A proof-theoretic perspective on SMT-solving for intuitionistic propositional logic

Camillo Fiorentini\textsuperscript{1} and Rajeev Goré\textsuperscript{2} and Stéphane Graham-Lengrand\textsuperscript{3}

\textsuperscript{1} Department of Computer Science, University of Milan
\textsuperscript{2} Research School of Computer Science, Australian National University
\textsuperscript{3} SRI International

Abstract. Claessen and Rösen have recently presented an automated theorem prover, \textit{intuit}, for intuitionistic propositional logic which utilises a SAT-solver. We present a sequent calculus perspective of the theory underpinning \textit{intuit} by showing that it implements a generalisation of the implication-left rule from the sequent calculus LJT, also known as G4ip and popularised by Roy Dyckhoff.

1 Introduction

Intuitionistic propositional logic IPL is one of the most important “non-classical” logics due to its constructive reading of implication. There is a long history of automated reasoning techniques for deciding validity of IPL-formulae, but most of them are based on either sequent or tableaux calculi. One of the simplest procedures for IPL is root-first (a.k.a. backward or goal-directed) proof search in the LJT (a.k.a. G4ip) sequent calculus [2], as it is guaranteed to terminate without implementing loop-checking.

Claessen and Rösen [1] have recently presented an automated theorem prover, \textit{intuit}, for intuitionistic propositional logic, based on a Satisfiability-Modulo-Theories (SMT) approach. Their procedure also terminates without requiring any loop-detection machinery. As of 2015, the \textit{intuit} prover was the best performing IPL prover, at least when evaluated on about 1200 standard benchmarks [1], which include for instance the ILTP library [12].

The SMT approach embraced by \textit{intuit}, organised around the top-level loop of the DPLL(\textit{T}) procedure [11], and the proof-theoretic approach based on root-first proof search, appear as radically different methodologies; the potential connections between them was left as an open question. In this paper we reconcile the two approaches, formalising an explicit connection. In particular, we reformulate a variant of \textit{intuit} using (a suitable generalisation of) one of the rules of LJT. The procedure builds an explicit proof when the input formula is valid, and builds an explicit Kripke counter-model when the formula is not valid.

2 Syntax and Kripke Semantics of IPL

In this paper, formulae of IPL, denoted by lowercase Greek letters, are built from an infinite set of propositional variables $V$, the “falsum” constant $\bot$ and
the connectives $\land$, $\lor$, $\rightarrow$ with negation defined as $\neg \alpha := (\alpha \rightarrow \bot)$. We use $\text{Atm} := V \cup \{\bot\}$ for the set of “atoms”, denoted by lowercase Roman letters.

A rooted Kripke model for IPL is a quadruple $(W, \leq, r, \theta)$ where $W$ is a non-empty set of “worlds” containing $r$, and $\leq$ is a reflexive and transitive binary relation over $W$, and the root world $r$ is minimal wrt $\leq$, and $\theta : W \rightarrow 2^\text{Atm}$ is a “valuation” mapping each world to a set of propositional variables which obeys the “persistence” condition: $\forall w, v \in W$, if $w \leq v$ and $p \in \theta(w)$ then $p \in \theta(v)$. Given a Kripke model $(W, \leq, r, \theta)$, the valuation $\theta$ can be extended into a “forcing” relation between worlds and formulae as shown below:

$$
\begin{align*}
  w \models p & \text{ iff } p \in \theta(w) \\
  w \models \alpha \land \beta & \text{ iff } w \models \alpha \land w \models \beta \\
  w \models \alpha \lor \beta & \text{ iff } w \models \alpha \lor w \models \beta \\
  w \models \bot & \text{ never holds }
\end{align*}
$$

A formula $\alpha$ is IPL-valid if, for all Kripke models $(W, \leq, r, \theta)$, we have $r \models \alpha$. The problem of deciding whether a formula is IPL-valid is known to be PSPACE-complete [13]. For formula set or multiset $\Gamma$, we write $w \models \Gamma$ for $\forall \gamma \in \Gamma. w \models \gamma$.

A model $M$ for classical propositional logic (CPL), or “classical model”, is just a set of propositional variables (assigned true). By $M \models \alpha$ we mean that $\alpha$ is true in model $M$ (following the Boolean truth tables). We write $M \models \Gamma$ iff $\forall \gamma \in \Gamma. M \models \gamma$. We write $\Gamma \vdash_{\text{ipl}} \delta$ when the formula $\bigwedge \Gamma \rightarrow \delta$ is IPL-valid and $\Gamma \vdash_{\text{cpl}} \delta$ when it is CPL-valid, that is, when $M \models \gamma$ for all classical models $M$.

3 The Theorem Prover intuit

The intuitionist theorem prover is an intuitionistic prover built on top of a SAT-solver, following a Satisfiability-Modulo-Theories (SMT) approach. Despite the fact that SMT-solving works primarily in classical logic and with first-order theories, Claessen and Rosén [1] show this approach to be relevant to the problem of deciding IPL-validity. They use a variant of the SMT scheme known as DPLL($\mathcal{T}$) [11], where DPLL is the well-known procedure for SAT-solving and $\mathcal{T}$ is here the “theory of intuitionistic implications”.

The main loop of DPLL($\mathcal{T}$) can be seen as a particular case of Counter-Example Guided Abstraction Refinement (CEGAR), as we describe next, for the particular case of IPL. A formula $\alpha$ whose IPL-validity is to be determined is transformed into a set $R$ of classical “flat clauses”, a set $X$ of intuitionistic “implication clauses”, and an atomic formula $q$, such that $\vdash_{\text{ipl}} \alpha$ iff $R, X \vdash_{\text{ipl}} q$ (the definitions are in Section 3.1). The sequent $R \Rightarrow q$ constitutes an “abstraction” of the input formula $\alpha$. A SAT-solver tries to find a classical counter-model $M$ for it, in that $M \models R$ but $M \not\models q$. If no such counter-model exists then $\alpha$ is not only CPL-valid but also IPL-valid. Otherwise the SAT-solver returns such a counter-model $M$, although the existence of $M$ does not necessarily mean that $R, X \not\vdash_{\text{ipl}} q$, as the implication clauses $X$ have so far been ignored. Therefore another procedure “checks” model $M$, in that it tries to produce a new abstraction $R' \Rightarrow q$ of $\alpha$ that refines $R \Rightarrow q$ (technically, $R \subseteq R'$) and defeats model $M$, meaning that $M \not\models R'$ while still ensuring $\vdash_{\text{ipl}} \alpha$ iff $R', X \vdash_{\text{ipl}} q$. If it fails, then indeed $\alpha$ is not IPL-valid. If
procedure prove\((R, X, q)\)
\[
\begin{align*}
s &= \text{newSolver}() \\
\text{for } \varphi \in R &\text{ do } \text{addClause}(s, \varphi) \\
\text{for } (a \rightarrow b) &\rightarrow c \in X \text{ do } \text{addClause}(s, b \rightarrow c) \\
\text{return } \text{intuitProve}(s, X, \emptyset, q)
\end{align*}
\]
end

procedure intuitProve\((s, X, A, q)\)
\[
\begin{align*}
\text{loop } // \text{CEGAR LOOP} \\
\tau_0 &\leftarrow \text{satProve}(s, A, q) \\
\text{if } \tau_0 = \text{Yes}(A) &\text{ then return Yes}(A') \\
\text{else } // \tau_0 = \text{No}(M), \text{ with } M \text{ a putative counter-model} \\
\text{if } \text{intuitCheck}(s, X, M) &\text{ then return No}(M)
\end{align*}
\]
end

procedure intuitCheck\((s, X, M)\)
\[
\begin{align*}
\text{for } \iota = (a \rightarrow b) &\rightarrow c \in X \text{ such that } a \notin M \text{ and } b \notin M \text{ and } c \notin M \text{ do} \\
\tau_1 &\leftarrow \text{intuitProve}(s, X, M \cup \{a\}, b) // X_i = X \setminus \{c\} \\
\text{if } \tau_1 = \text{Yes}(A_1) &\text{ then} \\
\text{addClause}(s, \bigwedge (A_1 \setminus \{a\}) \rightarrow c) \\
\text{return } \text{False}
\end{align*}
\]
end
\[
\text{return } \text{True}
\]
end

Fig. 1. Main algorithms of intuit \[1\]

it does produce a refinement, then the procedure loops with \(R'\) instead of \(R\). Eventually, it either finds a counter-model for \(\alpha\), or exhausts the set of putative counter-models and conclude that \(\alpha\) is IPL-valid.

