

A Ring Network Model

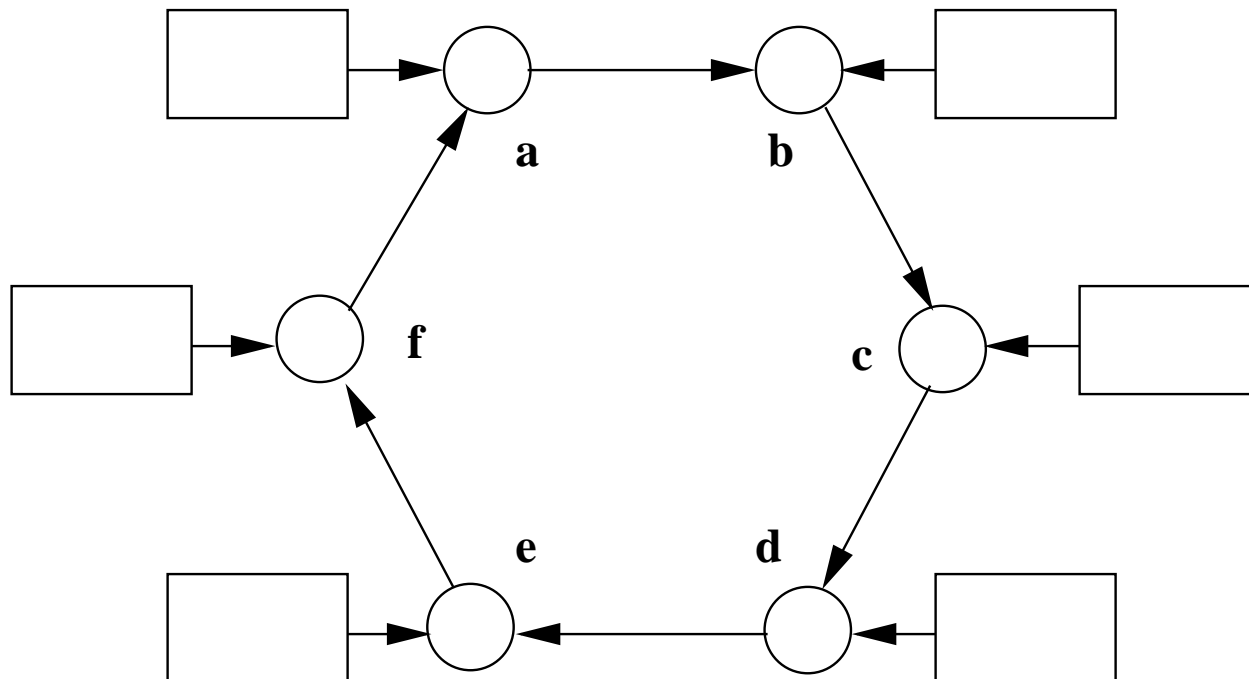
Jean-Raymond Abrial

September 2004

Purpose of this Lecture

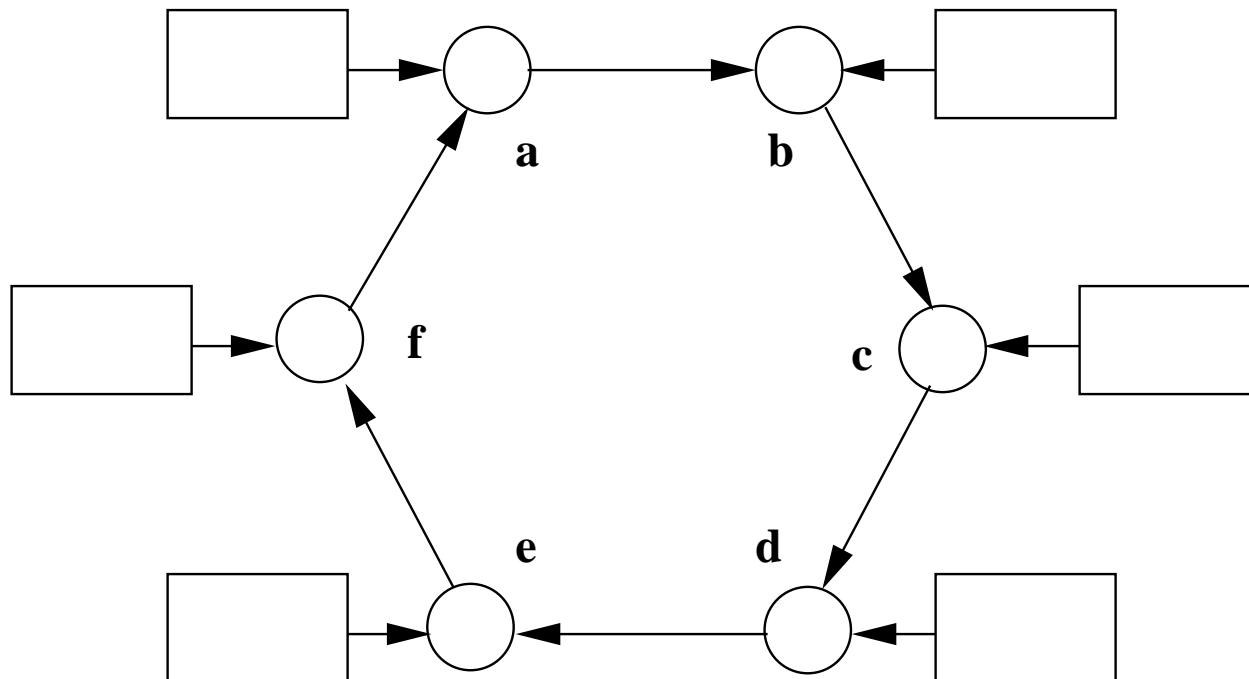
- Learning **more about modeling** (in particular **refinement**)
- Learning some more **modeling conventions**
- Learning how to **formalize** an interesting structure: a **ring**
- Study a classical problem in **distributed computing**
- The example comes from the following paper:
G. Le Lann. *Distributed systems - towards a formal approach*.
In B Gilchrist, editor *Information Processing 77* North-Holland 1977.

A Ring of Processes



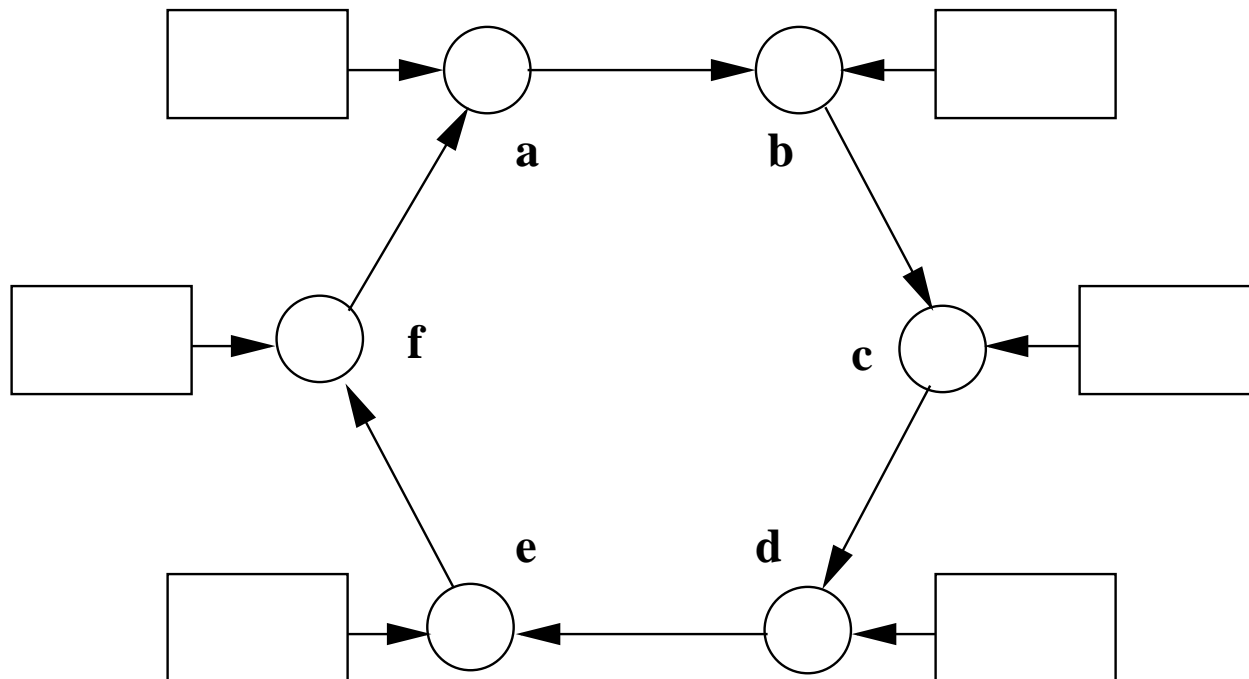
- Each process is able to **send messages** to its **right neighbor**
- Each process is able to **receive messages** from its **left neighbor**

A Ring of Processes with Buffers



- Messages can be **buffered** in each process **before being sent**
- Messages can be **reordered** in each buffer

Purpose of Distributed Program



- After some (finite) time a **unique process** becomes **the leader**
- **Constraint**: each process executes the **same piece of code**

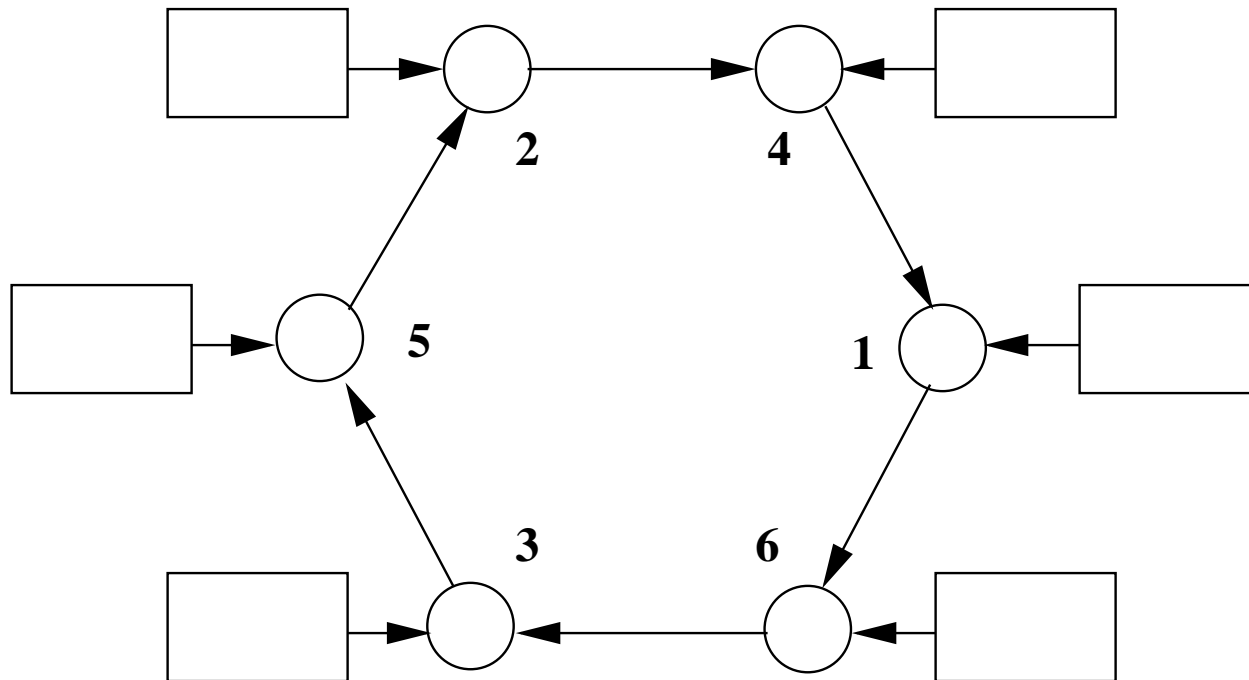
Discussion (1)

- 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

Discussion (2)

- 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

prp_1 : $N \in \mathbb{F}_1(\mathbb{N})$
prp_2 : $n \in N$

Variable: w

inv_1 : $w \in N$

init $\hat{=}$
begin
 $w := n$
end

elect $\hat{=}$
begin
 $w := \max(N)$
end

- Exercise: Prove this initial model

Reminder of Conventions for Modeling (1)

\in	set membership operator
\mathbb{N}	set of Natural Numbers: $\{0, 1, 2, 3, \dots\}$
$a .. b$	interval from a to b : $\{a, a + 1, \dots, b\}$
$S \rightarrow T$	set of total functions from S to T
$S \leftrightarrow T$	set of partial functions from S to T

Reminder of Conventions for Modeling (2)

\cup	set-theoretic union operator
\mapsto	pair constructing operator
$\{\dots\}$	set defined in extension
\emptyset	empty set
\triangleleft	domain restriction operator

More Conventions for Modeling

$\mathbb{F}_1(S)$	Non-empty set of finite subsets of S
$\mathbb{F}(S)$	Set of finite subsets of S
$\mathbb{P}_1(S)$	Non-empty set of subsets of S
$\mathbb{P}(S)$	Set of subsets of S
$\max(S)$	Maximum of a non-empty finite set of numbers

