Chapter 8: Reasoning with Knowledge

Ajay Kshemkalyani and Mukesh Singhal

Distributed Computing: Principles, Algorithms, and Systems

Cambridge University Press

Muddy Children Puzzle (Scenario A)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario A:` Father says:: ψ: "At least one of you has mud on the forehead."

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let $c =$ clean child, $d =$ dirty child

- $k = 0$: contradicts ψ
- $k = 1$: In $r = 1$, the d answers "Yes". For $r = 2$, the c answer "No".
- $k = 2$: In $r = 1$, no responses. In $r = 2$, both d answer "Yes". In $r = 3$, the c answer "No"
- $k = 3$: In $r = 1, 2$, no responses. In $r = 3$, the 3 d answer "Yes". In $r = 4$, the $n - 3$ c answer "No".
- $k \leq n$: In $r < k$, no responses. In $r = k$, the k d answer "Yes". In $r = k + 1$, the $n - k$ c answer "No".

Muddy Children Puzzle (Scenario A)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

Scenario A: Father says:: ψ: "At least one of you has mud on the forehead."

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: contradicts ψ
- $k = 1$: In $r = 1$, the d answers "Yes". For $r = 2$, the c answer "No".
- $k = 2$: In $r = 1$, no responses. In $r = 2$, both d answer "Yes". In $r = 3$, the c answer "No"
- $k = 3$: In $r = 1, 2$, no responses. In $r = 3$, the 3 d answer "Yes". In $r = 4$, the $n-3$ c answer "No".
- $k \leq n$: In $r < k$, no responses. In $r = k$, the k d answer "Yes". In $r = k+1$, the $n-k$ c answer "No".

Muddy Children Puzzle (Scenario A)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario A:` Father says:: ψ: "At least one of you has mud on the forehead."

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let $c =$ clean child, $d =$ dirty child

- $k = 0$: contradicts ψ
- $k = 1$: In $r = 1$, the d answers "Yes". For $r = 2$, the c answer "No".
- $k = 2$: In $r = 1$, no responses. In $r = 2$, both d answer "Yes". In $r = 3$, the c answer "No"
- $k = 3$: In $r = 1, 2$, no responses. In $r = 3$, the 3 d answer "Yes". In $r = 4$, the $n - 3$ c answer "No".
- $k \leq n$: In $r < k$, no responses. In $r = k$, the k d answer "Yes". In $r = k + 1$, the $n - k$ c answer "No".

Muddy Children Puzzle (Scenario A)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario A`: Father says:: ψ: "At least one of you has mud on the forehead."

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: contradicts ψ
- $k = 1$: In $r = 1$, the d answers "Yes". For $r = 2$, the c answer "No".
- $k = 2$: In $r = 1$, no responses. In $r = 2$, both d answer "Yes". In $r = 3$, the c answer "No"
- $k = 3$: In $r = 1, 2$, no responses. In $r = 3$, the 3 d answer "Yes". In $r = 4$, the $n-3$ c answer "No".
- $k \leq n$: In $r < k$, no responses. In $r = k$, the k d answer "Yes". In $r = k+1$, the $n-k$ c answer "No".

Muddy Children Puzzle (Scenario A)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario A`: Father says:: ψ: "At least one of you has mud on the forehead."

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: contradicts ψ
- $k = 1$: In $r = 1$, the d answers "Yes". For $r = 2$, the c answer "No".
- $k = 2$: In $r = 1$, no responses. In $r = 2$, both d answer "Yes". In $r = 3$, the c answer "No"
- $k = 3$: In $r = 1, 2$, no responses. In $r = 3$, the 3 d answer "Yes". In $r = 4$, the $n-3$ c answer "No".
- $k \leq n$: In $r < k$, no responses. In $r = k$, the k d answer "Yes". In $r = k+1$, the $n-k$ c answer "No".

Muddy Children Puzzle (Scenario A)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario A`: Father says:: ψ: "At least one of you has mud on the forehead."

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: contradicts ψ
- $k = 1$: In $r = 1$, the d answers "Yes". For $r = 2$, the c answer "No".
- $k = 2$: In $r = 1$, no responses. In $r = 2$, both d answer "Yes". In $r = 3$, the c answer "No"
- $k = 3$: In $r = 1, 2$, no responses. In $r = 3$, the 3 d answer "Yes". In $r = 4$, the $n - 3$ c answer "No".
- $k \leq n$: In $r < k$, no responses. In $r = k$, the k d answer "Yes". In $r = k + 1$, the $n - k$ c answer "No".

Muddy Children Puzzle (Scenario A)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario A:` Father says:: ψ: "At least one of you has mud on the forehead."

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: contradicts ψ
- $k = 1$: In $r = 1$, the d answers "Yes". For $r = 2$, the c answer "No".
- $k = 2$: In $r = 1$, no responses. In $r = 2$, both d answer "Yes". In $r = 3$, the c answer "No"
- $k = 3$: In $r = 1, 2$, no responses. In $r = 3$, the 3 d answer "Yes". In $r = 4$, the $n - 3$ c answer "No".
- $k \leq n$: In $r < k$, no responses. In $r = k$, the k d answer "Yes". In $r = k + 1$, the $n - k$ c answer "No".

Muddy Children Puzzle: Scenario A Proof

First $k-1$ times the father asks "Do you have mud on your forehead?", all say "No".

kth time: the k muddy children say "Yes"

Proof by induction

- $k=1$: The muddy child, seeing no other muddy child, and knowing ψ, can answer "Yes"
- $k=2$: The first round, neither answers "Yes".
  $d1$ concludes that were he clean, $d2$ would have answered "Yes"
  $\Rightarrow$ $d1$ must be muddy.
  $\Rightarrow$ In round 2, $d1$ answers "Yes"
  (likewise reasoning for $d2$)
- $k=x$: Assume hypothesis is true.
- $k=x+1$: Each muddy child reasons as follows.
  ``If there were x muddy children, then they would all have answered 'Yes' when the question is asked for the x^{th} time. As that did not happen, there must be more than x muddy children. As I can see only x other muddy children, I myself must also be muddy. So I will answer 'Yes' when the question is asked the $x+1^{th}$ time.''

Muddy Children Puzzle (Scenario B)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario B:` Father does <u>not</u> say ψ.

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let $c =$ clean child, $d =$ dirty child

- $k = 0$: $\forall r$, no child answers "Yes"
- $k = 1$: In $r = 1$, no child (c and d) answers "Yes". In $r > 1$, no child (c and d) answers "Yes".
- $k = 2$: In $r = 1, 2$, no child (c and d) answers "Yes". In $r > 2$, no child (c and d) answers "Yes".
- $k = 3$: In $r = 1, 2, 3$, no child (c and d) answers "Yes". In $r > 3$, no child (c and d) answers "Yes".
- $k \leq n$: $\forall r$, no child answers "Yes"

Muddy Children Puzzle (Scenario B)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario B:` Father does <u>not</u> say ψ.

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: $\forall r$, no child answers "Yes"
- $k = 1$: In $r = 1$, no child (c and d) answers "Yes". In $r > 1$, no child (c and d) answers "Yes".
- $k = 2$: In $r = 1, 2$, no child (c and d) answers "Yes". In $r > 2$, no child (c and d) answers "Yes".
- $k = 3$: In $r = 1, 2, 3$, no child (c and d) answers "Yes". In $r > 3$, no child (c and d) answers "Yes".
- $k \leq n$: $\forall r$, no child answers "Yes"

Muddy Children Puzzle (Scenario B)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

Scenario B: Father does not say ψ.

