A Proof-Producing CSP Solver$^1$

Michael Veksler
Ofer Strichman

Technion - Israel Institute of Technology

$CSP − SAT$

(Fast forward: Now at PTHG - 2020)

$^1$Originally presented at AAAI’10
Introduction

CSP proofs

- It is easy to validate a solution,
- ... but difficult to validate UNSAT.
It is easy to validate a solution,
... but difficult to validate UNSAT.

We introduce a CSP solver which produces a machine-checkable deductive proof.
It is easy to validate a solution,
... but difficult to validate UNSAT.

We introduce a CSP solver which produces a machine-checkable deductive proof.

This also gives us a better unsatisfiable core,
It is easy to validate a solution,
... but difficult to validate UNSAT.

We introduce a CSP solver which produces a machine-checkable deductive proof.

This also gives us a better unsatisfiable core,
... and facilitates developments as in the SAT world.
SAT solvers produce such proofs.
Several killer-applications:

(cont’d...)
SAT solvers produce such proofs.
Several killer-applications:

- Validate UNSAT results.

(cont’d...)
SAT solvers produce such proofs.
Several killer-applications:

- Validate UNSAT results.
- Uses of the proof itself:
  - Interpolation-based model checking [M03].
... Several killer-applications (... cont'd):
... Several killer-applications (... cont’d):

- Selective uses of the **UNSAT core**:
CSP proofs
Why bother?

... Several killer-applications (... cont'd):

- Selective uses of the **UNSAT core**:
  - Abstraction-refinement in model-checking [AM03],
... Several killer-applications (... cont'd):

- Selective uses of the **UNSAT core**:
  - Abstraction-refinement in model-checking [AM03],
  - Identify environment assumptions that are used in the proof [KKB09],
... Several killer-applications (... cont’d):

- Selective uses of the **UNSAT core**:
  - Abstraction-refinement in model-checking [AM03],
  - Identify environment assumptions that are used in the proof [KKB09],
  - Faster solving of bitvector formulas [BKOSSB07].
CSP proofs

Why bother?

... Several killer-applications (... cont'd):

- Selective uses of the UNSAT core:
  - Abstraction-refinement in model-checking [AM03],
  - Identify environment assumptions that are used in the proof [KKB09],
  - Faster solving of bitvector formulas [BKOSSB07].

Can we foresee usage for proofs in CSP?
A deductive proof DAG
Introduction to proofs

- A deductive proof DAG
- The roots: $c \in \text{CSP}$.
A deductive proof DAG

The roots: \( c \in \text{CSP} \).

The sink represents \( \perp \).
A deductive proof DAG
- The roots: $c \in \text{CSP}$.
- The sink represents $\perp$.
- The nodes in between are derived.
A deductive proof DAG

- The roots: \( c \in \text{CSP} \).
- The sink represents \( \perp \).
- The nodes in between are derived.

\[
\begin{align*}
\langle \text{parent 1} \rangle & \quad \cdots \quad \langle \text{parent } n \rangle \\
\langle \text{consequent} \rangle & \quad [\langle \text{rule name} \rangle]
\end{align*}
\]
Resolution based proofs

- SAT solvers generate proofs:
  - From initial clauses to ( ).
  - Inference is via the *binary-resolution* rule.
Resolution based proofs

- SAT solvers generate proofs:
  - From initial clauses to ()
  - Inference is via the binary-resolution rule.

- Unlike SAT solvers, CSPs:
  - have non-Boolean domains, and
  - non-clausal constraints.
Resolution based proofs

- SAT solvers generate proofs:
  - From initial clauses to ( ).
  - Inference is via the binary-resolution rule.

- Unlike SAT solvers, CSPs:
  - have non-Boolean domains, and
  - non-clausal constraints.

Can this gap be bridged?
Let $s$ be a set of values.

A positive signed literal: $a \in s$, e.g., $a \in \{1, 2, 3\}$. 

Resolution based proofs
Signed CNF [BHM00] - definition

- Let $s$ be a set of values.
- A positive signed literal: $a \in s$, e.g., $a \in \{1, 2, 3\}$.
- A negative signed literal: $a \in \bar{s}$, e.g., $a \in \{4\}$.
Let $s$ be a set of values.

