Efficient SAT-based Proof Search in Intuitionistic Propositional Logic

Camillo Fiorentini(✉)

Department of Computer Science, Università degli Studi di Milano, Milan, Italy

Abstract. We present an efficient proof search procedure for Intuitionistic Propositional Logic which involves the use of an incremental SAT-solver. Basically, it is obtained by adding a restart operation to the system `intuit` by Claessen and Rosén, thus we call our implementation `intuitR`. We gain some remarkable advantages: derivations have a simple structure; countermodels are in general small; using a standard benchmarks suite, we outperform `intuit` and other state-of-the-art provers.

1 Introduction

The `intuit` theorem prover by Claessen and Rosén [2] implements an efficient decision procedure for Intuitionistic Propositional Logic (IPL) based on a Satisfiability Modulo Theories (SMT) approach. Given an input formula α, the clausification module of `intuit` computes a sequent $\sigma = R, X \Rightarrow g$ equivalent to α with respect to IPL-validity, where R, X and g have a special form: R is a set of clauses, X is a set of implications $(a \to b) \to c$, with a, b, c atoms, g is an atom. The decision procedure at the core of `intuit` searches for a Kripke model $\mathcal{K}$ such that at its root all the formulas in R and X are forced and g is not forced; we call $\mathcal{K}$ a countermodel for σ, since it witnesses the non-validity of σ in IPL. The search is performed via a proper variant of the DPLL($\mathcal{T}$) procedure [12], whose top-level loop exploits an incremental SAT-solver. This leads to a highly performant decision strategy; actually, on the basis of a standard benchmarks suite, `intuit` outperforms two of the state-of-the-art provers for IPL, namely `fCube` [5] and `intHistGC` [11]. At first sight, the `intuit` decision procedure seems to be far away from the traditional techniques for deciding IPL validity; on the other hand, the in-depth investigation presented in [10] unveils a close and surprising connection between the `intuit` approach based on SMT and the known proof-theoretic methods. The crucial point is that the main loop of the decision procedure mimics a standard root-first proof search strategy for the sequent calculus $\mathrm{LJT}_{\mathtt{SAT}}$ [10] (see Fig. 7), a variant of Dyckhoff's calculus LJT [3]. In [10] the `intuit` decision procedure is re-formulated so that, given a sequent σ, it outputs either a derivation of σ in $\mathrm{LJT}_{\mathtt{SAT}}$ or a countermodel for σ.

Here we continue this investigation to better take advantage of the interplay between the SMT perspective and proof-theoretic methods. At first, we have enhanced the Haskell `intuit` code[1] by implementing the derivation/countermodel

[1] Available at `https://github.com/koengit/intuit`.

A. Platzer and G. Sutcliffe (Eds.): CADE 2021, LNAI 12699, pp. 217–233, 2021.
https://doi.org/10.1007/978-3-030-79876-5_13

extraction procedures discussed in [10]. We experimented some unexpected and weird phenomena: derivations are often convoluted and contain applications of the cut rule which cannot be trivially eliminated; countermodels in general contain lots of redundancies. To overcome these issues, we have redesigned the decision procedure. Differently from `intuit`, in the main loop we keep all the worlds of the countermodel under construction. Whenever the generation of a new world fails, the current model is emptied and the computation restarts with a new iteration of the main loop. We call the obtained prover `intuitR` (`intuit` with Restart). We gain some remarkable advantages. Firstly, the proof search procedure has a plain and intuitive presentation, consisting of two nested loops (see the flowchart in Fig. 3). Secondly, derivations have a linear structure, formalized by the calculus $C^{\rightarrow}$ in Fig. 1; basically, a derivation in $C^{\rightarrow}$ is a cut-free derivation in $\mathrm{LJT}_{\mathtt{SAT}}$ having only one branch. Thirdly, the countermodels obtained by `intuitR` are in general smaller than the ones obtained by `intuit`, since restarts cross out redundant worlds. We have replicated the experiments in [2] (1200 benchmarks): as reported in the table in Fig. 9 and in the scatter plot in Fig. 11, `intuitR` has better performances than `intuit`. The `intuitR` implementation and other additional material (e.g., the omitted proofs, a detailed report on experiments) can be downloaded at `https://github.com/cfiorentini/intuitR`.

2 Preliminary Notions

Formulas, denoted by lowercase Greek letters, are built from an infinite set of propositional variables V, the constant $\bot$ and the connectives $\wedge$, $\vee$, $\rightarrow$; the formula $\alpha \leftrightarrow \beta$ stands for $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$. Elements of the set $V \cup \{\bot\}$ are called *atoms* and are denoted by lowercase Roman letters, uppercase Greek letters denote sets of formulas. A *(classical) interpretation* M is a subset of V, identifying the propositional variables assigned to true. By $M \models \alpha$ we mean that α is true in M; moreover, $M \models \Gamma$ iff $M \models \alpha$ for every $\alpha \in \Gamma$. We write $\Gamma \vdash_{\mathrm{c}} \alpha$ iff, for every interpretation M, $M \models \Gamma$ implies $M \models \alpha$. A formula α is CPL-valid (valid in Classical Propositional Logic) iff $\emptyset \vdash_{\mathrm{c}} \alpha$.

A (rooted) Kripke model for IPL (Intuitionistic Propositional Logic) is a quadruple $\langle W, \leq, r, \vartheta \rangle$ where W is a finite and non-empty set (the set of *worlds*), $\leq$ is a reflexive and transitive binary relation over W, the world r (the *root* of $\mathcal{K}$) is the minimum of W w.r.t. $\leq$, and $\vartheta : W \mapsto 2^V$ (the *valuation* function) is a map obeying the persistence condition: for every pair of worlds w_1 and w_2 of $\mathcal{K}$, $w_1 \leq w_2$ implies $\vartheta(w_1) \subseteq \vartheta(w_2)$. The valuation ϑ is extended into a *forcing* relation between worlds and formulas as follows:

$w \Vdash p$ iff $p \in \vartheta(w)$, $\forall p \in V$ $\quad$ $w \nVdash \bot$ $\quad$ $w \Vdash \alpha \wedge \beta$ iff $w \Vdash \alpha$ and $w \Vdash \beta$
$w \Vdash \alpha \vee \beta$ iff $w \Vdash \alpha$ or $w \Vdash \beta$ $\quad$ $w \Vdash \alpha \rightarrow \beta$ iff $\forall w' \geq w$, $w' \Vdash \alpha$ implies $w' \Vdash \beta$.

By $w \Vdash \Gamma$ we mean that $w \Vdash \alpha$ for every $\alpha \in \Gamma$. A formula α is IPL-valid iff, for every Kripke model $\mathcal{K}$ we have $r \Vdash \alpha$ (here and below r designates the root of $\mathcal{K}$). Thus, if there exists a model $\mathcal{K}$ such that $r \nVdash \alpha$, then α is not IPL-valid; we call $\mathcal{K}$ a *countermodel* for α, written $\mathcal{K} \not\models \alpha$, and we say that α is *countersatisfiable*. We write $\Gamma \vdash_{\mathrm{i}} \delta$ iff, for every model $\mathcal{K}$, $r \Vdash \Gamma$ implies $r \Vdash \delta$; thus,

$$\frac{R \vdash_{\mathrm{c}} g}{R, X \Rightarrow g}\ \mathrm{cpl}_0 \qquad \frac{R,\, A \vdash_{\mathrm{c}} b \qquad R,\, \varphi,\, X \Rightarrow g}{R,\, X \Rightarrow g}\ \mathrm{cpl}_1 \qquad \begin{array}{l}(a \to b) \to c \in X \\ A \subseteq V \\ \varphi \;=\; \bigwedge(A \setminus \{a\}) \to c\end{array}$$

Fig. 1. The sequent calculus $C^{\to}$; $R, X \Rightarrow g$ is an r-sequent.

α is IPL-valid iff $\emptyset \vdash_{\mathrm{i}} \alpha$. Let σ be a sequent of the form $\Gamma \Rightarrow \delta$; σ is IPL-valid iff $\Gamma \vdash_{\mathrm{i}} \delta$. By $\mathcal{K} \not\models \sigma$ we mean that $r \Vdash \Gamma$ and $r \nVdash \delta$. Note that such a model $\mathcal{K}$ witnesses that σ is not IPL-valid; we say that $\mathcal{K}$ is a *countermodel* for σ and that σ is *counter-satisfiable*.

