $g = 1..r - 1 \triangleleft f$

init $\equiv$
begin
  $g := \emptyset$
  $r := 1$
end

receive $\equiv$
when $r \leq n$
then
  $g(r) := f(r)$
  $r := r + 1$
end
What is to be Proved

- Our task is not finished

- So far we have just observed that the invariants are maintained

- Observing is not enough

- We want to make precise what we have to prove
Transforming Assignments: Before-After Predicates

- 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 (by a tool) into the predicate

\[ v' = E(c, v) \]
Example: Event receive

These two forms of event receive are equivalent (more to come)

receive \equiv
\begin{align*}
\text{when} & \quad r \leq n \\
\text{then} & \quad g(r) := f(r) \\
\phantom{\text{then}} & \quad r := r + 1 \\
\text{end} & \quad
\end{align*}

receive \equiv
\begin{align*}
\text{when} & \quad r \leq n \\
\text{then} & \quad g' = g \cup \{ r \mapsto f(r) \} \\
\phantom{\text{then}} & \quad r' = r + 1 \\
\text{end} & \quad
\end{align*}

- We shall use the left form when writing models
- We shall use the right form when proving them
Invariant Preservation Statement

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

- Given an event of the form

  \[
  \textbf{when } G(c, v) \textbf{ then } v' = E(c, v) \textbf{ end}
  \]

- We have to prove

  \[
  P(c) \\
  I(c, v) \\
  G(c, v) \\
  v' = E(c, v) \\
  \Rightarrow \\
  I(c, v')
  \]
Simplification

\[ P(c) \]
\[ I(c, v) \]
\[ G(c, v) \]
\[ v' = E(c, v) \]
\[ \Rightarrow \]
\[ I(c, v') \]

which simplifies to

\[ P(c) \]
\[ I(c, v) \]
\[ G(c, v) \]
\[ \Rightarrow \]
\[ I(c, E(c, v)) \]

- This statement can be generated by a tool
Statement to be proved

\[ \text{prp}_1 : \quad n \in \mathbb{N} \]
\[ \quad \text{prp}_2 : \quad f \in 1..n \rightarrow D \]

\[ \text{inv}_3 : \quad r \in 1..n+1 \]
\[ \quad \text{inv}_4 : \quad g = 1..r-1 \triangle f \]

receive \( \equiv \)
\[
\begin{align*}
\text{when} & \quad r \leq n \\
\text{then} & \quad g' = g \cup \{r \mapsto f(r)\} \\
\text{end} & \quad r' = r + 1
\end{align*}
\]

\[
\begin{align*}
n & \in \mathbb{N} \\
f & \in 1..n \rightarrow D \\
r & \in 1..n+1 \\
g & = 1..r-1 \triangle f \\
r & \leq n \\
\Rightarrow & \quad g \cup \{r \mapsto f(r)\} = 1..r+1-1 \triangle f \\
r + 1 & \in 1..n + 1
\end{align*}
\]
Semi-formal Proof

\[
\begin{align*}
\n &\in \mathbb{N} \\
\in &\ 1..n \rightarrow D \\
\in &\ 1..n + 1 \\
\less{} &\ n \\
\equiv &\ \{r \mapsto f(r)\} = \\
\ 1..r+1-1 \less{} &\ f \\
\sum &\ 1..n+1 \\
\sum &\ 1..n+1
\end{align*}
\]
Invariant Establishment Statement

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

- Given an initialization of the form

\[
\begin{align*}
\text{begin} & \quad v' = E(c) \quad \text{end}
\end{align*}
\]

- We have to prove

\[
\begin{align*}
P(c) & \\
v' = E(c) & \\
\Rightarrow & \\
I(c, v') &
\end{align*}
\]

which simplifies to

\[
\begin{align*}
P(c) & \\
\Rightarrow & \\
I(c, E(c)) &
\end{align*}
\]
Statement to be proved

prp_1: \( n \in \mathbb{N} \)

prp_2: \( f \in 1..n \rightarrow D \)

inv_3: \( r \in 1..n+1 \)

inv_4: \( g = 1..r-1 \triangleleft f \)

init \( \equiv \)

\[
\begin{align*}
\text{begin} \\
g' &= \emptyset \\
r' &= 1 \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
n &\in \mathbb{N} \\
f &\in 1..n \rightarrow D \\
\Rightarrow \\
\emptyset &= 1..1 - 1 \triangleleft f \\
1 &\in 1..n+1
\end{align*}
\]
This Model of the “Protocol” is not Satisfactory

