Declare Mining

FOR DUMMIES

Learn to:
- Leverage parallelism
- Exploit symmetry
- Use super-scalarity

Michael Westergaard
Christian Stahl
Hajo Reijers

Online version of slides:
westergaard.eu/short12157
Declarative Models

If bored, we have to go to a Britney concert

We can only be truly happy after going to a Britney concert

We cannot go to both a Britney and an Xtina concert

No explicit flow of control
Good for flexible processes
Declarative Mining

[Diagram with states and transitions labeled with names such as Bored, Britney, Happy, and Xtina Concert]
If we accept enough (say \( \geq 50\% \)) traces, we accept the constraint.
Declarative Mining

- Bored, Britney, Happy
- Bored, Beer, Happy
- 2 parameters \((p)\)
- 4 event classes \((n)\)
- \(4 \times 3 = 12\) instantiations
- In general \(n \times (n-1) \times \ldots \times (n-p+1)\)

Realistic: \(n \geq 30\)
Sometimes: \(p = 5\)
Then:
\[30 \times 29 \times 28 \times 27 \times 26 = 17,100,720\] instantiations
Classic Solution

If often (say ≥ 50%)

Then and often (say ≥ 50%)

Contraposition
Classic Solution

- If not
  - Britney Concert
  - and
  - True Happiness
  - often

- Then not
  - Britney Concert
  - True Happiness
  - often

Only try mining constraints for often occurring parameters
Problem

- High Risk, High Cost
  - Happens Rarely

- Low Risk, Low Cost
  - Happens Often

- Double Check
  - Pay High
  - REJECTS

- Pay Low
Outline

- Symmetry reduction
- Prefix sharing
- Parallelism
- Superscalarity
- Support
- Use cases
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Old</th>
<th>MINERful</th>
<th>(p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>129m</td>
<td>450m</td>
<td>(2)</td>
</tr>
<tr>
<td>Easy</td>
<td>550m</td>
<td>26s</td>
<td>(\leq 2)</td>
</tr>
<tr>
<td>Almost</td>
<td></td>
<td></td>
<td>(\leq 3)</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td>(\leq 5)</td>
</tr>
</tbody>
</table>

7,224,024 checks

Real data:
- 13,087 traces
- \(n = 24\)
- 24 - 5,100,480 instances
Foundation

Bored, Britney, Happy
Bored, Beer, Happy

- Convert log to efficient structure
- Instead of instantiating constraints, use a single parametrized instance
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Old</th>
<th>MINERful</th>
<th>Naïve</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>129m</td>
<td>117 s</td>
<td>19 s</td>
<td>2</td>
</tr>
<tr>
<td>Easy</td>
<td>550m</td>
<td>26 s</td>
<td>19 s</td>
<td>1 ≤ 2</td>
</tr>
<tr>
<td>Almost</td>
<td></td>
<td></td>
<td></td>
<td>2 ≤ 3</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td>117 s</td>
<td>3 ≤ 5</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td>215 m</td>
<td></td>
</tr>
</tbody>
</table>

- **7,224,024 checks**
- **158,928,528 checks**
- **67,749,981,760 checks**
Symmetry

- not co-existence \((A, B) = \neg \text{co-existence}(B, A)\)
- Symmetry group: set \(S = \{ A_1, A_2, \ldots, A_n \} \) so that any \(A_i \in S\) can be swapped for any \(A_j \in S\)
Symmetry

- Normally, a constraint has $n \cdot (n-1) \cdot \ldots \cdot (n-p+1)$ instantiations.
- A symmetry group $S$ means that $|S|!$ of those are the same.
- If $|S| = p$, the constraint “only” has $n \cdot (n-1) \cdot \ldots \cdot (n-p+1) / p!$ instantiations.
- This number is also known as $\binom{n}{p}$. 
Symmetry

Normally, a constraint has \( n \cdot (n - 1) \cdot \ldots \cdot (n - p + 1) \) instantiations. A symmetry group \( S \) means that \(|S|!\) of those are the same.

If \(|S| = p\), the constraint "only" has \( n \cdot (n - 1) \cdot \ldots \cdot (n - p + 1) / p! \) instantiations.

This number is also known as \( \binom{n}{p} \).