Clausification We review the main concepts about the clausification procedure described in [2]. *Flat clauses* φ and *implication clauses* λ are defined as

$$\begin{array}{ll}\varphi \;:=\; \bigwedge A_1 \to \bigvee A_2 \mid \bigvee A_2 & \emptyset \subset A_k \subseteq V \cup \{\bot\}, \text{ for } k \in \{1,2\} \\ \lambda \;:=\; (a \to b) \to c & a \in V,\ \{b, c\} \subseteq V \cup \{\bot\}\end{array}$$

where $\bigwedge A_1$ and $\bigvee A_2$ denote the conjunction and the disjunction of the atoms in A_1 and A_2 respectively ($\bigwedge\{a\} = \bigvee\{a\} = a$). Henceforth, $\bigwedge \emptyset \to \bigvee A_2$ must be read as $\bigvee A_2$; moreover, R, R_1, ... denote sets of flat clauses; X, X_1, ... sets of implication clauses; A, A_1, ... sets of atoms. The `intuit` procedure relies on the following property (see Lemma 2 in [10]):

Lemma 1. *For every set of flat clauses R and every atom g, $R \vdash_{\mathrm{i}} g$ iff $R \vdash_{\mathrm{c}} g$.*

In the decision procedure, flat clauses are actively used only in classical reasoning. A pair (R, X) is $\to$*-closed* iff, for every $(a \to b) \to c \in X$, $b \to c \in R$. An *r-sequent* (reduced sequent) is a sequent $\Gamma \Rightarrow g$ where g is an atom, $\Gamma = R \cup X$ and (R, X) is $\to$-closed. Given a formula α, the clausification procedure yields a triple (R, X, g) such that $R, X \Rightarrow g$ is an r-sequent and:

(1) $\vdash_{\mathrm{i}} \alpha$ iff $R, X \vdash_{\mathrm{i}} g$; (2) $\mathcal{K} \not\models R, X \Rightarrow g$ implies $\mathcal{K} \not\models \alpha$, for every $\mathcal{K}$. [2]

Thus, IPL-validity of formulas can be reduced to IPL-validity of r-sequents.

3 The Calculus $C^{\to}$

The sequent calculus $C^{\to}$ consists of the rules cpl_0 and cpl_1 from Fig. 1. Rule cpl_0 (axiom rule) can only be applied if the condition $R \vdash_{\mathrm{c}} g$ holds, rule cpl_1 requires that $R, A \vdash_{\mathrm{c}} b$ holds. In rule cpl_1, $(a \to b) \to c$ is the *main formula* and A the *local assumptions*; note that A is any set of propositional variables (not necessarily containing a). Derivations are defined as usual (see e.g. [14]);

[2] In [2] the clausification procedure outputs a triple (R, X, g) satisfying (1) and (2); the $\to$-closure of (R, X) is performed at the beginning of the decision procedure (for every $(a \to b) \to c \in X$, the clause $b \to c$ is added to R).

$$\dfrac{\dfrac{\dfrac{}{R_0,\, A_0 \vdash_{\mathrm{c}} b_0} \qquad \dfrac{\dfrac{}{R_1,\, A_1 \vdash_{\mathrm{c}} b_1} \qquad \dfrac{\dfrac{}{R_m \vdash_{\mathrm{c}} g}}{R_m,\, X \Rightarrow g}}{R_2,\, X \Rightarrow g}\,\lambda_{m-1}}{\vdots}\,\lambda_1}{R_1,\, X \Rightarrow g}\,\lambda_0}{R_0,\, X \Rightarrow g}$$

$$\lambda_k = (a_k \to b_k) \to c_k \in X, \qquad \varphi_k = \bigwedge(A_k \setminus \{a_k\}) \to c_k, \qquad R_{k+1} = R_k \cup \{\varphi_k\}$$

Fig. 2. Derivation of $R_0,\, X \Rightarrow g$ in $C^{\to}$ $(0 \le k \le m-1)$.

by $\vdash_{C^{\to}} \sigma$ we mean that there exists a derivation of the r-sequent σ in $C^{\to}$. In showing derivations, we leave out rule names and we display the main formulas of cpl_1 applications. Soundness of rule cpl_1 relies on the following property:

(a) If $R, A \vdash_{\mathrm{c}} b$, then $R, (a \to b) \to c \vdash_{\mathrm{i}} \varphi$, where $\varphi = \bigwedge(A \setminus \{a\}) \to c$.

Indeed, let $R, A \vdash_{\mathrm{c}} b$. By Lemma 1 $R, A \vdash_{\mathrm{i}} b$, thus $R, A \setminus \{a\} \vdash_{\mathrm{i}} a \to b$. It follows that $R, (a \to b) \to c, A \setminus \{a\} \vdash_{\mathrm{i}} c$, hence $R, (a \to b) \to c \vdash_{\mathrm{i}} \varphi$. By Lemma 1 and (a), the soundness of $C^{\to}$ follows:

Proposition 1. $\vdash_{C^{\to}} R, X \Rightarrow g$ *implies* $R, X \vdash_{\mathrm{i}} g$.

A derivation of $\sigma_0 = R_0, X \Rightarrow g$ has the plain form shown in Fig. 2: it only contains the branch of sequents $\sigma_k = R_k, X \Rightarrow g$ where the sets R_k are increasing. Nevertheless, the design of a root-first proof search strategy for $C^{\to}$ is not obvious. Let σ_0 be the r-sequent to be proved; we try to bottom-up build the derivation in Fig. 2 by running a loop where, at each iteration $k \ge 0$, we search for a derivation of σ_k. It is convenient to firstly check whether $R_k \vdash_{\mathrm{c}} g$ so that, by applying rule cpl_0, we immediately get a derivation of σ_k. If this is not the case, we should pick an implication λ_k from X and guess a proper set of local assumptions A_k in order to bottom-up apply rule cpl_1.

$$\dfrac{R_k,\, b_k \vdash_{\mathrm{c}} b_k \qquad R_k,\, X \Rightarrow g}{R_k,\, X \Rightarrow g}\,\lambda_k$$

$\lambda_k = (a_k \to b_k) \to c_k \in X,\ b_k \to c_k \in R_k$
$A_k = \{b_k\},\ \varphi_k = b_k \to c_k,\ R_{k+1} = R_k$

If we followed a blind choice, the procedure would be highly inefficient; for instance, the application of rule cpl_1 shown on the left triggers a non-terminating loop. Instead, we pursue this strategy: we search for a countermodel for σ_k; if we succeed, then $R_k, X \nvdash_{\mathrm{i}} g$ and, being $R_0 \subseteq R_k$, we conclude that $R_0, X \nvdash_{\mathrm{i}} g$ and proof search ends. Otherwise, from the failure we learn the proper λ_k and A_k to be used in the application of rule cpl_1; in next iteration, proof search restarts with the sequent σ_{k+1}, where R_{k+1} is obtained by adding the learned clause φ_k to R_k. To check classical provability, we exploit a SAT-solver; each time the solver is invoked, the set R_k has increased, thus it is advantageous to use an incremental SAT-solver.

Countermodels Henceforth we define Kripke models by specifying the interpretations associated with its worlds. Let W be a finite set of interpretations with minimum M_0, namely: $M_0 \subseteq M$ for every $M \in W$. By $\mathcal{K}(W)$ we denote the Kripke model $\langle W, \leq, M_0, \vartheta \rangle$ where $\leq$ coincides with the subset relation $\subseteq$ and ϑ is the identity map, thus $M \Vdash p$ (in $\mathcal{K}(W)$) iff $p \in M$. We introduce the following *realizability relation* $\rhd_W$ between W and implication clauses:

$$M \rhd_W (a \to b) \to c \quad \text{iff} \quad \begin{array}{l}(a \in M) \text{ or } (b \in M) \text{ or } (c \in M) \text{ or} \\ (\ \exists M' \in W \text{ s.t. } M \subset M' \text{ and } a \in M' \text{ and } b \notin M'\ ).\end{array}$$

By $M \rhd_W X$ we mean that $M \rhd_W \lambda$ for every $\lambda \in X$. Countermodels of r-sequents can be characterized as follows:

Proposition 2. *Let $\sigma = R, X \Rightarrow g$ be an r-sequent and let W be a finite set of interpretations with minimum M_0. Then, $\mathcal{K}(W) \not\models \sigma$ iff:*
(i) $g \notin M_0$; (ii) for every $M \in W$, $M \models R$ and $M \rhd_W X$.