Set: $D$  
Constants: $n, f$  
Variables: $r, g$

<table>
<thead>
<tr>
<th>prp_1: $n \in \mathbb{N}$</th>
<th>inv_3: $r \in 1..n + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>prp_2: $f \in 1..n \rightarrow D$</td>
<td>inv_4: $g = 1..r - 1 &lt; f$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>init $\triangleq$ begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g ::= \emptyset$</td>
</tr>
<tr>
<td>$r ::= 1$</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>receive $\triangleq$ when</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r \leq n$</td>
</tr>
<tr>
<td>then</td>
</tr>
<tr>
<td>$g(r) ::= f(r)$</td>
</tr>
<tr>
<td>$r ::= r + 1$</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

The Receiver accesses the original file

We want to distribute the file transfer
Techniques of “Blue Printing” (Reminder)

- Adding details on different more accurate versions
- Postponing choices by having some open options
- Decomposing one blue print into several
- Reusing “old” blue prints (with slight changes)
A More Accurate Version (1)
A More Accurate Version (2)
Initial Situation

```
   f
 s a
 | b
 n c
   d
```

- r
Receive
Send

```
\[
\begin{array}{cccc}
\text{s} & \text{a} & \text{s} & \text{a} \\
\text{b} & \text{b} & \text{b} & \\
\text{n} & \text{c} & \text{n} & \text{c} \\
\end{array}
\]
\]
```

```
\[
\begin{array}{cccc}
\text{d} & \text{d} & \text{d} & \text{d} \\
\text{a} & \text{a} & \text{a} & \text{b} \\
\text{r} & \text{r} & \text{r} & \\
\end{array}
\]
\]
```

```
\[
\begin{array}{cccc}
\text{g} & \text{g} & \text{g} & \\
\text{r} & \text{r} & \text{r} & \\
\end{array}
\]
```

```
\[
\begin{array}{cccc}
\end{array}
\]
```
Receive
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>s</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
First Refinement of the Protocol Model

Set: $D$ \hspace{1cm} Constants: $n, f, e$ \hspace{1cm} Variables: $r, g, s, d$

$$\text{prp}_3 : \quad e \in D$$
$$\text{inv}_5 : \quad d \in D$$
$$\text{inv}_6 : \quad s \in 1..n + 1$$

**init** $\equiv$
begin
$g := \emptyset$
$r := 1$
$s := 1$
$d := e$
end

**send** $\equiv$
when
$s = r$
$s \leq n$
then
$d := f(s)$
$s := s + 1$
end

**receive** $\equiv$
when
$s \neq r$
then
$g(r) := d$
$r := r + 1$
end
- We have **added a constant** $e$

- We have **added two new variables** $s$ and $d$

- We have **modified the events** receive and init

- We have **added a new event** send

- Note that the **assignments** in send and receive are **not complete**
Analysis of Refinement

- Refining an abstract event

- The problem of event completion

- The problem of distinct abstract and refined spaces

- “Refining” a new event

- Refining initialization

- Additional requirement for refinement
Refinement: the Situation

- We have constants $c$
- We have an abstract state with variables $v$
- We have a refined state with variables $w$ DISTINCT from $v$
- We have an abstract event and a refined event of the forms

\[
\begin{align*}
\text{when} & \quad G(c, v) \\
\text{then} & \quad v' = E(c, v) \\
\text{end}
\end{align*}
\quad \text{when} \quad H(c, w) \\
\text{then} & \quad w' = F(c, w) \\
\text{end}
\]

- WHAT IS TO BE PROVED?
- We have some properties $P(c)$ on the constants $c$

- We have some invariants $I(c, v)$ on the abstract variables $v$

- We also have some gluing invariants $J(c, v, w)$ linking concrete variables $w$ to abstract variables $v$
State and Event Refinement