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**

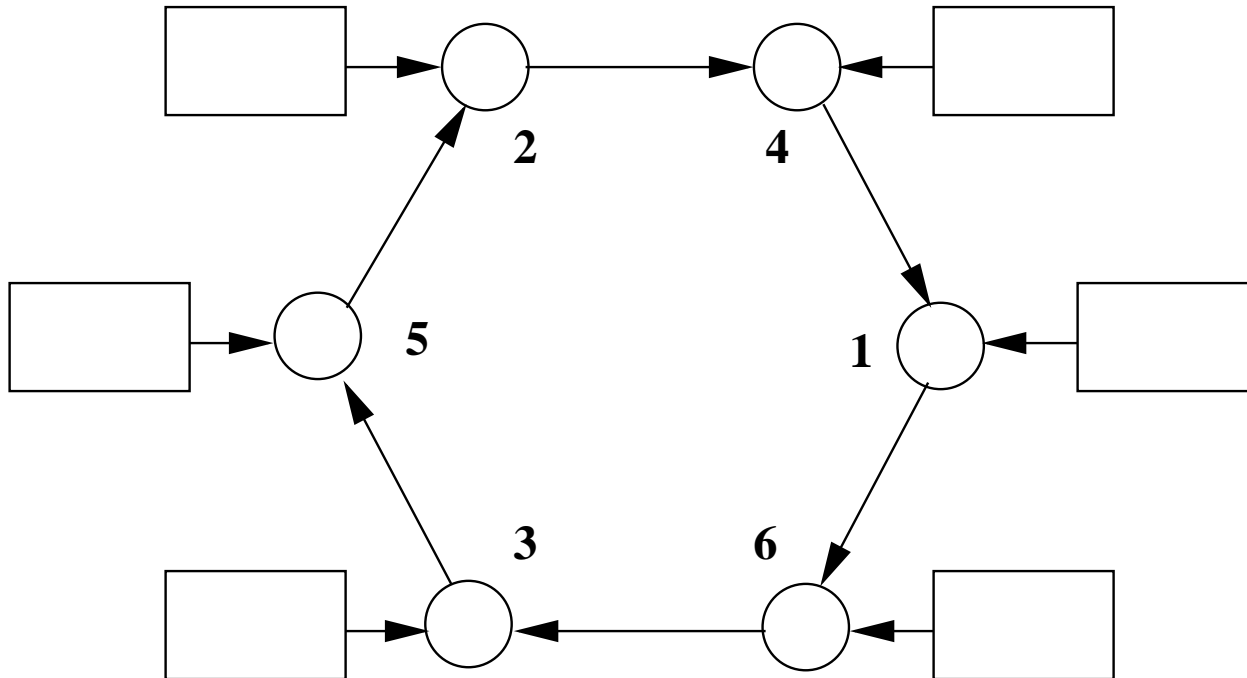
Solution 2

- 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**

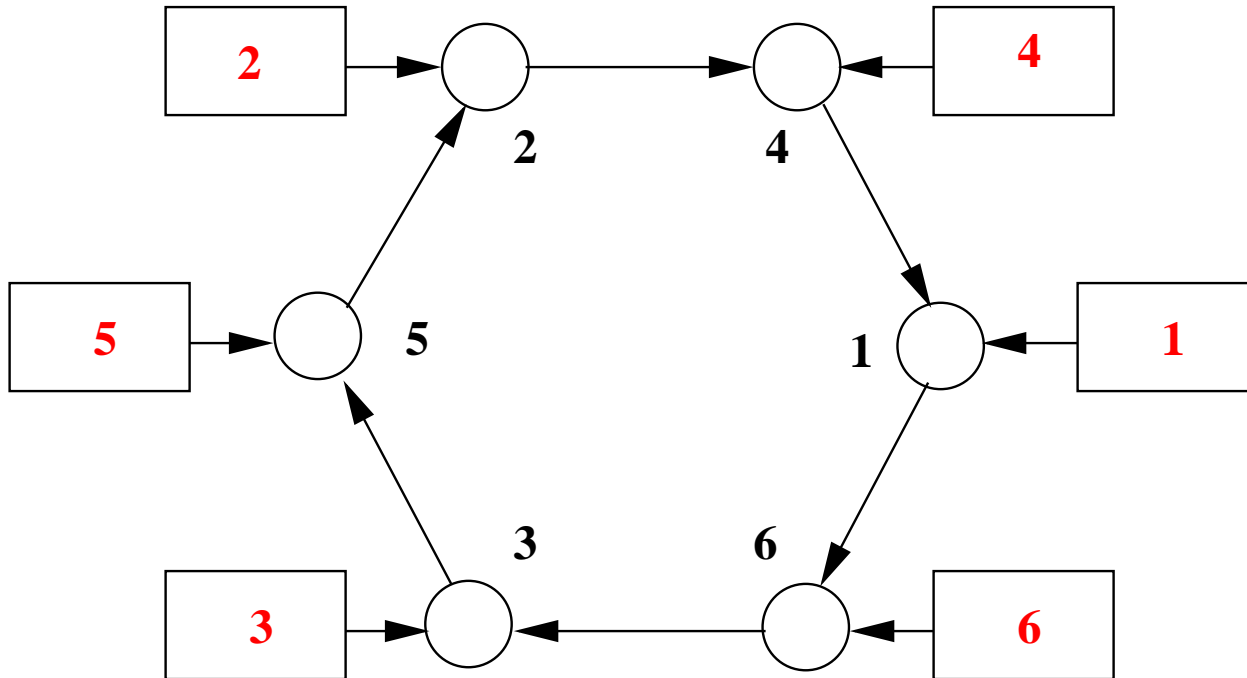
Solution 3

- **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???

