A Lattice-Theoretic Approach to Monitoring Distributed Computations

Vijay K. Garg and Neeraj Mittal

Parallel and Distributed Systems Laboratory
Department of Electrical and Computer Engineering
The University of Texas at Austin

Advanced Networking and Dependable Systems Laboratory
Department of Computer Science
The University of Texas at Dallas

September 22, 2014

Tutorial Outline

1. Complexity of General Predicate Detection
   - Detecting 3-CNF Predicates
   - Detecting 2-CNF Predicates
2. Slicing a Distributed Computation
   - Definitions and Techniques
     - What is Computation Slicing?
     - Regular Predicates
     - Slicing for Regular Predicates
     - Slice Composition
   - Other Extensions
     - Online Slicing Algorithm
     - Distributed Slicing Algorithm
   - Equivalence: One for All and All for One
   - Slicing for Temporal Logic Predicates

Detecting a 3-CNF Predicate

3-CNF Predicate:

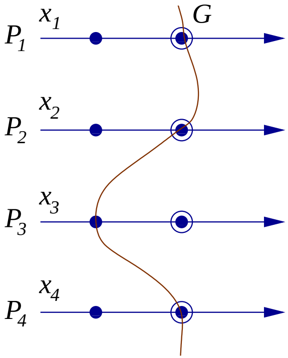
- A conjunction of clauses.
- Each clause is a disjunction of exactly three literals.

Example: $(x_1 \vee \overline{x_2} \vee x_3) \bigwedge (x_2 \vee \overline{x_3} \vee x_4) \bigwedge (\overline{x_1} \vee x_2 \vee \overline{x_4})$

The Transformation: From 3-SAT Problem

- For each variable x_i in the formula, there is a process P_i that hosts x_i in the computation.
- Each variable x_i is initially false and then becomes true.

$$(x_1 \vee x_2 \vee x_3) \\ \bigwedge \\ (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \\ \bigwedge \\ (\overline{x}_1 \vee x_3 \vee x_4)$$



Tutorial Outline

1. Complexity of General Predicate Detection
   - Detecting 3-CNF Predicates
   - Detecting 2-CNF Predicates

2. Slicing a Distributed Computation
   - Definitions and Techniques
     - What is Computation Slicing?
     - Regular Predicates
     - Slicing for Regular Predicates
     - Slice Composition
   - Other Extensions
     - Online Slicing Algorithm
     - Distributed Slicing Algorithm
   - Equivalence: One for All and All for One
   - Slicing for Temporal Logic Predicates

Detecting a 2-CNF Predicate

Singular 2-CNF Predicate: a global predicate in conjunctive normal (CNF) form such that:

- each clause has exactly two literals, and
- no two clauses contain variables from the same process.

[Mittal and Garg, ICDCS 2001]

Examples: Let x_i be a boolean variable on process P_i.

$+$ $(x_1 \vee x_2) \wedge (x_3 \vee x_4)$

$+$ $(x_1 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee x_4)$

$-$ $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$

More restrictive than a 3-CNF predicate.

2-SAT problem can be solved in polynomial time.

Detecting a Singular 2-CNF Predicate

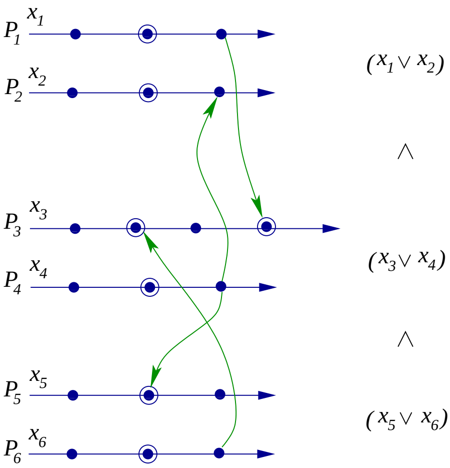
No two clauses in a singular 2-CNF predicate contain variables from the same process.

$$\Longrightarrow$$

The set of processes in the computation can be partitioned into pairwise disjoint groups such that each group consists of processes that host variables in the same clause.

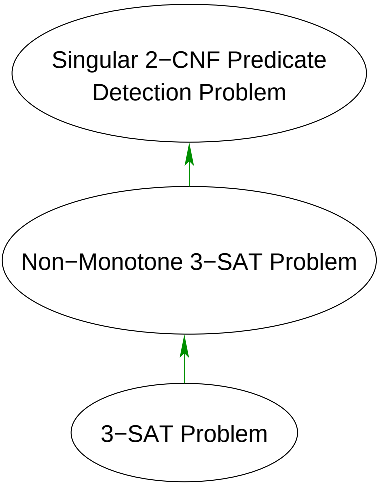
Observation: To find a consistent cut that satisfies a singular 2-CNF predicate, it is necessary and sufficient to find a subset of true events, one from some process in each group, that are mutually consistent.

Detecting Singular 2-CNF Predicates: Example



Here, $G_1 = \{P_1, P_2\}$, $G_2 = \{P_3, P_4\}$ and $G_3 = \{P_5, P_6\}$.

Proof Structure



Non-Monotone 3-CNF Formulae

Non-Monotone 3-CNF Formula: a formula in conjunctive normal form (CNF) such that:

- each clause has at most three literals, and
- each clause with exactly three literals has at least one positive and one negative literal.

Examples:

+ $(\overline{y}_1 \vee y_3) \bigwedge (\overline{y}_2 \vee \overline{y}_4 \vee y_1)$
+ $(y_1 \vee y_2) \bigwedge (\overline{y}_2 \vee y_3 \vee y_1)$
− $(\overline{y}_1 \vee y_2) \bigwedge (y_1 \vee y_3 \vee y_4)$

Non-Monotone 3-SAT Problem

Given a non-monotone 3-CNF formula, does there exist a satisfying truth assignment for the formula?

Complexity: Non-monotone 3-SAT problem is NP-complete in general.

Idea: Reduction from 3-SAT problem. Replace the clause $(y_1 \vee y_2 \vee y_3)$ with clauses $(y_1 \vee y_2 \vee \overline{z}_3)$, $(y_3 \vee z_3)$ and $(\overline{y}_3 \vee \overline{z}_3)$.