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: $\forall r$, no child answers "Yes"
- $k = 1$: In $r = 1$, no child (c and d) answers "Yes". In $r > 1$, no child (c and d) answers "Yes".
- $k = 2$: In $r = 1, 2$, no child (c and d) answers "Yes". In $r > 2$, no child (c and d) answers "Yes".
- $k = 3$: In $r = 1, 2, 3$, no child (c and d) answers "Yes". In $r > 3$, no child (c and d) answers "Yes".
- $k \leq n$: $\forall r$, no child answers "Yes"

Muddy Children Puzzle (Scenario B)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario B:` Father does not say ψ.

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: $\forall r$, no child answers "Yes"
- $k = 1$: In $r = 1$, no child (c and d) answers "Yes".
  In $r > 1$, no child (c and d) answers "Yes".
- $k = 2$: In $r = 1, 2$, no child (c and d) answers "Yes".
  In $r > 2$, no child (c and d) answers "Yes".
- $k = 3$: In $r = 1, 2, 3$, no child (c and d) answers "Yes".
  In $r > 3$, no child (c and d) answers "Yes".
- $k \leq n$: $\forall r$, no child answers "Yes"

Muddy Children Puzzle (Scenario B)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario B:` Father does not say ψ.

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: $\forall r$, no child answers "Yes"
- $k = 1$: In $r = 1$, no child (c and d) answers "Yes".
  In $r > 1$, no child (c and d) answers "Yes".
- $k = 2$: In $r = 1, 2$, no child (c and d) answers "Yes".
  In $r > 2$, no child (c and d) answers "Yes".
- $k = 3$: In $r = 1, 2, 3$, no child (c and d) answers "Yes".
  In $r > 3$, no child (c and d) answers "Yes".
- $k \leq n$: $\forall r$, no child answers "Yes"

Muddy Children Puzzle (Scenario B)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario B:` Father does not say ψ.

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: $\forall r$, no child answers "Yes"
- $k = 1$: In $r = 1$, no child (c and d) answers "Yes".
  In $r > 1$, no child (c and d) answers "Yes".
- $k = 2$: In $r = 1, 2$, no child (c and d) answers "Yes".
  In $r > 2$, no child (c and d) answers "Yes".
- $k = 3$: In $r = 1, 2, 3$, no child (c and d) answers "Yes".
  In $r > 3$, no child (c and d) answers "Yes".
- $k \leq n$: $\forall r$, no child answers "Yes"

Muddy Children Puzzle (Scenario B)

- n children, all intelligent, can see others but not their own faces
- k ($\leq n$) have mud on their forehead

`Scenario B:` Father does not say ψ.

- Father then repeatedly asks (i.e., broadcasts) in rounds (to model synchronous operation) to the assembled children:
  - Do you have mud on your forehead?
- How does each child respond in each round, $r = 1, 2, \ldots k-1, k, k+1, \ldots n, n+1, \ldots$? An answer is "broadcast" in that round.
- Let c = clean child, d= dirty child

- $k = 0$: $\forall r$, no child answers "Yes"
- $k = 1$: In $r = 1$, no child (c and d) answers "Yes".
  In $r > 1$, no child (c and d) answers "Yes".
- $k = 2$: In $r = 1, 2$, no child (c and d) answers "Yes".
  In $r > 2$, no child (c and d) answers "Yes".
- $k = 3$: In $r = 1, 2, 3$, no child (c and d) answers "Yes".
  In $r > 3$, no child (c and d) answers "Yes".
- $k \leq n$: $\forall r$, no child answers "Yes"

Muddy Children Puzzle: Scenario B Proof

Every time the father asks "Do you have mud on your forehead?", all say "No".
Proof by induction on # times q the father asks the question.

- $q = 1$: each child answers "No" because he cannot distinguish the two cases: he has and does not have mud on his forehead.
- $q = x$: Assume hypothesis is true.
- $q = x + 1$: the situation is unchanged because each child has no further knowledge to distinguish the two cases.

Why is Scenario B different from A?

- A: Father announcing ϕ introduces "common knowledge" of ψ, i.e., everyone knows everyone knows ... (infinitely often) everyone knows ψ is true
This allows children to reason and reach correct answer.
- B: Father does not announce ϕ. No common knowledge of ψ.
Children have no basis to start their reasoning process.

Logic of Knowledge

- Identify set of possible worlds (possible universes) and relationships between them
- At a process (in any global state): possible worlds are the global states which the process thinks consistent with its local state
- States expressible as logical formulae over facts ϕ
  - primitive proposition or formula including $\wedge, \vee, \neg$, knowledge operator K, everybody knows operator E
  - $K_i(\phi)$: process P_i knows ϕ
  - $E_i^1(\phi) = \bigwedge_{i \in N} K_i(\phi)$, every process knows ϕ
  - $E^2(\phi) = E\ (E^1(\phi))$, i.e., every process knows $E^1(\phi)$.
  - $E^k(\phi) = E^{k-1}\ (E^1(\phi))$ for $k > 1$.
- hierarchy of levels of knowledge $E^j(\phi)$ ($j \in Z*$), where $Z*$ is $\{0, 1, 2, 3, \ldots\}$.
- $E^{k+1}(\phi) \Longrightarrow E^k(\phi)$.
- Common knowledge $C(\phi)$: a state of knowledge X satisfying $X = E(\phi \wedge X)$. Captures notion of agreement.
- $C(\phi) \Longrightarrow \bigwedge_{j \in Z*} E^j(\phi)$.

Muddy Children Puzzle: Using Knowledge

- Each child sees at least $k-1$ muddy children $\Longrightarrow E^{k-1}(\psi)$
- A muddy child does not see k muddy children $\Longrightarrow \neg E^{k}(\psi)$
- Above is Scenario B. $E^{k-1}(\psi)$ not adequate for muddy children to ever answer "Yes"
- To answer "Yes," $E^{k}(\Psi)$ is required so that the children can progressively reason and answer correctly in the k^{th} round.
- In Scenario A: Father announcing ψ provided $C(\psi)$ which implied $E^{k}(\Psi)$

Kripke Structures (informal)

Labeled graph with labeled nodes

- set of nodes is the set of states
- label of a node s: set of propositions that are true and false at s
- label of edge (s, t): ID of each process that cannot distinguish between s and t
- Assume bidirectional edges and reflexive graph

Reachability of states

1. State t is reachable from state s in k steps if there exist states $s_0, s_1, \ldots, s_k$ such that $s_0 = s$, $s_k = t$, and for all $j \in [0, k-1]$, there exists some P_i such that $(s_j, s_{j+1}) \in \mathcal{K}_i$.
2. State t is reachable from state s if t is reachable from s in k steps, for some $k > 1$.