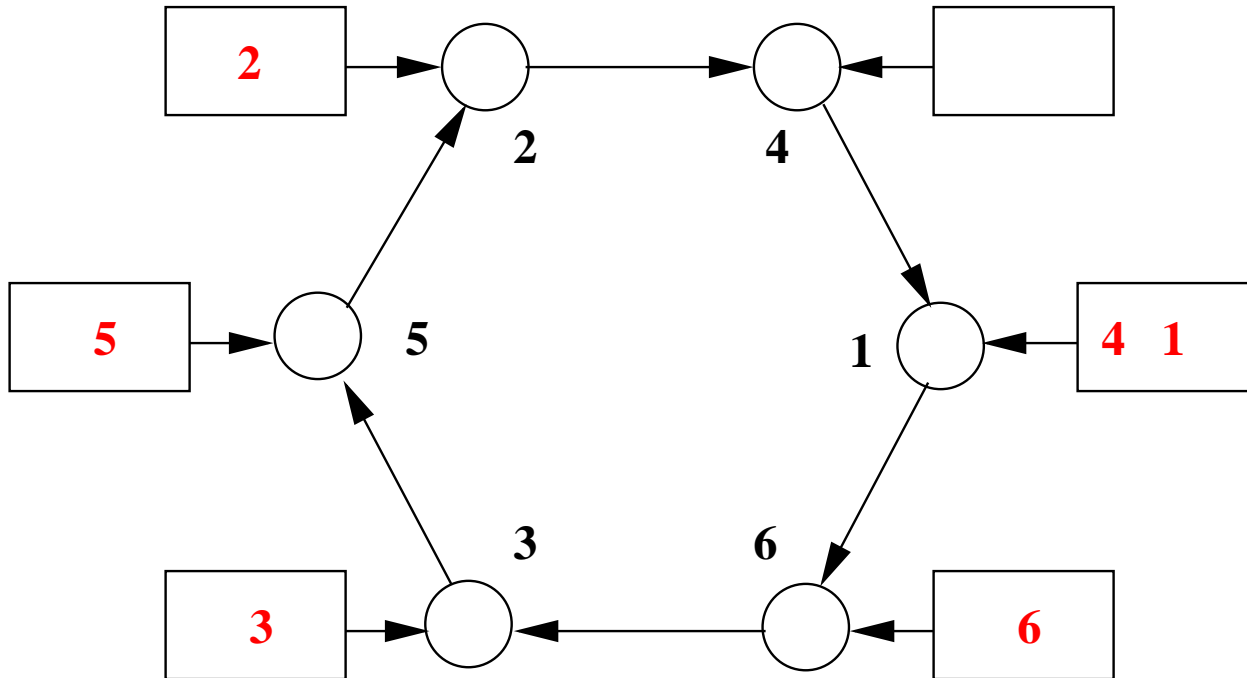
Basic Situation



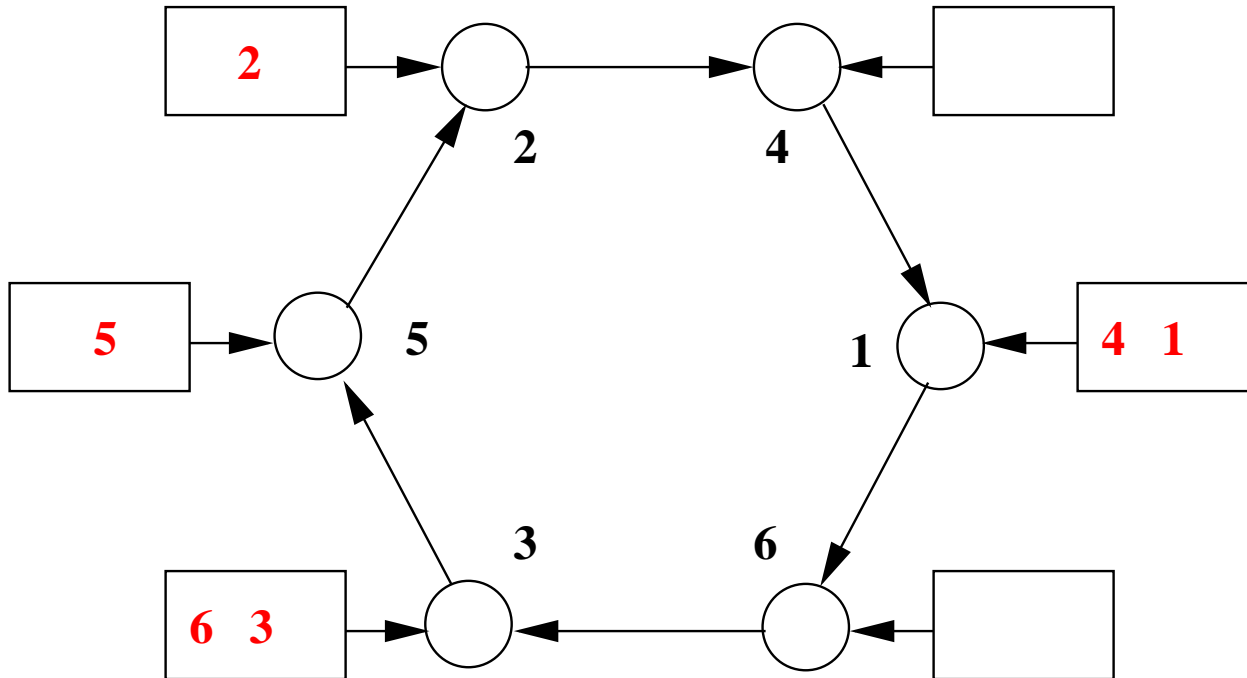
Initial Situation



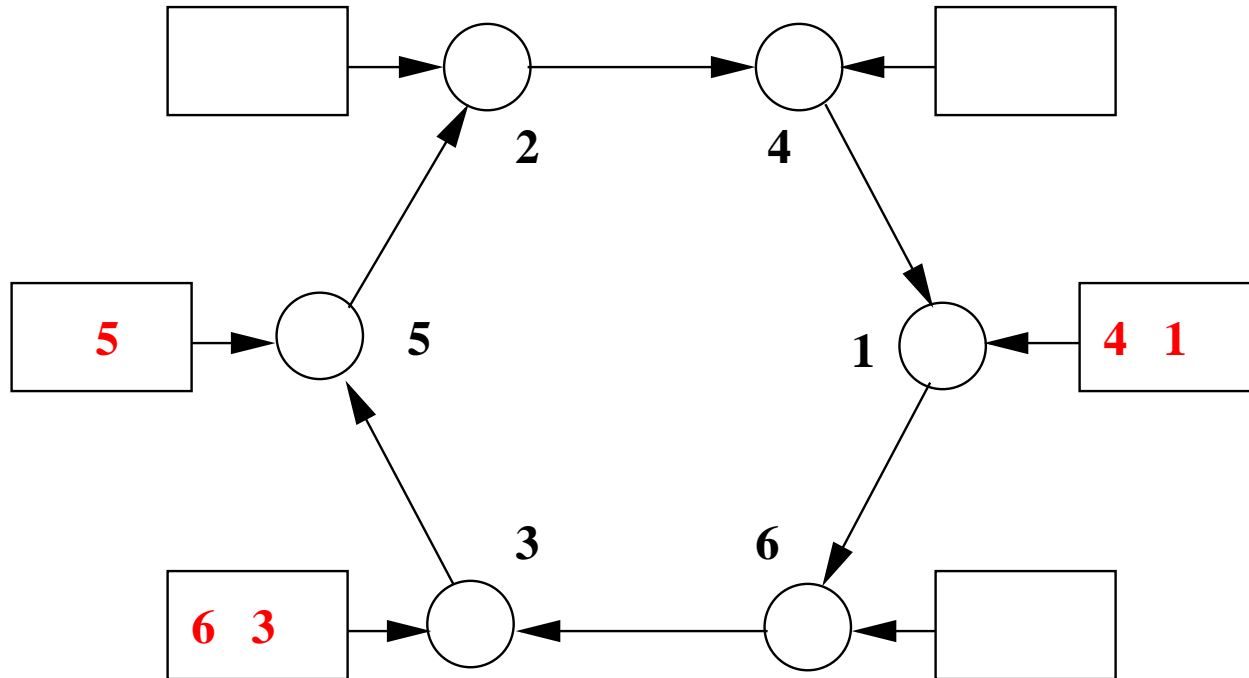
Accepting 4



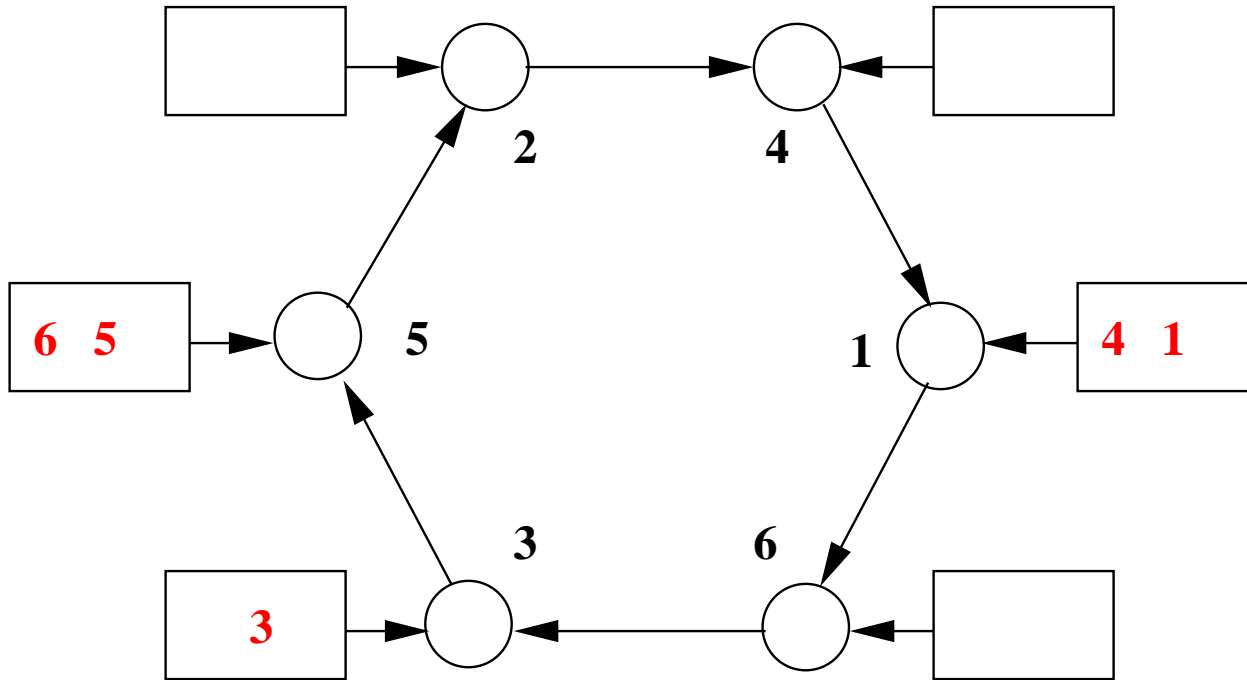
Accepting 6



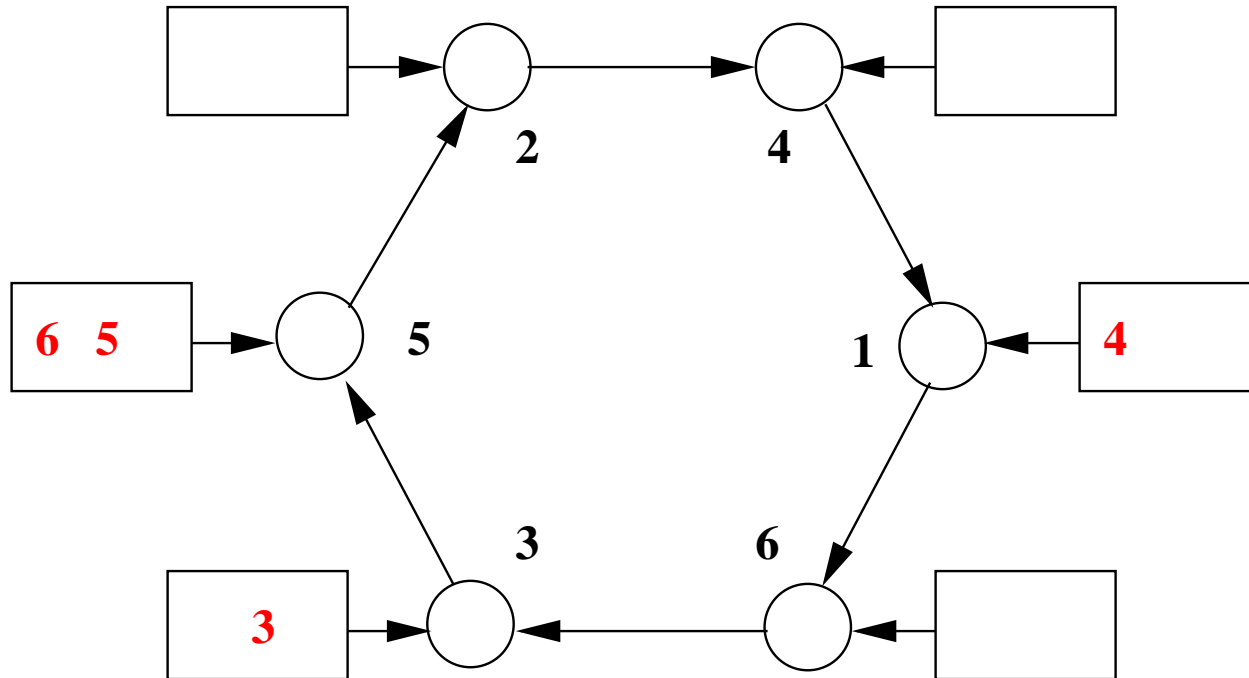
Rejecting 2



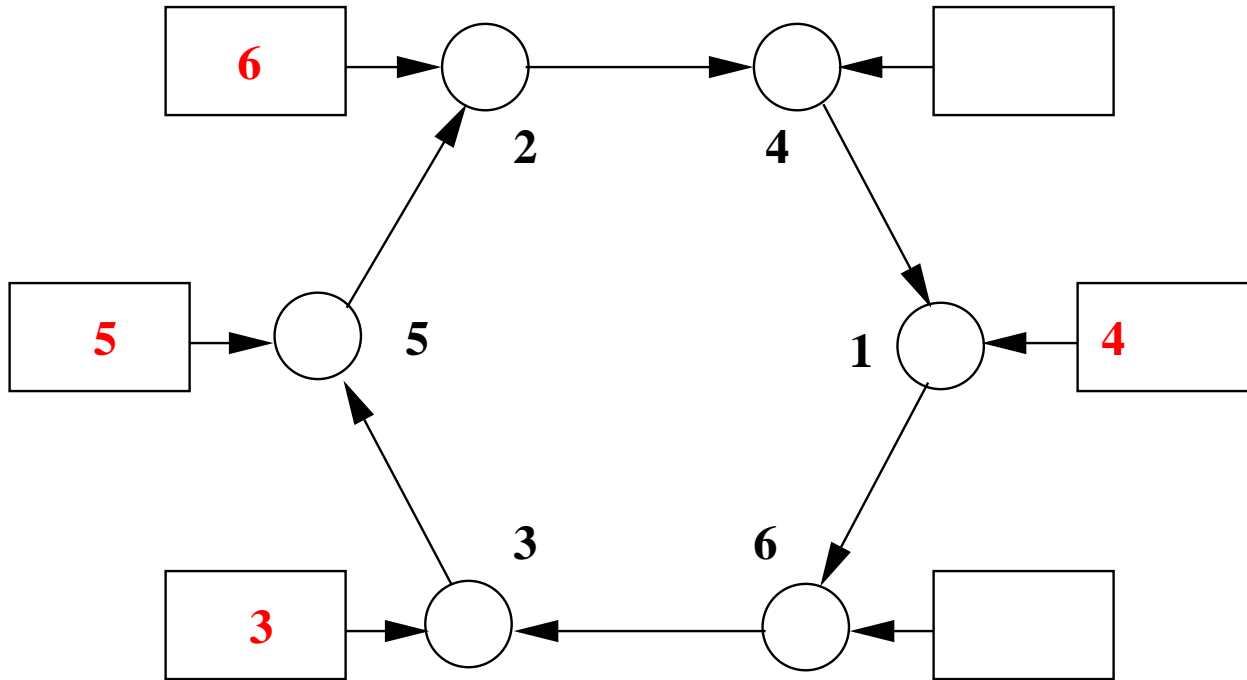
Accepting 6



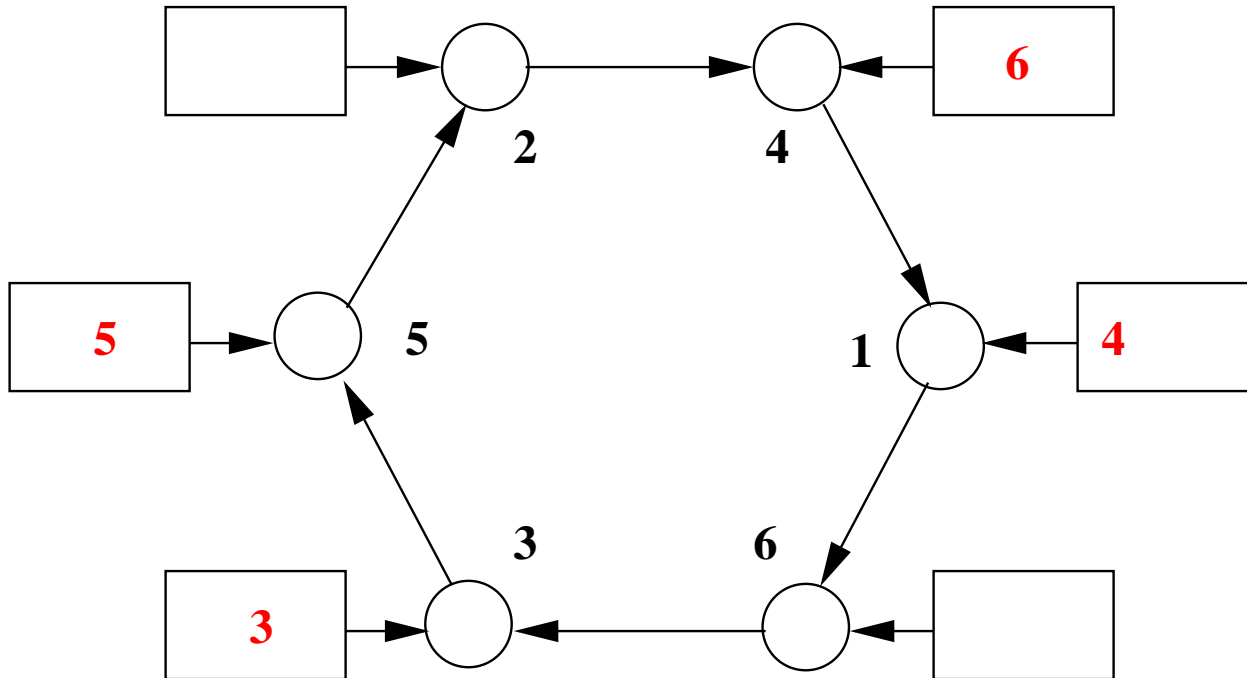
Rejecting 1



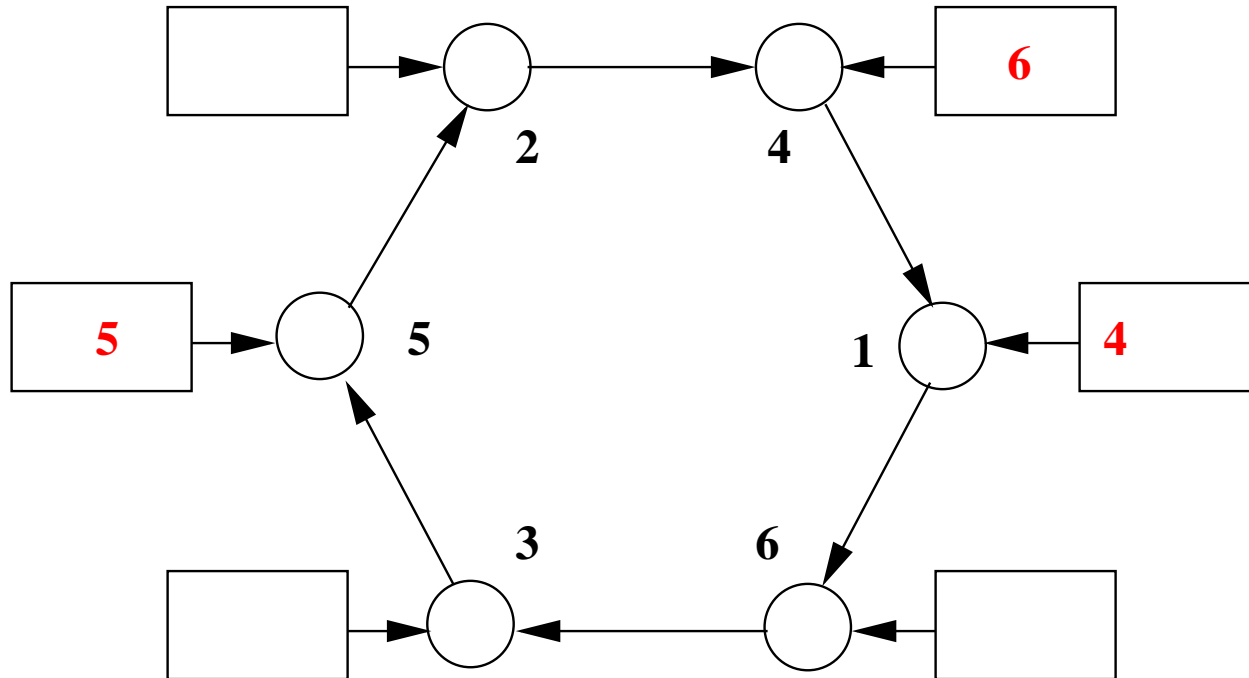
Accepting 6



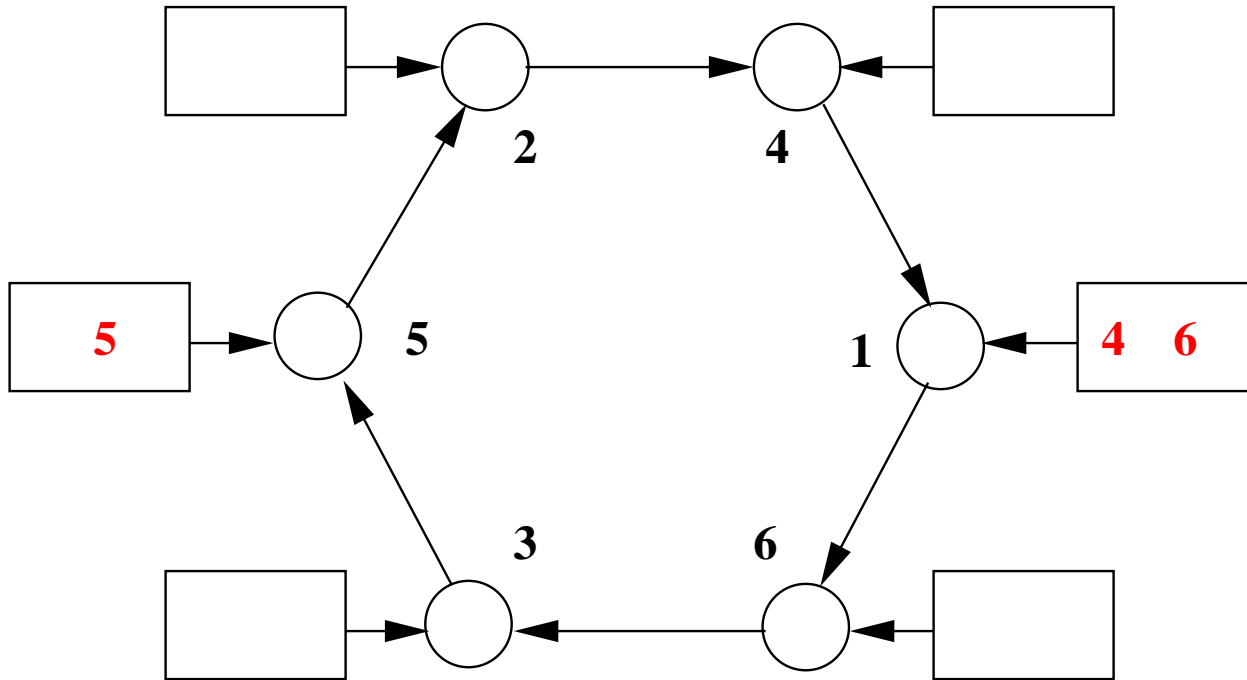
Accepting 6



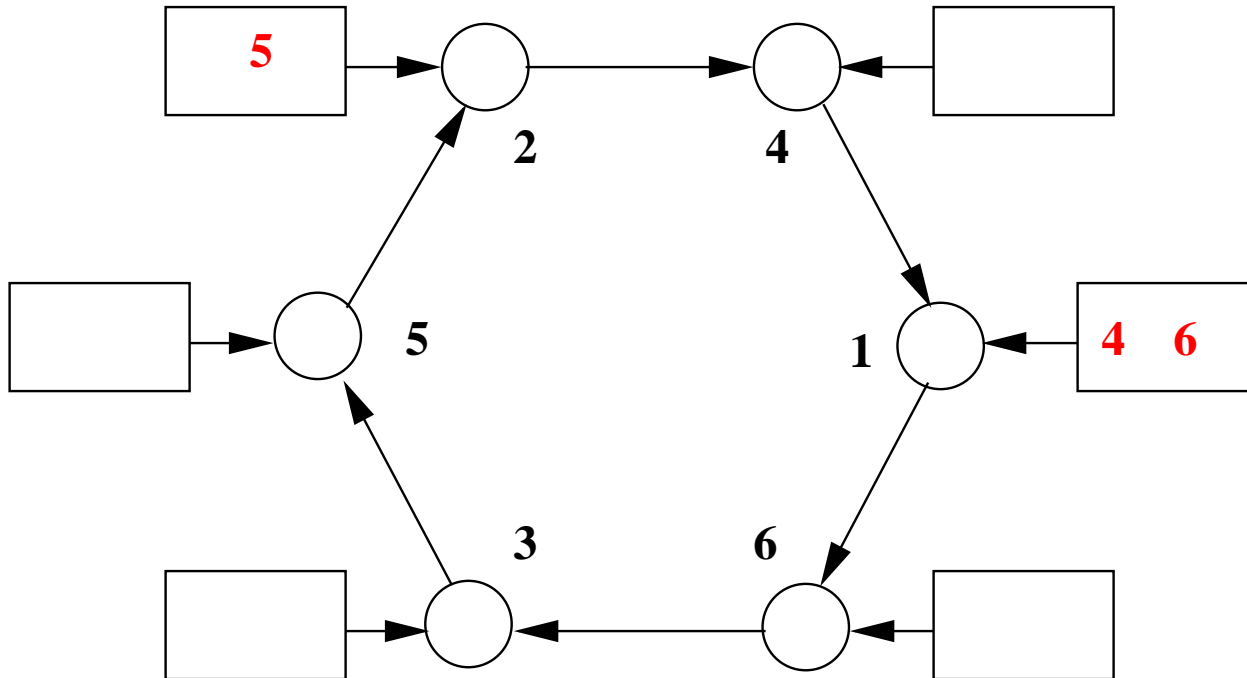
Rejecting 3



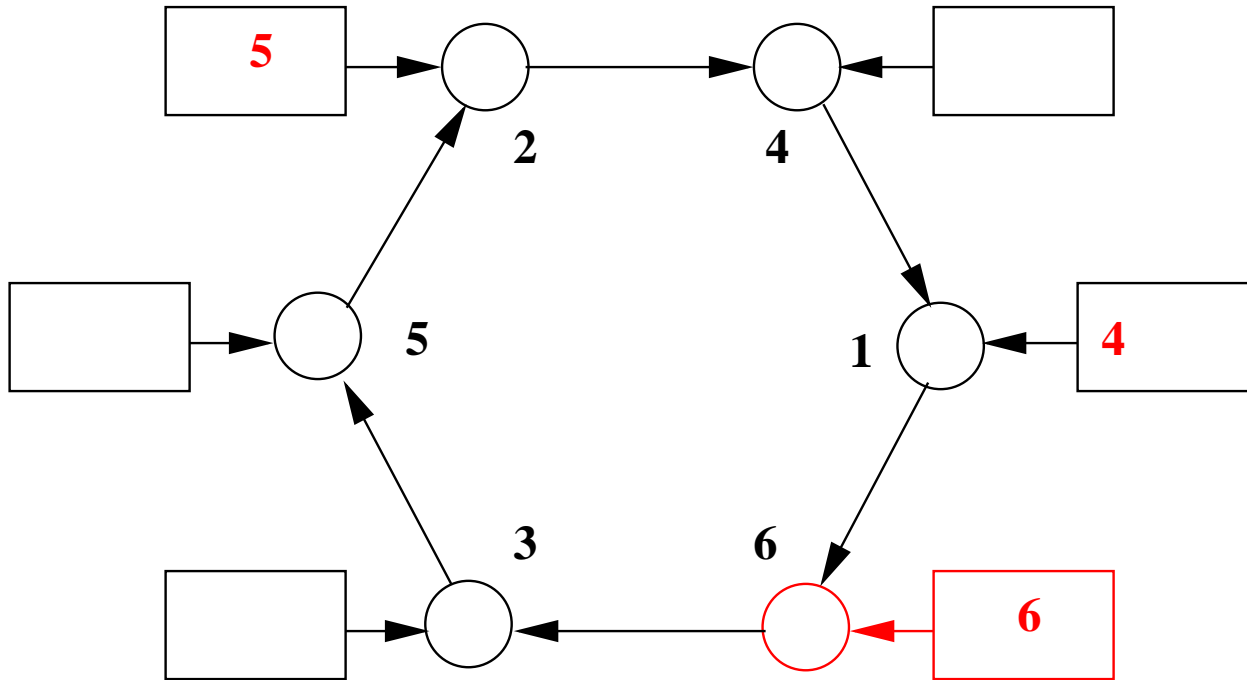
Accepting 6



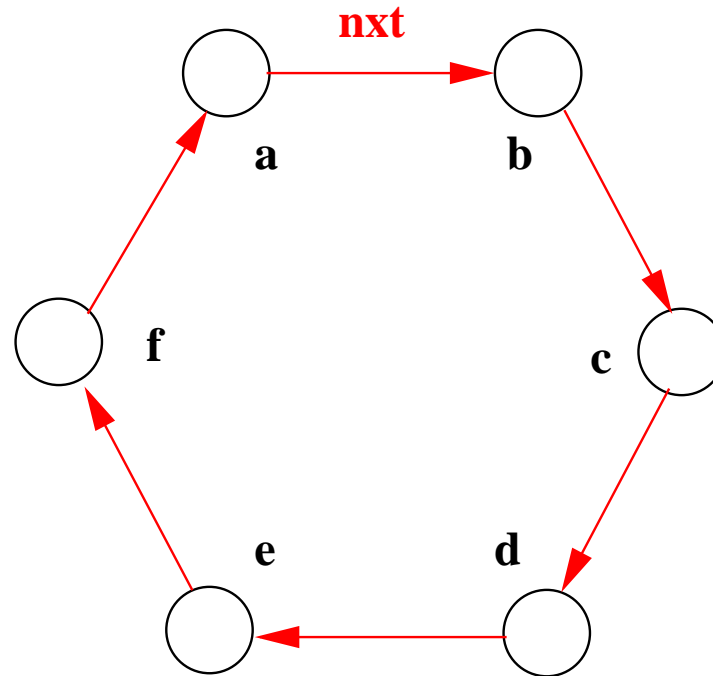
Accepting 5



Accepting 6. It is the Winner

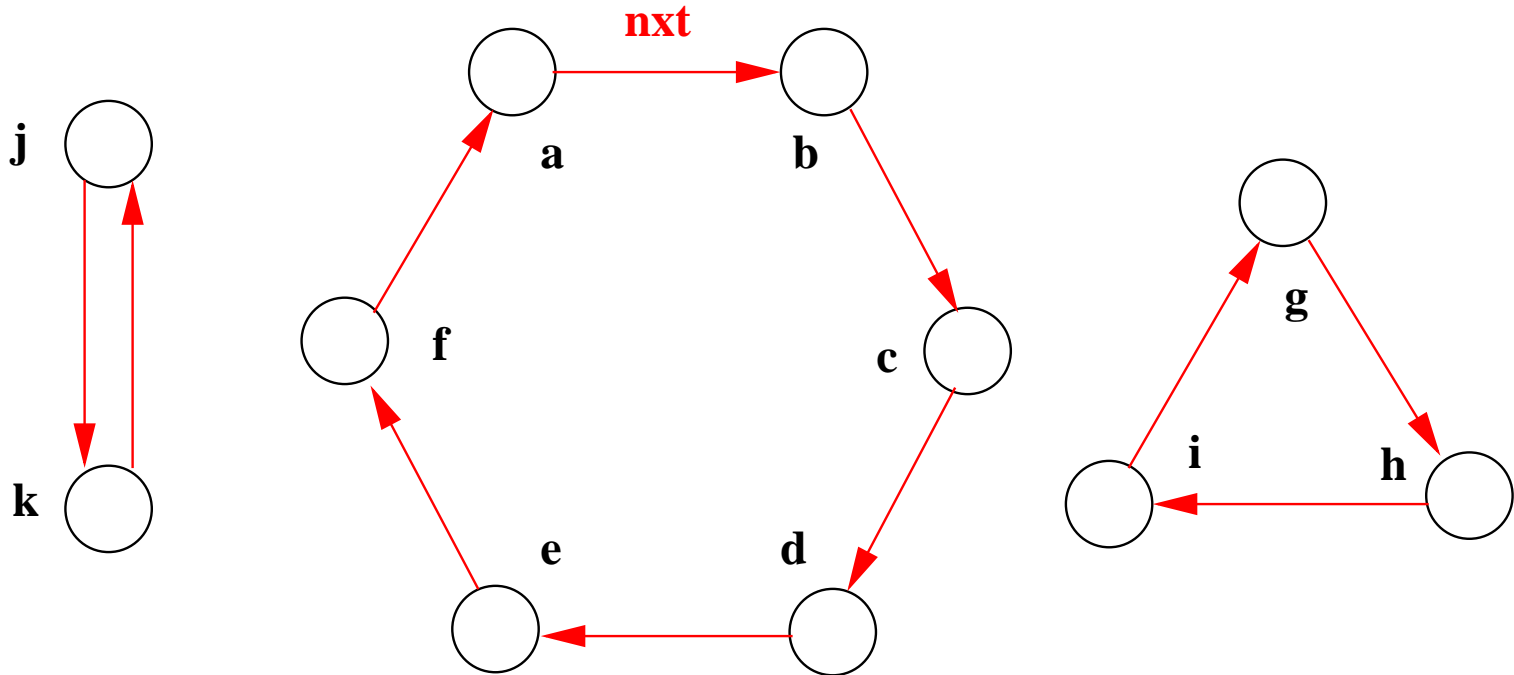


About Rings: the *nxt* Function



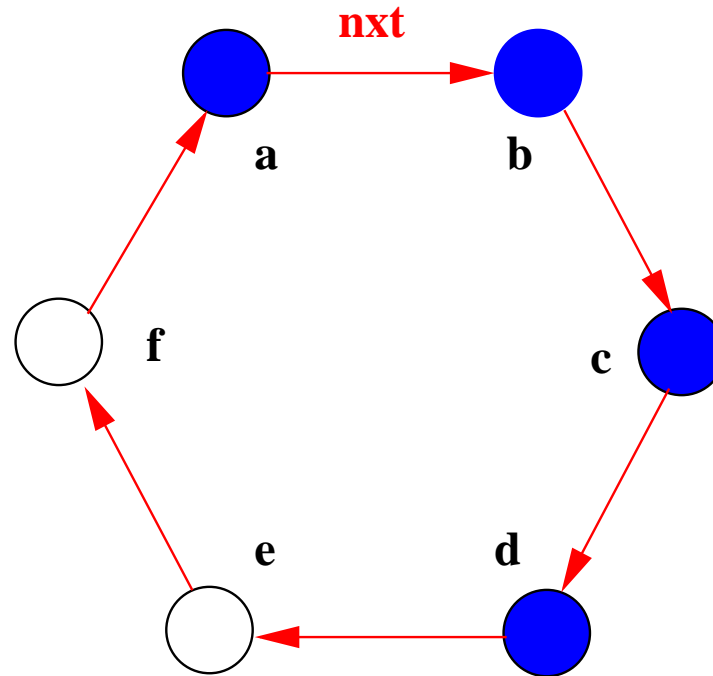
- The *nxt* function is a **bijection**
- A bijection is a **total function**
- The **converse** of a bijection is also a **total function**

About Rings: Problem with the *nxt* Function



- A bijection is **not sufficient** to define a **single ring**

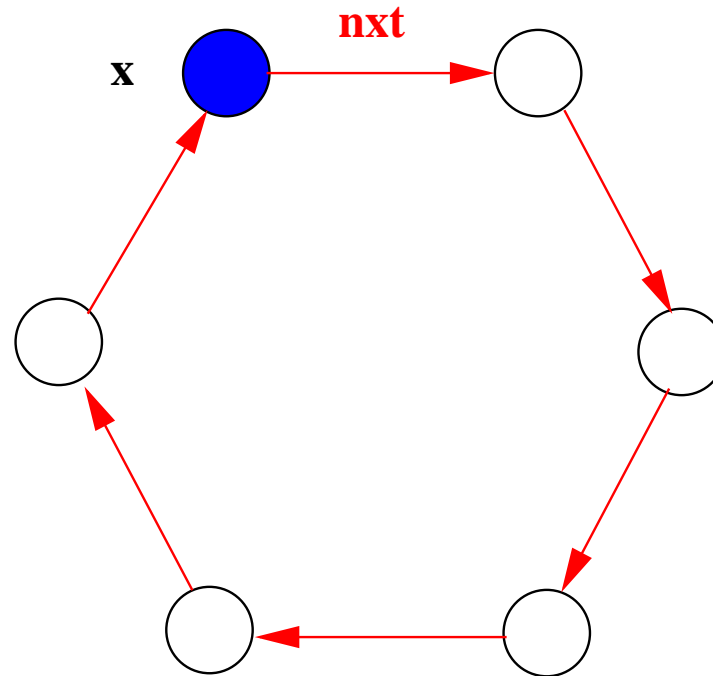
About Rings: a Notion of Interval (1)



- The interval **between a and d**

$$itv(a, d) = \{a, b, c, d\}$$

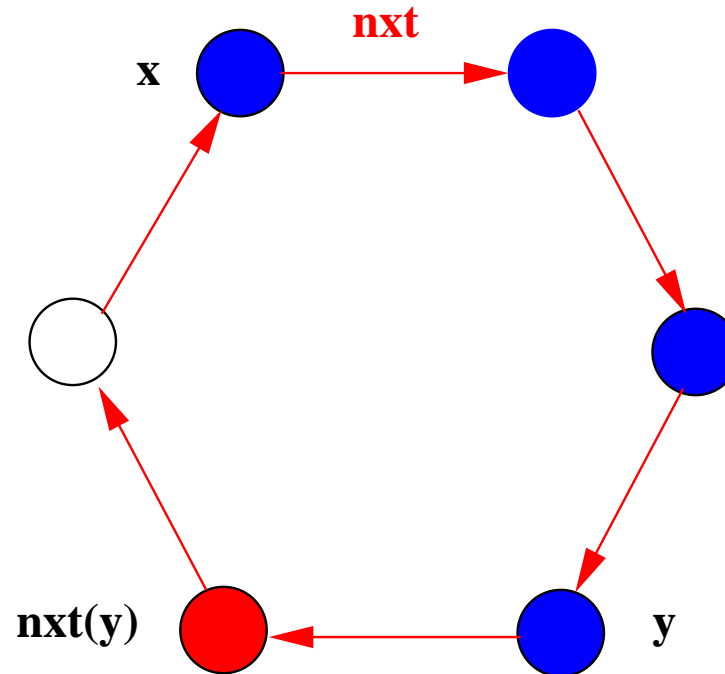
Definition of Interval (1)



- The **singleton interval**

