A proof-producing CSP solver: A proof supplement

Report IE/IS-2010-02

Michael Veksler
mveksler@tx.technion.ac.il

Ofer Strichman
ofers@ie.technion.ac.il

Technion Institute of Technology

April 12, 2010

Abstract

In [1] we described PCS, a CSP solver that can produce a machine-checkable deductive proof for an unsatisfiable input problem. This report supplements [1] in several ways: it provides soundness proof for the inference rules that were presented in Table 3; it provides proofs of correctness for several algorithms and claims; and, finally, it adds several missing algorithms and optimizations.

1 Introduction

This report is meant to be a supplemental document to [1]. It contains a mostly unstructured collection of things that were omitted from [1] mostly due to lack of space. It includes proofs for inference rules and algorithms, introduces more inference rules, and, finally, it presents missing algorithms and improvements to algorithms that were presented in [1].

2 Inference rules

In this section we prove the soundness of the inference rules that were introduced in Table 3 of [1], and prove that it is possible to match an inference rule for any possible constraint propagation. The latter established the completeness of CSP-Analyze-Conflict.

2.1 Soundness proofs for the inference rules

Here we prove the soundness of the inference rules from Table 3 in [1]. Throughout this section we assume that all constraints and inference rules refer to integer variables. This assumption is used only for convenience and is not a fundamental part of the work, which can be easily extended to reals.

Lemma 1 *For an integer constant $m \in \mathbb{Z}$ and integer variables $a, b \in \mathbb{Z}$*

$$\frac{a \leq b}{(a \in (-\infty, m] \vee b \in [m+1, \infty))} \quad (\text{LE}(m)) \ .$$

Proof: Combining the premise $a \leq b$ with the tautology $a \leq m \vee a > m$ yields $a \leq m \vee b > m$. Converting this to a signed clause gives the consequent of the rule $(a \in (-\infty, m] \vee b \in [m+1, \infty))$. □

Lemma 2 *Assuming $V \subseteq \{v_1, \ldots, v_k\}$ such that $|V| = 1 + |D|$*

$$\frac{\textit{All-diff}(v_1, \ldots, v_k)}{(\bigvee_{v \in V} v \notin D)} \quad (AD(D, V))$$

Proof: Given the premise All-diff$(v_1, \ldots, v_k)$, for $|D| = 0$, $|V| = 1$ the consequent is a tautology. Otherwise, due to counting considerations, there is no feasible assignment of $|D|$ different values to $|V| = |D|+1$ variables, or formally $\neg(\bigwedge_{v \in V} v \in D)$. After pushing the negation into the expression, this gives $(\bigvee_{v \in V} v \notin D)$ □

Lemma 3 *For an integer constant $m \in \mathbb{Z}$ and integer variables $a, b \in \mathbb{Z}$*

$$\frac{a \neq b}{(a \neq m \vee b \neq m)} \quad (\text{NE}(m))$$

Proof: Constraining the tautology $a \neq m \vee a = m$ by the premise $a \neq b$ results with $a \neq m \vee (a = m \wedge a \neq b)$ which implies $a \neq m \vee m \neq b$. Converting this to a signed clause gives the consequent of the rule $(a \neq m \vee b \neq m)$. □

Lemma 4 *For an arbitrary set of values D*

$$\frac{a = b}{(a \notin D \vee b \in D)} \quad (\text{EQ}(D))$$

Proof: Constraining the tautology $a \notin D \vee a \in D$ by the premise $a = b$ results with $a \notin D \vee (a \in D \wedge a = b)$ which implies the consequent of the rule $(a \notin D \vee b \in D)$. □

Lemma 5 *For integer constants $m, n \in \mathbb{Z}$ and integer variables $a, b, c \in \mathbb{Z}$*

$$\frac{a \leq b + c}{(a \in (-\infty, m+n] \vee b \in [m+1, \infty) \vee c \in [n+1, \infty))} \quad (LE_+(m, n))$$

Proof: Adding the premise $a \leq b + c$ to the right hand side of the tautology

$$\neg(b \leq m \wedge c \leq n) \vee (b \leq m \wedge c \leq n) \ ,$$

gives $\neg(b \leq m \wedge c \leq n) \vee a \leq m + n$. Pushing negation all the way to the literals yields $b > m \vee c > n \vee a \leq m + n$, which is equivalent to the consequent clause $(a \in (-\infty, m+n] \vee b \in [m+1, \infty) \vee c \in [n+1, \infty))$. □