Abstract Event

\[ I(v) \quad G(c,v) \quad \text{Concrete Event} \]

\[ v \quad v' = E(c,v) \quad v = F(c,w) \quad w' = F(c,w) \]

\[ J(c,v,w) \quad J(c,v',w') \]

\[ H(c,w) \]
Correct Refinement Proof

- One has to prove (more in next lecture):

\[
\begin{align*}
P(c) \\
I(c, v) \\
J(c, v, w) \\
H(c, w) \\
w' &= F(c, w) \\
v' &= E(c, v) \\
\implies G(c, v) \\
J(c, v', w')
\end{align*}
\]

which simplifies to

\[
\begin{align*}
P(c) \\
I(c, v) \\
J(c, v, w) \\
H(c, w) \\
\implies G(c, v) \\
J(c, E(c, v), F(c, w))
\end{align*}
\]

- This statement can be generated by a tool
More on Before-After Predicates: Completion

- Given constants $c$, and distinct variables $x$ and $y$, the assignment

$$x := E(c, x, y)$$

- can be transformed into the before-after predicate

$$x' = E(c, x)$$

$$y' = y$$

- Variables $x$ and $y$ are the only variables of our model

- This completion can be done mechanically by a tool
Completion Example: Refined Event \texttt{receive}

These two forms of event \texttt{receive} are equivalent

\[
\text{receive} \quad \overset{\equiv}{=} \quad
\begin{align*}
\text{when} & \quad r \leq n \\
\text{then} & \quad g(r) := d \\
& \quad r := r + 1 \\
\text{end}
\end{align*}
\]

\[
\text{receive} \quad \overset{\equiv}{=} \quad
\begin{align*}
\text{when} & \quad r \leq n \\
\text{then} & \quad g' = g \cup \{r \mapsto d\} \\
& \quad r' = r + 1 \\
& \quad s' = s \\
& \quad d' = d \\
\text{end}
\end{align*}
\]

- Notice the \textbf{difference} between the two
Completion Example: New Event $\textit{send}$

These two forms of event $\textit{send}$ are equivalent

\[
\begin{align*}
\text{send} & \equiv \\
& \quad \text{when} \\
& \quad \quad s = r \\
& \quad \quad s \leq n \\
& \quad \text{then} \\
& \quad \quad d := f(s) \\
& \quad \quad s := s + 1 \\
& \quad \text{end}
\end{align*}
\]

\[
\begin{align*}
\text{send} & \equiv \\
& \quad \text{when} \\
& \quad \quad s = r \\
& \quad \quad s \leq n \\
& \quad \text{then} \\
& \quad \quad g' = g \\
& \quad \quad r' = r \\
& \quad \quad d' = f(s) \\
& \quad \quad s' = s + 1 \\
& \quad \text{end}
\end{align*}
\]

- Notice the \textit{difference} between the two
Abstract and Concrete Events receive

\[(\text{abstract})\text{receive} \equiv \begin{align*}
\text{when} & \quad r \leq n \\
\text{then} & \quad g(r) := f(r) \\
r & := r + 1 \\
\text{end}
\end{align*}\]

\[(\text{refined})\text{receive} \equiv \begin{align*}
\text{when} & \quad s \neq r \\
\text{then} & \quad g(r) := d \\
r & := r + 1 \\
\text{end}
\end{align*}\]

- These events deal with the same variables \(g\) and \(r\)
- This cannot be the case
- Abstract and concrete states must have distinct variables
- Solution: change of variables and adding a trivial gluing invariant
- This can be done by a tool
First Refinement (Transformations made by a tool)

\[ \text{prp}_3 : \quad e \in D \]
\[ \text{inv}_5 : \quad d \in D \]
\[ \text{inv}_6 : \quad s \in 1..n + 1 \]

\[ \text{inv}_7 : \quad r_1 = r \]
\[ \text{inv}_8 : \quad g_1 = g \]

\[ \text{init} \equiv \begin{array}{l}
\text{begin} \\
g_1 := \emptyset \\
r_1 := 1 \\
s := 1 \\
d := e \\
\text{end}
\end{array} \]

\[ \text{send} \equiv \begin{array}{l}
\text{when} \\
s = r_1 \\
s \leq n \\
\text{then} \\
d := f(s) \\
s := s + 1 \\
\text{end}
\end{array} \]

\[ \text{receive} \equiv \begin{array}{l}
\text{when} \\
s \neq r_1 \\
\text{then} \\
g_1(r_1) := d \\
r_1 := r_1 + 1 \\
\text{end}
\end{array} \]
To be proved for Refinement of Event $\text{receive} (1)$

(abstract)receive $\equiv$

when
\[
    r \leq n
\]
then
\[
    g' = g \cup \{r \mapsto f(r)\}
    \quad r' = r + 1
\]
end