Muddy Children Puzzle: Using Kripke Structures

Assume $n = 3, k = 2$, actual state is $(1, 1, 0)$

- $(1, 1, 0) \models \neg E^2(\psi)$ because world $(0, 0, 0)$ is 2-reachable and ψ is false here
  - Child 2 believes $(1, 0, 0)$ possible; here child 1 believes $(0, 0, 0)$ possible
- $E^{k-1}(\psi)$ is true: each world reachable in $k - 1$ hops has at least one '1'
- $E^k(\psi)$ is false: world $(0, \ldots 0)$ reachable in k hops

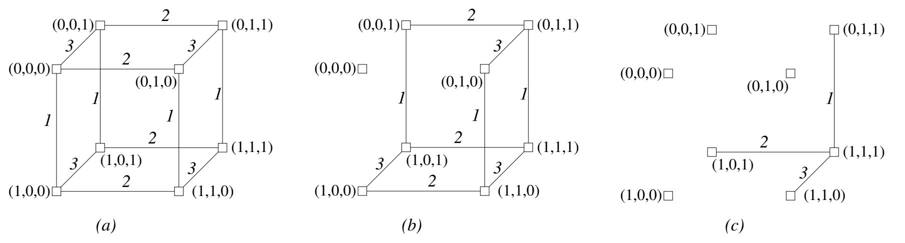

Fig 6.2: (a) Kripke structure. (b) After father announces ψ (Scenario A) (c) After round one (Scenario A)

Muddy Children Puzzle: Scenario A

Father announces ψ means common knowledge that 1 child has mud on his face

- $\Longrightarrow$ delete all edges connecting $(0,0,0)$ (change in group knowledge)
- After round 1 where all children say "No": all edges to all possible worlds with a single '1' get deleted
  - if there were a single muddy child, he would have answered "Yes" in round 1
  - now common knowledge that ≥ 2 muddy children
- After round x where all children say "No": all edges to all possible worlds with $\leq x$' '1's get deleted
  - now common knowledge that $\geq x + 1$ muddy children
- if there were x muddy children, they would have answered "Yes" in round x because they see $x - 1$ muddy children and rule out a world in which they are clean

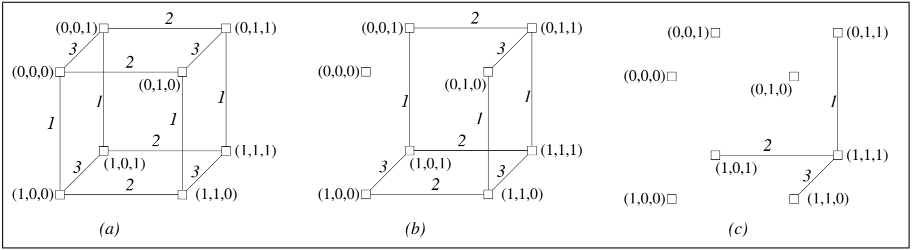

Fig 6.2: Actual state $(1,0,0)$. (a) Kripke structure. (b) After father announces ψ (Scenario A)

Muddy Children Puzzle: Scenarios A and B

Scenario A:
If in any iteration, it becomes common knowledge that world t is impossible, for each world s reachable from actual world r, edge (s, t) is deleted

Scenario B:
Children's state of knowledge never changes

- After the first question, each child is unsure of he is in '0' or '1' state
- This was same before the first question
- First round adds no new knowledge
- Inductively, same for subsequent rounds

No change in Kripke structure

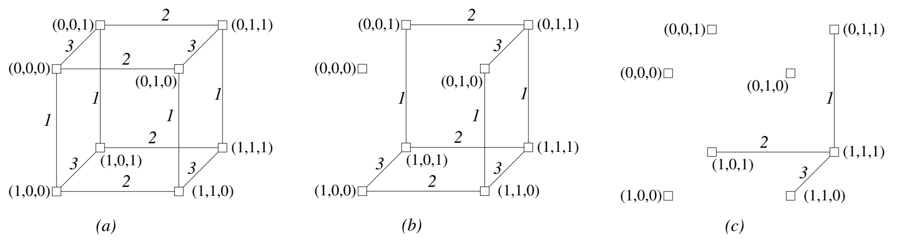

Fig 6.2: Actual state $(1, 0, 0)$. (a) Kripke structure. (b) After father announces ψ (Scenario A) (c) After round one (Scenario A)

Axioms of S5 Modal Logic

- Distribution Axiom: $K_i\psi \wedge K_i(\psi \Longrightarrow \phi) \Longrightarrow K_i\phi$
- Knowledge Axiom: $K_i\psi \Longrightarrow \psi$
  If a process knows a fact, then the fact is true. If $K_i\psi$ is true in a particular state, then ψ is true in all states the process considers possible.
- Positive Introspection Axiom: $K_i\psi \Longrightarrow K_iK_i\psi$
- Negative Introspection Axiom: $\neg K_i\psi \Longrightarrow K_i\neg K_i\psi$
- Knowledge Generalization Rule: For a valid formula or fact ψ, $K_i\psi$
  If ψ is true in all possible worlds, then ψ must be true in all the possible worlds with respect to any process and any given world.
  Assumption: a process knows all *valid* formulas, which are necessarily true.

Knowledge in Synchronous vs. Asynchronous Systems

Thus far, synchronous systems considered.
How to attain common knowledge in synchronous systems?

- Initialize all with common knowledge of ϕ
- Broadcast ϕ in a round of communication, and let all know that ϕ is being broadcast. Each process can begin supporting common knowledge from the next round.

Asynchronous system:

- possible worlds: the consistent cuts of the set of possible executions.
- Let (a, c) denote a cut c in asynchronous execution a.
- (a, c) also denotes the system state after (a, c).
- $(a, c)_i$: projection (i.e., state) of c on process i.
- Cuts c and c' are indistinguishable by process i, denoted $(a, c) \sim_i (a', c')$, if and only if $(a, c)_i = (a', c')_i$.
- The semantics of knowledge based on *asynchronous executions*, instead of timed executions.
- $K_i(\phi)$: ϕ is true in all possible consistent global states that include i's local state.
- Similarly for $E^k(\phi)$.

Knowledge in Asynchronous Systems: Logic, Definitions (1)

- $(a, c) \models \phi$ if and only if ϕ is true in cut c of asynchronous execution a.
- $(a, c) \models K_i(\phi)$ if and only if $\forall(a', c')$, $((a', c') \sim_i (a, c) \Longrightarrow (a', c') \models \phi)$
- $(a, c) \models E^0(\phi)$ if and only if $(a, c) \models \phi$
- $(a, c) \models E^1(\phi)$ if and only if $(a, c) \models \bigwedge_{i \in N} K_i(\phi)$
- $(a, c) \models E^{k+1}(\phi)$ for $k \geq 1$ if and only if $(a, c) \models \bigwedge_{i \in N} K_i(E^k(\phi))$, for $k \geq 1$
- $(a, c) \models C(\phi)$ if and only if $(a, c) \models$ the greatest fixed point knowledge X satisfying $X = E(X \wedge \phi)$.
  $C(\phi)$ implies $\wedge_{k \in Z*} E^k(\phi)$.

Knowledge in Asynchronous Systems: Logic, Definitions (2)

- "i knows ϕ in state s_i^x", denoted $s_i^x \models \phi$, is shorthand for $(\forall(a,c))$ $((a,c)_i = s_i^x \Longrightarrow (a,c) \models \phi)$.
- $s_i^x \models K_i(\phi)$ is shorthand for $(\forall(a,c))$ $((a,c)_i = s_i^x \Longrightarrow (a,c) \models K_i(\phi))$.
- Learning: Process i *learns* ϕ in state s_i^x of execution a if i knows ϕ in s_i^x and, for all states s_i^y in execution a such that $y < x$, i does not know ϕ.
- i attains ϕ: process learns ϕ in the present or an earlier state.
- ϕ is attained in an execution a: $\exists c, (a,c) \models \phi$
- Local fact: ϕ is *local* to process i in system A if $A \models (\phi \Longrightarrow K_i\phi)$ e.g., local state, clock value of a process, local component of vector clock
- Global fact: A fact that is not local, e.g., global state, timestamp of a cut

Common Knowledge in Asynchronous Systems

Reaching consensus over ϕ requires common knowledge of ϕ

Impossibility Result

There does not exist any protocol for two processes to reach common knowledge about a binary value in an asynchronous message-passing system with unreliable communication.

