Combinatorial Slice Theory

Mateus de Oliveira Oliveira

KTH Royal Institute of Technology
School of Computer Science and Communication
Dep. of Theoretical Computer Science

12 December 2013
- Slices and Slice Languages
- Slices in Concurrency Theory
- Slices in Combinatorial Graph Theory
- Slices in Equational Logic
Slices and Slice Languages
  ▶ General Theory

Slices in Concurrency Theory

Slices in Combinatorial Graph Theory

Slices in Equational Logic
Slices and Slice Languages
  ▶ General Theory

Slices in Concurrency Theory
  ▶ Hasse Diagram Generators and Petri Nets (Petri Nets 2009, Fundamenta Informaticae 2010)
  ▶ Canonizable Partial Order Generators (LATA 2012)

Slices in Combinatorial Graph Theory

Slices in Equational Logic
Slices and Slice Languages
  ▶ General Theory

Slices in Concurrency Theory
  ▶ Hasse Diagram Generators and Petri Nets (Petri Nets 2009, Fundamenta Informaticae 2010)
  ▶ Canonizable Partial Order Generators (LATA 2012)

Slices in Combinatorial Graph Theory
  ▶ Subgraphs Satisfying MSO Properties on \( z \)-Topologically Orderable Digraphs (IPEC 2013)

Slices in Equational Logic
- Slices and Slice Languages
  - General Theory
- Slices in Concurrency Theory
  - Hasse Diagram Generators and Petri Nets (Petri Nets 2009, Fundamenta Informaticae 2010)
  - Canonizable Partial Order Generators (LATA 2012)
- Slices in Combinatorial Graph Theory
  - Subgraphs Satisfying MSO Properties on $z$-Topologically Orderable Digraphs (IPEC 2013)
- Slices in Equational Logic
  - The Parameterized Complexity of Equational Logic (Submitted 2013)
PART I - SLICES AND SLICE LANGUAGES
● Roman Alphabet: $\Sigma = \{a, b, c, d, \ldots\}$
- Roman Alphabet: $\Sigma = \{a, b, c, d, \ldots\}$
- Greek Alphabet: $\Sigma = \{\alpha, \beta, \gamma, \delta, \ldots\}$
- **Roman Alphabet**: $\Sigma = \{a, b, c, d, \ldots\}$
- **Greek Alphabet**: $\Sigma = \{\alpha, \beta, \gamma, \delta, \ldots\}$
- **Slice Alphabet**: 

![Diagram of slice alphabets](image)
aquila non capit muscas
aquila non capit muscas

γνῶθι σεαυτόν
aquila non capiit muscas

γνῶθι σεαυτόν
aquila non capit muscas

γνῶθι σεαυτόν
Regular slice language: Generated by a DFA over a slice alphabet:
Regular slice language: Generated by a DFA over a slice alphabet:

Slice Language: $L$
Regular slice language: Generated by a DFA over a slice alphabet:

Slice Language: \( L \)
Regular slice language: Generated by a DFA over a slice alphabet:

Slice Language: L
Regular slice language: Generated by a DFA over a slice alphabet:
Regular slice language: Generated by a DFA over a slice alphabet:

Slice Language: L
- Regular slice language: Generated by a DFA over a slice alphabet:
Regular slice language: Generated by a DFA over a slice alphabet:

Slice Language: $L$

Graph Language: $L_G$
Regular slice language: Generated by a DFA over a slice alphabet:
Regular slice language: Generated by a DFA over a slice alphabet:
Regular slice language: Generated by a DFA over a slice alphabet:
Ways of representing regular slice languages slice languages:

- DFAs over slice alphabets
- Regular Expressions over Slice Alphabets
- Slice Graphs (To be defined later)
PART II - SLICES IN CONCURRENCY THEORY
DAG
DAG

PARTIAL ORDER
DAG

PARTIAL ORDER
DAG

PARTIAL ORDER

HASSE DIAGRAM
Petri Net
Petri Net
Petri Net
Petri Net
Petri Net
Some Protocols

Double Buffer
Some Protocols

Double Buffer

Producer-Consumer
Some Protocols

Double Buffer

Producer-Consumer

Mutual Exclusion
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets

Process

Diagram:

- p2
- a
- c
- p1
- b

Arrows and labels indicate the flow and connections between the processes.
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets

Process

Diagram of Petri Net:
- Places: p1, p2, b, i, p
- Transitions: a, c
- Arrows indicating transitions and places