Solving Non-Monotone 3-SAT Problem

Observation: To find a satisfying truth assignment for a non-monotone 3-CNF formula, it is necessary and sufficient to find a subset of literals, one from each clause, that are mutually non-conflicting.

For example: $(y_1 \vee y_2 \vee \overline{y}_3) \wedge (\overline{y}_1 \vee \overline{y}_2) \wedge (y_3 \vee \overline{y}_2)$

+ $\{y_1, \overline{y}_2, y_3\}$
+ $\{y_2, \overline{y}_1, y_3\}$
- $\{y_1, \overline{y}_1, y_3\}$

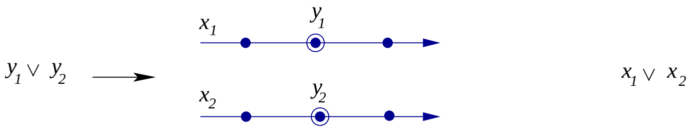
The Transformation

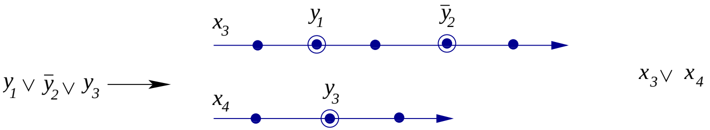
There is a one-to-one correspondence between a clause in the formula and a clause in the predicate.

There is a one-to-one correspondence between a literal in the formula and a true event in the computation.

Two literals conflict if and only if the corresponding true events are inconsistent.

The Transformation: Example





Potential Problems

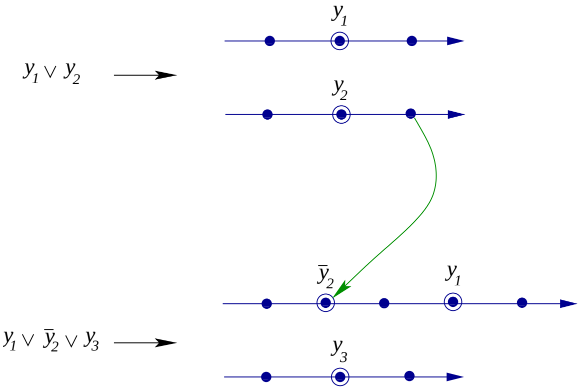
To make true events corresponding to conflicting literals (e.g., y_2 and $\overline{y_2}$) inconsistent:

- add an arrow from the successor of the true event for the positive literal to the true event for the negative literal.

Problems: If arrows are not added properly, then

- Added arrows can create cycles.
- Two true events corresponding to non-conflicting literals can become inconsistent.

Potential Problems: Example

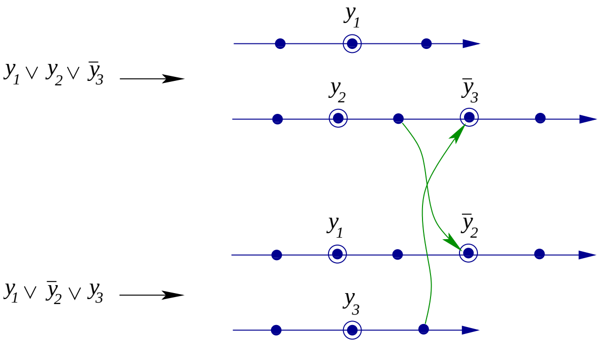


the true event for y_2 in the first clause is now inconsistent with the true event for y_1 in the second clause

The Solution

Whenever literals in a clause are forced to share a process:

- choose one positive literal and one negative literal for sharing, and
- put the true event for the positive literal before the true event for the negative literal.

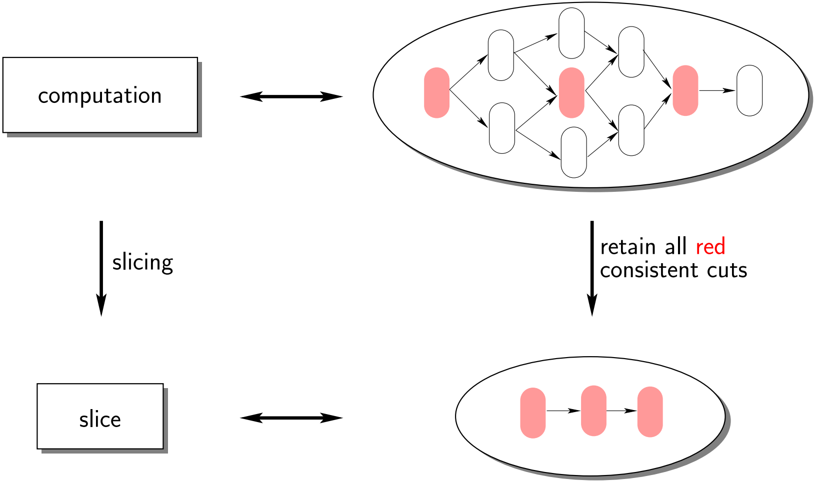


Tutorial Outline

1. Complexity of General Predicate Detection
   - Detecting 3-CNF Predicates
   - Detecting 2-CNF Predicates
2. Slicing a Distributed Computation
   - Definitions and Techniques
     - What is Computation Slicing?
     - Regular Predicates
     - Slicing for Regular Predicates
     - Slice Composition
   - Other Extensions
     - Online Slicing Algorithm
     - Distributed Slicing Algorithm
   - Equivalence: One for All and All for One
   - Slicing for Temporal Logic Predicates

Tutorial Outline

The Main Idea of Computation Slicing

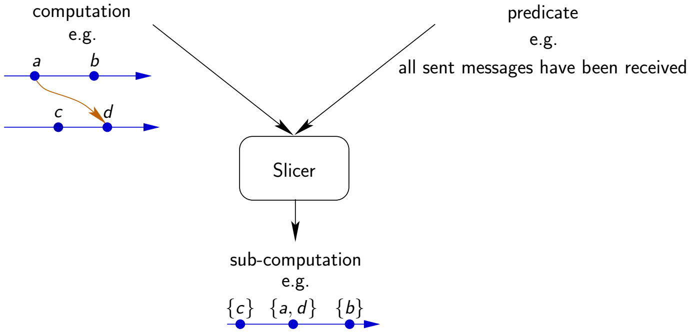


Computation Slice

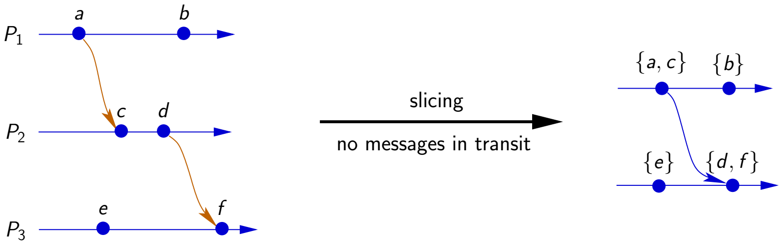
Computation slice: a sub-computation such that:

1. it contains all consistent cuts of the computation satisfying the given predicate, and
2. it contains the least number of consistent cuts

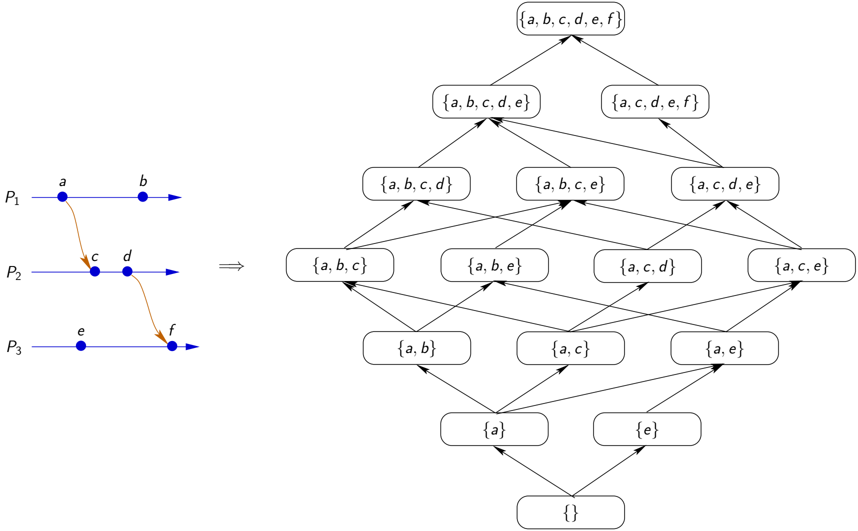
[Mittal and Garg, DC 2005]



Slicing Example



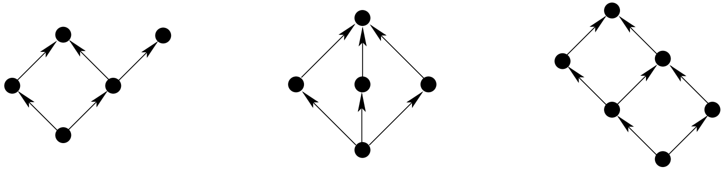
Slicing Example (Contd.)



Characterization of Consistent Cuts

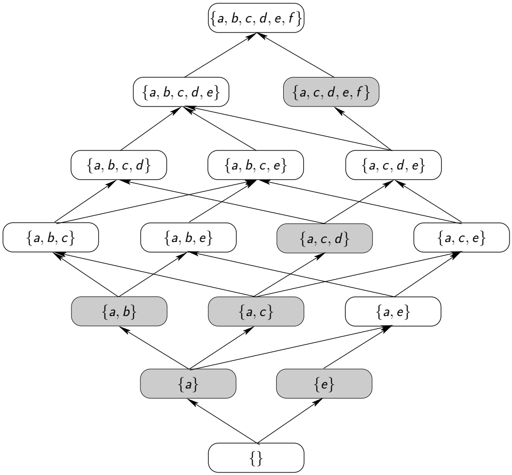
The set of consistent cuts of a distributed computation forms a distributive lattice.

1. meet operator: set intersection
2. join operator: set union
3. meet distributes over join



Basis Elements

Basis element: cannot be represented as join of two other elements

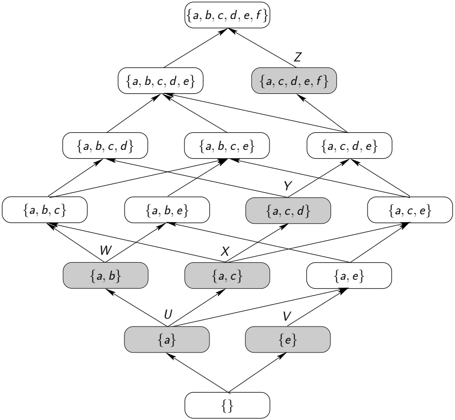


A basis element has exactly one incoming edge.

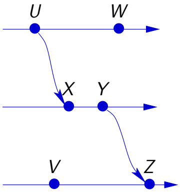
Birkhoff's Representation Theorem

Theorem

A distributive lattice can be **recovered exactly** *from the set of its basis elements.*

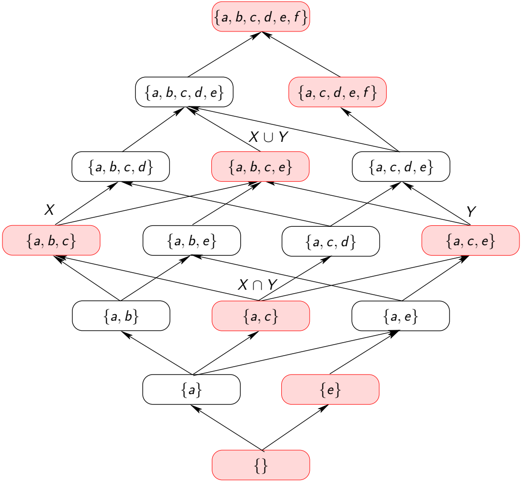


$\Longleftrightarrow$



All elements can be represented as join of some subset of its basis elements.

What about a Subset of Consistent Cuts?



