Point-Set Topology for Impossibility Results in Distributed Computing

Thomas Nowak

ENS

Overview

- Introduction
- Safety vs. Liveness
- First Example: Wait-Free Shared Memory
- Message Omission Model
- Execution Trees
- Three Classical Impossibility Results
- Conclusion

Introduction

- goal: explain some arguments in impossibility results in distributed computing via a topological view on the **execution** space (vs. the configuration space)

- in particular, highlight the role of compactness of execution (sub-)spaces in classical arguments

- only very basic results needed

- surveys arguments by: Alpern/Schneider, Lubitch/Moran, Moses/Rajsbaum

Introduction

- crash faults in asynchronous and synchronous systems
- communication media considered: shared memory, message passing, read-modify-write bits
- example problem: impossibility of **consensus**, i.e.,
  - Agreement: all decision values must be the same
  - Validity: if all processes start with the same initial value, then no other value can be decided
  - Termination: every non-faulty (non-crashed) process must eventually decide

Overview

- Introduction
- Safety vs. Liveness
- First Example: Wait-Free Shared Memory
- Message Omission Model
- Execution Trees
- Three Classical Impossibility Results
- Conclusion

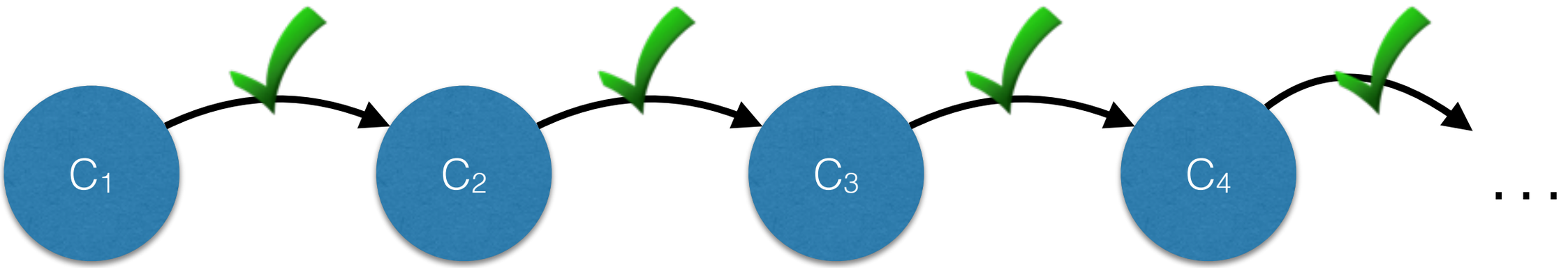
Configurations

- “configuration” = snapshot of the system
- all local states of processes + state of communication medium (maybe + state of adversary etc.)
- should be carefully defined for every concrete distributed computing model (not too much, not too little information)

Transitions and Executions

- “transition” = pair of (successive) configurations
- set of possible transitions = should encode all allowed configuration changes
- “shift focus from the structure of protocols for a distributed system to the structure of the set of possible schedules of a distributed system.” (Saks and Zaharoglou ’00)
- “execution” = sequence of configurations

A Set of Transitions is Not Always Enough



locally OK, globally: 

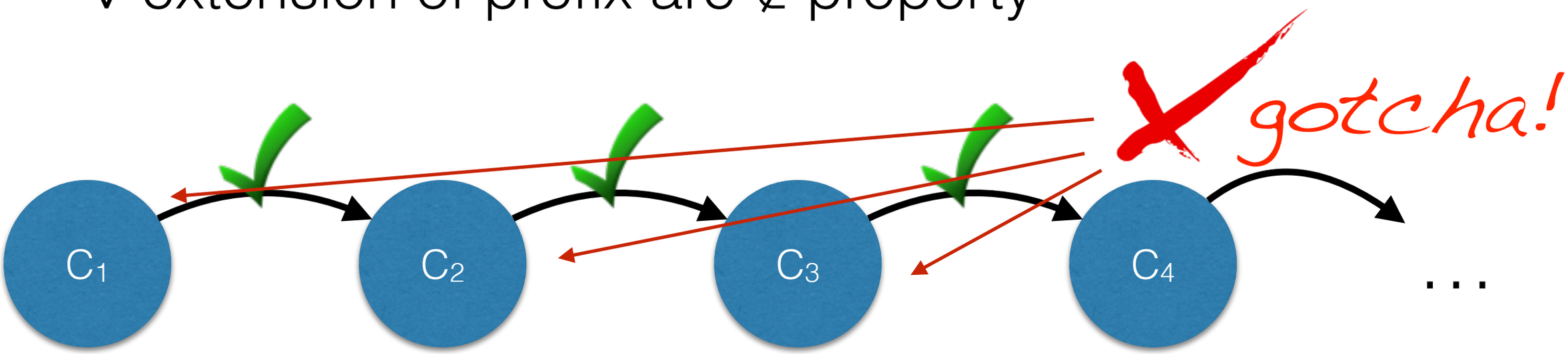
e.g., “Every message that was sent is *eventually* received.”