- A positive signed literal: $a \in s$, e.g., $a \in \{1, 2, 3\}$.
- A negative signed literal: $a \in \overline{s}$, e.g., $a \in \{4\}$.

- A signed clause is a disjunction of signed literals. e.g., $(a \in [1..3] \lor b \in \{4\})$
Resolution based proofs
Signed CNF - resolution

- A binary-resolution rule for signed-CNF:

\[
\frac{(\text{Literal}_1 \lor x \in A) \land (x \in B \lor \text{Literal}_2)}{(\text{Literal}_1 \lor x \in A \cap B \lor \text{Literal}_2)} (\text{sRes}(x))
\]
A binary-resolution rule for signed-CNF:

\[
\frac{(\text{Literals}_1 \lor x \in A) \quad (x \in B \lor \text{Literals}_2)}{(\text{Literals}_1 \lor x \in A \cap B \lor \text{Literals}_2)} \quad (s\text{Res}(x))
\]

This can be used with constraints given as signed-clauses.
A binary-resolution rule for signed-CNF:

$$(\text{Literals}_1 \lor x \in A) \quad (x \in B \lor \text{Literals}_2)$$

(sRes($x$))

$$(\text{Literals}_1 \lor x \in A \cap B \lor \text{Literals}_2)$$

This can be used with constraints given as signed-clauses.

But what about other constraints?

e.g. $\neq$, $\leq$, allDifferent($v_1, \ldots, v_k$)
Q: Why not convert constraints to signed clauses?
Q: Why not convert constraints to signed clauses?
A: A clause representation is inefficient.

e.g., \( x \neq y \) requires:

\[(x \neq 1 \lor y \neq 1) \land (x \neq 2 \lor y \neq 2) \land \cdots\]
CSP unsatisfiability proofs

The solution

Solution: introduce clauses lazily.
Solution: introduce clauses lazily.

Consider a general constraint \( c \), such that:

- In the context of \( l_1 \land l_2 \land \cdots \land l_n \),
- propagation of \( c \) implies \( l \):

\[
(l_1 \land \cdots \land l_n \land c) \rightarrow l
\]
Explanation clauses
The requirements

\[(l_1 \land \cdots \land l_n \land c) \rightarrow l\]

Find an explanation clause \(e\) such that:

- \(e\) is not too strong: \(c \rightarrow e\)
- \(e\) is strong enough: \((l_1 \land \cdots \land l_n \land e) \rightarrow l\)
The structure of a PCS proof (Now called HCSP)

$e_1, e_2, e_3$ – explanation clauses.
For every constraint there is an explanation clause:

\[
\frac{\langle constraint \rangle}{\langle explanation\ clause \rangle} (\langle rule name \rangle)
\]
Explanation rule – example 1

Constraint: $x \neq y$

\[
\frac{x \neq y}{(x \neq m \lor y \neq m)} (Ne(m))
\]
Explanation rule – example 1

Propagation:
- **context**: $l_1 : (x = 1), \quad l_2 : (y \in [1..100])$.
- **constraint**: $c : x \neq y$.
- **implies**: $l : (y \in [2..100])$. 
Explanation rule – example 1

Propagation:
- context: \( l_1 : (x = 1), \quad l_2 : (y \in [1..100]) \).
- constraint: \( c : x \neq y \).
- implies: \( l : (y \in [2..100]) \).

Explanation:
- \( e : (x \neq 1 \lor y \neq 1) \quad // = Ne(1) \)
Explanation rule – example 1

Propagation:
- context: $l_1: (x = 1), l_2: (y \in [1..100])$.
- constraint: $c: x \neq y$.
- implies: $l: (y \in [2..100])$.