$$itv(x, x) = \{x\}$$

Definition of Interval (2)



- The **non-singleton interval**

$$itv(x, nxt(y)) = itv(x, y) \cup \{nxt(y)\} \quad (\text{if } nxt(y) \neq x)$$

A Small Theory of Rings (1)

- Constants: nxt , itv

$$\text{prp_3 : } nxt \in N \rightsquigarrow N$$

$$\text{prp_4 : } itv \in N \times N \rightarrow \mathbb{P}(N)$$

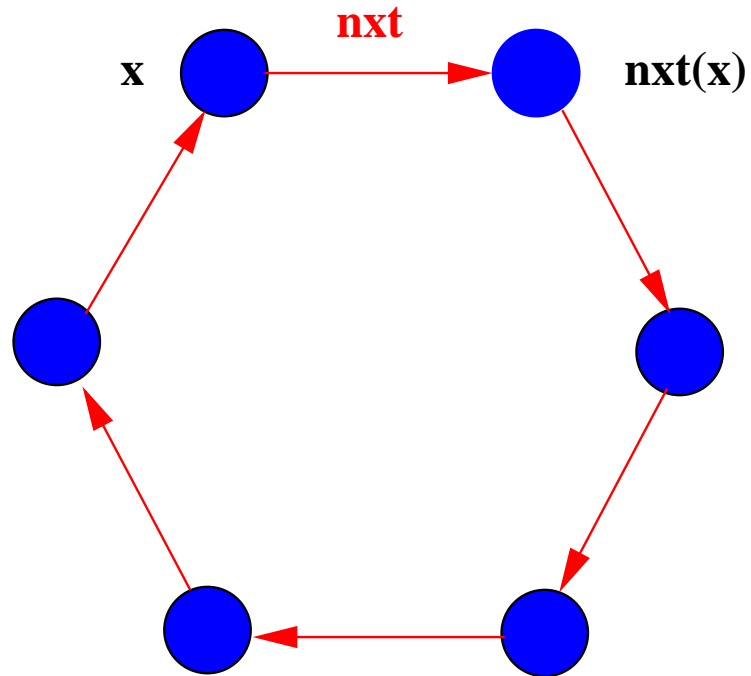
$$\text{prp_5 : } \forall x \cdot (x \in N \Rightarrow itv(x, x) = \{x\})$$

$$\text{prp_6 : } \forall x, y \cdot \left(\begin{array}{l} x \in N \\ y \in N \\ x \neq nxt(y) \\ \Rightarrow \\ itv(x, nxt(y)) = itv(x, y) \cup \{nxt(y)\} \end{array} \right)$$

More Conventions for Modeling

$S \rightsquigarrow T$	set of bijections from S to T
$S \times T$	Cartesian product of S and T

Complete Interval



- **In a ring**, the interval between $nxt(x)$ and x is N (the entire set of nodes)

A Small Theory of Rings (2)

- Constants: nxt , itv

$$\text{prp_2 : } \quad nxt \in N \mapsto N$$

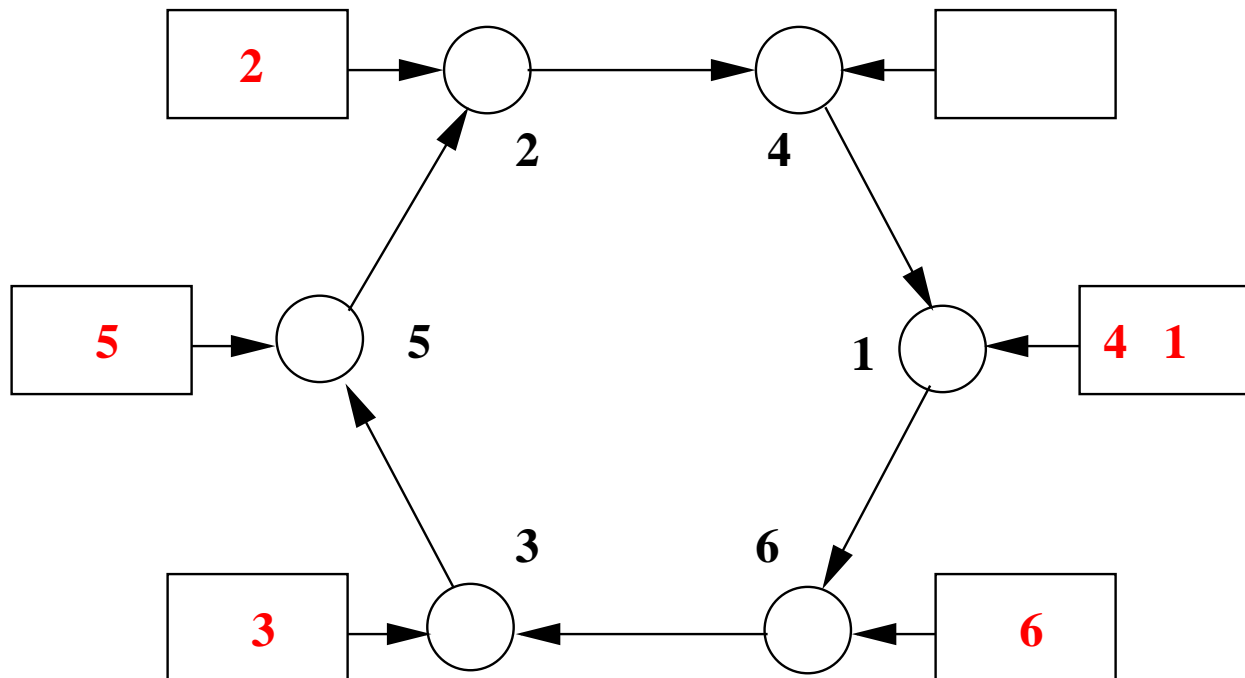
$$\text{prp_3 : } \quad itv \in N \times N \rightarrow \mathbb{P}(N)$$

$$\text{prp_4 : } \quad \forall x \cdot (x \in N \Rightarrow itv(x, x) = \{x\})$$

$$\text{prp_5 : } \quad \forall x, y \cdot \left(\begin{array}{l} x \in N \\ y \in N \\ x \neq nxt(y) \\ \Rightarrow \\ itv(x, nxt(y)) = itv(x, y) \cup \{nxt(y)\} \end{array} \right)$$