Choice 1 of 3:
12,144 instances
158,928,528 checks

Choice 1 of 3:
3! = 6

Choice 1 of 3:
2,024 instances
26,488,088 checks

Choice 1 of 5:
5! = 120

Choice 1 of 5:
5,100,480 instances
66,749,981,760 checks

Choice 1 of 5:
42,504 instances
556,249,848 checks
Symmetry

Britney Concert ➔ Xtina Concert

Britney Concert ➔ Xtina Concert

Xtina Concert ➔ Britney Concert

Xtina Concert ➔ Britney Concert

0 ➔ 1 ➔ 2 ➔ 3 ➔ 0

0 ➔ 1 ➔ 2 ➔ 3 ➔ 0

=
Symmetry

Here we mean whether \( L(A_1) = L(A_2) \).

We check this by checking that
\[
L(A_1) \setminus L(A_2) = L(A_1) \cap L(A_2)^c = \emptyset \text{ and }
L(A_1) \cdot L(A_2)^c = \emptyset \text{ and }
L(A_2) \cdot L(A_1)^c = \emptyset
\]
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Old</th>
<th>MINERful</th>
<th>Naïve</th>
<th>Sym</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>129m</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy</td>
<td>550m</td>
<td>26 s</td>
<td>19 s</td>
<td>16 s</td>
</tr>
<tr>
<td>Almost</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>215m</td>
<td>154 s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 ≤ 2</td>
<td></td>
<td>2 s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ≤ 3</td>
<td></td>
<td>7 s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 ≤ 5</td>
<td></td>
<td>56 s</td>
<td></td>
</tr>
</tbody>
</table>

- **Easy:** 26,488,088 checks
- **Almost:** 556,249,848 checks
- **All:** 66,749,981,760 checks
- **Old:** 129,000 checks
- **MINERful:** 550,000 checks
- **Naïve:** 117,000 checks
- **Sym:** 215,000 checks

**Total Checks:**
- 3,612,012 checks
- 7,224,024 checks
- 158,928,528 checks
Prefix Sharing

Bored, Britney, Happy

We replay the “Bored” transition twice...

Bored, Beer, Happy
Prefix Sharing

Bored, Britney, Happy

Beer, Happy

0 1 1

0 2

Britney

Happy

0 1 2

REJECTED
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Old</th>
<th>MINERful</th>
<th>Naïve</th>
<th>Sym</th>
<th>Prefix</th>
<th>Super</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>129m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy</td>
<td>550m</td>
<td>26 s</td>
<td>19 s</td>
<td>16 s</td>
<td>43 s</td>
<td>2 s</td>
</tr>
<tr>
<td>Almost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

While we do get sharing (factor 4), checking becomes more expensive.
Parallelization

- Mine each constraint on each core
- Split log and mine each fragment on each core
Parallelization
Parallelization
<table>
<thead>
<tr>
<th>Mine constraints in parallel</th>
<th>Mine traces in parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good for many constraints</td>
<td>Good for many traces</td>
</tr>
<tr>
<td>Assumes constraints are similar</td>
<td>Assumes traces are similar</td>
</tr>
<tr>
<td>Requires models are composable</td>
<td>Requires results are composable</td>
</tr>
</tbody>
</table>

True for our approach as we just count matching traces.

Better
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Old</th>
<th>MINERful</th>
<th>Naïve</th>
<th>Sym</th>
<th>Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>129m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy</td>
<td>550m</td>
<td>26 s</td>
<td>19 s</td>
<td>16 s</td>
<td>12 s</td>
</tr>
<tr>
<td>Almost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Just 2 computation threads
Superscalar Mining

- Superscalar CPUs (90s): improve speed by executing multiple instructions at once
- Superscalar mining: improve speed by mining multiple constraints at once
Superscalar Mining

Bored ➔ Britney Concert

Britney Concert ➔ True Happiness

Bored ➔ Britney

Britney ➔ Happy

2
Superscalar Mining

Graphical representation of states and transitions.
Superscalar Mining

Bored, Britney, Happy

Diagram showing transitions between states: Bored, Britney, Happy.
Superscalar Mining

- We do not want to split up symmetry groups (speed)
- We do not want to make the automaton too large (memory)
Superscalar Mining

- Given two sets of constraints, $A$ and $B$
- $A$ can be merged into $B$ if:
  - We do not break any symmetry groups doing so
  - All constraints in $A$ are special cases of all constraints in $B$
Superscalar Mining

Given two sets of constraints, $A$ and $B$

$A$ can be merged into $B$ if

- We do not break any symmetry groups doing so.
- All constraints in $A$ are special cases of all constraints in $B$ for all $S$ of $A$ there is a $S_B$ of $B$ such that $S_A \subseteq S_B$, and for all $S$ of $A$ with $S \cap S_B \neq \emptyset$, we have $S = S_A$.
- For all $C_A$ of $A$ there is a $C_B$ of $B$ so that $C_A \Rightarrow C_B$.

