A Proof-Producing CSP Solver\footnote{Originally presented at AAAI’10}

Michael Veksler
Ofer Strichman

Technion - Israel Institute of Technology

\textit{CSP – SAT}

June 18, 2011
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.
CSP proofs

Why bother?

SAT solvers produce such proofs.
Several killer-applications:

(cont’d...)
CSP proofs
Why bother?

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].
... 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
A deductive proof DAG

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

The roots: $c \in \text{CSP}$.

The sink represents $\perp$. 

\begin{itemize}
  \item A deductive proof DAG
  \item The roots: $c \in \text{CSP}$.
  \item The sink represents $\perp$.
\end{itemize}
A deductive proof DAG

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

- The roots: $c \in \text{CSP}$.
- The sink represents $\bot$.
- The nodes in between are derived.

\[
\frac{\langle \text{parent 1} \rangle \cdots \langle \text{parent n} \rangle}{\langle \text{consequent} \rangle} [\langle \text{rule name} \rangle]
\]
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., $a \in \{1, 2, 3\}$.

Alternative notations: $a \in [1..3]$, $a = 4$. 
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., $a \in \overline{\{4\}}$.
  - Alternative notations: $a \notin \{4\}$, $a \neq 4$. 
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., $a \in \{4\}$.
  - Alternative notations: $a \notin \{4\}$, $a \neq 4$.

- A signed clause is a disjunction of signed literals. e.g.,
  $(a \in [1..3] \lor b \in \{4\})$
A binary-resolution rule for signed-CNF:

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

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

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

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

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

But what about other constraints?
  e.g. \(\neq, \leq, \text{allDifferent}(v_1, \ldots, v_k)\)
Q: Why not convert constraints to signed clauses?
CSP unsatisfiability proofs

The challenge

Q: Why not convert constraints to signed clauses?
A: A clause representation is inefficient.

e.g., \( x \neq y \) requires:
\[
(x \neq 1 \lor y \neq 1) \land (x \neq 2 \lor y \neq 2) \land \cdots
\]
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**: \((l_1 \land \cdots \land l_n \land e) \rightarrow l\)
$e_1, e_2, e_3$ – explanation clauses.
For every constraint there is an explanation clause:

\[
\frac{\langle \text{constraint} \rangle}{\langle \text{explanation clause} \rangle} (\langle \text{rule name} \rangle)
\]
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**: $l_1 : (x = 1), \ l_2 : (y \in [1..100])$.
- **constraint**: $c : x \neq y$.
- **implies**: $l : (y \in [2..100])$.

Explanation:
- $e : (x \neq 1 \vee y \neq 1) \hspace{1em} // = Ne(1)$
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:

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

... indeed:

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

$$\frac{x \leq y}{(x \in (-\infty, m] \lor y \in [m + 1, \infty))} (LE(m))$$
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 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>$\text{Ne}(m)$</td>
<td>$a = b$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(a \neq m \lor b \neq m)$</td>
</tr>
<tr>
<td>$x \leq y$</td>
<td>$\text{LE}(m)$</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>$\text{Eq}(D)$</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>$\text{LE}_+(m, n)$</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>$\text{EQ}^a$</td>
<td>$a = b + c$</td>
</tr>
<tr>
<td></td>
<td>$(l_b, u_b, l_c, u_c)$</td>
<td>$(a \in [l_b+u_c, u_b+u_c] \lor b \not\in [l_b, u_b] \lor c \not\in [l_c, u_c])$</td>
</tr>
<tr>
<td>AllDiff($v_1, \ldots, v_k$)</td>
<td>$\text{AD}(D, V)$</td>
<td>AllDiff($v_1, \ldots, v_k$)</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</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.,

\[
\frac{b \lor x \leq y}{b \lor (x \in (-\infty, m] \lor y \in [m + 1, \infty))}
\]
PCS: architecture

- **Decide**
  - Full assignment
- **BackTrack**
  - $bl \geq 0$
- **CP**
  - Conflict
- **Analyze-Conflict**
  - $bl < 0$
- **SAT**
- **UNSAT**

- Partial assignment
- No conflict
From search to proof

- 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.
Implication graph (\(\Rightarrow\))

- 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

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

- Shows the context of implications.