4 The Procedure `proveR`

The strategy outlined in Sec. 3 is implemented by the decision procedure `proveR` (prove with Restart) defined by the flowchart in Fig. 3. The call `proveR(`R`,`X`,`g`)` returns `Valid` if the r-sequent $\sigma = R, X \Rightarrow g$ is IPL-valid, `CountSat` otherwise; by tracing the computation, we can build a $C^{\rightarrow}$-derivation of σ in the former case, a countermodel for σ in the latter. We exploit a single incremental SAT-solver s: clauses can be added to s but not removed; by $\mathrm{R}(s)$ we denote the set of clauses stored in s. The solver s has associated a set of propositional variables $\mathrm{U}(s)$ (the universe of s); we assume that every clause φ supplied to s is built over $\mathrm{U}(s)$ (namely, every variable occurring in φ belongs to $\mathrm{U}(s)$). The SAT-solver is required to support the following operations:

- `newSolver()`
  Create a new SAT-solver.
- `addClause(`s`,` φ`)` // s is a SAT-solver, φ a flat clause built over $\mathrm{U}(s)$
  Add the clause φ to s.
- `satProve(`s`,` A`,` g`)` // s is a SAT-solver, $A \subseteq \mathrm{U}(s)$, $g \in \mathrm{U}(s) \cup \{\bot\}$
  Call s to decide whether $\mathrm{R}(s), A \vdash_{\mathrm{c}} g$ (A is a set of local assumptions). The solver outputs one of the following answers:
  - Yes(A'): thus, $A' \subseteq A$ and $\mathrm{R}(s), A' \vdash_{\mathrm{c}} g$;
  - No(M): thus, $A \subseteq M \subseteq \mathrm{U}(s)$ and $M \models \mathrm{R}(s)$ and $g \notin M$.

  In the former case it follows that $\mathrm{R}(s), A \vdash_{\mathrm{c}} g$, in the latter $\mathrm{R}(s), A \nvdash_{\mathrm{c}} g$.

The procedure `newSolver(`R`)`, defined using the primitive operations, creates a new SAT-solver containing all the clauses in R. The computation of the call `proveR(`R`,` X`,` g`)` consists of the following steps:

(S0) A new SAT-solver s storing all the clauses in R is created.
(S1) A loop starts *(main loop)* with empty W.

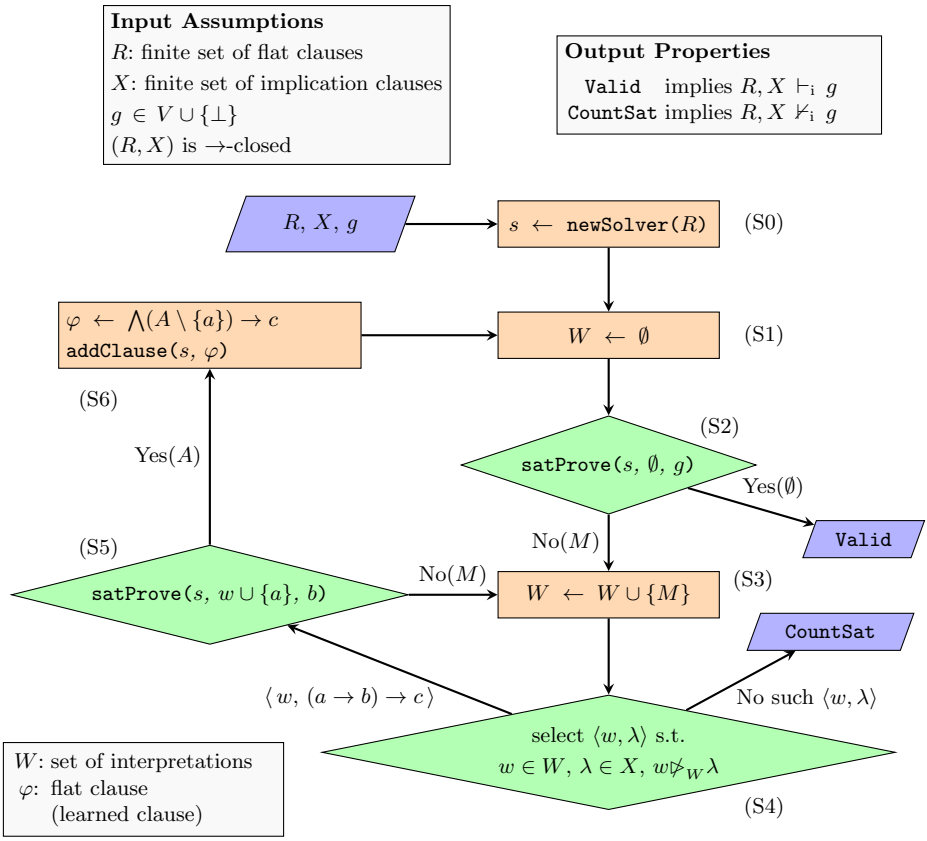

Fig. 3. Computation of `proveR(`R, X, g`)`.

(S2) The SAT-solver s is called to check whether $\mathrm{R}(s) \vdash_{\mathrm{c}} g$. If the answer is Yes($\emptyset$), the computation stops yielding `Valid`. Otherwise, the output is No(M) and the computation continues at Step (S3).

(S3) A loop starts *(inner loop)* by adding the interpretation M computed at Step (S2) to the set W (thus, $W = \{M\}$).

(S4) We have to select a pair $\langle w, \lambda \rangle$ such that $w \in W$, $\lambda \in X$ and $w \not\Vdash_W \lambda$. If such a pair does not exist, the procedure ends with output `CountSat`. Otherwise, the computation continues at Step (S5).

(S5) Let $\langle w, (a \to b) \to c \rangle$ be the pair selected at Step (S4). The SAT-solver s is called to check whether $\mathrm{R}(s), w, a \vdash_{\mathrm{c}} b$. If the result is No($M$), then a new iteration of the inner loop is performed where M is added to W. Otherwise, the answer is Yes(A) and the computation continues at Step (S6); we call A the *learned assumptions* and $\langle w, (a \to b) \to c \rangle$ the *learned pair*.

(S6) The clause φ (the *learned clause*) is added to the solver s and the computation restarts from Step (S1) with a new iteration of the main loop.

Note that during the computation no new variables are created, thus $\mathrm{U}(s)$ can be defined as the set of propositional variables occurring in $R \cup X \cup \{g\}$. We show that the call `proveR(R,X,g)` is correct, namely: if R, X, g match the Input Assumptions, then the Output Properties hold (see Fig. 3). We stipulate that:

- R_k denotes the set $\mathrm{R}(s)$ at the beginning of iteration k of the main loop;
- φ_k denotes the clause learned at iteration k of the main loop;
- $W_{k,j}$ denotes the set W at iteration k of the main loop and just after Step (S3) of iteration j of the inner loop.
- $\sim_c$ denotes classical equivalence, namely: $\alpha \sim_c \beta$ iff $\vdash_{\mathrm{c}} \alpha \leftrightarrow \beta$.

We prove some properties about the computation of `proveR(R, X, g)`.

(P1) Let $k, j \geq 0$ be such that $W_{k,j}$ is defined. Then:
  (i) The set $W_{k,j}$ has a minimum element M_0 and $g \notin M_0$.
  (ii) For every $M \in W_{k,j}$, $M \models R_k$.
  (iii) If $W_{k,j+1}$ is defined, then $W_{k,j} \subset W_{k,j+1}$.

(P2) For every $0 \leq h < k$ such that φ_k is defined, $\varphi_h \not\sim_c \varphi_k$.

Let $W_{k,0} = \{M\}$; one can easily check that, setting $M_0 = M$, (i) holds. Point (ii) follows by the fact that each M in $W_{k,j}$ comes from an answer $\mathrm{No}(M)$, thus $M \models R_k$. Let $W_{k,j+1}$ be defined and let $W_{k,j+1} = W_{k,j} \cup \{M\}$, with M computed at step (S5); there is $w \in W_{k,j}$ and $\lambda = (a \to b) \to c \in X$ such that $w \not\rhd_{W_{k,j}} \lambda$ and $w \cup \{a\} \subseteq M$ and $b \notin M$. We cannot have $M \in W_{k,j}$, otherwise, since $w \subseteq M$ and $a \in M$ and $b \notin M$, we would get $w \rhd_{W_{k,j}} \lambda$, a contradiction. Thus $M \notin W_{k,j}$, and this proves (iii).