(refined)receive $\iff$

when
\[
    s \neq r_1
\]
then
\[
    g'_1 = g_1 \cup \{r \mapsto d\}
    \quad r'_1 = r_1 + 1
    \quad s' = s
    \quad d' = d
\]
end

inv_6 : $s \in 1..n + 1$

inv_7 : $r_1 = r$

inv_8 : $g_1 = g$
To be proved for Refinement of Event receive (2)

\[\begin{align*}
P(c) \\
I(c, v) \\
J(c, v, w) \\
H(c, w) \\
w' &= F(c, w) \\
v' &= E(c, v) \\
\Rightarrow \\
G(c, v) \\
J(c, v', w')
\end{align*}\]

\[\begin{align*}
s &\in 1 \ldots n + 1 \\
r_1 &= r \\
g_1 &= g \\
s &\neq r_1 \\
g'_1 &= g_1 \cup \{r \mapsto d\} \\
r'_1 &= r_1 + 1 \\
g' &= g \cup \{r \mapsto f(r)\} \\
r' &= r + 1 \\
s' &= s \\
d' &= d \\
\Rightarrow \\
r &\leq n \\
r'_1 &= r' \\
g'_1 &= g'
\end{align*}\]
Informal Proof: Applying Equalities

\[ s \in 1 \ldots n + 1 \]
\[ r_1 = r \]
\[ g_1 = g \]
\[ s \neq r_1 \]
\[ g'_1 = g_1 \cup \{ r \mapsto d \} \]
\[ r'_1 = r_1 + 1 \]
\[ g' = g \cup \{ r \mapsto f(r) \} \]
\[ r' = r + 1 \]
\[ s' = s \]
\[ d' = d \]

\[ \Rightarrow \]
\[ r \leq n \]
\[ r' = r + 1 = r + 1 \]
\[ g \cup \{ r \mapsto d \} = g \cup \{ r \mapsto f(r) \} \]

\[ s \in 1 \ldots n + 1 \]
\[ s \neq r \]

\[ \Rightarrow \]
\[ r \leq n \]
\[ r + 1 = r + 1 \]
\[ g \cup \{ r \mapsto d \} = g \cup \{ r \mapsto f(r) \} \]
What remains to be proved

\[
\begin{align*}
    s & \in 1 \ldots n + 1 \\
    s & \neq r \\
    \Rightarrow \\
    r & \leq n \\
    r + 1 & = r + 1 \\
    g \cup \{r \mapsto d\} & = g \cup \{r \mapsto f(r)\}
\end{align*}
\]

This cannot be proved. But the following invariants are suggested

inv\_9 : \quad s \in r \ldots r + 1

inv\_10 : \quad s \neq r \Rightarrow d = f(r)
Observing the invariant \( s \in r .. r + 1 \)
Observing the invariant \( s \neq r \Rightarrow d = f(r) \)
- After applying equalities again, we obtain

\[
\begin{align*}
    s & \in 1 \ldots n + 1 \\
    s & \in r \ldots r + 1 \\
    s \neq r & \implies d = f(r) \\
    s \neq r & \\
    \implies r \leq n \\
    s & \in r + 1 \ldots r + 1 + 1 \\
    s \neq r + 1 & \implies d = f(r + 1) \\
    d & = f(r)
\end{align*}
\]
The Final Step

\[
\begin{align*}
    s & \in 1 \ldots n + 1 \\
    d &= f(r) \\
    s &= r + 1 \\
    \Rightarrow & \\
    r &\leq n \\
    s &\in r + 1 \ldots r + 2 \\
    s &\neq r + 1 \ \Rightarrow \ d = f(r + 1) \\
    d &= f(r) \\
\end{align*}
\]

\[
\begin{align*}
    r + 1 &\in 1 \ldots n + 1 \\
    \Rightarrow & \\
    r &\leq n \\
    r + 1 &\in r + 1 \ldots r + 2 \\
\end{align*}
\]

All this can be done by a tool
Adding New Events in a Refinement

- Each new event must (in general) refine skip
- New events must not take control for ever
- For this, they all decrease a variant $V(c, w)$
- For a new event of the form

```
when
  $S(c, w)$
then
  $w' = K(c, w)$
end
```