A key element of the approach is that \(R \vdash_{\text{cpl}} q\) iff \(R \vdash_{\text{ipl}} q\), as the clauses in \(R\) are “flat”. A twist of the approach, compared to the standard DPLL(\(T\)) loop, is that the procedure that checks model \(M\) has to solve a new IPL-validity problem (for a different \(R, X, q\)), so that it recursively calls a new DPLL(\(T\)) loop. In other words intuit implements a recursive version of DPLL(\(T\)), although a single SAT-solver is used, incrementally, for all recursive calls.

3.1 intuit in detail

Firstly, the formula \(\alpha\) is transformed into a formula \(\bigwedge \Gamma \rightarrow q\), where \(q\) is an atom and \(\Gamma\) is a set of flat clauses \(\varphi\) and implication clauses \(\iota\), where:
\[
\begin{align*}
\varphi &::= \bigwedge A_1 \rightarrow \bigvee A_2 & A_1 \cup A_2 \subseteq \text{Atm} \\
\iota &::= (a \rightarrow b) \rightarrow c & \{a, b, c\} \subseteq \text{Atm} \\
\bigwedge A_1 &\text{ is the conjunction of the atoms in } A_1 \\
\bigvee A_2 &\text{ is the disjunction of the atoms in } A_2 & \text{if } A_1 = \emptyset \text{ then } \varphi = \bigvee A_2
\end{align*}
\]
procedure intuitProve1(s, X, A, q)
    loop
        τ₀ ← satProve(s, A, q)
        if τ₀ = Yes(A') then return Yes(A')
        else // τ₀ = No(M)
            for ι = (a → b) → c ∈ X such that a /∈ M and b /∈ M and c /∈ M do
                τ₁ ← intuitProve1(s, Xι, M ∪ {a}, b) // Xι = X\{ι}
                if τ₁ = Yes(A₁) then
                    addClause(s, \(A_1 \setminus \{a\} \rightarrow c\)) // Xι = X\{ι'
                    go to line 2 (outer loop)
                end
            end
            return No(M)
        end
    end
end

Fig. 2. The function intuitProve1

A flat clause where A₁ is empty is simply a disjunction \(\bigvee A_2\) of atoms, and simply an atom a when A₂ is the singleton \(\{a\}\). Henceforth, we write \(R, R_1, R'\) etc. to denote sets of flat clauses; \(X, X_1, X'\) etc. for sets of implication clauses; \(A, A_1, A'\) etc. for sets of atoms; and \(X_ι, X_ι'\) for the sets \(X\setminus\{ι\}\) and \(X\setminus\{ι'\}\), respectively. A clausification procedure is presented in [1], similar to Tseitin’s [15], where clauses are created by naming subformulae with new propositional atoms. Technically, it transforms any IPL formula \(\alpha\) into a triple clausal \((\alpha) = (R,X,q)\) whose cumulative size is linear in the size of \(\alpha\) and that is equiprovable to \(\alpha\):

**Lemma 1.** For every \(\alpha\) with clausal \((\alpha) = (R,X,q)\), \(\vdash_{ipl} \alpha\) iff \(R, X \vdash_{ipl} q\) [1].

From now on, we focus on deciding \(R, X \vdash_{ipl} q\). The **intuit** algorithm [1], outlined in Fig. 1 with only slight modifications, consists of three procedures. It exploits a single SAT-solver \(s\) that is incremental: clauses can be added to \(s\) but not removed, and problems can be solved with varying atomic assumptions. So the clauses in \(s\) are “global clauses” which must hold at any point of proof-search. Technically, the SAT-solver has the following API, i.e., it supports the following operations, where R(s) denotes the set of clauses stored in \(s\):

- **newSolver()**: create a new SAT-solver;
- **addClause(s, \(\varphi\))**: add the flat clause \(\varphi\) to the SAT-solver \(s\);
- **satProve(s, A, q)**: call the SAT-solver \(s\) to decide whether \(R(s), A \vdash_{cpl} q\), where \(A\) is a local set of assumptions and \(q\) is an atom.

The call **satProve(s, A, q)** yields one of the following answers:

- Yes(A'): thus \(A' \subseteq A\) and \(R(s), A' \vdash_{cpl} q\);
- No(M): thus \(M\) is a classical model such that \(M \models R(s) \cup A\) and \(M \not\models q\).
procedure intuitPR(s, X, A, q)
    \(\tau_0 \leftarrow \text{satProve}(s, A, q)\)
    if \(\tau_0 = \text{Yes}(A')\) then return Yes(A')
    else // \(\tau_0 = \text{No}(M)\)
        for \(i = (a \rightarrow b) \rightarrow c \in X\) such that \(a \notin M\) and \(b \notin M\) and \(c \notin M\) do
            \(\tau_1 \leftarrow \text{intuitPR}(s, X_i, M \cup \{a\}, b)\) // \(X_i = X \setminus \{i\}\)
            if \(\tau_1 = \text{Yes}(A_1)\) then
                addClause(s, \(\bigwedge(A_1 \setminus \{a\}) \rightarrow c\))
                return intuitPR(s, X, A, q)
            end
        end
    end
    return No(M)
end

Fig. 3. The procedure intuitPR (recursive variant of intuitProve)

The main function prove\((R, X, q)\) of intuit yields:

Yes(\(\emptyset\)) if \(R, X \vdash_{\text{ipl}} q\);
No(M) if there is a Kripke model \(K = \langle W, \leq, r, \vartheta \rangle\) such that \(\vartheta(r) = M\) and \(r \not\models R \cup X\) and \(r \not\models q\); this implies \(R, X \not\models_{\text{ipl}} q\).

As sketched by Claessen and Rosén [1], if prove\((R, X, q)\) returns No(M), then one can actually build the mentioned model \(K\) by tracking the sets \(M'\) returned by intuitProve.

To reason about intuit, it is convenient to merge the functions intuitProve and intuitCheck into one recursive function. Firstly, we plug intuitCheck into intuitProve and obtain the function intuitProve1 in Fig. 2. Then, we remove the outer loop by replacing the “go to” statement at line 10 with a recursive call; we get the recursive procedure intuitPR in Fig. 3. We henceforth consider the intuit algorithm as implemented by the main function prove in Fig. 1, with function intuitProve at line 5 replaced by function intuitPR in Fig. 3.

4 Adapting the sequent calculus LJT to clausal forms

The sequent calculus LJT is a variant of Gentzen’s sequent calculus LJ for intuitionistic logic [6, 7] that was discovered many times, as outlined by Roy Dyckhoff [2]. Its rules are given in Fig. 4, where \(\Gamma \Rightarrow \alpha\) denotes a sequent whose antecedent \(\Gamma\) is a multiset of assumptions, and whose provability in LJT is denoted \(\vdash_{\text{LJT}} \Gamma \Rightarrow \alpha\). The key difference from LJ lies in the left introduction rules for implication. In order to introduce an implication \(\eta \rightarrow \gamma\) on the left, LJT offers four rules, depending on the form of \(\eta\), namely: either \(\eta = p\), with \(p \in V\); or \(\eta = \alpha \wedge \beta\); or \(\eta = \alpha \vee \beta\); or \(\eta = \alpha \rightarrow \beta\)\(^4\).

