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

Camillo Fiorentini[1] and Rajeev Goré[2] and Stéphane Graham-Lengrand[3]

[1] Department of Computer Science, University of Milan
[2] Research School of Computer Science, Australian National University
[3] SRI International

Abstract. Claessen and Rósen have recently presented an automated theorem prover, `intuit`, for intuitionistic propositional logic which utilises a SAT-solver. We present a sequent calculus perspective of the theory underpinning `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 Rósen [1] have recently presented an automated theorem prover, `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 `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 `intuit`, organised around the top-level loop of the DPLL($\mathcal{T}$) 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 `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 $\wedge$, $\vee$, $\rightarrow$ with negation defined as $\neg\alpha := (\alpha \rightarrow \bot)$. We use $\mathrm{Atm} := V \cup \{\bot\}$ for the set of "atoms", denoted by lowercase Roman letters.

A rooted Kripke model for IPL is a quadruple $\langle W, \leq, r, \vartheta\rangle$ 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 $\vartheta : W \mapsto 2^{\mathrm{Atm}}$ 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 $p \in \vartheta(w)$ then $p \in \vartheta(v)$. Given a Kripke model $\langle W, \leq, r, \vartheta\rangle$, the valuation ϑ can be extended into a "forcing" relation between worlds and formulae as shown below:

$w \Vdash p$ iff $p \in \vartheta(w)$	$w \Vdash \alpha \rightarrow \beta$ iff $\forall v \geq w$, $v \Vdash \alpha$ implies $v \Vdash \beta$
$w \Vdash \alpha \wedge \beta$ iff $w \Vdash \alpha$ and $w \Vdash \beta$	$w \Vdash \bot$ never holds
$w \Vdash \alpha \vee \beta$ iff $w \Vdash \alpha$ or $w \Vdash \beta$	

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

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 α 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 $\Gamma \vdash_{\mathrm{ipl}} \delta$ when the formula $\bigwedge \Gamma \rightarrow \delta$ is IPL-valid and $\Gamma \vdash_{\mathrm{cpl}} \delta$ when it is CPL-valid, that is, when $M \models \gamma$ for all classical models M.

3 The Theorem Prover `intuit`

The `intuit` 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 α 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 $\vdash_{\mathrm{ipl}} \alpha$ iff $R, X \vdash_{\mathrm{ipl}} q$ (the definitions are in Section 3.1). The *sequent* $R \Rightarrow q$ constitutes an "abstraction" of the input formula α. 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 α 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 $R, X \nvdash_{\mathrm{ipl}} q$, 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 α that refines $R \Rightarrow q$ (technically, $R \subseteq R'$) and *defeats* model M, meaning that $M \not\models R'$ while still ensuring $\vdash_{\mathrm{ipl}} \alpha$ iff $R', X \vdash_{\mathrm{ipl}} q$. If it fails, then indeed α is not IPL-valid. If

```
1  procedure prove(R, X, q)
2  |  s = newSolver()
3  |  for φ ∈ R do addClause(s, φ)
4  |  for (a → b) → c ∈ X do addClause(s, b → c)
5  |  return intuitProve(s, X, ∅, q)
6  end
7  procedure intuitProve(s, X, A, q)
8  |  loop // CEGAR LOOP
9  |  |  τ0 ← satProve(s, A, q)
10 |  |  if τ0 = Yes(A′) then return Yes(A′)
11 |  |  else  // τ0 = No(M), with M a putative counter-model
12 |  |  |  if intuitCheck(s, X, M) then return No(M)
13 |  |  end
14 |  end
15 end
16 procedure intuitCheck(s, X, M)
17 |  for ι = (a → b) → c ∈ X such that a ∉ M and b ∉ M and c ∉ M do
18 |  |  τ1 ← intuitProve(s, Xι, M ∪ {a}, b)   // Xι = X \ {ι}
19 |  |  if τ1 = Yes(A1) then
20 |  |  |  addClause(s, ⋀(A1 \ {a}) → c)
21 |  |  |  return False
22 |  |  end
23 |  end
24 |  return True
25 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 α, or exhausts the set of putative counter-models and conclude that α 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($\mathcal{T}$) 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($\mathcal{T}$) loop. In other words `intuit` implements a recursive version of DPLL($\mathcal{T}$), although a single SAT-solver is used, incrementally, for all recursive calls.

3.1 `intuit` in detail

Firstly, the formula α is transformed into a formula $\bigwedge \Gamma \to q$, where q is an atom and Γ is a set of *flat clauses* φ and *implication clauses* ι, where:

$$\begin{array}{lll} \varphi & ::= & \bigwedge A_1 \to \bigvee A_2 \qquad A_1 \cup A_2 \subseteq \text{Atm} \\ \iota & ::= & (a \to b) \to c \qquad \{a, b, c\} \subseteq \text{Atm} \end{array}$$

$\bigwedge A_1$ is the conjunction of the atoms in A_1

$\bigvee A_2$ is the disjunction of the atoms in A_2 if $A_1 = \emptyset$ then $\varphi = \bigvee A_2$

```
1  procedure intuitProve1(s, X, A, q)
2  |  loop
3  |  |  τ0 ← satProve(s, A, q)
4  |  |  if τ0 = Yes(A′) then return Yes(A′)
5  |  |  else  // τ0 = No(M)
6  |  |  |  for ι = (a → b) → c ∈ X such that a ∉ M and b ∉ M and c ∉ M do
7  |  |  |  |  τ1 ← intuitProve1(s, Xι, M ∪ {a}, b)   // Xι = X \ {ι}
8  |  |  |  |  if τ1 = Yes(A1) then
9  |  |  |  |  |  addClause(s, ⋀(A1 \ {a}) → c)
10 |  |  |  |  |  go to line 2 (outer loop)
11 |  |  |  |  end
12 |  |  |  end
13 |  |  |  return No(M)
14 |  |  end
15 |  end
16 end
```

Fig. 2. The function `intuitProve1`

A flat clause where A_1 is empty is simply a disjunction $\bigvee A_2$ of atoms, and simply an atom a when A_2 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, $X_{\iota'}$ for the sets $X \setminus \{\iota\}$ and $X \setminus \{\iota'\}$, 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 α into a triple $\mathsf{clausal}(\alpha) = (R, X, q)$ whose cumulative size is linear in the size of α and that is equiprovable to α:

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

From now on, we focus on deciding $R, X \vdash_{\mathrm{ipl}} q$. 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 $\mathrm{R}(s)$ denotes the set of clauses stored in s:

`newSolver()`: create a new SAT-solver;
`addClause(`s`,` φ`)`: add the flat clause φ to the SAT-solver s;
`satProve(`s`,` A`,` q`)`: call the SAT-solver s to decide whether $\mathrm{R}(s), A \vdash_{\mathrm{cpl}} q$, 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:

$\mathrm{Yes}(A')$: thus $A' \subseteq A$ and $\mathrm{R}(s), A' \vdash_{\mathrm{cpl}} q$;
$\mathrm{No}(M)$: thus M is a classical model such that $M \models \mathrm{R}(s) \cup A$ and $M \not\models q$.