One has to prove

$$
\begin{align*}
P(c) \\
I(c, v) \\
J(c, v, w) \\
S(c, w) \\
w' = K(c, w) \\
\Rightarrow \\
J(v, w') \\
0 \leq V(c, w') \\
V(c, w') < V(c, w)
\end{align*}
$$
To be proved for Refinement of Event send

send \equiv

\textbf{when}
\begin{align*}
s &= r_1 \\
s &\leq n
\end{align*}
\textbf{then}
\begin{align*}
  d' &= f(s) \\
  s' &= s + 1 \\
  r_1' &= r_1 \\
  g_1' &= g_1
\end{align*}
\textbf{end}

The variant $V$ is $n + 1 - s$

\begin{align*}
s &\in 1 \ldots n + 1 \\
s &\in r_1 \ldots r_1 + 1 \\
s \neq r_1 \Rightarrow d = f(r_1) \\
r_1 &= r \\
g_1 &= g \\
s &= r_1 \\
s &\leq n \\
d' &= f(s) \\
s' &= s + 1 \\
r_1' &= r_1 \\
g_1' &= g_1 \\
\Rightarrow \\
s' &\in 1 \ldots n + 1 \\
s' &\in r_1' \ldots r_1' + 1 \\
s' \neq r_1' \Rightarrow d' = f(r_1') \\
0 &\leq n + 1 - s' \\
n + 1 - s' &< n + 1 - s
Informal Proof: Applying Equalities

\[ s \in 1 .. n + 1 \]
\[ s \in r_1 .. r_1 + 1 \]
\[ s \not= r_1 \Rightarrow d = f(r_1) \]
\[ r_1 = r \]
\[ g_1 = g \]
\[ s = r_1 \]
\[ s \leq n \]
\[ d' = f(s) \]
\[ s' = s + 1 \]
\[ r'_1 = r_1 \]
\[ g'_1 = g_1 \]
\[ r \in 1 .. n + 1 \]
\[ r \in r .. r + 1 \]
\[ r \not= r \Rightarrow d = f(r) \]
\[ r \leq n \]
\[ r + 1 \in 1 .. n + 1 \]
\[ r + 1 \in r .. r + 1 \]
\[ r + 1 \not= r \Rightarrow f(r) = f(r) \]
\[ 0 \leq n + 1 - (r + 1) \]
\[ n + 1 - (r + 1) < n + 1 - r \]
Informal Proof: Applying Simple Reasoning

\[ r \in 1 \ldots n + 1 \]
\[ r \in r \ldots r + 1 \]
\[ r \neq r \Rightarrow d = f(r) \]
\[ r \leq n \]
\[ \Rightarrow \]
\[ r + 1 \in 1 \ldots n + 1 \]
\[ r + 1 \in r \ldots r + 1 \]
\[ r + 1 \neq r \Rightarrow f(r) = f(r) \]
\[ 0 \leq n + 1 - (r + 1) \]
\[ n + 1 - (r + 1) < n + 1 - r \]

- Such a proof could be made by a tool
Refinement of Initialization

- We have constants \( c \) and properties \( P(c) \)
- We have an abstract state with variables \( v \)
- We have a refined state with variables \( w \) DISTINCT from \( v \)
- We have a gluing invariant \( J(c, v, w) \)
- We have an abstract init and a refined init of the forms

\[
\begin{align*}
\text{begin} \\
v' &= E(c) \\
\text{end}
\end{align*}
\]
\[
\begin{align*}
\text{begin} \\
w' &= F(c) \\
\text{end}
\end{align*}
\]

To prove

\[
\frac{P(c) \quad v' = E(c) \quad w' = F(c)}{J(c, v', w')}
\]
To be Proved for \( \text{init} \)

\[
\text{init} \equiv \\
\text{begin} \\
\quad g' = \emptyset \\
\quad r' = 1 \\
\text{end}
\]

\[
e \in D \\
g' = \emptyset \\
r' = 1 \\
g_1' = \emptyset \\
r_1' = 1 \\
s' = 1 \\
d' = e \\
\Rightarrow \\
d' \in D \\
s' \in 1 \ldots n + 1 \\
s' \in r' \ldots r' + 1 \\
s' \not= r' \Rightarrow d' = f(r') \\
r' = r_1' \\
g' = g_1'
\]
Informal Proof: Applying Equalities

\[ e \in D \]
\[ g' = \emptyset \]
\[ r' = 1 \]
\[ g'_1 = \emptyset \]
\[ r'_1 = 1 \]
\[ s' = 1 \]
\[ d' = e \]
\[ \Rightarrow \]
\[ d' \in D \]
\[ s' \in 1 \ldots n + 1 \]
\[ s' \in r' \ldots r' + 1 \]
\[ s' \neq r' \Rightarrow d' = f(r') \]
\[ r' = r'_1 \]
\[ g' = g'_1 \]