\(^4\) We follow Troelstra and Schwichtenberg [14] where the calculus is called G4ip; the original \((L \rightarrow \rightarrow)\) rule by Dyckhoff [2] has \(\Gamma, \beta \rightarrow \gamma \Rightarrow \alpha \rightarrow \beta\) as the left premise.
\[
\begin{align*}
\Gamma, p &\Rightarrow p \quad \text{Ax} & \quad \Gamma, \bot \Rightarrow \delta \quad L\bot & \quad \Gamma, \alpha, \beta \Rightarrow \delta \quad L\wedge & \quad \Gamma \Rightarrow \alpha & \quad \Gamma \Rightarrow \beta \quad R\wedge \\
\Gamma, \alpha \Rightarrow \delta & \quad \Gamma, \beta \Rightarrow \delta \quad L\lor & \quad \Gamma \Rightarrow \alpha_k & \quad \Gamma \Rightarrow \alpha_1 \lor \alpha_2 \quad R\lor_k \\
\Gamma, p, \beta \Rightarrow \delta & \quad L0\rightarrow & \quad \Gamma, \alpha \rightarrow (\beta \rightarrow \gamma) \Rightarrow \delta \quad L\land\rightarrow & \quad \Gamma, \alpha \rightarrow \gamma, \beta \rightarrow \gamma \Rightarrow \delta \quad L\lor\rightarrow \\
\Gamma, \beta \rightarrow \gamma, \alpha \Rightarrow \beta & \quad \Gamma, \gamma \Rightarrow \delta \quad L\rightarrow \rightarrow & \quad \quad p \in V, \ k \in \{1, 2\} \\
\Gamma, \beta \rightarrow \gamma, \alpha \Rightarrow \beta & \quad \Gamma, \gamma \Rightarrow \delta \quad L\rightarrow \rightarrow
\end{align*}
\]

Fig. 4. The calculus LJT (a.k.a. G4ip)

4.1 Root-first proof search, invertibility and recursivity

The purpose of replacing Gentzen’s left-introduction of implication by those four rules is to ensure that root-first proof search terminates. When given a sequent to prove, root-first proof search matches it against the conclusion of one of the rules, and recursively tries to prove each of its premises. In every rule of LJT, the multiset \( \Gamma_i, \alpha_i \) corresponding to the \( i \)th premise \( \Gamma_i \Rightarrow \alpha_i \) is strictly smaller than the multiset \( \Gamma, \alpha \) corresponding to the conclusion \( \Gamma \Rightarrow \alpha \), according to the well-founded multiset ordering based on formula size.\(^5\) Hence, the recursions of root-first proof search terminate. Note that keeping several copies of an assumption is never useful for proof search, so from now on, the antecedents of sequents will be considered sets. If, following the application of a rule, the recursive call that attempts to prove any one of its premises fails, then proof search attempts to apply another rule or another instance of the rule. Conceptually, a backtrack point was set when the original rule was applied. However, no such backtrack is needed if the rule is invertible, in the sense that, whenever the rule’s conclusion is provable (in LJT), so is each of its premises. In that case indeed, if any one of the premises is not provable, then neither is the conclusion, so there is no point in trying out another rule. In LJT, all rules are invertible except \( R\lor_k \) and \( (L\rightarrow\rightarrow) \), and therefore backtrack points need to be set only when applying those two rules. However \( (L\rightarrow\rightarrow) \) is right-invertible in that, if the conclusion is provable (in LJT), then so is the right premise (while the left premise may or may not be provable). This means that \((L\rightarrow\rightarrow)\) can be seen as a one-premise invertible rule, guarded by a side-condition:

\[
(\Gamma, \beta \rightarrow \gamma, \alpha \Rightarrow \beta) \quad \Gamma, \gamma \Rightarrow \delta \quad \Gamma, (\alpha \Rightarrow \beta) \rightarrow \gamma \Rightarrow \delta.
\]

When applying this rule, root-first proof search makes a first recursive call that checks that the side-condition holds and, if successful, makes a second one on the

\(^5\) This is the number of connectives, each conjunction counting for two [2].
premise. Because of invertibility, that second call is *tail-recursive*, as no backtrack point is needed: the output of proof search is the output of the recursive call. The next sections describe how these two recursive calls, for a generalisation of rule \((L \rightarrow \rightarrow)\) satisfying the same invertibility properties, correspond to the two recursive calls of \texttt{intuitPR} in Fig. 3 (lines 6 and 9). The (right-)invertibility of that generalised rule means that the recursive call on the (right) premise is tail-recursive and proof-search can thus be implemented by a while loop: namely the DPLL\((T)\) loop of \texttt{intuit}. To see this we specialise LJT to clausal forms.

4.2 LJT specialised to clausal forms

We now consider sequents in clausal form, namely sequents of the form \(R, X \Rightarrow q\). The only LJT rule manipulating \(X\) is then \((L \rightarrow \rightarrow)\), which becomes:

\[
\begin{align*}
R, b & \rightarrow c, X, a \Rightarrow b \\
R, X, c & \Rightarrow q \\
\end{align*}
\]

\(\text{\(\vdash\)} = (a \rightarrow b) \rightarrow c \quad \text{in} \quad X \quad X = X \setminus \{i\}\)

All other rules concern \(R\) and \(q\), or do not apply because the sequent to prove is already in clausal form. Hence, these rules can be replaced by the use of a SAT-solver, remembering that \(R \vdash q\) iff \(R \vdash_{ipl} q\).

To prove the left premise of \((L \rightarrow \rightarrow)\), atom \(a\) is added as an assumption, and will be taken into account by the SAT-solver. As \(a\) is not present in the right premise, it must be removed from the assumptions if the same SAT-solver is used for the right premise. Hence, it is useful to refine the notion of sequent in clausal form into sequents of the form \(R, X, A \Rightarrow q\), where \(R\) is the set of clauses present in the SAT solver, which can only grow bigger, and \(A\) is a set of atomic assumptions that can vary from one call to the next, relying on the SAT-solver’s API presented in Section 3.1. On such sequents, rule \((L \rightarrow \rightarrow)\) becomes:

\[
\begin{align*}
R, b & \rightarrow c, X, A, a \Rightarrow b \\
R, X, A, c & \Rightarrow q \\
\end{align*}
\]

\(\text{\(\vdash\)} = (a \rightarrow b) \rightarrow c \quad \text{in} \quad X \quad X = X \setminus \{i\}\)

We can start relating root-first proof search to \texttt{intuit} by relating the application of the above rule to a call to function \texttt{intuitPR}\((s, X, A, q)\), described in Fig. 3, considering \(R = R(s)\).

Indeed the left premise of the above rule is very similar to the first recursive call \texttt{intuitPR}\((s, X, M \cup \{a\}, b)\) on line 6 of Fig. 3, except that \(b \rightarrow c\) is added to \(R\) in the premise, and model \(M\) may differ from \(A\): According to its definition on line 4 of Fig. 3, \(M\) must satisfy (and therefore contain) all atoms in \(A\), but other atoms could be assigned true in \(M\) that are not in \(A\). Note however that \(R(s)\) does contain \(b \rightarrow c\) if \((a \rightarrow b) \rightarrow c\) is in \(X\): it has been added to \(s\) at the beginning of the computation of function \texttt{prove} from Fig. 1.

Likewise, the right premise of the above rule is very similar to the second recursive call \texttt{intuitPR}\((s, X, A, q)\) on line 9 of Fig. 3, except that the right premise contains the atomic formula \(c\) and the recursive call keeps the implication \((a \rightarrow b) \rightarrow c\).

Furthermore, the use of the SAT-solver in procedure \texttt{intuitPR} has side-effects: let \(R^0\) denote \(R(s)\) at the time when \texttt{intuitPR} is called, \(R^1\) denote \(R(s)\) at
the time of the first recursive call (line 6), and \( R^2 \) denote \( R(s) \) at the time of the second recursive call (line 9). We have \( R^0 \subseteq R^1 \subseteq R^1 \cup \{ \land (A_1 \setminus \{a\}) \rightarrow c \} \subseteq R^2 \), for a subset \( A_1 \) of \( M \cup \{a\} \) such that \( R^1, X, A_1 \vdash_{ipl} b \). This incremental use of the SAT-solver is not reflected in root-first proof search using rule \( (L \rightarrow \rightarrow) \). Hence, rule \( (L \rightarrow \rightarrow) \) has to be generalised to account for these differences.

4.3 A generalised version of \( (L \rightarrow \rightarrow) \)

The first generalisation consists in allowing the addition, in the left premise of \( (L \rightarrow \rightarrow) \), of extra atomic assumptions that were not assumptions in the conclusion. By allowing this, the rule can model either the extra atoms that are in \( M \) but not in \( A \) for the first recursive call of \texttt{intuitPR} (line 6 of Fig. 3), or the atoms \( A_1 \) that are returned by the call if successful. This ambiguity is systemic to the description of root-first proof search as presented in Section 4.1, which makes a double usage of sequent calculus rules. Consider a sequent calculus rule.

- Firstly, if the conclusion describes the arguments of a proof search, then the premises describe the arguments of the recursive calls; the rule describes the descent into the recursions.
- Secondly, if the recursive calls succeed, then a proof of each premise has been completed, either explicitly or implicitly, and a proof of the conclusion can be constructed; the rule describes the ascent back from the recursions.