### Example

- \(a = 1\oplus 1\)
- \(b \in \{1, 2\}\oplus 0\)
- \(c \in \{1, 2\}\oplus 0\)
- \(D(b) = \{1, 2\}\)
- \(D(c) = \{1, 2\}\)
- \(a \neq b\)
- \(b = 2\oplus 1\)
- \(a = 1\oplus 1\)
- \(c \neq b\)
Shows the context of implications.

Example

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

- Shows the context of implications.

**Example**

- \(a = 1\@1\)
- \(D(b) = \{, \emptyset\}\)
- \(D(c) = \{, \emptyset\}\)
- \(b \in \{1, 2\}@0\)
- \(a \neq b\)
- \(b \neq c\)
- \(b = 2\@1\)
- \(a = 1\@1\)
- \(a \neq b\)
- \(a \neq c\)
- \(c = 2\@1\)
- \(c \in \{1, 2\}@0\)
- \(a \neq c\)
- \(b \neq c\)
- Conflict
Implication graph (=>)

- Shows the context of implications.

Example

```
b ∈ \{1, 2\} @ 0
a = 1 @ 1

\text{context}

\text{constraint}
```

```
b = 2 @ 1
a \ne b

\text{conflict}
```

```
c = 2 @ 1
a \ne c
b \ne c

\text{implied}
```
Implication graph (=>)

- Shows the context of implications.

Example

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

Conflict

Context

Constraint

Implied
Implication graph (⇒)

- Shows the context of implications.

Example

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

Conflict

Implied context

Constraint
**Analyze-Conflict (<=)**

\[ b \in \{1, 2\} \land a \neq b \land \text{conflict} \]

\[ b = 2 \land a \neq c \land b \neq c \]

\[ c = 2 \land a \neq c \land b \neq c \]

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

\[ cl = \]

**Invariant:** \( cl \) contradicts the literals of \( \text{front} \).
**ANALYZE-CONFLICT (<=)**

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

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

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

\begin{itemize}
  \item \(b \in \{1, 2\} @ 0\)
  \item \(a = 1 @ 1\)
  \item \(c \in \{1, 2\} @ 0\)
  \item \(b = 2 @ 1\)
  \item \(c = 2 @ 1\)
\end{itemize}

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

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

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

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

- $e = (a \neq 1 \lor b \neq 1)$
- $cl = (a \neq 1 \lor b \neq 2 \lor c \not\in \{1, 2\})$
- $cl \leftarrow \text{Resolve}(cl, e, b)$

Diagram:

- $a \neq b$
- $b \neq c$
- $a \neq c$
- $b \neq c$

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

Arrows:
- $a \neq b$
- $b \neq c$
- $a \neq c$

Conflict:
- $b = 2 @ 1$
- $c = 2 @ 1$
- $b \neq c$

Front:
- $b \neq 1$
- $c \neq 1$

Resolve:
- $cl \leftarrow \text{Resolve}(cl, e, b)$
**ANALYZE-CONFLICT (\(\leq=\))**

\[
\begin{align*}
\text{front} & \quad a \neq b \\
& \quad a \neq c \\
& \quad b \neq c \\
\quad b \in \{1, 2\} \oplus 0 \\
& \quad a \neq 1 \oplus 1 \\
& \quad c \in \{1, 2\} \oplus 0 \\
& \quad b = 2 \oplus 1 \\
& \quad c = 2 \oplus 1 \\
\text{conflict} & \\
\end{align*}
\]

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

**Invariant:** \(cl\) contradicts the literals of \(\text{front}\).
**ANALYZE-CONFLICT (<=)**

The resulting proof

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

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

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

\[(a \neq 1 \lor b \notin \{1, 2\} \lor c \notin \{1, 2\}]\]
The resulting proof

\[
\begin{align*}
(a \neq 1 \lor b \neq 2 \lor c \notin \{1, 2\}) \\
(a \neq 1 \lor b \neq 1) \\
(a \neq 1 \lor b \notin \{1, 2\} \lor c \notin \{1, 2\}) \\
(a \neq \{1, 2\} \lor b \notin \{1, 2\} \lor c \notin \{1, 2\})
\end{align*}
\]
\textbf{ANALYZE-CONFLICT (\(\leq\))}

The resulting proof (2)

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

\[(a \in \{1, 2\})\]

