Non-deterministic Distributed Decision

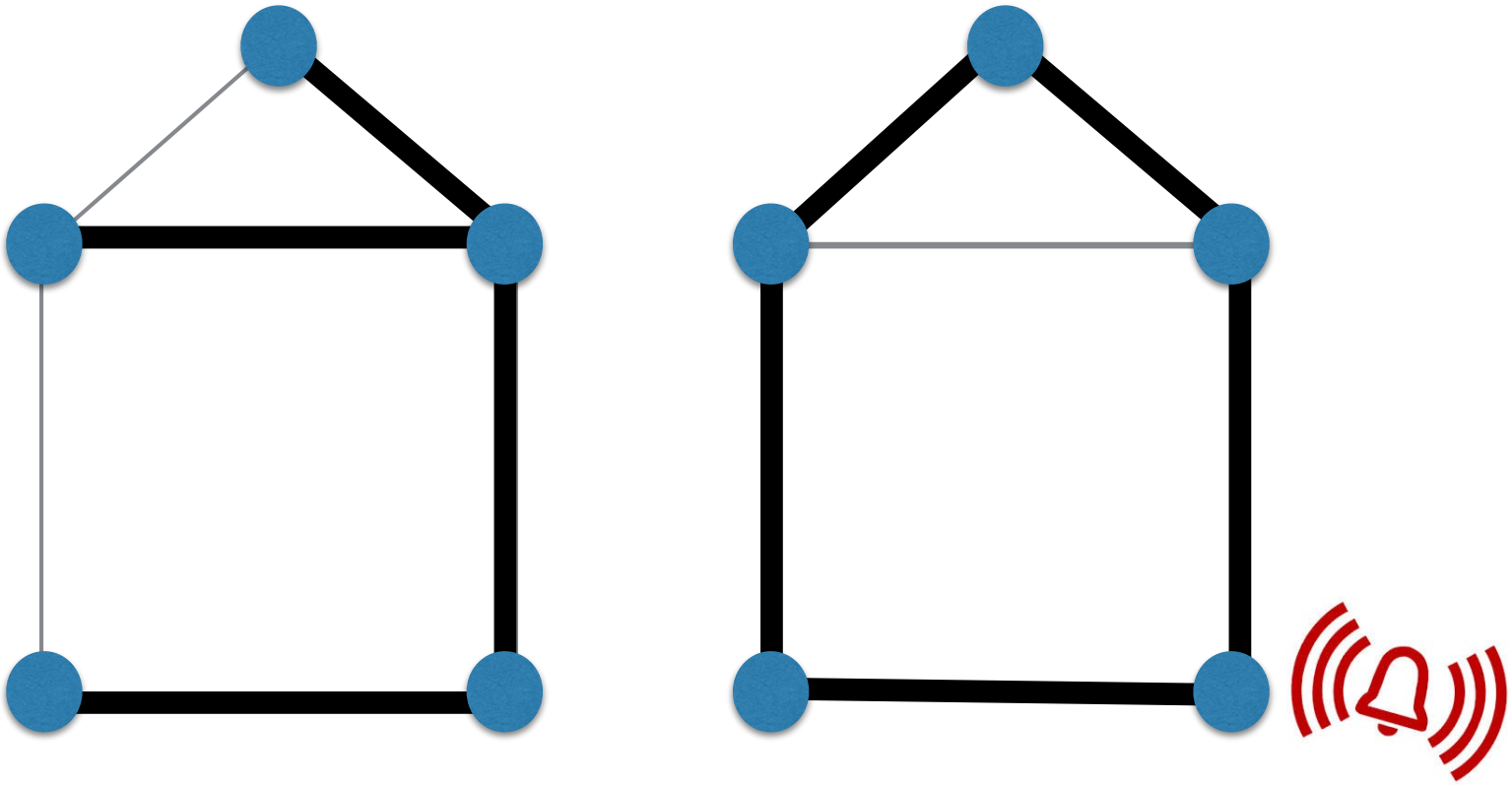
Pierre Fraigniaud

Workshop on Distributed Runtime Verification

Bertinoro, May 17-20, 2016

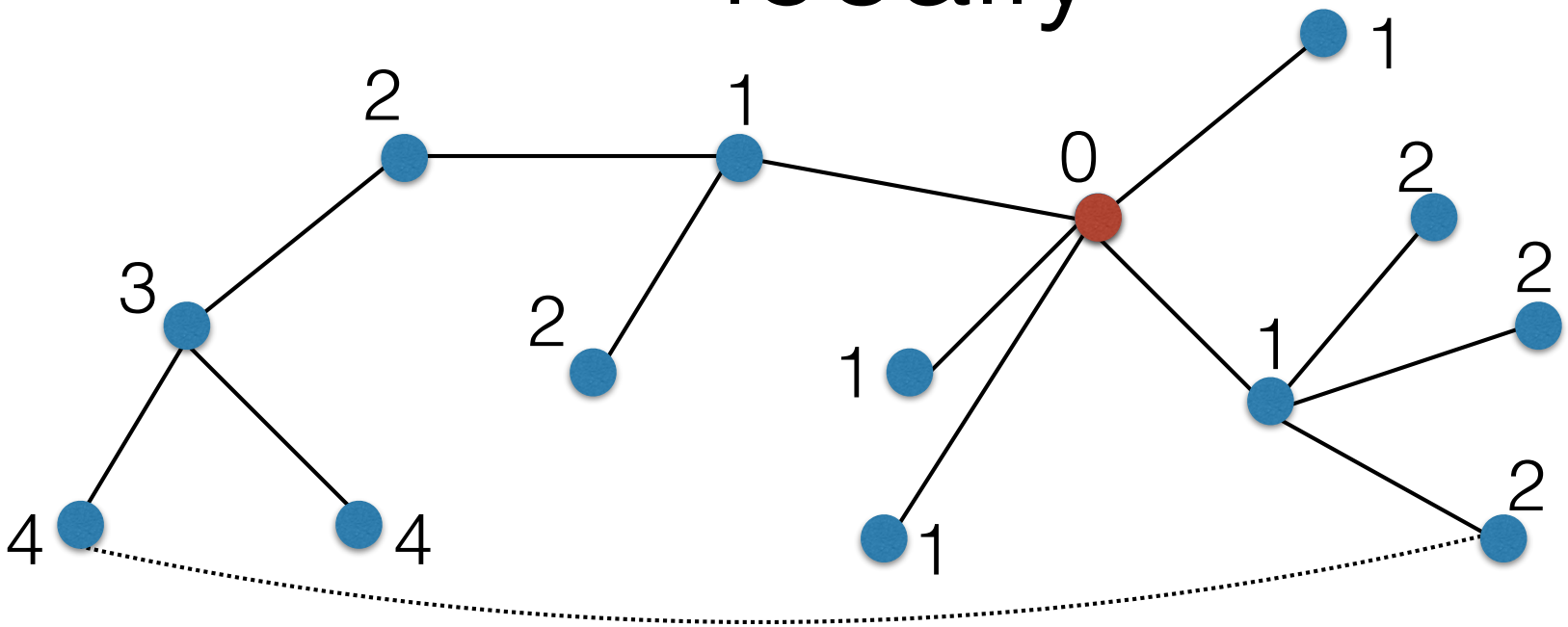
Distributed network computing (synchronous, non failures)