```
1  procedure intuitPR(s, X, A, q)
2  |  τ0 ← satProve(s, A, q)
3  |  if τ0 = Yes(A′) then return Yes(A′)
4  |  else // τ0 = No(M)
5  |  |  for ι = (a → b) → c ∈ X such that a ∉ M and b ∉ M and c ∉ M do
6  |  |  |  τ1 ← intuitPR(s, Xι, M ∪ {a}, b)   // Xι = X \ {ι}
7  |  |  |  if τ1 = Yes(A1) then
8  |  |  |  |  addClause(s, ⋀(A1 \ {a}) → c)
9  |  |  |  |  return intuitPR(s, X, A, q)
10 |  |  |  end
11 |  |  end
12 |  |  return No(M)
13 |  end
14 end
```

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

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

Yes($\emptyset$) if $R, X \vdash_{\mathrm{ipl}} q$;
No(M) if there is a Kripke model $\mathcal{K} = \langle W, \leq, r, \vartheta \rangle$ such that $\vartheta(r) = M$ and $r \Vdash R \cup X$ and $r \nVdash q$; this implies $R, X \nvdash_{\mathrm{ipl}} q$.

As sketched by Claessen and Rósen [1], if `prove(`R, X, q`)` returns No(M), then one can actually build the mentioned model $\mathcal{K}$ 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* Γ is a multiset of assumptions, and whose provability in LJT is denoted $\vdash_{\mathrm{LJT}} \Gamma \Rightarrow \alpha$. The key difference from LJ lies in the left introduction rules for implication. In order to introduce an implication $\eta \to \gamma$ on the left, LJT offers four rules, depending on the form of η, namely: either $\eta = p$, with $p \in V$; or $\eta = \alpha \wedge \beta$; or $\eta = \alpha \vee \beta$; or $\eta = \alpha \to \beta$ [4].

[4] We follow Troelstra and Schwichtenberg [14] where the calculus is called G4ip; the original ($L{\to}{\to}$) rule by Dyckhoff [2] has $\Gamma, \beta \to \gamma \Rightarrow \alpha \to \beta$ as the left premise.

$$\frac{}{\Gamma, p \Rightarrow p}\,\mathrm{Ax} \qquad \frac{}{\Gamma, \bot \Rightarrow \delta}\,L\bot \qquad \frac{\Gamma, \alpha, \beta \Rightarrow \delta}{\Gamma, \alpha \wedge \beta \Rightarrow \delta}\,L\wedge \qquad \frac{\Gamma \Rightarrow \alpha \qquad \Gamma \Rightarrow \beta}{\Gamma \Rightarrow \alpha \wedge \beta}\,R\wedge$$

$$\frac{\Gamma, \alpha \Rightarrow \delta \qquad \Gamma, \beta \Rightarrow \delta}{\Gamma, \alpha \vee \beta \Rightarrow \delta}\,L\vee \qquad \frac{\Gamma \Rightarrow \alpha_k}{\Gamma \Rightarrow \alpha_1 \vee \alpha_2}\,R\vee_k \qquad \frac{\Gamma, \alpha \Rightarrow \beta}{\Gamma \Rightarrow \alpha \to \beta}\,R\to$$

$$\frac{\Gamma, p, \beta \Rightarrow \delta}{\Gamma, p, p \to \beta \Rightarrow \delta}\,L0\to \qquad \frac{\Gamma, \alpha \to (\beta \to \gamma) \Rightarrow \delta}{\Gamma, (\alpha \wedge \beta) \to \gamma \Rightarrow \delta}\,L\wedge\to \qquad \frac{\Gamma, \alpha \to \gamma, \beta \to \gamma \Rightarrow \delta}{\Gamma, (\alpha \vee \beta) \to \gamma \Rightarrow \delta}\,L\vee\to$$

$$\frac{\Gamma, \beta \to \gamma, \alpha \Rightarrow \beta \qquad \Gamma, \gamma \Rightarrow \delta}{\Gamma, (\alpha \to \beta) \to \gamma \Rightarrow \delta}\,L\to\to \qquad p \in V,\ k \in \{1, 2\}$$

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 Γ_i, α_i corresponding to the ith premise $\Gamma_i \Rightarrow \alpha_i$ is strictly smaller than the multiset Γ, α 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 $R\vee_k$ and $(L\to\to)$, and therefore backtrack points need to be set only when applying those two rules. However $(L\to\to)$ 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 $(L\to\to)$ can be seen as a one-premise invertible rule, guarded by a side-condition:

$$(\Gamma, \beta \to \gamma, \alpha \Rightarrow \beta)\ \frac{\Gamma, \gamma \Rightarrow \delta}{\Gamma, (\alpha \to \beta) \to \gamma \Rightarrow \delta}.$$

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 $(L{\to}{\to})$ satisfying the same invertibility properties, corrrespond to the two recursive calls of `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($\mathcal{T}$) loop of `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 $(L{\to}{\to})$, which becomes:

$$\frac{R, b \to c, X_\iota, a \Rightarrow b \qquad R, X_\iota, c \Rightarrow q}{R, X \Rightarrow q} \qquad \begin{array}{l} \iota = (a \to b) \to c \text{ in } X \\ X_\iota = X \setminus \{\iota\} \end{array}$$

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 $R \vdash_{\text{cpl}} q$ iff $R \vdash_{\text{ipl}} q$.

