Knowledge and Distributed Coordination

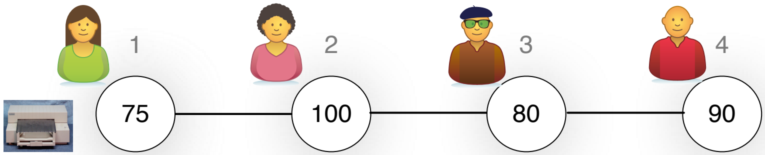
Yoram Moses

Technion

Outline

- Indistinguishability and knowledge
- Modeling knowledge
- The Knowledge of Preconditions principle
- Knowledge and coordination
- Applications

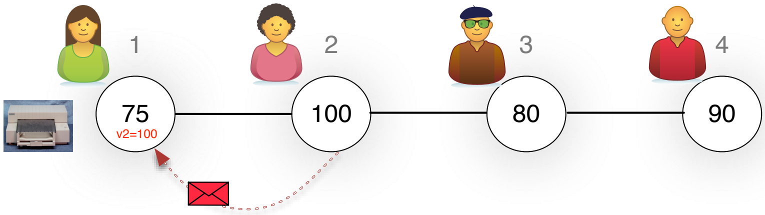
Indistinguishability in Computing the Maximum (CtM)



- Each node i has an initial value v_i
- Agent 1 must print the maximal value
- After receiving "$v_2 = 100$" Agent 1 has the maximum.

  Can she act?

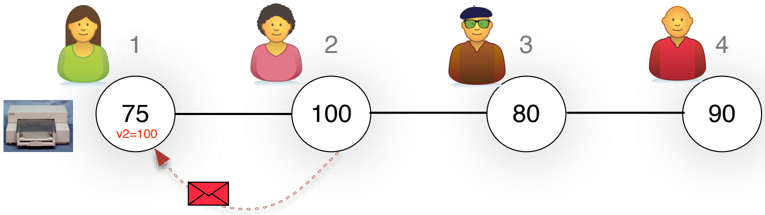
Indistinguishability in Computing the Maximum (CTM)



- Each node i has an initial value v_i
- Agent 1 must print the maximal value
- After receiving "$v_2 = 100$" Agent 1 has the maximum.

  Can she act?

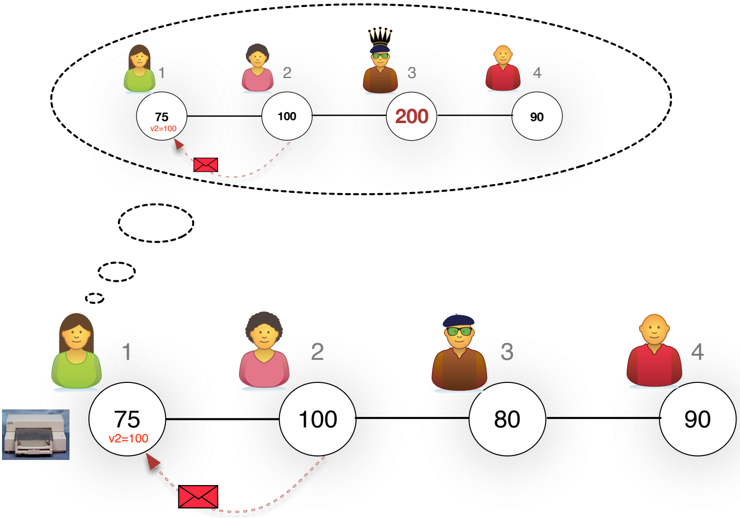
Indistinguishability in Computing the Maximum (CtM)



- Each node i has an initial value v_i
- Agent 1 must print the maximal value
- After receiving "$v_2 = 100$" Agent 1 has the maximum.

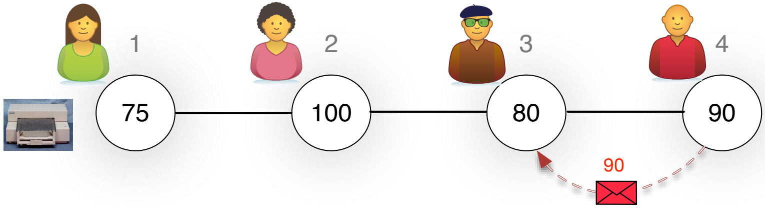
  Can she act?

Indistinguishability in Computing the Maximum (CTM)



No!

Collecting Values

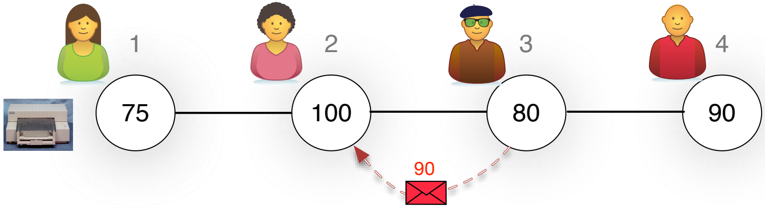


Collecting all values is not necessary

Collecting all values is not sufficient

if more participants are possible

Collecting Values

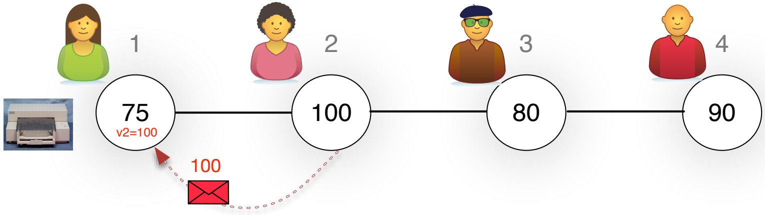


Collecting all values is not necessary

Collecting all values is not sufficient

if more participants are possible

Collecting Values

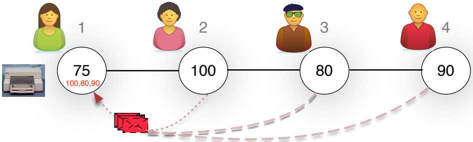


Collecting all values is not necessary

Collecting all values is not sufficient

if more participants are possible

Collecting Values

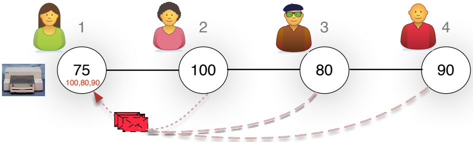


Collecting all values is not necessary

Collecting all values is not sufficient

if more participants are possible

Collecting Values

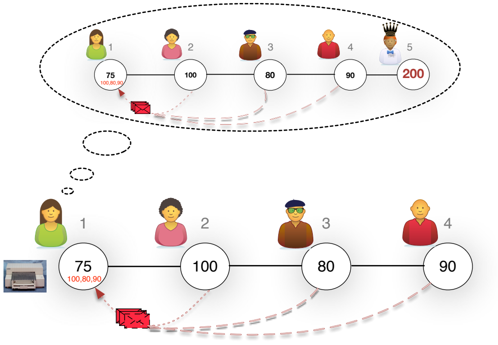


Collecting all values is not necessary

Collecting all values is not sufficient...

if more participants are possible

Collecting Values



Collecting all values is not necessary

Collecting all values is not sufficient…

if more participants are possible

What is CTM about?

Not collecting values!

A process takes the same actions at indistinguishable points

Its actions depend on its local state.

What is CTM about?

Not collecting values!

Indistinguishability

A process takes the same actions at indistinguishable points

Its actions depend on its local state.

What is CTM about?

Not collecting values!

Indistinguishability

A process takes the same actions at indistinguishable points

Its actions depend on its local state.

What is CTM about?

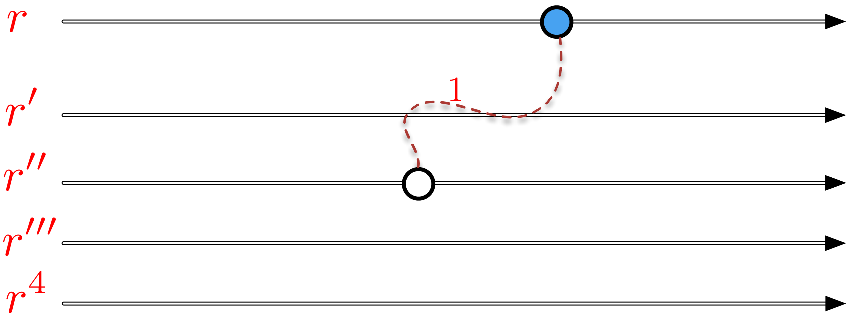
Not collecting values!

Indistinguishability

A process takes the same actions at indistinguishable points

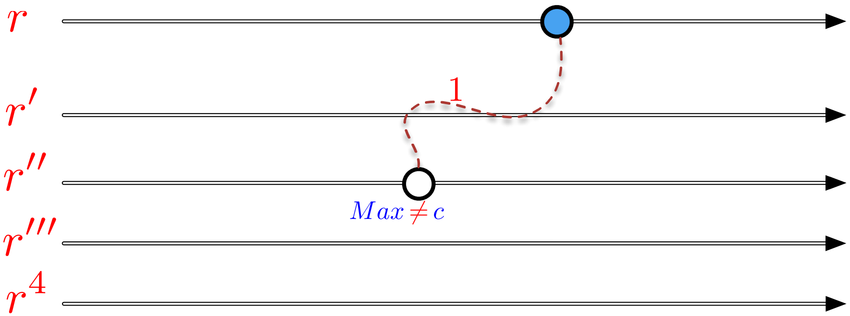
Its actions depend on its local state.

In Pictures



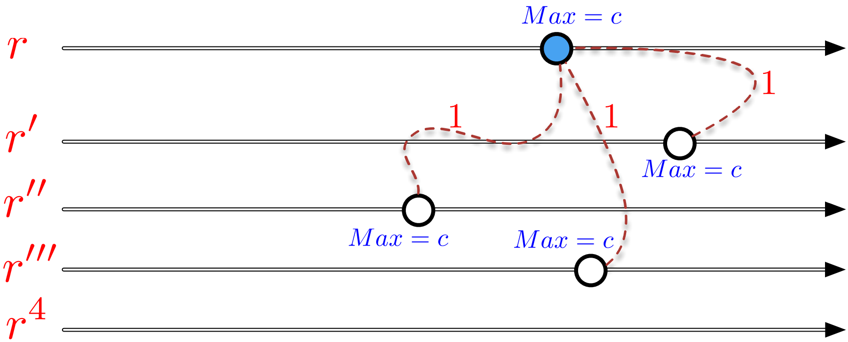
Agent 1 can have the same state in different runs of the protocol

In Pictures



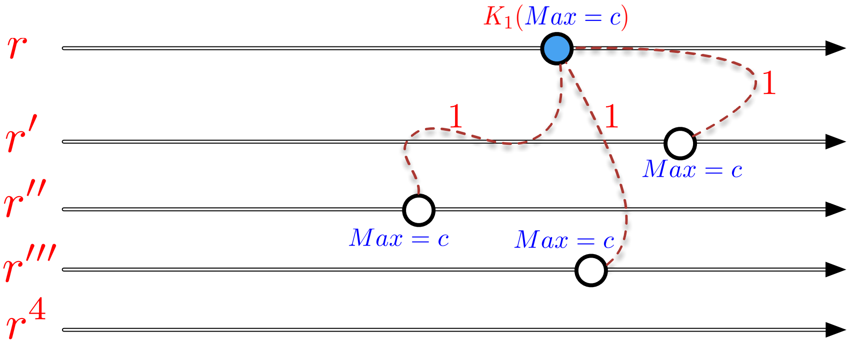
Printing c is precluded by indistinguishability

In Pictures



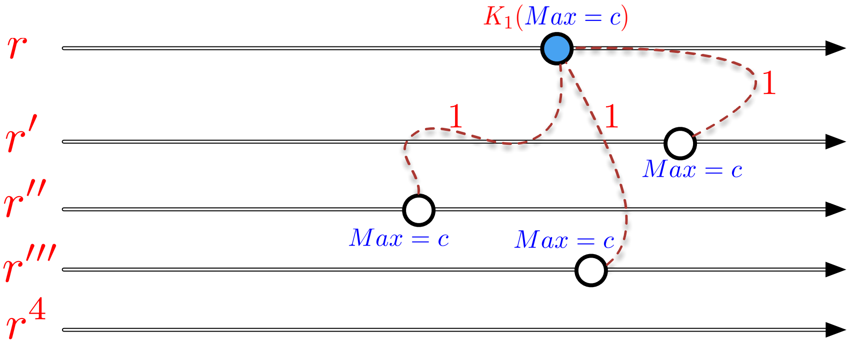
Printing c is allowed iff $Max = c$ at all indistinguishable points

In Pictures



Printing c is allowed iff $Max = c$ at all indistinguishable points

In Pictures



Printing c is allowed iff agent 1 knows that $Max = c$

Knowledge in CTM

Knowing that $Max = c$ is necessary and sufficient for printing c

Knowledge in CTM

Knowing that $Max = c$ can depend on:

- Messages received
- The protocol
- The possible initial values
- Network topology
- Timing guarantees re: communication, synchrony, activation
- Possibility of failures, ...