X in subset and
Y in subset

$\Longrightarrow$

$X \cap Y$ in subset and
$X \cup Y$ in subset

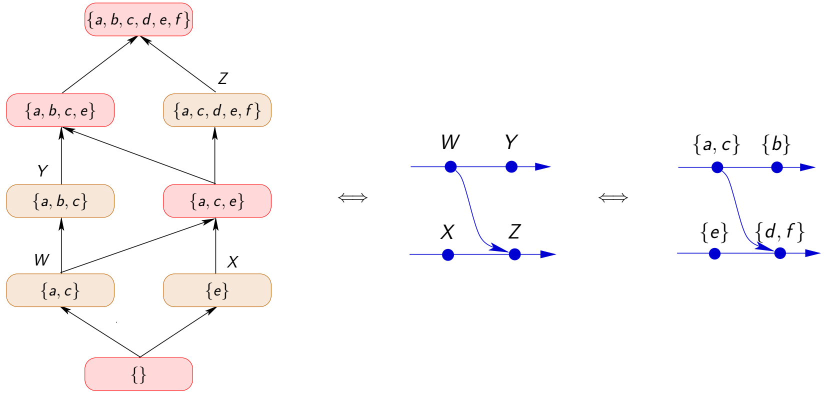
Sublattice: subset of consistent cuts closed under intersection and union.

Representing a Sublattice

Theorem

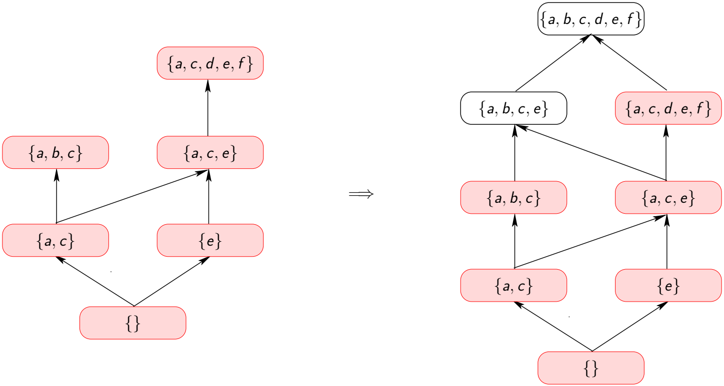
A sublattice of a distributive lattice is also a **distributive lattice**.

A sublattice has a **succinct representation**.



What if the Subset is not a Sublattice?

Add consistent cuts to complete the sublattice.



Computing the Slice

Algorithm:

1. Find all consistent cuts that satisfy the predicate.
2. Add consistent cuts to complete the sublattice.
3. Find the basis elements of the sublattice.

Can we find the basis elements without computing the sublattice?

Tutorial Outline

1. Complexity of General Predicate Detection
   - Detecting 3-CNF Predicates
   - Detecting 2-CNF Predicates

2. Slicing a Distributed Computation
   - Definitions and Techniques
     - What is Computation Slicing?
     - Regular Predicates
     - Slicing for Regular Predicates
     - Slice Composition
   - Other Extensions
     - Online Slicing Algorithm
     - Distributed Slicing Algorithm
   - Equivalence: One for All and All for One
   - Slicing for Temporal Logic Predicates

Regular Predicate

Regular predicate: the set of consistent cuts satisfying the predicate is closed under intersection and union.

$$(X \text{ satisfies } b) \;\; \text{and} \;\; (Y \text{ satisfies } b)$$

$$\Longrightarrow$$

$$(X \cap Y \text{ satisfies } b) \;\; \text{and} \;\; (X \cup Y \text{ satsfies } b)$$

[Mittal and Garg, DC 2005]

Examples:

- conjunctive predicate—conjunction of local predicates
- there are at most (or at least) k messages in transit from process P_i to process P_j
- every "request" message has been "acknowledged" in the system

Properties of Regular Predicates

The set of consistent cuts satisfying a regular predicate forms a sublattice of the set of all consistent cuts.

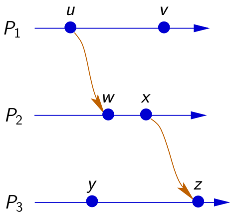
The class of regular predicates is closed under conjunction:

If b_1 and b_2 are regular predicates then so is $b_1 \wedge b_2$

The class of regular predicates is a subset of the class of linear predicates.

Tutorial Outline

Computing the Slice for Regular Predicate



$b =$ "no messages in transit"

Algorithm:

Step 1: Compute the least consistent cut L that satisfies b.
$L = \{\}$

Step 2: Compute the greatest consistent cut G that satisfies b.
$G = \{u, v, w, x, y, z\}$

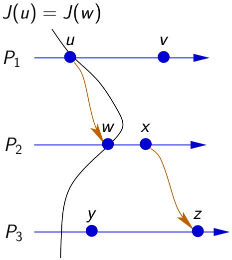
Computing the Slice for Regular Predicate

Algorithm:

Step 3: For every event $e \in G - L$, compute $J(e)$ defined as:
(1) $J(e)$ contains e.
(2) $J(e)$ satisfies b.
(3) $J(e)$ is the least consistent cut satisfying (1) and (2).

- $J(e)$ is a basis element of the sublattice.
- $J(e)$ is a regular predicate.

Slicing Example



$J(u) = \{u, w\}$
$J(v) = \{u, v, w\}$
$J(w) = \{u, w\}$ (duplicate)
$J(x) = \{u, w, x, y, z\}$
$J(y) = \{y\}$
$J(z) = \{u, w, x, y, z\}$ (duplicate)

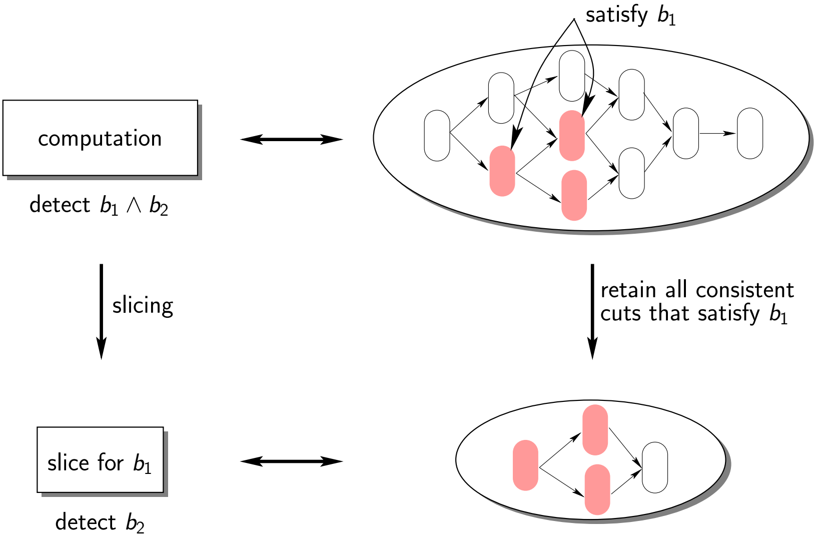
Slicing Algorithms

Efficient slicing algorithms have been developed for many classes of predicates.