Note: This is not symmetrical! It captures that it is ok to add something easy to something hard, but not the other way around.
Superscalar Mining

[[A, B, C, D, E]]

- exclusive choice 1 of 3
  - exactly2
  - existence3
- exclusive choice 1 of 2
  - absence
  - existence2
  - strong init
- choice 1 of 3
  - choice 1 of 4
  - choice 1 of 5

[[A, B, C]]

- exclusive choice 1 of 3
- absence2
- absence3
- existence3

[[A, B]]

- exactly1
- not co-existence

[[A]]

- existence
- init
- choice 2 of 3

[[A, B, C, D]]

- exactly2
- existence3

[[A, B, D]]

- strong init

[[A, C]]

- init

[[A, B, C, D, E]]
for all $C_A$ of $A$ there is a $C_B$ of $B$ so that $C_A \Rightarrow C_B$.
Superscalar Mining

chain succession [[A], [B]]

chain response [[A], [B]]

alternate succession [[A], [B]]

not succession [[A], [B]]

response [[A], [B]]

responded existence [[A], [B]]

alternate response [[A], [B]]

succession [[A], [B]]

not chain succession [[A], [B]]

existence [[A]]

existence2 [[A]]

existence3 [[A]]

exactly1 [[A]]

exactly2 [[A]]

strong init [[A]]

absence2 [[A]]

absence3 [[A]]

init [[A]]

precedence [[A], [B]]

alternate precedence [[A], [B]]
Superscalar Mining

for all $C_A$ of $A$ there is a $C_B$ of $B$ so that $C_A \Rightarrow C_B$

for all $S_A$ of $A$ there is a $S_B$ of $B$ such that $S_A \subseteq S_B$, and for all $S$ of $A$ with $S \cap S_B \neq \emptyset$, we have $S = S_A$
Superscalar Mining

- We can check 34 Declare constraints using just 3 checks:
  - 20 “choices”: 53 states, SG \{{A, B, C, D, E}\}
  - 22 “orders”: 112 states, SG \{{A\}, \{B\}\}
  - 2 “outliers”: 7 states, SG \{{A\}, \{B\}\}
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Old</th>
<th>MINERful</th>
<th>Naïve</th>
<th>Sym</th>
<th>Par</th>
<th>Super</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>129m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy</td>
<td>550m</td>
<td>26 s</td>
<td>19 s</td>
<td>16 s</td>
<td>12 s</td>
<td>2 s</td>
</tr>
<tr>
<td>Almost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td>117 s</td>
<td>35 s</td>
<td>27 s</td>
<td>7 s</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td>215 m</td>
<td>154 s</td>
<td>108 s</td>
<td>56 s</td>
</tr>
</tbody>
</table>

Using 8 (slower) CPUs, we get below 30 seconds.
<table>
<thead>
<tr>
<th></th>
<th>Old</th>
<th>Naïve</th>
<th>Sym</th>
<th>Par</th>
<th>Super</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>129m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy</td>
<td>550m</td>
<td>26 s</td>
<td>19 s</td>
<td>16 s</td>
<td>12 s</td>
</tr>
<tr>
<td>Almost All</td>
<td></td>
<td>117 s</td>
<td>35 s</td>
<td>27 s</td>
<td>7 s</td>
</tr>
<tr>
<td>All</td>
<td>215 m</td>
<td>154 s</td>
<td>108 s</td>
<td>56 s</td>
<td></td>
</tr>
</tbody>
</table>
Support

Classically: Detect this using complex formula operations
Support

Classically: Detect this using complex formula operations

Declare doesn’t have non-trivial sub-behavior, so this just corresponds to preventing a task

The idea of the formula operations is to prevent certain sub-behavior and see if the constraint becomes trivially true.
Support

\[ \exists \] Britney Concert \equiv Xtina Concert \equiv true?

\[ \exists \] Britney Concert \equiv Xtina Concert \equiv true?

\[ \exists \] Britney Concert \equiv Xtina Concert \equiv true?
Support

\[
\left(\begin{array}{c}
\text{Britney Concert} \\
\text{Xtina Concert}
\end{array}\right)
\] = 0

\[
\left(\begin{array}{c}
\text{Britney Concert} \\
\text{Xtina Concert}
\end{array}\right)
\] = 0

\[
\left(\begin{array}{c}
\text{Britney Concert} \\
\text{Xtina Concert}
\end{array}\right)
\] = 0

Support

\[
\epsilon = \emptyset
\]
Support

\[ \exists \ ( \text{Britney Concert} \mid \text{Xtina Concert} \mid 0 \ \text{Britney Concert} \mid 0 \ \text{Xtina Concert} ) = \text{true} \]
This expression is our measurement of support for the not co-existence constraint
This expression is our measurement of support for the precedence constraint
This expression is our measurement of support for the response constraint.