Let $0 \leq h < k$ be such that φ_k is defined, let $\langle w_k, \lambda_k = (a_k \to b_k) \to c_k \rangle$ and A_k be the pair and the assumptions learned at iteration k respectively; note that $A_k \subseteq w_k \cup \{a_k\}$. Since $R_h \cup \{\varphi_h\} = R_{h+1} \subseteq R_k$, we have $\varphi_h \in R_k$; by (P1)(ii), it holds that $w_k \models R_k$, hence $w_k \models \varphi_h$. We show that $w_k \not\models \varphi_k$, and this proves (P2). Since $\langle w_k, \lambda_k \rangle$ has been selected at Step (S4), $c_k \notin w_k$; by the fact that $\varphi_k = \bigwedge(A_k \setminus \{a_k\}) \to c_k$ and $A_k \setminus \{a_k\} \subseteq w_k$, we conclude $w_k \not\models \varphi_k$.

Exploiting the above properties, we prove the correctness of `proveR`, also showing how to extract derivations and countermodels from computations.

Proposition 3. *The call* `proveR(R,X,g)` *is correct.*

Proof. We start by proving that the computation never diverges. By (P2), the learned clauses φ_k are pairwise not classically equivalent; since each φ_k is built over the finite set $\mathrm{U}(s)$, at most $2^{|\mathrm{U}(s)|}$ such clauses can be generated, and this proves the termination of the main loop. Since every interpretation M in W is a subset of $\mathrm{U}(s)$, by (P1)(iii) the termination of the inner loop follows.

Let $\sigma = R, X \Rightarrow g$. If `proveR(R,X,g)` returns `CountSat`, then the computation ends at Step (S4) since no pair $\langle w, \lambda \rangle$ can be selected. By (P1), the current set W satisfies the assumptions (i),(ii) of Prop. 2; accordingly, $\mathcal{K}(W)$ is a countermodel for σ, thus $R, X \nvdash_{\mathrm{i}} g$. If `proveR(R,X,g)` outputs `Valid`, then there exists $m \geq 0$ such that, at Step (S2) of iteration m of the main loop, the

SAT-solver yields Yes($\emptyset$), hence $R_m \vdash_{\mathrm{c}} g$. For every iteration k in $0 \dots m-1$ of the main loop, let $\langle w_k, \lambda_k = (a_k \to b_k) \to c_k \rangle$ be the learned pair and A_k the learned assumptions (thus, $R_k,\ A_k \vdash_{\mathrm{c}} b_k$). We can apply rule cpl_1 as follows:

$$\frac{R_k,\ A_k \vdash_{\mathrm{c}} b_k \qquad R_{k+1},\ X \Rightarrow g}{R_k,\ X \Rightarrow g}\ \lambda_k \qquad \begin{array}{l} \varphi_k \ = \ \bigwedge(A_k \setminus \{a_k\}) \to c_k \\ R_0 = R, \quad R_{k+1} = R_k \cup \{\varphi_k\} \end{array}$$

Accordingly, we can build the derivation of $R, X \Rightarrow g$ displayed in Fig. 2 and, by Prop. 1, we conclude $R, X \vdash_{\mathrm{i}} g$. □

As a corollary, we get the completeness of the calculus $C^{\to}$:

Proposition 4. *For every r-sequent $\sigma = R, X \Rightarrow g$, $\vdash_{C^{\to}} \sigma$ iff $R, X \vdash_{\mathrm{i}} g$.*

We give two examples of computations using formulas from the ILTP (Intuitionistic Logic Theorem Proving) library [13].

Example 1. Let χ be the first instance of problem class `SYJ201` from the ILTP library [13], where $\eta_{ij} = p_i \leftrightarrow p_j$ and $\gamma = \ p_1 \wedge p_2 \wedge p_3$:

$$\chi \ = \ ((\eta_{12} \to \gamma) \wedge (\eta_{23} \to \gamma) \wedge (\eta_{31} \to \gamma)) \ \to \ \gamma$$

The clausification of χ yields the triple $(R_0, X, \tilde{g})$, where X contains the implication clauses $\lambda_0, \dots, \lambda_5$ defined in Fig. 4 and R_0 the following 17 clauses (we mark by a tilde the fresh variables introduced during clausification): [3]

$$\begin{array}{l} \tilde{p}_0 \to \tilde{p}_4, \quad \tilde{p}_3 \to p_2, \quad \tilde{p}_3 \to p_3, \quad \tilde{p}_4 \to p_1, \quad \tilde{p}_4 \to \tilde{p}_3, \quad \tilde{p}_5 \to \tilde{p}_4, \quad \tilde{p}_8 \to \tilde{p}_4, \\ \tilde{p}_1 \wedge \tilde{p}_2 \to \tilde{p}_0, \qquad \tilde{p}_6 \wedge \tilde{p}_7 \to \tilde{p}_5, \qquad \tilde{p}_9 \wedge \tilde{p}_{10} \to \tilde{p}_8, \qquad p_1 \wedge p_2 \wedge p_3 \to \tilde{g}, \\ p_1 \to \tilde{p}_2, \qquad p_1 \to \tilde{p}_9, \qquad p_2 \to \tilde{p}_1, \qquad p_2 \to \tilde{p}_7, \qquad p_3 \to \tilde{p}_6, \qquad p_3 \to \tilde{p}_{10}. \end{array}$$

The trace of the computation of `proveR(`R_0,X,$\tilde{g}$`)` is shown in Fig. 4. Each row displays the validity tests performed by the SAT-solver and the computed answers. If the result is No(_), the last two columns show the worlds w_k in the current set W and, for each w_k, the list of λ such that $w_k \not\rhd_W \lambda$; the pair selected at Step (S4) is underlined. For instance, after call (0) we have $W = \{w_0\}$ and $w_0 \not\rhd_W \lambda_k$ for every $0 \le k \le 5$; the selected pair is $\langle w_0, \lambda_0 \rangle$. After call (1), the set W is updated by adding the world w_1 and $w_1 \not\rhd_W \lambda_3$, $w_1 \not\rhd_W \lambda_5$ and $w_0 \not\rhd_W \lambda_k$ for every $2 \le k \le 5$ (since $w_1 \in W$, we get $w_0 \rhd_W \lambda_0$); the selected pair is $\langle w_1, \lambda_3 \rangle$. Whenever the SAT-solver outputs Yes(A), we display the learned clause φ_k. The SAT-solver is invoked 15 times and there are 6 restarts. Fig. 4 also shows the derivation of $R_0, X \Rightarrow \tilde{g}$ extracted from the computation. ◊

Example 2. Let ψ be the second instance of problem class `SYJ207` from the ILTP library [13], where $\eta_{ij} = p_i \leftrightarrow p_j$ and $\gamma = p_1 \wedge p_2 \wedge p_3 \wedge p_4$:

$$\psi \ = ((\eta_{12} \to \gamma) \wedge (\eta_{23} \to \gamma) \wedge (\eta_{34} \to \gamma) \wedge (\eta_{41} \to \gamma)) \ \to \ (p_0 \vee \neg p_0 \vee \gamma)$$

[3] With `intuit`, the set R_0 consists of the 11 clauses in the first two rows; the remaining 6 clauses are added when the $\to$-closure of (R_0, X) is performed (see footnote 2).

$\lambda_0 = (p_3 \to p_2) \to \tilde{p}_7 \quad \lambda_1 = (p_3 \to p_1) \to \tilde{p}_9 \quad \lambda_2 = (p_2 \to p_3) \to \tilde{p}_6$
$\lambda_3 = (p_2 \to p_1) \to \tilde{p}_2 \quad \lambda_4 = (p_1 \to p_3) \to \tilde{p}_{10} \quad \lambda_5 = (p_1 \to p_2) \to \tilde{p}_1$
$w_0 = \emptyset \quad w_1 = \{p_3, \tilde{p}_6, \tilde{p}_{10}\} \quad w_2 = \{p_2, \tilde{p}_1, \tilde{p}_7, \tilde{p}_{10}\} \quad w_3 = \{p_3, \tilde{p}_2, \tilde{p}_6, \tilde{p}_{10}\}$
$w_4 = \{p_1, \tilde{p}_2, \tilde{p}_6, \tilde{p}_9\} \quad w_5 = \{\tilde{p}_1, \tilde{p}_7, \tilde{p}_9\} \quad w_6 = w_5 \cup \{p_2\} \quad w_7 = \{p_1, \tilde{p}_2, \tilde{p}_7, \tilde{p}_9\}$