Some examples:

- regular predicates, co-regular predicates, linear predicates, stable predicates, observer-independent predicates, etc.

How does Computation Slicing Help?

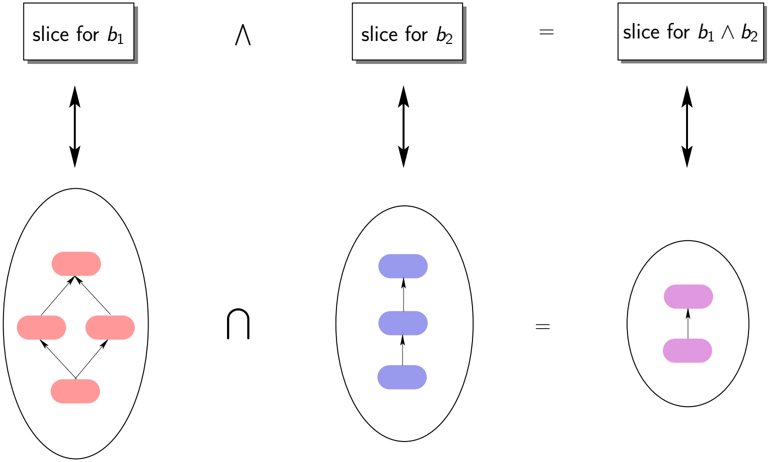


Tutorial Outline

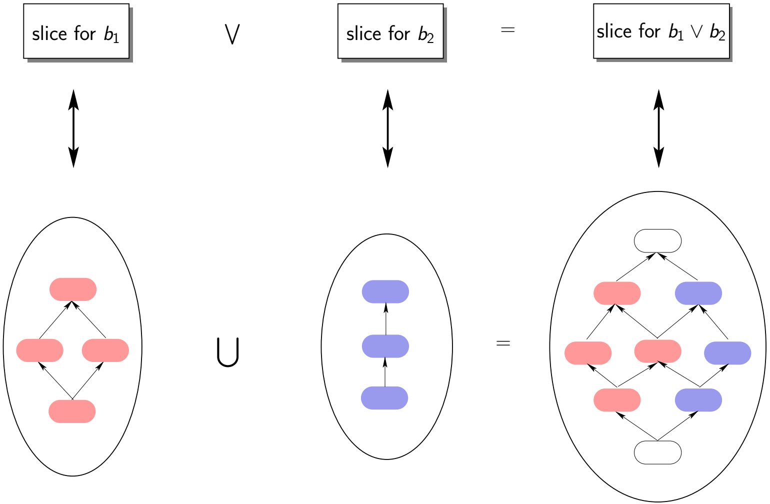
1. Complexity of General Predicate Detection
   - Detecting 3-CNF Predicates
   - Detecting 2-CNF Predicates

2. Slicing a Distributed Computation
   - Definitions and Techniques
     - What is Computation Slicing?
     - Regular Predicates
     - Slicing for Regular Predicates
     - Slice Composition
   - Other Extensions
     - Online Slicing Algorithm
     - Distributed Slicing Algorithm
   - Equivalence: One for All and All for One
   - Slicing for Temporal Logic Predicates

Composing Two Slices: Conjunction



Composing Two Slices: Disjunction



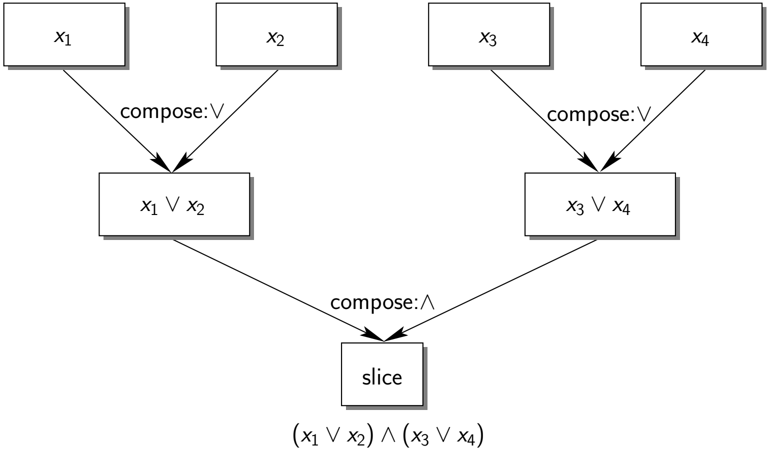
Composition Algorithms

Efficient algorithms for both conjunction and disjunction.

Time complexity: $O(n|E|)$ where
  n: number of processes
$|E|$: number of events [Mittal and Garg, DC 2005]

Computing a Slice using Composition

Example: $(x_1 \vee x_2) \bigwedge (x_3 \vee x_4)$



Tutorial Outline

Tutorial Outline

Why Online Slicing Algorithm?

Compute the slice incrementally as new events are generated.

Slice for the computation is available more quickly.

Fault can be detected in a more timely manner.
[Mittal et al., TPDS 2007]

The Main Idea

Off-line Algorithm:

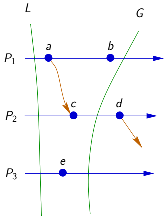
1. Compute L and G—the least and the greatest consistent cuts that satisfy the predicate.
2. Compute $J(x)$ for all events x in $G \setminus L$.

On-Line Algorithm:

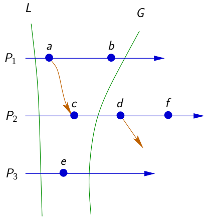
- When a new event arrives, compute the new G.
- Only need to compute $J(x)$ for events $x \in G_{new} \setminus G_{old}$.
- Amortized time-complexity: $O(n^2)$ per event, where n denotes the number of processes.

Online Slicing Algorithm: Example

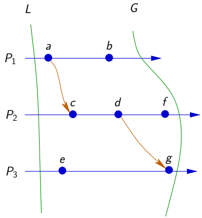
$b =$ "no messages in transit"



Already computed $J(x)$ for $x \in \{a, b, c, e\}$.