Explanation:
- $e: (x \neq 1 \lor y \neq 1) \quad // = Ne(1)$

... indeed:
- $c \xrightarrow{Ne(1)} e$
- $(l_1 \land l_2 \land e) \rightarrow l$
Constraint: $x \leq y$

\[
\frac{x \leq y}{(x \in (-\infty, m] \lor y \in [m + 1, \infty))} \text{(LE}(m)\text{)}
\]
Explanation rule – example 2

Propagation:

- context: $l_1 : (x \in [1..3]), \quad l_2 : (y \in [0..2])$
- constraint: $c : x \leq y$.
- implies: $l : x \in [1..2]$
Explanation rule – example 2

Propagation:
- context: \( l_1 : (x \in [1..3]), l_2 : (y \in [0..2]) \)
- constraint: \( c : x \leq y \).
- implies: \( l : x \in [1..2] \)

Explanation:
- \( e : (x \in (-\infty, 2] \lor y \in [3, \infty)) \).  
  //  = LE(2)
Explanation rule – example 2

Propagation:
- context: $l_1 : (x \in [1..3])$, $l_2 : (y \in [0..2])$
- constraint: $c : x \leq y$.
- implies: $l : x \in [1..2]

Explanation:
- $e : (x \in (-\infty, 2] \lor y \in [3, \infty))$.  //  $= LE(2)$

...indeed:
- $c \xrightarrow{LE(2)} e$
- $(l_1 \land l_2 \land e) \rightarrow l$
Q: How does PCS instantiate the rules?

- Consider the last example \((LE(m))\). We took \(m = \max(Domain(y))\).
- Should we consider other values?
Q: How does PCS instantiate the rules?

- Consider the last example ($LE(m)$). We took $m = \max(Domain(y))$.
- Should we consider other values?
- Yes! (to be shown later)
Each constraint has its rule

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Name</th>
<th>Inference rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \neq b$</td>
<td>$Ne(m)$</td>
<td>$a \neq b$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(a \neq m \lor b \neq m)$</td>
</tr>
<tr>
<td>$x \leq y$</td>
<td>$LE(m)$</td>
<td>$x \leq y$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(x \in (-\infty, m] \lor y \in [m+1, \infty))$</td>
</tr>
<tr>
<td>$a = b$</td>
<td>$Eq(D)$</td>
<td>$a = b$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(a \not\in D \lor b \in D)$</td>
</tr>
<tr>
<td>$a \leq b + c$</td>
<td>$LE_+(m, n)$</td>
<td>$a \leq b + c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(a \in (-\infty, m+n] \lor b \in [m+1, \infty) \lor c \in [n+1, \infty))$</td>
</tr>
<tr>
<td>$a = b + c$</td>
<td>$EQ^a(l_b, u_b, l_c, u_c)$</td>
<td>$a = b + c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(a \in [l_b + l_c, u_b + u_c] \lor b \not\in [l_b, u_b] \lor c \not\in [l_c, u_c])$</td>
</tr>
<tr>
<td>$\text{AllDiff}(v_1, \ldots, v_k)$</td>
<td>$AD(D, V)$</td>
<td>$\text{AllDiff}(v_1, \ldots, v_k)$</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>D</td>
</tr>
</tbody>
</table>
The structure of a PCS proof

\[ e_1, e_2, e_3 \text{ – explanation clauses.} \]
But constraints are not axioms...

- So far we assumed that the constraints are axioms (unconditioned).
- Constraints can be conditioned, e.g., \((b \lor x \leq y)\).
But constraints are not axioms...

- So far we assumed that the constraints are axioms (unconditioned).
- Constraints can be conditioned, e.g., \((b \lor x \leq y)\).
- Each of the above rules can be extended trivially to handle disjunction, e.g.,

\[
\begin{align*}
    b \lor x & \leq y \\
    b \lor (x \in (-\infty, m] \lor y \in [m + 1, \infty))
\end{align*}
\]
PCS: architecture

Start → CP
  ↓ no conflict
  ↓ partial assignment
  ↓ full assignment
Decision → SAT
  ↓ backtrack level
  ↑ decision-level=0
Analyze-Conflict → UNSAT
  ↓ conflict
  ↑ CP
  → BackTrack
  ↑ no conflict
PCS is inspired by modern CDCL$^2$ SAT solvers.

The learning mechanism is used for constructing a resolution proof.
PCS is inspired by modern CDCL\(^2\) SAT solvers.

The learning mechanism is used for constructing a resolution proof.