It is useful to enhance proof search by considering, for each rule, a variant used for the first purpose and a variant used for the second purpose. Typically for the first purpose, it is convenient to integrate the weakening rule of the sequent calculus (say in LJ) to the axiom rule, and use context-sharing rules, as in Fig. 4. For the second purpose, it is useful to push all weakenings down towards the proof-tree root, using context-splitting rules. This strengthens the proved sequents by pruning the input sequent of all assumptions that were not used in the proof. This was described for instance in [8, 9], which also connects the said pruning to the notion of conflict analysis used in SAT and SMT-solving.

This is relevant for the connection between LJ and \texttt{intuit}, as a call to function \texttt{intuitPR}(s, X, A, q), if successful, precisely performs this pruning by outputing a subset \( A' \) of \( A \) that is sufficient for provability. In that spirit, we present the generalisation of \( (L \rightarrow \rightarrow) \) in a context-splitting style, emphasising what happens upon the completion of the recursive calls.

The addition of extra atomic assumptions in the left premise of \( (L \rightarrow \rightarrow) \) makes the premise easier to prove than with the original rule (when seen as a one-premise rule with a side-condition, the rule applies more often). The price to pay for this is that the new assumption that is learnt and that helps proving the right premise, has to be weakened from \( c \) to the flat clause \( \varphi = \land (A_1 \setminus \{a\}) \rightarrow c \), which in \texttt{intuitPR} is added to the SAT-solver on line 8 in Fig. 3. Note that clause \( \varphi \) is a consequence of the original problem. Weakening this newly available assumption in turn means that it no longer subsumes the original implication clause \( \iota \), which has to stay in the right premise. The resulting rule is rule \( (ljt) \) in Fig. 5.
When emulating SAT or SMT-solving, the sequent calculus has to deal with the effectful aspect of these solvers, which learn clauses that are consequences of the input problem, such as clause \( \varphi \) above.\(^6\) This effect was described in terms of memoisation of root-first proof search in [8, 9], and in terms of cuts in [4, 9]. Here again we use cuts to model the phenomenon: the added clauses can be deleted at the end of the proof search computation by applying rule (cut\(_{\text{ipl}}\)) of Fig. 5.

5 The Calculus LJT\(_{\text{SAT}}\)

We introduce the calculus LJT\(_{\text{SAT}}\) (LJT with SAT-solver) for sequents of the form \( R, X \Rightarrow q \), whose provability in LJT\(_{\text{SAT}}\) is denoted \( \vdash_{\text{LJT}_{\text{SAT}}} R, X \Rightarrow q \). It consists of the three rules of Fig. 5. Rule (cpl) has the premise judgment \( R \vdash_{\text{cpl}} q \) and the sequent \( R, X \Rightarrow q \) as conclusion, with \( X \) any set of implicational clauses. The rule can be applied if \( R \vdash_{\text{cpl}} q \) holds, as checked by a SAT-solver (hence LJT\(_{\text{SAT}}\)).

The rule (cut\(_{\text{ipl}}\)) is a cut rule having the judgment \( R, X \vdash_{\text{ipl}} \varphi \) as left premise. In the proof-search procedure, whenever we apply (cut\(_{\text{ipl}}\)), the left-premise is a judgment of the kind \( R_0, X_0 \vdash_{\text{ipl}} \varphi \), where the cut formula \( \varphi \) is a clause already stored in the SAT-solver, and we can take for granted that the assertion holds (we do not have to invoke an external prover to check it). The rule (ljt) is a sort of context-splitting generalisation of \((L \rightarrow \rightarrow)\) needed to capture the intuit procedure. We point out that the sets \( R_1 \) and \( R_2 \) may overlap, thus the common part \( R_1 \cap R_2 \) is kept in both the premises. The formula \( \bigwedge (A_1 \setminus \{a\}) \rightarrow c \) in the right premise is needed to guarantee the soundness of the rule, since \( A_1 \) is any set of atoms. To get completeness, we have to keep the main formula \((a \rightarrow b) \rightarrow c\) in the right premise; as a side-effect, the termination of proof-search is now trickier to prove. If the sets \( R_1 \) and \( R_2 \) coincide and \( A_1 = \{a\} \) (thus \( \bigwedge (A_1 \setminus \{a\}) \rightarrow c \) is the atom \( c \)), we get the instance

\[
\frac{R, b \rightarrow c, X, a \Rightarrow b}{R, X, (a \rightarrow b) \rightarrow c \Rightarrow q} \quad \text{(ljt)}
\]

The formula \((a \rightarrow b) \rightarrow c\) in the right premise is now redundant since it is implied by the occurrence of \( c \), thus we recover Dyckhoff’s rule \((L \rightarrow \rightarrow)\).

To prove the soundness of LJT\(_{\text{SAT}}\), we show that an LJT\(_{\text{SAT}}\)-derivation can be translated into the calculus LJT. In derivations, a double line marks the application of more than one rule. Firstly, we prove the soundness of rule (cpl).

\(^6\) These effectful additions account for the distinction between \( R^0 \), \( R^1 \) and \( R^2 \) above.
Lemma 2 (Soundness of rule (cpl)). If $R \vdash_{cpl} q$ then $\vdash_{LJT} R \Rightarrow q$.

Proof. We proceed via contraposition, so suppose $\not\vdash_{LJT} R \Rightarrow q$. By completeness, there is a Kripke model $(W, \leq, r, \vartheta)$ containing a world $w \in W$ such that $w \models \bigwedge R$ and $w \not\models q$. Now consider any flat clause $\varphi = \bigwedge A_1 \rightarrow \bigvee A_2 \in R$. The valuation $\vartheta(w)$ either has $A_2 \cap \vartheta(w) \neq \emptyset$ or $A_2 \cap \vartheta(w) = \emptyset$. If $A_2 \cap \vartheta(w) \neq \emptyset$ then $\varphi$ is classically true at $w$. If $A_2 \cap \vartheta(w) = \emptyset$ then reflexivity demands $A_1 \not\subseteq \vartheta(w)$, as otherwise $w \not\models \varphi$, contradicting our assumption. Again, $\varphi$ is classically true at $w$. That is, $w$ by itself is a classical model that also makes $\bigwedge R$ true and $q$ false, so $R \not\vdash_{cpl} q$. By contraposition, if $R \vdash_{cpl} q$ then $\vdash_{LJT} R \Rightarrow q$. Notice that this proof only works because $R$ contains flat clauses.

A syntactic way of proving Lemma 2 is to consider the proof returned by a (proof-producing) SAT-solver, justifying the unsatisfiability of $R, \overline{q}$ (where $\overline{q}$ is the negation of $q$) with a resolution proof concluding the empty clause $\bot$ from the flat clauses $R$, and $\overline{q}$. Indeed, the resolution rule is perfectly valid in intuitionistic logic if a clause $\overline{a_1} \lor \cdots \lor \overline{a_n} \lor b_1 \lor \cdots \lor b_m$ is read as the flat clause $a_1 \land \cdots \land a_n \rightarrow b_1 \lor \cdots \lor b_m$, as pointed out by Claessen and Rósen [1]. Removing $q \rightarrow \bot$ from the leaves of the resolution tree leaves $q$ at its root, yielding an intuitionistic proof of $R \vdash_{ipl} q$. Completeness of LJT [2] concludes $\vdash_{LJT} R \Rightarrow q$.

We prove the main lemma for the soundness of $LJT_{\text{SAT}}$.

Lemma 3. If $\vdash_{LJT_{\text{SAT}}} R, X \Rightarrow q$ then $\vdash_{LJT} R, X \Rightarrow q$.