	@SAT	Answer	W	λ s.t. $w \not\Vdash_W \lambda$
Start	(0) $R_0 \vdash_c^? \tilde{g}$	No(w_0)	$\underline{w_0}$	$\underline{\lambda_0}, \ldots, \lambda_5$
	(1) $R_0, w_0, p_3 \vdash_c^? p_2$	No(w_1)	$\underline{w_1}$	$\underline{\lambda_3}$, λ_5
			w_0	$\lambda_2, \ldots, \lambda_5$
	(2) $R_0, w_1, p_2 \vdash_c^? p_1$	Yes($\{p_2, \tilde{p}_6\}$)	$\varphi_0 = \tilde{p}_6 \to \tilde{p}_2$	
Rest 1	(3) $R_1 \vdash_c^? \tilde{g}$	No(w_2)	$\underline{w_2}$	$\underline{\lambda_1}$
	(4) $R_1, w_2, p_3 \vdash_c^? p_1$	Yes($\{p_3, \tilde{p}_1\}$)	$\varphi_1 = \tilde{p}_1 \to \tilde{p}_9$	
Rest 2	(5) $R_2 \vdash_c^? \tilde{g}$	No(w_3)	$\underline{w_3}$	$\underline{\lambda_5}$
	(6) $R_2, w_3, p_1 \vdash_c^? p_2$	Yes($\{p_1, \tilde{p}_{10}\}$)	$\varphi_2 = \tilde{p}_{10} \to \tilde{p}_1$	
Rest 3	(7) $R_3 \vdash_c^? \tilde{g}$	No(w_4)	$\underline{w_4}$	$\underline{\lambda_0}$
	(8) $R_3, w_4, p_3 \vdash_c^? p_2$	Yes($\{p_3\}$)	$\varphi_3 = \tilde{p}_7$	
Rest 4	(9) $R_4 \vdash_c^? \tilde{g}$	No(w_5)	$\underline{w_5}$	$\underline{\lambda_2}$, λ_3, λ_4
	(10) $R_4, w_5, p_2 \vdash_c^? p_3$	No(w_6)	$\underline{w_6}$	$\underline{\lambda_4}$
			w_5	λ_4
	(11) $R_4, w_6, p_1 \vdash_c^? p_3$	Yes($\{p_1, \tilde{p}_1\}$)	$\varphi_4 = \tilde{p}_1 \to \tilde{p}_{10}$	
Rest 5	(12) $R_5 \vdash_c^? \tilde{g}$	No(w_7)	$\underline{w_7}$	$\underline{\lambda_2}$
	(13) $R_5, w_7, p_2 \vdash_c^? p_3$	Yes($\{p_2\}$)	$\varphi_5 = \tilde{p}_6$	
Rest 6	(14) $R_6 \vdash_c^? \tilde{g}$	Yes($\emptyset$)	**Valid**	

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{R_0, p_2, \tilde{p}_6 \vdash_c p_1 \quad R_1, X \Rightarrow \tilde{g}}{R_0, X \Rightarrow \tilde{g}}\,\lambda_3 \quad R_1, p_3, \tilde{p}_1 \vdash_c p_1 \quad R_2, X \Rightarrow \tilde{g}}{}\,\lambda_1 \quad R_2, p_1, \tilde{p}_{10} \vdash_c p_2 \quad R_3, X \Rightarrow \tilde{g}}{}\,\lambda_5 \quad R_3, p_3 \vdash_c p_2 \quad R_4, X \Rightarrow \tilde{g}}{}\,\lambda_0 \quad R_4, p_1, \tilde{p}_1 \vdash_c p_3 \quad R_5, X \Rightarrow \tilde{g}}{}\,\lambda_4 \quad R_5, p_2 \vdash_c p_3 \quad R_6, X \Rightarrow \tilde{g}}{}\,\lambda_2 \quad R_6 \vdash_c \tilde{g}$$

Fig. 4. Computation of `proveR(`R_0,X,$\tilde{g}$`)`, see Ex. 1.

$\lambda_0 = (p_4 \to p_3) \to \tilde{p}_{11}$ $\lambda_1 = (p_4 \to p_1) \to \tilde{p}_{13}$ $\lambda_2 = (p_3 \to p_4) \to \tilde{p}_{10}$
$\lambda_3 = (p_3 \to p_2) \to \tilde{p}_8$ $\lambda_4 = (p_2 \to p_3) \to \tilde{p}_7$ $\lambda_5 = (p_2 \to p_1) \to \tilde{p}_2$
$\lambda_6 = (p_1 \to p_4) \to \tilde{p}_{14}$ $\lambda_7 = (p_1 \to p_2) \to \tilde{p}_1$ $\lambda_8 = (p_0 \to \bot) \to \tilde{g}$
$w_0 = \emptyset$ $w_1 = \{p_4, \tilde{p}_{10}, \tilde{p}_{14}\}$ $w_2 = \{p_3, \tilde{p}_7, \tilde{p}_{11}, \tilde{p}_{14}\}$ $w_3 = \{p_4, \tilde{p}_8, \tilde{p}_{10}, \tilde{p}_{14}\}$
$w_4 = w_3 \cup \{p_2, \tilde{p}_1\}$ $w_5 = w_4 \cup \{p_0, \tilde{g}\}$ $w_6 = \{p_1, \tilde{p}_2, \tilde{p}_8, \tilde{p}_{10}, \tilde{p}_{13}\}$
$w_7 = \{p_4, \tilde{p}_1, \tilde{p}_8, \tilde{p}_{10}, \tilde{p}_{14}\}$ $w_8 = w_7 \cup \{p_2\}$ $w_9 = w_7 \cup \{p_0, \tilde{g}\}$

	@SAT	Answer	W	λ s.t. $w \not\vdash_W \lambda$
Start	(0) $R_0 \vdash_c^? \tilde{g}$	No(w_0)	$\underline{w_0}$	$\underline{\lambda_0}, \dots, \lambda_8$
	(1) $R_0, w_0, p_4 \vdash_c^? p_3$	No(w_1)	$\underline{w_1}$	$\underline{\lambda_3}, \lambda_4, \lambda_5, \lambda_7, \lambda_8$
			w_0	$\lambda_2, \dots, \lambda_8$
	(2) $R_0, w_1, p_3 \vdash_c^? p_2$	Yes($\{p_3, \tilde{p}_{10}\}$)	$\varphi_0 = \tilde{p}_{10} \to \tilde{p}_8$	
Rest 1	(3) $R_1 \vdash_c^? \tilde{g}$	No(w_2)	$\underline{w_2}$	$\underline{\lambda_1}, \lambda_5, \lambda_7, \lambda_8$
	(4) $R_1, w_2, p_4 \vdash_c^? p_1$	Yes($\{p_4, \tilde{p}_{11}\}$)	$\varphi_1 = \tilde{p}_{11} \to \tilde{p}_{13}$	
Rest 2	(5) $R_2 \vdash_c^? \tilde{g}$	No(w_3)	$\underline{w_3}$	$\underline{\lambda_4}, \lambda_5, \lambda_7, \lambda_8$
	(6) $R_2, w_3, p_2 \vdash_c^? p_3$	No(w_4)	$\underline{w_4}$	$\underline{\lambda_8}$
			w_3	$\lambda_7,\ \lambda_8$
	(7) $R_2, w_4, p_0 \vdash_c^? \bot$	No(w_5)	w_5	$\emptyset$
			w_4	$\emptyset$
			$\underline{w_3}$	$\underline{\lambda_7}$
	(8) $R_2, w_3, p_1 \vdash_c^? p_2$	Yes($\{p_1, \tilde{p}_{14}\}$)	$\varphi_2 = \tilde{p}_{14} \to \tilde{p}_1$	
Rest 3	(9) $R_3 \vdash_c^? \tilde{g}$	No(w_6)	$\underline{w_6}$	$\underline{\lambda_0}, \lambda_4, \lambda_8$
	(10) $R_3, w_6, p_4 \vdash_c^? p_3$	Yes($\{p_4, \tilde{p}_{13}\}$)	$\varphi_3 = \tilde{p}_{13} \to \tilde{p}_{11}$	
Rest 4	(11) $R_4 \vdash_c^? \tilde{g}$	No(w_7)	$\underline{w_7}$	$\underline{\lambda_4}, \lambda_5, \lambda_6$
	(12) $R_4, w_7, p_2 \vdash_c^? p_3$	No(w_8)	$\underline{w_8}$	$\underline{\lambda_8}$
			w_7	λ_8
	(13) $R_4, w_8, p_0 \vdash_c^? \bot$	No(w_9)	w_9	$\emptyset$
	CountSat		w_8	$\emptyset$
			w_7	$\emptyset$