To prove the left premiss of $(L{\to}{\to})$, 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 $(L{\to}{\to})$ becomes:

$$\frac{R, b \to c, X_\iota, A, a \Rightarrow b \qquad R, X_\iota, A, c \Rightarrow q}{R, X, A \Rightarrow q} \qquad \begin{array}{l} \iota = (a \to b) \to c \text{ in } X \\ X_\iota = X \setminus \{\iota\} \end{array}$$

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

Indeed the left premise of the above rule is very similar to the first recursive call `intuitPR(`s`,` X_ι`,` $M \cup \{a\}$`,` b`)` on line 6 of Fig. 3, except that $b \to c$ 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 $\mathrm{R}(s)$ does contain $b \to c$ if $(a \to b) \to c$ is in X: it has been added to s at the beginning of the computation of function `prove` from Fig. 1.

Likewise, the right premise of the above rule is very similar to the second recursive call `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 $(a \to b) \to c$.

Furthermore, the use of the SAT-solver in procedure `intuitPR` has side-effects: let R^0 denote $\mathrm{R}(s)$ at the time when `intuitPR` is called, R^1 denote $\mathrm{R}(s)$ at

the time of the first recursive call (line 6), and R^2 denote $\mathrm{R}(s)$ at the time of the second recursive call (line 9). We have $R^0 \subseteq R^1 \subseteq R^1 \cup \{\bigwedge(A_1 \setminus \{a\}) \to c\} \subseteq R^2$, for a subset A_1 of $M \cup \{a\}$ such that $R^1, X_\iota, A_1 \vdash_{\mathrm{ipl}} b$. This incremental use of the SAT-solver is not reflected in root-first proof search using rule $(L{\to}{\to})$. Hence, rule $(L{\to}{\to})$ has to be generalised to account for these differences.

4.3 A generalised version of $(L{\to}{\to})$

The first generalisation consists in allowing the addition, in the left premise of $(L{\to}{\to})$, 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 `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 LJT and `intuit`, as a call to function `intuitPR(`*s, X, A, q*`)`, if succesful, precisely performs this pruning by outputing a subset A' of A that is sufficient for provability. In that spirit, we present the generalisation of $(L{\to}{\to})$ 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 $(L{\to}{\to})$ 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 $\varphi = \bigwedge(A_1 \setminus \{a\}) \to c$, which in `intuitPR` is added to the SAT-solver on line 8 in Fig. 3. Note that clause φ is a consequence of the original problem. Weakening this newly available assumption in turn means that it no longer subsumes the original implication clause ι, which has to stay in the right premise. The resulting rule is rule (ljt) in Fig. 5.

$$\frac{R \vdash_{\mathrm{cpl}} q}{R, X \Rightarrow q}\ (\mathrm{cpl}) \qquad \frac{R_1, X_1 \vdash_{\mathrm{ipl}} \varphi \qquad \varphi, R_2, X_2 \Rightarrow q}{R_1, R_2, X_1, X_2 \Rightarrow q}\ (\mathrm{cut_{ipl}})$$

$$\frac{R_1,\ b \to c,\ X,\ A_1 \Rightarrow b \qquad R_2,\ \bigwedge(A_1 \setminus \{a\}) \to c,\ X,\ (a \to b) \to c \Rightarrow q}{R_1,\ R_2,\ X,\ (a \to b) \to c \Rightarrow q}\ (\mathrm{ljt})$$

Fig. 5. The calculus $\mathrm{LJT}_{\mathtt{SAT}}$

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 φ 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 $(\mathrm{cut_{ipl}})$ of Fig. 5.

5 The Calculus $\mathbf{LJT}_{\mathtt{SAT}}$

We introduce the calculus $\mathrm{LJT}_{\mathtt{SAT}}$ (LJT with SAT-solver) for sequents of the form $R, X \Rightarrow q$, whose provability in $\mathrm{LJT}_{\mathtt{SAT}}$ is denoted $\vdash_{\mathrm{LJT}_{\mathtt{SAT}}} R, X \Rightarrow q$. It consists of the three rules of Fig. 5. Rule (cpl) has the premise judgment $R \vdash_{\mathrm{cpl}} q$ and the sequent $R, X \Rightarrow q$ as conclusion, with X any set of implicational clauses. The rule can be applied if $R \vdash_{\mathrm{cpl}} q$ holds, as checked by a SAT-solver (hence $\mathrm{LJT}_{\mathtt{SAT}}$). The rule $(\mathrm{cut_{ipl}})$ is a cut rule having the judgment $R_1, X_1 \vdash_{\mathrm{ipl}} \varphi$ as left premise. In the proof-search procedure, whenever we apply $(\mathrm{cut_{ipl}})$, the left-premise is a judgment of the kind $R_0, X_0 \vdash_{\mathrm{ipl}} \varphi$, where the cut formula φ 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 $(L{\to}{\to})$ 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 $\bigwedge(A_1 \setminus \{a\}) \to c$ 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 $(a \to b) \to c$ 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 $\bigwedge(A_1 \setminus \{a\}) \to c$ is the atom c), we get the instance

$$\frac{R,\ b \to c,\ X,\ a \Rightarrow b \qquad R,\ c,\ X,\ (a \to b) \to c \Rightarrow q}{R,\ X,\ (a \to b) \to c \Rightarrow q}\ (\mathrm{ljt})$$

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

To prove the soundness of $\mathrm{LJT}_{\mathtt{SAT}}$, we show that an $\mathrm{LJT}_{\mathtt{SAT}}$-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* $R \vdash_{\mathrm{cpl}} q$ *then* $\vdash_{\mathrm{LJT}} R \Rightarrow q$.