Fault-tolerant spanning tree construction



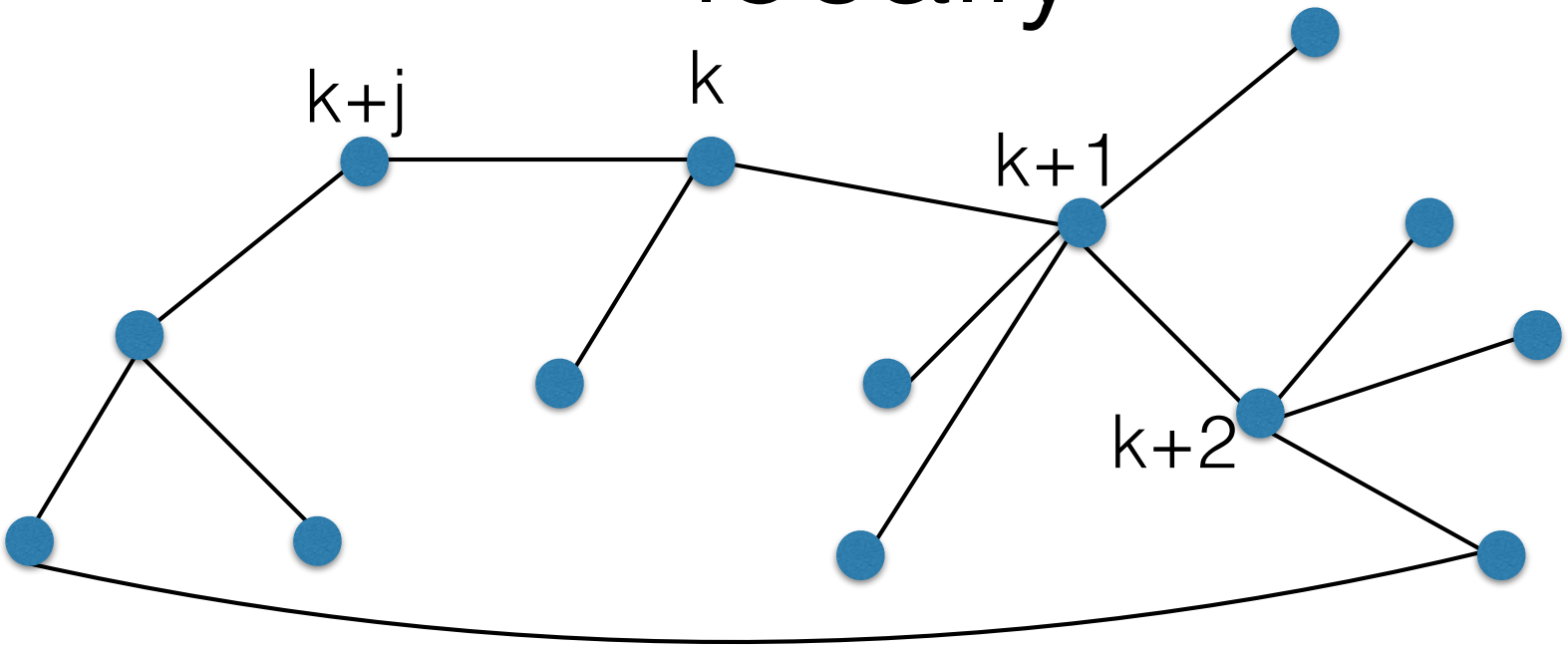
Accept if and only if **all** processes accept

Checking cycle-freeness locally



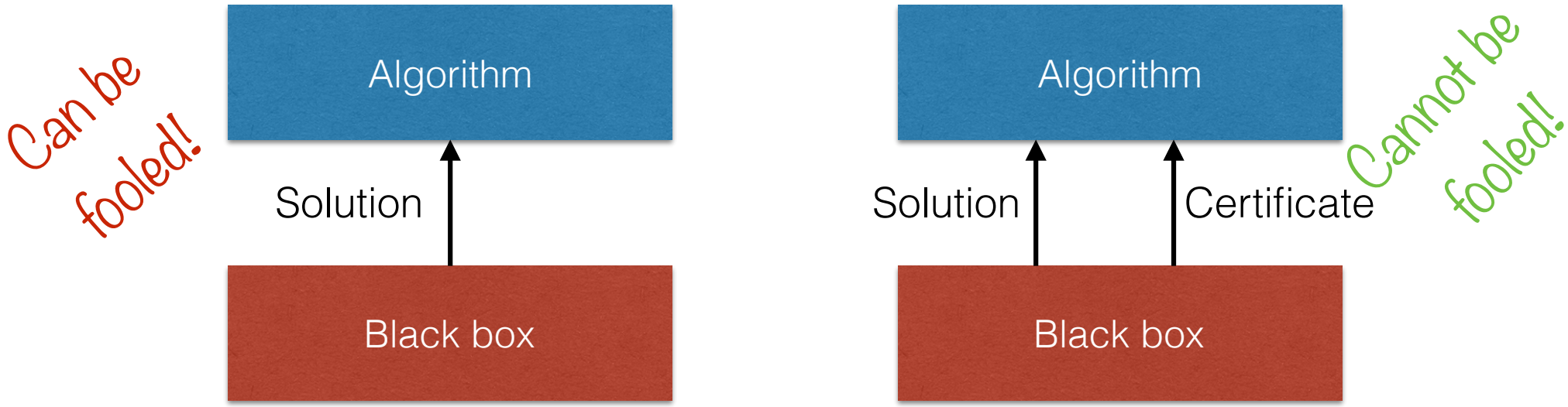
Accept if and only if **all** processes accept

Checking cycle-freeness locally



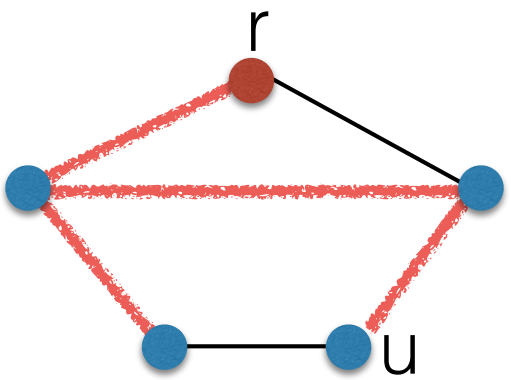
Accept if and only if **all** processes accept

A Generic Principle



- Example: Self-stabilisation (i.e., transient faults)
- Proof-labeling scheme
- E.g., spanning tree construction:

$c(u) = (ID(r), dist(u,r))$



Distributed languages

- ‘Sequential’ languages (TM)

$\Sigma=\{0,1\}$ (or any finite alphabet)