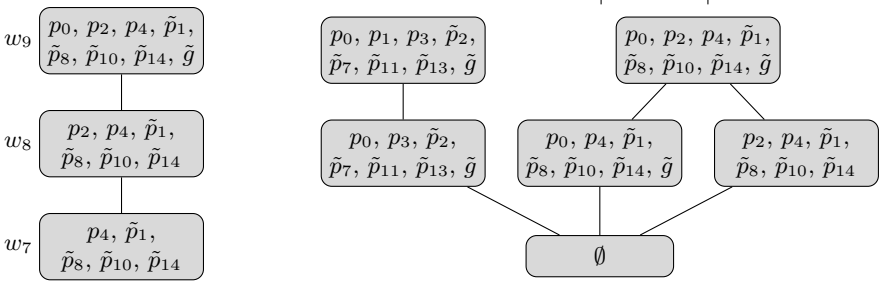

Fig. 5. Computation of `proveR(`R_0`,`X`,`$\tilde{g}$`)`, see Ex. 2.

```
1  procedure prove(R, X, g)
2  |  // Same Input Ass. and Output Prop. as for intuitR (Fig. 3)
3  |  s ← newSolver(R);   τ ← prAux(X, ∅, g)
4  |  if τ = Yes(∅) then return Valid else return CountSat
5  |  procedure prAux(X̃, Ã, q)
6  |  |  // Output: Yes(A) or No(M), where A ⊆ Ã and M ⊆ Ã
7  |  |  τ0 ← satProve(s, Ã, q)
8  |  |  if τ0 = Yes(A) then return Yes(A)
9  |  |  else  // τ0 = No(M)
10 |  |  |  for λ = (a → b) → c ∈ X s.t. a ∉ M and b ∉ M and c ∉ M do
11 |  |  |  |  τ1 ← prAux(X̃ \ {λ}, M ∪ {a}, b)
12 |  |  |  |  if τ1 = Yes(A) then
13 |  |  |  |  |  φ ← ⋀(A \ {a}) → c;    addClause(s, φ)
14 |  |  |  |  |  return prAux(X̃, Ã, q)
15 |  |  |  return No(M)
16 end
```

Fig. 6. The `prove` procedure of `intuit` [2,10].

We proceed as in Ex. 1. The clausification procedure yields $(R_0, X, \tilde{g})$, where X consists of the implication clauses $\lambda_0, \ldots, \lambda_8$ in Fig. 5 and the set R_0 contains the 24 flat clauses below:

$p_0 \to \tilde{g}, p_1 \to \tilde{p}_2,\ p_1 \to \tilde{p}_{13},\ p_2 \to \tilde{p}_1, p_2 \to \tilde{p}_8,\ p_3 \to \tilde{p}_7, p_3 \to \tilde{p}_{11},\ p_4 \to \tilde{p}_{10},\ p_4 \to \tilde{p}_{14},$
$\tilde{p}_0 \to \tilde{p}_5,\ \tilde{p}_3 \to p_3,\ \tilde{p}_3 \to p_4,\ \tilde{p}_4 \to p_2,\ \tilde{p}_4 \to \tilde{p}_3,\ \tilde{p}_5 \to p_1,\ \tilde{p}_5 \to \tilde{p}_4,\ \tilde{p}_6 \to \tilde{p}_5,\ \tilde{p}_9 \to \tilde{p}_5$
$\tilde{p}_1 \wedge \tilde{p}_2 \to \tilde{p}_0,\ \tilde{p}_7 \wedge \tilde{p}_8 \to \tilde{p}_6,\ \tilde{p}_{10} \wedge \tilde{p}_{11} \to \tilde{p}_9,\ \tilde{p}_{13} \wedge \tilde{p}_{14} \to \tilde{p}_{12},\ \tilde{p}_{12} \to \tilde{p}_5,\ \gamma \to \tilde{g}.$

The execution of `proveR(`R_0`,`X`,`$\tilde{g}$`)` (see Fig. 5) requires 14 calls to the SAT-solver and 4 restarts. After the last call we get $W = \{w_7, w_8, w_9\}$ and $w_k \rhd_W X$ for every $w_k \in W$, thus the computation ends yielding `CountSat`. The model $\mathcal{K}(W)$, depicted at the bottom left of the figure, is a countermodel for $R_0, X \Rightarrow \tilde{g}$ and for ψ (see Sec. 2). ◊

5 Related Work and Experimental Results

We compare the procedure `proveR` of `intuitR` with its `intuit` counterpart, namely the procedure `prove` defined in Fig. 6. Here we comply with the presentation in [10], equivalent to the original one in [2]. The recursive auxiliary function `prAux` plays the role of the main loop of `proveR` (but in `proveR` the set of atoms $\tilde{A}$ is not used); the loop inside `prAux` corresponds to the inner loop of `proveR`. [4] We point out some major differences. Firstly, in `prAux` the interpretations M computed by the SAT-solver are not collected; in the loop, only the interpretation M computed at line 8 is considered, thus at the beginning of each

[4] Actually `intuit` implements a variant of `prAux` where as much as possible clauses φ are added to the solver.

$$\frac{R \vdash_{\mathrm{c}} q}{R, X \Rightarrow q}\ \mathrm{cpl}_0 \qquad \frac{R_1, b \to c, X, A \Rightarrow b \qquad R_2, \varphi, X, (a \to b) \to c \Rightarrow q}{R_1, R_2, X, (a \to b) \to c \Rightarrow q}\ \mathrm{ljt}$$

$$\frac{R_1, X_1 \vdash_{\mathrm{i}} \varphi \qquad \varphi, R_2, X_2 \Rightarrow q}{R_1, R_2, X_1, X_2 \Rightarrow q}\ \mathrm{cut} \qquad \begin{array}{l} A \subseteq V,\ q \in V \cup \{\bot\} \\ \varphi\ =\ \bigwedge(A \setminus \{a\}) \to c \end{array}$$

Fig. 7. The calculus $\mathrm{LJT}_{\mathtt{SAT}}$.

iteration just the "local" conditions of the test $M \not\models_W \lambda$ are checked (line 10). Secondly, the call `satProve(`s`,` $w \cup \{a\}$`,` b`)` to the SAT-solver at Step (S5) is replaced by the recursive call `prAux(`$\tilde{X} \setminus \{\lambda\}$`,` $M \cup \{a\}$`,` b`)` at line 11; as a consequence, we cannot build derivations by applying rule cpl_1. As thoroughly discussed in [10], the calculus underlying `intuit` is the sequent calculus $\mathrm{LJT}_{\mathtt{SAT}}$ in Fig. 7, obtained from $C^{\rightarrow}$ by replacing the rule cpl_1 with the more general rule ljt and introducing a cut rule. Rule ljt can be seen as a generalization of Dyckhoff's implication-left rule from the calculus LJT (alias G4ip) [3,14]. We remark that a $C^{\rightarrow}$-derivation is isomorphic to a cut-free $\mathrm{LJT}_{\mathtt{SAT}}$-derivation where, in every application of rule ljt, the left-premise has a trivial proof (just apply rule cpl_0). In [10] it is shown how countermodels and $\mathrm{LJT}_{\mathtt{SAT}}$-derivations can be extracted from `prove` computations. In brief, countermodels are obtained by considering some of the interpretations coming from No(_) answers; countermodels are in general bigger than the ones built by `proveR`, where at each restart the model is emptied. As an example, let $\sigma_0 = R_0, X \Rightarrow \tilde{g}$ be defined as in Ex. 2; the computation of `prove(`R_0`,`X`,`$\tilde{g}$`)` requires 31 calls to the SAT-solver (24 No(_) answers) and the computed countermodel for σ_0 has 6 worlds (see Fig. 5); instead, `proveR(`R_0`,`X`,`$\tilde{g}$`)` requires 14 calls and the countermodel has 3 worlds. Derivation extraction presents some awkward aspects. The key insight is that, for every recursive call `prAux(`$\tilde{X}$`,`$\tilde{A}$`,`q`)` occurring in the computation of `prove(`R`,`X`,`g`)`, if `prAux(`$\tilde{X}$`,`$\tilde{A}$`,`q`)` returns Yes(A) (where $A \subseteq \tilde{A}$), then we can build an $\mathrm{LJT}_{\mathtt{SAT}}$-derivation of a sequent $R, R', A, \tilde{X} \Rightarrow q$, where R' contains some of the clauses added to the SAT-solver. The derivation is built either by applying the rule cpl_0 if `prAux` ends at line 8, or else by applying rule ljt, exploiting the derivations obtained by the recursive calls at lines 11 and 14. Accordingly, the main call `prove(`R`,`X`,`g`)` yields a derivation of $R, R', X \Rightarrow g$. The crucial point is that the redundant clauses φ in R' satisfy $R, X \vdash_{\mathrm{i}} \varphi$ (this ultimately follows by property (a) in Sec. 3), thus we can eliminate them by applying the cut rule.