No new computation needed.



Only need to compute $J(x)$ for $x \in \{d, f, g\}$.

Tutorial Outline

Why Distributed Algorithm?

Centralized Algorithm: All events sent to a single process.

Distributed Algorithm: Overhead evenly distributed among all processes:

- Less work per process.
- Less storage space per process.

[Chauhan et al., SRDS 2013]

Challenges

Simple decomposition of centralized algorithm into n independent executions is inefficient.

- Results in large number of redundant communications.
- Multiple computations may lead to identical results.

The Main Idea

There is one token per process.

- T_i denotes the token for process P_i.

T_i is responsible for computing $J(e)$ for all events e on P_i.

- T_i calculates $J(e)$s one event at a time.
- T_i may move from one process to another process to compute $J(e)$.

Algorithm for Process P_i

Consider event e on process P_i:

- T_i starts at P_i.
- Keeps tracks of the current cut under consideration:
  - Initialized using J(predecessor of e).
- If the cut either is not consistent or does not satisfy the predicate, then find the process along with the cut needs to be advanced.
- T_i moves to that process.

Optimizations

Optimization I: Copy information from another token whenever two tokens meet.

Optimization II: Stall computations that would lead to duplication of steps.

Tutorial Outline

1. Complexity of General Predicate Detection
   - Detecting 3-CNF Predicates
   - Detecting 2-CNF Predicates

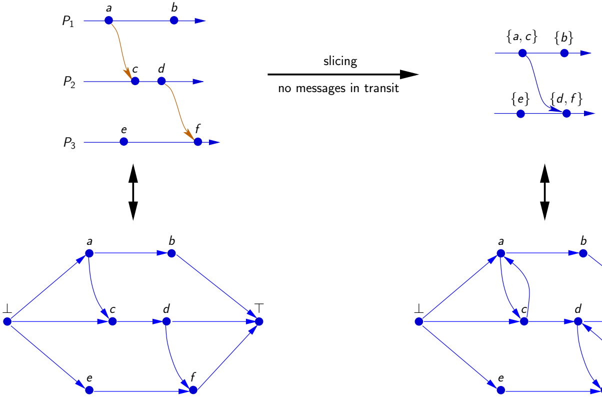
2. Slicing a Distributed Computation
   - Definitions and Techniques
     - What is Computation Slicing?
     - Regular Predicates
     - Slicing for Regular Predicates
     - Slice Composition
   - Other Extensions
     - Online Slicing Algorithm
     - Distributed Slicing Algorithm
   - **Equivalence: One for All and All for One**
   - Slicing for Temporal Logic Predicates

Generalized Model

Model both distributed computation and its slice using a directed graph on events.

- Two special events: the initial event $\perp$ and the final event $\top$.

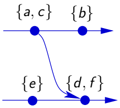
[Mittal and Garg, DC 2005, Mittal et al., TPDS 2007]

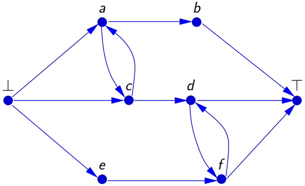


Cycles in a Graph

What is the meaning of a cycle in a graph?

- A consistent of a graph either contains all the events in a cycle or none of them.



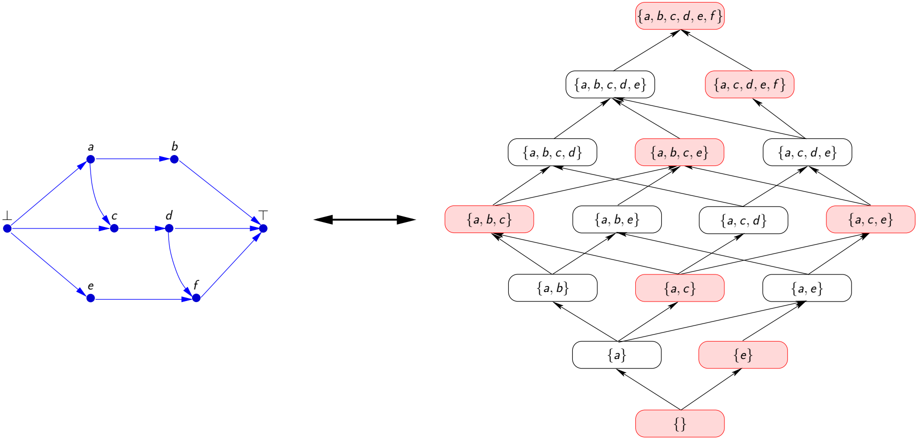


Not Consistent Cuts	Consistent Cuts
$\{\perp, a\}$	$\{\perp\}$
$\{\perp, c\}$	$\{\perp, e\}$
$\{\perp, a, b\}$	$\{\perp, \underline{a}, \underline{c}\}$
	$\{\perp, \underline{a}, b, \underline{c}\}$
	$\{\perp, \underline{a}, b, \underline{c}, e\}$

Edges and Consistent Cuts

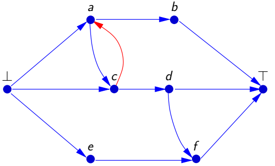
Anti-monotonic relation.

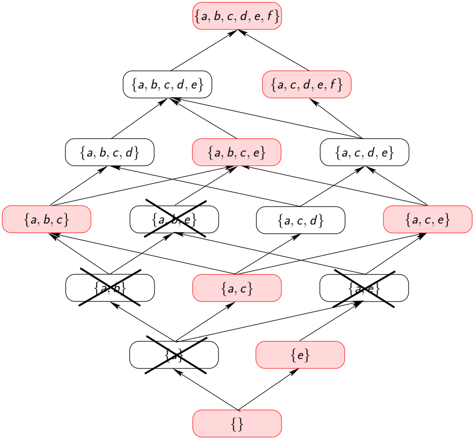
Adding an edge to a graph shrinks its set of consistent cuts.



Edges and Consistent Cuts

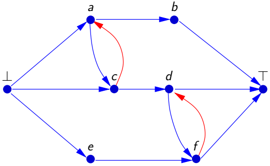
Anti-monotonic relation.
Adding an edge to a graph shrinks its set of consistent cuts.