- Constraints propagation can be depicted in an implication graph.
- .. which is called a conflict graph in case of a conflict.
PCS is inspired by modern CDCL$^2$ SAT solvers.

The learning mechanism is used for constructing a resolution proof.

⇒ Constraints propagation can be depicted in an implication graph.

.. which is called a conflict graph in case of a conflict.

⇐ Analyze-Conflict learns a new clause from the conflict graph.
PCS is inspired by modern CDCL$^2$ SAT solvers.

The learning mechanism is used for constructing a resolution proof.

- Constraints propagation can be depicted in an implication graph.
  - which is called a conflict graph in case of a conflict.

- Analyze-Conflict learns a new clause from the conflict graph.

If unsat:

- Starting from the empty clause, find the proof ‘cone’.
- Reconstruct a full proof.
Shows the context of implications.

**Example**

- $D(a) = \{1, 2\}$
- $D(b) = \{1, 2\}$
- $D(c) = \{1, 2\}$
Implication graph (\(\Rightarrow\))

- Shows the context of implications.

**Example**

![Implication graph diagram]

- \(D(b) = \{1, 2\}\)
- \(D(c) = \{1, 2\}\)
- \(a = 1\) \(\@\) \(1\)
- \(b \in \{1, 2\}\) \(\@\) \(0\)
- \(c \in \{1, 2\}\) \(\@\) \(0\)
Implication graph (\(\Rightarrow\))

- Shows the context of implications.

**Example**

- \(a = 1\, @\, 1\)
- \(b \in \{1, 2\} @ 0\)
- \(a \neq b\)
- \(a = 1 @ 1\)
- \(b = 2 @ 1\)
- \(a \neq b\)
- \(c \in \{1, 2\} @ 0\)

\(D(b) = \{1, 2\}\)
\(D(c) = \{1, 2\}\)
Implication graph ($\Rightarrow$)

- Shows the context of implications.

Example

- $a = 1 \oplus 1$
- $\neg b \in \{1, 2\} \oplus 0$
- $\neg a \neq b$
- $b = 2 \oplus 1$
- $\neg a = b$
- $a = 1 \oplus 1$
- $\neg a \neq c$
- $c = 2 \oplus 1$
- $\neg a = c$

$D(b) = \{1, 2\}$
$D(c) = \{\overline{1}, 2\}$
Implication graph (=>)

- Shows the context of implications.

Example

\[ D(b) = \{1, 2\} \cap 0 \]
\[ D(c) = \{1, 2\} \cap 0 \]

- \(a = 1\cap 1\)
- \(b = 2\cap 1\)
- \(c = 2\cap 1\)

- \(a \neq b\)
- \(b \neq c\)
- \(a \neq c\)

Conflict
Implication graph (=>)

• Shows the context of implications.

Example

\[
\begin{align*}
    b &\in \{1, 2\}@0 & a \neq b & b \neq c \\
    a &= 1@1 & a \neq b & c \neq 2@1 \\
    c &\in \{1, 2\}@0 & a \neq c & b \neq c
\end{align*}
\]
Implication graph (=>)

- Shows the context of implications.

Example

- \( b \in \{1, 2\} \) @ 0
- \( a = 1 \) @ 1
- \( c \in \{1, 2\} \) @ 0
- \( a = 1 \) @ 1
- \( b = 2 \) @ 1
- \( c = 2 \) @ 1

Arrows indicate constraints:
- \( a \neq b \)
- \( b \neq c \)
- \( a \neq c \)

Conflict:
- \( a \neq b \)
- \( b \neq c \)

Implied:
- \( a \neq c \)
Implication graph (=>)

- Shows the context of implications.