Diagram details:
- Place p1 connected to p2 and p
- Transition a connecting p2 to p1
- Place c connecting to b and p1
- Transition c connecting p1 to b
- Place i connected to a
- Place b connected to p1

Net structure visualized with arrows and nodes.
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets
Partial Order Semantics for Petri Nets
Transitively Reduced Slice Graph

DAG

\[ \text{DAG} \]

\[ \text{Transitively Reduced Slice Graph} \]
Transitively Reduced Slice Graph

DAG

Partial Order
Transitively Reduced Slice Graph

DAG

Partial Order

Hasse Diagram
Transitively Reduced Slice Graph
Transitively Reduced Slice Graph

Each DAG in $L_{G}(SG)$ is the Hasse diagram of the partial order it gives rise to in $L_{PO}(SG)$.
A slice graph that is not transitively reduced.

\[\text{i} \xrightarrow{} \text{a} \xrightarrow{} \text{b} \xrightarrow{} \text{f}\]

\[\text{b} \cdots \text{b} \xrightarrow{m \text{ times}} \text{i} \xleftarrow{} \text{a} \cdots \text{a} \xrightarrow{n \text{ times}} \text{f}\]
A slice graph that is not transitively reduced.
A slice graph that is not transitively reduced.
Expressibility

Theorem (Expressibility Theorem)

*For every bounded $p/t$-net $N$ one can compute a transitively reduced slice graph $SG_N$, such that $\mathcal{L}_{PO}(N) = \mathcal{L}_{PO}(SG_N)$.*

- For many years it was even unclear whether the partial order behavior of general Petri nets could be *canonically* represented by a finite behavioral structure.
- Several approaches have been proposed: Event structures, Unfoldings, Concurrent Automata...
Verification - Desired Scenarios - Inclusion

\[ N = \text{System} \]

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{c}
\end{array}
\]

\[
\begin{array}{c}
\text{b} \\
\text{c}
\end{array}
\]
Verification - Desired Scenarios - Inclusion

\[ N = \text{System} \]

Desired Scenarios.
Verification - Desired Scenarios - Inclusion

Desired Scenarios.
Verification - Desired Scenarios - Inclusion

Desired Scenarios.

Does the behavior of $N$ includes all the desired scenarios?
Verification - Desired Scenarios - Inclusion

Desired Scenarios.

\[ L_{PO}(SG) \subseteq L_{PO}(N) \]
Verification - Undesired Scenarios - Emp. Intersection

Undesired Scenarios.
Verification - Undesired Scenarios - Emp. Intersection

SG

N = System

Does the behavior of N includes some undesired scenario?

Undesired Scenarios.
Verification - Undesired Scenarios - Emp. Intersection

Does the behavior of N includes some undesired scenario?

\[ L_{PO}(SG) \cap L_{PO}(N) = \emptyset \]
Corollary: Comparison of the behavior of two Nets

- Given bounded nets $N_1, N_2$, is $L_{PO}(N_1) \subseteq L_{PO}(N_2)$?
- Construct $SG_{N_1}$. Test whether $L_{PO}(SG_{N_1}) \subseteq L_{PO}(N_2)$?
- This was an open problem!
Synthesis

Given SG

```
   a  
  /   
 i    b
     ↘
     f
  ↙    
  c  
```

Mateus de Oliveira Oliveira (KTH Royal Institute of Technology School of Computer Science and Communication Dep. of Theoretical Combinatorial Slice Theory)

12 December 2013 75 / 139
Synthesis

Given SG

Synthesize N
Synthesis

Given SG

Synthesize N

Such That

\[ L_{PO}(SG) \subseteq L_{PO}(N_{SG}) \]
Synthesis

Given SG

Synthesize N

Such That

\[ L_{PO}(HG) \subseteq L_{PO}(N_{SG}) \]

AND

\[ L_{PO}(HG) \subseteq L_{PO}(N') \implies L_{PO}(N_{SG}) \subseteq L_{PO}(N') \]
Synthesis of distributed systems is considered a killer application for true concurrency semantics.

Synthesis for the sequential semantics (Badouel-Darondeau).

Synthesis for the execution semantics for finite families of scenarios (Lorenz et. Al.).
Transitive Reduction of Slice Graphs

Given a Slice graph $SG$ construct a Transitively Reduced Slice Graph $SG'$ such that $L_{PO}(SG) = L_{PO}(SG')$.