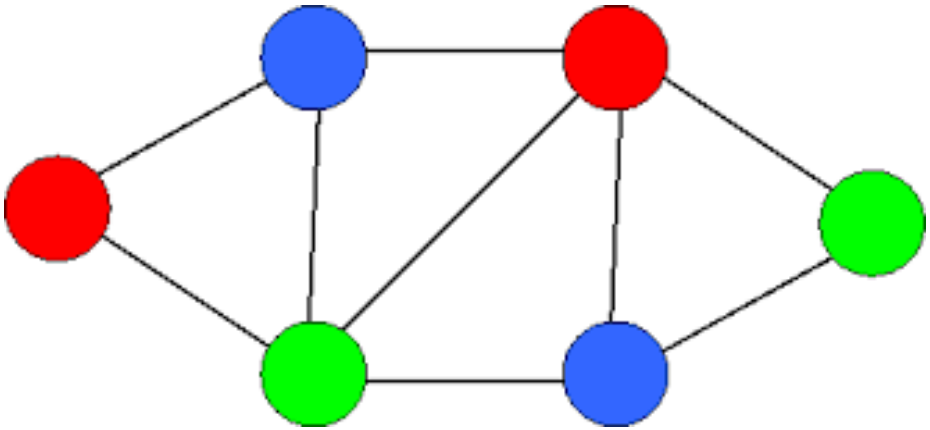
$L \subseteq \Sigma^*$

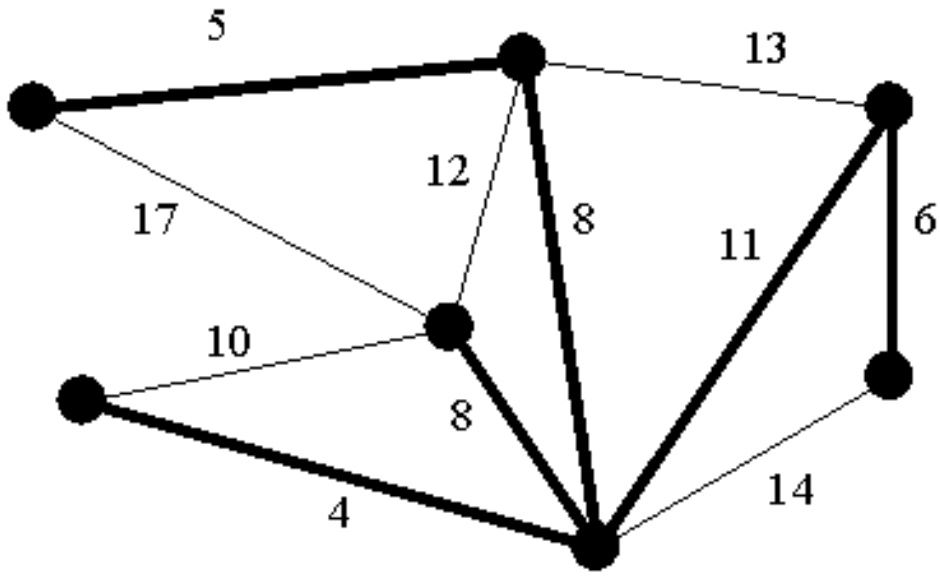
- ‘Distributed’ languages

configuration: (G,λ) where $\lambda:V(G) \rightarrow \Sigma^*$

A language L is a subset of configuration

Examples





Decision vs. Verification

Classical seq. computing:

- decision class **P**: $x \in L \Leftrightarrow A(x)$ accepts
- verification class **NP**: $x \in L \Leftrightarrow \exists y,\ A(x,y)$ accepts

Distributed netwok computing:

- distributed **decision:** $(G,\lambda) \in L \Leftrightarrow$ the system accepts (G,λ)
- distributed **verification:**

$(G,\lambda) \in L \Leftrightarrow \exists c$: the system accepts (G,λ,c)

Decision and verification is emotionally well understood in the distributed network setting

SURVEY OF DISTRIBUTED DECISION*

Laurent Feuilloley and Pierre Fraigniaud

Institut de Recherche en Informatique Fondamentale

CNRS and University Paris Diderot

Abstract

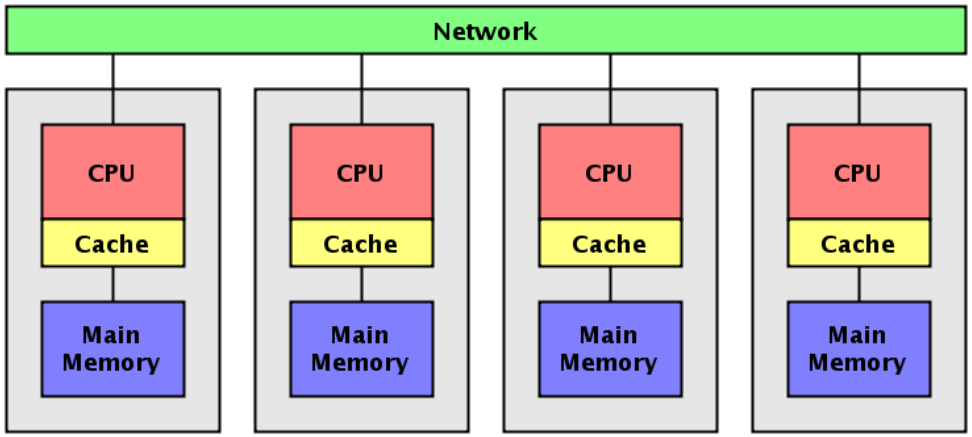
We survey the recent distributed computing literature on checking whether a given distributed system configuration satisfies a given boolean predicate, i.e., whether the configuration is legal or illegal w.r.t. that predicate. We consider classical distributed computing environments, including mostly synchronous fault-free network computing (LOCAL and CONGEST models), but also asynchronous crash-prone shared-memory computing (WAIT-FREE model), and mobile computing (FSYNC model).

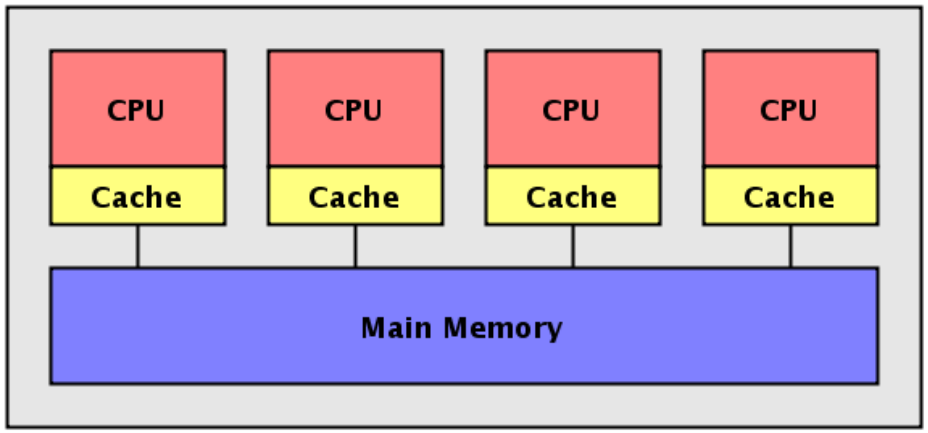
1 Introduction

The objective of this note is to survey the recent achievements in the framework of *distributed decision*: the computing entities of a distributed system aim at checking whether the system is in a legal state with respect to some boolean predicate. For

Distributed asynchronous crash-prone computing

Computing power
(asynchronous, crash-prone)