Safety and Liveness

- Lamport's informal definitions:
- Safety = "something (bad) will not happen"
  - e.g., "Every message is received after at most 3 steps."
- Liveness = "something (good) must happen"
  - e.g., "Every message that was sent is eventually received."

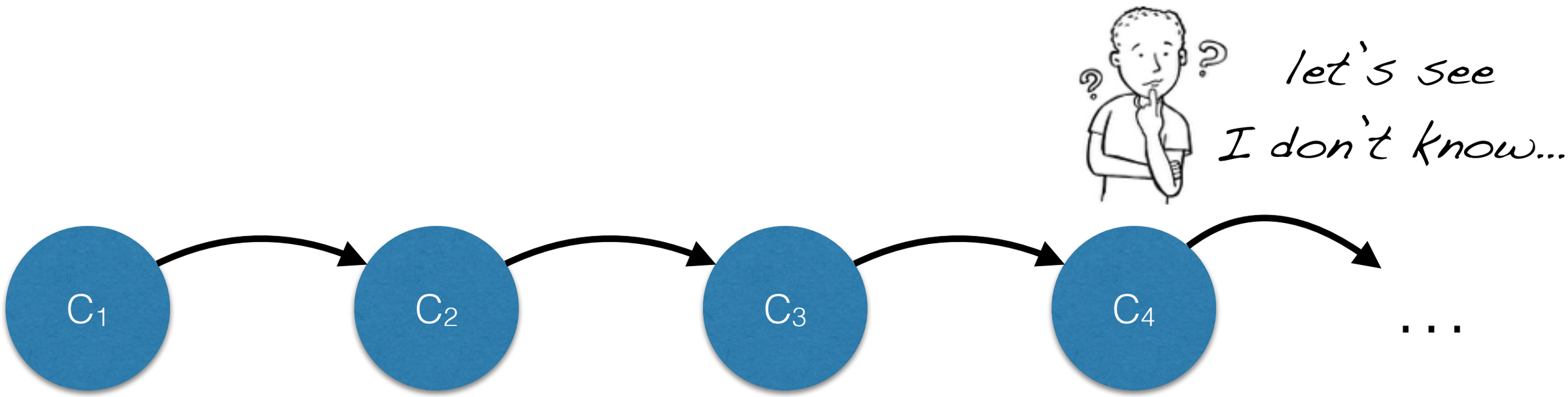
Safety

- “property” = subset of executions
- “safety property” = if it’s violated, some finite prefix is a witness
- i.e., ∀ executions ∉ property ∃ finite prefix ∀ extension of prefix are ∉ property



Liveness

- “liveness property” = you can never tell its violation by a finite prefix
- i.e., ∀ finite prefix ∃ extension ∈ property



Topology and Executions

- on set of configurations = discrete topology
- on set of executions (sequences of configurations) = product topology
- is compact if number of configurations is finite (Tychonoff)
- “model” = subset of executions

Topology and Executions

- Alpern and Schneider (1985) observed:
- safety = closed
- liveness = dense
- proof: $d\big((C_k),(C'_k)\big) = 2^{-\inf\{j|C_j \neq C'_j\}}$

Topology and Executions

- a consequence (proof's a bit harder without topology):

- Thm: Every property is intersection of a safety and a liveness property.

- Proof: Let P be a property. Set $S = cl(P)$ and $L = S^c \cup P$. By definition, $P = S \cap L$. Also, S is closed by definition. Finally $cl(L) = cl(S^c) \cup cl(P) \supseteq S^c \cup S = C^\omega$, i.e., L is dense.