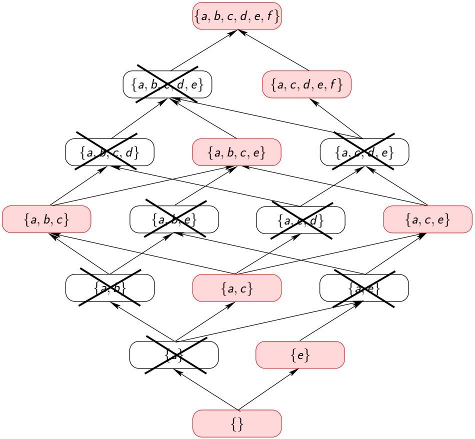




Edges and Consistent Cuts

Anti-monotonic relation.
Adding an edge to a graph shrinks its set of consistent cuts.





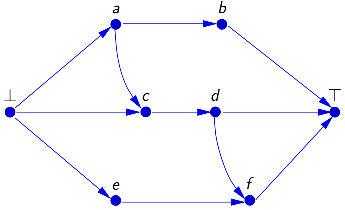
Computing the Slice: Revisited

Find the largest set of edges whose addition to the graph does not eliminate any relevant consistent cut.

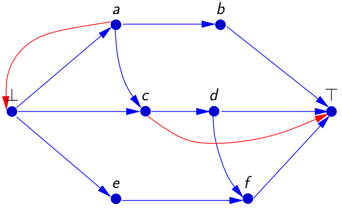
Complement Graph

Given: Graph G and a pair of events (x, y).
Output: Complement graph $G^c[x, y]$ obtained by adding two edges to G:

1. the edge from y to $\perp$, and
2. the edge from x to $\top$.9



Graph G and events (c, a)



Complement graph $G^c[c, a]$

$G^c[c, a]$ only contains those consistent cuts of G that include a but not c.

General Slicing Algorithm

Data: (1) graph G, (2) predicate b, and (3) algorithm Detect(b)
Result: Slice(b)

```
K := G;
foreach pair of events (x, y) in G do
    Compute G^c[x, y];
    if Detect(b) is false in G^c[x, y] then
        Add edge (x, y) in K;
    end if
end foreach
return K;
```

Time complexity: $O(n|E|T)$

n: number of processes $|E|$: number of events

T: time complexity of detection algorithm

Tutorial Outline

1. Complexity of General Predicate Detection
   - Detecting 3-CNF Predicates
   - Detecting 2-CNF Predicates
2. Slicing a Distributed Computation
   - Definitions and Techniques
     - What is Computation Slicing?
     - Regular Predicates
     - Slicing for Regular Predicates
     - Slice Composition
   - Other Extensions
     - Online Slicing Algorithm
     - Distributed Slicing Algorithm
   - Equivalence: One for All and All for One
   - Slicing for Temporal Logic Predicates

Path Based Properties

Some system properties are defined on paths rather than states.

Examples:

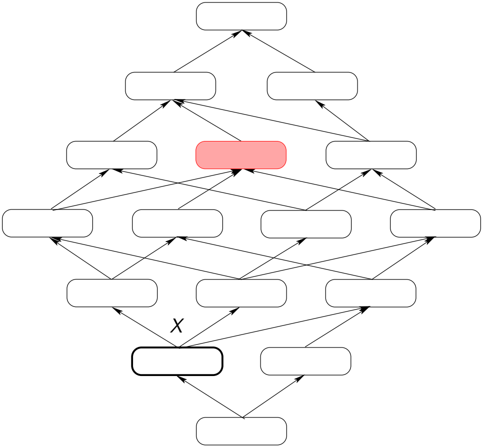
- Starvation freedom: Every request is eventually fulfilled.
- Deadlock freedom: If there is one or more request in the system, then some request is eventually fulfilled.

Temporal Logic Predicates

A path is a sequence of consistent cuts ending at the final consistent cut such that a successor of a cut is obtained by addition of a single vertex.

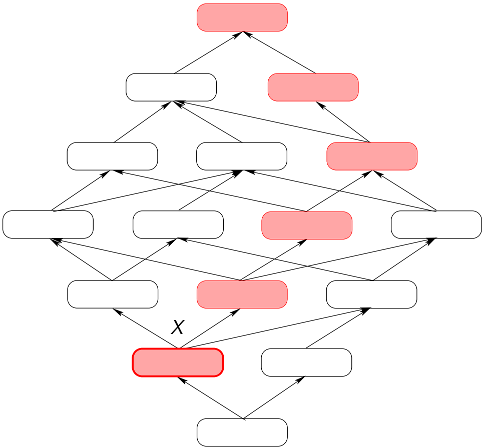
Temporal Operators: EF, EG, AG, EF and EX[j].

Temporal Operator: EF



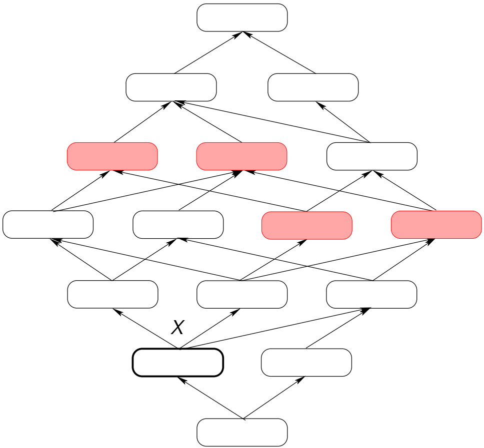
X satisfies EF(b).

Temporal Operator: EG



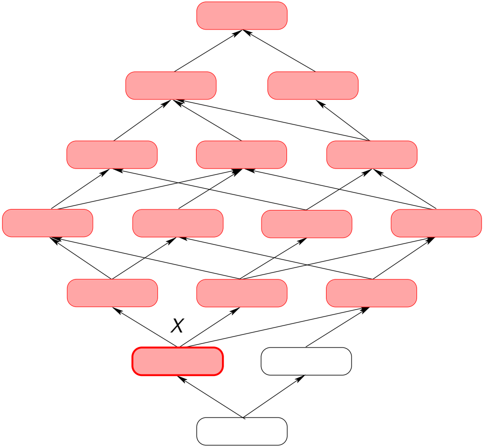
X satisfies EG(b).

Temporal Operator: EF



X satisfies AF(b).

Temporal Operator: AG



X satisfies AG(b).