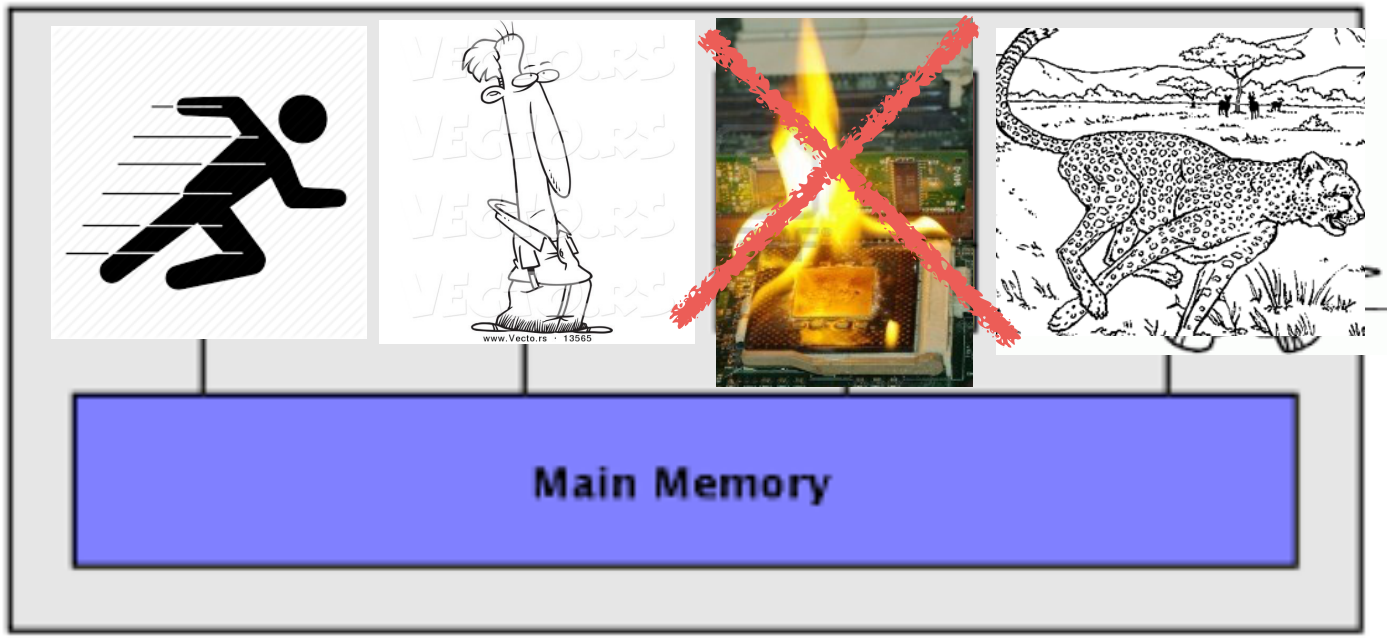


message-passing vs. shared memory

They have the same computing power

Asynchronous Fault-tolerant Computing

crash



Distributed Wait-Free Computing

Wait-Free Algorithm

```
Repeat X times
   Write what was learned so far
   Snapshot the entire shared memory
Output f(all what was learned)
```

No crashes during execution
A process may participate or not

Issues

1. Can we verify more system predicates than those we can decide?

2. Can we construct the certificates together with the construction of the solutions?

Distributed Languages Revisited

- ‘Sequential’ languages (TM)

$\sum=\{0,1\}$ (or any finite alphabet)

$$\mathbf{L \subseteq \Sigma^*}$$

- ‘Distributed’ languages

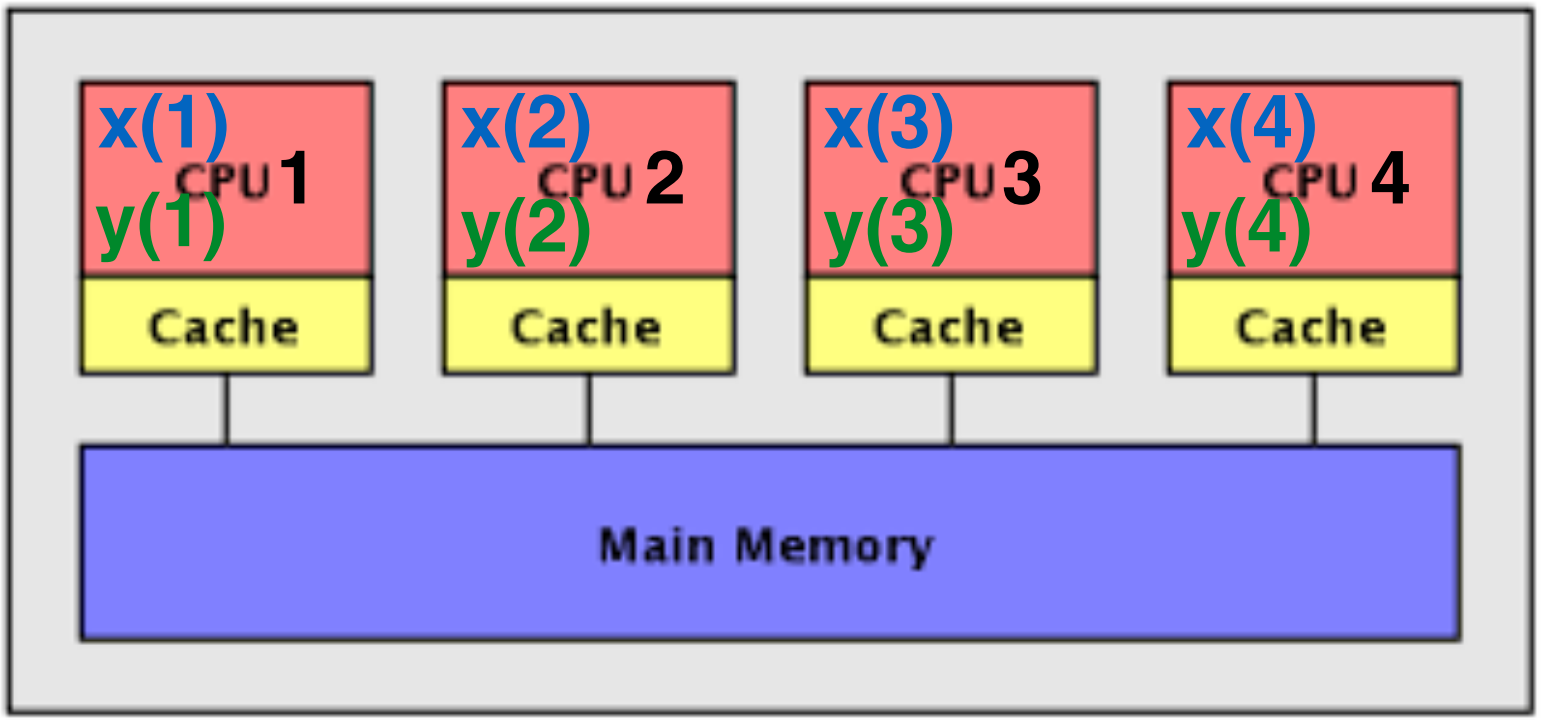
$\sum=\{0,1\}$ (or any finite alphabet)

$$\mathbf{L \subseteq U_{n\geq 1} (\Sigma^*)^n}$$

Setting

Inputs

Certificates



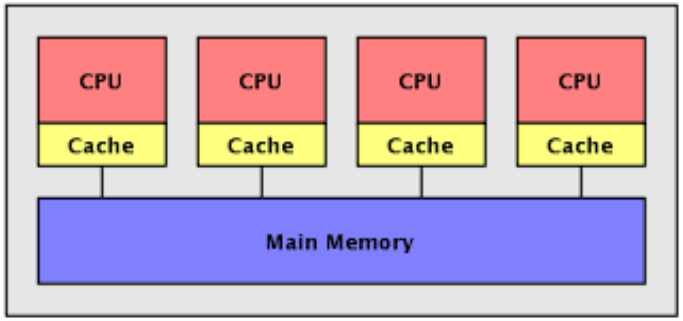
Does **x=(x(1),x(2),x(3),x(4)) satisfy P**?

Equivalently: does **x** $\in$ **L**?

Setting

Each process p has:

- an identity id(p)
- an input x(p)
- a certificate y(p)



the ‘opinion’ of p

Each process p produces an output out(p)

$x \in L \Leftrightarrow \exists y$, {out(p), p=1,…,n} is accepted

Acceptance rules

Typical example:

$$\text{out}(p) \in \{\text{true},\text{false}\}$$

$\{\text{out}(p),\ p=1,\ldots,n\}$ is accepted iff $\bigwedge_p \text{out}(p) = \text{true}$

Other examples: majority, unanimity, exclusive-or, etc.