Proof. Let $\mathcal{D}$ be a $LJT_{\text{SAT}}$-derivation of $R, X \Rightarrow q$; we prove the lemma by induction on the depth of $\mathcal{D}$. If the root rule of $\mathcal{D}$ is (cpl), the assertion follows by Lemma 2. Let us assume that $\mathcal{D}$ is

\[
\frac{\mathcal{D}_1}{R_1, X_1 \vdash_{ipl} \varphi, R_2, X_2 \Rightarrow q} \quad \text{(cut)}
\]

By the completeness of LJT [2], there exists an LJT-derivation $\mathcal{E}_1$ of $R_1, X_1 \Rightarrow \varphi$. By the induction hypothesis, there exists an LJT-derivation $\mathcal{E}_2$ of $\varphi, R_2, X_2 \Rightarrow q$. Since (cut) is admissible in LJT [2], from $\mathcal{E}_1$ and $\mathcal{E}_2$ we get an LJT-derivation of $R_1, R_2, X_1, X_2 \Rightarrow q$. Otherwise, $\mathcal{D}$ has the form

\[
\frac{\mathcal{D}_1}{R_1, b \rightarrow c, X_1, A_1 \Rightarrow b} \quad \frac{\mathcal{D}_2}{R_2, \varphi, X \Rightarrow q} \quad \text{(ljt)}
\]

\[
\frac{R_1, R_2, X \Rightarrow q}{i = (a \rightarrow b) \rightarrow c} \quad \frac{X_i = X \setminus \{i\}}{\varphi = \bigwedge (A_1 \setminus \{a\}) \rightarrow c}
\]

Obtaining $\mathcal{E}_1$ (resp. $\mathcal{E}_2$) from $\mathcal{E}_1$ (resp. $\mathcal{D}_2$) by the induction hypothesis, we get the following LJT-derivation of $R, X \Rightarrow q$. We use here the fact that in LJT, weakenings are admissible (so we can assume $a \in A_1$), and so are cuts.

\[
\frac{R_1, b \rightarrow c, X_1, A_1 \Rightarrow b}{R_1, X, A_1 \setminus \{a\} \Rightarrow c} \quad \frac{R_1, X, c, A_1 \setminus \{a\} \Rightarrow c}{R_1, X, A_1 \setminus \{a\} \Rightarrow c} \quad \frac{A x}{L \rightarrow \rightarrow} \quad \frac{L \land, R \rightarrow}{R_1, X \Rightarrow \varphi} \quad \frac{R_2, \varphi, X \Rightarrow q}{R_1, R_2, X \Rightarrow q} \quad \text{(cut)}
\]

\[\square\]
procedure LJTSatMain($R_0$, $X_0$, $q_0$)
  // $s$, $R_0$, $X_0$ are global parameters
  $s$ ← newSolver()
  for $\varphi \in R_0$ do addClause($s$, $\varphi$)
  for $(a \rightarrow b) \rightarrow c \in X_0$ do addClause($s$, $b \rightarrow c$)
  $\tau$ ← LJTSat($R_0$, $X_0$, $\emptyset$, $q_0$, $[]$)
  if $\tau = K$ then return $K$
  else // $\tau = (D, R, \emptyset)$, with $R = \{\varphi_1, \ldots, \varphi_n\}$
    return the LJTSat-derivation
end

Fig. 6. The procedure LJTSatMain

By Lemma 3 and the soundness of LJT, we conclude:

Theorem 1 (Soundness of LJTSat). $\vdash_{\text{LJTSat}} R, X \Rightarrow q$ implies $R, X \vdash_{\text{ipl}} q$.

6 Proof-search using LJTSat

We present the proof-search procedure based on the calculus LJTSat, implemented by the main function LJTSatMain (Fig. 6), which exploits the auxiliary recursive function LJTSat (Fig. 7). They correspond to the functions prove and intuitPR respectively, enhanced with explicit proof/counter-model-construction. The worlds of the counter-models to be constructed are sequences of implication clauses, with $[]$ denoting the empty sequence and $w::\iota$ denoting the extension of sequence $w$ with clause $\iota$. Function LJTSat takes as its last argument the root world of the counter-model to be constructed, if the input is not IPL-valid. Initially, LJTSatMain calls LJTSat with the empty sequence as root world, as shown in Fig. 6, where $(\text{cut}_{\text{ipl}}^*)$ also denotes a chain of $n$ successive applications of $(\text{cut}_{\text{ipl}})$.

A Kripke model $K = \langle W, \leq, r, \varnothing \rangle$ is a counter-model for a sequent $\sigma = R, X \Rightarrow q$, written $K \not\models \sigma$, if $r \not\models R \cup X$ and $r \not\models q$. In our incarnation of intuit, the counter-model is obtained by gluing Kripke models, as explained next.

Let $\Theta$ be a family $(K_i)_{i \in I}$ of Kripke models and $M$ a classical model such that, for every $i \in I$, the root $r_i$ of $K_i$ obeys $r_i \models M$. We write $\text{Mod}(r, \Theta, M)$ for the model $K = \langle W, \leq, r, \varnothing \rangle$ obtained by gluing all the models in $\Theta$ over $r$, where $r$ is a new world such that $\varnothing(r) = M$. More specifically, if $\Theta = (\langle W_i, \leq_i, r_i, \varnothing_i \rangle)_{i \in I}$, where the sets $W_i$ are pairwise disjoint and none of them contains
The procedure \( \text{LJTSat} \)

\[
\textbf{procedure} \text{LJTSat}(R, X, A, q, r) \\
\tau_0 \leftarrow \text{satProve}(s, A, q) \\
\text{if } \tau_0 = \text{Yes}(A') \text{ then} \\
\quad \text{return } (D, R(s), A') \text{ where } D \text{ is} \\
\quad R(s), A' \models q \quad \text{(cpl)} \\
\text{else} \quad \text{/} \quad \tau_0 = \text{No}(M) \\
\quad \Theta \leftarrow \emptyset \quad \text{// Empty family of Kripke models} \\
\quad \text{for } i = (a \to b) \to c \in X \text{ such that } a \notin M \text{ and } b \notin M \text{ and } c \notin M \text{ do} \\
\quad \quad \tau_1 \leftarrow \text{LJTSat}(R \cup \{b \to c\}, X_i, M \cup \{a\}, b, r_x) \quad \text{// where } X_i = X \setminus \{i\} \\
\quad \quad \text{if } \tau_1 = K_i \text{ then } \Theta \leftarrow \Theta \cup \{i \mapsto K_i\} \quad \text{// adding } K_i \text{ with index } i \\
\quad \quad \text{else} \quad \text{// } \tau_1 = (D_1, R_1, A_1) \\
\quad \quad \quad \tilde{\varphi} \leftarrow \bigwedge (A_1 \setminus \{a\}) \to c \\
\quad \quad \quad \text{addClause}(s, \tilde{\varphi}) \\
\quad \quad \text{addClause}(\tilde{\tau}_2, \text{LJTSat}(R \cup \{\tilde{\varphi}\}, X, A, q, r)) \\
\quad \quad \text{if } \tau_2 = K_2 \text{ then return } K_2 \\
\quad \quad \text{else} \quad \text{// } \tau_2 = (D_2, R_2, A_2) \\
\quad \quad \quad \text{return } (D, D_1 \cup D_2, A_2) \text{ where } D \text{ is} \\
\quad \quad \quad \quad D_1 \\
\quad \quad \quad \quad R, R_1, b \to c, X_i, A_1 \Rightarrow b \\
\quad \quad \quad \quad D_2 \\
\quad \quad \quad \quad R, R_2, \tilde{\varphi}, X, A_2 \Rightarrow q \\
\quad \quad \quad \quad (\text{ljt}) \\
\text{end} \\
\text{end} \\
\text{return } \text{Mod}(r, \Theta, M) \\
\text{end}
\]

Fig. 7. The procedure \( \text{LJTSat} \)

\( r \), then \( \text{Mod}(r, \Theta, M) \) is the model \( K = (W; \leq, r, \vartheta) \) such that

\[
W = \{r\} \cup \bigcup_{i=1}^{n} W_i \quad \leq_0 = \{(r, r_1), \cdots, (r, r_n)\} \cup \bigcup_{i=1}^{n} \leq_i \quad \vartheta = \bigcup_{i=1}^{n} \vartheta_i \cup \{(r, M)\}
\]

and \( \leq \) is the reflexive-transitive closure of \( \leq_0 \) (\( \cup \) denotes disjoint union).

If \( \Theta = \emptyset \), then \( K = \langle \{r\}, \{(r, r)\}, r, \vartheta \rangle \) only contains the reflexive world \( r \) with \( \vartheta(r) = M \). Given a set \( X \) and a classical model \( M \), we write \( X_M \) for \( \{((a \to b) \to c) \in X \mid a \notin M, b \notin M, c \notin M\} \).

\textbf{Lemma 4.} Let \( \sigma = R, X \Rightarrow q \) be a sequent, \( r \) be a world, \( M \) be a classical model and \( \Theta \) be a family \( (K_i)_{i \in X_M} \) of Kripke models indexed by \( X_M \) such that:

(i) \((a \to b) \to c \in X \) implies \( b \to c \in R \);
(ii) \( M \models R \) and \( M \not\models q \);
(iii) For every \( i \in X_M \), we have \( K_i \not\models R, X_i, M, a \Rightarrow b \).