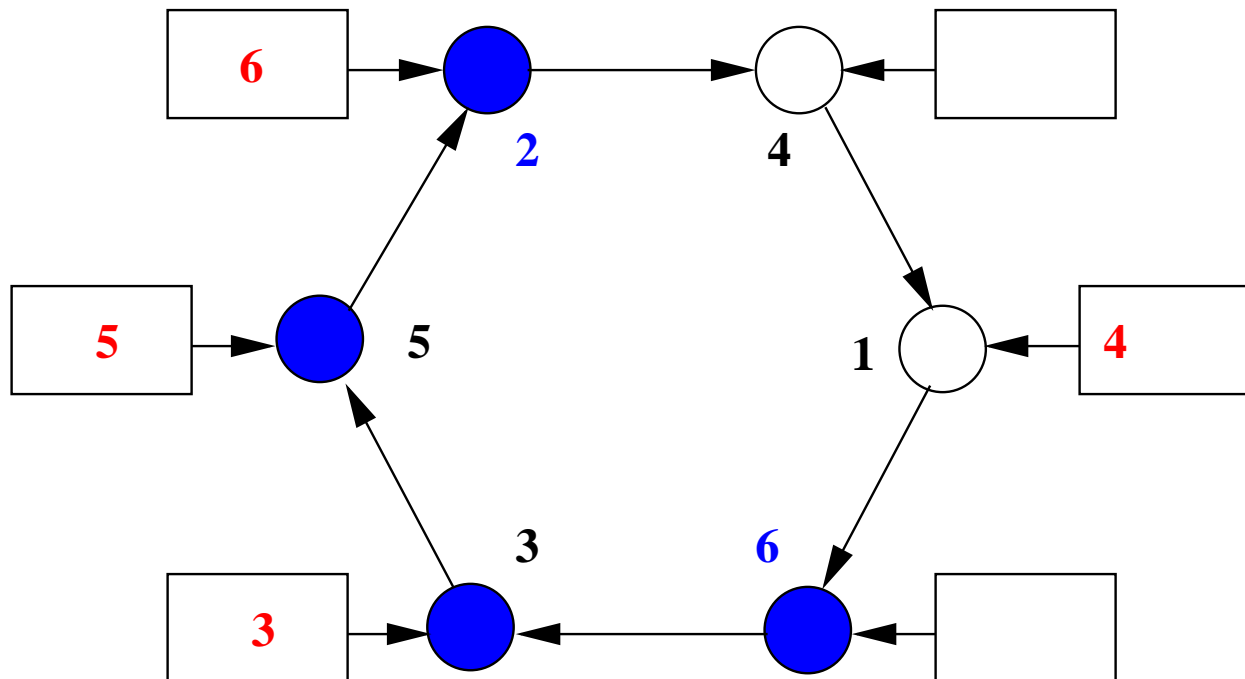
$$\text{prp_7 : } \quad \forall x \cdot (x \in N \Rightarrow itv(nxt(x), x) = N)$$

Observing the State



- Each name is **at most in one position**
- Each name is either **rejected** or **accepted** to the next position

The Key Observation



- **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

$$\text{inv_2} : pos \in N \leftrightarrow N$$

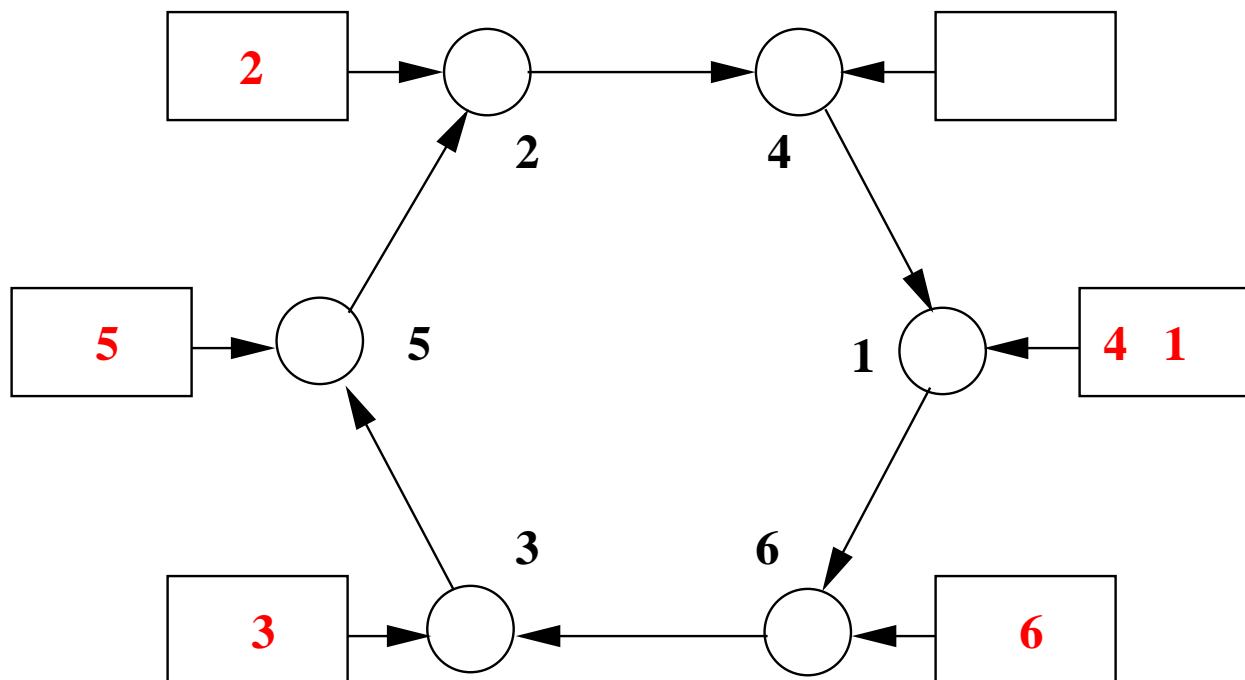
$$\text{inv_3} : \forall x \cdot (x \in \text{dom}(pos) \Rightarrow x = \max(itv(x, pos(x))))$$

- pos yields the position of each node that has **not yet been rejected**
- inv_3 is the **key invariant**

More Conventions for Modeling

dom	domain of a function
ran	range of a function
\triangleleft	domain restriction operator
\triangleleft	domain subtraction operator
$\text{id}(S)$	identity function built on the set S

Non-determinacy



- **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

when < *guard* > **then** < *action* > **end**

Non-deterministic Assignment

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

- Example (reject)

```
any  $x$  where  
   $x \in \text{dom}(pos)$   
   $x < \text{next}(pos(x))$   
then  
   $pos := \{x\} \triangleleft pos$   
end
```

The Complete Event

- Explication follows

```
reject  $\hat{=}$   
  when  
     $\exists x \cdot \left( \begin{array}{l} x \in \text{dom}(pos) \\ x < \text{next}(pos(x)) \end{array} \right)$   
  then  
    any  $x$  where  
       $x \in \text{dom}(pos)$   
       $x < \text{next}(pos(x))$   
    then  
       $pos := \{x\} \triangleleft pos$   
    end  
end
```

rewritten simply as

```
reject  $\hat{=}$   
  any  $x$  where  
     $x \in \text{dom}(pos)$   
     $x < \text{next}(pos(x))$   
  then  
     $pos := \{x\} \triangleleft pos$   
  end
```

First Refinement

```
init ≐  
  begin  
     $w := n$   
     $pos := \text{id}(N)$   
  end
```

```
elect ≐  
  any  $x$  where  
     $x \in \text{dom}(pos)$   
     $x = \text{nxt}(pos(x))$   
  then  
     $w := x$   
  end
```

```
accept ≐  
  any  $x$  where  
     $x \in \text{dom}(pos)$   
     $\text{nxt}(pos(x)) < x$   
  then  
     $pos(x) := \text{nxt}(pos(x))$   
  end
```

```
reject ≐  
  any  $x$  where  
     $x \in \text{dom}(pos)$   
     $x < \text{nxt}(pos(x))$   
  then  
     $pos := \{x\} \triangleleft pos$   
  end
```

Special Convention for Assignment

$f(x) := E$ stands for $f := \{x\} \triangleleft f \cup \{x \mapsto E\}$

$\{x\} \triangleleft f \cup \{x \mapsto E\}$ also written $f \triangleleft \{x \mapsto E\}$

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

Further Studies

- Feasibility
- Before-after predicates for non-deterministic assignments
- Generalization of invariant preservation statement
- Generalization of refinement

Feasibility Statement

- An event of the following form must be **feasible**

```
when
   $G(v)$ 
then
  any  $x$  where
     $C(x)$ 
  then
     $v := E(x, v)$ 
  end
end
```

Statement to be proved

$$G(v) \Rightarrow \exists x \cdot C(x)$$

Special Case

Such an event is **automatically feasible**

reject $\hat{=}$

when

$\exists x \cdot \left(\begin{array}{l} x \in \text{dom}(pos) \\ x < \text{next}(pos(x)) \end{array} \right)$

then

any x **where**

$x \in \text{dom}(pos)$

$x < \text{next}(pos(x))$

then

$pos := \{x\} \triangleleft pos$

end

end

rewritten simply as

reject $\hat{=}$

any x **where**

$x \in \text{dom}(pos)$

$x < \text{next}(pos(x))$

then

$pos := \{x\} \triangleleft pos$

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

any x **where**
 $C(x)$
then
 $v := E(x, v)$
end

- Corresponding **Before-after predicate**

$$\exists x \cdot \left(\begin{array}{l} C(x) \\ v' = E(x, v) \end{array} \right)$$

Invariant Preservation Statement (Reminder)

- Given **constants** c , **properties** $P(c)$, **variables** v , and **invariant** $I(c, v)$
- Given an **event** of the form

when $G(c, v)$ **then** $v' = E(c, v)$ **end**

- We have **to prove**

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

Invariant Preservation Statement (Generalization)

- Given **constants** c , **properties** $P(c)$, **variables** v , and **invariant** $I(c, v)$
- Given an **event** of the form