An impossibility result

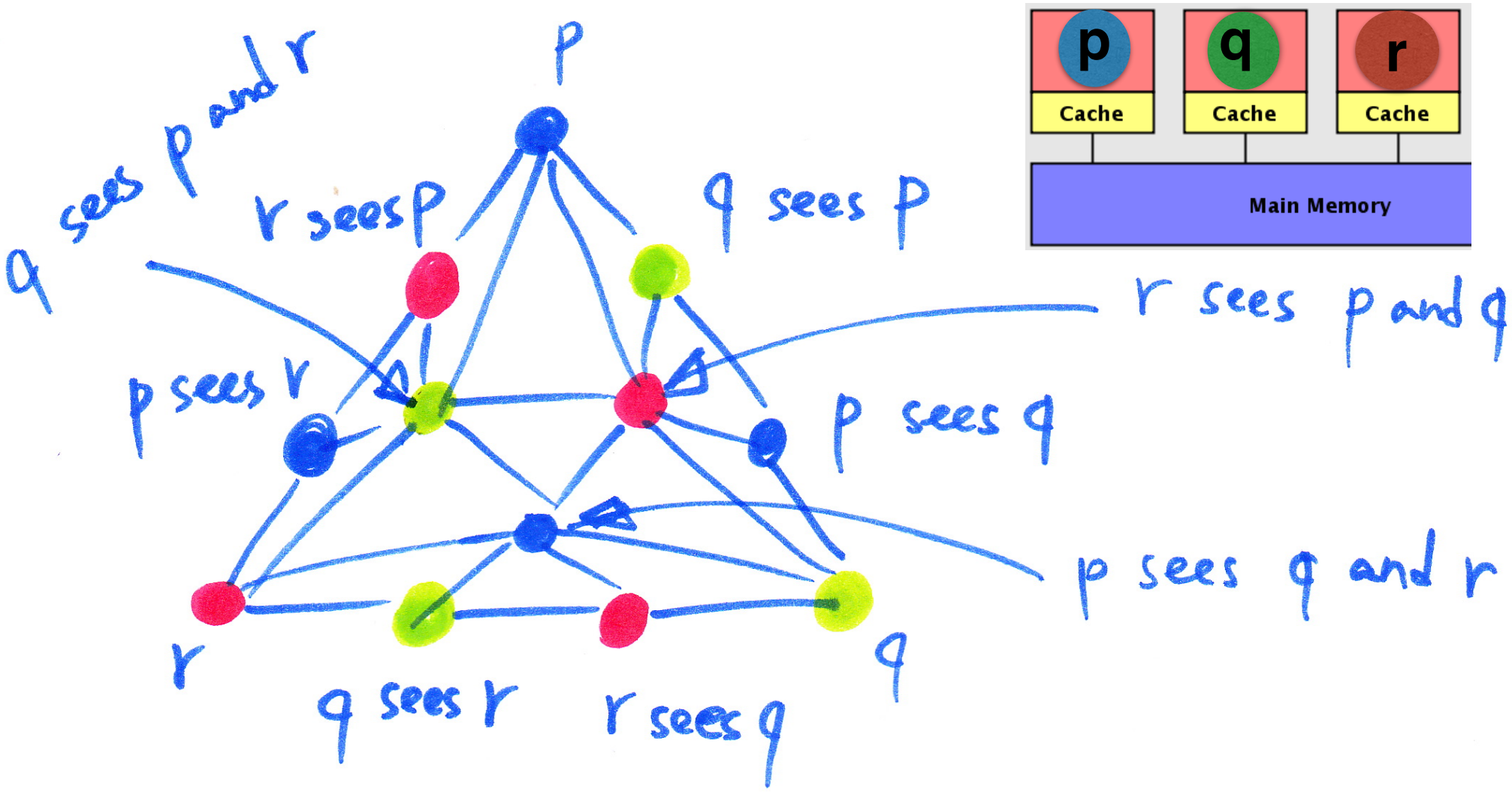
Bad news

Theorem. There are distributed languages that cannot be verified using a set of only 2 opinions, even restricted to 3 processes, and regardless to the size of the certificates.

$$\{(s_1,t_1),\ldots,(s_n,t_n)\} \in \mathcal{L} \iff \begin{cases} \forall i : (s_i,t_i) \in \{0,1\}^2 \\ \exists i : t_1 = \cdots = t_n = s_i \end{cases}$$

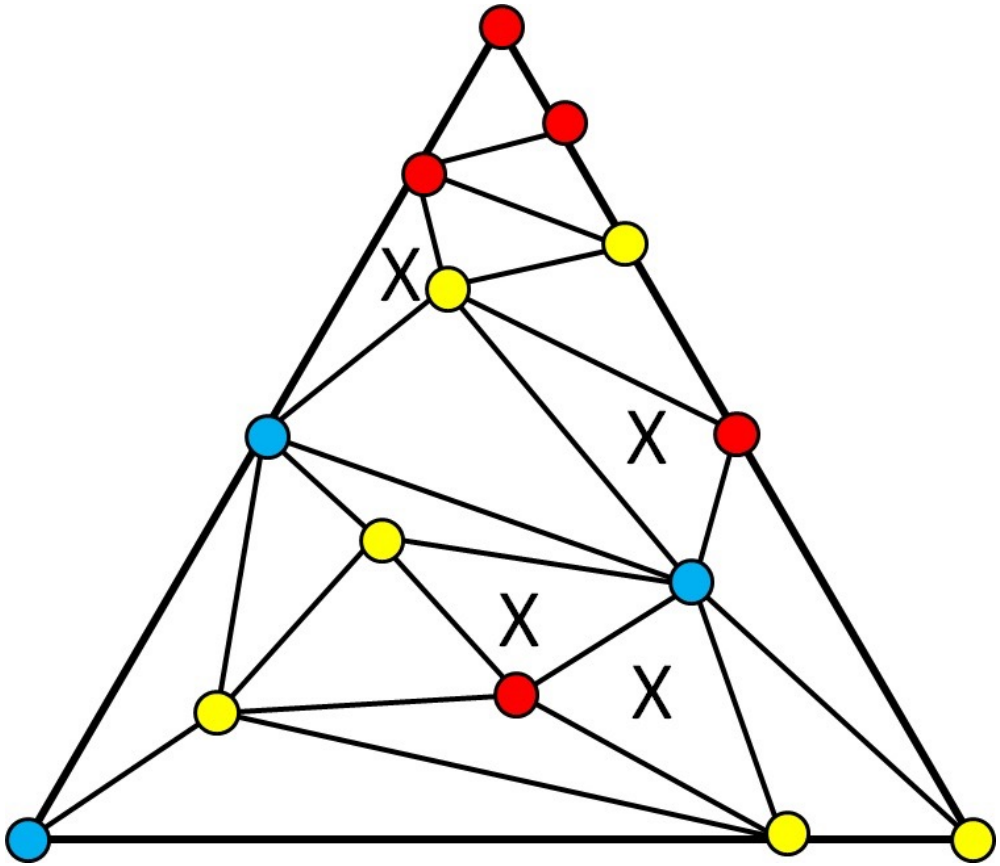
Binary consensus cannot be verified using a set of only 2 opinions, even restricted 3 processes, and regardless to the size of the certificates.