Then, \( \text{Mod}(r, \Theta, M) \not\models \sigma \).
Proof. Let $\mathcal{K}$ be $\text{Mod}(r, \Theta, M) = (W, \leq, r, \vartheta)$, whence $r \models M$. We have to show that, at $r$, all the formulae in $R \cup X$ are forced and $q$ is not forced. By (ii), we immediately get $r \not\models q$. We prove the cases for $R$ and $X$.

Proof that $r \models R$: Suppose $\varphi \in R$ and assume $\varphi = \bigwedge A_1 \rightarrow \bigvee A_2$ with $A_1 \neq \emptyset$. Let $w \in W$ be any world such that $r \leq w$ and $w \models \bigwedge A_1$; we prove $w \models \bigvee A_2$.

If $w = r$, we have $r \models \bigwedge A_1$, which implies $A_1 \subseteq M$. Since $M \models \varphi$, we get $M \models \bigvee A_2$, which implies $r \models \bigvee A_2$. If $w \neq r$, $w$ must be in some $W_1$ for some $\mathcal{K}_i = (W_1, \leq_i, r_i, \vartheta_i)$ in $\Theta$, with $i \in X_M$. Thus $r_i \leq w$ in $\mathcal{K}_i$. By (iii), $r_i \models R$ in $\mathcal{K}_i$, and $\varphi \in R$, so $r_i \models \varphi$ in $\mathcal{K}_i$. Hence $r_i \models \varphi$ in $\mathcal{K}_i$, and the persistence of $\models$ gives $w \models \varphi$ in $\mathcal{K}_i$. Since $w \models \bigwedge A_1$, we obtain $w \models \bigvee A_2$. The case $A_1 = \emptyset$ (namely, $\varphi = \bigvee A_2$) is similar.

Proof that $r \models X$: First note that by (i), $r \models b \rightarrow c$ for every $(a \rightarrow b) \rightarrow c \in X$. Choose any $i = (a \rightarrow b) \rightarrow c \in X$ and let $w$ be any world such that $r \leq w$ and $w \models a \rightarrow b$; we prove $w \models c$.

If $c \in M$, then $r \models c$ by construction, hence $w \models c$. If $b \in M$ then $r \models b$ by construction, and we already have $r \models b \rightarrow c$, so we get $r \models c$, hence $w \models c$. If $a \in M$ then $r \models a$ and $r \models b \rightarrow c$ and $w \models a \rightarrow b$, giving $w \models c$. The previous three cases are independent, thus $w \models c$ if $a \in M$ or $b \in M$ or $c \in M$.

So suppose $a \notin M$ and $b \notin M$ and $c \notin M$. By (iii), $\Theta$ contains a model $\mathcal{K}_i = (W_i, \leq_i, r_i, \vartheta_i)$ such that $\mathcal{K}_i \not\models R, X, M, a \Rightarrow b$. Thus $r_i \not\models a \rightarrow b$ in $\mathcal{K}_i$, hence $r \not\models a \rightarrow b$ in $\mathcal{K}$. By reverse persistence $r \not\models a \rightarrow b$ (in $\mathcal{K}$), which implies $w \not\models r$. There is also a model $\mathcal{K}_i = (W_i', \leq', r', \vartheta')$ of $\Theta$ such that $w \in W_i'$, hence $r \leq r' \leq w$ (in $\mathcal{K}$). We need to cover the two cases $i = i'$ and $i \neq i'$:

1. If $i' = i$, by (iii) $r_i \models a$ in $\mathcal{K}_i$, thus $w \models a$ (in $\mathcal{K}$). Since $r \models b \rightarrow c$, we have $w \models b \rightarrow c$. Then $w \models a \rightarrow b$ gives $w \models c$.

2. If $i' \neq i$ then $i \in X_{i'}$, hence $r' \models i$ in $\mathcal{K}_{i'}$, whence it follows that $w \models i$ (in $\mathcal{K}$). Our initial assumption that $w \models a \rightarrow b$ gives $w \models c$. \qed

In Fig. 6 we define the proof-search function $\text{LJTSatMain}$ such that $\text{LJTSatMain}(R_0, X_0, q_0)$ returns either an $\text{LJT}_{\text{Sat}}$-derivation of $\sigma_0 = R_0, X_0 \Rightarrow q_0$ or a counter-model $\mathcal{K}$ for $\sigma_0$. Worlds of $\mathcal{K}$ are sequences of implication clauses, ordered by the prefix order on sequences, and the empty sequence $[]$ is its root world. We set:

$$H_0 = \{ b \rightarrow c \mid (a \rightarrow b) \rightarrow c \in X_0 \}$$
$$V_0 = \{ p \in V \mid p \text{ occurs in } \sigma_0 \}$$
$$M_{\psi_0}(R) = \{ M \subseteq V_0 \mid M \models R \}$$

for any set $R$ of flat clauses.

The call $\text{LJTSatMain}(R_0, X_0, q_0)$ defines a SAT-solver $s$ and initializes it by storing all the clauses in $R_0 \cup H_0$; we consider $s$, $R_0$ and $X_0$ as global parameters.

It exploits the auxiliary recursive procedure $\text{LJTSat}$ defined in Fig. 7. A call $\text{LJTSat}(R, X, A, q, r)$ performed during the computation of the main call $\text{LJTSatMain}(R_0, X_0, q_0)$ has the following specification.

Input Assumptions (IA):
- $R \subseteq R(s)$ and $X \subseteq X_0$;
- for every $\varphi \in R(s)$, we have $R_0, X_0 \vdash_{\text{ipd}} \varphi$;
- $r$ is a sequence of implication clauses.
Output Properties (OP):

\( \text{LJTSat}(R, X, A, q, r) \) yields a triple \((D, R', A')\) or else a model \(K\) with:
- \( R' \subseteq R(s) \) and \( A' \subseteq A \);
- for every \( \varphi \in R(s) \), we have \( R_0, X_0 \vdash_{\text{ipl}} \varphi \);
- \( D \) is an \( \text{LJTSat} \)-derivation of \( R, R', X, A' \Rightarrow q \);
- \( K \) has root \( r \) and worlds are ordered by the prefix order on sequences;
- \( K \not\models R, H_0, X, A \Rightarrow q \).

In (IA), \( R(s) \) refers to the clauses in the SAT-solver \( s \) at the beginning of the computation of \( \text{LJTSat}(R, X, A, q, r) \); in (OP), \( R(s) \) is the set of clauses in \( s \) at the end of the computation. Note that (OP) implies that the call \( \text{LJTSat}(R, X, A, q) \) terminates. To prove the correctness of \( \text{LJTSat} \), we have to show that, if the assumptions (IA) are matched, then (OP) holds. We need the following property about derivability in IPL.

**Lemma 5.** \( R, X, A \vdash_{\text{ipl}} b \) implies \( R, X, (a \rightarrow b) \vdash_{\text{ipl}} \bigwedge (A \setminus \{ a \}) \rightarrow c \).

**Proof.** Let \( A' = A \setminus \{ a \} \). If \( R, X, A \vdash_{\text{ipl}} b \), then \( R, X, A' \vdash_{\text{ipl}} a \rightarrow b \), which implies \( R, X, (a \rightarrow b) \rightarrow c, A' \vdash_{\text{ipl}} c \), hence \( R, X, (a \rightarrow b) \rightarrow c \vdash_{\text{ipl}} \bigwedge A' \rightarrow c \). \( \square \)

To prove correctness, put the following order relation on pairs \((R, X)\) such that \( R \) is any set of flat clauses and \( X \subseteq X_0 \):

\[
(R', X') \prec (R, X) \quad \text{iff} \quad \big( X' \subset X \big) \quad \text{or} \quad \big( X' = X \quad \text{and} \quad M_{/\nu_0} (R') \subseteq M_{/\nu_0} (R) \big)
\]

Since the sets \( X \) and \( M_{/\nu_0} (R) \) are finite, the relation \( \prec \) is well-founded, hence we can prove correctness of \( \text{LJTSat} \) (Lemma 6) by induction on \( \prec \).