when $G(c, v)$ **then** $Q(c, v, v')$ **end**

- We have **to prove**

$P(c)$
 $I(c, v)$
 $G(c, v)$
 $Q(c, v, v')$
 \Rightarrow
 $I(c, v')$

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

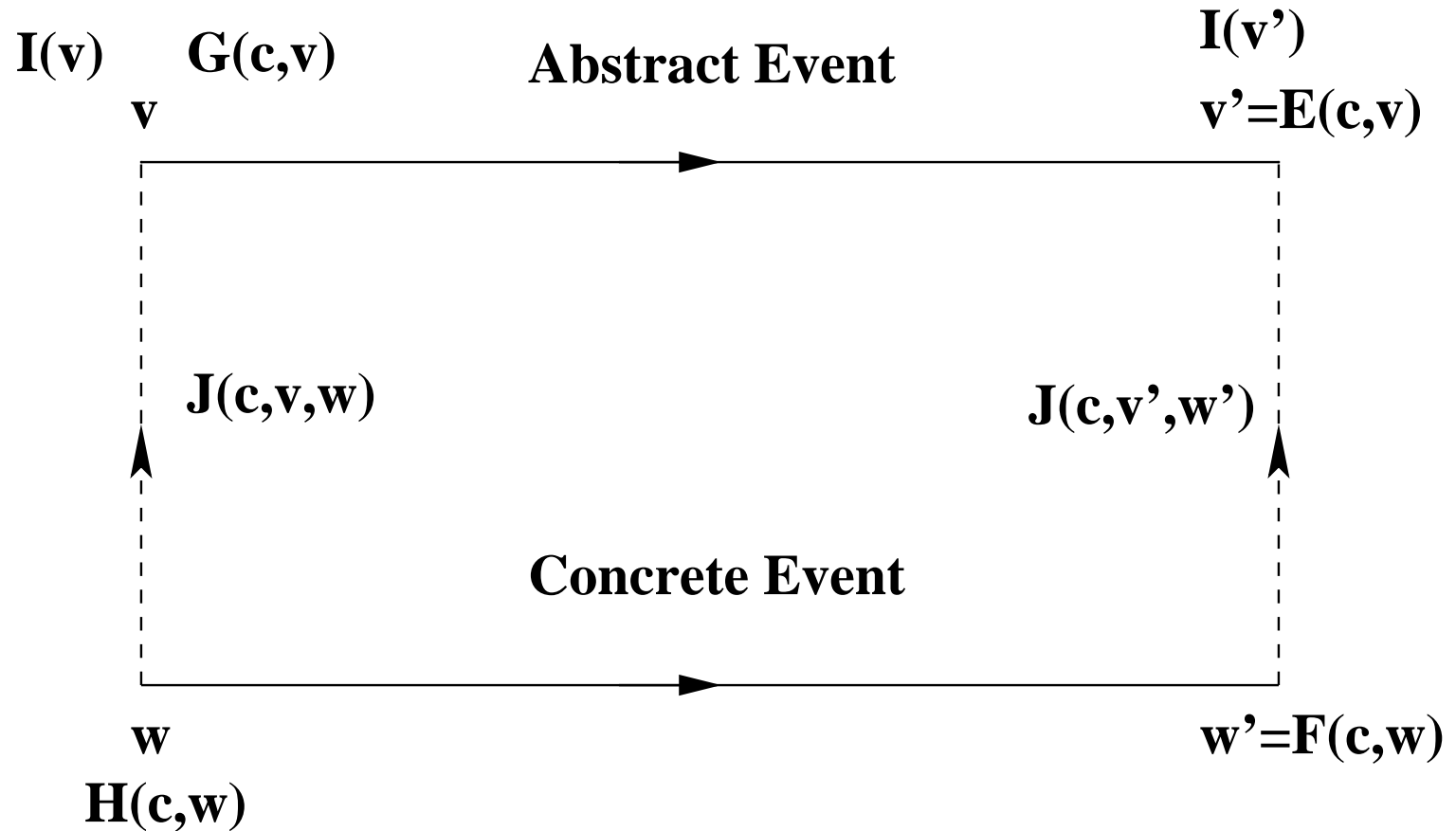
$H(c, w)$

then

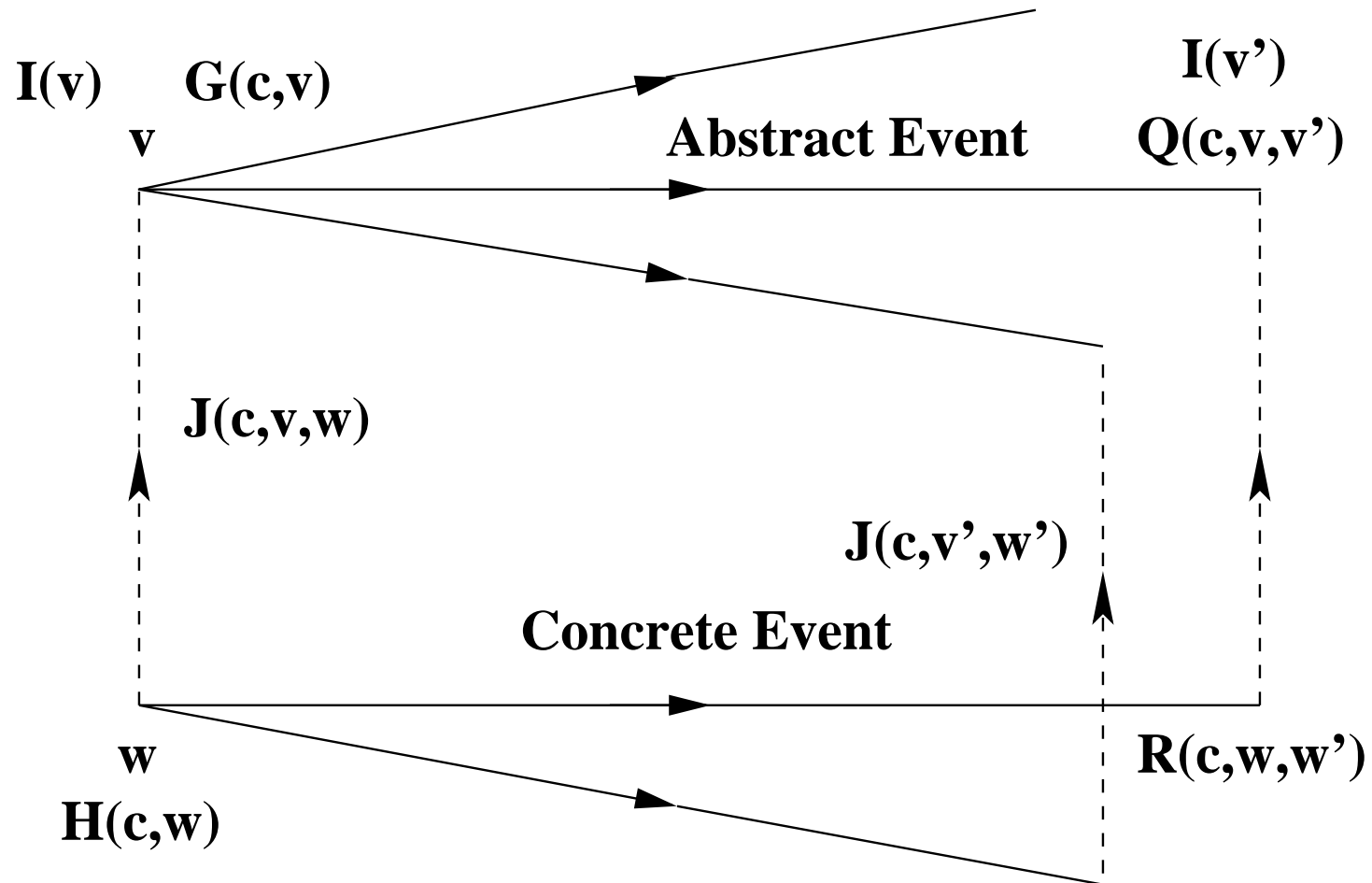
$R(c, w, w')$

end

State and Event Refinement (Deterministic Case)



State and Event Ref. (Non-deterministic Case)



Correct Refinement Proof: Generalization (cont'd)

- One has to **prove**:

$$\begin{array}{l} P(c) \\ I(c, v) \\ J(c, v, w) \\ H(c, w) \\ R(c, w, w') \\ \Rightarrow \\ G(c, v) \\ \exists v'. \left(\begin{array}{l} Q(c, v, v') \\ J(c, v', w') \end{array} \right) \end{array}$$

- **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

$\exists x \cdot \left(\begin{array}{l} A(c, x, v) \\ v' = E(x, c, v) \end{array} \right)$

end

when

$H(c, w)$

then

$\exists y \cdot \left(\begin{array}{l} B(c, y, w) \\ w' = F(y, c, w) \end{array} \right)$

end

Correct Refinement Proof

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

$$\begin{array}{l} P(c) \\ I(c, v) \\ J(c, v, w) \\ H(c, w) \\ \exists y \cdot \left(\begin{array}{l} B(c, y, w) \\ w' = F(y, c, w) \end{array} \right) \\ \Rightarrow \\ G(c, v) \\ \exists v' \cdot \left(\begin{array}{l} \exists x \cdot \left(\begin{array}{l} A(c, x, v) \\ v' = E(x, c, v) \end{array} \right) \\ J(c, v', w') \end{array} \right) \end{array}$$

$$\begin{array}{l} P(c) \\ I(c, v) \\ J(c, v, w) \\ H(c, w) \\ B(c, y, w) \\ \Rightarrow \\ G(c, v) \\ \exists x \cdot \left(\begin{array}{l} A(c, x, v) \\ J(c, E(x, c, v), \\ F(y, c, w)) \end{array} \right) \end{array}$$

Correct Refinement Proof: Providing a “witness”

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

$$\begin{array}{l} P(c) \\ I(c, v) \\ J(c, v, w) \\ H(c, w) \\ B(c, y, w) \\ \Rightarrow \\ G(c, v) \\ \exists x \cdot \left(\begin{array}{l} A(c, x, v) \\ J(c, E(x, c, v), \\ F(y, c, w)) \end{array} \right) \end{array}$$

$$\begin{array}{l} P(c) \\ I(c, v) \\ J(c, v, w) \\ H(c, w) \\ B(c, y, w) \\ \Rightarrow \\ G(c, v) \\ A(c, W(c, w, y), v) \\ J(c, E(W(c, w, y), c, v), \\ F(y, c, w)) \end{array}$$

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

$\exists y \cdot \left(\begin{array}{l} B(c, y, w) \\ w' = F(y, c, w) \end{array} \right)$

end

Correct Refinement Proof

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

$$\begin{array}{l} P(c) \\ I(c, v) \\ J(c, v, w) \\ H(c, w) \\ \exists y \cdot \left(\begin{array}{l} B(c, y, w) \\ w' = F(y, c, w) \end{array} \right) \\ \Rightarrow \\ G(c, v) \\ \exists v' \cdot \left(\begin{array}{l} v' = E(x, c, v) \\ J(c, v', w') \end{array} \right) \end{array}$$

$$\begin{array}{l} P(c) \\ I(c, v) \\ J(c, v, w) \\ H(c, w) \\ B(c, y, w) \\ \Rightarrow \\ G(c, v) \\ J(c, E(x, c, v), F(y, c, w)) \end{array}$$

The Events (Reminder)

```
init ≐  
  begin  
     $w := n$   
     $pos := \text{id}(N)$   
  end
```

```
elect ≐  
  any  $x$  where  
     $x \in \text{dom}(pos)$   
     $x = \text{nxt}(pos(x))$   
  then  
     $w := x$   
  end
```

```
accept ≐  
  any  $x$  where  
     $x \in \text{dom}(pos)$   
     $\text{nxt}(pos(x)) < x$   
  then  
     $pos(x) := \text{nxt}(pos(x))$   
  end
```

```
reject ≐  
  any  $x$  where  
     $x \in \text{dom}(pos)$   
     $x < \text{nxt}(pos(x))$   
  then  
     $pos := \{x\} \triangleleft pos$   
  end
```

Refinement Proof for event elect

$$\begin{aligned} (\text{abs_})\text{elect} &\hat{=} \\ &\text{begin} \\ &\quad w' = \max(N) \\ &\text{end} \end{aligned}$$
$$\begin{aligned} (\text{ref_})\text{elect} &\hat{=} \\ &\text{when} \\ &\quad \exists x \cdot \left(\begin{array}{l} x \in \text{dom}(pos_1) \\ x = \text{next}(pos_1(x)) \end{array} \right) \\ &\text{then} \\ &\quad \exists x \cdot \left(\begin{array}{l} x \in \text{dom}(pos_1) \\ x = \text{next}(pos_1(x)) \\ w'_1 = x \end{array} \right) \\ &\text{end} \end{aligned}$$

Refinement Proof for event elect (cont'd)

$$\begin{aligned} & \forall x \cdot \left(\begin{array}{l} x \in N \\ \Rightarrow \\ itv(next(x), x) = N \end{array} \right) \\ & pos \in N \leftrightarrow N \\ & \forall x \cdot \left(\begin{array}{l} x \in \text{dom}(pos) \\ \Rightarrow \\ x = \max(itv(x, pos(x))) \end{array} \right) \\ & x \in \text{dom}(pos_1) \\ & pos = pos_1 \\ & x = next(pos_1(x)) \\ \Rightarrow & \\ & \max(N) = x \end{aligned}$$

$$\begin{aligned} & pos_1 \in N \leftrightarrow N \\ & \forall x \cdot \left(\begin{array}{l} x \in N \\ \Rightarrow \\ itv(next(x), x) = N \end{array} \right) \\ & x = \max(itv(x, pos_1(x))) \\ & x \in \text{dom}(pos_1) \\ & x = next(pos_1(x)) \\ \Rightarrow & \\ & \max(N) = x \end{aligned}$$

- Applying equality $pos = pos_1$
- **Instantiating** the second universal quantification with x

Refinement Proof for event elect (cont'd)

$$\begin{aligned} & pos_1 \in N \leftrightarrow N \\ & \forall x \cdot \left(\begin{array}{l} x \in N \\ \Rightarrow \\ itv(nxt(x), x) = N \end{array} \right) \Rightarrow itv(nxt(pos_1(x)), pos_1(x)) = N \\ & x = \max(itv(x, pos_1(x))) \Rightarrow \max(N) = \\ & x \in \text{dom}(pos_1) \Rightarrow \max(itv(nxt(pos_1(x)), pos_1(x))) \\ & x = nxt(pos_1(x)) \\ & \Rightarrow \\ & \max(N) = x \end{aligned}$$

- 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**

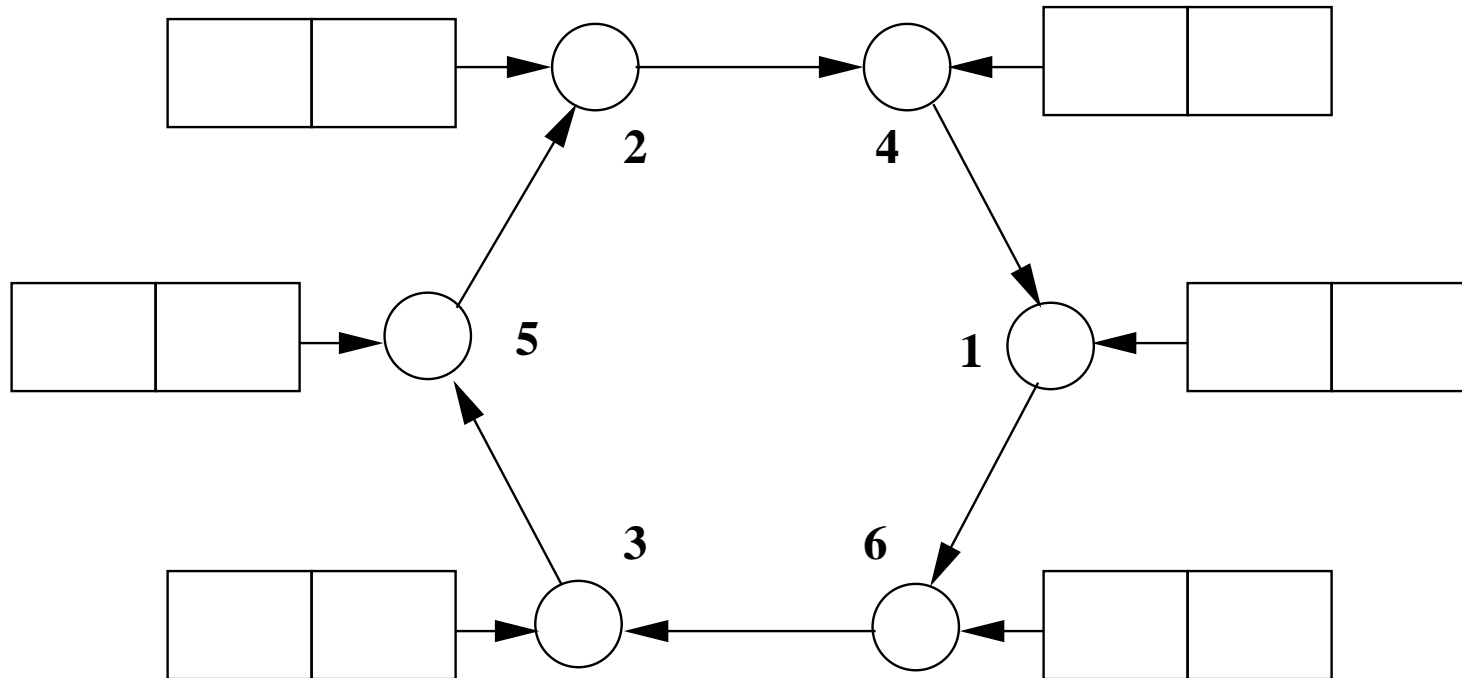
Exercises

- 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)

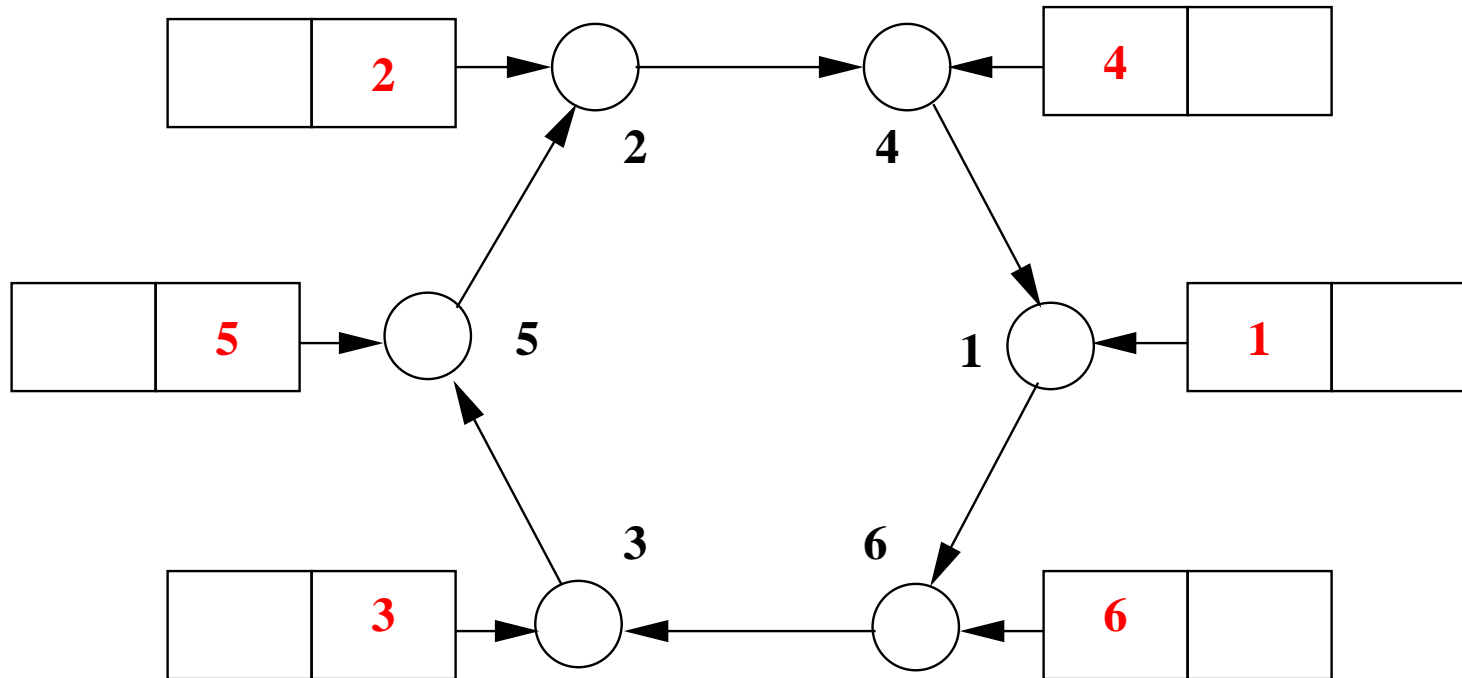
Some Ideas for a Refinement

- We introduce **two buffers** *in* and *out* at each node
- We **remove** the variable *pos*
- We introduce a new event **send**