- Makes specification easier.
- Establishes a connection between Petri Nets and other methods for specifying Partial Orders.
PART III - SLICES IN COMBINATORIAL GRAPH THEORY

MSO FORMULA → ALGORITHMIC METATHEOREM → ALGORITHM
Motivation for Algorithmic Metatheorems:

- Many problems on graphs are very hard to be solved by computers.
- Ex: NP-complete problems.
- Turnaround:
  - Fix a way of writing the definition of a problem. Ex: MSO logic.
  - Identify large classes of graphs for which problems specified in this way can be solved efficiently.
Monadic Second Order Logic

- MSO\(_2\) : Extends first order logic by adding quantification over sets of vertices and edges.
- 3-colorability: Let \( G = (V, E) \) be a digraph.

\[
(\exists V_1, V_2, V_3) (V_1 \cup V_2 \cup V_3 = V) \land (\forall uv \in E) [u \in V_i \Rightarrow v \notin V_i]
\]

- Hamiltonian Path:

\[
(\exists E_1) [PATH(V, E_1)]
\]
Courcelle: Given MSO formula $\varphi$ and tree decomposition of width $w$ of $G$. Decide whether $G \models \varphi$ in time $f(\varphi, w) \cdot n^{O(1)}$. 
Courcelle: Given MSO formula $\varphi$ and tree decomposition of width $w$ of $G$. Decide whether $G \models \varphi$ in time $f(\varphi, w) \cdot n^{O(1)}$.

Arnborg-Lagergren-Seese: Count the number of solutions for problems expressible in MSO$_2$ in time $f(\varphi, w) \cdot n^{O(1)}$. 
Courcelle: Given MSO formula $\varphi$ and tree decomposition of width $w$ of $G$. Decide whether $G \models \varphi$ in time $f(\varphi, w) \cdot n^{O(1)}$.

Arnborg-Lagergren-Seese: Count the number of solutions for problems expressible in MSO$_2$ in time $f(\varphi, w) \cdot n^{O(1)}$.

In this work: Partial extension of the results above to directed width measures.
Zig-Zag Number
3 - Topological Order

\[
\begin{align*}
&\text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \\
&\text{a} \rightarrow \text{d} \rightarrow \text{c} \rightarrow \text{b}
\end{align*}
\]
3 - Topological Order
3 - Topological Order
3 - Topological Order

2 - Topological Order
3 - Topological Order

2 - Topological Order

zig-zag number $zn(G)$: Minimal $z$ for which $G$ admits a $z$-topological ordering.
zig-zag number $zn(G)$: Minimal $z$ for which $G$ admits a $z$-topological ordering.

Any DAG has zig-zag number 1.
**Definition**

A slice language $\mathcal{L}$ is $z$-saturated if whenever $G \in \mathcal{L}_G$ we have that all unit decompositions of $G$ of zig-zag number at most $z$ belong to $\mathcal{L}$. 

\[
\begin{array}{c}
\text{Diagram:}
\end{array}
\]
**Definition**

A slice language $\mathcal{L}$ is $z$-saturated if whenever $G \in \mathcal{L}_G$ we have that all unit decompositions of $G$ of zig-zag number at most $z$ belong to $\mathcal{L}$.

$$\in \mathcal{L}_G$$

2-Satur.
Definition

A slice language $\mathcal{L}$ is $z$-saturated if whenever $G \in \mathcal{L}_G$ we have that all unit decompositions of $G$ of zig-zag number at most $z$ belong to $\mathcal{L}$.
Slice Theoretic Algorithmic Metatheorem
A digraph $G$ is the union of $k$ paths if $G = p_1 \cup p_2 \cup \ldots \cup p_k$.

**Theorem**

For any $z, k \in \mathbb{N}$ and any MSO$_2$ formula $\varphi$, there exists a $z$-saturated regular slice language $L(\varphi, z, k)$ such that $L_G(\varphi, z, k)$ consists of all digraphs

- satisfying $\varphi$
- of zig-zag number at most $z$
- which are the union of $k$ directed paths.
$\mathbf{U} =$

![Diagram showing a graph with nodes labeled a, b, c, and d connected by arrows with weights 1, 2, and 3.](image)
\( U = \)

\[ L(U, c): \text{Set of all sub-unit decompositions of } U \text{ of width at most } c. \]
\( \mathcal{L}(U, c) \): Set of all sub-unit decompositions of \( U \) of width at most \( c \).