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

\[(b \in \{1, 2\})\]

\[(c \notin \{1, 2\})\]

\[(c \in \{1, 2\})\]

\[R(a)\]

\[R(b)\]

\[R(c)\]
Optimization 1: \textit{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:
- $e : (x \in (-\infty, 3] \lor y \in [4, \infty)). \quad // = LE(3)$
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:
- \( e : (x \in (-\infty, 3] \lor y \in [4, \infty)). \quad // = LE(3) \)

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

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

\[
\text{Resolve}(e, cl, x) = (y \in [4, \infty) \lor z \in [1..2])
\]
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: \textit{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 $cl = (x \in [6..8] \lor z \in [1..2])$.
  
  $$Resolve(e, cl, x) = (y \in [4, \infty) \lor z \in [1..2])$$

- Now consider $LE(5)$:
  
  $e' : (x \in (-\infty, 5] \lor y \in [6, \infty))$. \hspace{1cm} // = LE(5)

- Resolve with $cl$:
  
  $$Resolve(e', cl, x) = (y \in [6, \infty] \lor z \in [1, 2])$$
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 $LE(5)$:
  
  $e' : (x \in (-\infty, 5] \lor y \in [6, \infty))$.  // $= LE(5)$

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

- $c \rightarrow e'$
- $(l_1 \land \cdots \land l_n \land e') \rightarrow \neg l'$
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
  - \( c \rightarrow e' \)
  - \( (l_1 \land \cdots \land l_n \land e') \rightarrow \neg l' \)
- 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: \( \text{vars(explanation)} \subseteq \text{vars(predecessors)}. \)
Optimization 2: Only consider relevant nodes

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

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

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

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

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

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

- \( z \) becomes irrelevant.
Consider a conflict graph that includes a chain:

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

Consider a conflict graph that includes a chain:

\[ x \in [1..4]@2 \rightarrow x \in [1..3]@3 \rightarrow x \in [1..2]@4 \rightarrow \ldots \]

Only right-most node matters.
Others will not change the resolvent.
**Performance**

- PCS participated in CSC’09
- For $n$-ary constraints, out of 14:

<table>
<thead>
<tr>
<th>category</th>
<th>rank SAT</th>
<th>rank UNSAT</th>
<th>rank total</th>
</tr>
</thead>
<tbody>
<tr>
<td>extension</td>
<td>9/14</td>
<td>6/14</td>
<td>9/14</td>
</tr>
<tr>
<td>intention</td>
<td>4/14</td>
<td>1/14</td>
<td>4/14</td>
</tr>
</tbody>
</table>

- 2-ary constraints PCS got poor results.
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></th>
<th>Mistral</th>
<th>PCS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>shared</td>
<td>shared</td>
</tr>
<tr>
<td>all</td>
<td>time *</td>
<td>success</td>
</tr>
<tr>
<td></td>
<td>4.62</td>
<td>9.23 (x2.0)</td>
</tr>
<tr>
<td></td>
<td>2187</td>
<td>2104 (-83)</td>
</tr>
<tr>
<td></td>
<td>success</td>
<td>success</td>
</tr>
<tr>
<td></td>
<td>2104</td>
<td>1963</td>
</tr>
<tr>
<td>with tables</td>
<td>4.38</td>
<td>12.7 (x2.9)</td>
</tr>
<tr>
<td></td>
<td>1216</td>
<td>1112 (-104)</td>
</tr>
<tr>
<td>w/o tables</td>
<td>4.91</td>
<td>3.88 (x0.79)</td>
</tr>
<tr>
<td></td>
<td>971</td>
<td>992 (+21)</td>
</tr>
<tr>
<td>with ≤</td>
<td>6.44</td>
<td>3.69 (x0.57)</td>
</tr>
<tr>
<td></td>
<td>576</td>
<td>628 (+52)</td>
</tr>
</tbody>
</table>

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

- Non-clausal conflict analysis,
- Interpolation algorithms (✓),
  - Word-level model checking?
- Performance, performance, performance.
PCS is a COOL CSP solver, which
PCS is a *COOL* CSP solver, which...

... performs similar to Mistral, but
PCS is a **COOL** CSP solver, which
... performs similar to Mistral, but
... produces machine-checkable proofs.

PCS: [http://tx.technion.ac.il/~mveksler/PCS/index.html](http://tx.technion.ac.il/~mveksler/PCS/index.html)