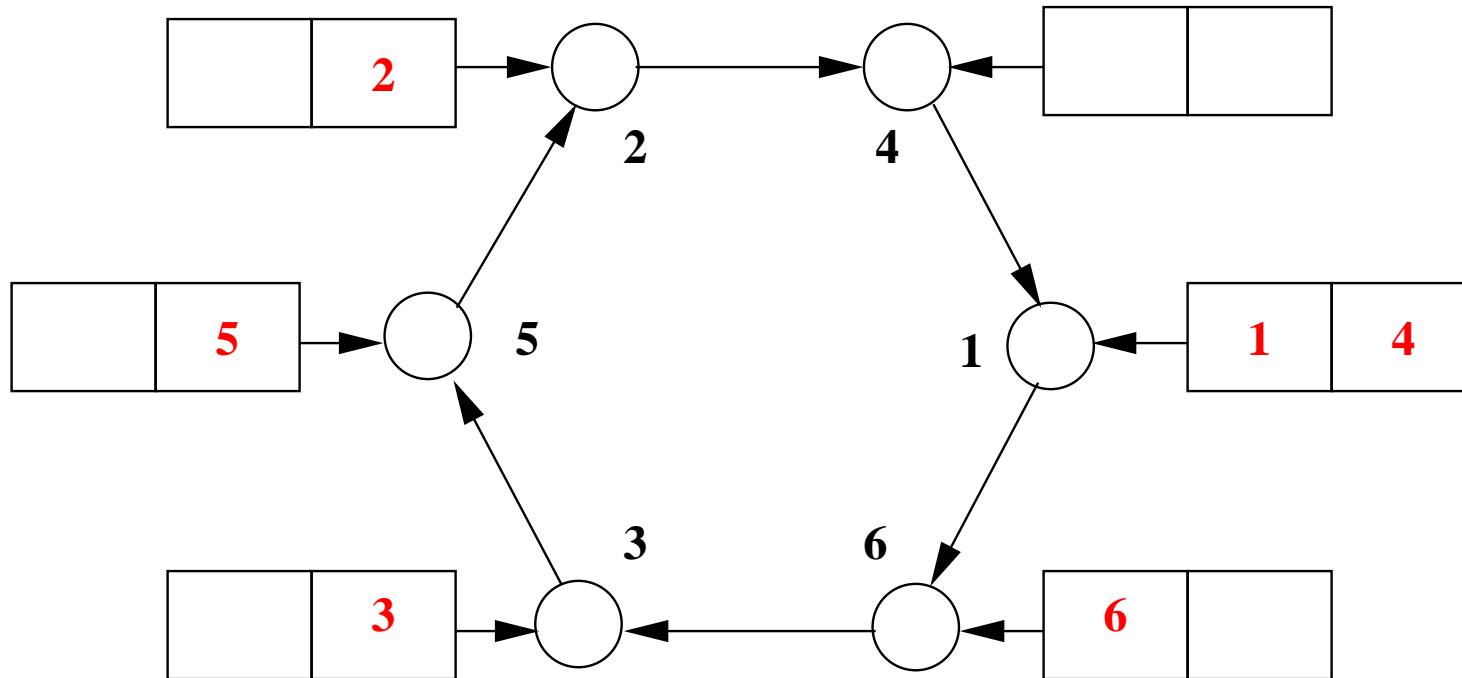
Basic Situation



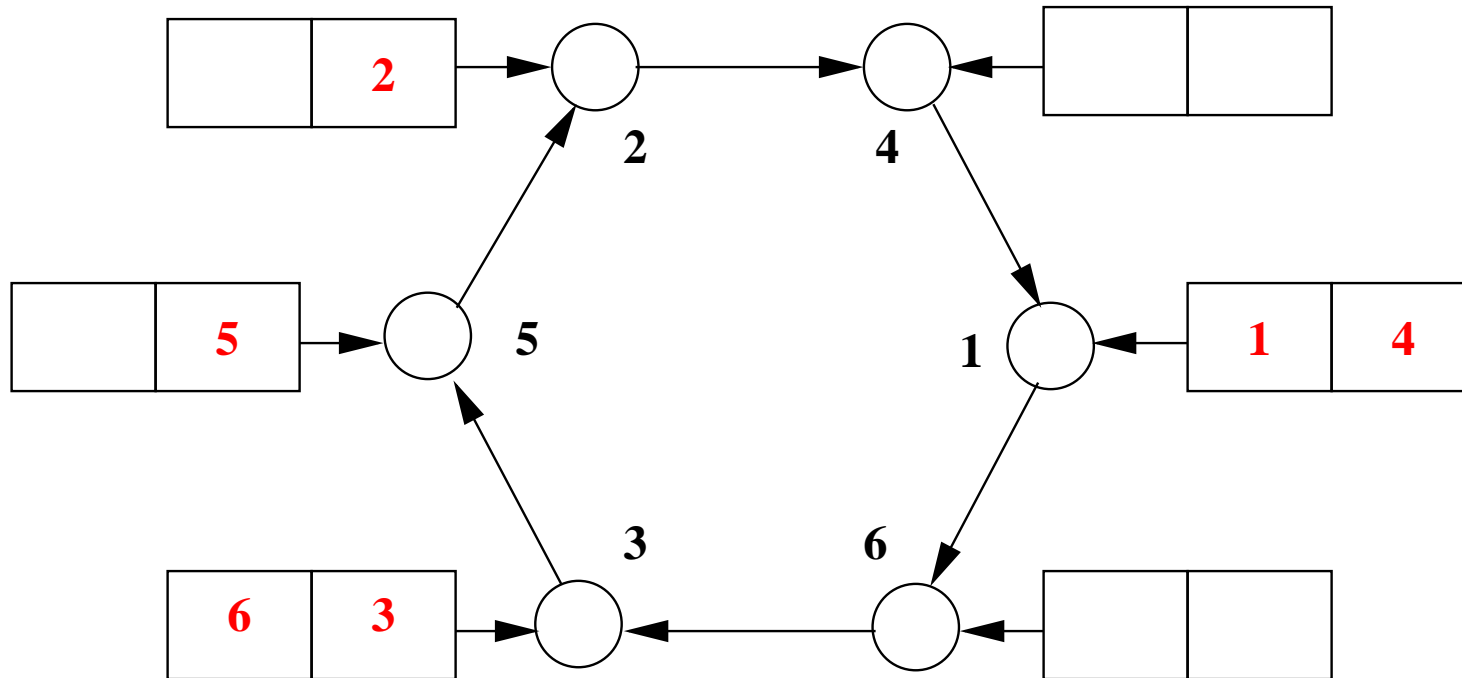
Initial Situation



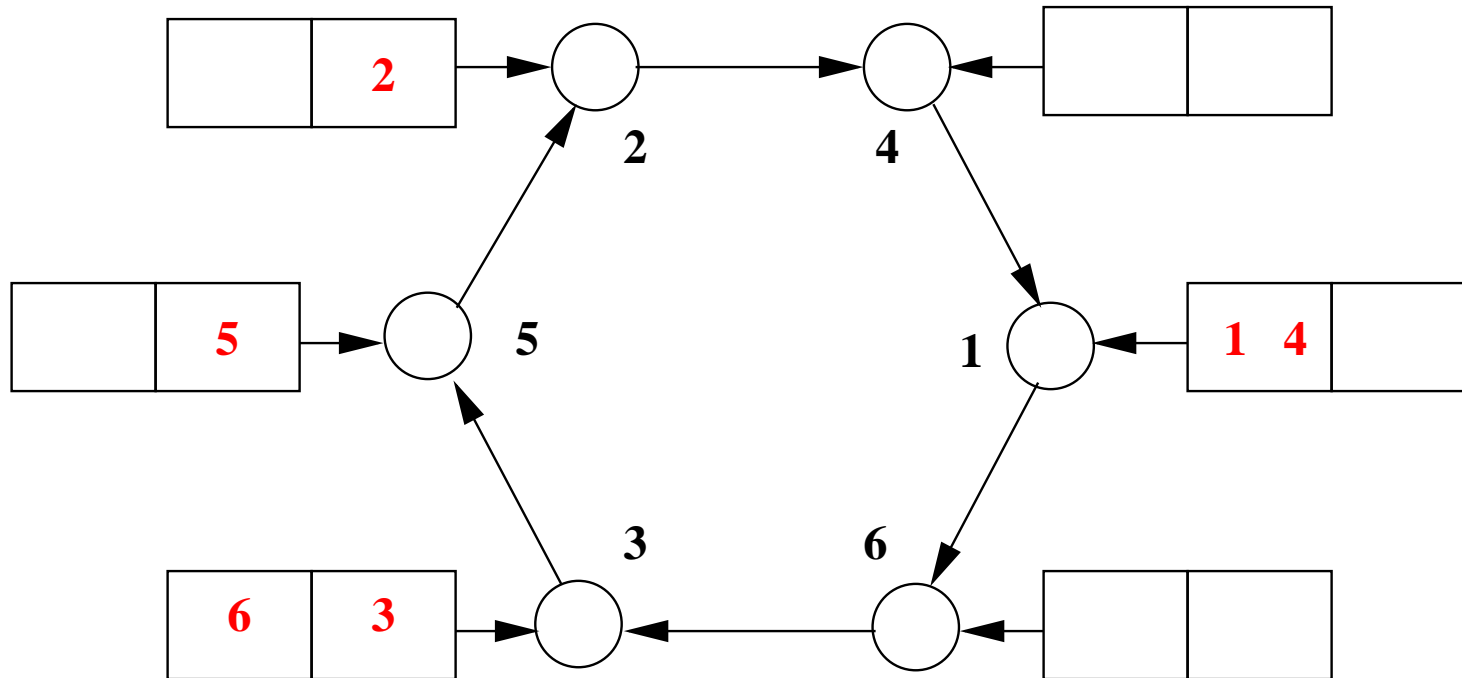
Sending 4



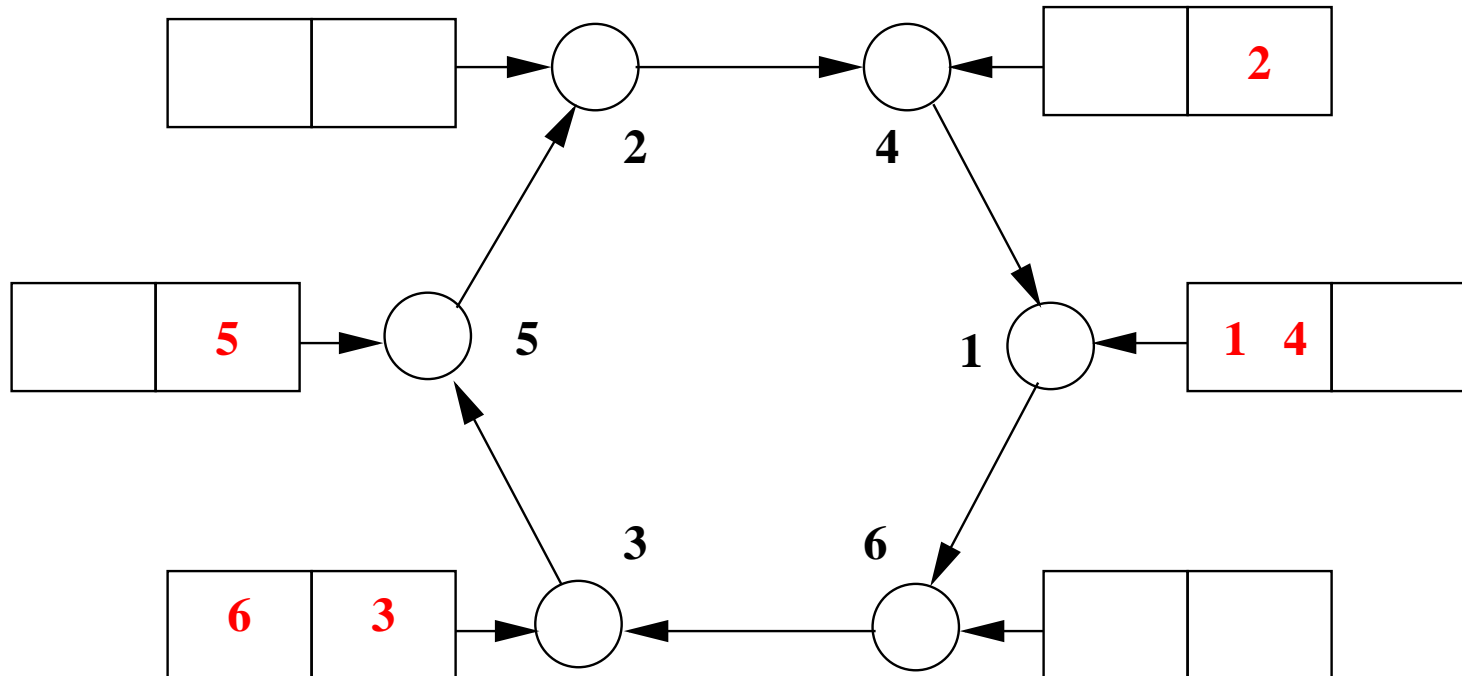
Sending 6



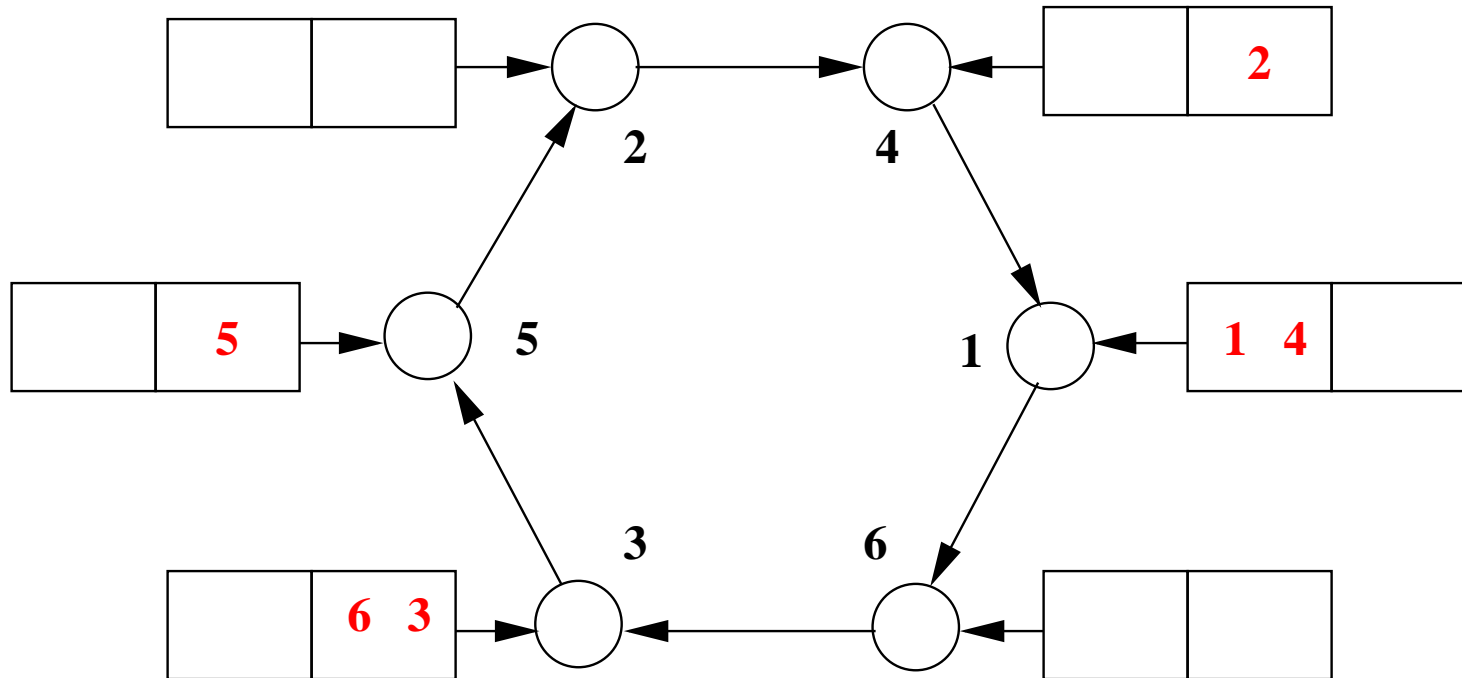
Accepting 4



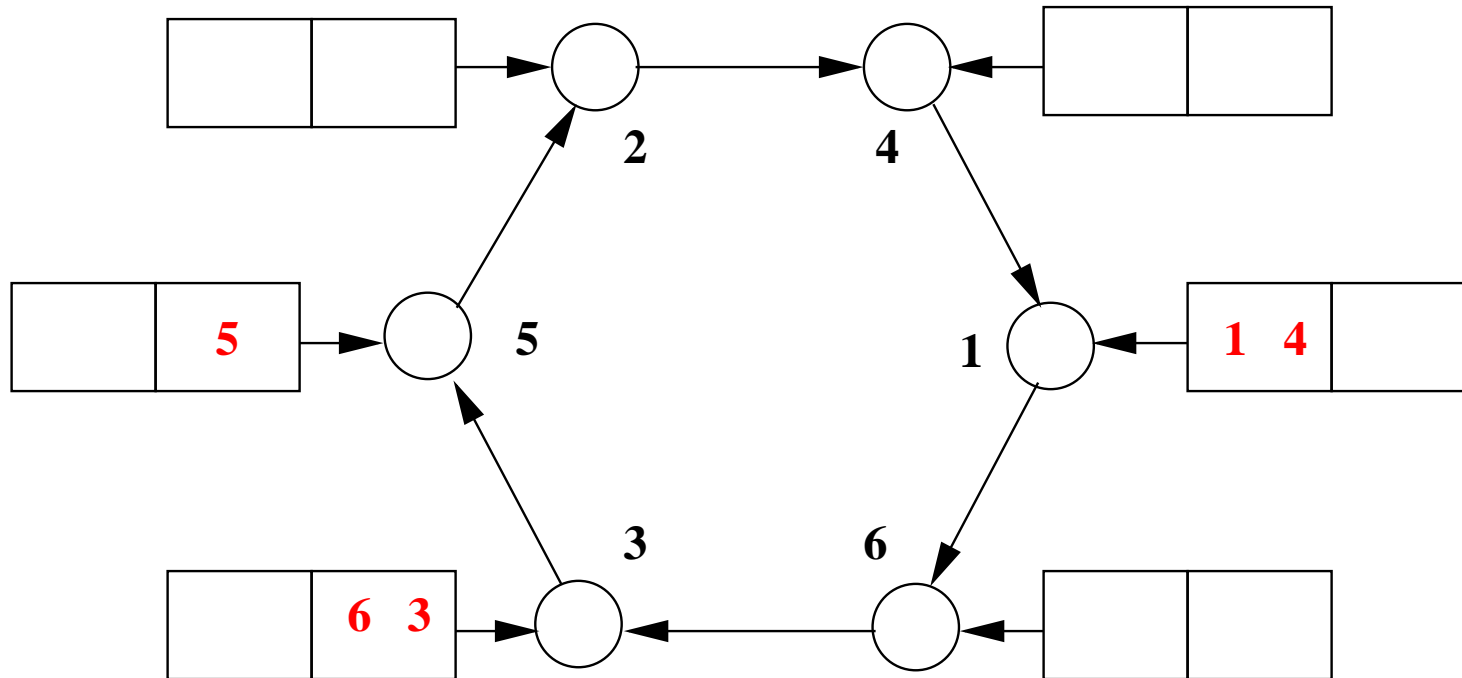
Sending 2



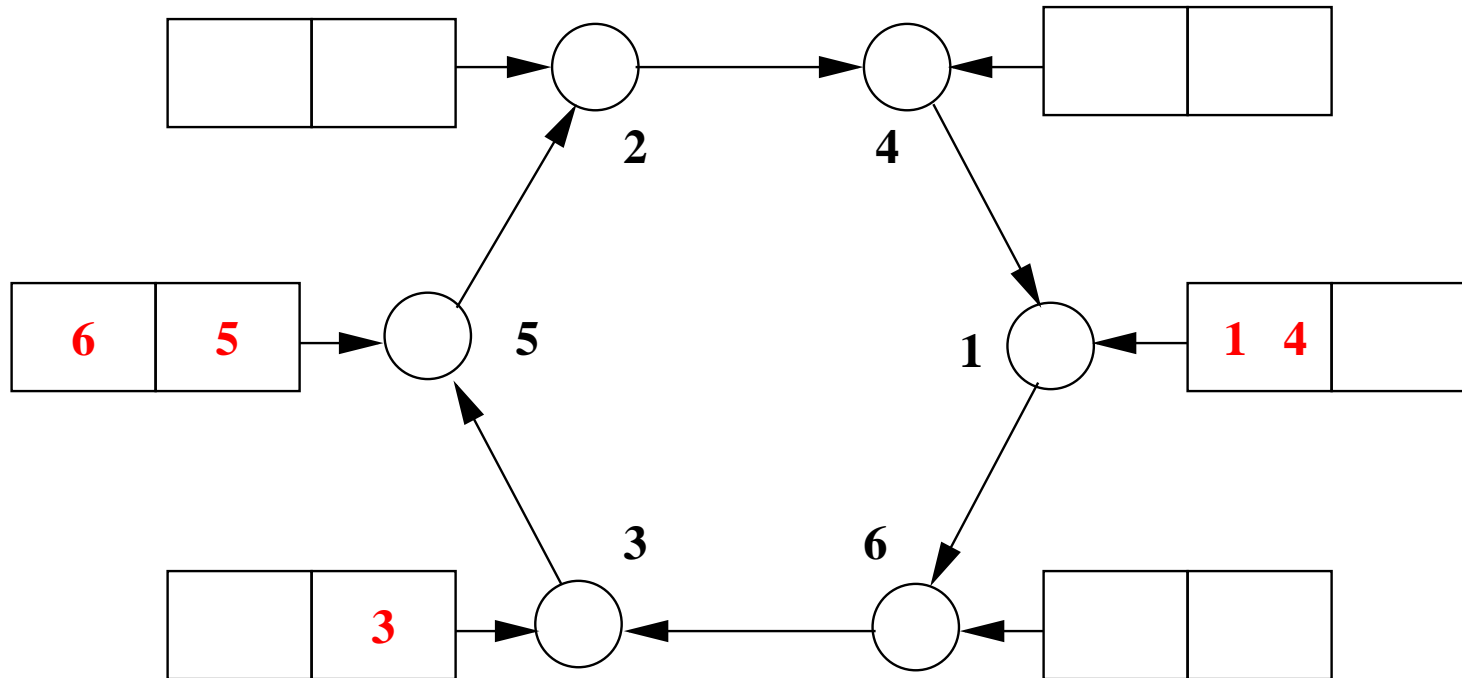
Accepting 6



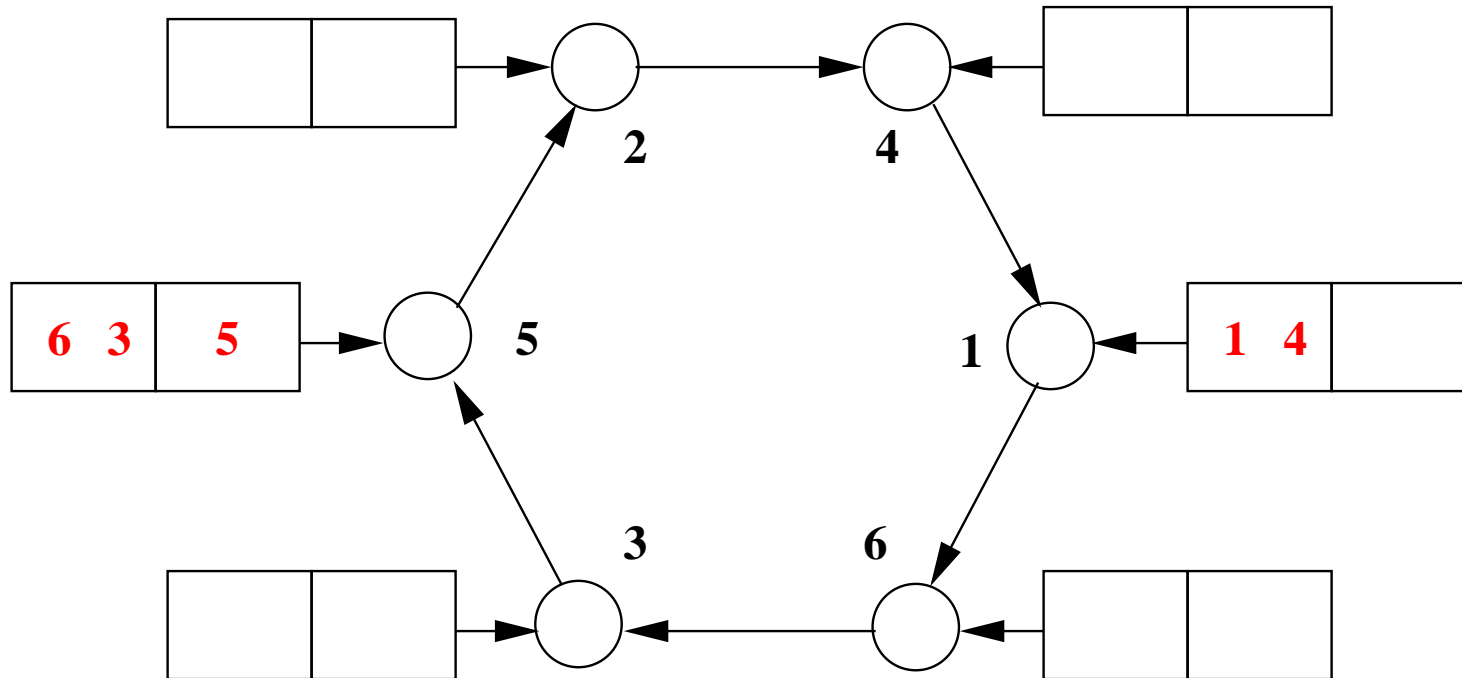
Rejecting 2



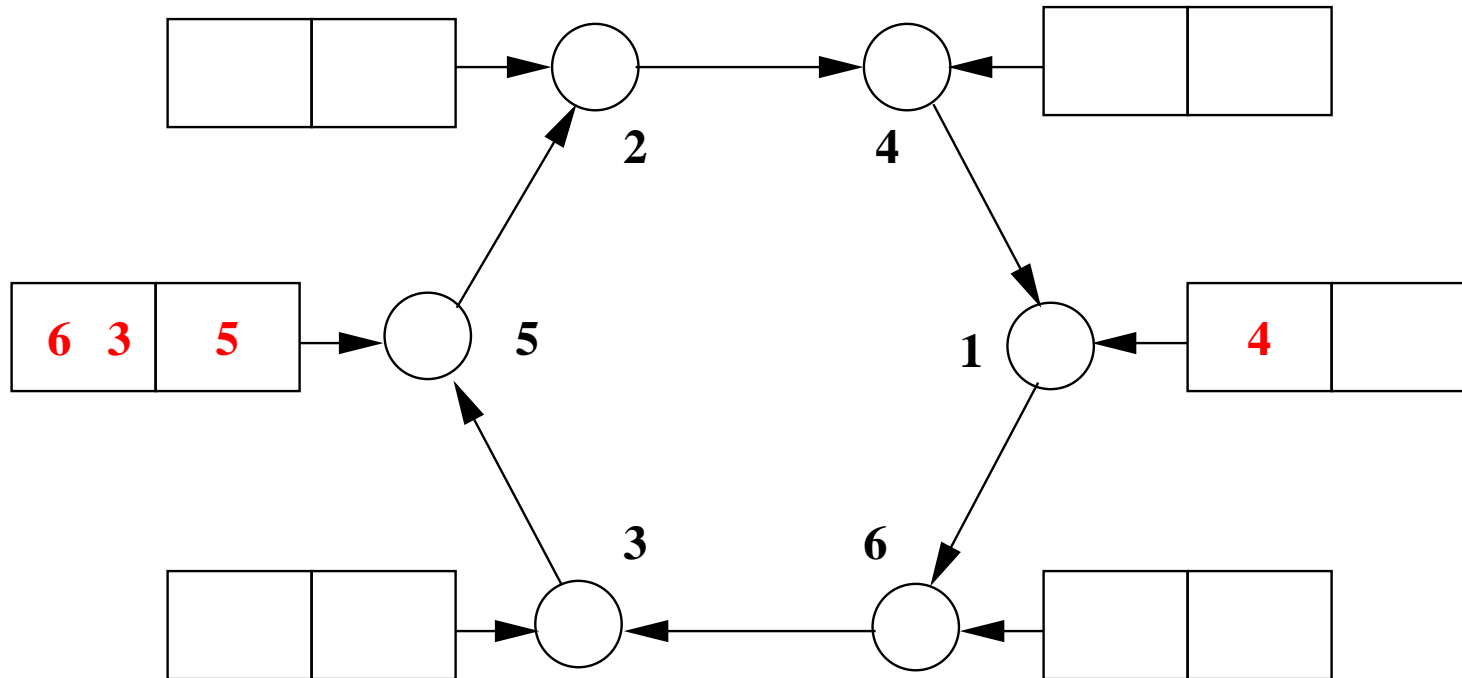
Sending 6



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)

\in	set membership operator
\mathbb{N}	set of Natural Numbers: $\{0, 1, 2, 3, \dots\}$
$a .. b$	interval from a to b : $\{a, a + 1, \dots, b\}$
$S \rightarrow T$	set of total functions from S to T
$S \leftrightarrow T$	set of partial functions from S to T

Reminder of Conventions for Modeling (2)

\cup	set-theoretic union operator
\mapsto	pair constructing operator
$\{\dots\}$	set defined in extension
\emptyset	empty set
\triangleleft	domain restriction operator

Reminder of Conventions for Modeling (3)