Proof. We proceed via contraposition, so suppose $\nvdash_{\mathrm{LJT}} R \Rightarrow q$. By completeness, there is a Kripke model $\langle W, \leq, r, \vartheta \rangle$ containing a world $w \in W$ such that $w \Vdash \bigwedge R$ and $w \nVdash q$. Now consider any flat clause $\varphi = \bigwedge A_1 \to \bigvee A_2 \in R$. 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 φ is classically true at w. If $A_2 \cap \vartheta(w) = \emptyset$ then reflexivity demands $A_1 \nsubseteq \vartheta(w)$, as otherwise $w \nVdash \varphi$, contradicting our assumption. Again, φ is classically true at w. That is, w by itself is a classical model that also makes $\bigwedge R$ true and q false, so $R \nvdash_{\mathrm{cpl}} q$. By contraposition, if $R \vdash_{\mathrm{cpl}} q$ then $\vdash_{\mathrm{LJT}} R \Rightarrow q$. 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 $\overline{a_1} \vee \cdots \vee \overline{a_n} \vee b_1 \vee \cdots \vee b_m$ is read as the flat clause $a_1 \wedge \cdots \wedge a_n \to b_1 \vee \cdots \vee b_m$, as pointed out by Claessen and Rósen [1]. Removing $q \to \bot$ from the leaves of the resolution tree leaves q at its root, yielding an intuitionistic proof of $R \vdash_{\mathrm{ipl}} q$. Completeness of LJT [2] concludes $\vdash_{\mathrm{LJT}} R \Rightarrow q$.

We prove the main lemma for the soundness of $\mathrm{LJT}_{\mathtt{SAT}}$.

Lemma 3. *If* $\vdash_{\mathrm{LJT}_{\mathtt{SAT}}} R, X \Rightarrow q$ *then* $\vdash_{\mathrm{LJT}} R, X \Rightarrow q$.

Proof. Let $\mathcal{D}$ be an $\mathrm{LJT}_{\mathtt{SAT}}$-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

$$\dfrac{R_1,\, X_1 \vdash_{\mathrm{ipl}} \varphi \qquad \overset{\mathcal{D}_2}{\varphi,\, R_2,\, X_2 \Rightarrow q}}{R_1,\, R_2,\, X_1,\, X_2 \Rightarrow q}\,(\mathrm{cut}_{\mathrm{ipl}})$$

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

$$\dfrac{\overset{\mathcal{D}_1}{R_1,\; b \to c,\; X_\iota,\; A_1 \Rightarrow b} \qquad \overset{\mathcal{D}_2}{R_2,\, \varphi,\, X \Rightarrow q}}{R_1,\, R_2,\, X \Rightarrow q}\,(\mathrm{ljt}) \qquad \begin{array}{l} \iota = (a \to b) \to c \\ X_\iota = X \setminus \{\iota\} \\ \varphi = \bigwedge(A_1 \setminus \{a\}) \to c \end{array}$$

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.

$$\dfrac{\dfrac{\dfrac{\overset{\mathcal{E}_1}{R_1,\; b \to c,\; X_\iota,\; A_1 \Rightarrow b} \qquad \dfrac{}{R_1,\, X_\iota,\, c,\, A_1 \setminus \{a\} \Rightarrow c}\,\mathrm{Ax}}{R_1,\, X,\, A_1 \setminus \{a\} \Rightarrow c}\,L{\to}{\to}}{R_1,\, X \Rightarrow \varphi}\,L{\wedge},\, R{\to} \qquad \overset{\mathcal{E}_2}{R_2,\, \varphi,\, X \Rightarrow q}}{R_1,\, R_2,\, X \Rightarrow q}\,(\mathrm{cut})$$

□

```
1  procedure LJTSatMain(R_0, X_0, q_0)
2  |  // s, R_0, X_0 are global parameters
3  |  s ← newSolver()
4  |  for φ ∈ R_0 do addClause(s, φ)
5  |  for (a → b) → c ∈ X_0 do addClause(s, b → c)
6  |  τ ← LJTSat(R_0, X_0, ∅, q_0, [])
7  |  if τ = K then return K
8  |  else // τ = (D, R, ∅), with R = {φ_1, ..., φ_n}
9  |  |  return the LJT_SAT-derivation
   |  |                                              D
10 |  |  R_0, X_0 ⊢_ipl φ_1  ...  R_0, X_0 ⊢_ipl φ_n   R_0, R, X_0 ⇒ q_0
   |  |  ------------------------------------------------------------- (cut*_ipl)
   |  |                        R_0, X_0 ⇒ q_0
11 |  end
12 end
```

Fig. 6. The procedure `LJTSatMain`

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

Theorem 1 (Soundness of $\mathrm{LJT}_{\mathtt{SAT}}$). $\vdash_{\mathrm{LJT}_{\mathtt{SAT}}} R, X \Rightarrow q$ *implies* $R, X \vdash_{\mathrm{ipl}} q$.

6 Proof-search using $\mathbf{LJT}_{\mathtt{SAT}}$

We present the proof-search procedure based on the calculus $\mathrm{LJT}_{\mathtt{SAT}}$, 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 ι. 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 $(\mathrm{cut}^*_{\mathrm{ipl}})$ also denotes a chain of n successive applications of $(\mathrm{cut}_{\mathrm{ipl}})$.