Such a proof could be made by a tool.
Additional Requirement for Refinement

- A system stops when all the guards of its events are false

- When a refinement stops, its abstraction must have stopped

- In other words: no early stop for the refinement

- We have thus to prove one of the following

  all refined guards false
  \[\Rightarrow\]
  all abstract guards false

  some abstract guards true
  \[\Rightarrow\]
  some refined guards true
To be proved

\[
\begin{align*}
    r \leq n & \implies s \neq r \vee (s = r \land s \leq n) \\
\end{align*}
\]

simplified to

\[
\begin{align*}
    r \leq n & \implies s \leq n \\
\end{align*}
\]
Some Ideas for a Second Refinement?

Set: $D$  
Constants: $n, f, e$  
Variables: $r, g, s, d$

$prp_3 : e \in D$
$inv_5 : d \in D$
$inv_6 : s \in 1 .. n + 1$

$inv_9 : s \in r .. r + 1$
$inv_10 : s \neq r \Rightarrow d = f(r)$

$init \triangleq$
$\begin{align*}
  \text{begin} & \\
  g & := \emptyset \\
  r & := 1 \\
  s & := 1 \\
  d & := e \\
  \text{end}
\end{align*}$

$send \triangleq$
$\begin{align*}
  \text{when} & \\
  s & = r \\
  s & \leq n \\
  \text{then} & \\
  d & := f(s) \\
  s & := s + 1 \\
  \text{end}
\end{align*}$

$receive \triangleq$
$\begin{align*}
  \text{when} & \\
  s & \neq r \\
  \text{then} & \\
  g(r) & := d \\
  r & := r + 1 \\
  \text{end}
\end{align*}$
Third Version (1)
A Small Theory of Parities

**prp.4**: \( pty \in \mathbb{N} \rightarrow \{0, 1\} \)

**prp.5**: \( pty(0) = 0 \)

**prp.6**: \( \forall n \cdot (n \in \mathbb{N} \Rightarrow pty(n + 1) = 1 - pty(n)) \)

**thm.2**: \( \forall x, y \cdot \begin{cases} x \in \mathbb{N} \\ y \in \mathbb{N} \\ x \in y \ldots y + 1 \\ pty(x) = pty(y) \\ \Rightarrow \\ x = y \end{cases} \)
The 2nd Refinement. Adding two Variables $p$ and $q$

inv\_11 : $p = pty(r)$

inv\_12 : $q = pty(s)$

Exercises:
- Determine the before-after pred.
- State the theorems to prove
- Prove them

init $\equiv$
begin
    $g := \emptyset$
    $r := 1$
    $s := 1$
    $d := e$
    $p := 1$
    $q := 1$
end

send $\equiv$
begin
    when
        $q = p$
        $s \leq n$
        then
            $d := f(s)$
            $s := s + 1$
            $q := 1 - q$
        end
end

receive $\equiv$
begin
    when
        $q \neq p$
        then
            $g(r) := d$
            $r := r + 1$
            $p := 1 - p$
        end
end
Adding an Initial Model

**prp_1**: \( n \in \mathbb{N} \)

**prp_2**: \( f \in 1..n \rightarrow D \)

**inv_0**: \( g \in \mathbb{N} \rightarrow D \)

**init**: \( \equiv \begin{align*}
\text{begin} \\
g &: \in \mathbb{N} \rightarrow D \\
\text{end}
\end{align*} \)

**terminate**: \( \equiv \begin{align*}
\text{begin} \\
g &: := f \\
\text{end}
\end{align*} \)

This model is necessary to prove that the protocol terminates.
First Refinement

inv_3 : \( r \in 1..n+1 \)

inv_4 : \( g = 1..r-1 \triangleleft f \)