Example 3. Let $\sigma_0 = R_0, X \Rightarrow \tilde{g}$ be defined as in Ex. 1; `prove(`R_0`,`X`,`$\tilde{g}$`)` yields the $\mathrm{LJT}_{\mathtt{SAT}}$-derivation $\mathcal{D}_0$ of $R_2, \varphi_4, X \Rightarrow \tilde{g}$ in Fig. 8. By applying the cut rule three times, we get an $\mathrm{LJT}_{\mathtt{SAT}}$-derivation of σ_0. We stress that the $C^{\rightarrow}$-derivation of σ_0 obtained with `intuitR` (see Fig. 4) has a simpler structure. □

Finally, we remark that the clauses φ computed in `prAux` do not enjoy property (P2) (Sec. 4); we have experimented cases where such clauses are even duplicated (e.g., with formulas from class SYJ205 of ILTP library).

$$\dfrac{\dfrac{\dfrac{}{R_0, p_2, \tilde{p}_6 \vdash_c p_1}}{R_0, X_{\{0,3\}}, p_2, \tilde{p}_6 \Rightarrow p_1} \quad \dfrac{\dfrac{\dfrac{}{R_1, p_1, \tilde{p}_{10} \vdash_c p_2}}{R_1, X_{\{0,5\}}, p_1, \tilde{p}_{10} \Rightarrow p_2} \quad \dfrac{\dfrac{}{R_2, p_3 \vdash_c p_2}}{R_2, X_{\{0\}}, p_3 \Rightarrow p_2}\,\lambda_5}{R_1, X_{\{0\}}, p_3 \Rightarrow p_2}}{\hat{\sigma} \;=\; R_0, X_{\{0\}}, p_3 \Rightarrow p_2}\,\lambda_3 = (p_2 \to p_1) \to \tilde{p}_2$$

$$\dfrac{\begin{array}{c}\vdots \\ \text{shown} \\ \text{above} \\ \vdots \\ \hat{\sigma}\end{array} \quad \dfrac{\dfrac{\dfrac{}{R_4, p_1, \tilde{p}_1 \vdash_c p_3}}{R_4, X_{\{2,4\}}, p_1, \tilde{p}_1 \Rightarrow p_3} \quad \dfrac{\dfrac{}{R_5, p_2 \vdash_c p_3}}{R_5, X_{\{2\}}, p_2 \Rightarrow p_3}\,\lambda_4 \quad \dfrac{\dfrac{}{R_6 \vdash_c \tilde{g}}}{R_6, X \Rightarrow \tilde{g}}}{R_4, X_{\{2\}}, p_2 \Rightarrow p_3} \quad \dfrac{\dfrac{\dfrac{}{R_3, p_3 \vdash_c p_1}}{R_3, X_{\{1\}}, p_3 \Rightarrow p_1} \quad \dfrac{}{R_5, X \Rightarrow \tilde{g}}\,\lambda_1 = (p_3 \to p_1) \to \tilde{p}_9}{R_3, \varphi_4, X \Rightarrow \tilde{g}}\,\lambda_2}{R_2, \varphi_4, X \Rightarrow \tilde{g}}\,\lambda_0 = (p_3 \to p_2) \to \tilde{p}_7$$

$\lambda_2 = (p_2 \to p_3) \to \tilde{p}_6 \quad \lambda_4 = (p_1 \to p_3) \to \tilde{p}_{10} \quad \lambda_5 = (p_1 \to p_2) \to \tilde{p}_1$
$\varphi_0 = \tilde{p}_6 \to \tilde{p}_2 \quad \varphi_1 = \tilde{p}_{10} \to \tilde{p}_1 \quad \varphi_2 = \tilde{p}_7 \quad \varphi_3 = \tilde{p}_9 \quad \varphi_4 = \tilde{p}_1 \to \tilde{p}_{10} \quad \varphi_5 = \tilde{p}_6$
$X_I = X \setminus \{\, \lambda_k \mid k \in I \,\} \quad R_{k+1} = R_k \cup \{\varphi_k\}$

Fig. 8. Derivation $\mathcal{D}_0$ of $R_2, \varphi_4, X \Rightarrow \tilde{g}$ in $\mathrm{LJT}_{\mathrm{SAT}}$ (see Ex. 3).

Experimental results We have implemented `intuitR` in Haskell on the top of `intuit`: we have replaced the function `prove` with `proveR` and added some features (e.g., trace of computations, construction of derivations/countermodels); as in `intuit`, we exploit the module MiniSat, a Haskell bundle of the MiniSat SAT-solver [4] (but in principle we can use any incremental SAT-solver). We compare `intuitR` with `intuit` and with two of the state-of-the-art provers for IPL by replicating the experiments in [2]. The first prover is `fCube` [5]; it is based on a standard tableaux calculus and exploits a variety of simplification rules [6] that can significantly reduce branching and backtracking. The second prover is `intHistGC` [11]; it relies on a sequent calculus with histories and uses dependency directed backtracking for global caching to restrict the search space; we run it with its best flags (`-b -c -c3`). All tests were conducted on a machine with an Intel i7-8700 CPU@3.20GHz and 16GB memory. We considered the benchmarks provided with `intuit` implementation, including the ILTP library, the `intHistGC` benchmarks and the API problems introduced by `intuit` developers. This amounts to a total of 1200 problems, 498 `Valid` and 702 `CountSat`; we used a 600s (seconds) timeout. Fig. 9 reports the more significant results, among which the classes where at least a prover fails and the classes where `intuitR` performs poorly. In all the tests, the time required by clausification is negligible. Even though no optimized data structure has been implemented, `intuitR` solve more problems than its competitors; in families SYJ201 (`Valid` formulas) and SYJ207 (`CountSat` formulas) `intuitR` outperforms its rivals, in all the other cases, except the families EC, negEC and portia, `intuitR` is comparable to the best prover (which is `intuit` in most cases). The most remarkable improvement with respect to `intuit` occurs with class SYJ212 (see Fig. 10), where `intuit` timings are fluc-

Class (number of problems)	`intuitR`	`intuit`	`fCube`	`intHistGC`
SYJ201(50)	50 (2.259)	50 (11.494)	50 (259.776)	50 (39.466)
SYJ202(38)	10* (49.265)	10* (50.658)	9* (176.984)	6* (324.673)
SYJ203(50)	50 (0.250)	50 (0.335)	50 (1.671)	50 (0.293)
SYJ204(50)	50 (0.442)	50 (0.477)	50 (0.972)	50 (0.203)
SYJ205(50)	50 (0.500)	50 (0.730)	50 (1.317)	50 (4.129)
SYJ206(50)	50 (0.303)	50 (0.348)	50 (0.759)	50 (0.112)
SYJ207(50)	50 (2.291)	50 (109.919)	50 (138.546)	50 (1014.476)
SYJ208(38)	38 (5.225)	38 (5.479)	29* (2.755)	38 (497.715)
SYJ209(50)	50 (0.226)	50 (0.278)	50 (1.690)	50 (0.254)
SYJ210(50)	50 (0.272)	50 (0.252)	50 (0.988)	50 (0.288)
SYJ211(50)	50 (0.462)	50 (1.251)	50 (1.073)	50 (63.686)
SYJ212(50)	50 (0.669)	42* (587.794)	50 (2.698)	50 (1.624)
EC(100)	100 (2.738)	100 (0.821)	100 (6.183)	100 (0.651)
negEC(100)	100 (3.614)	100 (1.116)	100 (13.733)	100 (5.807)
cross(4)	4 (0.100)	4 (0.097)	4 (3.417)	2* (0.005)
jm_cross(4)	4 (0.120)	4 (0.090)	4 (5.404)	3* (4.324)
jm_lift(3)	3 (0.170)	3 (0.133)	3 (6.847)	2* (0.028)
lift(3)	3 (0.119)	3 (0.102)	3 (6.494)	2* (0.012)
mapf(4)	4 (0.187)	4 (0.400)	4 (446.921)	3* (0.043)
portia(100)	100 (32.878)	100 (22.596)	100 (3255.818)	100 (3200.135)
negportia(100)	100 (7.956)	100 (8.309)	98* (3826.011)	100 (28.289)
negportiav2(100)	100 (8.081)	100 (8.411)	98* (1264.103)	100 (3212.293)
nishimura2(28)	28 (9.784)	28 (12.285)	27* (141.326)	28 (7.616)
Unsolved	28	36	43	38