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

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

procedure `LJTSat`$(R,\ X,\ A,\ q,\ r)$
  $\tau_0 \leftarrow$ `satProve`$(s,\ A,\ q)$
  if $\tau_0 = \mathrm{Yes}(A')$ **then**
    return $(\mathcal{D},\ \mathrm{R}(s),\ A')$ where $\mathcal{D}$ is
    $$\frac{\mathrm{R}(s), A' \vdash_{\mathrm{cpl}} q}{\mathrm{R}(s), X, A' \Rightarrow q}\ (\mathrm{cpl})$$
  else // $\tau_0 = \mathrm{No}(M)$
    $\Theta \leftarrow \emptyset$ // Empty family of Kripke models
    for $\iota = (a \to b) \to c \in X$ *such that* $a \notin M$ *and* $b \notin M$ *and* $c \notin M$ **do**
      $\tau_1 \leftarrow$ `LJTSat`$(R \cup \{b \to c\},\ X_\iota,\ M \cup \{a\},\ b,\ r{::}\iota)$ // where $X_\iota = X \setminus \{\iota\}$
      if $\tau_1 = \mathcal{K}_1$ **then** $\Theta \leftarrow \Theta \uplus (\iota \mapsto \mathcal{K}_1)$ // adding $\mathcal{K}_1$ with index ι
      else // $\tau_1 = (\mathcal{D}_1, R_1, A_1)$
        $\tilde{\varphi} \leftarrow \bigwedge(A_1 \setminus \{a\}) \to c$
        `addClause`$(s,\ \tilde{\varphi})$
        $\tau_2 \leftarrow$ `LJTSat`$(R \cup \{\tilde{\varphi}\},\ X,\ A,\ q,\ r)$
        if $\tau_2 = \mathcal{K}_2$ **then return** $\mathcal{K}_2$
        else // $\tau_2 = (\mathcal{D}_2, R_2, A_2)$
          return $(\mathcal{D},\ R_1 \cup R_2,\ A_2)$ where $\mathcal{D}$ is
          $$\frac{\mathcal{D}_1 \qquad\qquad \mathcal{D}_2}{R, R_1, b \to c, X_\iota, A_1 \Rightarrow b \qquad R, R_2, \tilde{\varphi}, X, A_2 \Rightarrow q}{R,\ R_1,\ R_2,\ X,\ A_2 \Rightarrow q}\ (\mathrm{ljt})$$
          end
        end
      end
    end
    return $\mathrm{Mod}(r, \Theta, M)$
  end
end

Fig. 7. The procedure `LJTSat`

r, then $\mathrm{Mod}(r, \Theta, M)$ is the model $\mathcal{K} = \langle W, \leq, r, \vartheta \rangle$ such that

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

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

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

Lemma 4. *Let $\sigma = R, X \Rightarrow q$ be a sequent, r be a world, M be a classical model and Θ be a family $(\mathcal{K}_\iota)_{\iota \in X_M}$ 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 $\iota \in X_M$, *we have* $\mathcal{K}_\iota \not\models R, X_\iota, M, a \Rightarrow b$.

Then, $\mathrm{Mod}(r, \Theta, M) \not\models \sigma$.

Proof. Let $\mathcal{K}$ be $\mathrm{Mod}(r, \Theta, M) = \langle W, \leq, r, \vartheta \rangle$, whence $r \Vdash M$. 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 $r \nVdash q$. We prove the cases for R and X.

Proof that $r \Vdash R$: Suppose $\varphi \in R$ and assume $\varphi = \bigwedge A_1 \to \bigvee A_2$ with $A_1 \neq \emptyset$. Let $w \in W$ be any world such that $r \leq w$ and $w \Vdash \bigwedge A_1$; we prove $w \Vdash \bigvee A_2$. If $w = r$, we have $r \Vdash \bigwedge A_1$, which implies $A_1 \subseteq M$. Since $M \models \varphi$, we get $M \models \bigvee A_2$, which implies $r \Vdash \bigvee A_2$. If $w \neq r$, w must be in some W_ι for some $\mathcal{K}_\iota = \langle W_\iota, \leq_\iota, r_\iota, \vartheta_\iota \rangle$ in Θ, with $\iota \in X_M$. Thus $r_\iota \leq w$ in $\mathcal{K}$. By (iii), $r_\iota \Vdash R$ in $\mathcal{K}_\iota$ and $\varphi \in R$, so $r_\iota \Vdash \varphi$ in $\mathcal{K}_\iota$. Hence $r_\iota \Vdash \varphi$ in $\mathcal{K}$, and the persistence of $\Vdash$ gives $w \Vdash \varphi$ in $\mathcal{K}$. Since $w \Vdash \bigwedge A_1$, we obtain $w \Vdash \bigvee A_2$. The case $A_1 = \emptyset$ (namely, $\varphi = \bigvee A_2$) is similar.

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

If $c \in M$, then $r \Vdash c$ by construction, hence $w \Vdash c$. If $b \in M$ then $r \Vdash b$ by construction, and we already have $r \Vdash b \to c$, so we get $r \Vdash c$, hence $w \Vdash c$. If $a \in M$ then $r \Vdash a$ and $r \Vdash b \to c$ and $w \Vdash a \to b$, giving $w \Vdash c$. The previous three cases are independent, thus $w \Vdash c$ 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), Θ contains a model $\mathcal{K}_\iota = \langle W_\iota, \leq_\iota, r_\iota, \vartheta_\iota \rangle$ such that $\mathcal{K}_\iota \not\models R, X_\iota, M, a \Rightarrow b$. Thus $r_\iota \nVdash a \to b$ in $\mathcal{K}_\iota$, hence $r_\iota \nVdash a \to b$ in $\mathcal{K}$. By reverse persistence $r \nVdash a \to b$ (in $\mathcal{K}$), which implies $w \neq r$. There is also a model $\mathcal{K}_{\iota'} = \langle W', \leq', r', \vartheta' \rangle$ of Θ such that $w \in W'$, hence $r \leq r' \leq w$ (in $\mathcal{K}$). We need to cover the two cases $\iota = \iota'$ and $\iota \neq \iota'$:

1. If $\iota' = \iota$, by (iii) $r_\iota \Vdash a$ in $\mathcal{K}_\iota$, thus $w \Vdash a$ (in $\mathcal{K}$). Since $r \Vdash b \to c$, we have $w \Vdash b \to c$. Then $w \Vdash a \to b$ gives $w \Vdash c$.