Overview

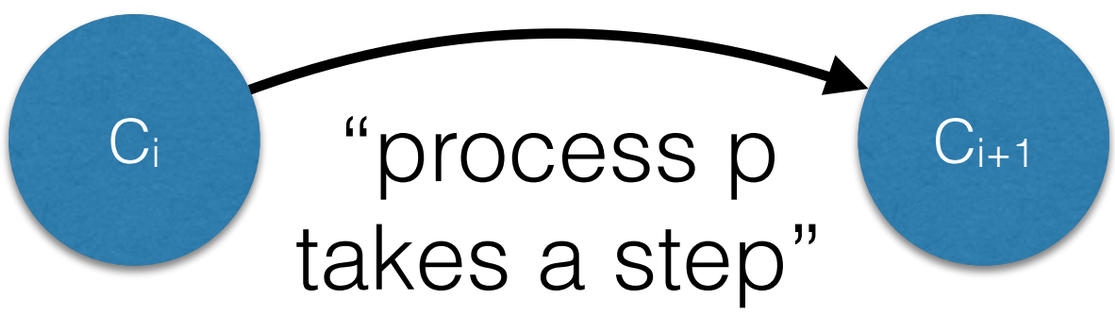
- Introduction
- Safety vs. Liveness
- First Example: Wait-Free Shared Memory
- Message Omission Model
- Execution Trees
- Three Classical Impossibility Results
- Conclusion

First Example

- now: wait-free (i.e., t=n-1) w/ SRMW registers
- show: consensus is impossible (by contradiction, i.e., assume that some algorithm solves it)
- configuration: tuple $(s_1,\ldots,s_n)$ of local states and tuple $(v_1,\ldots,v_m)$ of shared register states
- set of transitions = those allowed by the algorithm when a single process takes a step

First Example

- Q: is the set of transitions finite?
- A: could be, but we can even do without
- → use a **scheduler**, i.e., don't use transitions (executions) but events (schedules)



First Example

- in our example (no message deliveries etc.): event = process number

- wait-free = all but one processes could crash = no liveness condition

- set of admissible schedules = $\{1,\ldots,n\}^{\omega}$

- schedules $\xrightarrow{f}$ executions $\xrightarrow{\Delta}$ decisions $\{0,1\}$

First Example

- if maps *f* and *Δ* are continuous…
- the decision space {0,1} is disconnected (with discrete topology)
- the schedule space $\{1,\ldots,n\}^{\omega}$ is **completely** disconnected (balls are clopen)

First Example

- if maps *f* and *Δ* are continuous, then the inverse images of both {0} and {1} are clopen (thus compact because $\{1,\ldots,n\}^{\omega}$ is)

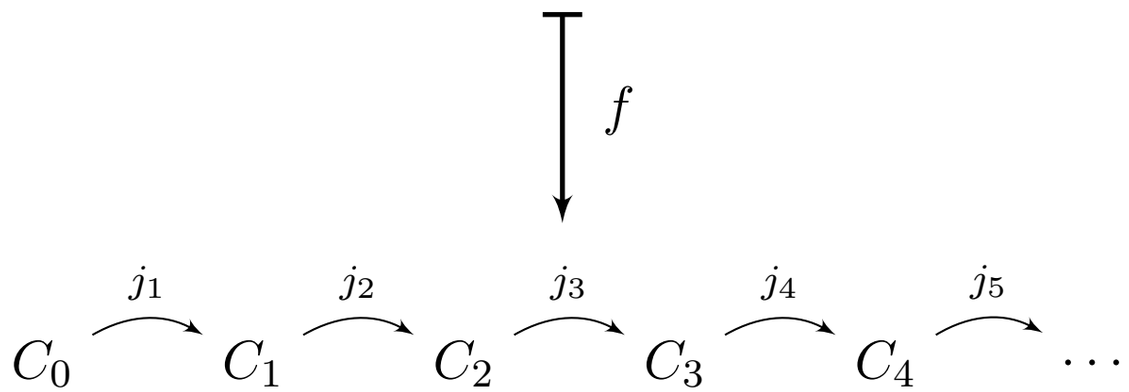
- contradiction after applying some specifics of computational model

- Lem: *Δ* is continuous

- Proof: *Δ* is locally constant (decision doesn't change once taken)

First Example

initial config. C_0; schedule $(j_1, j_2, j_3, j_4, \dots)$



- $f : \{1,\dots,n\}^\omega \rightarrow \boldsymbol{E}$

- Lem: f is continuous

- Proof: Let $\varepsilon = 2^{-n} > 0$, set $\delta = 2^{-n}$. If $d(\sigma_1,\sigma_2) < \delta$, then the first n events are fixed. By definition of f, also the first n configurations are fixed, i.e., $d(f(\sigma_1),f(\sigma_2)) < \varepsilon$.