Lemma 6 *For integer constants $l_b, u_b, l_c, u_c \in \mathbb{Z}$ and integer variables $a, b, c \in \mathbb{Z}$*

$$\frac{a = b + c}{(a \in [l_b + l_c, u_b + u_c] \vee b \notin [l_b, u_b] \vee c \notin [l_c, u_c])} \quad (EQ^a_+(l_b, u_b, l_c, u_c))$$

Proof: The tautology $\neg(l_b \leq b \leq u_b \wedge l_c \leq c \leq u_c) \vee (l_b \leq b \leq u_b \wedge l_c \leq c \leq u_c)$ can be rewritten as $\neg(b \in [l_b, u_b] \wedge c \in [l_c, u_c]) \vee (l_b \leq b \leq u_b \wedge l_c \leq c \leq u_c)$, which, after combining with the premise $a = b + c$, becomes

$$\neg(b \in [l_b, u_b] \wedge c \in [l_c, u_c] \vee (l_b + l_c \leq a \leq u_b + u_c) \ .$$

Writing comparisons as signed literals yields

$$\neg(b \in [l_b, u_b] \wedge c \in [l_c, u_c]) \vee a \in [l_b + l_c, u_b + u_c] \ .$$

Pushing negation all the way to the literals, and rewriting as a clause yields the consequent:

$$(b \notin [l_b, u_b] \vee c \notin [l_c, u_c]) \vee a \in [l_b + l_c, u_b + u_c]) \ .$$

□

Lemma 7 *For integer constants $l_a, l_b, m, n \in \mathbb{Z}$ and integer variables $a, b \in \mathbb{Z}$*

$$\frac{\mathit{NoOverlap}(a, l_a, b, l_b)}{(a \notin [m, n + l_b - 1] \vee b \notin [n, m + l_a - 1])} \quad (\text{NO(m,n)})$$

Proof: The semantics of the premise NoOverlap(a, l_a, b, l_b) is

$$\phi = (a \geq b + l_b \vee b \geq a + l_a) \ ,$$

which can be written as $\phi = (b + l_b \leq a \vee a \leq b - l_a)$. Combining ϕ with the tautology $\neg(m \leq a \leq n + l_b - 1) \vee (m \leq a \leq n + l_b - 1])$ yields

$$\neg(m \leq a \leq n + l_b - 1) \vee b + l_b \leq n + l_b - 1 \vee m \leq b - l_a) \ .$$

After simple rewriting this becomes

$$\neg(m \leq a \leq n + l_b - 1) \vee b \leq n - 1 \vee b \geq m + l_a) \ ,$$

and then $\neg(m \leq a \leq n + l_b - 1]) \vee \neg(b > n - 1 \wedge b < m + l_a)$. Rewriting with literal notation gives us the consequent of the rule:

$$(a \notin [m, n + l_b - 1] \vee b \notin [n, m + l_a - 1]) \ .$$

□

2.2 Completeness of inference rules

For CSP-Analyze-Conflict to be complete, it is mandatory to be able to create an explanation for any type of constraint, including for explicit table relations. When an unsatisfiability proof is printed, the proof derives each explanation from a constraint using an inference rule. For the proof system to be complete, we must be able to match an inference rule for any implication. Like in the proof, CSP-Analyze-Conflict may also use inference rules to derive explanations.

In order to discuss inference rules and their consequent explanations, we need to focus on valid implications first. The following definition summarizes what we refer to as a propagator. These are weak requirement that are satisfied by any propagator that we are aware of.

Definition 1 (Constraint propagator assumption) *Every constraint $r \in \mathcal{C}$ is associated with a propagator by the same name. We assume that the constraint propagator $r(v_1, \ldots, v_k)$ meets the following criteria:*

- *it recognizes when a complete assignment to $v_1, \ldots, v_k$ violates the constraint $r(v_1, \ldots, v_k)$;*
- *if it modifies a domain from D_i to D'_i then $D'_i \subseteq D_i$ and the values in $D_i \setminus D'_i$ are not supported by r (in other words, for every assignment $v_i \leftarrow x_i$ such that $x_i \in D_i \setminus D'_i$, there is no assignment to $v_1, \ldots, v_k$ from their respective domains that satisfies the constraint r); and*
- *it terminates in finite time.*

Note that this assumption leaves the freedom to choose the strength of the propagator. Some known strategies are *arc consistency*, which is the strongest, *bounds consistency*, and pure *backtrack-search* algorithm, which is the weakest.