Problem Specifications and Necessary Conditions

Cash from the ATM:

$$\text{Dispense}(\$100) \quad \Rightarrow \quad \text{good credit}$$

Problem Specifications and Necessary Conditions

Agreement Protocols:

$$\texttt{decide}_i(v) \quad \Rightarrow \quad \text{nobody decides } v' \neq v$$

Problem Specifications and Necessary Conditions

Autonomous Cars:

$$\text{Enter\_intersection} \quad \Rightarrow \quad \text{no cross-traffic}$$

Problem Specifications and Necessary Conditions

Computing the Max:

$$\text{print}(c) \quad \Rightarrow \quad \text{Max} = c$$

and we have seen

$$\text{print}(c) \quad \Rightarrow \quad K_1(\text{Max} = c)$$

Problem Specifications and Necessary Conditions

Computing the Max:

$$\text{print}(c) \quad \Rightarrow \quad \text{Max} = c$$

and we have seen

$$\text{print}(c) \quad \Rightarrow \quad K_1(\text{Max} = c)$$

Problem Specifications and Necessary Conditions

Computing the Max:

$$\mathrm{print}(c) \quad \Rightarrow \quad \mathrm{Max} = c$$

and we have seen

$$\mathrm{print}(c) \quad \Rightarrow \quad K_1(\mathrm{Max} = c)$$

The **Knowledge of Preconditions** Principle (**K*o*P**)

If performing $\alpha \Rrightarrow \varphi$

Then $\mathtt{i}$ performs $\alpha \Rrightarrow \boldsymbol{K}_{\mathtt{i}}\varphi$

An essential connection between knowledge and action

Let's Prove it

The **Knowledge of Preconditions** Principle (**K*o*P**)

If performing α $\Rrightarrow$ φ

Then i performs α $\Rrightarrow$ $K_i\varphi$

An essential connection between knowledge and action

Let's Prove it

The **Knowledge of Preconditions** Principle (**K*o*P**)

If performing $\alpha \Rrightarrow \varphi$

Then i performs $\alpha \Rrightarrow K_{i}\varphi$

An essential connection between knowledge and action

Let's Prove it

The **Knowledge of Preconditions** Principle (**K*o*P**)

If	performing $\alpha \Rightarrow \varphi$
Then	$\mathtt{i}$ performs $\alpha \Rightarrow K_{\mathtt{i}}\varphi$

An essential connection between knowledge and action

Let's Prove it

The **Knowledge of Preconditions** Principle (**K*o*P**)

If performing $\alpha \Rrightarrow \varphi$

Then $\mathtt{i}$ performs $\alpha \Rrightarrow K_{\mathtt{i}}\varphi$

An essential connection between knowledge and action

Let's Prove it

A Theory of Knowledge in Distributed Systems

A three decades old theory of knowledge is based on

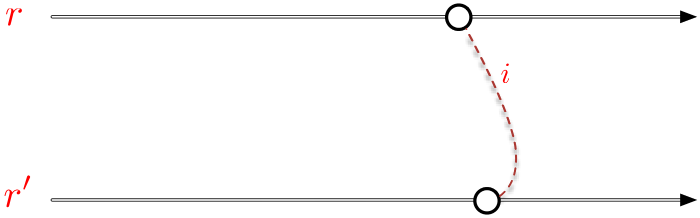
- Halpern and M. [1984]
- Parikh and Ramanujam [1985]
- Chandy and Misra [1986]

- Fagin *et al.* [1995], Reasoning about Knowledge
- and earlier Kripke 1950's, Hintikka [1962], Aumann [1976]

Basic notion: Indistinguishability

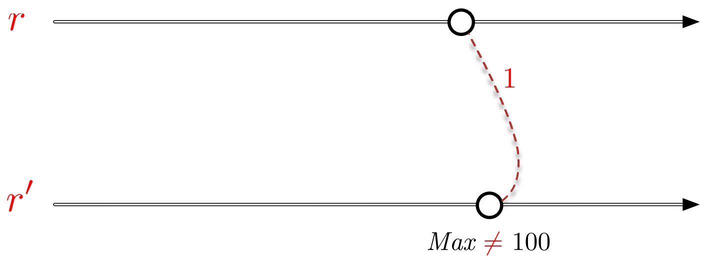
r

Basic notion: Indistinguishability

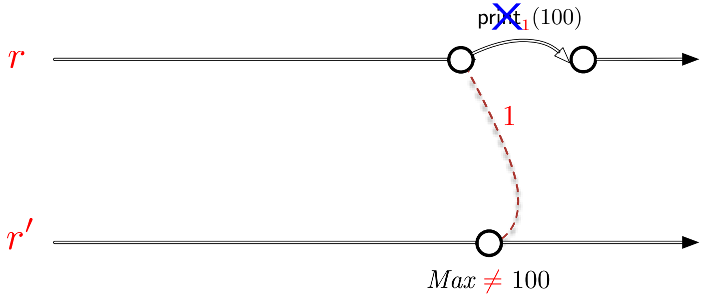


i has the same state at both points

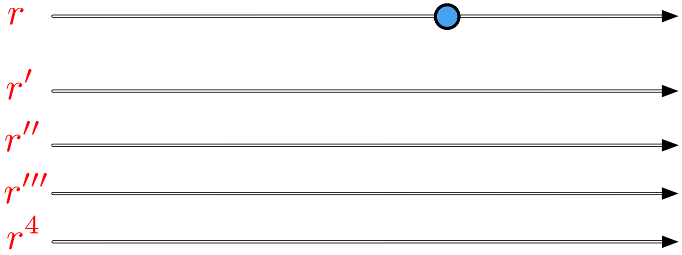
Basic notion: Indistinguishability



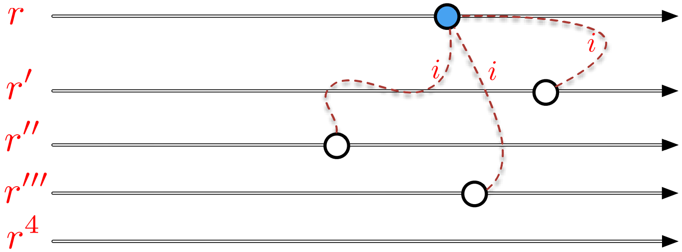
Basic notion: Indistinguishability



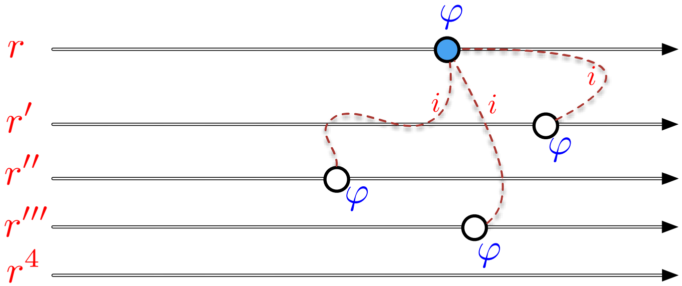
Defining Knowledge in Pictures



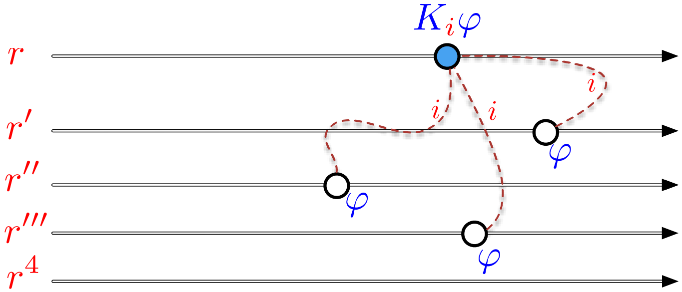
Defining Knowledge in Pictures



Defining Knowledge in Pictures



Defining Knowledge in Pictures



Defining Knowledge more formally [Fagin *et al.* 1995]

- A run is a sequence $r : \mathbb{N} \to \mathcal{G}$ of global states.
- A system is a set R of runs.
- Typically, $R = \{\text{runs of a protocol } P \text{ in a model } M\}$.

Assumption

Each global state $r(t)$ determines a *local state* $r_i(t)$ for every agent i.

A point (r, t) refers to time t in run r.

Defining Knowledge more formally [Fagin *et al.* 1995]

- A run is a sequence $r : \mathbb{N} \to \mathcal{G}$ of global states.
- A system is a set R of runs.
- Typically, $R = \{\text{runs of a protocol } P \text{ in a model } M\}$.

Assumption

Each global state $r(t)$ determines a *local state* $r_{\mathtt{i}}(t)$ for every agent $\mathtt{i}$.

A point (r, t) refers to time t in run r.

Defining Knowledge more formally [Fagin *et al.* 1995]

- A run is a sequence $r : \mathbb{N} \to \mathcal{G}$ of global states.
- A system is a set R of runs.
- Typically, $R = \{$runs of a protocol P in a model $M\}$.

Assumption

Each global state $r(t)$ determines a *local state* $r_{\mathtt{i}}(t)$ for every agent $\mathtt{i}$.

A point (r, t) refers to time t in run r.

A Propositional Logic of Knowledge

Facts are considered "true" or "false" at a point.

$\boldsymbol{(R, r, t) \models \varphi}$ denotes that φ is true at (r, t) wrt R.

A Propositional Logic of Knowledge

Starting from a set $\mathbf{\Phi}$ of primitive propositions, define $\mathcal{L}_n^K = \mathcal{L}_n^K(\mathbf{\Phi})$ by

$$\varphi \quad := \quad p \in \Phi \;\mid\; \neg\varphi \;\mid\; \varphi \wedge \varphi \;\mid\; K_1\varphi \;\mid\; \cdots \;\mid\; K_n\varphi$$

Given an interpretation $\pi : \Phi \times \mathrm{Pts}(R) \to \{\mathrm{True}, \mathrm{False}\}$

$(R, r, t) \models p$, for $p \in \Phi$, iff $\pi(p, r, t) = \mathrm{True}$.

$(R, r, t) \models \neg\varphi$ iff $(R, r, t) \not\models \varphi$

$(R, r, t) \models \varphi \wedge \psi$ iff both $(R, r, t) \models \varphi$ and $(R, r, t) \models \psi$.

Knowledge = Truth in All Possible Worlds

$(\boldsymbol{R}, \boldsymbol{r}, \boldsymbol{t}) \models \boldsymbol{K}_i\varphi$ iff for all points (r', t') of R such that $r_i(t) = r'_i(t')$ we have $(\boldsymbol{R}, \boldsymbol{r'}, \boldsymbol{t'}) \models \varphi$.

Comments:

The definition ignores the complexity of computing knowledge

Local information = current local state

$K_i\varphi$ holds if φ is guaranteed to hold in R given i's local state

The definition is model independent

Knowledge = Truth in All Possible Worlds

$$(\boldsymbol{R}, \boldsymbol{r}, \boldsymbol{t}) \models \boldsymbol{K}_{\mathtt{i}}\varphi \quad \text{iff} \quad \text{for all points } (r', t') \text{ of } R \text{ such that } r_{\mathtt{i}}(t) = r'_{\mathtt{i}}(t') \text{ we have } (\boldsymbol{R}, \boldsymbol{r'}, \boldsymbol{t'}) \models \varphi.$$

Comments:

The definition ignores the complexity of computing knowledge

Local information **=** current local state

$K_{\mathtt{i}}\varphi$ holds if φ is guaranteed to hold in R given i's local state

The definition is model independent

Knowledge = Truth in All Possible Worlds

$$(\boldsymbol{R}, \boldsymbol{r}, \boldsymbol{t}) \models \boldsymbol{K}_{\mathtt{i}}\varphi \quad \text{iff} \quad \text{for all points } (r', t') \text{ of } R \text{ such that } r_{\mathtt{i}}(t) = r'_{\mathtt{i}}(t') \text{ we have } (\boldsymbol{R}, \boldsymbol{r'}, \boldsymbol{t'}) \models \varphi.$$

Comments:

The definition ignores the complexity of computing knowledge

Local information = current local state

$K_{\mathtt{i}}\varphi$ holds if φ is guaranteed to hold in R given $\mathtt{i}$'s local state

The definition is model independent

Specifications and Knowledge

Problems in Distributed Computing are presented via specifications

- A bank's system of ATMs
- Autonomous cars
- A distributed database
- A Google data center

Specifications impose epistemic constraints on actions!

Specifications and Knowledge

Problems in Distributed Computing are presented via specifications

- A bank's system of ATMs
- Autonomous cars
- A distributed database
- A Google data center

Specifications impose epistemic constraints on actions!

Knowledge of Preconditions

$(R, r, t) \models \mathtt{does}_{\mathtt{i}}(\alpha)$ iff $\mathtt{i}$ performs α at time t in r.

Theorem (KoP)

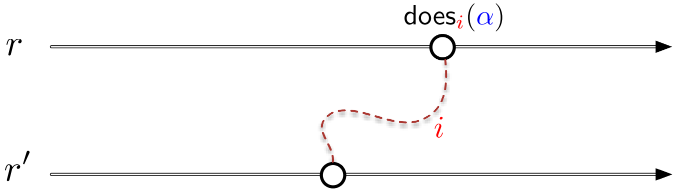
Under minor assumptions on α and φ in R:

If φ *is a necessary condition for* $\mathtt{does}_{\mathtt{i}}(\alpha)$ *in* R,

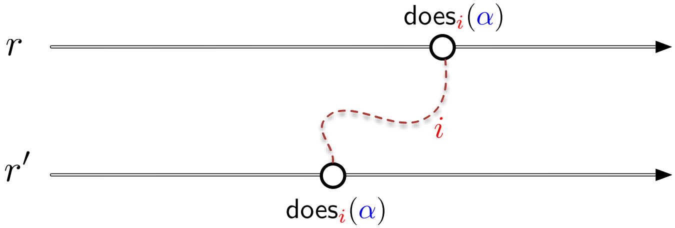
then $K_{\mathtt{i}}\varphi$ *is a necessary condition for* $\mathtt{does}_{i}(\alpha)$ *in* R.