First Example

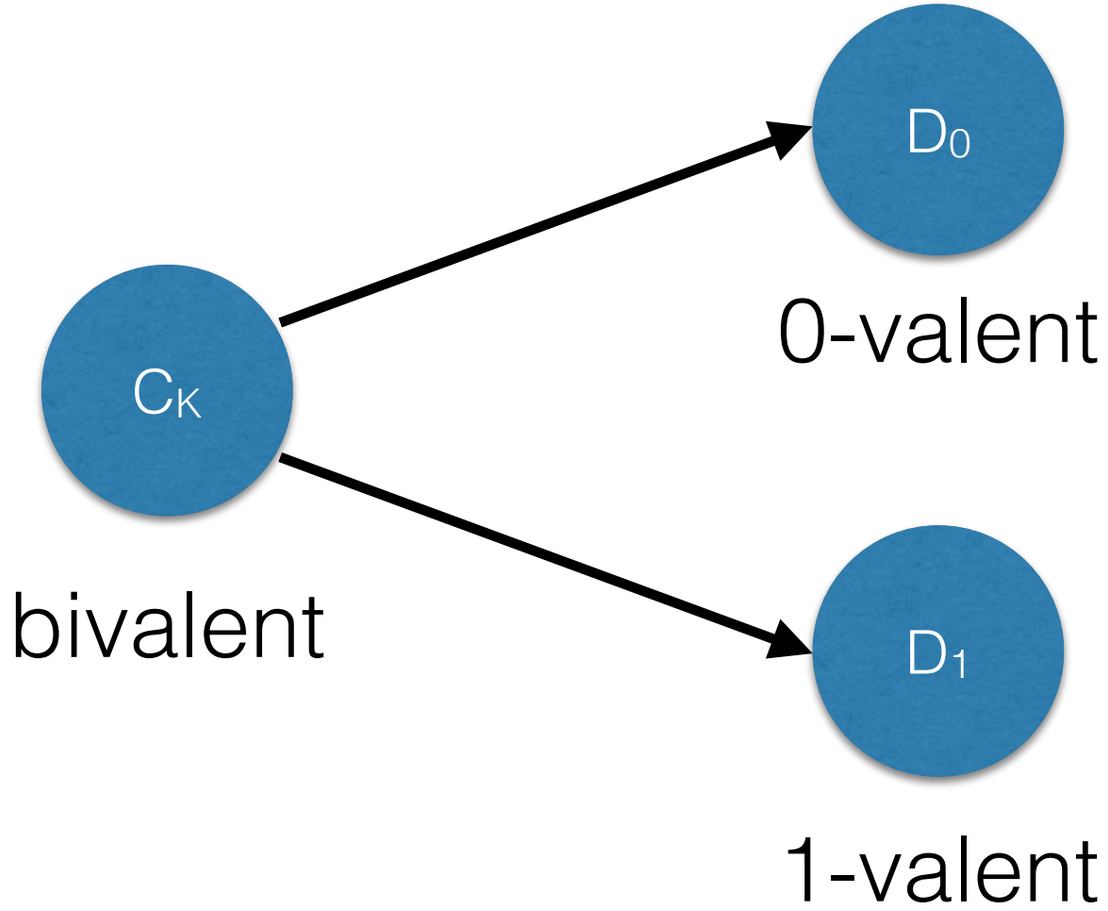
- inverse images $\Sigma_0 = \{\sigma \mid \Delta(f(\sigma)) = 0\}$ and $\Sigma_1 = \{\sigma \mid \Delta(f(\sigma)) = 1\}$ are compact

- Lem: In a metric space, there is a minimal distance between a closed and a compact set.

- if $\delta = 2^{-K}$ is a lower bound on the distance between Σ_0 and Σ_1, then every execution is univalent after at most K steps

First Example

- Lem: There are bivalent initial configurations.
- going from bivalent to univalent after at most K steps implies existence of a fork:



First Example

- Let p be the active process in the transition (C_K,D_0) and q that in the transition (C_K,D_1).

- Case 1: both p and q do read operations
  Pick a third process r and do r,r,r,… ad infinitum. Since p and q only change their local state, their operations cannot influence r, so r should decide on both 0 and 1; contradiction.

- Case 2: p reads, q writes
  Choose process r other than p and the written register's reader.

- Case 3: both write
  Pick r different from the readers of both registers.