RCTL Predicates

Examples:

- Violation of mutual exclusion: Processes P_1 and P_2 are not in their critical sections simultaneously.
$$\text{EF}(CS_1 \wedge CS_2)$$
- Starvation Freedom: Every request is eventually fulfilled.
$$\text{AG}(request \implies \text{AF}(granted))$$

RCTL: subset of CTL (Computation Tree Logic) where atomic propositions are regular and the operators are EF, EG, AG, EX[j] and $\wedge$. [Sen and Garg, TC 2007]

An RCTL predicate is a regular predicate.

Computing the Slice for EG(b)

Observation 1: Any consistent cut that satisfies EG(b) also satisfies b.

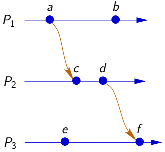
Idea 1: To compute Slice(EG(b)), first compute Slice(b) and then add edges to it.

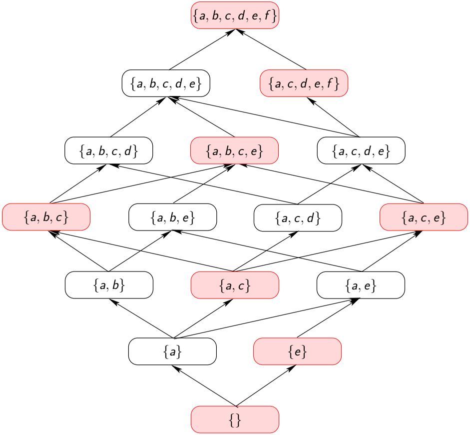
Observation 2:

- Let $\mathcal{C}$ denote a cycle in Slice(b).
- If a consistent cut X satisfies EG(b), then X includes all the vertices from $\mathcal{C}$.

Idea 2: Eliminate all consistent cuts that do not contain all vertices of a cycle.

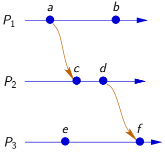
Computing the Slice for EG(b)

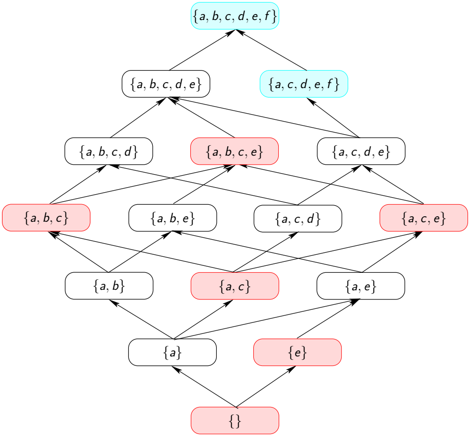




$b =$ no messages in transit

Computing the Slice for EG(b)

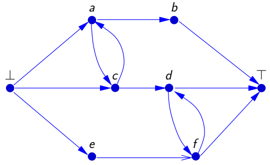


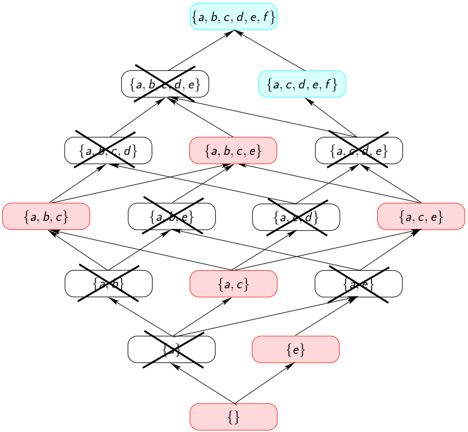


$b =$ no messages in transit

Consistent cuts that satisfy EG(b): $\{a, c, d, e, f\}$ and $\{a, b, c, d, e, f\}$.

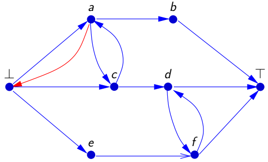
Computing the Slice for EG(b)

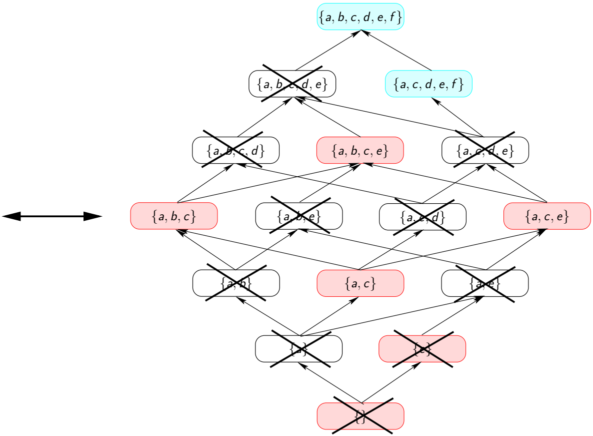




All consistent cuts that do not satisfy b have been eliminated.

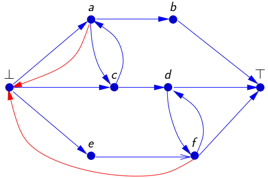
Computing the Slice for EG(b)

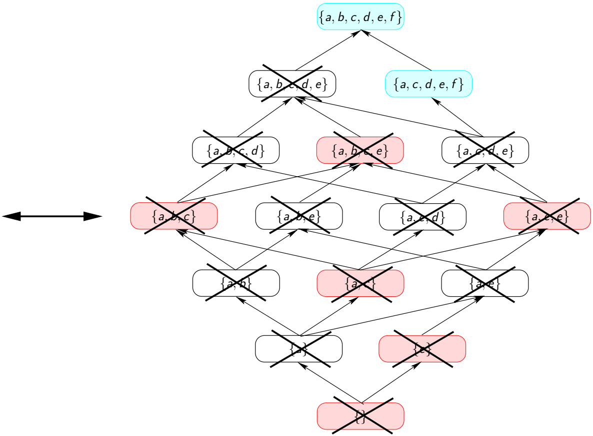




All consistent cuts that do not include a or c are eliminated.

Computing the Slice for EG(b)





All consistent cuts that do not include d or f are eliminated.

Computing the Slice for RCTL Predicates

- Efficient slicing algorithms have been developed for other predicates in RCTL.
- Time complexity: $O(|b|n^2|E|)$, where
  $|b|$: number of boolean and temporal operators in b
  n: number of processes
  $|E|$: number of events