2. If $\iota' \neq \iota$ then $\iota \in X_{\iota'}$, hence $r' \Vdash \iota$ in $\mathcal{K}_{\iota'}$, whence it follows that $w \Vdash \iota$ (in $\mathcal{K}$). Our initial assumption that $w \Vdash a \to b$ gives $w \Vdash c$. □

In Fig. 6 we define the proof-search function `LJTSatMain` such that `LJTSatMain(`R_0`,` X_0`,` q_0`)` returns either an $\mathrm{LJT}_{\mathtt{SAT}}$-derivation of $\sigma_0 = R_0, X_0 \Rightarrow q_0$ or a counter-model $\mathcal{K}$ for σ_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:

$$H_0 \;=\; \{\, b \to c \mid (a \to b) \to c \in X_0 \,\} \qquad V_0 \;=\; \{\, p \in V \mid p \text{ occurs in } \sigma_0 \,\}$$
$$\mathrm{M}_{/V_0}(R) \;=\; \{\; M \subseteq V_0 \mid M \models R \,\} \quad \text{for any set } R \text{ of flat clauses}$$

The call `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 `LJTSat` defined in Fig. 7. A call `LJTSat(`R`,` X`,` A`,` q`,` r`)` performed during the computation of the main call `LJTSatMain(`R_0`,` X_0`,` q_0`)` has the following specification.

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

Output Properties (OP):

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

In (IA), $\mathrm{R}(s)$ refers to the clauses in the SAT-solver s at the beginning of the computation of `LJTSat(`R,X,A,q,r`)`; in (OP), $\mathrm{R}(s)$ is the set of clauses in s at the end of the computation. Note that (OP) implies that the call `LJTSat(`R,X,A,q`)` terminates. To prove the correctness of `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. *$R, X, A \vdash_{\mathrm{ipl}} b$ implies $R, X, (a \to b) \to c \vdash_{\mathrm{ipl}} \bigwedge(A \setminus \{a\}) \to c$.*

Proof. Let $A' = A \setminus \{a\}$. If $R, X, A \vdash_{\mathrm{ipl}} b$, then $R, X, A' \vdash_{\mathrm{ipl}} a \to b$, which implies $R, X, (a \to b) \to c, A' \vdash_{\mathrm{ipl}} c$, hence $R, X, (a \to b) \to c \vdash_{\mathrm{ipl}} \bigwedge A' \to c$. □

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

$$(R', X') \prec (R, X) \quad \text{iff} \quad (X' \subset X) \text{ or } \big(X' = X \text{ and } \mathrm{M}_{/V_0}(R') \subset \mathrm{M}_{/V_0}(R)\big)$$

Since the sets X and $\mathrm{M}_{/V_0}(R)$ are finite, the relation $\prec$ is well-founded, hence we can prove correctness of `LJTSat` (Lemma 6) by induction on $\prec$.

Lemma 6. *If a call `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:

(MIH): if $(R', X') \prec (R, X)$, then the lemma holds for `LJTSat(`R',X',A',q',r'`)`;
(Inv): for every $\varphi \in \mathrm{R}(s)$, we have $R_0, X_0 \vdash_{\mathrm{ipl}} \varphi$.

At the start of the computation (Inv) holds by (IA). Let τ_0 be the value computed at line 2. If $\tau_0 = \mathrm{Yes}(A')$, then the triple $(\mathcal{D}, \mathrm{R}(s), A')$ is returned at line 4, with $\mathcal{D}$ defined at line 5; by definition of `satProve`, it holds that $\mathrm{R}(s), A' \vdash_{\mathrm{cpl}} q$ and $A' \subseteq A$, hence (OP) holds. Otherwise, since $R \cup H_0 \subseteq \mathrm{R}(s)$, we have:

(P0) $\tau_0 = \mathrm{No}(M)$ and $M \models R \cup H_0 \cup A$ and $M \not\models q$.

Without loss of generality, we can assume $M \subseteq V_0$, namely $M \in \mathrm{M}_{/V_0}(R)$. If, for every $(a \to b) \to c \in X$, the loop condition at line 8 does not hold, then the loop at lines 8–21 is skipped and the model $\mathrm{Mod}(r, \emptyset, M)$ 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 τ_1 and τ_2 are the values computed at lines 9 and 14 respectively, and $\tilde{\varphi} = \bigwedge(A_1 \setminus \{a\}) \to c$ is defined at line 12:

(P1) $\tau_1 = (\mathcal{D}_1, R_1, A_1)$ or $\tau_1 = \mathcal{K}_1$ where:
- $R_1 \subseteq \mathrm{R}(s)$ and $A_1 \subseteq M \cup \{a\}$;
- $\mathcal{D}_1$ is an $\mathrm{LJT}_{\mathtt{SAT}}$-derivation of $R, R_1, b \to c, X_\iota, A_1 \Rightarrow b$;
- $\mathcal{K}_1$ has root $r{::}\iota$ and worlds are ordered by the prefix order on sequences;
- $\mathcal{K}_1 \not\models R, H_0, X_\iota, M, a \Rightarrow b$.

(P2) $\tau_2 = (\mathcal{D}_2, R_2, A_2)$ or $\tau_2 = \mathcal{K}_2$ where:
- $R_2 \subseteq \mathrm{R}(s)$ and $A_2 \subseteq A$;
- $\mathcal{D}_2$ is an $\mathrm{LJT}_{\mathtt{SAT}}$-derivation of $R, R_2, \tilde{\varphi}, X, A_2 \Rightarrow q$;
- $\mathcal{K}_2$ has root r and worlds are ordered by the prefix order on sequences;
- $\mathcal{K}_2 \not\models R, \tilde{\varphi}, H_0, X, A \Rightarrow q$.