Recall that CSP-Analyze-Conflict requires an explanation of inferred literals, and hence we need to show that such an explanation can always be given. For this purpose we define a generic inference rule and then prove that it is applicable to all constraints.

Definition 2 (The generic inference rule) *Consider an arbitrary constraint $r(v_1, \ldots, v_k)$ which when combined with literals $v_1 \in D_1, \ldots, v_k \in D_k$ implies literal $v_i \in D'_i$. The generic, parameterized, implication rule $G_{r,i}(A_1, \ldots, A_k)$, as defined below, generates an explanation for this implication*

$$\frac{r}{(v_1 \in A_1 \vee \cdots \vee v_k \in A_k)} \quad (G_{r,i}(A_1, \ldots, A_k)) \ ,$$

where for each $j = 1, \ldots, i-1, i+1, \ldots k$, we define $A_j = \overline{D_j}$ and where $A_i = D'_i \cup \overline{D_i}$

We now show that Definition 2 gives a valid inference rule for an arbitrary constraint.

Lemma 8 (Existence of an inference rule) *Let u be a node in an implication graph that represents a literal that was implied by an arbitrary constraint r. There exists an inference rule that its premise is r and its consequent is an explanation of u.*

Proof: For any constraint $r(v_1, \ldots, v_k)$, Definition 2 provides the generic propagation rule $G_{r,i}(A_1, \ldots, A_k)$, which generates the following explanation for an implication of $v_i \in D'_i$:

$$c = (v_1 \in \overline{D_1} \vee \cdots \vee v_{i-1} \in \overline{D_{i-1}} \vee v_i \in (D'_i \cup \overline{D_i}) \vee v_{i+1} \in \overline{D_{i+1}} \vee \cdots \vee v_k \in \overline{D_k}) \ ,$$

which can also be written as:

$$c = (v_1 \notin D_1 \vee \cdots \vee v_k \in D_k \vee v_i \in D'_i) \ .$$

To prove that the clause c is indeed an explanation, recall, we have to show that, for $l_1 = (v_1 \in D_1), \ldots, l_k = (v_k \in D_k)$:

1. $r \rightarrow c$,
2. $(l_1 \wedge \cdots \wedge l_k \wedge c) \rightarrow l$.

To prove the first property we will show that every possible assignment $x_1, \ldots, x_k$ satisfies $\neg c(x_1, \ldots, x_k) \rightarrow \neg r(x_1, \ldots, x_k)$. The clause c is falsified only if all its literals are *false*, i.e., $v_1 \in D_1 \wedge \cdots \wedge v_k \in D_k \wedge v_i \notin D'_i$. In other words, all values are taken from their respective domains D_j and $x_i \in D_i \setminus D'_i$, which is one of the values removed by the propagator r. According to Definition 1, r will not remove x_i from D_i if $x_1 \in D_1 \wedge \cdots \wedge x_k \in D_k$ satisfies $r(x_1, \ldots, x_k)$, implying that $\neg r(x_1, \ldots, x_k)$. The conclusion is that $\neg r(x_1, \ldots, x_k)$ whenever $\neg c(x_1, \ldots, x_k)$.

For the second property we analyze all assignments $v_1 = x_1 \wedge \cdots v_k = x_k$ which satisfy the left side of the implication. Such an assignment has to satisfy $x_1 \in D_1 \wedge \cdots \wedge x_k \in D_k$, in which case all literals $v_j \notin D_j$ of c are falsified except for $v_i \in D'_i$, i.e., l. This shows that an assignment that satisfies the left-hand side of the implication also satisfies its right-hand side $\square$

3 Algorithms

3.1 CSP-Analyze-Conflict algorithm, the proof

First, we repeat the algorithm as it appears in [1]:

```
1: function CSP-Analyze-Conflict
2:     cl:= Explain(conflict-node);
3:     pred:= Predecessors(conflict-node);
4:     front:= Relvant (pred, cl);
5:     while (¬Stop-criterion-met(front)) do
6:         curr-node:= Last-node(front);
7:         front:= front\curr-node;
8:         expl:= Explain(curr-node);
9:         cl:= Resolve(cl, expl, var(lit(curr-node)));
10:        pred:= Predecessors(curr-node);
11:        front:= Distinct (Relvant (front ∪ pred, cl));
12:    end while
13:    add-clause-to-database(cl);
14:    return clause-asserting-level(cl);
15: end function
```

To prove the correctness of CSP-Analyze-Conflict, we first prove the loop invariant that is mentioned briefly in [1].