Deterministic Actions

Deterministic Actions



Deterministic Actions



Deterministic Actions

Definition

Action α is deterministic for $\mathtt{i}$ in R if whenever $r_{\mathtt{i}}(t) = r'_{\mathtt{i}}(t')$:

$$(R, r, t) \models \mathtt{does}_{\mathtt{i}}(\alpha) \quad \text{iff} \quad (R, r', t') \models \mathtt{does}_{\mathtt{i}}(\alpha).$$

$\mathtt{i}$'s local state determines whether it performs α at points of R.

Deterministic Actions

Definition

Action α is deterministic for $\mathtt{i}$ in R if whenever $r_{\mathtt{i}}(t) = r'_{\mathtt{i}}(t')$:

$$(R, r, t) \models \mathtt{does}_{\mathtt{i}}(\alpha) \quad \text{iff} \quad (R, r', t') \models \mathtt{does}_{\mathtt{i}}(\alpha).$$

$\mathtt{i}$'s local state determines whether it performs α at points of R.

The KoP Theorem for Deterministic Actions

Theorem (**KoP**, [M. 2015])

Let α be a deterministic action for $\mathtt{i}$ in R.

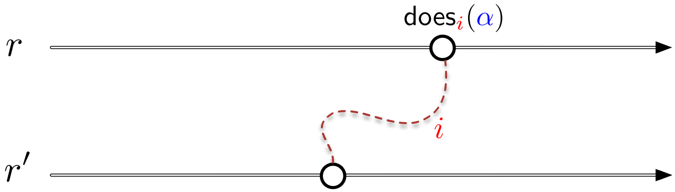
If φ is a necessary condition for $\mathtt{does}_{\mathtt{i}}(\alpha)$ in R,

then $K_{\mathtt{i}}\varphi$ is a necessary condition for $\mathtt{does}_{\mathtt{i}}(\alpha)$ in R.

Proof of **K*o*P**

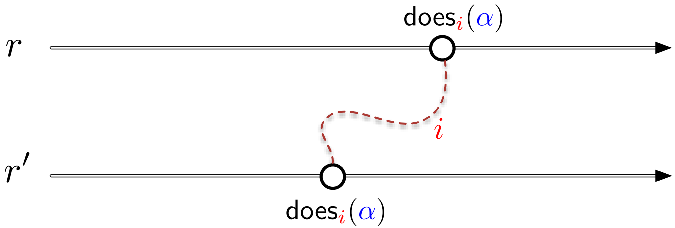
$$(R, r, t) \models \texttt{does}_i(\alpha)$$

Proof of **K*o*P**



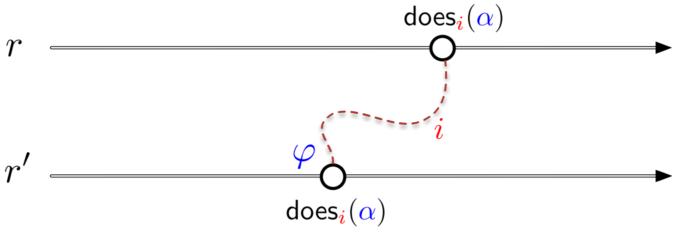
$$(r, t) \approx_i (r', t')$$

Proof of **K*o*P**



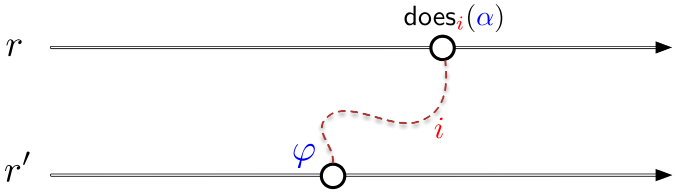
α is deterministic

Proof of **KoP**



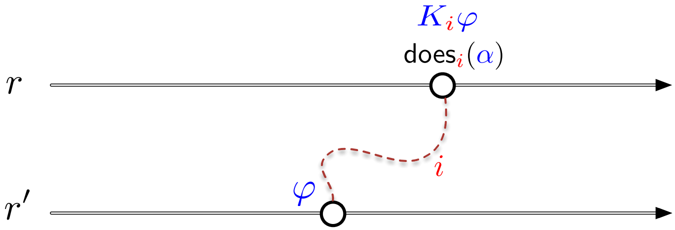
φ is a necessary condition

Proof of **K*o*P**



φ holds at all indistinguishable points

Proof of **K*o*P**



so $K_i\varphi$ holds

Proof of **KoP**

$\texttt{does}_i(\alpha) \;\Rightarrow\; K_i\varphi$ QED

K*o*P Applies Very Broadly

The **K*o*P** is a universal theorem for distributed systems

K*o*P applies to ATMs, autonomous cars, and even more generally:

- Legal systems:

  Judge Punishes $X \Rightarrow X$ committed the crime

  Judge Punishes $X \Rightarrow K_J(X$ committed the crime$)$

- Nature:

  Jellyfish stings $X \Rightarrow X \neq$ a rock

  Jellyfish stings $X \Rightarrow K_J(X \neq$ a rock$)$

- Betting:

  Don bets on `Phar Lap` $\Rightarrow$ `PL` will win

  Don bets on `Phar Lap` $\Rightarrow K_D($`PL` will win$)$

KoP Applies Very Broadly

The **KoP** is a universal theorem for distributed systems

KoP applies to ATMs, autonomous cars, and even more generally:

- Legal systems:

  Judge Punishes $X \Rightarrow X$ committed the crime

  Judge Punishes $X \Rightarrow K_J(X$ committed the crime$)$

- Nature:

  Jellyfish stings $X \Rightarrow X \neq$ a rock

  Jellyfish stings $X \Rightarrow K_J(X \neq$ a rock$)$

- Betting:

  Don bets on `Phar Lap` $\Rightarrow$ `PL` will win

  Don bets on `Phar Lap` $\Rightarrow K_D($`PL` will win$)$

KoP Applies Very Broadly

The **KoP** is a universal theorem for distributed systems

KoP applies to ATMs, autonomous cars, and even more generally:

- Legal systems:

  Judge Punishes $X \Rrightarrow X$ committed the crime

  Judge Punishes $X \Rrightarrow K_J(X$ committed the crime$)$

- Nature:

  Jellyfish stings $X \Rrightarrow X \neq$ a rock

  Jellyfish stings $X \Rrightarrow K_J(X \neq$ a rock$)$

- Betting:

  Don bets on `Phar Lap` $\Rrightarrow$ `PL` will win

  Don bets on `Phar Lap` $\Rrightarrow K_D($`PL` will win$)$

KoP Applies Very Broadly

The **KoP** is a universal theorem for distributed systems

KoP applies to ATMs, autonomous cars, and even more generally:

- Legal systems:

  Judge Punishes $X \quad \Rrightarrow \quad X$ committed the crime

  Judge Punishes $X \quad \Rrightarrow \quad K_J(X$ committed the crime$)$

- Nature:

  Jellyfish stings $X \quad \Rrightarrow \quad X \neq$ a rock

  Jellyfish stings $X \quad \Rrightarrow \quad K_J(X \neq$ a rock$)$

- Betting:

  Don bets on `Phar Lap` $\quad \Rrightarrow \quad$ `PL` will win

  Don bets on `Phar Lap` $\quad \Rrightarrow \quad K_D($`PL` will win$)$

KoP Applies Very Broadly

The **KoP** is a universal theorem for distributed systems

KoP applies to ATMs, autonomous cars, and even more generally:

- Legal systems:

  Judge Punishes $X \quad \Rrightarrow \quad X$ committed the crime

  Judge Punishes $X \quad \Rrightarrow \quad K_J(X$ committed the crime$)$

- Nature:

  Jellyfish stings $X \quad \Rrightarrow \quad X \neq$ a rock

  Jellyfish stings $X \quad \Rrightarrow \quad K_J(X \neq$ a rock$)$

- Betting:

  Don bets on `Phar Lap` $\quad \Rrightarrow \quad$ `PL` will win

  Don bets on `Phar Lap` $\quad \Rrightarrow \quad K_D($`PL` will win$)$

An Application: Binary Consensus

Model:

- Each process $\mathtt{i} = 1, \ldots, n$ starts with a value $v_i \in \{0, 1\}$.
- Communication network is a complete graph
- Synchronous message passing
- At most $t < n$ crash failures
- We assume a full-information protocol

An Application: Binary Consensus

Specification: A consensus protocol must guarantee

- Decision: Every correct process decides on a value in $\{0, 1\}$.
- Agreement: All correct processes decide on the same value.
- Validity: A decision value must be an initial value.

Validity means $\mathtt{decide_i}(v) \Rightarrow \exists v$

and so

$$\mathtt{decide_i}(0) \Rightarrow K_i \exists 0$$

An Application: Binary Consensus

Specification: A consensus protocol must guarantee

- Decision: Every correct process decides on a value in $\{0, 1\}$.
- Agreement: All correct processes decide on the same value.
- Validity: A decision value must be an initial value.

Validity means $\texttt{decide}_{\texttt{i}}(v) \Rightarrow \exists v$

and so

$$\texttt{decide}_{\texttt{i}}(0) \Rightarrow K_i \exists 0$$

An Application: Binary Consensus

Specification: A consensus protocol must guarantee

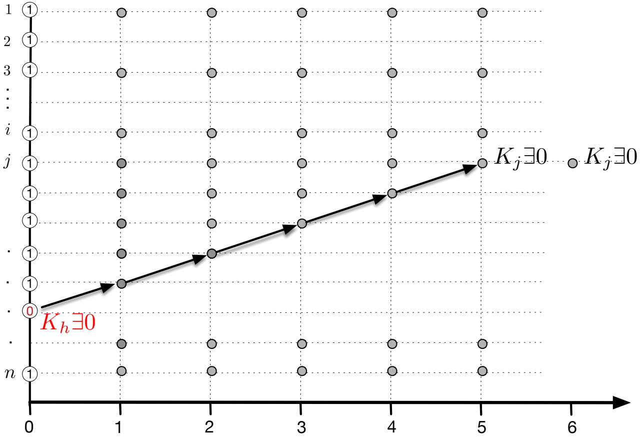
- Decision: Every correct process decides on a value in $\{0, 1\}$.
- Agreement: All correct processes decide on the same value.
- Validity: A decision value must be an initial value.

Validity means $\texttt{decide}_{\texttt{i}}(v) \Rightarrow \exists v$

and so

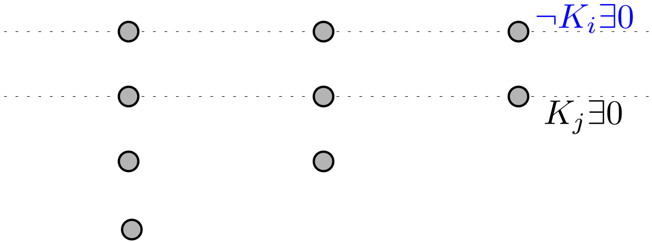
$$\texttt{decide}_{\texttt{i}}(0) \Rightarrow K_i \exists 0$$

Knowing $\exists 0$



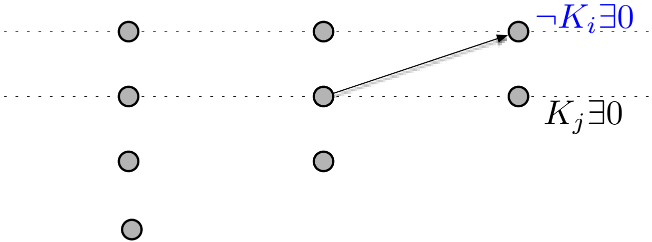
$K_j\exists 0$ holds iff there is a message chain from an initial value of 0 to j.

Knowing $\exists 0$



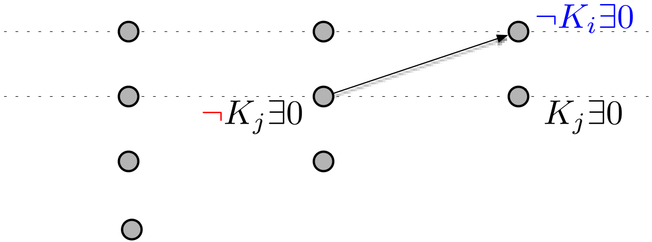
How can one proc know $\exists 0$ when another does not?

Knowing $\exists 0$



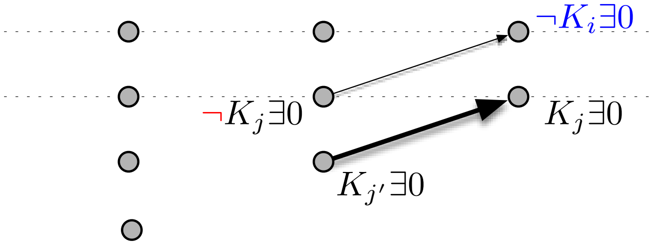
How can one proc know $\exists 0$ when another does not?