\( \mathcal{L}(U, c) \) is a finite regular slice language.
Given a digraph $G$ and a unit decomposition $U = S_1 S_2 \ldots S_n$ of zig-zag number $z$, the set of all subgraphs of $G$ which satisfy $\varphi$ and are the union of $k$ paths is represented by the slice language

$$\mathcal{L}(U, \varphi, z, k) = \mathcal{L}(\varphi, z, k) \cap \mathcal{L}(U, k \cdot z)$$  (1)
**Theorem (Main Theorem)**

Given a digraph $G$ and a unit decomposition $U = S_1 S_2 \ldots S_n$ of zig-zag number $z$, the set of all subgraphs of $G$ which satisfy $\varphi$ and are the union of $k$ paths is represented by the slice language

$$\mathcal{L}(U, \varphi, z, k) = \mathcal{L}(\varphi, z, k) \cap \mathcal{L}(U, k \cdot z) \quad (1)$$

- $\mathcal{L}(U, \varphi, z, k)$ can be represented by an acyclic DFA on $f(\varphi, k, z) \cdot n^{O(z \cdot k)}$ states.
Theorem (Main Theorem)

Given a digraph $G$ and a unit decomposition $U = S_1 S_2 ... S_n$ of zig-zag number $z$, the set of all subgraphs of $G$ which satisfy $\varphi$ and are the union of $k$ paths is represented by the slice language

$$\mathcal{L}(U, \varphi, z, k) = \mathcal{L}(\varphi, z, k) \cap \mathcal{L}(U, k \cdot z)$$ (1)

- $\mathcal{L}(U, \varphi, z, k)$ can be represented by an acyclic DFA on $f(\varphi, k, z) \cdot n^{O(z \cdot k)}$ states.
- Counting the number of subgraphs of $G$ that satisfy $\varphi$ and that are the union of $k$ paths can be done in time $f(\varphi, k, z) \cdot n^{O(z \cdot k)}$. 
Applications
Hamiltonian cycles are the union of two paths. Then counting the number of Hamiltonian cycles can be done in time $n^{O(2^z)}$. 
Hamiltonian cycles are the union of two paths. Then counting the number of Hamiltonian cycles can be done in time $n^{O(2^z)}$.

Other applications:
- Hamiltonian cycles are the union of two paths. Then counting the number of Hamiltonian cycles can be done in time $n^{O(2^z)}$.
- Other applications:
  - Counting the number of 3-colorable subgraphs that are the union of $k$ paths.
- Hamiltonian cycles are the union of two paths. Then counting the number of Hamiltonian cycles can be done in time $n^{O(2^x)}$.
- Other applications:
  - Counting the number of 3-colorable subgraphs that are the union of $k$ paths.
  - Counting the number of planar subgraphs that are the union of $k$ paths.
Hamiltonian cycles are the union of two paths. Then counting the number of Hamiltonian cycles can be done in time $n^{O(2^z)}$.

Other applications:
- Counting the number of 3-colorable subgraphs that are the union of $k$ paths.
- Counting the number of planar subgraphs that are the union of $k$ paths.
- Counting the number of Hamiltonian subgraphs that are the union of $k$ paths.
Hamiltonian cycles are the union of two paths. Then counting the number of Hamiltonian cycles can be done in time $n^{O(2^z)}$.

Other applications:

- Counting the number of 3-colorable subgraphs that are the union of $k$ paths.
- Counting the number of planar subgraphs that are the union of $k$ paths.
- Counting the number of Hamiltonian subgraphs that are the union of $k$ paths.
- Counting the number of subgraphs satisfying any minor closed property and which are the union of $k$ paths.
Hamiltonian cycles are the union of two paths. Then counting the number of Hamiltonian cycles can be done in time $n^{O(2^z)}$.

Other applications:
- Counting the number of 3-colorable subgraphs that are the union of $k$ paths.
- Counting the number of planar subgraphs that are the union of $k$ paths.
- Counting the number of Hamiltonian subgraphs that are the union of $k$ paths.
- Counting the number of subgraphs satisfying any minor closed property and which are the union of $k$ paths.
- ...
Slices in Equational Logic
- A set $E$ of equations (axioms):
  - $x + (y + z) = (x + y) + z$
  - $x + 0 = x = 0 + x$
  - $inv(x) + x = 0$

- A set of inference rules:
  - Reflexivity
  - Symmetry
  - Transitivity
  - Congruence
  - Substitution

- Is it true that $inv((x + y) + z) + (x + (y + z)) = 0$?
It is undecidable to determine whether an equation follows from a set of axioms. (Computers cannot solve.)