The Topological Structure of Asynchronous Computability



Drawing taken from Petr Kuznetsov's web pages

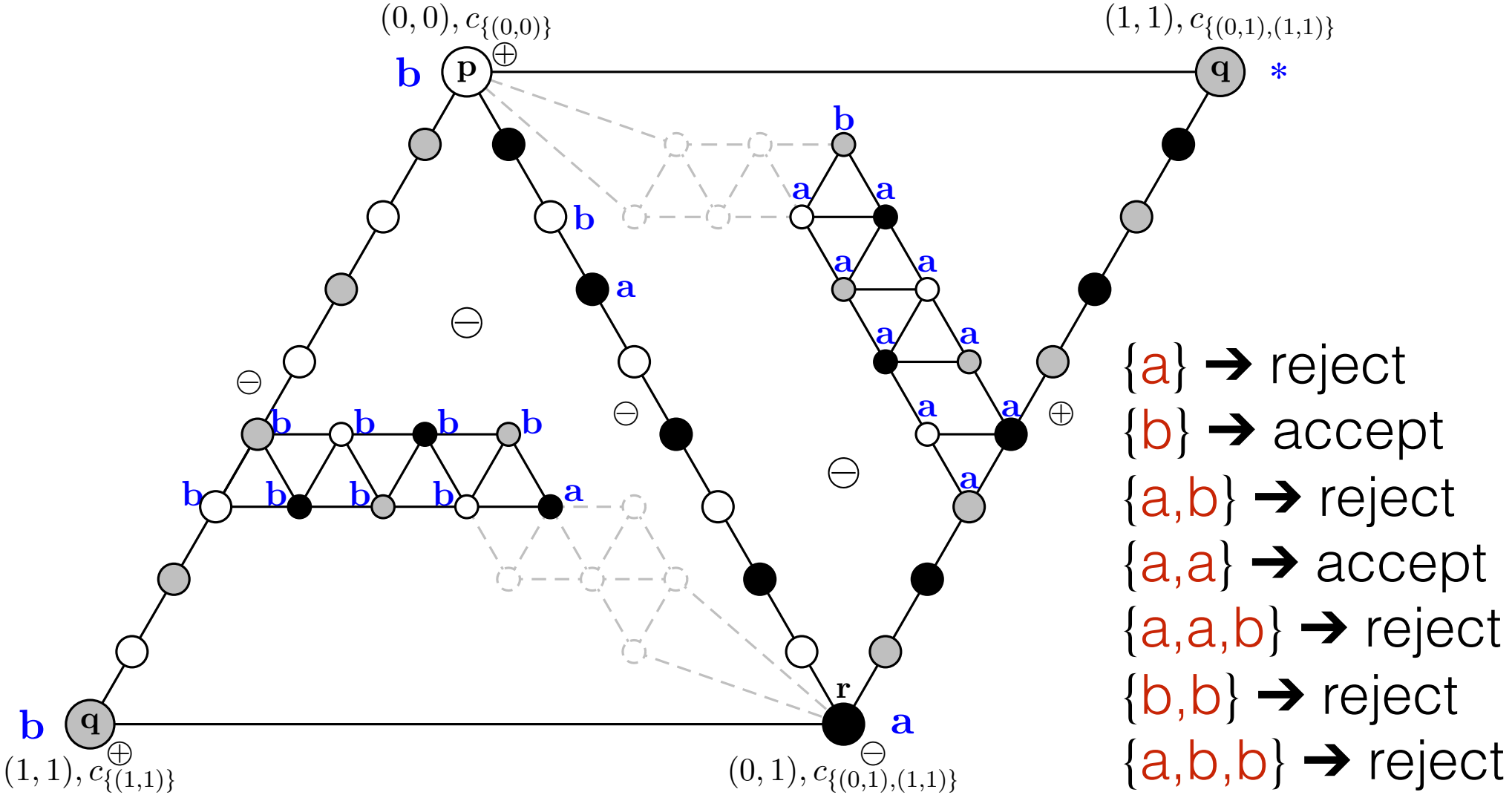
Sperner Lemma



Proof: Two opinons a and b i.e. $o_p \in \{a,b\}$

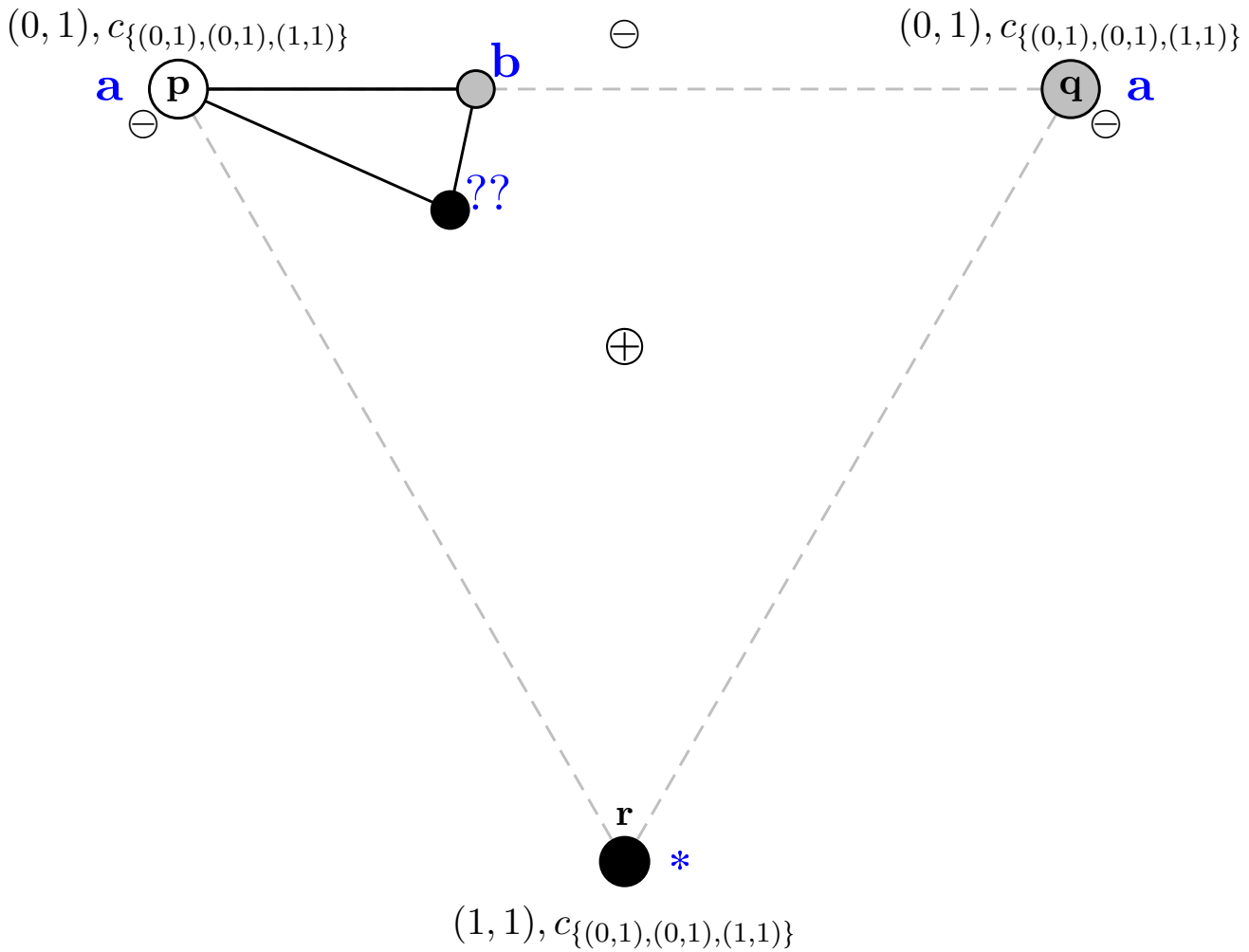
Interpretation of sets $\{o_1\}$, $\{o_1o_2\}$,and $\{o_1o_2o_3\}$

$$\{(s_1,t_1),\ldots,(s_n,t_n)\} \in \mathcal{L} \iff \begin{cases} \forall i : (s_i,t_i) \in \{0,1\}^2 \\ \exists i : t_1 = \cdots = t_n = s_i \end{cases}$$



{a} ➔ reject
{b} ➔ accept
{a,b} ➔ reject
{a,a} ➔ accept
{a,a,b} ➔ reject
{b,b} ➔ reject
{a,b,b} ➔ reject

Proof continued:



□

A verification algorithm

Good news

Theorem Every distributed language can be **verified** using a set of 3 opinions (with certificates of size O(log n) bits for n-process instances).