Knowing $\exists 0$



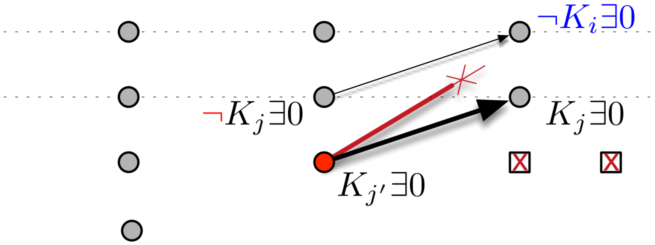
How can one proc know $\exists 0$ when another does not?

Knowing $\exists 0$



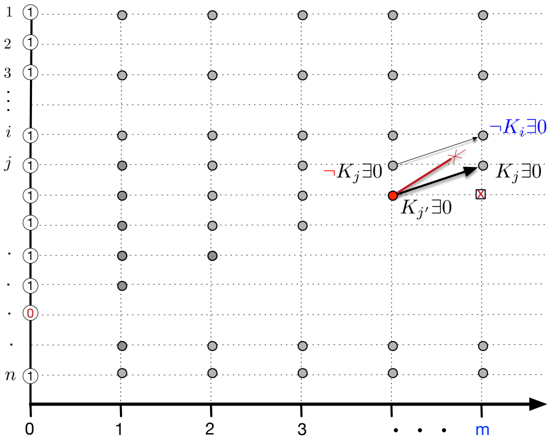
How can one proc know $\exists 0$ when another does not?

Knowing $\exists 0$

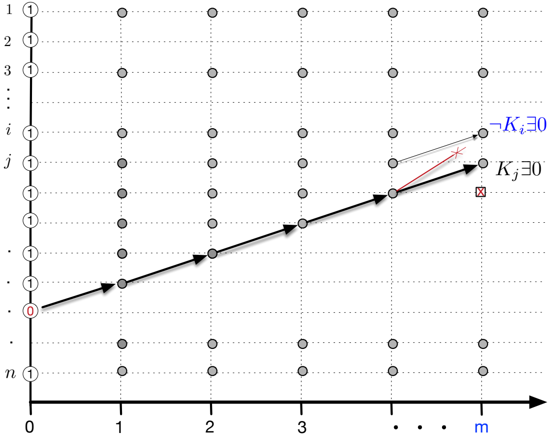


How can one proc know $\exists 0$ when another does not?

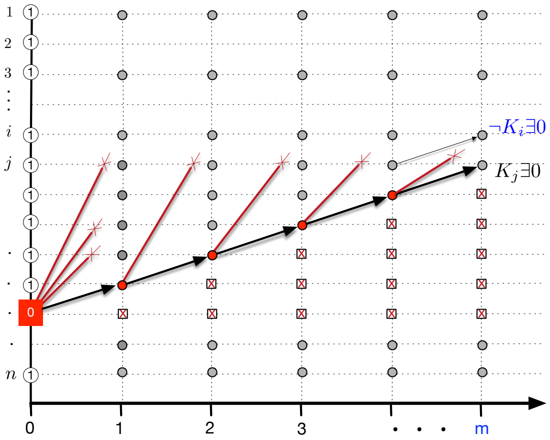
Knowing $\exists 0$



Knowing ∃0

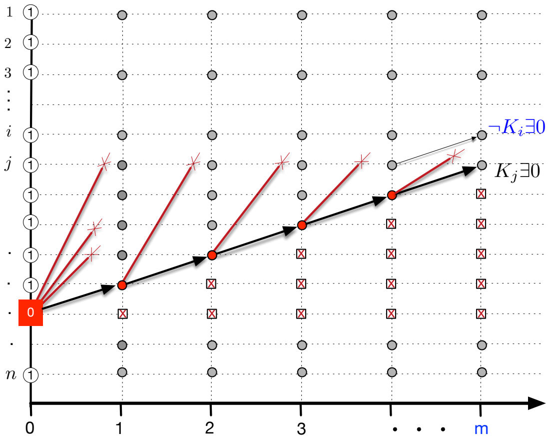


Knowing $\exists 0$



Claim: If $K_j \exists 0$ & $\neg K_i \exists 0$ at time m, then $\geq m$ crashes have occurred

Knowing $\exists 0$



Corollary: At time $t+1$, either everyone knows $\exists 0$ or nobody does

A Simple Consensus Protocol

Protocol P_0 (for undecided process i):

if $\texttt{time} = t+1$ & $K_i \exists 0$ **then** $\texttt{decide}_\texttt{i}(0)$

elseif $\texttt{time} = t+1$ & $\neg K_i \exists 0$ **then** $\texttt{decide}_\texttt{i}(1)$

Communication is according to the fip.

All decisions at time $t+1$

A Simple Consensus Protocol

Protocol P_0 (for undecided process i):

if $\texttt{time} = t+1 \ \& \ K_i\exists 0$ **then** $\texttt{decide}_{\texttt{i}}(0)$

elseif $\texttt{time} = t+1 \ \& \ \neg K_i\exists 0$ **then** $\texttt{decide}_{\texttt{i}}(1)$

Communication is according to the fip.

All decisions at time $t+1$

A Better Protocol

Protocol Q_0 (for undecided process i):

if $K_i \exists 0$ **then** $\texttt{decide}_\texttt{i}(0)$

elseif $\texttt{time} = t + 1$ & $\neg K_i \exists 0$ **then** $\texttt{decide}_\texttt{i}(1)$

All decisions by time $t + 1$

A Better Protocol

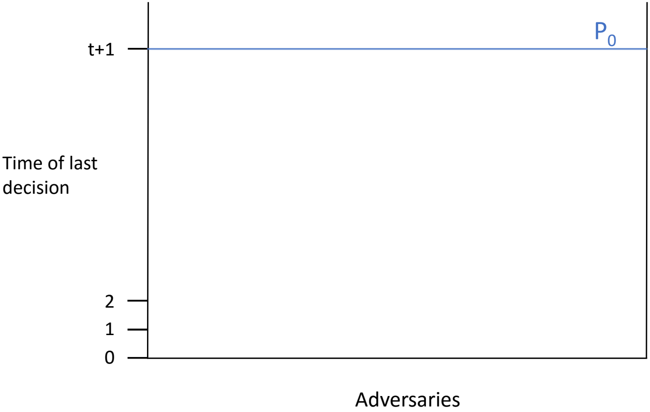
Protocol Q_0 (for undecided process i):

if $K_i \exists 0$ **then** $\texttt{decide}_{\texttt{i}}(0)$

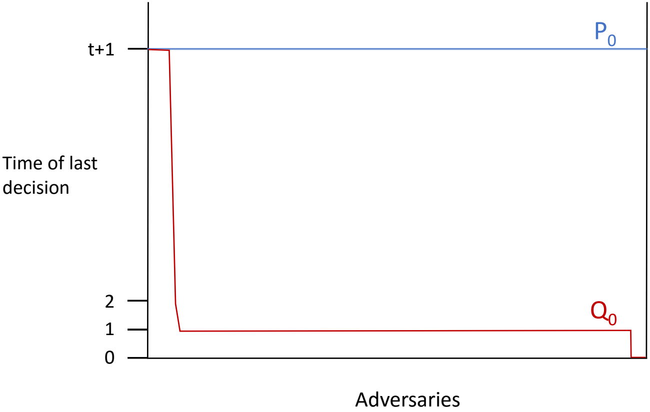
elseif $\texttt{time} = t + 1$ & $\neg K_i \exists 0$ **then** $\texttt{decide}_{\texttt{i}}(1)$

All decisions by time $t + 1$

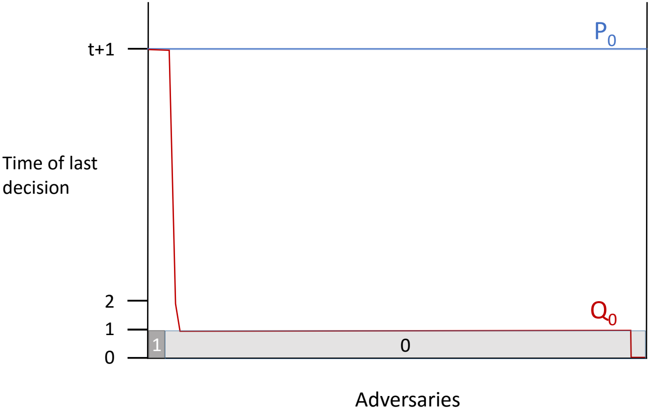
Performance of P_0 and Q_0



Performance of P_0 and Q_0



Performance of P_0 and Q_0



Deciding Efficiently on 1

Design Decision: $K_j \exists 0 \Leftrightarrow \texttt{decide}_\texttt{j}(0)$.

When can $\texttt{decide}_\texttt{i}(1)$ be performed?

Recall:

Agreement: $\texttt{decide}_\texttt{i}(1) \Rightarrow$ Nobody decides 0

$\texttt{decide}_\texttt{i}(1) \Rightarrow$ "no currently active process knows $\exists 0$"

By **K*o*P**, $\texttt{decide}_\texttt{i}(1) \Rightarrow K_i(\texttt{nobody\_knows}\exists 0)$

Deciding Efficiently on 1

Design Decision: $K_j \exists 0 \Leftrightarrow \mathtt{decide_j(0)}$.

When can $\mathtt{decide_i(1)}$ be performed?

Recall:

Agreement: $\mathtt{decide_i(1)} \Rrightarrow$ Nobody decides 0

$\mathtt{decide_i(1)} \Rrightarrow$ "no currently active process knows $\exists 0$"

By **K*o*P**, $\mathtt{decide_i(1)} \Rrightarrow K_i(\mathtt{nobody\_knows}\exists 0)$

Deciding Efficiently on 1

Design Decision: $K_j \exists 0 \Leftrightarrow \texttt{decide}_{\texttt{j}}(0)$.

When can $\texttt{decide}_{\texttt{i}}(1)$ be performed?

Recall:

Agreement: $\texttt{decide}_{\texttt{i}}(1) \Rrightarrow$ Nobody decides 0

$\texttt{decide}_{\texttt{i}}(1) \Rrightarrow$ "no currently active process knows $\exists 0$"

By **K*o*P**, $\texttt{decide}_{\texttt{i}}(1) \Rrightarrow K_i(\texttt{nobody\_knows}\exists 0)$

Deciding Efficiently on 1

Design Decision: $K_j \exists 0 \Leftrightarrow \texttt{decide}_\texttt{j}(0)$.

When can $\texttt{decide}_\texttt{i}(1)$ be performed?

Recall:

Agreement: $\texttt{decide}_\texttt{i}(1) \;\Rrightarrow\;$ Nobody decides 0

$\texttt{decide}_\texttt{i}(1) \;\Rrightarrow\;$ "no currently active process knows $\exists 0$"

By **K*o*P**, $\texttt{decide}_\texttt{i}(1) \;\Rrightarrow\; K_i(\texttt{nobody\_knows}\exists 0)$

Unbeatable Consensus

[Castañeda, Gonczarowski & M. '14]

Protocol OPT_0 (for undecided process i):

if $K_{\mathtt{i}}\exists 0$ **then** $\mathtt{decide_i}(0)$

elseif $K_{\mathtt{i}}(\mathtt{nobody\_knows}\exists 0)$ **then** $\mathtt{decide_i}(1)$

My name is Sherlock Holmes.
It is my business to know
what other people don't know.

The Adventure of the Blue Carbuncle, 1892

Unbeatable Consensus

[Castañeda, Gonczarowski & M. '14]

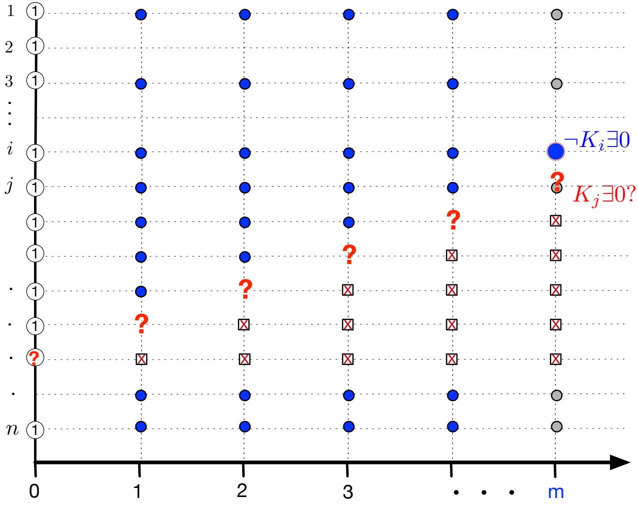
Protocol OPT_0 (for undecided process i):

if $K_{\mathtt{i}}\exists 0$ **then** $\mathtt{decide_i}(0)$

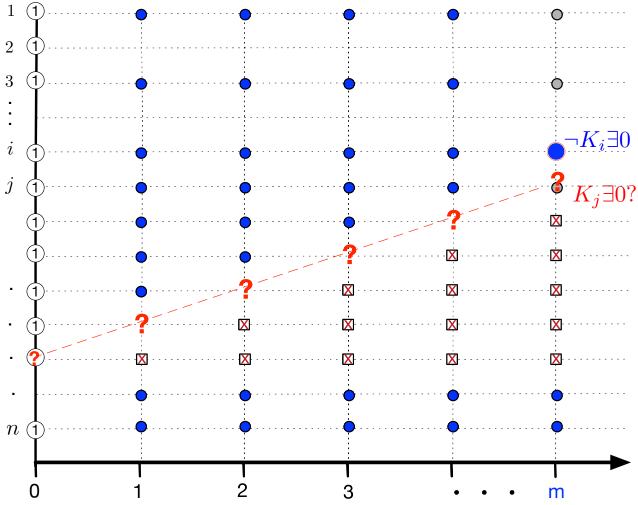
elseif $K_{\mathtt{i}}(\mathtt{nobody\_knows}\exists 0)$ **then** $\mathtt{decide_i}(1)$

```
My name is Sherlock Holmes.
It is my business to know
what other people don't know.
```