Lemma 9 (CSP-Analyze-Conflict **loop invariant)** *The following invariant holds just before line 6: The clause cl is inconsistent with the labels in* front*.*

Proof: Consider the conflicting constraint p, i.e., the constraint that labels the edges leading to *conflict-node*. On the first iteration, the literals of *pred* conflict the constraint p . Because cl is the explanation clause of p it also conflicts on the conjunction of *pred*. Because Relvant at line 4 keeps the nodes which relevant to cl, i.e., share the same variables, then the labels of *front* also conflict cl.

Assuming that the invariant holds on iteration $n-1$, we will show that the invariant holds at the n-th iteration, if executed. For this proof we denote by cl and cl' the values of cl before and after the update at line 9, respectively, and similarly we use *front* and *front*'. Except for the literal with $v = var(lit($*curr-node*$))$, according to the definition of *expl*, all other literals are falsified by *pred*. cl' has three types of literals: the first is from *expl* and do not refer to v, the second is from cl and do not refer to v , and the third refers to v and contains a conjunction of literals from the former two sources.

1. The literals from cl are falsified by the conjunction of $lits($*front*$)$ and hence also by the conjunction of $lits($*front*$\cup$*pred*$)$.

2. For an implication going from literals $\{l_1, \ldots, l_k\} = lits($*pred*$)$ to literal $l = lit($*curr-node*$)$, recall, the second requirement from an explanation c is $(l_1 \wedge \cdots \wedge l_k \wedge c) \rightarrow l$. In order for c to be able to imply a literal l, as required, all literals of cl must be falsified by $l_1 \wedge \cdots \wedge l_k$, except for the literal that constrains v. Because $l_1 \wedge \cdots \wedge l_k$ is the conjunction of

$lits(pred)$, then literals cl that do not constrain v, which are of the second type of literals, are also falsified by the conjunction of $lits(front \cup pred)$.

3. We want to prove that the literal of cl' labeled with v is falsified by the $lits(front')$. First we introduce the following naming convention: ω_v is the disjunction of all literals of clause ω which affect v, and similarly N_v is the conjunction $\bigwedge_{l \in lits(N) | var(l)=v} l$. We also reuse the notation used in the definition of explanations where l is the literal of *curr-node* and $l_1, \ldots, l_k$ are the literals of *pred*.

   The claim to be proven can now be reformulated as $(cl'_v \wedge lits(front')) = false$. To prove this, first consider the invariant from iteration $n-1$ which gives $(cl_v \wedge l) = false$, or as a clause:

   $$(\neg cl_v \vee \neg l) \ .$$

   The definition of explanations states that $(l_1 \wedge \cdots \wedge l_k \wedge expl) \rightarrow l$, or as a clause:

   $$(\neg l_1 \vee \cdots \vee \neg l_k \vee \neg expl \vee l) \ .$$

   Resolving the last two clauses with v as pivot results with:

   $$(\neg l_1 \vee \cdots \vee \neg l_k \vee \neg expl \vee \neg cl_v) \ ,$$

   which is the same as

   $$\neg((l_1 \wedge \cdots \wedge l_k \wedge) expl \wedge cl_v)) \ ,$$

   which shows that $expl \wedge cl_v$ is falsified by $lits(pred)$, and hence $cl'_v = expl_v \wedge cl_v$ is also falsified by $lits(front \cup pred)$.

   The call to RELVANT at line 11 removes only nodes which are irrelevant to the falsification of cl', hence, it is left to show that *front*, produced by DISTINCT at line 11, also falsifies cl'_v. For this part of the proof we need some formalism first. We depict $n_1, \ldots, n_q$ as the nodes of $lits(front \cup pred)$ for which $var(lit(n_j)) = v$ according to the propagation order that created them. Using this formalism we can say that DISTINCT will remove $n_1, \ldots, n_{q-1}$ and keep only n_q, we need to show that this node is sufficient to falsify cl'_v. Because how propagators are allowed to work, literals of succeeding nodes must refer to decreasing domain sized, and hence $n_q \rightarrow n_j$ for each $j = 1, \ldots, q$. This result means that all nodes $n_1, \ldots, n_{q-1}$ are redundant, such that if $(lit(n_1) \wedge \cdots \wedge lit(n_q) \wedge cl'_v) = false$ then it is also true that $(lit(n_q) \wedge cl'_v) = false$. Because cl'_v was falsified by the literals of $n_1, \ldots, n_q$, as shown above, then it must be falsified by n_q which is entered into *front*'.