Recall from Sergio and Corentin's talks that **decision** may need up to n opinions!

Recall from Borzoo's talk that **decision** may need up to n logical values!

3-valued “logic”

out(p) ∈ {true,undetermined,false} = {T,U,F}

$\wedge$	T	U	F
T	T	T	F
U	T	U	F
F	F	F	F

{out(p), p=1,…,n} is accepted iff $\bigwedge_p$out(p) = true

Opinion-maker (a.k.a. 'traffic light' checker)

- certificate y(p) is the number of participating processes
- Algorithm of process p:

```
write (id(p),x(p),y(p))
snapshot memory
if #processes = y(p) then
   if x ∈ L then out(p):=true
   else out(p):=false
else
   if #processes < y(p) then
   out(p):=undetermined
   else out(p):=false
```

L must be decidable

Optimizing the certificate size

Optimisation issues

- Are $\Omega(\log n)$-bit certificates required to certify all distributed languages on n processes?
- Is verification achievable with smaller certificates?

Distributed encoding of the integers

Definition A distributed encoding of the integers is a collection of code-words providing every integer n with a code $w = (w_i)_{i=1,\ldots,n}$ in Σ^n, where Σ is a (possibly infinite) alphabet, such that: for any $k \in [1, n)$, no subwords $w' \in \Sigma^k$ of w is encoding k.

Example: Every $n \geq 1$ can be encoded by the word

$$w = (\mathrm{bin}(n), \ldots, \mathrm{bin}(n)) \in \Sigma^n \text{ with } \Sigma = \{0, 1\}^*$$

Looking for distributed encoding with ‘smaller letters’.

Basic idea

8: 1111111**0**
9: 111111**0**11
10: 11111**0**1111
11: 1111**0**111111
12: 111**0**11111111
13: 11**0**1111111111
14: 1**0**11111111111
15: **0**111111111111111
16: 11111111111111**2**
17: 1111111111111**2**11
18: 111111111111**2**1111
19: 11111111111**2**111111
20: 1111111111**2**11111111
21: 1111111111**2**1111111111

Slightly more formally

A distributed encoding of the positive integers is a pair (Σ, f) where Σ is a (possibly infinite) alphabet, and

$$f : \Sigma^* \rightarrow \{\text{true}, \text{false}\}$$

satisfying that, for every integer $n \geq 1$, there exists a word $w \in \Sigma^n$ such that

1. $f(w) = \text{true}$
2. for every subword w' of w, $f(w') = \text{false}$.

The word w is called the distributed code of n.

‘Trafic light’ checker revisited

Algorithm 1 Universal verifier with 3-valued opinions: code of the opinion maker M for process i

Require: input pair value-certificate (x_i, c_i)

1: write (i, x_i, c_i)
2: *view* $\leftarrow$ snapshot()
3: let $view = \{(j_1, x_{j_1}, c_{j_1}), \ldots, (j_k, x_{j_k}, c_{j_k})\}$
4: let $s = \big((j_1, x_{j_1}), \ldots, (j_k, x_{j_k})\big)$
5: let $c = (c_{j_1}, \ldots, c_{j_k})$ ▷ assuming $j_1 < j_2 < \cdots < j_k$
6: **if** $s \in \mathcal{L}$ **then**
7: **if** $f(c)$ **then**
8: decide true
9: **else**
10: decide undetermined
11: **end if**
12: **else** ▷ $s \notin \mathcal{L}$
13: **if** (not $f(c)$) **and** $(\exists c' \prec c : f(c'))$ **then**
14: decide false
15: **else**
16: decide undetermined
17: **end if**
18: **end if**

A slowly growing function

$A_k : \mathbb{N} \to \mathbb{N}, k \geq 1$ be the family of functions defined recursively as follows:

$$A_k(n) = \begin{cases} 2n+2 & \text{if } k = 1 \\ \underbrace{A_{k-1}(\ldots A_{k-1}(0))}_{n+1} = A_{k-1}^{(n+1)}(0) & \text{otherwise.} \end{cases}$$

$$Ack(n) = A_n(1))$$

Let $F : \mathbb{N} \to \mathbb{N}$ be the function:

$$F(k) = A_1(A_2(\ldots(A_{k-1}(A_k(0))))) + 1.$$

Finally, let $\alpha : \mathbb{N} \to \mathbb{N}$ be the function:

$$\alpha(k) = \min\{i \geq 1 : F^{(i)}(1) > k\}.$$

$$\begin{aligned} F^{(n)}(1) &= A_1(A_2(\ldots A_{F^{(n-1)}(1)}(0))) \\ &> A_1(A_2(\ldots A_{F^{(n-1)}(1)-1}(2))) \\ &> A_1(A_2(\ldots A_{n-1}(1))) \\ &> A_{n-1}(1) \\ &= Ack(n-1). \end{aligned}$$

$$
\begin{aligned}
\mathcal{S}^{(1)} &= 0 \text{ (1st bad sequence starts)} \\
\mathcal{S}^{(2)} &= 11 \text{ (1st bad sequence ends)} \\
\mathcal{S}^{(3)} &= 000 \text{ (2nd bad sequence starts)} \\
\mathcal{S}^{(4)} &= 0110 \\
\mathcal{S}^{(5)} &= 11010 \\
\mathcal{S}^{(6)} &= 101011 \\
\mathcal{S}^{(7)} &= 0101111 \\
\mathcal{S}^{(8)} &= 11111100 \\
\mathcal{S}^{(9)} &= 111110011 \\
\mathcal{S}^{(10)} &= 1111001111 \\
\mathcal{S}^{(11)} &= 11100111111 \\
\mathcal{S}^{(12)} &= 110011111111 \\
\mathcal{S}^{(13)} &= 1001111111111 \\
\mathcal{S}^{(14)} &= 00111111111111 \\
\mathcal{S}^{(15)} &= 111111111111110 \\
\mathcal{S}^{(16)} &= 1111111111111011 \\
\mathcal{S}^{(17)} &= 11111111111101111 \\
\mathcal{S}^{(18)} &= 111111111110111111 \\
\vdots &\quad \vdots \qquad\qquad \vdots \\
\mathcal{S}^{(29)} &= 01111111111111111111111111111 \\
\mathcal{S}^{(30)} &= 11111111111111111111111111111 \text{ (2nd bad sequence ends)} \\
\mathcal{S}^{(31)} &= 00000000000000000000000000000 \text{ (3rd bad sequence starts)} \\
\mathcal{S}^{(32)} &= 00000000000000000000000000110 \\
\mathcal{S}^{(33)} &= 00000000000000000000000011010 \\
\mathcal{S}^{(34)} &= 00000000000000000000000001101010 \\
\vdots &\quad \vdots \qquad\qquad \vdots
\end{aligned}
$$