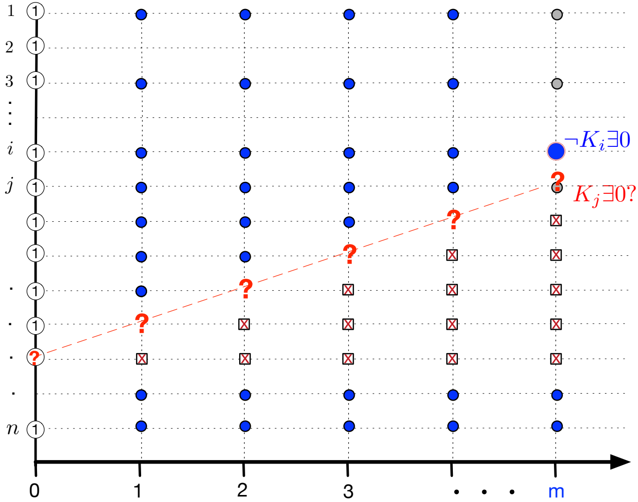
The Adventure of the Blue Carbuncle, 1892



W.r.t. (i, m), nodes are seen, crashed, or hidden



A hidden path wrt (i, m)



Theorem: $\exists$ a hidden path iff $\neg K_{\mathtt{i}}(\mathtt{nobody\_knows}\exists 0)$

Implementing OPT_0 [Castañeda, Gonczarowski & M. '14]

Standard OPT_0 (for undecided process i):

if seen 0 **then** $\texttt{decide}_{\texttt{i}}(0)$

elseif no hidden path **then** $\texttt{decide}_{\texttt{i}}(1)$

Theorem (CGM)

- OPT_0 *strictly dominates* Q_0
- OPT_0 *is unbeatable: No consensus protocol dominates it.*
- OPT_0 *is implementable using* $O(t \log n)$ *bits of communication per process*

Implementing OPT_0 [Castañeda, Gonczarowski & M. '14]

Standard OPT_0 (for undecided process i):

if seen 0 **then** $\texttt{decide}_{\texttt{i}}(0)$

elseif no hidden path **then** $\texttt{decide}_{\texttt{i}}(1)$

Theorem (CGM)

- *OPT_0 strictly dominates Q_0*
- *OPT_0 is unbeatable: No consensus protocol dominates it.*
- *OPT_0 is implementable using $O(t \log n)$ bits of communication per process*

Ordering Actions

Definition (Ordered Actions)

Actions $\langle \boldsymbol{\alpha_1}, \ldots, \boldsymbol{\alpha_k} \rangle$ (for agents $1, \ldots, k$) are ordered in R if

$$\mathtt{does}_j(\boldsymbol{\alpha_j}) \quad \Rightarrow \quad \mathtt{Did}_{j-1}(\boldsymbol{\alpha_{j-1}}) \qquad \text{in } R$$

I.e., $\boldsymbol{t_{j-1} \leq t_j}$ if t_i denotes when α_i occurs.

Nested Knowledge and Ordered Actions

Theorem (Nested Knowledge of Preconditions)

Let $\langle \boldsymbol{\alpha_1}, \ldots, \boldsymbol{\alpha_k} \rangle$ be ordered in R.

$$
\begin{array}{llcll}
\text{If} & \mathtt{does}_1(\boldsymbol{\alpha_1}) & \Rightarrow & \mathrm{occ'd}(e) & \text{in } R \\
\text{then} & \mathtt{does}_j(\boldsymbol{\alpha_j}) & \Rightarrow & K_j K_{j-1} \cdots K_1\, \mathrm{occ'd}(e) & \text{in } R
\end{array}
$$

Nested Knowledge and Ordered Actions

Theorem (Nested Knowledge of Preconditions)

Let $\langle \boldsymbol{\alpha_1}, \ldots, \boldsymbol{\alpha_k} \rangle$ be ordered in R.

If	$\mathtt{does}_1(\boldsymbol{\alpha_1})$	$\Rrightarrow$	$\mathrm{occ'd}(e)$	in R
then	$\mathtt{does}_j(\boldsymbol{\alpha_j})$	$\Rrightarrow$	$K_j K_{j-1} \cdots K_1\, \mathrm{occ'd}(e)$	in R

Relating Knowledge and Communication

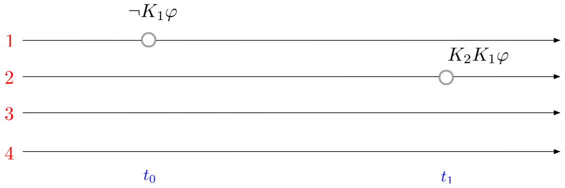
In "How Processes Learn" in 1985 Chandy and Misra showed

Theorem (Knowledge Gain)

Let R be asynchronous and $t_1 > t_0$.

If $(R, r, t_0) \models \neg K_1\varphi$ *and*

$(R, r, t_1) \models K_2 K_1 \varphi$



Relating Knowledge and Communication

In "`How Processes Learn`" in 1985 Chandy and Misra showed

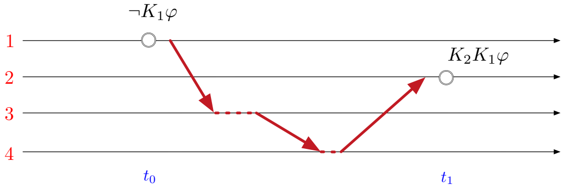
Theorem (Knowledge Gain)

Let R be asynchronous and $t_1 > t_0$.

If $(R, r, t_0) \models \neg K_1\varphi$ *and*

$(R, r, t_1) \models K_2 K_1 \varphi$

then there must be a (Lamport) message chain in r from process 1 to process 2 between times t_0 and t_1.



Relating Knowledge and Communication

In "How Processes Learn" in 1985 Chandy and Misra showed

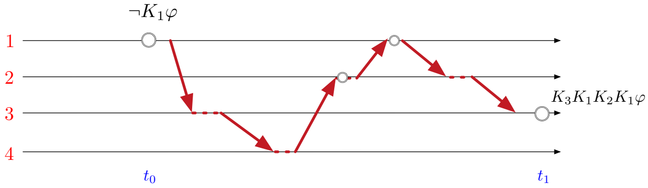
Theorem (Knowledge Gain)

Let R be asynchronous and $t_1 > t_0$.

If $(R, r, t_0) \models \neg K_1\varphi$ *and*

$(R, r, t_1) \models K_m K_{m-1} \cdots K_1\varphi$

then there must be a (Lamport) message chain in r from process 1 through process 2, 3, ..., m between times t_0 and t_1.



Relating Knowledge and Communication

In "How Processes Learn" in 1985 Chandy and Misra showed

Theorem (Knowledge Gain)

Let R be asynchronous and $t_1 > t_0$.

If $(R, r, t_0) \models \neg K_1\varphi$ *and*

$(R, r, t_1) \models K_m K_{m-1} \cdots K_1\varphi$

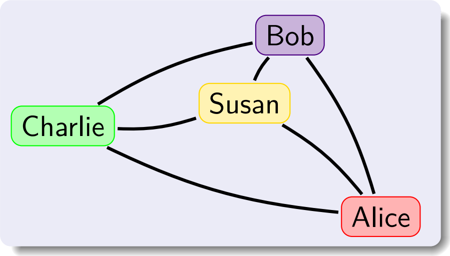
then there must be a (Lamport) message chain in r from process 1 through process 2, 3, ..., m between times t_0 and t_1.

Corollary

Message chains are necessary for ordering actions under asynchrony

Temporal Ordering Example: The Frozen Account

Alice, Bob, Charlie and Susan are nodes in a network.



- Alice needs to cash Charlie's cheque
- Charlie's account is frozen

$\Rightarrow$ they must coordinate. . .

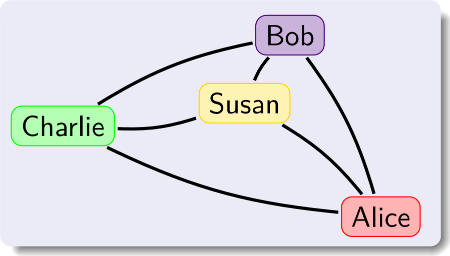
Charlie deposits a sufficient sum

Bob reactivates Charlie's account

Alice cashes the cheque

Temporal Ordering Example: The Frozen Account

Alice, Bob, Charlie and Susan are nodes in a network.



- Alice needs to cash Charlie's cheque
- Charlie's account is frozen

$\Rightarrow$ they must coordinate...

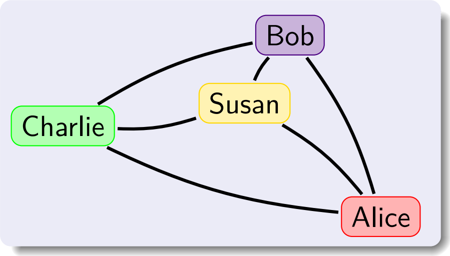
Charlie deposits a sufficient sum

Bob reactivates Charlie's account

Alice cashes the cheque

Temporal Ordering Example: The Frozen Account

Alice, Bob, Charlie and Susan are nodes in a network.



- Alice needs to cash Charlie's cheque
- Charlie's account is frozen

⇒ they must coordinate. . .

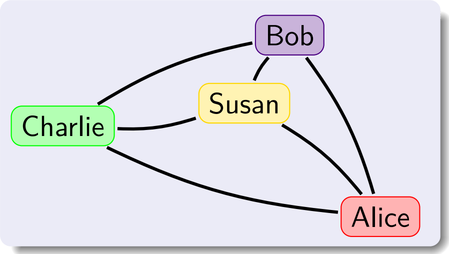
Charlie deposits a sufficient sum

Bob reactivates Charlie's account

Alice cashes the cheque

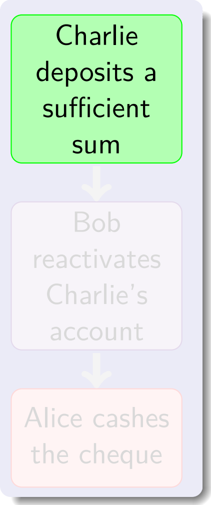
Temporal Ordering Example: The Frozen Account

Alice, Bob, Charlie and Susan are nodes in a network.



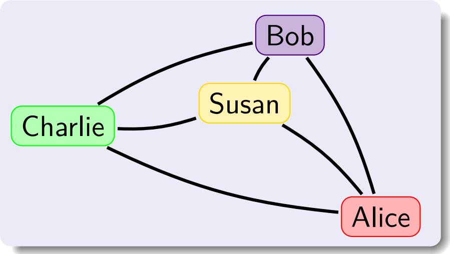
- Alice needs to cash Charlie's cheque
- Charlie's account is frozen

⇒ they must coordinate. . .



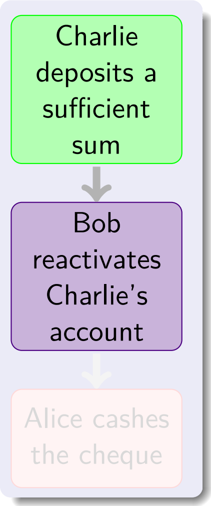
Temporal Ordering Example: The Frozen Account

Alice, Bob, Charlie and Susan are nodes in a network.



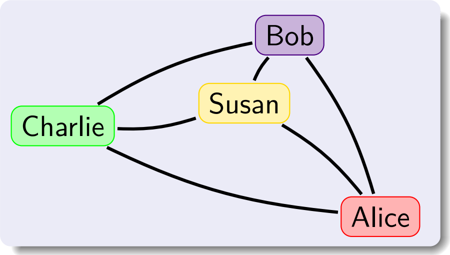
- Alice needs to cash Charlie's cheque
- Charlie's account is frozen

⇒ they must coordinate...



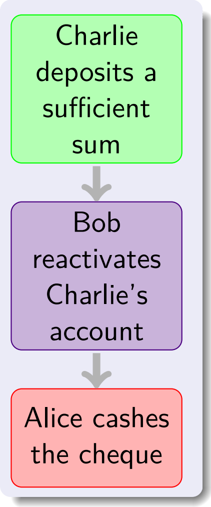
Temporal Ordering Example: The Frozen Account

Alice, Bob, Charlie and Susan are nodes in a network.



- Alice needs to cash Charlie's cheque
- Charlie's account is frozen

⇒ they must coordinate...



The Clocks and Bounds Model

We assume a directed network graph and

- Global clocks
- An upper $bound_{ij}$ on transmission times per channel $i \mapsto j$
  - $1 \leq bound_{ij} < \infty$
  - Delivery within the bound is guaranteed
- Lower bounds of 1 on message transmission.