Example

```
\begin{align*}
\text{context implied constraint} \\
\end{align*}
```
**ANALYZE-CONFLICT** ($\leq$)

- $b \in \{1, 2\} @ 0$
  - $a \neq b$
- $a = 1 @ 1$
  - $a \neq b$
  - $a \neq c$
- $c \in \{1, 2\} @ 0$
  - $a \neq c$
- $b = 2 @ 1$
  - $b \neq c$
- $c = 2 @ 1$
  - $b \neq c$

- $e = (b \neq 2 \lor c \neq 2)$
- $cl =$

**Invariant:** $cl$ contradicts the literals of *front*. 
**Analyse-Conflict** ($\leq$)

- $b \in \{1, 2\} \oplus 0$
- $a = 1 \oplus 1$
- $c = 1 \oplus 1$
- $a \neq b$
- $a \neq c$
- $b \neq c$

**Invariant:** $cl = (b \neq 2 \lor c \neq 2)$

**Invariant:** $cl$ contradicts the literals of $front$. 
**Analyze-Conflict** ($\triangleleft=\triangleright$)

\[
\begin{align*}
  b &\in \{1, 2\}@0 \\
  a &\neq b \\
  c &\in \{1, 2\}@0 \\
  a &\neq c \\
  b &\neq c
\end{align*}
\]

\[
\begin{align*}
  a &\neq b \\
  b &\neq c \\
  a &\neq c
\end{align*}
\]

**Invariant**: $cl$ contradicts the literals of $front$.

- $e = (a \neq 1 \lor c \neq 1)$
- $cl = (b \neq 2 \lor c \neq 2)$
- $cl \leftarrow Resolve(cl, e, c)$
\textbf{ANALYZE-CONFLICT} \((\leq\!\!\!\!\!\leq)\)

\[
\begin{align*}
\text{front} & \quad a \neq b \\
\text{conflict} & \quad b \neq c
\end{align*}
\]

\[b \in \{1, 2\} @ 0\]

\[a = 1 @ 1\]

\[c \in \{1, 2\} @ 0\]

\[c = 2 @ 1\]

\[b = 2 @ 1\]

\[a \neq c\]

\[a \neq b\]

\[b \neq c\]

\[c \neq 2 @ 1\]

\[a \neq 1\]

\[b \neq 2\]

\[c \not\in \{1, 2\}\]

\[cl = (a \neq 1 \lor b \neq 2 \lor c \not\in \{1, 2\})\]

\textbf{Invariant:} \(cl\) contradicts the literals of \textbf{front}. 

Michael Veksler Ofer Strichman (Technion - Israel Institute of Technology)

CSP − SAT

A Proof-Producing CSP Solver

June 18, 2011 25 / 36
**Analyse-Conflict ($\leq$)**

- $b \in \{1, 2\} @ 0$
- $a = 1 @ 1$
- $c \in \{1, 2\} @ 0$
- $b = 2 @ 1$
- $c = 2 @ 1$

- $e = (a \neq 1 \lor b \neq 1)$
- $cl = (a \neq 1 \lor b \neq 2 \lor c \notin \{1, 2\})$
- $cl \leftarrow \text{Resolve}(cl, e, b)$

**Invariant:** $cl$ contradicts the literals of $front$. 
**ANALYZE-CONFLICT** ($\leq$)

- $b \in \{1, 2\} @ 0$
- $a \neq b$
- $a \neq c$
- $c = 2 @ 1$
- $b = 2 @ 1$
- $b \neq c$

**front**

**conflict**

- $cl = (a \neq 1 \lor b \notin \{1, 2\} \lor c \notin \{1, 2\})$

**Invariant:** $cl$ contradicts the literals of **front.**
**Analyze-Conflict** ($\leq$)

The resulting proof

\[(b \neq 2 \lor c \neq 2)\]

\[(a \neq 1 \lor c \neq 1)\]

\[(a \neq 1 \lor b \neq 2 \lor c \notin \{1, 2\})\]

\[(a \neq 1 \lor b \notin \{1, 2\} \lor c \notin \{1, 2\})\]
The resulting proof
The resulting proof (2)
Optimization 1: augmented explanation

Propagation:

- context: $l_1: (x \in [1..5]), l_2: (y \in [2..3])$
- constraint: $c: x \leq y.$
- implies: $l: x \in [1..3]$
Optimization 1: augmented explanation

Propagation:
- context: $l_1 : (x \in [1..5])$, $l_2 : (y \in [2..3])$
- constraint: $c : x \leq y$.
- implies: $l : x \in [1..3]$

Explanation:
- $e : (x \in (-\infty, 3] \lor y \in [4, \infty))$.  // = LE(3)
Optimization 1: augmented explanation