First Example

- Proof plan:

  1. pick a compact space of schedules

  2. show continuity of *f*

  3. show that there is a bivalent initial configuration

  4. get existence of a fork

  5. show that fork is impossible by arguments specific to the semantics of the computational model (indistinguishability)

Overview

- Introduction
- Safety vs. Liveness
- First Example: Wait-Free Shared Memory
- Message Omission Model
- Execution Trees
- Three Classical Impossibility Results
- Conclusion

Message Omission Model

- **synchronous** message passing
- but in every round up to n-1 messages may be lost
  - define schedules not only by process numbers, but by set of messages lost
  - it suffices to consider the events of the form omits(i,k), i.e., process i omits to send its messages to processes 1,…,k

Message Omission Model

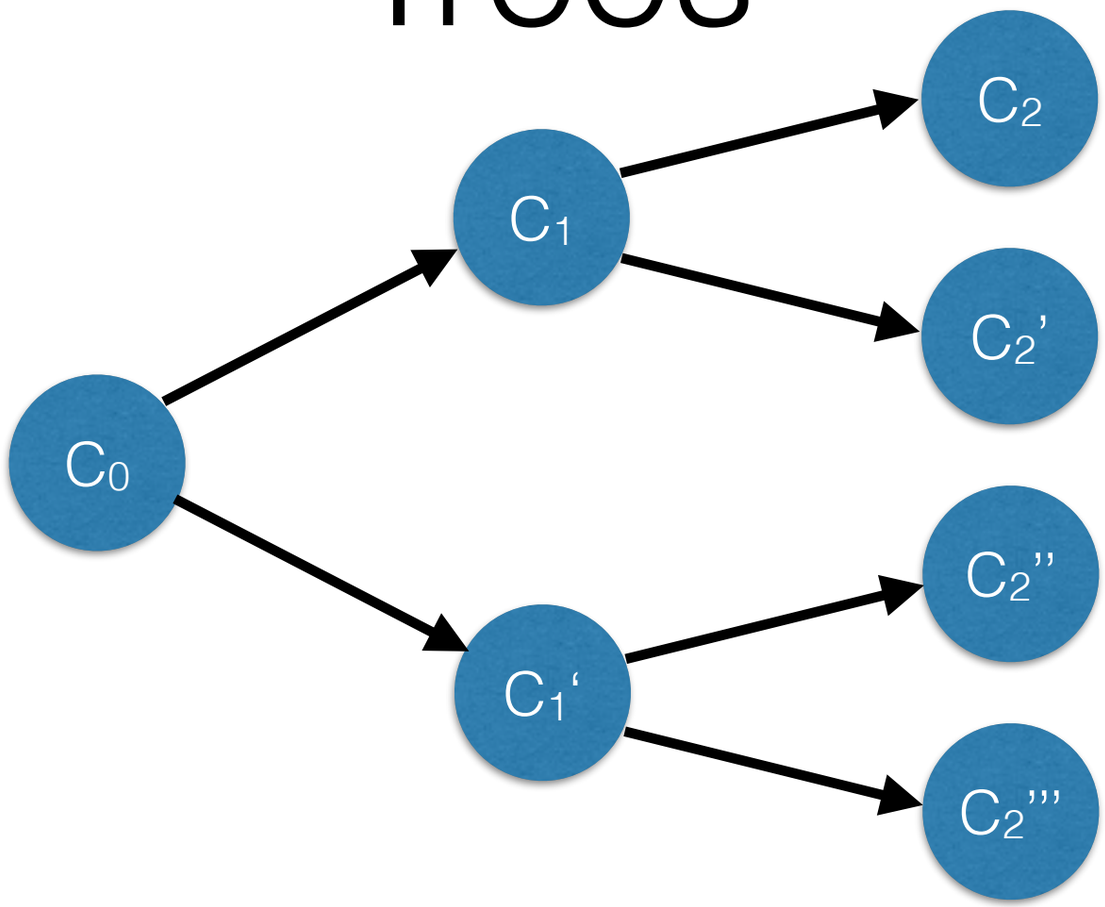
- Thm: Consensus in the message omission model with n-1 omissions per round is impossible, even if the omissions all occur on the same process in every round.

- Proof: Set of schedules is compact. Function *f* is continuous since 1-to-1 correspondence to transitions. Bivalent initial configuration exists (silence a process). Fork is impossible since we can silence the one process that would know the difference between 0-valent and 1-valent.