The Clocks and Bounds Model

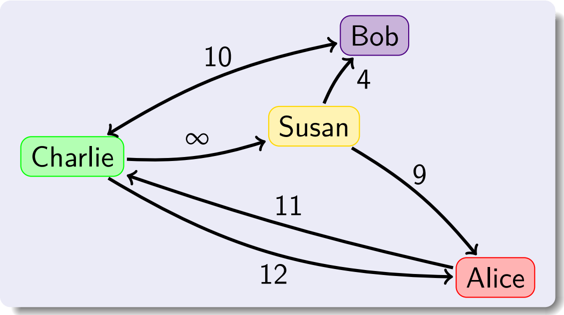
We assume a directed network graph and

- Global clocks
- An upper $bound_{ij}$ on transmission times per channel $i \mapsto j$
  - $1 \leq bound_{ij} < \infty$
  - Delivery within the bound is guaranteed
- Lower bounds of 1 on message transmission.

The Clocks and Bounds Model

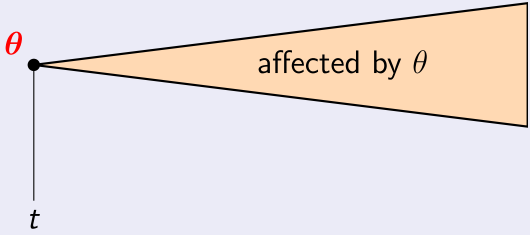
We assume a directed network graph and

- Global clocks
- An upper $bound_{ij}$ on transmission times per channel $i \mapsto j$
  - $1 \leq bound_{ij} < \infty$
  - Delivery within the bound is guaranteed
- Lower bounds of 1 on message transmission.



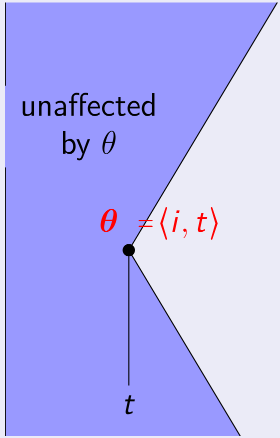
Causal Cones — Digital Time/Space

Upper bounds determine a cone of necessarily affected nodes.



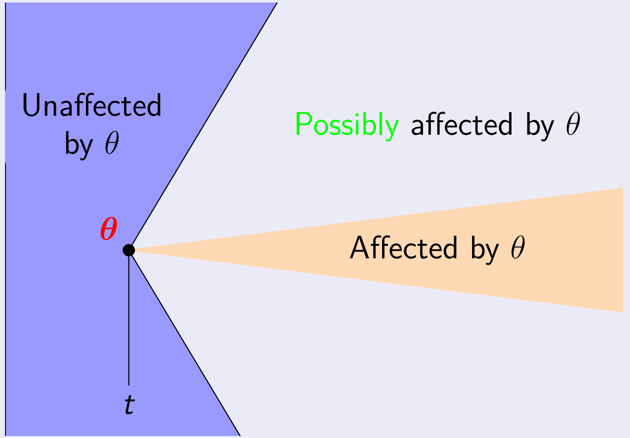
Causal Cones — Digital Time/Space

Lower bounds determine a co-cone of necessarily unaffected nodes.



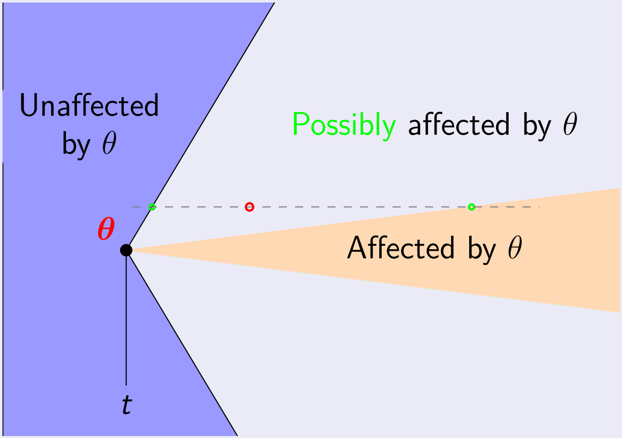
Causal Cones — Digital Time/Space

Bounds create 3 regions:



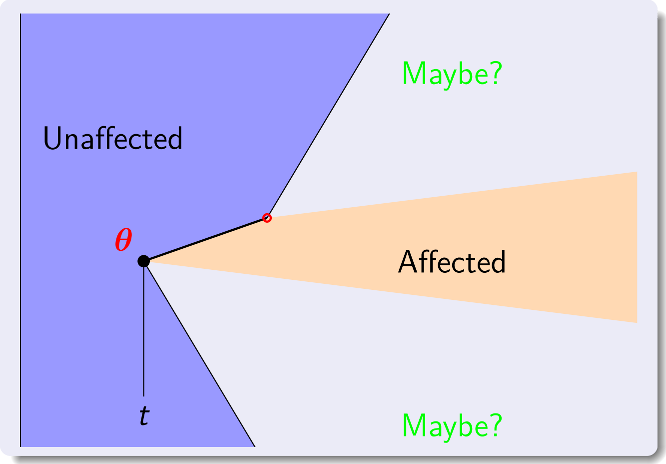
Causal Cones — Digital Time/Space

Impact of a message delivery



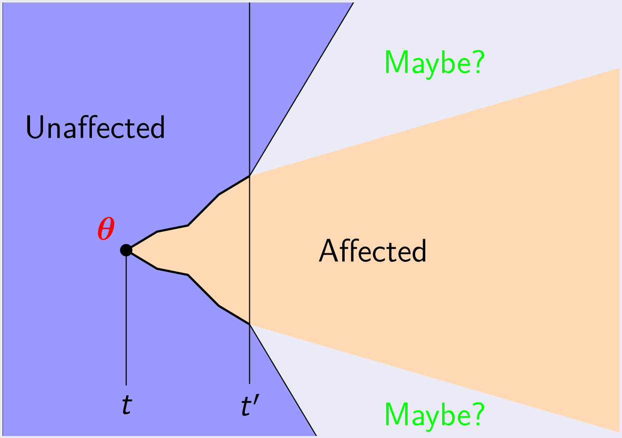
Causal Cones — Digital Time/Space

A delivery extends inner and outer regions



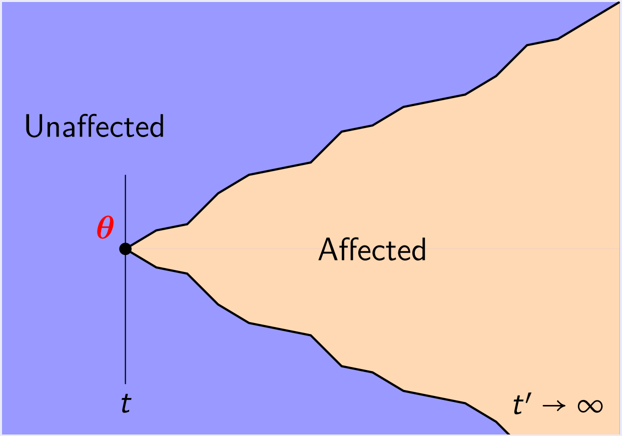
Causal Cones — Digital Time/Space

All past uncertainty at $t' > t$ is resolved



Causal Cones — Digital Time/Space

Ex-post, all uncertainty is resolved



Event Ordering for Alice, Bob & Charlie

- With clocks, ordering seems simple. . .
- But Charlie's deposit is *spontaneous*
- Information flow is then required for
  - notifying Alice and Bob of the deposit and
  - managing coordination
- Lamport message chains can be used

Do clocks help?

Charlie deposits on **Sunday**

Bob reactivates on **Monday**

Alice cashes in on **Tuesday**

Event Ordering for Alice, Bob & Charlie

- With clocks, ordering seems simple...
- But Charlie's deposit is *spontaneous*
- Information flow is then required for
  - notifying Alice and Bob of the deposit and
  - managing coordination
- Lamport message chains can be used

Do clocks help?

Charlie deposits on **Sunday**

↓

Bob reactivates on **Monday**

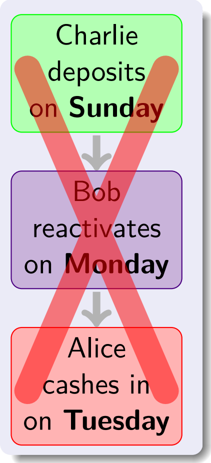
↓

Alice cashes in on **Tuesday**

Event Ordering for Alice, Bob & Charlie

- With clocks, ordering seems simple. . .
- But Charlie's deposit is *spontaneous*
- Information flow is then required for
  - notifying Alice and Bob of the deposit and
  - managing coordination
- Lamport message chains can be used

Do clocks help?

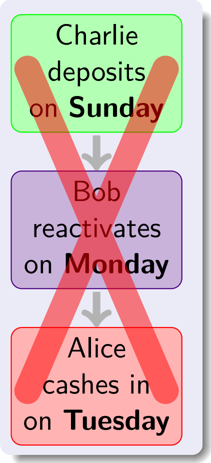


Event Ordering for Alice, Bob & Charlie

- With clocks, ordering seems simple. . .
- But Charlie's deposit is *spontaneous*
- Information flow is then required for
  - notifying Alice and Bob of the deposit and
  - managing coordination

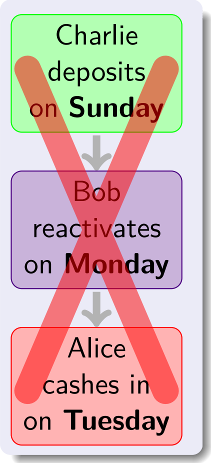
- Lamport message chains can be used

Do clocks help?



Event Ordering for Alice, Bob & Charlie

- With clocks, ordering seems simple...
- But Charlie's deposit is *spontaneous*
- Information flow is then required for
  - notifying Alice and Bob of the deposit and
  - managing coordination

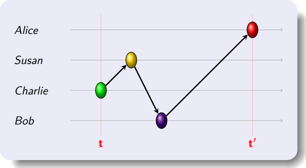


- Lamport message chains can be used

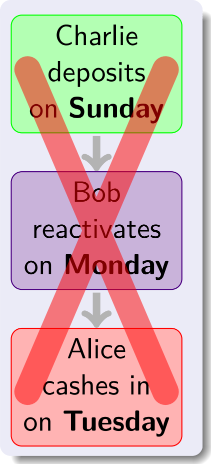
Do clocks help?

Event Ordering for Alice, Bob & Charlie

- With clocks, ordering seems simple. . .
- But Charlie's deposit is *spontaneous*
- Information flow is then required for
  - notifying Alice and Bob of the deposit and
  - managing coordination
- Lamport message chains can be used

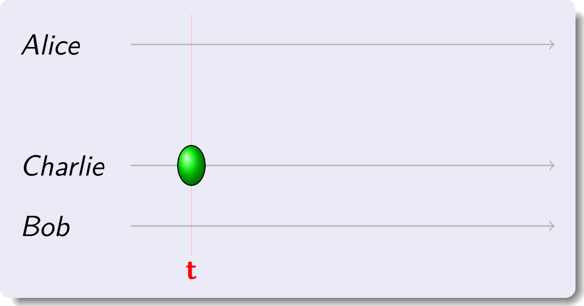


Do clocks help?



Ordering Based on Time Bounds

Time bounds:

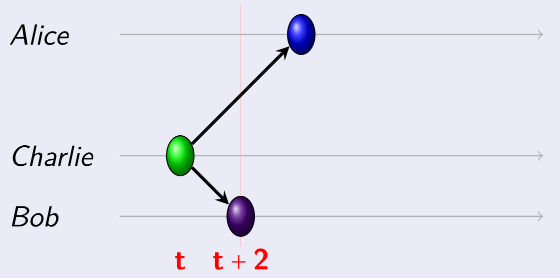


Charlie deposits a sufficient sum

Ordering Based on Time Bounds

Time bounds:

$bound_{Charlie,Bob} = 10$



Charlie deposits a sufficient sum

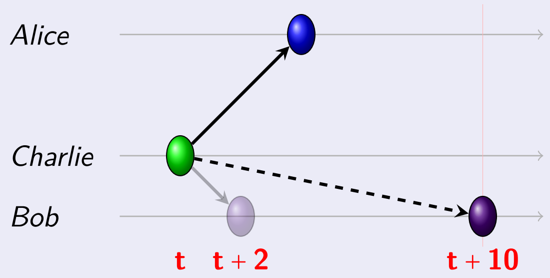
↓

Bob reactivates Charlie's account

Ordering Based on Time Bounds

Time bounds:

$bound_{Charlie,Bob} = 10$



Charlie deposits a sufficient sum

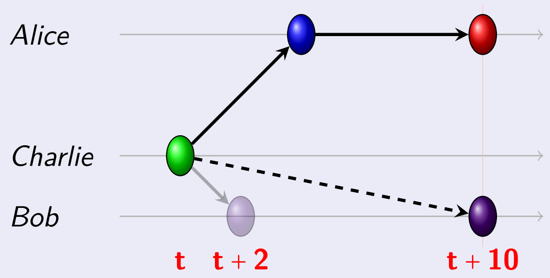
↓

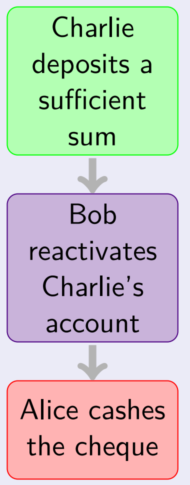
Bob reactivates Charlie's account

Ordering Based on Time Bounds

Time bounds:

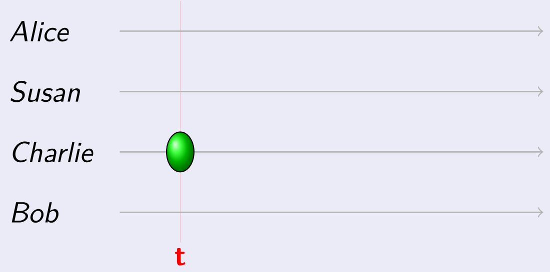
$bound_{Charlie,Bob} = 10$





Susan Steps In

Time bounds:

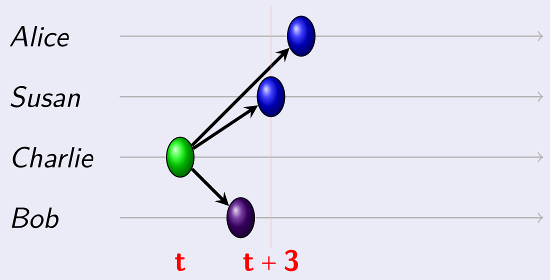


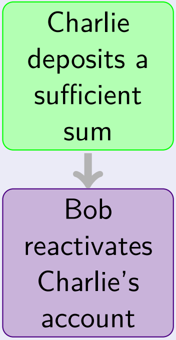
Charlie deposits a sufficient sum

Susan Steps In

Time bounds:

$bound_{Charlie,Bob} = 10$

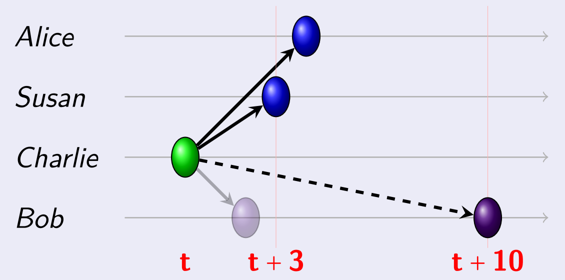




Susan Steps In

Time bounds:

$bound_{Charlie,Bob} = 10$



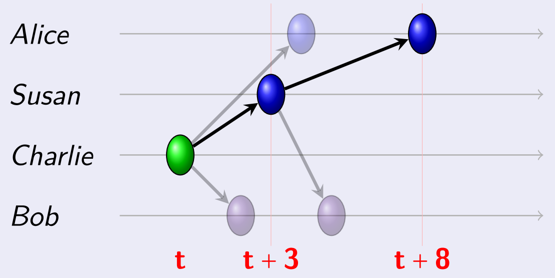
Charlie deposits a sufficient sum

Bob reactivates Charlie's account

Susan Steps In

Time bounds:

$bound_{Charlie,Bob} = 10$
$bound_{Susan,Bob} = 4$



Charlie deposits a sufficient sum

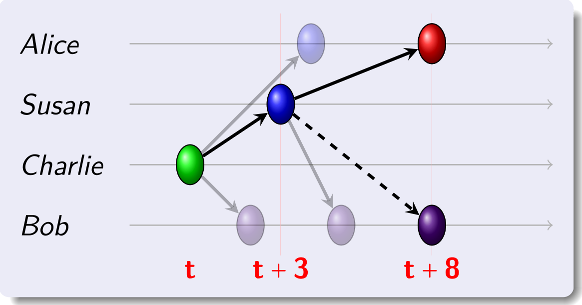
↓

Bob reactivates Charlie's account

Susan Steps In

Time bounds:

$bound_{Charlie,Bob} = 10$
$bound_{Susan,Bob} = 4$

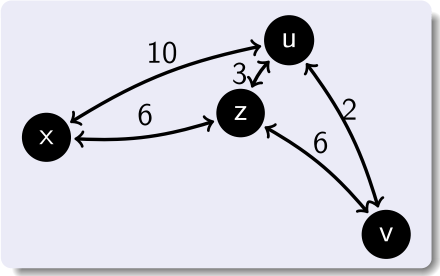


Charlie deposits a sufficient sum

Bob reactivates Charlie's account

Alice cashes the cheque

The Bound Guarantee Relation $\dashrightarrow$



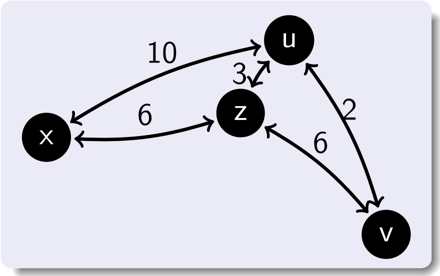
- Let D_{ij} = shortest *bound*-weighted path between i and j

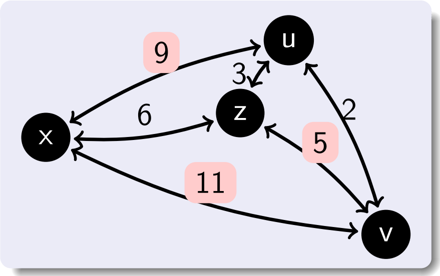
Definition (Bound Guarantees)

$\langle i, t\rangle \dashrightarrow \langle j, t'\rangle$ iff $t' \geq t + \mathrm{D}_{ij}$

There is enough time from t to t' to guarantee delivery from i to j

The Bound Guarantee Relation $\dashrightarrow$





- Let D_{ij} = shortest *bound*-weighted path between i and j

Definition (Bound Guarantees)

$\langle i, t\rangle \dashrightarrow \langle j, t'\rangle$ iff $t' \geq t + D_{ij}$

There is enough time from t to t' to guarantee delivery from i to j

The Bound Guarantee Relation $\dashrightarrow$

- Let D_{ij} = shortest *bound*-weighted path between i and j

Definition (Bound Guarantees)

$\langle i, t\rangle \dashrightarrow \langle j, t'\rangle$ iff $t' \geq t + D_{ij}$

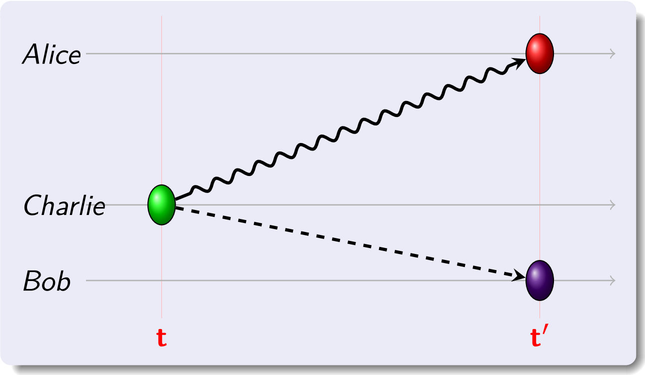
There is enough time from t to t' to guarantee delivery from i to j

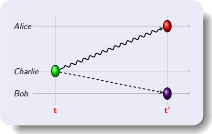
Revisiting the Frozen Account

Message chains vs. *Bound guarantees*

Revisiting the Frozen Account

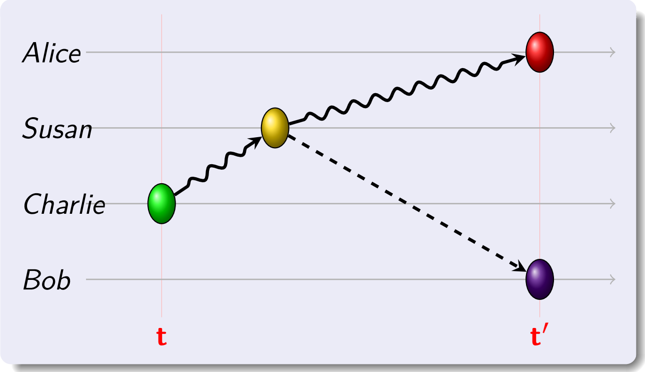
Charlie notifies and coordinates both responses

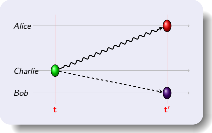


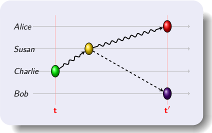


Revisiting the Frozen Account

Charlie notifies both, but Susan coordinates

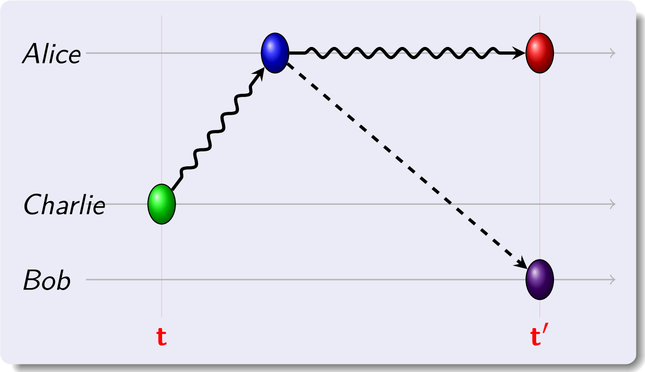


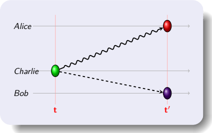


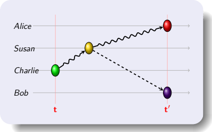


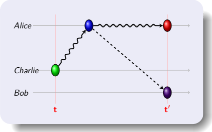
Revisiting the Frozen Account

Charlie notifies Alice,
who notifies and coordinates with Bob



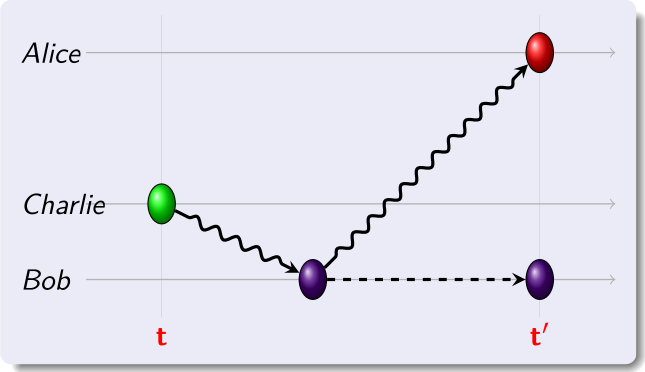


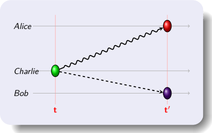


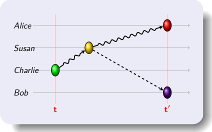


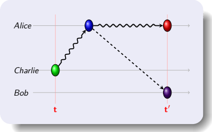
Revisiting the Frozen Account

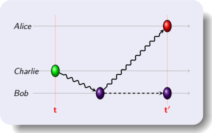
Charlie notifies Bob,
who notifies and coordinates with Alice





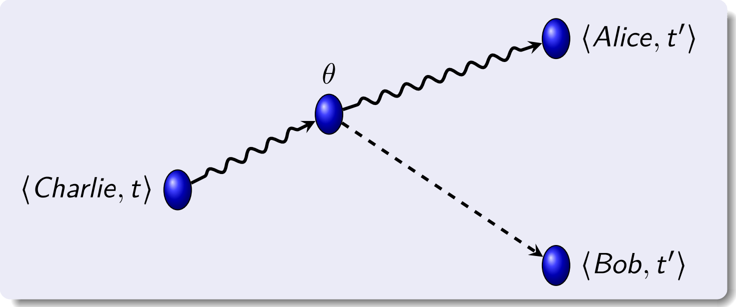


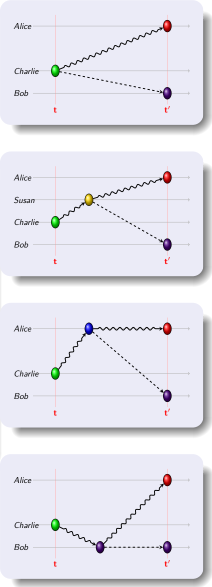




Revisiting the Frozen Account

The four patterns are instances of





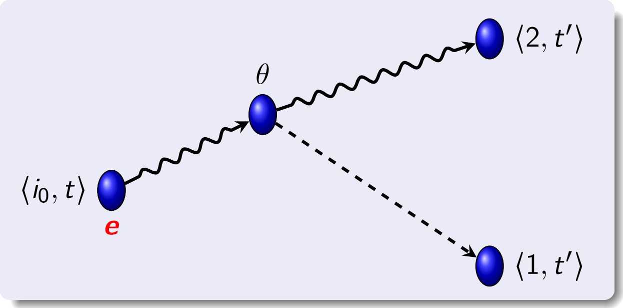
Knowledge Gain with Clocks

Theorem (Ben Zvi and M.)