**Lemma 6.** If a call \( \text{LJTSat}(R, X, A, q, r) \) satisfies (IA) then (OP) holds.

**Proof.** We use the main induction hypothesis (MIH) below and show that the invariant (Inv) holds at any point of the computation described in Fig. 7:

\[
(\text{MIH}): \text{if } (R', X') \prec (R, X), \text{ then the lemma holds for } \text{LJTSat}(R', X', A', q', r');
\]

\[
(\text{Inv}): \text{for every } \varphi \in R(s), \text{ we have } R_0, X_0 \vdash_{\text{ipl}} \varphi.
\]

At the start of the computation (Inv) holds by (IA). Let \( \tau_0 \) be the value computed at line 2. If \( \tau_0 = \text{Yes}(A') \), then the triple \((D, R(s), A')\) is returned at line 4, with \( D \) defined at line 5; by definition of \( \text{satProve} \), it holds that \( R(s), A' \vdash_{\text{cpl}} q \) and \( A' \subseteq A \), hence (OP) holds. Otherwise, since \( R \cup H_0 \subseteq R(s) \), we have:

\[
(\text{P0}) \quad \tau_0 = \text{No}(M) \quad \text{and} \quad M \models R \cup H_0 \cup A \quad \text{and} \quad M \not\models q.
\]

Without loss of generality, we can assume \( M \subseteq V_0 \), namely \( M \in M_{/\nu_0}(R) \). If, for every \( (a \rightarrow b) \rightarrow c \in X \), the loop condition at line 8 does not hold, then the loop at lines 8–21 is skipped and the model \( \text{Mod}(r, \emptyset, M) \) is returned at line 22, which is a counter-model for \( R, H_0, A \Rightarrow q \) by Lemma 4; thus (OP) holds. Let us assume that the loop at lines 8–21 is entered. We prove that at every iteration of the loop the following properties hold, where \( \tau_1 \) and \( \tau_2 \) are the values computed at lines 9 and 14 respectively, and \( \check{\varphi} = \bigwedge (A_1 \setminus \{ a \}) \rightarrow c \) is defined at line 12:
\(\tau_1 = (D_1, R_1, A_1)\) or \(\tau_1 = K_1\) where:
- \(R_1 \subseteq R(s)\) and \(A_1 \subseteq M \cup \{a\}\);
- \(D_1\) is an \(\text{LJT}_{\text{SAT}}\)-derivation of \(R, R_1, b \rightarrow c, X_1, A_1 \Rightarrow b\);
- \(K_1\) has root \(r::u\) and worlds are ordered by the prefix order on sequences;
- \(K_1 \not\models R, H_0, X, M, a \Rightarrow b\).

\((P2)\) \(\tau_2 = (D_2, R_2, A_2)\) or \(\tau_2 = K_2\) where:
- \(R_2 \subseteq R(s)\) and \(A_2 \subseteq A\);
- \(D_2\) is an \(\text{LJT}_{\text{SAT}}\)-derivation of \(R, R_2, \tilde{\varphi}, X, A_2 \Rightarrow q\);
- \(K_2\) has root \(r\) and worlds are ordered by the prefix order on sequences;
- \(K_2 \not\models R, \tilde{\varphi}, H_0, X, A \Rightarrow q\).

Let us consider the first iteration of the loop and let \(\epsilon = (a \rightarrow b) \rightarrow c \in X\) be the selected clause (hence, \(a \not\in M\), \(b \not\in M\) and \(c \not\in M\)). The call to \(\text{LJTSat}\) at line 9 satisfies (IA). Since \(X_1 \subseteq X\), we have \((R \cup \{b \rightarrow c\}, X_1) \prec (R, X)\). By (MIH) \(\tau_1\) satisfies (OP); this proves (P1). Note that (OP) guarantees that (Inv) holds after the computation of \(\tau_1\). At line 13, \(\tilde{\varphi}\) is added to \(s\); we check that (Inv) is preserved, namely \(R_0, X_0 \vdash_{\text{ipl}} \tilde{\varphi}\). By (P1) and Soundness of \(\text{LJT}_{\text{SAT}}\) (Theorem 1), \(R, R_1, b \rightarrow c, X_1, A_1 \vdash_{\text{ipl}} \tilde{\varphi}\) and, by Lemma 5, we get \(R, R_1, b \rightarrow c, X \vdash_{\text{ipl}} \tilde{\varphi}\), hence \(R, R_1, b \rightarrow c, X_0 \vdash_{\text{ipl}} \tilde{\varphi}\). Since \(R \cup R_1 \cup \{b \rightarrow c\} \subseteq R(s)\), from (Inv) it follows that \(R_0, X_0 \vdash_{\text{ipl}} \tilde{\varphi}\). The call to \(\text{LJTSat}\) at line 14 matches (IA). To apply (MIH), we have to check that:

\((P3)\) \((R \cup \{\tilde{\varphi}\}, X) \prec (R, X)\).

Clearly, \(M_{/ V_0}(R \cup \{\tilde{\varphi}\}) \subseteq M_{/ V_0}(R)\); to conclude the proof of (P3), we show that the inclusion is strict. Note that \(M \in M_{/ V_0}(R)\) (see (P0) and the subsequent remark). For a contradiction, assume \(M \models \tilde{\varphi}\). Since \(A_1 \subseteq M \cup \{a\}\), we get \(A_1 \setminus \{a\} \subseteq M\). By definition of \(\tilde{\varphi}\), it follows that \(M \models c\), namely \(c \in M\), a contradiction. Thus, \(M \not\models \tilde{\varphi}\), which implies \(M \not\in M_{/ V_0}(R \cup \{\tilde{\varphi}\})\); this proves (P3).

We can apply (MIH) to the recursive call at line 14 and we get (P2) and the preservation of (Inv). Let us consider the iteration \(k + 1\) of the loop \((k \geq 1)\). We can repeat the above reasoning to prove that (P1) and (P2) hold at iteration \(k + 1\); the invariant property (Inv) is crucial to guarantee that the recursive calls at lines 9 and 14 satisfy (IA). We conclude that (P1) and (P2) hold at every iteration of the loop.

Let us assume that, at some iteration of the loop, \(\tau_1\) is not a model, namely \(\tau_1 = (D_1, R_1, A_1)\). If \(\tau_2\) is a model \(K_2\), then \(K_2\) is returned at line 15 and (OP) follows from (P2). Otherwise, \(\tau_2 = (D_2, R_2, A_2)\) and \((D, R_1 \cup R_2, A_2)\) is returned at line 17, where \(D\) is the \(\text{LJT}_{\text{SAT}}\)-derivation displayed at line 18; accordingly (OP) holds. Finally, let us assume that, at every iteration, \(\tau_1\) is a model. Since \(X\) is finite, the loop eventually ends and the model \(\text{Mod}(r, \Theta, M)\) is returned at line 22. At that point, \(\Theta\) has been completed into a family \((K_c)_{c \in X_M}\) such that for every \(c \in X_M\), \(K_c\) has root \(r::u\) and its worlds are ordered by the prefix order on sequences, by (P1). \(\text{Mod}(r, \Theta, M)\) has root \(r\) and also uses the prefix order on sequences. So in order to prove (OP), we only have to check that \(\text{Mod}(r, \Theta, M)\) is a counter-model for \(R, H_0, X, A \Rightarrow q\). Let \(R' = R \cup H_0 \cup M\) and \(\sigma' = R', X \Rightarrow q\); by (P0) and (P1), it follows that \(\sigma', M, \Theta\) satisfy the assumptions of Lemma 4. Since \(A \subseteq M, \text{Mod}(r, \Theta, M) \not\models \sigma'\) and (OP) holds. \(\square\)
By Lemma 6, we get:

**Theorem 2 (Correctness of LJT SATMain).** \( \text{LJT SATMain}(R_0, X_0, q_0) \) returns either an \( \text{LJT SAT} \)-derivation of \( \sigma_0 = R_0, X_0 \Rightarrow q_0 \) or a counter-model for \( \sigma_0 \).