- Justify: P_i and P_j need to send each other ACKs ... nonterminating argument
- `or` Let there be a *minimal* protocol that has k msgs. Then the kth msg is redundant $\Rightarrow$ contradiction

Is common knowledge attainable in the async system with `reliable` communication without an upper bound on message transmission times?

- No. construct a similar argument

Is common knowledge attainable in the async system with `reliable` communication with an upper bound on message transmission times?

- No, for when does a process begin supporting that knowledge?

Common Knowledge in Asynchronous Systems

Reaching consensus over ϕ requires common knowledge of ϕ

Impossibility Result

There does not exist any protocol for two processes to reach common knowledge about a binary value in an asynchronous message-passing system with unreliable communication.

- Justify: P_i and P_j need to send each other ACKs ... nonterminating argument
- or Let there be a *minimal* protocol that has k msgs. Then the kth msg is redundant $\Rightarrow$ contradiction

Is common knowledge attainable in the async system with `reliable` communication without an upper bound on message transmission times?

- No. construct a similar argument

Is common knowledge attainable in the async system with `reliable` communication with an upper bound on message transmission times?

- No, for when does a process begin supporting that knowledge?

Common Knowledge in Asynchronous Systems

Reaching consensus over ϕ requires common knowledge of ϕ

Impossibility Result

There does not exist any protocol for two processes to reach common knowledge about a binary value in an asynchronous message-passing system with unreliable communication.

- Justify: P_i and P_j need to send each other ACKs ... nonterminating argument
- or Let there be a *minimal* protocol that has k msgs. Then the kth msg is redundant $\Rightarrow$ contradiction

Is common knowledge attainable in the async system with `reliable` communication without an upper bound on message transmission times?

- No. construct a similar argument

Is common knowledge attainable in the async system with `reliable` communication with an upper bound on message transmission times?

- No, for when does a process begin supporting that knowledge?

Common Knowledge in Asynchronous Systems

Reaching consensus over ϕ requires common knowledge of ϕ

Impossibility Result

There does not exist any protocol for two processes to reach common knowledge about a binary value in an asynchronous message-passing system with unreliable communication.

- Justify: P_i and P_j need to send each other ACKs ... nonterminating argument
- `or` Let there be a *minimal* protocol that has k msgs. Then the kth msg is redundant $\Rightarrow$ contradiction

Is common knowledge attainable in the async system with `reliable` communication without an upper bound on message transmission times?

- No. construct a similar argument

Is common knowledge attainable in the async system with `reliable` communication with an upper bound on message transmission times?

- No, for when does a process begin supporting that knowledge?

Common Knowledge in Asynchronous Systems

Reaching consensus over ϕ requires common knowledge of ϕ

Impossibility Result

There does not exist any protocol for two processes to reach common knowledge about a binary value in an asynchronous message-passing system with unreliable communication.

- Justify: P_i and P_j need to send each other ACKs ... nonterminating argument
- `or` Let there be a *minimal* protocol that has k msgs. Then the kth msg is redundant $\Rightarrow$ contradiction

Is common knowledge attainable in the async system with `reliable` communication without an upper bound on message transmission times?

- No. construct a similar argument

Is common knowledge attainable in the async system with `reliable` communication with an upper bound on message transmission times?

- No, for when does a process begin supporting that knowledge?

Common Knowledge in Asynchronous Systems

Reaching consensus over ϕ requires common knowledge of ϕ

Impossibility Result

There does not exist any protocol for two processes to reach common knowledge about a binary value in an asynchronous message-passing system with unreliable communication.

- Justify: P_i and P_j need to send each other ACKs ... nonterminating argument
- or Let there be a *minimal* protocol that has k msgs. Then the kth msg is redundant $\Rightarrow$ contradiction

Is common knowledge attainable in the async system with `reliable` communication without an upper bound on message transmission times?

- No. construct a similar argument

Is common knowledge attainable in the async system with `reliable` communication with an upper bound on message transmission times?

- No, for when does a process begin supporting that knowledge?

Variants of Common Knowledge for Asynchronous Systems

Common knowledge requires "simultaneity of actions" across processes.
Perfectly synchronized clocks not practical. But we can `weaken` common knowledge!

- Epsilon-common knowledge: $C^{\epsilon}(\phi)$ is the greatest fixed point of $X = E^{\epsilon}(\phi \wedge X)$
  - E^{ϵ} denotes "everyone knows within ϵ time units"
  - Assumes timed runs
- Eventual common knowledge: $C^{\diamond}(\phi)$ is the greatest fixed point of $X = E^{\diamond}(\phi \wedge X)$
  - $E^{\diamond}$ denotes "everyone will eventually know (at some point in their execution)"
  - reach agreement at some (not necessarily consistent) global state
- Timestamped common knowledge: $C^{T}(\phi)$ is the greatest fixed point of $X = E^{T}(\phi \wedge X)$
  - processes reach agreement at local states having the same local clock value.
  - It is applicable to asynchronous systems
  - $E^{T}(\phi) = \wedge_i K_i^{T}(\phi)$, where $K_i^{T}(\phi)$: process i knows ϕ at local clock value T
- Concurrent common knowledge $C^{C}(\phi)$: processes reach agreement at local states that belong to a consistent cut. When P_i attains $C^{C}(\phi)$, it also knows that each other process P_j has also attained the same concurrent common knowledge in its local state which is consistent with P_i's local state.
  - Most widely used weakening of common knowledge; studied next

Variants of Common Knowledge for Asynchronous Systems

Common knowledge requires "simultaneity of actions" across processes.
Perfectly synchronized clocks not practical. But we can `weaken` common knowledge!

- Epsilon-common knowledge: $C^{\epsilon}(\phi)$ is the greatest fixed point of $X = E^{\epsilon}(\phi \wedge X)$
  - E^{ϵ} denotes "everyone knows within ϵ time units"
  - Assumes timed runs
- Eventual common knowledge: $C^{\diamond}(\phi)$ is the greatest fixed point of $X = E^{\diamond}(\phi \wedge X)$
  - $E^{\diamond}$ denotes "everyone will eventually know (at some point in their execution)"
  - reach agreement at some (not necessarily consistent) global state
- Timestamped common knowledge: $C^{T}(\phi)$ is the greatest fixed point of $X = E^{T}(\phi \wedge X)$
  - processes reach agreement at local states having the same local clock value.
  - It is applicable to asynchronous systems
  - $E^{T}(\phi) = \wedge_i K_i^{T}(\phi)$, where $K_i^{T}(\phi)$: process i knows ϕ at local clock value T
- Concurrent common knowledge $C^{C}(\phi)$: processes reach agreement at local states that belong to a consistent cut. When P_i attains $C^{C}(\phi)$, it also knows that each other process P_j has also attained the same concurrent common knowledge in its local state which is consistent with P_i's local state.
  - Most widely used weakening of common knowledge; studied next

Variants of Common Knowledge for Asynchronous Systems

Common knowledge requires "simultaneity of actions" across processes.
Perfectly synchronized clocks not practical. But we can `weaken` common knowledge!