The reason for having exclusions of more than one constraint is disjunctive splits.
Support

Conjunction with the support expression is always true $\Rightarrow$ positive support

$\left(\begin{array}{c}
\text{Britney Concert} \\
\text{Xtina Concert}
\end{array}\right) \land \left(\begin{array}{c}
\text{Britney Concert} \\
\text{Xtina Concert}
\end{array}\right) = \text{true}$

Conjunction with the support expression is always false $\Rightarrow$ negative support

$\left(\begin{array}{c}
\text{Plague} \\
\text{Cholera}
\end{array}\right) \land \left(\begin{array}{c}
\text{Plague} \\
\text{Cholera}
\end{array}\right) = \text{false}$
Support

\[ \text{Britney Concert} \rightarrow \text{Xtina Concert} \]

\[ \text{Plague} \rightarrow \text{Cholera} \]

"Holds against all odds"

\[ \text{Britney Concert} \land \text{Xtina Concert} = \text{true} \]

"Never had a chance to hold"

\[ \text{Plague} \land \text{Cholera} = \text{false} \]

This is by the way very related to apriori reduction

\[ (\text{Britney Concert} \land \text{Xtina Concert}) \land (\text{Plague} \land \text{Cholera}) = \text{true} \land \text{false} = \text{false} \]
Support

- Our notion of support coincides with previous
- Previous notion and ours induce a notion of confidence that is identical when previous is defined
- Our is computed automatically and works for all constraints
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>ProM Miner</th>
<th>MINERful</th>
<th>Unconstrained Miner</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed</strong></td>
<td>Slow!!</td>
<td>Fast</td>
<td>Faster</td>
</tr>
<tr>
<td><strong>Extensibility</strong></td>
<td>Good</td>
<td>None</td>
<td>Better</td>
</tr>
<tr>
<td><strong>Confidence choices</strong></td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Confidence generality</strong></td>
<td>None</td>
<td>None</td>
<td>Full</td>
</tr>
</tbody>
</table>
### Comparison

<table>
<thead>
<tr>
<th></th>
<th>ProM Miner</th>
<th>MINERful</th>
<th>Unconstrained Miner</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed</strong></td>
<td>Slow!!</td>
<td>None</td>
<td>Fast</td>
</tr>
<tr>
<td><strong>Extensibility</strong></td>
<td>Good</td>
<td>None</td>
<td>Better</td>
</tr>
<tr>
<td><strong>Confidence choices</strong></td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Confidence generality</strong></td>
<td>None</td>
<td>None</td>
<td>Full</td>
</tr>
</tbody>
</table>

The reason is that some choices of confidence do not make sense in general.
Use Case

Assuming this block structure...

Events of block A and block B can never happen together
not co-existence (A, B)

Events of block B can never happen after events of block C
not succession (C, B)
Output: log with 8986 traces and 20 event classes
Use Case

The UnconstrainedMiner 1.1 (August 2013) software is used to analyze log data. The log file is located at `/Users/michael/Downloads/mined_data.txt`. The output file is set to `mined_data.txt`. The software allows for the selection of constraints and mining options.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Mine</th>
</tr>
</thead>
<tbody>
<tr>
<td>choice 1 of 2</td>
<td>✔️</td>
</tr>
<tr>
<td>choice 1 of 3</td>
<td>✔️</td>
</tr>
<tr>
<td>choice 1 of 4</td>
<td>✔️</td>
</tr>
<tr>
<td>choice 1 of 5</td>
<td>✔️</td>
</tr>
<tr>
<td>choice 2 of 3</td>
<td>✔️</td>
</tr>
<tr>
<td>exclusive choice 1 of 2</td>
<td>✔️</td>
</tr>
<tr>
<td>exclusive choice 1 of 3</td>
<td>✔️</td>
</tr>
<tr>
<td>exclusive choice 2 of 3</td>
<td>✔️</td>
</tr>
</tbody>
</table>