Until now the problem has been addressed using rewriting techniques:

- e.g. Knuth-Bendix Method and its variants.
New Method: Ordered Equations and Unit Decompositions

\[ x \cdot (y + z) = x \cdot y + x \cdot z \]
New Method: Ordered Equations and Unit Decompositions

\[ x \cdot ( y + z ) = x \cdot y + x \cdot z \]
New Method: Ordered Equations and Unit Decompositions

\[ x \cdot (y + z) = x \cdot y + x \cdot z \]

\[ x \cdot (y + z) = x \cdot y + x \cdot z \]
New Method: Ordered Equations and Unit Decompositions

\[ x \cdot (y + z) = x \cdot y + x \cdot z \]

\[ x \cdot (y + z) = x \cdot y + x \cdot z \]
Representing an infinite set of equations

\[ f(x_1, x_2) = g(x_1, x_2) \]
Representing an infinite set of equations

\[ f(x_1, x_2) = g(x_1, x_2) \]
Representing an infinite set of equations

\[ f(x_1, x_2) = g(x_1, x_2) \]

\[ f(x_1, h(x_2, x_3)) = g(x_1, h(x_2, x_3)) \]
Representing an infinite set of equations

\[ f(x_1, x_2) = g(x_1, x_2) \]

\[ f(x_1, h(x_2, x_3)) = g(x_1, h(x_2, x_3)) \]
Representing an infinite set of equations

\[ f(x_1, x_2) = g(x_1, x_2) \]

\[ f(x_1, h(x_2, x_3)) = g(x_1, h(x_2, x_3)) \]

\[ f(x_1, h(x_2, h(x_3, x_4))) = g(x_1, h(x_2, h(x_3, x_4))) \]
Representing an infinite set of equations

\[ f(x_1, x_2) = g(x_1, x_2) \]

\[ f(x_1, h(x_2, x_3)) = g(x_1, h(x_2, x_3)) \]

\[ f(x_1, h(x_2, h(x_3, x_4))) = g(x_1, h(x_2, h(x_3, x_4))) \]
Representing an infinite set of equations

\[ f(x_1, x_2) = g(x_1, x_2) \]

\[ f(x, h(x_2, x_3)) = g(x, h(x_2, x_3)) \]

\[ f(x, h(x_2, h(x_3, x_4))) = g(x, h(x_2, h(x_3, x_4))) \]

\[ f(x, h(x_2, h(x_3, \ldots))) = g(x, h(x_2, h(x_3, \ldots))) \]
Representing an infinite set of equations

\[ f(x_1, x_2) = g(x_1, x_2) \]

\[ f(x_1, h(x_2, x_3)) = g(x_1, h(x_2, x_3)) \]

\[ f(x_1, h(x_2, h(x_3, x_4))) = g(x_1, h(x_2, h(x_3, x_4))) \]

\[ f(x_1, h(x_2, h(x_3, \ldots ))) = g(x_1, h(x_2, h(x_3, \ldots ))) \]
New Question:
- Can we find a proof of an equation $t_1 = t_2$ in depth $d$ using only equations of width $c$ and with a bound $b$?
New Question:

- Can we find a proof of an equation \( t_1 = t_2 \) in depth \( d \) using only equations of width \( c \) and with a bound \( b \)?
- No exhaustive search: already in depth 1, an infinite number of possibilities to try:
New Question:
- Can we find a proof of an equation $t_1 = t_2$ in depth $d$ using only equations of width $c$ and with a bound $b$?
- No exhaustive search: already in depth 1, an infinite number of possibilities to try:
- For undecidable equational theories, rewriting techniques may fail.
Using Slice Theory: Yes! In Polynomial time!

**Theorem**

For all $d, c, b \in \mathbb{N}$, and all set of axioms $E$, all ordered equation $(t_1 = t_2, \omega)$ one may determine whether $E^c, b_d \vdash (t_1 = t_2, \omega)$ in time $f(E, d, c, b) \cdot |t_1 = t_2|$. 

**Theorem**

For all $d, c, b \in \mathbb{N}$, all set of axioms $E$ and all classical equation $t_1 = t_2$ one may determine whether there exists an oriented ordering $\omega$ such that $E^c, b_d \vdash (t_1 = t_2, \omega)$ in time $f(E, d, c, b) \cdot |t_1 = t_2|^O(c)$. 