Let us consider the first iteration of the loop and let $\iota = (a \to b) \to c \in X$ be the selected clause (hence, $a \notin M$, $b \notin M$ and $c \notin M$). The call to `LJTSat` at line 9 satisfies (IA). Since $X_\iota \subset X$, we have $(R \cup \{b \to c\}, X_\iota) \prec (R, X)$. By (MIH) τ_1 satisfies (OP); this proves (P1). Note that (OP) guarantees that (Inv) holds after the computation of τ_1. At line 13, $\tilde{\varphi}$ is added to s; we check that (Inv) is preserved, namely $R_0, X_0 \vdash_{\mathrm{ipl}} \tilde{\varphi}$. By (P1) and Soundness of $\mathrm{LJT}_{\mathtt{SAT}}$ (Theorem 1), $R, R_1, b \to c, X_\iota, A_1 \vdash_{\mathrm{ipl}} b$ and, by Lemma 5, we get $R, R_1, b \to c, X \vdash_{\mathrm{ipl}} \tilde{\varphi}$, hence $R, R_1, b \to c, X_0 \vdash_{\mathrm{ipl}} \tilde{\varphi}$. Since $R \cup R_1 \cup \{b \to c\} \subseteq \mathrm{R}(s)$, from (Inv) it follows that $R_0, X_0 \vdash_{\mathrm{ipl}} \tilde{\varphi}$. The call to `LJTSat` at line 14 matches (IA). To apply (MIH), we have to check that:

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

Clearly, $\mathrm{M}_{/V_0}(R \cup \{\tilde{\varphi}\}) \subseteq \mathrm{M}_{/V_0}(R)$; to conclude the proof of (P3), we show that the inclusion is strict. Note that $M \in \mathrm{M}_{/V_0}(R)$ (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 $M \notin \mathrm{M}_{/V_0}(R \cup \{\tilde{\varphi}\})$; 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, τ_1 is not a model, namely $\tau_1 = (\mathcal{D}_1, R_1, A_1)$. If τ_2 is a model $\mathcal{K}_2$, then $\mathcal{K}_2$ is returned at line 15 and (OP) follows from (P2). Otherwise, $\tau_2 = (\mathcal{D}_2, R_2, A_2)$ and $(\mathcal{D}, R_1 \cup R_2, A_2)$ is returned at line 17, where $\mathcal{D}$ is the $\mathrm{LJT}_{\mathtt{SAT}}$-derivation displayed at line 18; accordingly (OP) holds. Finally, let us assume that, at every iteration, τ_1 is a model. Since X is finite, the loop eventually ends and the model $\mathrm{Mod}(r, \Theta, M)$ is returned at line 22. At that point, Θ has been completed into a family $(\mathcal{K}_\iota)_{\iota \in X_M}$ such that for every ι in X_M, $\mathcal{K}_\iota$ has root $r{::}\iota$ and its worlds are ordered by the prefix order on sequences, by (P1). $\mathrm{Mod}(r, \Theta, M)$ has root r and also uses the prefix order on sequences. So in order to prove (OP), we only have to check that $\mathrm{Mod}(r, \Theta, M)$ 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 σ', M and Θ satisfy the assumptions of Lemma 4. Since $A \subseteq M$, $\mathrm{Mod}(r, \Theta, M) \not\models \sigma'$ and (OP) holds. □

By Lemma 6, we get:

Theorem 2 (Correctness of `LJTSatMain`). `LJTSatMain`(R_0, X_0, q_0) *returns either an* $\mathrm{LJT}_{\mathtt{SAT}}$*-derivation of* $\sigma_0 = R_0, X_0 \Rightarrow q_0$ *or a counter-model for* σ_0.

Proof. Consider the call `LJTSat`(R_0,X_0,$\emptyset$,q_0,[ ]) at line 6. When `LJTSat` is called, $\mathrm{R}(s) = R_0 \cup H_0$. Let $b \to c \in H_0$; since X_0 contains a formula of the kind $(a \to b) \to c$ and $(a \to b) \to c \vdash_{\mathrm{ipl}} b \to c$, it follows that $R_0, X_0 \vdash_{\mathrm{ipl}} b \to c$. Thus, the call to `LJTSat` satisfies (IA); by Lemma 6, the returned value τ satisfies (OP). If τ 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 $\tau = (\mathcal{D}, R, \emptyset)$, where $\mathcal{D}$ is an $\mathrm{LJT}_{\mathtt{SAT}}$-derivation $\mathcal{D}$ of $R_0, R, X_0 \Rightarrow q_0$ and $R_0, X_0 \vdash_{\mathrm{ipl}} \varphi$, for every $\varphi \in R$. Accordingly, the returned derivation, displayed at line 10, is an $\mathrm{LJT}_{\mathtt{SAT}}$-derivation of σ_0. □

As a consequence, we get:

Theorem 3 ($\mathrm{LJT}_{\mathtt{SAT}}$ completeness). $R, X \vdash_{\mathrm{ipl}} q$ *implies* $\vdash_{\mathrm{LJT}_{\mathtt{SAT}}} R, X \Rightarrow q$.