- Epsilon-common knowledge: $C^{\epsilon}(\phi)$ is the greatest fixed point of $X = E^{\epsilon}(\phi \wedge X)$
  - E^{ϵ} denotes "everyone knows within ϵ time units"
  - Assumes timed runs
- Eventual common knowledge: $C^{\diamond}(\phi)$ is the greatest fixed point of $X = E^{\diamond}(\phi \wedge X)$
  - $E^{\diamond}$ denotes "everyone will eventually know (at some point in their execution)"
  - reach agreement at some (not necessarily consistent) global state
- Timestamped common knowledge: $C^{T}(\phi)$ is the greatest fixed point of $X = E^{T}(\phi \wedge X)$
  - processes reach agreement at local states having the same local clock value.
  - It is applicable to asynchronous systems
  - $E^{T}(\phi) = \wedge_i K_i^{T}(\phi)$, where $K_i^{T}(\phi)$: process i knows ϕ at local clock value T
- Concurrent common knowledge $C^{C}(\phi)$: processes reach agreement at local states that belong to a consistent cut. When P_i attains $C^{C}(\phi)$, it also knows that each other process P_j has also attained the same concurrent common knowledge in its local state which is consistent with P_i's local state.
  - Most widely used weakening of common knowledge; studied next

Variants of Common Knowledge for Asynchronous Systems

Common knowledge requires "simultaneity of actions" across processes.
Perfectly synchronized clocks not practical. But we can `weaken` common knowledge!

- Epsilon-common knowledge: $C^{\epsilon}(\phi)$ is the greatest fixed point of $X = E^{\epsilon}(\phi \wedge X)$
  - E^{ϵ} denotes "everyone knows within ϵ time units"
  - Assumes timed runs
- Eventual common knowledge: $C^{\diamond}(\phi)$ is the greatest fixed point of $X = E^{\diamond}(\phi \wedge X)$
  - $E^{\diamond}$ denotes "everyone will eventually know (at some point in their execution)"
  - reach agreement at some (not necessarily consistent) global state
- Timestamped common knowledge: $C^{T}(\phi)$ is the greatest fixed point of $X = E^{T}(\phi \wedge X)$
  - processes reach agreement at local states having the same local clock value.
  - It is applicable to asynchronous systems
  - $E^{T}(\phi) = \wedge_i K_i^{T}(\phi)$, where $K_i^{T}(\phi)$: process i knows ϕ at local clock value T
- Concurrent common knowledge $C^{C}(\phi)$: processes reach agreement at local states that belong to a consistent cut. When P_i attains $C^{C}(\phi)$, it also knows that each other process P_j has also attained the same concurrent common knowledge in its local state which is consistent with P_i's local state.
  - Most widely used weakening of common knowledge; studied next

Variants of Common Knowledge for Asynchronous Systems

Common knowledge requires "simultaneity of actions" across processes.
Perfectly synchronized clocks not practical. But we can `weaken` common knowledge!

- Epsilon-common knowledge: $C^{\epsilon}(\phi)$ is the greatest fixed point of $X = E^{\epsilon}(\phi \wedge X)$
  - E^{ϵ} denotes "everyone knows within ϵ time units"
  - Assumes timed runs
- Eventual common knowledge: $C^{\diamond}(\phi)$ is the greatest fixed point of $X = E^{\diamond}(\phi \wedge X)$
  - $E^{\diamond}$ denotes "everyone will eventually know (at some point in their execution)"
  - reach agreement at some (not necessarily consistent) global state
- Timestamped common knowledge: $C^{T}(\phi)$ is the greatest fixed point of $X = E^{T}(\phi \wedge X)$
  - processes reach agreement at local states having the same local clock value.
  - It is applicable to asynchronous systems
  - $E^{T}(\phi) = \wedge_i K_i^{T}(\phi)$, where $K_i^{T}(\phi)$: process i knows ϕ at local clock value T
- Concurrent common knowledge $C^{C}(\phi)$: processes reach agreement at local states that belong to a consistent cut. When P_i attains $C^{C}(\phi)$, it also knows that each other process P_j has also attained the same concurrent common knowledge in its local state which is consistent with P_i's local state.
  - Most widely used weakening of common knowledge; studied next

Concurrent Common Knowledge: Definition

- $(a, c) \models \phi$ if and only if ϕ is true in cut c of execution a.
- $(a, c) \models K_i(\phi)$ if and only if $\forall(a', c')$, $((a', c') \sim_i (a, c) \Longrightarrow (a', c') \models \phi)$
- $(a, c) \models P_i(\phi)$ if and only if $\exists(a, c')$, $((a, c') \sim_i (a, c) \wedge (a, c') \models \phi)$
- $(a, c) \models E^{C^0}(\phi)$ if and only if $(a, c) \models \phi$
- $(a, c) \models E^{C^1}(\phi)$ if and only if $(a, c) \models \bigwedge_{i \in N} K_i P_i(\phi)$
- $(a, c) \models E^{C^{k+1}}(\phi)$ for $k \geq 1$ if and only if $(a, c) \models \bigwedge_{i \in N} K_i P_i(E^{C^k}(\phi))$, for $k \geq 1$
- $(a, c) \models C^C(\phi)$ if and only if $(a, c) \models$ the greatest fixed point knowledge X satisfying $X = E^C(X \wedge \phi)$.
  $C^C(\phi)$ implies $\wedge_{k \in Z*}(E^C)^k(\phi)$.

Concurrent Knowledge

- Possibly operator $P_i(\phi)$ means "ϕ is true in *some* consistent state in the same asynchronous run, that includes process i's local state".
- $E^C(\phi)$ is defined as $\bigwedge_{i \in N} K_i(P_i(\phi))$.
- $E^C(\phi)$: every process at the (given) cut knows only that ϕ is true in *some* cut that is consistent with its own local state.
- Concurrent knowledge is weaker than regular knowledge
  - But, for a *local*, *stable* fact, and assuming other processes learn the fact via message chains, the two are equivalent
- $C^C(\phi)$ is attained at a consistent cut:
  (informally speaking), each process at its local cut state knows that "in some state consistent with its own local cut state, ϕ is true *and that* all other process know all this same knowledge (described within quotes)".
- $C^C(\phi)$ underlies all protocols that reach agreement about properties of the global state

Concurrent Common Knowledge: Snapshot-based Algorithm

Protocol 1 (Snapshot-based algorithm).

1. At some time when the initiator I knows ϕ:
   - it sends a marker $MARKER(I, \phi, CCK)$ to each neighbour P_j, and atomically reaches its *cut state*.
2. When a process P_i receives for the first time, a message $MARKER(I, \phi, CCK)$ from a process P_j:
   - process P_i forwards the message to all of its neighbours except P_j, and atomically reaches its *cut state*.

- attains $C^C(\phi)$ when it reaches its *cut state*.
- Complexity: $2l$ messages; time complexity: $O(d)$

Concurrent Common Knowledge: Three-phase Send Inhibitory Algorithm

Protocol 2 (Three-phase send-inhibitory algorithm).