Overview

- Introduction
- Safety vs. Liveness
- First Example: Wait-Free Shared Memory
- Message Omission Model
- Execution Trees
- Three Classical Impossibility Results
- Conclusion

Execution (or Schedule) Trees



- Thm: A set of executions (or schedules) is closed in $\mathbf{C}^{\omega}$ if and only if every maximal path in its tree is an execution (schedule) in the set.
  If so, it is compact if and only if its tree is locally finite (cf. König's Lemma).

Execution (or Schedule) Trees

- in our first example: wait-free, i.e., up to t=n-1 crashes
- there, easy to find a tree that guarantees t-fair schedules (i.e., that at least n-t = 1 processes appear infinitely often): just let every node have all children 1, …,n
- likewise, wait-free IIS seems to be convenient to work with
- Q: how to do enforce t-fair schedules for other values of parameter t?

Configuration Similarity

- two configurations *C* and *C'* are ***p*-equivalent**, written $C \sim_p C'$, if the local state of *p* (message passing) and the state of the registers that *p* writes (+ in shared memory) are the same in both

- analogously ***Q*-equivalent** for sets *Q* of processes if *p*-equivalent for all $p \in Q$

- Lem: If we apply a Q-only schedule to two Q-equivalent configurations (and we *can* apply them), then both decision values must be the same.

Lubitch and Moran's Schedule Trees

- Lubitch and Moran (DC'95) defined a family of trees $T_{n,t}$ for schedules with *n* processes at most *t* crashes

- nodes are labeled with processes 1,…,n

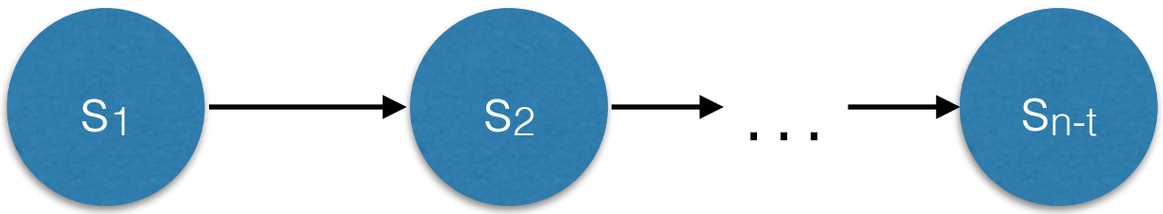
- to determine children of node x, look at the (n-t)-history leading up to x

    1. if not all processes in history are different, then the children of x are those processes not in the history

    2. if all are different, choose the first or second one in the history

Lubitch and Moran's Schedule Trees

(n-t)-history:



- if all s_k different: either s_1 or s_2 or a non-s_k process
- if not: choose a non-s_k process

- Thm: All schedules in the set described by $T_{n,t}$ are (n-t)-fair.

Lubitch and Moran's Schedule Trees

- Lem: From any node in $T_{n,t}$ on, for all sets Q of at least n-t processes, we can extend the schedule such that only processes in Q appear.

- Lem: If i and j are applicable to a node x, then both (i,j) and (j,i) are. Furthermore, for all sets Q of at least n-t processes:

  1. x.(i,j) and x.(j,i) are Q-equivalent

  2. if $i \notin Q$, then x.(i,j) and x.j are Q-equivalent

  3. if $i,j \notin Q$, then x.i and x.j are Q-equivalent

Overview

- Introduction
- Safety vs. Liveness
- First Example: Wait-Free Shared Memory
- Message Omission Model
- Execution Trees
- Three Classical Impossibility Results
- Conclusion

Consequences of Lubitch and Moran's Tree Construction

- two results in shared memory (Loui and Abu-Amara '87):

- Thm: 1-resilient asynchronous consensus with shared memory is impossible.

- Thm: 2-resilient asynchronous consensus with read-modify-write bits is impossible.

- generalization of a result in message passing (Fischer, Lynch, and Paterson '85):

- Thm: 1-resilient asynchronous consensus in message passing with global FIFO on outgoing messages at each process is impossible.

Overview

- Introduction
- Safety vs. Liveness
- First Example: Wait-Free Shared Memory
- Message Omission Model
- Execution Trees
- Three Classical Impossibility Results
- Conclusion

Conclusion

- you can also look at the topology of executions (vs. topology of configurations)

- popular strategy for impossibility results: find a safety-only (closed) submodel (or different model + reduction) in which impossibility also holds

- in closed models: compactness argument on executions + model-dependent indistinguishability

- Q: combine topology on configurations and topology on executions?

Thank You!