Mateus de Oliveira Oliveira (KTH Royal Institute of Technology School of Computer Science and Communication Dep. of Theoretical Combinatorial Slice Theory)
Highlight of The Proof
PART V - Conclusion: Combinatorial Slice Theory

- Represent infinite families of combinatorial objects via slice languages.
- Provide ways of manipulating infinite families of combinatorial objects by manipulating their respective slice languages.
- Applications:
  - Solve several problems within the partial order theory of concurrency.
  - Provide an algorithmic metatheorem for directed width measures.
  - New parameterized algorithms for the provability of equations in equational logic.
Thank you!
Token Flow

Is PO a Scenario of N?
Theorem (Juhás-Lorenz-Desel-2005)

A partial order po is an execution of N if for each place p of N it is possible to associate a p-token-flow $f_p$ to the edges of PO.

Theorem

If for each edge e of the Hasse diagram of po there is at least a place for which $f_p(e) > 0$, then po is a causal order of N.
Interlaced Flow

A Set of Flows

A Process

Process
Interlaced Flow

N

Is HD the Hasse diagram of a partial order generated by N?
Interlaced Flow

**Theorem (Oliveira2009)**

The partial order $p_0$ induced by a Hasse diagram $hd$ is an execution of $N$ if for each place $p$ of $N$ it is possible to associate a $p$-Interlaced-Flow $f_p$ to the edges of $hd$.

**Theorem**

If for each edge $e$ of the $hd$ there is at least a place for which $f_p(e)[1] > 0$, then $p_0$ is a causal order of $N$. 

Interlaced Flow

\[ \text{Produced Before } v \text{ and consumed by After } v' \]
\[ \text{Produced Before } v \text{ and consumed by } By \ v' \]
\[ \text{Produced } By \ v \text{ and consumed After } v' \]
\[ \text{Produced } By \ v \text{ and consumed } By \ v' \]
Interlaced Flow

$N$

$\pi$

$b_1$

$b_2$

$a$

$b_3$

$a$

$b_4$

$\varepsilon$

$b_1: \ \iota \quad (1,0,0,0) \quad a \quad (0,0,1) \quad a \quad (0,0,1,0) \quad \varepsilon$

$b_2: \ \iota \quad (1,0,0,0) \quad a \quad a \quad \varepsilon$

$b_3: \ \iota \quad (1,0,0,0) \quad a \quad (1,0,0,0) \quad \varepsilon$

$b_4: \ \iota \quad (1,0,0,0) \quad a \quad (1,0,0,0) \quad \varepsilon$

$H$

$(S_1,R_1)\circ(S_2,R_2)\circ(S_3,R_3)\circ(S_4,R_4)$

$(S_1,R_1)$

$(S_2,R_2)$

$(S_3,R_3)$

$(S_4,R_4)$
Interlaced Flow

Many possible Interlaced Flows

Many possible equivalent Processes
Hasse Coloring

\[ \text{Diagram 1} \]

\[ \text{Diagram 2} \]
Hasse Coloring

\begin{align*}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
1 \\
0
\end{array} \\
\begin{array}{c}
1 \\
0
\end{array}
\end{array}
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
1 \\
0
\end{array} \\
\begin{array}{c}
1 \\
0
\end{array}
\end{array}
\end{array}
\end{align*}
Hasse Coloring

\begin{center}
\begin{tikzpicture}
\draw[red, thick] (0,0) rectangle (1,2);
\node at (0.5,1) {a};
\node at (0.5,0.5) {1};
\node at (0.5,0) {1};
\node at (0.5,1.5) {1};
\end{tikzpicture}
\quad
\begin{tikzpicture}
\draw[red, thick] (0,0) rectangle (1,2);
\node at (0.5,1) {a};
\node at (0.5,0.5) {0};
\node at (0.5,0) {1};
\node at (0.5,1.5) {1};
\end{tikzpicture}
\end{center}
Hasse Coloring
Hasse Coloring

\[
\begin{array}{cccc}
\text{a} & \text{b} & \text{c} & \text{d} \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]
Hasse Coloring

\begin{center}
\begin{tikzpicture}
\node at (0,0) {a};
\node at (1.5,0) {b};
\node at (3,0) {c};
\node at (4.5,0) {d};
\node at (6,0) {e};
\node at (7.5,0) {f};
\end{tikzpicture}
\end{center}
Eliminating DAGs that are not Hasse diagrams
Eliminating DAGs that are not Hasse diagrams
Eliminating DAGs that are not Hasse diagrams