The minimal matches percentage is 0.0. The software allows for in-memory loading and super-scalar mining. The number of computing threads is set to 4. The Start button is available to initiate the analysis.
Use Case

<table>
<thead>
<tr>
<th>constraint</th>
<th>parameters</th>
<th>matches</th>
<th>positive support</th>
<th>negative support</th>
<th>dependent support</th>
<th>parameter1</th>
<th>parameter2</th>
</tr>
</thead>
<tbody>
<tr>
<td>not chain success</td>
<td>[4-T2]-[2-T4]</td>
<td>8986</td>
<td>1319</td>
<td>0</td>
<td>1313</td>
<td>4-T2</td>
<td>2-T4</td>
</tr>
<tr>
<td>not success</td>
<td>[4-T2]-[2-T4]</td>
<td>8986</td>
<td>1313</td>
<td>0</td>
<td>1313</td>
<td>4-T2</td>
<td>2-T4</td>
</tr>
<tr>
<td>not chain success</td>
<td>[4-T4]-[2-T4]</td>
<td>8986</td>
<td>1310</td>
<td>0</td>
<td>1310</td>
<td>4-T4</td>
<td>2-T4</td>
</tr>
<tr>
<td>not success</td>
<td>[4-T4]-[2-T4]</td>
<td>8986</td>
<td>1310</td>
<td>0</td>
<td>1310</td>
<td>4-T4</td>
<td>2-T4</td>
</tr>
<tr>
<td>not chain success</td>
<td>[4-T3]-[2-T1]</td>
<td>8986</td>
<td>1299</td>
<td>0</td>
<td>1299</td>
<td>4-T3</td>
<td>2-T1</td>
</tr>
<tr>
<td>not success</td>
<td>[4-T3]-[2-T1]</td>
<td>8986</td>
<td>1299</td>
<td>0</td>
<td>1299</td>
<td>4-T3</td>
<td>2-T1</td>
</tr>
<tr>
<td>not chain success</td>
<td>[5-T4]-[2-T1]</td>
<td>8986</td>
<td>1290</td>
<td>0</td>
<td>1290</td>
<td>5-T4</td>
<td>2-T1</td>
</tr>
<tr>
<td>not success</td>
<td>[5-T4]-[2-T1]</td>
<td>8986</td>
<td>1290</td>
<td>0</td>
<td>1290</td>
<td>5-T4</td>
<td>2-T1</td>
</tr>
<tr>
<td>not chain success</td>
<td>[3-T3]-[2-T4]</td>
<td>8986</td>
<td>1289</td>
<td>0</td>
<td>1289</td>
<td>3-T3</td>
<td>2-T4</td>
</tr>
<tr>
<td>not success</td>
<td>[3-T3]-[2-T4]</td>
<td>8986</td>
<td>1289</td>
<td>0</td>
<td>1289</td>
<td>3-T3</td>
<td>2-T4</td>
</tr>
<tr>
<td>not chain success</td>
<td>[5-T4]-[3-T3]</td>
<td>8986</td>
<td>1287</td>
<td>0</td>
<td>1287</td>
<td>5-T4</td>
<td>3-T3</td>
</tr>
<tr>
<td>not success</td>
<td>[5-T4]-[3-T3]</td>
<td>8986</td>
<td>1287</td>
<td>0</td>
<td>1287</td>
<td>5-T4</td>
<td>3-T3</td>
</tr>
<tr>
<td>not chain success</td>
<td>[3-T4]-[2-T4]</td>
<td>8986</td>
<td>1286</td>
<td>0</td>
<td>1286</td>
<td>3-T4</td>
<td>2-T4</td>
</tr>
<tr>
<td>not success</td>
<td>[3-T4]-[2-T4]</td>
<td>8986</td>
<td>1286</td>
<td>0</td>
<td>1286</td>
<td>3-T4</td>
<td>2-T4</td>
</tr>
<tr>
<td>not chain success</td>
<td>[5-T4]-[2-T4]</td>
<td>8986</td>
<td>1283</td>
<td>0</td>
<td>1283</td>
<td>5-T4</td>
<td>2-T4</td>
</tr>
<tr>
<td>not success</td>
<td>[5-T4]-[2-T4]</td>
<td>8986</td>
<td>1283</td>
<td>0</td>
<td>1283</td>
<td>5-T4</td>
<td>2-T4</td>
</tr>
<tr>
<td>not chain success</td>
<td>[5-T4]-[3-T3]</td>
<td>8986</td>
<td>1282</td>
<td>0</td>
<td>1282</td>
<td>5-T4</td>
<td>3-T3</td>
</tr>
<tr>
<td>not success</td>
<td>[5-T4]-[3-T3]</td>
<td>8986</td>
<td>1282</td>
<td>0</td>
<td>1282</td>
<td>5-T4</td>
<td>3-T3</td>
</tr>
<tr>
<td>not chain success</td>
<td>[5-T2]-[2-T4]</td>
<td>8986</td>
<td>1280</td>
<td>0</td>
<td>1280</td>
<td>5-T2</td>
<td>2-T4</td>
</tr>
<tr>
<td>not success</td>
<td>[5-T2]-[2-T4]</td>
<td>8986</td>
<td>1280</td>
<td>0</td>
<td>1280</td>
<td>5-T2</td>
<td>2-T4</td>
</tr>
</tbody>
</table>
Use Case