1. At some time when the initiator I knows ϕ:
   - it sends a marker $PREPARE(I, \phi, CCK)$ to each process P_j.
2. When a (non-initiator) process receives a marker $PREPARE(I, \phi, CCK)$:
   - it begins send-inhibition for non-protocol events.
   - sends a marker $CUT(I, \phi, CCK)$ to the initiator I.
   - it reaches its *cut state* at which it attains $C^C(\phi)$.
3. When the initiator I receives a marker $CUT(I, \phi, CCK)$ from each other process:
   - the initiator reaches its *cut state*
   - sends a marker $RESUME(I, \phi, CCK)$ to all other processes.
4. When a (non-initiator) process receives a marker $RESUME(I, \phi, CCK)$:
   - it resumes sending its non-protocol messages which had been inhibited in step 2.

- attains $C^C(\phi)$ when it reaches its *cut state*. Needs FIFO.
- Complexity: $3(n-1)$ messages; time complexity: 3 hops; send-inhibitory

Concurrent Common Knowledge: Three-phase Send Inhibitory Tree Algorithm

Protocol 3 (Three-phase send-inhibitory tree algorithm).

Phase I (broadcast): The root initiates *PREPARE* control messages down the ST; when a process receives such a message, it inhibits computation message sends and propagates the received control message down the ST.

Phase II (convergecast): A leaf node initiates this phase after it receives the *PREPARE* control message broadcast in phase I. The leaf reaches and records its *cut state*, and sends a *CUT* control message up the ST. An intermediate (and the root) node reaches and records its *cut state* when it receives such a *CUT* control message from each of its children, and then propagates the control message up the ST.

Phase III (broadcast): The root initiates a broadcast of a *RESUME* control message down the ST after Phase II terminates. On receiving such a *RESUME* message, a process resumes inhibited computation message send activity and propagates the control message down the ST.

- attains $C^C(\phi)$ when it reaches its *cut state*. non-FIFO.
- Complexity: $3(n-1)$ messages; time complexity: $O(depth)$ hops; send-inhibitory

Concurrent Common Knowledge: Inhibitory Ring Algorithm

Protocol 4 (Send-inhibitory ring algorithm).

1. Once a fact ϕ about the system state is known to some process, the process atomically reaches its *cut state* and begins supporting $C(\phi)$, begins send inhibition, and sends a control message $CUT(\phi)$ along the ring.
2. This $CUT(\phi)$ message announces ϕ. When a process receives the $CUT(\phi)$ message, it reaches its *cut state* and begins supporting $C(\phi)$, begins send inhibition, and forwards the message along the ring.
3. When the initiator gets back $CUT(\phi)$, it stops send inhibition, and forwards a *RESUME* message along the ring.
4. When a process receives the *RESUME* message, it stops send-inhibition, and forwards the *RESUME* message along the ring. The protocol terminates when the initiator gets back the *RESUME* it initiated.

- attains $C^C(\phi)$ when it reaches its *cut state*. FIFO.
- Complexity: $2n$ messages; time complexity: $O(2n)$ hops; send-inhibitory

Knowledge Transfer (1)

Message chain and Process chain

A *message chain* in an execution is a sequence of messages $\langle m_{i_k}, m_{i_{k-1}}, m_{i_{k-2}}, \ldots, m_{i_1} \rangle$ such that for all $0 < j \leq k$, m_{i_j} is sent by process i_j to process i_{j-1} and $receive(m_{i_j}) \prec send(m_{i_{j-1}})$. The message chain identifies *process chain* $\langle i_0, i_1, \ldots, i_{k-2}, i_{k-1}, i_k \rangle$.

- If ϕ is false and later P_1 knows that P_2 knows that $\ldots$ P_k knows ϕ, then there must exist a process chain $\langle i_1, i_2, \ldots i_k \rangle$.
- Indistinguishability of cuts $(a, c) \sim_i (a', c')$ is expressible in the `interleaving model` using isomorphism of executions. Let:
  - x, y, z denote executions or execution prefixes in interleaving model.
  - x_p: projection of execution x on process p.

Isomorphism of executions

1. For x and y, relation $x[p]y$ is true iff $x_p = y_p$.
2. For x and y and a process group G, relation $x[G]y$ is true iff, for all $p \in G$, $x_p = y_p$.
3. Let G_i be process group i and let $k > 1$. Then, $x[G_0, G_1, \ldots, G_k]z$ if and only if $x[G_0, G_1, \ldots, G_{k-1}]y$ and $y[G_k]z$.

Exercise: Examine isomorphism (items 1,2,3 each) using Kripke structures!

Knowledge Transfer (1)

Message chain and Process chain

A *message chain* in an execution is a sequence of messages $\langle m_{i_k}, m_{i_{k-1}}, m_{i_{k-2}}, \ldots, m_{i_1} \rangle$ such that for all $0 < j \leq k$, m_{i_j} is sent by process i_j to process i_{j-1} and $receive(m_{i_j}) \prec send(m_{i_{j-1}})$. The message chain identifies *process chain* $\langle i_0, i_1, \ldots, i_{k-2}, i_{k-1}, i_k \rangle$.

- If ϕ is false and later P_1 knows that P_2 knows that ... P_k knows ϕ, then there must exist a process chain $\langle i_1, i_2, \ldots i_k \rangle$.
- Indistinguishability of cuts $(a, c) \sim_i (a', c')$ is expressible in the `interleaving model` using isomorphism of executions. Let:
  - x, y, z denote executions or execution prefixes in interleaving model.
  - x_p: projection of execution x on process p.

Isomorphism of executions

1. For x and y, relation $x[p]y$ is true iff $x_p = y_p$.
2. For x and y and a process group G, relation $x[G]y$ is true iff, for all $p \in G$, $x_p = y_p$.
3. Let G_i be process group i and let $k > 1$. Then, $x[G_0, G_1, \ldots, G_k]z$ if and only if $x[G_0, G_1, \ldots, G_{k-1}]y$ and $y[G_k]z$.

Exercise: Examine isomorphism (items 1,2,3 each) using Kripke structures!

Knowledge Transfer (1)

Message chain and Process chain

A *message chain* in an execution is a sequence of messages $\langle m_{i_k}, m_{i_{k-1}}, m_{i_{k-2}}, \ldots, m_{i_1} \rangle$ such that for all $0 < j \leq k$, m_{i_j} is sent by process i_j to process i_{j-1} and $receive(m_{i_j}) \prec send(m_{i_{j-1}})$. The message chain identifies *process chain* $\langle i_0, i_1, \ldots, i_{k-2}, i_{k-1}, i_k \rangle$.

- If ϕ is false and later P_1 knows that P_2 knows that $\ldots$ P_k knows ϕ, then there must exist a process chain $\langle i_1, i_2, \ldots i_k \rangle$.
- Indistinguishability of cuts $(a, c) \sim_i (a', c')$ is expressible in the `interleaving model` using isomorphism of executions. Let:
  - x, y, z denote executions or execution prefixes in interleaving model.
  - x_p: projection of execution x on process p.

Isomorphism of executions

1. For x and y, relation $x[p]y$ is true iff $x_p = y_p$.
2. For x and y and a process group G, relation $x[G]y$ is true iff, for all $p \in G$, $x_p = y_p$.
3. Let G_i be process group i and let $k > 1$. Then, $x[G_0, G_1, \ldots, G_k]z$ if and only if $x[G_0, G_1, \ldots, G_{k-1}]y$ and $y[G_k]z$.

Exercise: Examine isomorphism (items 1,2,3 each) using Kripke structures!