Fig. 9. For each prover, we report the number of solved problems within 600s timeout and between brackets the total time in seconds required for the solved problems. The best prover is highlighted, a star reports that there are some unsolved problems.

tuating. To give a close comparison, let us consider the case $k = 25$; clausification produces 246 flat clauses and 100 implications clauses (176 atoms). Our `intuit` implementation requires 11214 calls to the SAT-solvers (10181 No(_)) and the computed countermodel has 1955 worlds. Instead, `intuitR` requires 45 calls to the SAT-solvers, 8 restarts and yields a countermodel consisting of 4 worlds; the set W contains 26 worlds before the first restart, one world before the remaining ones. With all the benchmarks the models generated during the computation are small (typically, big models occur before the first restart); however, differently from [7,8,9], we cannot guarantee that countermodels have minimum depth or minimum number of worlds. To complete the picture, the scatter plot in Fig. 11 compares `intuitR` and `intuit` on all the benchmarks.

k	intuitR	intuit
1 .. 24	< 0.01	< 0.1
25	0.007	0.691
26	0.007	25.064
27	0.007	0.020
28	0.008	0.083
29	0.009	8.412
30	0.008	-

k	intuitR	intuit
31	0.007	8.724
32	0.007	4.216
33	0.012	0.034
34	0.010	2.445
35	0.033	77.226
36	0.018	0.038
37	0.016	22.445
38	0.017	-

k	intuitR	intuit
39	0.020	0.404
40	0.016	0.838
41	0.027	-
42	0.020	0.785
43	0.036	435.324
44	0.026	0.098
45	0.070	0.639
46 .. 50	≤ 0.07	-

Problem k:
$(\ldots((\neg\neg p_1 \leftrightarrow p_2) \leftrightarrow p_3) \leftrightarrow \ldots \leftrightarrow p_k) \;\leftrightarrow\; (\ldots((p_1 \leftrightarrow p_2) \leftrightarrow p_3) \leftrightarrow \ldots \leftrightarrow p_k) \quad \neg\alpha \coloneqq \alpha \to \bot$

Fig. 10. Timings for problems $k = 1..50$ of SYJ212 (`CountSat`), - means timeout (600s).

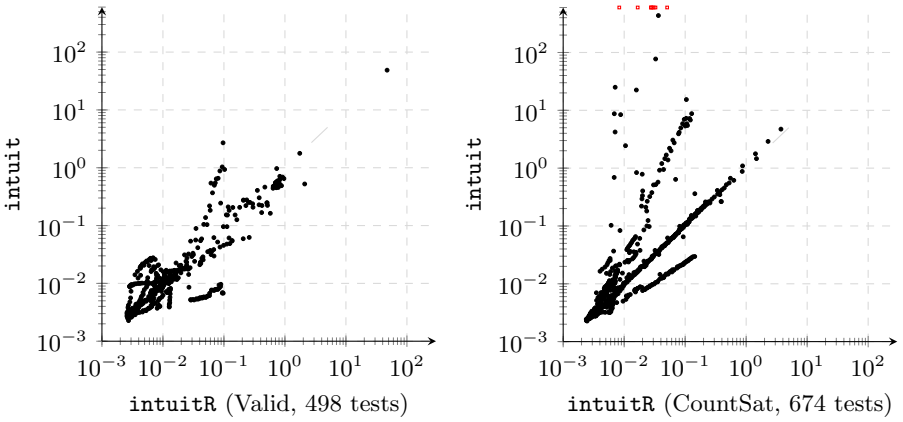

Fig. 11. Comparison between `intuitR` and `intuit` (1172 problems, the 28 problems where both provers run out of time have been omitted); time axis are logarithmic, the 8 red squares indicates that `intuit` has exceeded the timeout.

To conclude, we point out that `intuitR` can be extended to deal with some superintuitionistic logics [1]. For instance, let us consider the Göedel-Dummett logic GL, characterized by linear models; at any step of the computation of `proveR`, the model $\mathcal{K}(W)$ must be kept linear. Whenever the insertion of a new world to W breaks linearity, we follow a "restart with learning" strategy [12]: let $\gamma = (a \to b) \vee (b \to a)$ be the instance of the GL-axiom falsified at the root of $\mathcal{K}(W)$; we restart by taking γ as "learned axiom", so to avoid the repetition of the flaw. However, we cannot add γ to the SAT-solver, because γ is not a clause, but the clausification of γ, namely the clauses $\tilde{q}_1 \vee \tilde{q}_2$, $\tilde{q}_1 \wedge a \to b$, $\tilde{q}_2 \wedge b \to a$, where $\tilde{q}_1$ and $\tilde{q}_2$ are fresh atoms; despite the language of the SAT-solver must be extended, the process converges. The other generalizations suggested in [2] (modal logics, fragments of first-order logic) seem to be more challenging.

Acknowledgments. I am grateful to the reviewers for their valuable suggestions. This work has been funded by the INdAM-GNCS project 2020 "Estensioni del *Property-based Testing* di e con linguaggi di programmazione dichiarativa".

References

1. Chagrov, A.V., Zakharyaschev, M.: Modal Logic, Oxford logic guides, vol. 35. Oxford University Press (1997)
2. Claessen, K., Rosén, D.: SAT Modulo Intuitionistic Implications. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9450, pp. 622–637. Springer (2015), https://doi.org/10.1007/978-3-662-48899-7_43
3. Dyckhoff, R.: Contraction-free sequent calculi for intuitionistic logic. J. Symb. Log. **57**(3), 795–807 (1992), https://doi.org/10.2307/2275431
4. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers. Lecture Notes in Computer Science, vol. 2919, pp. 502–518. Springer (2003), https://doi.org/10.1007/978-3-540-24605-3_37
5. Ferrari, M., Fiorentini, C., Fiorino, G.: fCube: An Efficient Prover for Intuitionistic Propositional Logic. In: Fermüller, C.G., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6397, pp. 294–301. Springer (2010), https://doi.org/10.1007/978-3-642-16242-8_21
6. Ferrari, M., Fiorentini, C., Fiorino, G.: Simplification Rules for Intuitionistic Propositional Tableaux. ACM Trans. Comput. Log. **13**(2), 14:1–14:23 (2012), https://doi.org/10.1145/2159531.2159536
7. Ferrari, M., Fiorentini, C., Fiorino, G.: Contraction-Free Linear Depth Sequent Calculi for Intuitionistic Propositional Logic with the Subformula Property and Minimal Depth Counter-Models. J. Autom. Reason. **51**(2), 129–149 (2013), https://doi.org/10.1007/s10817-012-9252-7
8. Fiorentini, C.: An ASP Approach to Generate Minimal Countermodels in Intuitionistic Propositional Logic. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 1675–1681. ijcai.org (2019), https://doi.org/10.24963/ijcai.2019/232
9. Fiorentini, C., Ferrari, M.: Duality between unprovability and provability in forward refutation-search for intuitionistic propositional logic. ACM Trans. Comput. Log. **21**(3), 22:1–22:47 (2020), https://doi.org/10.1145/3372299
10. Fiorentini, C., Goré, R., Graham-Lengrand, S.: A Proof-Theoretic Perspective on SMT-Solving for Intuitionistic Propositional Logic. In: Cerrito, S., Popescu, A. (eds.) Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11714, pp. 111–129. Springer (2019), https://doi.org/10.1007/978-3-030-29026-9_7
11. Goré, R., Thomson, J., Wu, J.: A History-Based Theorem Prover for Intuitionistic Propositional Logic Using Global Caching: IntHistGC System Description. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8562, pp. 262–268. Springer (2014), https://doi.org/10.1007/978-3-319-08587-6_19

12. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J. ACM **53**(6), 937–977 (2006), https://doi.org/10.1145/1217856.1217859
13. Raths, T., Otten, J., Kreitz, C.: The ILTP problem library for intuitionistic logic. J. Autom. Reason. **38**(1-3), 261–271 (2007), https://doi.org/10.1007/s10817-006-9060-z
14. Troelstra, A.S., Schwichtenberg, H.: Basic proof theory, Second Edition, Cambridge tracts in theoretical computer science, vol. 43. Cambridge University Press (2000)