**Proof.** Consider the call \( \text{LJT SAT}(R_0, X_0, \emptyset, q_0, [\]) \) at line 6. When \( \text{LJT SAT} \) is called, \( R(s) = R_0 \cup H_0 \). Let \( b \rightarrow c \in H_0 \); since \( X_0 \) contains a formula of the kind \((a \rightarrow b) 
(\rightarrow c \in H_0 \); since \( X_0 \) contains a formula of the kind \((a \rightarrow b) \rightarrow c \) and \((a \rightarrow b) \rightarrow c \vdash \text{ipl} b \rightarrow c \), it follows that \( R_0, X_0 \vdash \text{ipl} b \rightarrow c \). Thus, the call to \( \text{LJT SAT} \) satisfies (IA); by Lemma 6, the returned value \( \tau \) satisfies (OP). If \( \tau \) is a counter-model \( \mathcal{K} \), then \( \mathcal{K} \) is returned at line 7; since \( \mathcal{K} \not\models R_0, H_0, X_0 \Rightarrow q \), we get \( \mathcal{K} \not\models \sigma_0 \). Otherwise \( \tau = (D, \emptyset, \emptyset) \), where \( D \) is an \( \text{LJT SAT} \)-derivation of \( R_0, R, X_0 \Rightarrow q_0 \) and \( R_0, X_0 \vdash \text{ipl} \varphi \), for every \( \varphi \in R \). Accordingly, the returned derivation, displayed at line 10, is an \( \text{LJT SAT} \)-derivation of \( \sigma_0 \).

As a consequence, we get:

**Theorem 3 (LJT SAT completeness).** \( R, X \vdash \text{ipl} q \) implies \( \models \text{LJT SAT} R, X \Rightarrow q \).

Let us consider the call \( \text{LJT SATMain}(R_0, X_0, q_0) \); we show that we can build an \( \text{LJT} \)-derivation \( D_\varphi \) of \( R_0, X_0 \Rightarrow \varphi \), for every clause \( \varphi \) stored in the SAT-solver \( s \) during the computation. Let \( \varphi = b \rightarrow c \) be a clause introduced at the beginning of \( \text{LJT SATMain} \) (line 5 in Fig. 6). Then, \((a \rightarrow b) \rightarrow c \in X_0 \) and, setting \( X' = X_0 \setminus \{(a \rightarrow b) \rightarrow c\} \), \( D_\varphi \) is:

\[
\frac{R_0, X', b, b \rightarrow c, a \rightarrow b}{R_0, X', c, b \rightarrow c} \quad \frac{R_0, X_0, b \rightarrow c}{R_0, X_0 \Rightarrow b \rightarrow c} \quad \frac{R_0, X_0 \Rightarrow b \rightarrow c}{} \quad \frac{R_0, X_0 \Rightarrow b \rightarrow c}{R \Rightarrow} \quad L \rightarrow \to
\]

Let us consider the clause \( \tilde{\varphi} \) added in the loop of \( \text{LJT SAT} \) when the clause \( \iota = (a \rightarrow b) \rightarrow c \) is considered (line 13 in Fig. 7). By Point (P1) in the proof of Lemma 6, there exists an \( \text{LJT SAT} \)-derivation \( D_1 \) of \( R, R_1, b \rightarrow c, X_i, A_1 \Rightarrow b \). Note that \( R \cup R_1 \cup \{b \rightarrow c\} \) has the form \( R_0 \cup \{\varphi_1, \ldots, \varphi_n\} \), where the clauses \( \varphi_1, \ldots, \varphi_n \) are in \( s \) and \( X_i \subseteq X_0 \). Let \( D'_1 \) be the \( \text{LJT SAT} \)-derivation:

\[
\frac{R_0, X_0, A_1 \Rightarrow b}{R_0, X_0, A_1 \Rightarrow b} \quad (\text{cut}_{\varphi})
\]

By construction, for every judgment \( R_0, X_0 \vdash \text{ipl} \varphi' \) occurring in \( D'_1 \), the clause \( \varphi' \) is in \( s \), thus we can assume that the \( \text{LJT} \)-derivation \( D_\varphi \) has already been defined. We can turn \( D'_1 \) into an \( \text{LJT} \)-derivation \( \mathcal{E}_1 \) of \( R_0, X_0, A_1 \Rightarrow b \).

If \( a \not\in A_1 \), then \( \tilde{\varphi} = \bigwedge A_1 \rightarrow c \) and \( D_\tilde{\varphi} \) is the \( \text{LJT} \)-derivation

\[
\frac{R_0, X_0, A_1 \Rightarrow b}{b, b \rightarrow c, a \rightarrow b} \quad \frac{b, c \Rightarrow c}{Ax} \quad \frac{b, (a \rightarrow b) \rightarrow c \Rightarrow c}{L \rightarrow \to} \quad \frac{R_0, X_0, A_1 \Rightarrow c}{R_0, X_0 \Rightarrow \bigwedge A_1 \rightarrow c} \quad (\text{cut}) \quad L \rightarrow \to
\]
If $A_1 = \emptyset$ (hence $\tilde{\varphi} = c$) the bottom applications of $L \land$ and $R \rightarrow$ are crossed out.

Let $a \in A_1$ and $A'_1 = A_1 \setminus \{a\}$. Thus, $\tilde{\varphi} = \bigwedge A' \rightarrow c$ and $D_{\tilde{\varphi}}$ is the derivation

$$
\frac{b, b \rightarrow c, a \Rightarrow a}{R_0, X_0, A'_1 \Rightarrow a \rightarrow b} \quad \frac{a \Rightarrow b, b \rightarrow c, a \Rightarrow b}{R_0, X_0, A'_1 \Rightarrow a \rightarrow b} \quad \frac{a \rightarrow b, (a \rightarrow b) \rightarrow c \Rightarrow c}{L \rightarrow \rightarrow} \quad \frac{a \Rightarrow b, c \Rightarrow c}{L \rightarrow \rightarrow} \quad \frac{L \rightarrow \rightarrow}{L \rightarrow \rightarrow}
$$

If $A' = \emptyset$ (hence $\tilde{\varphi} = c$) the bottom applications of $L \land$ and $R \rightarrow$ are skipped.

By the above discussion, we can enhance the procedures $LJTSatMain$ and $LJTSat$ so that, whenever a flat clause $\varphi$ is added to the SAT-solver, an LJT-derivation $D_\varphi$ of $R_0, X_0 \Rightarrow \varphi$ is stored. Let us assume that $LJTSatMain(R_0, X_0, q_0)$ returns an $LJT_{\text{SAT}}$-derivation $D$ of $\sigma_0 \Rightarrow R_0, X_0 \Rightarrow q_0$ Proceeding as in the proof of Lemma 3, we can exploit the derivations $D_{\tilde{\varphi}}$ to translate $D$ into an LJT-derivation of $\sigma_0$.

7 Discussion, Further Work and Conclusions

The construction of Kripke countermodels from failed proof search in (a variant of) LJT was already explored by Dyckhoff and Pinto [3]. In this paper we have merged proof search and countermodel construction into one procedure based on LJT while benefitting at the same time from the insights from $\text{intuit}$ regarding incremental SAT-solvers.

The rule $(L \rightarrow \rightarrow)$ from Fig. 4 can be interpreted semantically as follows by reading it from conclusion to premises. The antecedent of the conclusion requires the current world $w$ to make $(a \rightarrow b) \rightarrow c$ true, to make all members of $\Gamma$ true and to make $\delta$ false. If $w \models c$ then we have the right premise. Else $w \not\models c$ and therefore, $w \not\models a \rightarrow b$. But that means that there exists a $v \geq w$ such that $v \models \Gamma$ and $v \models a$ and $v \not\models b$, implying that $v \models b \rightarrow c$, which is the left premise.

We have shown that our simpler recursive version of $\text{intuit}$ can be reconciled with this semantic view if we generalise the rule $(L \rightarrow \rightarrow)$ into the rule $(ljt)$. By doing so, we can utilise an incremental SAT-solver to implement the right premise of the rule $(ljt)$ by “restarting” the SAT-solver with additional flat clauses learned during the process of finding the derivation of the left premise of $(ljt)$.

There are many sequent and natural deduction calculi that contain rules which have this “here” or “at some successor” flavour. For example, the LSJ calculus of Ferrari et al [5] and the traditional tableau calculus for linear temporal logic PLTL [16]. Can we extend our insights to such calculi to obtain incremental SAT-based decision procedures for these calculi too [10]?

Another direction for future work is the extent to which this approach relies on the clausification of the input formula. Indeed, LJT is able to natively treat any IPL formula. Technically, could the calculus $LJT_{\text{SAT}}$ be extended to any sequent, not necessarily in clausal form? If so how would the interaction with the SAT solver be organised?
Acknowledgment This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 689176.

References