Let R be a system with clocks and bounds and let e be a spontaneous event occurring at $\langle i_0, t\rangle$ in $r \in R$. If

$$(R, r, t') \vDash K_2 K_1 \mathrm{occ'd}(e)$$

then the following picture must hold in r:

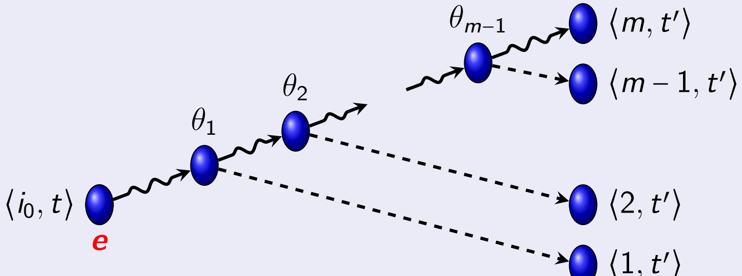


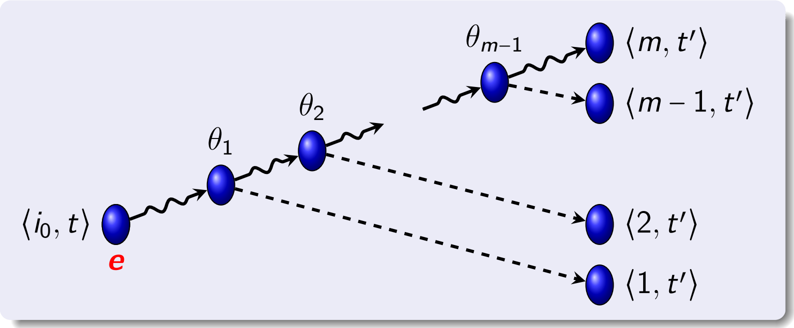
Theorem (Centipede Theorem)

Let R be a system with clocks and bounds, and let e be a spontaneous event occurring at $\langle i_0, t\rangle$ in $r \in R$. If

$$(R, r, t') \models K_m K_{m-1} \cdots K_1 \text{occ}'\text{d}(e)$$

then there is a centipede for $\langle i_0, 1, 2, \ldots, m\rangle$ in $r[t..t']$:





- Centipedes are the analogue of message chains in this model
- Centipedes are necessary for ordering action in this case

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad \mathtt{does}_2(\alpha_2)$ and
- $\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad \mathtt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$$
\begin{aligned}
\mathtt{does}_1(\alpha_1) &\quad \Rrightarrow \quad K_1\mathtt{does}_2(\alpha_2) \text{ by KoP} \\
\mathtt{does}_2(\alpha_2) &\quad \Rrightarrow \quad K_1\mathtt{does}_2(\alpha_2)\text{, so} \\
\mathtt{does}_2(\alpha_2) &\quad \Rrightarrow \quad K_2K_1\mathtt{does}_2(\alpha_2) \text{ by KoP} \\
\mathtt{does}_1(\alpha_1) &\quad \Rrightarrow \quad K_2K_1\mathtt{does}_2(\alpha_2)\text{, so} \\
\mathtt{does}_1(\alpha_1) &\quad \Rrightarrow \quad K_1K_2K_1\mathtt{does}_2(\alpha_2) \text{ by KoP} \\
&\cdots
\end{aligned}
$$

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad \mathtt{does}_2(\alpha_2)$ and
- $\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad \mathtt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$$\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1\mathtt{does}_2(\alpha_2) \text{ by KoP}$$

$$\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_1\mathtt{does}_2(\alpha_2), \text{ so}$$

$$\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_2K_1\mathtt{does}_2(\alpha_2) \text{ by KoP}$$

$$\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_2K_1\mathtt{does}_2(\alpha_2), \text{ so}$$

$$\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1K_2K_1\mathtt{does}_2(\alpha_2) \text{ by KoP}$$

$$\ldots$$

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad \mathtt{does}_2(\alpha_2)$ and
- $\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad \mathtt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$$\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1\mathtt{does}_2(\alpha_2) \text{ by KoP}$$

$$\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_1\mathtt{does}_2(\alpha_2)\text{, so}$$

$$\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_2K_1\mathtt{does}_2(\alpha_2) \text{ by KoP}$$

$$\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_2K_1\mathtt{does}_2(\alpha_2)\text{, so}$$

$$\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1K_2K_1\mathtt{does}_2(\alpha_2) \text{ by KoP}$$

$$\cdots$$

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad \texttt{does}_2(\alpha_2)$ and
- $\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad \texttt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1\texttt{does}_2(\alpha_2)$ by KoP

$\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_1\texttt{does}_2(\alpha_2)$, so

$\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_2K_1\texttt{does}_2(\alpha_2)$ by KoP

$\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_2K_1\texttt{does}_2(\alpha_2)$, so

$\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1K_2K_1\texttt{does}_2(\alpha_2)$ by KoP

$\dots$

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\mathtt{does}_1(\alpha_1) \Rrightarrow \mathtt{does}_2(\alpha_2)$ and
- $\mathtt{does}_2(\alpha_2) \Rrightarrow \mathtt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$\mathtt{does}_1(\alpha_1) \Rrightarrow K_1\mathtt{does}_2(\alpha_2)$ by KoP

$\mathtt{does}_2(\alpha_2) \Rrightarrow K_1\mathtt{does}_2(\alpha_2)$, so

$\mathtt{does}_2(\alpha_2) \Rrightarrow K_2K_1\mathtt{does}_2(\alpha_2)$ by KoP

$\mathtt{does}_1(\alpha_1) \Rrightarrow K_2K_1\mathtt{does}_2(\alpha_2)$, so

$\mathtt{does}_1(\alpha_1) \Rrightarrow K_1K_2K_1\mathtt{does}_2(\alpha_2)$ by KoP

...

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad \mathtt{does}_2(\alpha_2)$ and
- $\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad \mathtt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1\mathtt{does}_2(\alpha_2)$ by KoP

$\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_1\mathtt{does}_2(\alpha_2)$, so

$\mathtt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_2K_1\mathtt{does}_2(\alpha_2)$ by KoP

$\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_2K_1\mathtt{does}_2(\alpha_2)$, so

$\mathtt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1K_2K_1\mathtt{does}_2(\alpha_2)$ by KoP

...

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad \texttt{does}_2(\alpha_2)$ and
- $\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad \texttt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1\texttt{does}_2(\alpha_2)$ by KoP

$\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_1\texttt{does}_2(\alpha_2)$, so

$\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_2K_1\texttt{does}_2(\alpha_2)$ by KoP

$\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_2K_1\texttt{does}_2(\alpha_2)$, so

$\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1K_2K_1\texttt{does}_2(\alpha_2)$ by KoP

$\cdots$

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad \texttt{does}_2(\alpha_2)$ and
- $\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad \texttt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$$\begin{array}{lcl}
\texttt{does}_1(\alpha_1) & \Rrightarrow & K_1\texttt{does}_2(\alpha_2) \text{ by KoP} \\
\texttt{does}_2(\alpha_2) & \Rrightarrow & K_1\texttt{does}_2(\alpha_2)\text{, so} \\
\texttt{does}_2(\alpha_2) & \Rrightarrow & K_2K_1\texttt{does}_2(\alpha_2) \text{ by KoP} \\
\texttt{does}_1(\alpha_1) & \Rrightarrow & K_2K_1\texttt{does}_2(\alpha_2)\text{, so} \\
\texttt{does}_1(\alpha_1) & \Rrightarrow & K_1K_2K_1\texttt{does}_2(\alpha_2) \text{ by KoP} \\
& \cdots &
\end{array}$$

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad \texttt{does}_2(\alpha_2)$ and
- $\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad \texttt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1\texttt{does}_2(\alpha_2)$ by KoP

$\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_1\texttt{does}_2(\alpha_2)$, so

$\texttt{does}_2(\alpha_2) \quad \Rrightarrow \quad K_2K_1\texttt{does}_2(\alpha_2)$ by KoP

$\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_2K_1\texttt{does}_2(\alpha_2)$, so

$\texttt{does}_1(\alpha_1) \quad \Rrightarrow \quad K_1K_2K_1\texttt{does}_2(\alpha_2)$ by KoP

$\cdots$

Simultaneous Actions

Definition

Actions α_1 and α_2 are (necessarily) simultaneous in R if both

- $\mathtt{does}_1(\alpha_1) \Rrightarrow \mathtt{does}_2(\alpha_2)$ and
- $\mathtt{does}_2(\alpha_2) \Rrightarrow \mathtt{does}_1(\alpha_1)$.

Corollaries

Let α_1 and α_2 be simultaneous in R. Then

$\mathtt{does}_1(\alpha_1) \Rrightarrow K_1\mathtt{does}_2(\alpha_2)$ by KoP

$\mathtt{does}_2(\alpha_2) \Rrightarrow K_1\mathtt{does}_2(\alpha_2)$, so

$\mathtt{does}_2(\alpha_2) \Rrightarrow K_2K_1\mathtt{does}_2(\alpha_2)$ by KoP

$\mathtt{does}_1(\alpha_1) \Rrightarrow K_2K_1\mathtt{does}_2(\alpha_2)$, so

$\mathtt{does}_1(\alpha_1) \Rrightarrow K_1K_2K_1\mathtt{does}_2(\alpha_2)$ by KoP

...

Simultaneity Requires Common Knowledge

The agents in G have common knowledge of φ, denoted by $C_G\varphi$, if

$$K_{i_1} K_{i_2} \cdots K_{i_m} \varphi$$

holds for all sequences $\langle i_1, i_2, \ldots, i_m \rangle$ of agents in G, for all $m > 0$.

Theorem (Common Knowledge of Preconditions)

Suppose that $A = \{\alpha_i\}_{i \in G}$ are simultaneous actions in R.

If φ is a necessary condition for $\mathtt{does}_\mathtt{i}(\alpha_\mathtt{i})$ for some $\mathtt{i} \in G$, then $C_G\varphi$ is a necessary condition for $\mathtt{does}_j(\alpha_j)$, for all $j \in G$.

cf. [Halpern and M. '90]

Simultaneity Requires Common Knowledge

The agents in G have common knowledge of φ, denoted by $C_G\varphi$, if

$$K_{i_1} K_{i_2} \cdots K_{i_m} \varphi$$

holds for all sequences $\langle i_1, i_2, \ldots, i_m \rangle$ of agents in G, for all $m > 0$.

Theorem (Common Knowledge of Preconditions)

Suppose that $A = \{\alpha_i\}_{i \in G}$ are simultaneous actions in R.

If φ is a necessary condition for $\mathtt{does}_{\mathtt{i}}(\alpha_{\mathtt{i}})$ for some $\mathtt{i} \in G$, then
$C_G\varphi$ is a necessary condition for $\mathtt{does}_j(\alpha_j)$, for all $j \in G$.

cf. [Halpern and M. '90]

Knowledge and Coordination

Individual Action	$\Leftrightarrow$	Knowledge of Preconditions (**KoP**)
Ordered Action	$\Leftrightarrow$	Nested Knowledge of Preconditions
Simultaneous Action	$\Leftrightarrow$	Common Knowledge of Preconditions

Summary

- The **K*o*P** relates knowledge and action
- Knowledge is defined in a model-independent fashion
- Applies very broadly: Social science, Life sciences, VLSI design…
- Useful for designing efficient distributed protocols
- Effective for analyzing coordination
- Next step: Probabilistic variants of the **K*o*P**.

Thank You!

Summary

- The **K*o*P** relates knowledge and action
- Knowledge is defined in a model-independent fashion
- Applies very broadly: Social science, Life sciences, VLSI design...
- Useful for designing efficient distributed protocols
- Effective for analyzing coordination
- Next step: Probabilistic variants of the **K*o*P**.

Thank You!

Summary

- The **K*o*P** relates knowledge and action
- Knowledge is defined in a model-independent fashion
- Applies very broadly: Social science, Life sciences, VLSI design...
- Useful for designing efficient distributed protocols
- Effective for analyzing coordination
- Next step: Probabilistic variants of the **K*o*P**.

Thank You!

Summary

- The **K*o*P** relates knowledge and action
- Knowledge is defined in a model-independent fashion
- Applies very broadly: Social science, Life sciences, VLSI design. . .
- Useful for designing efficient distributed protocols
- Effective for analyzing coordination
- Next step: Probabilistic variants of the **K*o*P**.

Thank You!

Summary

- The **K*o*P** relates knowledge and action
- Knowledge is defined in a model-independent fashion
- Applies very broadly: Social science, Life sciences, VLSI design. . .
- Useful for designing efficient distributed protocols
- Effective for analyzing coordination
- Next step: Probabilistic variants of the **K*o*P**.

Thank You!

Summary

- The **KoP** relates knowledge and action
- Knowledge is defined in a model-independent fashion
- Applies very broadly: Social science, Life sciences, VLSI design...
- Useful for designing efficient distributed protocols
- Effective for analyzing coordination
- Next step: Probabilistic variants of the **KoP**.

Thank You!

Summary

- The **K*o*P** relates knowledge and action
- Knowledge is defined in a model-independent fashion
- Applies very broadly: Social science, Life sciences, VLSI design...
- Useful for designing efficient distributed protocols
- Effective for analyzing coordination
- Next step: Probabilistic variants of the **K*o*P**.

Thank You!

Summary

- The **K*o*P** relates knowledge and action
- Knowledge is defined in a model-independent fashion
- Applies very broadly: Social science, Life sciences, VLSI design...
- Useful for designing efficient distributed protocols
- Effective for analyzing coordination
- Next step: Probabilistic variants of the **K*o*P**.

Thank You!

References

Chandy, Mani and Jay Misra.
How Processes Learn.
Distributed Computing, 1986.

Fagin, Ronald, Joseph Y. Halpern, YM, and Moshe Y. Vardi.
Reasoning About Knowledge.
MIT press, 2003.

Ben-Zvi, Ido, and YM.
Beyond Lamport's Happened-before: On Time Bounds and the Ordering of Events in Distributed Systems.
Journal of the ACM (2014).

Castañeda, Armando, Yannai A. Gonczarowski, and YM.
Unbeatable Consensus.
Proceedings of DISC 2014.

Moses, Yoram.
Relating Knowledge and Coordinated Action: The Knowledge of Preconditions Principle.
Proceedings of TARK 2015, arXiv preprint arXiv:1606.07525 (2016).

Goren, Guy, and YM.
Silence.
Proceedings of PODC 2018