Propagation:
- context: $l_1 : (x \in [1..5]), l_2 : (y \in [2..3])$
- constraint: $c : x \leq y$.
- implies: $l : x \in [1..3]$

Explanation:
- $e : (x \in (-\infty, 3] \lor y \in [4, \infty))$. // $= LE(3)$

But we now continue to resolve $e$ with $cl$. 
Optimization 1: **augmented** explanation

Propagation:
- **context:** $l_1 : (x \in [1..5]), l_2 : (y \in [2..3])$
- **constraint:** $c : x \leq y$.
- **implies:** $l : x \in [1..3]$

Explanation:
- **$e : (x \in (-\infty, 3] \lor y \in [4, \infty)).$** $// = LE(3)$

But we now continue to resolve $e$ with $cl$.
- Let $cl = (x \in [6..8] \lor z \in [1..2])$.

$$\text{Resolve}(e, cl, x) = (y \in [4, \infty) \lor z \in [1..2])$$
Optimization 1: \textit{augmented} explanation

Let $cl = (x \in [6..8] \lor z \in [1..2])$.

$$Resolve(e, cl, x) = (y \in [4, \infty) \lor z \in [1..2])$$
Optimization 1: augmented explanation

- Let $cl = (x \in [6..8] \lor z \in [1..2])$.

  \[ \text{Resolve}(e, cl, x) = (y \in [4, \infty) \lor z \in [1..2]) \]

- Now consider $LE(5)$:

  \[ e' : (x \in (-\infty, 5] \lor y \in [6, \infty)). \quad // = LE(5) \]
Optimization 1: augmented explanation

Let \( cl = (x \in \mathbb{R} \cap [6..8] \lor z \in \mathbb{R} \cap [1..2]) \).

\[
\text{Resolve}(e, cl, x) = (y \in \mathbb{R} \cap [4, \infty) \lor z \in \mathbb{R} \cap [1..2])
\]

Now consider \( LE(5) \):

\[
e' : (x \in \mathbb{R} \cap (-\infty, 5] \lor y \in \mathbb{R} \cap [6, \infty)) \]

// = \( LE(5) \)

Resolve with \( cl \):

\[
\text{Resolve}(e', cl, x) = (y \in \mathbb{R} \cap [6, \infty) \lor z \in \mathbb{R} \cap [1, 2])
\]
Optimization 1: **augmented** explanation

- Let \( cl = (x \in [6..8] \lor z \in [1..2]) \).

  \[
  \text{Resolve}(e, cl, x) = (y \in [4, \infty) \lor z \in [1..2])
  \]

- Now consider \( LE(5) \):

  \[
  e' : (x \in (-\infty, 5] \lor y \in [6, \infty)) \quad // = LE(5)
  \]

- Resolve with \( cl \):

  \[
  \text{Resolve}(e', cl, x) = (y \in [6, \infty) \lor z \in [1, 2])
  \]

- \( e' \) is *not* an explanation, but it is good enough.
- We call it an **augmented** explanation.
Assume that $l_1 \land \cdots \land l_n \land c \rightarrow l$. 
Optimization 1: formalization

- Assume that $l_1 \land \cdots \land l_n \land c \rightarrow l$.
- Let $l' \in \text{cl}$ be a literal such that $\text{var}(l') = \text{var}(l)$.
Optimization 1: formalization

- Assume that $l_1 \land \cdots \land l_n \land c \rightarrow l$.
- Let $l' \in cl$ be a literal such that $\text{var}(l') = \text{var}(l)$.
- $e'$ is an augmented explanation if
  - $c \rightarrow e'$
  - $(l_1 \land \cdots \land l_n \land e') \rightarrow \neg l'$

Michael Veksler Ofer Strichman (Technion - Israel Institute of Technology)

CSP − SAT

A Proof-Producing CSP Solver

June 18, 2011 30 / 36
Assume that $l_1 \land \cdots \land l_n \land c \rightarrow l$.

Let $l' \in cl$ be a literal such that $\text{var}(l') = \text{var}(l)$.

$e'$ is an augmented explanation if

- $c \rightarrow e'$
- $(l_1 \land \cdots \land l_n \land e') \rightarrow \neg l'$

We choose $e'$ that results in the strongest resolvent.

