

# Efficient SAT-based Proof Search in Intuitionistic Propositional Logic

Camillo Fiorentini

Department of Computer Science  
University of Milan, Italy

CADE 2021

On line

# Motivations

- In 2015, Claessen and Rosén introduced `intuit`, an efficient decision procedure for `IPL` (Intuitionistic Propositional Logic) based on a Satisfiability Modulo Theories (SMT) approach.

K. Claessen and D. Rosén. SAT Modulo Intuitionistic Implications, LPAR 2015

The `intuit` decision procedure exploits an incremental SAT-solver.

- On the top of `intuit`, we have implemented `intuitR` (`intuit` with `Restart`), obtaining significant advantages.

# intuit: specification

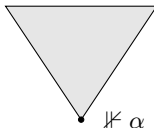
- Input

A formula  $\alpha$

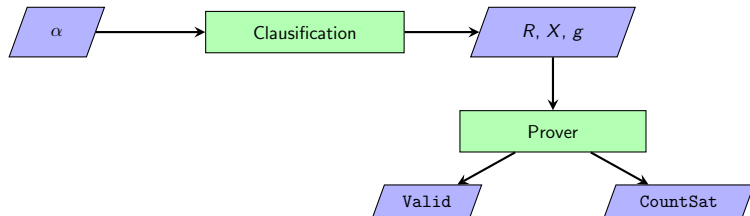
- Output

- **Valid** if  $\alpha$  is intuitionistically valid
- **CountSat (counter-satisfiable)** if  $\alpha$  is not intuitionistically valid

Thus, there exists a **countermodel** for  $\alpha$ , namely:  
a Kripke model such that at its root  $\alpha$  is not forced



# intuit: architecture



Two main modules:

- **Classification**

Pre-processing of the input formula  $\alpha$ :

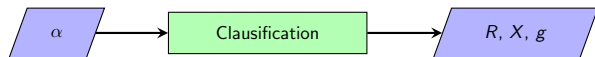
- ✓ the validity of  $\alpha$  is reduced to the validity of a sequents of the kind  $R, X \Rightarrow g$ , where  $R, X$  and  $g$  have a simple form.

- **Prover**

Decide the validity of  $R, X \Rightarrow g$ .

Most of the computation is performed by an incremental SAT-solver.

## intuit: clausification



- $R$  is a set of **flat clauses**  $\varphi$  of the form

$$\varphi := \bigwedge A_1 \rightarrow \bigvee A_2 \quad A_1, A_2: \text{sets of atoms}$$

Flat clauses are actively used in classical reasoning (SAT-solver)

- $X$  is a set of **implication clauses**  $\lambda$  of the form

$$\lambda := (a \rightarrow b) \rightarrow c \quad a, b, c: \text{atoms}$$

- $g$  is an atom

We also assume that  $(R, X)$  is  **$\rightarrow$ -closed**:

$$(a \rightarrow b) \rightarrow c \in X \implies b \rightarrow c \in R$$

# intuit: clausification



$R$  : set of flat clauses  $\varphi$   
 $X$  : set of impl. clauses  $\lambda$   
 $\varphi := \bigwedge A_1 \rightarrow \bigvee A_2$   
 $\lambda := (a \rightarrow b) \rightarrow c$   
 $a, b, c, g$  : atoms  
 $A_1, A_2$  : sets of atoms

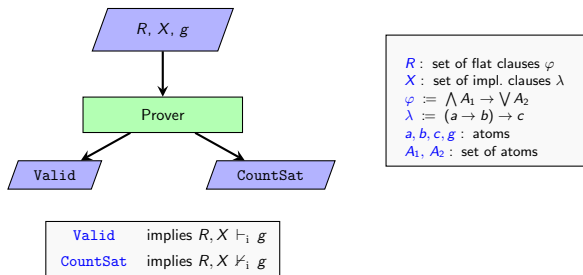
- (1)  $\alpha \in \text{IPL}$  iff  $R, X \vdash_i g$        $\vdash_i$  : intuitionistic provability
- (2) Let  $\mathcal{K}$  be a countermodel for the sequent  $R, X \Rightarrow g$ , namely:
- at the root  $r$  of  $\mathcal{K}$ , all the formulas in  $R$  and  $X$  are forced and  $g$  is not forced.

Then,  $\mathcal{K}$  is a countermodel for  $\alpha$ .



Clausification is performed by applying standard rewriting steps.