$\mathbb{F}_1(S)$	Non-empty set of finite subsets of S
$\mathbb{F}(S)$	Set of finite subsets of S
$\mathbb{P}_1(S)$	Non-empty set of subsets of S
$\mathbb{P}(S)$	Set of subsets of S
$\max(S)$	Maximum of a non-empty finite set of numbers

Reminder of Conventions for Modeling (4)

$S \rightsquigarrow T$	set of bijections from S to T
$S \times T$	Cartesian product of S and T
$f \triangleleft g$	overwriting operator for functions

Reminder of Conventions for Modeling (5)

dom	domain of a function
ran	range of a function
\triangleleft	domain restriction operator
\triangleleft	domain subtraction operator
$\text{id}(S)$	identity function built on the set S

Non-deterministic Assignment

```
any  < variable >  where  
      < condition >  
      ...  
then  
      < variable > := < expression >  
      ...  
end
```

A Small Theory of Rings

Set: N

Constants: nxt, itv

$$nxt \in N \mapsto N$$

$$itv \in N \times N \rightarrow \mathbb{P}(N)$$

$$\forall x \cdot (x \in N \Rightarrow itv(x, x) = \{x\})$$

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

$$\forall x \cdot (x \in N \Rightarrow itv(nxt(x), x) = N)$$