As we seen, all literals of the new clause cl' are falsified by the literals of *front*'. This shows that iteration n also satisfies the loop invariant. □

Theorem 1 (CSP-ANALYZE-CONFLICT **correctness)** *The algorithm of* CSP-ANALYZE-CONFLICT *is sound and complete:*

- *soundness – the returned clause cl is derived from the CSP ϕ such that $\phi \rightarrow cl$, and it is either falsified at the target decision level or is an asserting clause that will cause propagation at clause-asserting-level(cl).*
- *completeness – the algorithm terminates and returns cl.*

Also, CSP-Analyze-Conflict *returns a target level which forces backtrack of at least one level.*

Note that this theorem allows the resulting *cl* to be *false* even at the target of backtrack. This situation happens often in practice and can be easily eliminated by a small modification to the algorithm, as described in Section 4.

Proof: Soundness – The requirement of $\phi \rightarrow cl$, is trivial because *cl* is created through a set of proven inference rules from premises and through applications of the resolution rule. The requirement of *cl* to be an asserting clause is met through the Stop-criterion-met which is true in two cases:

- *front* nodes have no predecessors, i.e., they refer to initial domains. According to the loop invariant (Lemma 9) all literals of *cl* are falsified by the conjunction of the literals of *front*, meaning that *cl* is *false* even with the initial domains.
- The latest node of *front* is the only one from decision level d and the next latest node is from decision level $d' < d$, such that function clause-asserting-level(cl) will return this d'. Backtracking will undo all decisions and implications done from $d' + 1$ and later a target state at which, we need to show, *cl* will be an asserting clause. At the target state all nodes of *front*, except for the latest one, are still present, which according to Lemma 9 must falsify all literals except for the latest one. This means that *cl* is either falsified or an asserting clause at the end of decision level d'.

Completeness – First we show that the preconditions of all auxiliary functions are fulfilled allowing them to successfully terminate. Assuming that CSP-Analyze-Conflict is passed a conflict-node then the first Predecessors must succeed. The first Explain must succeed because it is possible to generate an explanation for the conflicting constraint (Lemma 8). Inside the loop ¬Stop-criterion-met makes sure that *curr-node* has predecessors making predecessors succeed. Like with the first Explain the one in the loop must also find an explanation. All other used algorithms do not have special assumptions or preconditions.

The algorithm must terminate because each iteration removes the latest node, and inserts earlier nodes of *pred* instead of it, coupled with the fact that the implication graph is a DAG, each iteration is guaranteed to have *curr-node* which is earlier than at the previous iteration. Because the implication graph is finite then the number of iteration must be finite.

Backtrack level – as shown above, the target level is d' which is smaller than d which itself the current decision level, or earlier. This means that the caller will be instructed to backtrack to at least one decision level back. □

4 Enhancements and optimizations

In this section we describe several enhancements to the definitions and to the algorithms that may either improve performance or minimize proof size. At the present time we do not have exact measurements of the effectiveness of these modifications. Nevertheless these enhancements have interesting properties, making them worth mentioning.

4.1 CSP-Analyze-Conflict node rejuvenation

A problem that the proof of CSP-Analyze-Conflict (Theorem 1) showed is that a conflict clause may be conflicting immediately after backtrack, i.e., with no new decision. This problem is solved in PCS by the following, trivial, modification to the algorithm.

At lines 4 and 11 Distinct is called. After this call we add a call to a new function Rejuvenate(cl,*front*), which finds the earliest nodes that still falsify cl. This modification preserves the invariant of the loop because it moves only to such earlier nodes that still falsify cl. The proof of Theorem 1 with the amended algorithm is mostly unchanged since it relies on the invariant, which is not affected by the modification.

This rejuvenating can have a positive impact on conflict clause and proof sizes as it allows CSP-Analyze-Conflict to ignore all implications done between the original node and the rejuvenated node. However, it may also have a negative effect since by ignoring an implication we may lose a good candidate for clause resolution that would erase the literal from cl and reach a UIP. Instead, by ignoring a good candidate for resolution, we will have to apply resolution many times before a UIP is reached.