In particular:

$$\text{Resolve}(e', cl, \text{var}(l)) \rightarrow \text{Resolve}(e, cl, \text{var}(l))$$
Optimization 2: Only consider relevant nodes

- Observation: \( \text{vars} \text{(explanation)} \subseteq \text{vars} \text{(predecessors)}. \)
Optimization 2: Only consider relevant nodes

- Observation: \( \text{vars(\text{explanation})} \subseteq \text{vars(\text{predecessors})} \).

- Example: \( \text{AllDiff} (x, y, z) \).

\[
\begin{align*}
  x &= 1 \\
  y &\in [1, 3] \\
  z &\in [1, 2] \\
  y &\in [2, 3] \\
  z &= 2 \\
\end{align*}
\]
Optimization 2: Only consider relevant nodes

- Observation: \( \text{vars(explanation)} \subseteq \text{vars(predecessors)}. \)

- Example: \( \text{AllDiff}(x, y, z). \)

\[
\begin{align*}
x &= 1 \\
y &\in [1, 3] \\
z &\in [1, 2]
\end{align*}
\]

\[
y \in [2, 3] \\
z = 2
\]

- \( z \) becomes irrelevant.
Optimization 3: Only consider distinct nodes

Consider a conflict graph that includes a chain:

\[ x \in [1..4] \] @ 2 \rightarrow x \in [1..3] \] @ 3 \rightarrow x \in [1..2] \] @ 4 \rightarrow \ldots \]
Optimization 3: Only consider *distinct* nodes

Consider a conflict graph that includes a chain:

![Graph](image)

Only right-most node matters.
Others will not change the resolvent.
New results (2011)

- PCS is now ≈ on-par with Mistral.
- Out of 2847 supported CSC’09 test cases (t/o is 200 secs)

<table>
<thead>
<tr>
<th>Case</th>
<th>Mistral</th>
<th>PCS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>shared</td>
<td>shared</td>
</tr>
<tr>
<td></td>
<td>time *</td>
<td>time</td>
</tr>
<tr>
<td>all</td>
<td>4.62</td>
<td>9.23 (x2.0)</td>
</tr>
<tr>
<td>with tables</td>
<td>4.38</td>
<td>12.7 (x2.9)</td>
</tr>
<tr>
<td>w/o tables</td>
<td>4.91</td>
<td>3.88 (x0.79)</td>
</tr>
<tr>
<td>with ≤</td>
<td>6.44</td>
<td>3.69 (x0.57)</td>
</tr>
</tbody>
</table>

*Shared time* - average time on cases solved by both.

* PCS is now ≈ on-par with Mistral.

Out of 2847 supported CSC’09 test cases (t/o is 200 secs)

<table>
<thead>
<tr>
<th>Case</th>
<th>Mistral</th>
<th>PCS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>shared</td>
<td>shared</td>
</tr>
<tr>
<td></td>
<td>time *</td>
<td>time</td>
</tr>
<tr>
<td>all</td>
<td>4.62</td>
<td>9.23 (x2.0)</td>
</tr>
<tr>
<td>with tables</td>
<td>4.38</td>
<td>12.7 (x2.9)</td>
</tr>
<tr>
<td>w/o tables</td>
<td>4.91</td>
<td>3.88 (x0.79)</td>
</tr>
<tr>
<td>with ≤</td>
<td>6.44</td>
<td>3.69 (x0.57)</td>
</tr>
</tbody>
</table>

*Shared time* - average time on cases solved by both.
Future work

- Non-clausal conflict analysis, (Update: see our JAIR'16 publication*)
- Interpolation algorithms (√),
- Performance, performance, performance. (Update: HCSP won 2 gold + 1 silver medal in the Minizinc challenge of 2016)

* Learning general constraints in CSP / JAIR 238, pp 135-153
HCSP is a Cool CSP solver that
... performs similar to Mistral, but
Summary

- HCSP is a Cool CSP solver that
- ... performs similar to Mistral, but
- ... produces machine-checkable proofs

(Download: https://strichman.net.technion.ac.il/haifacsp/download/)