\[
\begin{align*}
\text{init} & \equiv \\
\begin{aligned}
\text{begin} \\
g & := \emptyset \\
r & := 1
\end{aligned} \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\text{receive} & \equiv \\
\begin{aligned}
\text{when} \\
r \leq n
\end{aligned} \\
\text{then} \\
g & := g \cup \{r \mapsto f(r)\} \\
r & := r + 1
\end{align*}
\]

\[
\begin{align*}
\text{terminate} & \equiv \\
\begin{aligned}
\text{when} \\
r = n + 1
\end{aligned} \\
\text{then} \\
\text{skip}
\end{align*}
\]
Second Refinement

init $\equiv$

\begin{align*}
\text{begin} \\
g, r, s &:= \emptyset, 1, 1 \\
d &\in D \\
\text{end}
\end{align*}

inv_5 : s \in 1 .. n + 1

inv_6 : s \in r .. r + 1

inv_7 : s \neq r \Rightarrow d = f(r)

send $\equiv$

\begin{align*}
\text{when} \\
s &= r \\
s &\leq n \\
\text{then} \\
d &:= f(s) \\
s &:= s + 1 \\
\text{end}
\end{align*}

receive $\equiv$

\begin{align*}
\text{when} \\
s &\neq r \\
\text{then} \\
g &:= g \cup \{r \mapsto d\} \\
r &:= r + 1 \\
\text{end}
\end{align*}

terminate $\equiv$

\begin{align*}
\text{when} \\
r &= n + 1 \\
\text{then} \\
\text{skip} \\
\text{end}
\end{align*}
Third Refinement

inv_10: \( p = pty(r) \)
inv_11: \( q = pty(s) \)

terminate \( \triangleq \)
\[
\text{when } r = n + 1 \text{ then skip end}
\]

init \( \triangleq \)
begin
\[
\begin{align*}
g & :\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\![298x279]send \triangleq
begin
\[
\begin{align*}
q &= p \\
\text{when} & \ s \leq n \\
\text{then} & \ d : = f(s) \\
\text{then} & \ s : = s + 1 \\
\text{then} & \ q : = 1 - q \\
\end{align*}
end
\]

receive \( \triangleq \)
when
\[
\begin{align*}
q & \neq p \\
\text{then} & \ g : = g \cup \{r \mapsto d\} \\
r & \leftarrow r + 1 \\
p & \leftarrow 1 - p \\
\end{align*}
end
What we Have Learned

- Some mathematical conventions

- How to write a model (only a little more on next lecture)

- What kind of things we have to prove

- How the proof can help finding invariants

- Many things can be done by tools

- A small theory of parities
| $\in$ | set membership operator |
| $\mathbb{N}$ | set of Natural Numbers: \{0, 1, 2, 3, \ldots\} |
| $a \ldots b$ | interval from $a$ to $b$: \{a, a + 1, \ldots, b\} |
| $S \rightarrow T$ | set of total functions from $S$ to $T$ |
| $S \mapsto T$ | set of partial functions from $S$ to $T$ |
## 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>ø</td>
<td>empty set</td>
</tr>
<tr>
<td>△</td>
<td>domain restriction operator</td>
</tr>
</tbody>
</table>
Structure of a Model

- List of **Sets** (identifiers)

- List of **Constants** (identifiers)

- List of **Properties** (predicates built on sets and constants)

- List of **Variables** (identifiers)

- List of **Invariants** (predicates built on sets, constants, and variables)

- List of **Events** (next slide)
Shape of an Event

\[
\begin{align*}
\langle \text{name} \rangle & \quad \equiv \\
\text{when} & \\
\langle \text{guard} \rangle & \\
\ldots & \\
\text{then} & \\
\langle \text{variable} \rangle & :\!\!\!= \langle \text{expression} \rangle \\
\ldots & \\
\text{end} & 
\end{align*}
\]

- \langle guards \rangle are predicates built on sets, constants, and variables

- \langle expressions \rangle are terms built on sets, constants, and variables
A Small Theory of Parities

Constant: \( pty \)

\[
pty \in \mathbb{N} \rightarrow \{0, 1\}
\]

\[
pty(0) = 0
\]

\[
\forall n \cdot (n \in \mathbb{N} \Rightarrow pty(n + 1) = 1 - pty(n))
\]

\[
\forall x, y \cdot \begin{cases} 
x \in \mathbb{N} \\
y \in \mathbb{N} \\
x \in y \ldots y + 1 \\
pty(x) = pty(y) \\
\Rightarrow \\
x = y
\end{cases}
\]