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

$$\dfrac{\dfrac{\dfrac{}{R_0,\ X',\ b,\ b \to c,\ a \Rightarrow b} \qquad \dfrac{}{R_0,\ X',\ c,\ b \Rightarrow c}}{R_0,\ X_0,\ b \Rightarrow c}\, L{\to}{\to}}{R_0,\ X_0 \Rightarrow b \to c}\, R{\to}$$

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

$$\dfrac{R_0, X_0 \vdash_{\mathrm{ipl}} \varphi_1 \quad \ldots \quad R_0, X_0 \vdash_{\mathrm{ipl}} \varphi_n \qquad \overset{\mathcal{D}_1}{R_0,\ \varphi_1, \ldots, \varphi_n,\ X_\iota,\ A_1 \Rightarrow b}}{R_0,\ X_0,\ A_1 \Rightarrow b}\, (\mathrm{cut}^*_{\mathrm{ipl}})$$

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

If $a \notin A_1$, then $\tilde{\varphi} = \bigwedge A_1 \to c$ and $\mathcal{D}_{\tilde{\varphi}}$ is the LJT-derivation

$$\dfrac{\dfrac{\dfrac{\mathcal{E}_1}{R_0, X_0, A_1 \Rightarrow b} \qquad \dfrac{\dfrac{}{b,\ b \to c,\ a \Rightarrow b}\,\mathrm{Ax} \qquad \dfrac{}{b,\ c \Rightarrow c}\,\mathrm{Ax}}{b,\ (a \to b) \to c \Rightarrow c}\, L{\to}{\to}}{R_0, X_0, A_1 \Rightarrow c}\,(\mathrm{cut})}{R_0, X_0 \Rightarrow \bigwedge A_1 \to c}\, L{\wedge},\ R{\to}$$

If $A_1 = \emptyset$ (hence $\tilde{\varphi} = c$) the bottom applications of $L\wedge$ and $R\to$ are crossed out. Let $a \in A_1$ and $A_1' = A_1 \setminus \{a\}$. Thus, $\tilde{\varphi} = \bigwedge A' \to c$ and $\mathcal{D}_{\tilde{\varphi}}$ is the derivation

$$\dfrac{\dfrac{\dfrac{\mathcal{E}_1}{R_0, X_0, A_1', a \Rightarrow b}}{R_0, X_0, A_1' \Rightarrow a \to b}\, R\to \qquad \dfrac{\dfrac{\dfrac{}{b,\, b \to c,\, a \Rightarrow b}\,\mathrm{Ax}}{a \to b,\, b \to c,\, a \Rightarrow b}\, L0\to \qquad \dfrac{}{a \to b,\, c \Rightarrow c}\,\mathrm{Ax}}{a \to b,\, (a \to b) \to c \Rightarrow c}\, L{\to}{\to}}{\dfrac{R_0, X_0, A_1' \Rightarrow c}{R_0, X_0 \Rightarrow \bigwedge A_1' \to c}\, L\wedge,\, R\to}\,(\mathrm{cut})$$

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

By the above discussion, we can enhance the procedures `LJTSatMain` and `LJTSat` so that, whenever a flat clause φ is added to the SAT-solver, an LJT-derivation $\mathcal{D}_\varphi$ of $R_0, X_0 \Rightarrow \varphi$ is stored. Let us assume that `LJTSatMain(`R_0`,` X_0`,` q_0`)` returns an $\mathrm{LJT}_{\mathtt{SAT}}$-derivation $\mathcal{D}$ of $\sigma_0 = R_0, X_0 \Rightarrow q_0$ Proceeding as in the proof of Lemma 3, we can exploit the derivations $\mathcal{D}_\varphi$ to translate $\mathcal{D}$ into an LJT-derivation of σ_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 `intuit` regarding incremental SAT-solvers.

The rule $(L{\to}{\to})$ 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 $(a \to b) \to c$ true, to make all members of Γ true and to make δ false. If $w \Vdash c$ then we have the right premise. Else $w \nVdash c$ and therefore, $w \nVdash a \to b$. But that means that there exists a $v \geq w$ such that $v \Vdash \Gamma$ and $v \Vdash a$ and $v \nVdash b$, implying that $v \Vdash b \to c$, which is the left premise.

We have shown that our simpler recursive version of `intuit` can be reconciled with this semantic view if we generalise the rule $(L{\to}{\to})$ 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 $\mathrm{LJT}_{\mathtt{SAT}}$ 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

1. Koen Claessen and Dan Rosén. SAT modulo intuitionistic implications. In *LPAR*, volume 9450 of *Lecture Notes in Computer Science*, pages 622–637. Springer, 2015.
2. Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *J. Symb. Log.*, 57(3):795–807, 1992.
3. Roy Dyckhoff and L Pinto. Implementation of a loop-free method for construction of counter-models for intuitionistic propositional logic, 1996. University of St Andrews Report CS/96/8.
4. Mahfuza Farooque, Stéphane Graham-Lengrand, and Assia Mahboubi. A bisimulation between DPLL(T) and a proof-search strategy for the focused sequent calculus. In Alberto Momigliano, Brigitte Pientka, and Randy Pollack, editors, *Proceedings of the 2013 International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2013)*. ACM Press, September 2013.
5. Mauro Ferrari, Camillo Fiorentini, and Guido Fiorino. Contraction-free linear depth sequent calculi for intuitionistic propositional logic with the subformula property and minimal depth counter-models. *J. Autom. Reasoning*, 51(2):129–149, 2013.
6. Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische Zeitschrift*, 39:176–210, 1934.
7. Gerhard Gentzen. Investigations into logical deduction i. In M. Szabo, editor, *The collected papers of Gerhard Gentzen*. North-Holland, Amsterdam, 1969.
8. Stéphane Graham-Lengrand. Psyche: a proof-search engine based on sequent calculus with an LCF-style architecture. In Didier Galmiche and Dominique Larchey-Wendling, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 22th International Conference, TABLEAUX 2013, Nancy, France, September 16-19, 2013. Proceedings*, volume 8123 of *Lecture Notes in Computer Science*, pages 149–156. Springer, 2013.
9. Stéphane Graham-Lengrand. *Polarities & Focussing: a journey from Realisability to Automated Reasoning*. Habilitation thesis, Université Paris-Sud, 2014.
10. Jianwen Li, Shufang Zhu, Geguang Pu, Lijun Zhang, and Moshe Y. Vardi. Sat-based explicit ltl reasoning and its application to satisfiability checking. *Formal Methods in System Design*, https://doi.org/10.1007/s10703-018-00326-5, 2019.
11. Robert Nieuwenhuis, Alberto Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.
12. Thomas Raths, Jens Otten, and Christoph Kreitz. The ILTP problem library for intuitionistic logic. *Journal of Automated Reasoning*, 38(1):261–271, Apr 2007.
13. Richard Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 07 1979.
14. A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Camb. Univ. Press, 2ed edition, 2000.
15. G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.
16. Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.