Knowledge Transfer (2)

Knowledge operator in the interleaving model

p knows ϕ at execution x if and only if, for all executions y such that $x[p]y$, ϕ is true at y.

When a message is received, set of isomorphic executions can only decrease.

Knowledge transfer theorem

For process groups $G_1, \ldots, G_k$, and executions x and y,
$(K_{G_1} K_{G_2} \ldots K_{G_k}(\phi)$ at x and $x[G_1, \ldots G_k]y) \Longrightarrow K_{G_k}(\phi)$ at y.

Proof by induction.

- Trivial for $k = 1$.
- $k, k > 1$: We infer $\exists$ some $z \mid x[G_1, \ldots G_{k-1}]z$ and $z[G_k]y$.
  From $K_{G_1} K_{G_2} \ldots K_{G_{k-1}}[K_{G_k}(\phi)]$ at x, and from the induction hypothesis:
  infer that $K_{G_{k-1}}[K_{G_k}(\phi)]$ at z.
  Hence, $K_{G_k}(\phi)$ at z. As $z[G_k]y$, $K_{G_k}(\phi)$ at y.

I.t.o. Kripke structures, there is a path from state node $x = s_0$ to state node $y = s_k$, via state nodes $s_1, s_2, \ldots, s_{k-1}$, such that the k edges (s_i, s_{i+1}), $0 \leq i \leq k-1$ are labeled by G_{i+1}.

Knowledge Transfer (2)

Knowledge operator in the interleaving model

p knows ϕ at execution x if and only if, for all executions y such that $x[p]y$, ϕ is true at y.

When a message is received, set of isomorphic executions can only decrease.

Knowledge transfer theorem

For process groups $G_1, \ldots, G_k$, and executions x and y,
$(K_{G_1} K_{G_2} \ldots K_{G_k}(\phi)$ at x and $x[G_1, \ldots G_k]y) \Longrightarrow K_{G_k}(\phi)$ at y.

Proof by induction.

- Trivial for $k = 1$.
- $k, k > 1$: We infer $\exists$ some $z \mid x[G_1, \ldots G_{k-1}]z$ and $z[G_k]y$.
  From $K_{G_1} K_{G_2} \ldots K_{G_{k-1}}[K_{G_k}(\phi)]$ at x, and from the induction hypothesis:
  infer that $K_{G_{k-1}}[K_{G_k}(\phi)]$ at z.
  Hence, $K_{G_k}(\phi)$ at z. As $z[G_k]y$, $K_{G_k}(\phi)$ at y.

I.t.o. Kripke structures, there is a path from state node $x = s_0$ to state node $y = s_k$, via state nodes $s_1, s_2, \ldots, s_{k-1}$, such that the k edges (s_i, s_{i+1}), $0 \leq i \leq k-1$ are labeled by G_{i+1}.

Knowledge Transfer (2)

Knowledge operator in the interleaving model

p knows ϕ at execution x if and only if, for all executions y such that $x[p]y$, ϕ is true at y.

When a message is received, set of isomorphic executions can only decrease.

Knowledge transfer theorem

For process groups $G_1, \ldots, G_k$, and executions x and y,
$(K_{G_1} K_{G_2} \ldots K_{G_k}(\phi)$ at x and $x[G_1, \ldots G_k]y) \Longrightarrow K_{G_k}(\phi)$ at y.

Proof by induction.

- Trivial for $k = 1$.
- $k, k > 1$: We infer $\exists$ some $z \mid x[G_1, \ldots G_{k-1}]z$ and $z[G_k]y$.
  From $K_{G_1} K_{G_2} \ldots K_{G_{k-1}}[K_{G_k}(\phi)]$ at x, and from the induction hypothesis:
  infer that $K_{G_{k-1}}[K_{G_k}(\phi)]$ at z.
  Hence, $K_{G_k}(\phi)$ at z. As $z[G_k]y$, $K_{G_k}(\phi)$ at y.

I.t.o. Kripke structures, there is a path from state node $x = s_0$ to state node $y = s_k$, via state nodes $s_1, s_2, \ldots, s_{k-1}$, such that the k edges (s_i, s_{i+1}), $0 \leq i \leq k-1$ are labeled by G_{i+1}.

Knowledge Transfer (2)

Knowledge operator in the interleaving model

p knows ϕ at execution x if and only if, for all executions y such that $x[p]y$, ϕ is true at y.

When a message is received, set of isomorphic executions can only decrease.

Knowledge transfer theorem

For process groups G_1, ..., G_k, and executions x and y,
$(K_{G_1} K_{G_2} \dots K_{G_k}(\phi)$ at x and $x[G_1, \dots G_k]y) \Longrightarrow K_{G_k}(\phi)$ at y.

Proof by induction.

- Trivial for $k = 1$.
- $k, k > 1$: We infer $\exists$ some $z \mid x[G_1, \dots G_{k-1}]z$ and $z[G_k]y$.
  From $K_{G_1} K_{G_2} \dots K_{G_{k-1}}[K_{G_k}(\phi)]$ at x, and from the induction hypothesis:
  infer that $K_{G_{k-1}}[K_{G_k}(\phi)]$ at z.
  Hence, $K_{G_k}(\phi)$ at z. As $z[G_k]y$, $K_{G_k}(\phi)$ at y.

I.t.o. Kripke structures, there is a path from state node $x = s_0$ to state node $y = s_k$, via state nodes s_1, s_2, ..., s_{k-1}, such that the k edges (s_i, s_{i+1}), $0 \leq i \leq k-1$ are labeled by G_{i+1}.

Knowledge Transfer (2)

Knowledge operator in the interleaving model

p knows ϕ at execution x if and only if, for all executions y such that $x[p]y$, ϕ is true at y.

When a message is received, set of isomorphic executions can only decrease.

Knowledge transfer theorem

For process groups $G_1, \ldots, G_k$, and executions x and y,
$(K_{G_1} K_{G_2} \ldots K_{G_k}(\phi)$ at x and $x[G_1, \ldots G_k]y) \Longrightarrow K_{G_k}(\phi)$ at y.

Proof by induction.

- Trivial for $k = 1$.
- $k, k > 1$: We infer $\exists$ some $z \mid x[G_1, \ldots G_{k-1}]z$ and $z[G_k]y$.
  From $K_{G_1} K_{G_2} \ldots K_{G_{k-1}}[K_{G_k}(\phi)]$ at x, and from the induction hypothesis:
  infer that $K_{G_{k-1}}[K_{G_k}(\phi)]$ at z.
  Hence, $K_{G_k}(\phi)$ at z. As $z[G_k]y$, $K_{G_k}(\phi)$ at y.

I.t.o. Kripke structures, there is a path from state node $x = s_0$ to state node $y = s_k$, via state nodes $s_1, s_2, \ldots, s_{k-1}$, such that the k edges (s_i, s_{i+1}), $0 \leq i \leq k-1$ are labeled by G_{i+1}.

Knowledge Transfer (3)

Knowledge gain theorem

For processes $P_1, \ldots, P_k$, and executions x and y, where x is a prefix of y, let

- $\neg K_k(\phi)$ at x and $K_1 K_2 \ldots K_k(\phi)$ at y.

Then there is a process chain $\langle i_1, \ldots i_{k-1}, i_k \rangle$ in (x, y).