4.2 Augmented explanations

Let $(l_1, l), \ldots, (l_n, l)$ be the incoming edges of a node u such that $lit(u) = l$. If c is an explanation clause of u, recall, then $(l_1 \wedge \cdots \wedge l_n \wedge c) \rightarrow l$. We now propose to weaken this requirement.

Definition 3 (augmented explanations) *Let u be a node in the implication graph such that $lit(u) = l$. Let $(l_1, l) \ldots (l_n, l)$ be the incoming edges of u, all of which are labeled with a constraint r. Let l' be the literal in the clause cl just before the resolution step in* CSP-Analyze-Conflict*, such that $var(l) = var(l')$. A signed clause c' is an* augmented explanation clause *of a node u and a clause cl if it satisfies:*

1. $r \rightarrow c'$,
2. $(l_1 \wedge \cdots \wedge l_n \wedge c') \rightarrow \neg l'$.

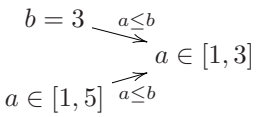
This definition lets us find c' which is not an explanation clause according to the original definition, but which is sufficient for our purpose, since while it

is still implied by the original constraint r, it also holds that

$$\text{Resolve}(cl, c', var(l)) \rightarrow \text{Resolve}(cl, c, var(l)) \ .$$

In other words, we derive a stronger resolvent. This may lead to shorter proofs down the line.

Example 1 *The literals $l_1 = (b = 3)$, $l_2 = (a \in [1, 5])$ together with the constraint $r = (a \leq b)$ imply $l = (a \in [1, 3])$. This implication is depicted in the following small diagram:*



The only valid explanation clause is derived using $LE(3)$:

$$c = (a \in (-\infty, 3] \vee b \in [4, \infty)) \ .$$

Indeed $(l_1 \wedge l_2 \wedge c) \rightarrow l$.

Now suppose that $cl = (a = 5 \vee c = 1)$ and hence $l' = (a = 5)$. We need to find c' such that $(l_1 \wedge l_2 \wedge c') \rightarrow \neg l'$. Although c is an explanation clause, we can get a stronger such clause with $LE(4)$:

$$c' = (a \in (-\infty, 4] \vee b \in [5, \infty)) \ ,$$

which is sufficient. Resolution of cl with c' results in $(b \in [5, \infty) \vee c = 1)$ whereas resolving with c would result in the weaker clause $(b \in [4, \infty) \vee c = 1)$.

What happens in this example is that c' does not have to eliminate the value 4 from the domain of a because it is allowed by $\neg l'$. As a result the other literal, referring to b, becomes stronger. □

4.3 Lookahead explanations

The weaker definition of *augmented explanations* gives us more freedom to choose a clause from a bigger set of possible clauses. Some freedom is also present in regular explanation clauses, especially when explaining a conflicting constraint. For example, the constraint $a \leq b$ is conflicting for domains $D_a = [10, 20]$ and $D_b = [0, 8]$, for which both $c = (a \in (-\infty, 8] \vee b \in [9, \infty))$ and $c = (a \in (-\infty, 9] \vee b \in [10, \infty))$ are valid explanations.

We have discovered that using this freedom wisely can considerably shorten both proof size and run-time. If *curr-node* is associated with variable v, we simply prefer an explanation which has a stronger literal associated with the rightmost node of *predecessors* which is not associated with v. The effect of this preference is twofold:

1. At the next iteration of CSP-Analyze-Conflict, there is a bigger chance that this literal l' will be a pivot than any other literal of the explanation clause. Making this literal smaller may strengthen the next augmented explanation because it will weaken the right-hand side of the implication in the requirement of *augmented explanations*:

$$(l_1 \wedge \cdots \wedge l_n \wedge c') \rightarrow \neg l' \ ,$$

   which will give more freedom for constructing c'.

2. By producing a stronger literal, there is a bigger chance that Rejuvenate will rejuvenate the node of $var(l')$ to a previous decision level, saving the need for some resolutions and, eventually, facilitate the creation of a smaller conflict clause.

References

[1] Michael Veksler and Ofer Strichman. A proof-producing CSP solver. 2010. Accepted to the AAAI'10 conference.