Sort by name and parameters (to group similar constraints)
Use Case

Aggregate results for not co-existence and not succession
Use Case

<table>
<thead>
<tr>
<th></th>
<th>1_1</th>
<th>1_2</th>
<th>1_3</th>
<th>1_4</th>
<th>2_1</th>
<th>2_2</th>
<th>2_3</th>
<th>2_4</th>
<th>3_1</th>
<th>3_2</th>
<th>3_3</th>
<th>3_4</th>
<th>4_1</th>
<th>4_2</th>
<th>4_3</th>
<th>4_4</th>
<th>5_1</th>
<th>5_2</th>
<th>5_3</th>
<th>5_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1_1</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>1_2</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>1_3</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>1_4</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>2_1</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>2_2</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>2_3</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>2_4</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>3_1</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>3_2</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>3_3</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>3_4</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>4_1</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>4_2</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>4_3</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>4_4</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>5_1</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>5_2</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>5_3</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
<tr>
<td>5_4</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
<td>x, &lt;=</td>
</tr>
</tbody>
</table>

We clearly recognize the blocks...

...except for parallelism.
Use Case

A xor B, C, D, E

B → C, D, E

C → D, E
Use Case

A xor B, C, D, E

What about loops?

B → C, D, E

C → D, E
Use Case

Support and location in the hierarchy is computed automatically.
Use Case

weak relations

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case
```

```
weak relations
```

```
Use Case

weak relations

- loop
- !not chain succession
- !choice 1 of 5
- exclusive choice 2 of 3
- !a or b
- exactly2
- !alternate
- !a | b
- !choice 1 of 3
- !a xor b
- !a <- b
- !a -> b
- !not coexistence
- !not succession
- !not co-existence
- exactly1
- choice 2 of 3
- chain succession
- responded existence
- alternate
- !exclusive choice 1 of 3
- !exclusive choice 1 of 3
- !exclusive choice 1 of 3
Use Case
Use Case

alpha relations

top
a # b

a -> b

a <- b

bottom
a | b
## Use Cases

### Lattice Model

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>[&lt;-]</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
</tr>
<tr>
<td>D</td>
<td>[&lt;-]</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
</tr>
<tr>
<td>E</td>
<td>[&lt;-]</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
</tr>
</tbody>
</table>
## Use Cases

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[-&gt;]</td>
<td>[-&gt;]</td>
<td>[?-&lt;&gt;, ?(#), ?(-)&gt;], ?(</td>
<td></td>
<td>)]</td>
</tr>
<tr>
<td>B</td>
<td>[&lt;-&gt;]</td>
<td>[</td>
<td></td>
<td>]</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>[&lt;-&gt;]</td>
<td>[</td>
<td></td>
<td>]</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>[?-&lt;&gt;, ?(#), ?(-)&gt;], ?(</td>
<td></td>
<td>)]</td>
<td>[&lt;-]</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>[&lt;-&gt;]</td>
<td>[?-&lt;&gt;, ?(#), ?(-)&gt;], ?(</td>
<td></td>
<td>)]</td>
<td>[?-&lt;&gt;, ?(#), ?(-)&gt;], ?(</td>
</tr>
</tbody>
</table>

We just implemented the Alpha algorithm!
Use Cases
Use Cases
Conclusion

- Efficient implementation
- Tricks: symmetry, parallelism, superscalarity
- General concept of support

Anything other miners can produce, we can produce

...and we are fully extensible!
Future Work

- Better recognition of modules
- Use of model-checking techniques
- Directed model-checking for LTL
Conclusion

- Efficient implementation
- Tricks: symmetry, parallelism, superscalarity
- General concept of support

Anything other miners can produce, we can produce

...and we are fully extensible!