			$(x^{(i)})$, μ_i
$M^{(1)}$	=	0000	$(0,0,0,0),0$
$M^{(2)}$	=	00110	$(0,0,2),0$ (projection operation)
$M^{(3)}$	=	011010	$(0,2,1),0$ (maximum operation)
$M^{(4)}$	=	1101010	$(2,1,1),0$
$M^{(5)}$	=	10101011	$(1,1,1),2$
$M^{(6)}$	=	010101111	$(0,1,1),4$
$M^{(7)}$	=	1111110010	$(6,0,1),0$
$M^{(8)}$	=	11111001011	$(5,0,1),2$
$M^{(9)}$	=	111100101111	$(4,0,1),4$
$M^{(10)}$	=	1110010111111	$(3,0,1),6$
$M^{(11)}$	=	11001011111111	$(2,0,1),8$
$M^{(12)}$	=	100101111111111	$(1,0,1),10$
$M^{(13)}$	=	0010111111111111	$(0,0,1),12$
$M^{(14)}$	=	01111111111111100	$(0,14,0),0$
$M^{(15)}$	=	110111111111111100	$(2,13,0),0$
$M^{(16)}$	=	101111111111110011	$(1,13,0),2$
$M^{(17)}$	=	011111111111001111	$(0,13,0),4$
$M^{(18)}$	=	11111101111111111100	$(6,12,0),0$
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
$M^{(24)}$	=	011111111111100111111111111	$(0,12,0),12$
$M^{(25)}$	=	1111111111111101111111111100	$(14,11,0),0$
⋮	⋮	⋮	⋮
			$(0,0,0),A_3(2)-2$
			$(0,A_3(2)),0$ (projection operation)
⋮	⋮	⋮	⋮
			$(0,0),A_2(A_3(2))-2$
			$(A_2(A_3(2))),0$ (projection operation)
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
$M^{(F(4)-5)}$	=	01111111111.....111111111111111111	$(0),A_1(A_2(A_3(2)))-2$
$M^{(F(4)-4)}$	=	11111111111.....1111111111111111111	$(),A_1(A_2(A_3(2)))$ (projection operation)

Certification with very, very,…, very small certificates

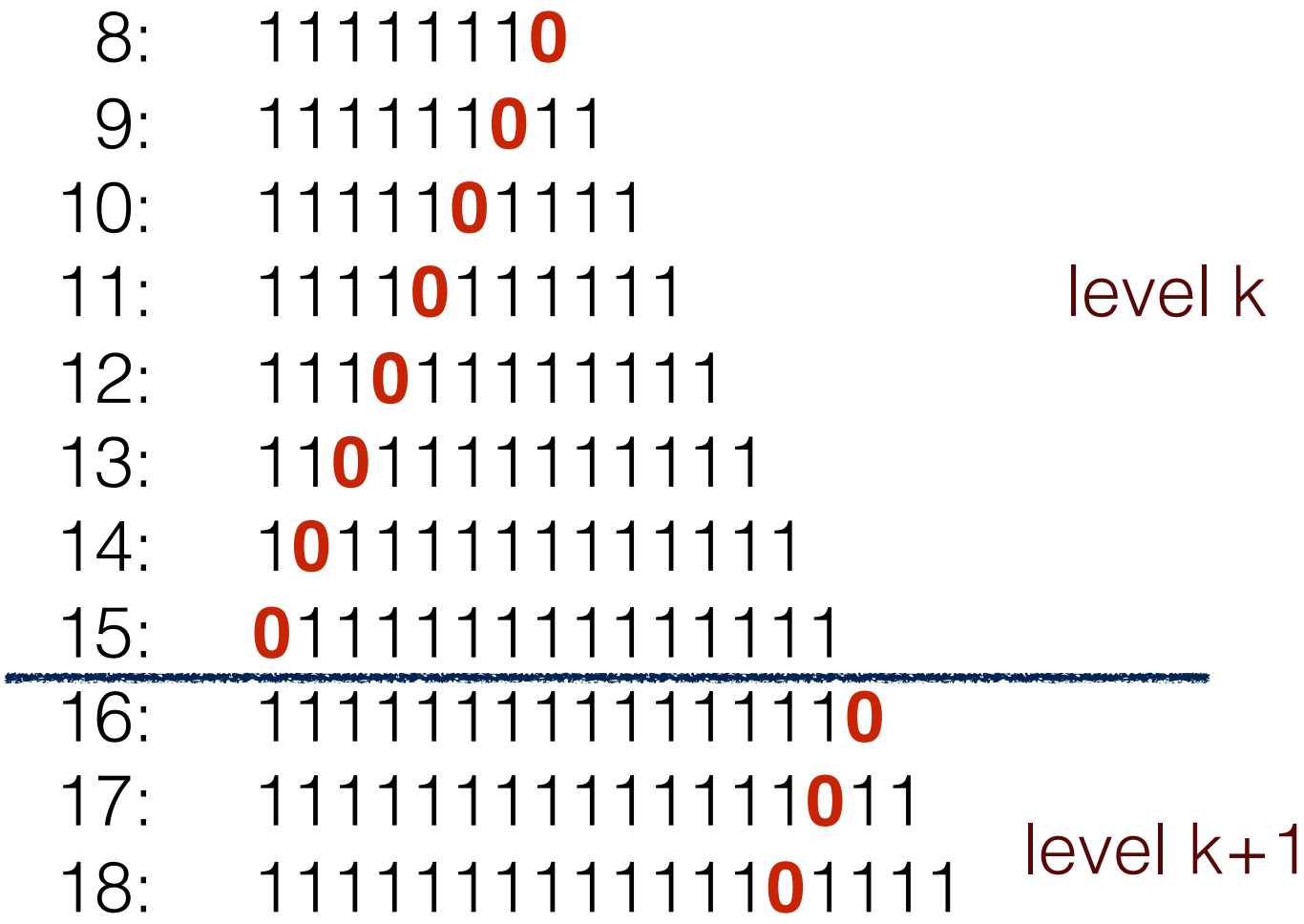
Theorem There is a distributed encoding (Σ,f) of the positive integers which encodes the first n integers using words on an alphabet with symbols in $O(\log \alpha(n))$ bits.

Corollary Every distributed language can be verified using 3 opinions, with certificates of size $O(\log \alpha(n))$ bits for n-process instances (cf. Algorithm 1).

Remark (well-quasi-ordering) Let Σ be a finite alphabet, and suppose that (Σ,f) is a distributed encoding of the integers in $[1, n]$. Then $n \leq g(1)$ where g is a multiply-recursive function.

Duality

Corollary Every language can be verified with 1-bit certificates (using $O(\alpha(n))$ opinions for n-dimensional instances).



Algorithm

out(p)=(val(p),level(p))

{out(p), p=1,…,n} is accepted iff

(1) ∃p, val(p) = true and (2) ∀p'≠p, level(p')≤level(p)

resemble LTL_k

Algorithm 2 Universal verifier with 1-bit certificate: code of the opinion maker M for process i

Require: input pair value-certificate (x_i, c_i)

1: write (i, x_i, c_i)
2: *view* $\leftarrow$ snapshot()
3: let $view = \{(j_1, x_{j_1}, c_{j_1}), \ldots, (j_k, x_{j_k}, c_{j_k})\}$
4: let $s = ((j_1, x_{j_1}), \ldots, (j_k, x_{j_k}))$
5: let $c = (c_{j_1}, \ldots, c_{j_k})$ ▷ assuming $j_1 < j_2 < \cdots < j_k$
6: **if** $s \in \mathcal{L}$ **then**
7: **if** $c = \mathcal{S}^{(k)}$ **then** ▷ $k = |c|$
8: decide (true, level(k))
9: **else**
10: decide (false, level(k))
11: **end if**
12: **else** ▷ $s \notin \mathcal{L}$
13: **if** $\exists (s', c') : s' \subset s,\ s' \in \mathcal{L},\ c' \prec c,\ c' = \mathcal{S}^{(|c'|)}$, and level($|c'|$) = level($k$)) **then**
14: decide (false, level(k) + 1)
15: **else**
16: decide (false, level(k))
17: **end if**
18: **end if**

Conclusion

Conclusion

- What is the nature of a certificate in shared-memory crash-prone asynchronous systems? Can they be produced by the processes themselves (e.g., using failure detector)?

- Decision problems in other asynchronous distributed computing models (e.g., t-resilient, message-passing, etc.).

- **Verification vs. decision in runtime verification?**