This formalizes that there must exist a message chain $\langle m_{i_k}, m_{i_{k-1}}, m_{i_{k-2}}, \ldots, m_{i_1} \rangle$ in order that a fact ϕ that becomes known to P_k after execution prefix x of y, leads to the state of knowledge $K_1 K_2 \ldots K_k(\phi)$ after execution y.

Knowledge and Clocks

- Assumption: Facts are timestamped by the time of their becoming true and by PID at which they became true.
- *Full-information protocol* (FIP): protocol in which a process piggybacks all its knowledge on outgoing messages, & a process adds to its knowledge all the knowledge that is piggybacked on any message it receives.
- Knowledge always increases when a message is received.
- The amount of knowledge keeps increasing ⇒ impractical
- Facts can always be appropriately encoded as integers.
- *Monotonic facts:* Facts about a property that keep increasing monotonically (e.g., the latest time of taking a checkpoint at a process).
- By using a mapping between logical clocks and monotonic facts, information about the monotonic facts can be communicated between processes using piggybacked timestamps.
- Being monotonic, all earlier facts can be inferred from the fixed amount of information that is maintained and piggybacked.
- E.g., $Clk_i[j]$ indicates the local time at each P_j, and implicitly that all lower clock values at P_j have occurred.
- With appropriate encoding, facts about a monotonic property can be represented using vector clocks.

Knowledge and Clocks

- Assumption: Facts are timestamped by the time of their becoming true and by PID at which they became true.
- *Full-information protocol* (FIP): protocol in which a process piggybacks all its knowledge on outgoing messages, & a process adds to its knowledge all the knowledge that is piggybacked on any message it receives.
- Knowledge always increases when a message is received.
- The amount of knowledge keeps increasing $\Rightarrow$ impractical
- Facts can always be appropriately encoded as integers.
- *Monotonic facts:* Facts about a property that keep increasing monotonically (e.g., the latest time of taking a checkpoint at a process).
- By using a mapping between logical clocks and monotonic facts, information about the monotonic facts can be communicated between processes using piggybacked timestamps.
- Being monotonic, all earlier facts can be inferred from the fixed amount of information that is maintained and piggybacked.
- E.g., $Clk_i[j]$ indicates the local time at each P_j, and implicitly that all lower clock values at P_j have occurred.
- With appropriate encoding, facts about a monotonic property can be represented using vector clocks.

Knowledge, Scalar Clocks, and Matrix Clocks (2)

- Vector clock: $Clk_i[j]$ represents $K_iK_j(\phi_j)$, where ϕ_j is the local component of P_j's clock.
- Matrix clock: $Clk_i[j,k]$ represents $K_iK_jK_k(\phi_k)$, where ϕ_k is the local component $Clk_k[k,k]$ of P_k's clock.
- The j^{th} row of MC $Clk_i[j,\cdot]$: the latest VC value of P_j's clock, as known to P_i.
- The j^{th} column of MC $Clk_i[\cdot,j]$: the latest scalar clock values of P_j, i.e., $Clk[j,j]$, as known to each process in the system.
- Vector and matrix clocks: knowledge is imparted via the *inhibition-free ambient message-passing* that (i) *eliminates protocol messages* by using piggybacking, and (ii) *diffuses* the latest *knowledge* using only messages, whenever sent, by the underlying execution.
- VC provides knowledge $E^0(\phi)$, where ϕ is a property of the global state, namely, the local scalar clock value of each process.
- MC at P_j provides knowledge $K_j(E^1(\phi)) = K_j(\wedge_{i \in N} K_i(\phi))$, where ϕ is the same property of the global state.
- Matrix clocks: used to design distributed database protocols, fault-tolerant protocols, and protocols to discard obsolete information in distributed databases. Also to solve the distributed dictionary and distributed log problems.

Knowledge, Scalar Clocks, and Matrix Clocks (2)

- Vector clock: $Clk_i[j]$ represents $K_i K_j(\phi_j)$, where ϕ_j is the local component of P_j's clock.
- Matrix clock: $Clk_i[j,k]$ represents $K_i K_j K_k(\phi_k)$, where ϕ_k is the local component $Clk_k[k,k]$ of P_k's clock.
- The j^{th} row of MC $Clk_i[j,\cdot]$: the latest VC value of P_j's clock, as known to P_i.
- The j^{th} column of MC $Clk_i[\cdot,j]$: the latest scalar clock values of P_j, i.e., $Clk[j,j]$, as known to each process in the system.
- Vector and matrix clocks: knowledge is imparted via the *inhibition-free ambient message-passing* that (i) *eliminates protocol messages* by using piggybacking, and (ii) *diffuses* the latest *knowledge* using only messages, whenever sent, by the underlying execution.
- VC provides knowledge $E^0(\phi)$, where ϕ is a property of the global state, namely, the local scalar clock value of each process.
- MC at P_j provides knowledge $K_j(E^1(\phi)) = K_j(\wedge_{i \in N} K_i(\phi))$, where ϕ is the same property of the global state.
- Matrix clocks: used to design distributed database protocols, fault-tolerant protocols, and protocols to discard obsolete information in distributed databases. Also to solve the distributed dictionary and distributed log problems.

Knowledge, Scalar Clocks, and Matrix Clocks (2)

- Vector clock: $Clk_i[j]$ represents $K_i K_j(\phi_j)$, where ϕ_j is the local component of P_j's clock.
- Matrix clock: $Clk_i[j, k]$ represents $K_i K_j K_k(\phi_k)$, where ϕ_k is the local component $Clk_k[k, k]$ of P_k's clock.
- The j^{th} row of MC $Clk_i[j, \cdot]$: the latest VC value of P_j's clock, as known to P_i.
- The j^{th} column of MC $Clk_i[\cdot, j]$: the latest scalar clock values of P_j, i.e., $Clk[j, j]$, as known to each process in the system.
- Vector and matrix clocks: knowledge is imparted via the *inhibition-free ambient message-passing* that (i) *eliminates protocol messages* by using piggybacking, and (ii) *diffuses* the latest *knowledge* using only messages, whenever sent, by the underlying execution.
- VC provides knowledge $E^0(\phi)$, where ϕ is a property of the global state, namely, the local scalar clock value of each process.
- MC at P_j provides knowledge $K_j(E^1(\phi)) = K_j(\wedge_{i \in N} K_i(\phi))$, where ϕ is the same property of the global state.
- Matrix clocks: used to design distributed database protocols, fault-tolerant protocols, and protocols to discard obsolete information in distributed databases. Also to solve the distributed dictionary and distributed log problems.

Matrix Clocks

(local variables)
array of int $Clk_i[1 \ldots n, 1 \ldots n]$

MC0. $Clk_i[j, k]$ is initialized to 0 for all j and k

MC1. Before process i executes an internal event, it does the following.
$Clk_i[i, i] = Clk_i[i, i] + 1$

MC2. Before process i executes a send event, it does the following:
$Clk_i[i, i] = Clk_i[i, i] + 1$
Send message timestamped by Clk_i.

MC3. When process i receives a message with timestamp T from process j, it does the following.
$(k \in N)$ $Clk_i[i, k] = \max(Clk_i[i, k], T[j, k])$;
$(l \in N \setminus \{i\})$ $(k \in N)$, $Clk_i[l, k] = \max(Clk_i[l, k], T[l, k])$;
$Clk_i[i, i] = Clk_i[i, i] + 1$;
deliver the message.

- Message overhead: $O(n^2)$ space and processing time