# intuit: prover



- Decision algorithm: a variant of the  $\text{DPLL}(\mathcal{T})$  procedure.
- Most of the computation is performed by an **incremental SAT-solver**:

✓ **Learning mechanism:**

during the computation, flat clauses  $\varphi$  of the form  $\varphi := \bigwedge A \rightarrow b$  are learned and permanently added to the solver (**learned clauses**)

# intuit

- Implemented in Haskell
- `intuit` outperforms the best state-of-the-art provers for IPL

## But

- YES/NO procedure (no informative output)
- The procedure seems to be far away from the traditional techniques for deciding IPL validity.

## Recent work (joint paper with S. Graham-Lengrand and R. Goré):

*A Proof-Theoretic Perspective on SMT-Solving for Intuitionistic Propositional Logic, Tableaux 2019.*

We unveil a close and surprising connection between the `intuit` decision procedure based on SMT and the known proof-theoretic methods.

- the decision procedure mimics a standard root-first proof search strategy for the sequent calculus  $\text{LJT}_{\text{SAT}}$ , a variant of Dyckhoff's calculus LJ<sub>T</sub>



# The calculus $\text{LJT}_{\text{SAT}}$

We consider **r-sequents** (**reduced sequents**) of the form:

$$R, X \Rightarrow g$$

$R$ : set of fat clauses  $\bigwedge A_1 \rightarrow \bigvee A_2$   
 $X$ : set of impl. clauses  $(a \rightarrow b) \rightarrow c$   
 $g$ : atom

The calculus  $\text{LJT}_{\text{SAT}}$  is **sound and complete** w.r.t. IPL.

$$\vdash_{\text{LJT}_{\text{SAT}}} R, X \Rightarrow g \quad \text{iff} \quad R, X \vdash_i g$$

**Axiom rule**

$$\frac{R \vdash_c g}{R, X \Rightarrow g} \text{cpl}_0 \quad \vdash_c: \text{classical provability}$$

The soundness follows by this well-known property:

- $R \vdash_c g$  iff  $R \vdash_i g$

To check the condition  $R \vdash_c g$  we use the SAT-solver.

# The calculus $\text{LJT}_{\text{SAT}}$

## Rule for left $\rightarrow$

Let us consider Dyckhoff rule adapted to r-sequents:

$$\frac{R, X, \textcolor{red}{b} \rightarrow \textcolor{red}{c}, \textcolor{red}{a} \Rightarrow \textcolor{red}{b} \quad R, X, \textcolor{red}{c} \Rightarrow g}{R, X, (\textcolor{red}{a} \rightarrow \textcolor{red}{b}) \rightarrow \textcolor{red}{c} \Rightarrow g}$$

We are assuming  $b \rightarrow c \in R$ , thus it can be rewritten as

$$\frac{R, X, \textcolor{red}{a} \Rightarrow \textcolor{red}{b} \quad R, X, \textcolor{red}{c} \Rightarrow g}{R, X, (\textcolor{red}{a} \rightarrow \textcolor{red}{b}) \rightarrow \textcolor{red}{c} \Rightarrow g}$$

## Generalization

- multiplicative contexts (split  $R$  into  $R_1, R_2$  and  $X$  into  $X_1, X_2$ )
- Replace the atom  $\textcolor{red}{a}$  in the left premise with a **set of atoms  $\textcolor{red}{A}$**

$$\frac{R_1, X_1, \textcolor{red}{A} \Rightarrow \textcolor{red}{b} \quad R_2, X_2, \textcolor{red}{???} \Rightarrow g}{R_1, R_2, X_1, X_2, (\textcolor{red}{a} \rightarrow \textcolor{red}{b}) \rightarrow \textcolor{red}{c} \Rightarrow g}$$

# The calculus $\text{LJT}_{\text{SAT}}$

Rule **ljt** for left implication:

$$\frac{R_1, X_1, A \Rightarrow b \quad \varphi, R_2, X_2, (a \rightarrow b) \rightarrow c \Rightarrow g}{R_1, R_2, X_1, X_2, (a \rightarrow b) \rightarrow c \Rightarrow g} \quad \begin{array}{l} A \text{ is any set of atoms} \\ \varphi = \bigwedge(A \setminus \{a\}) \rightarrow c \end{array}$$

In the right premise:

- the main formula  $(a \rightarrow b) \rightarrow c$  is kept
- the flat clause  $\varphi$  is added

Intuitively:

- $R_1, R_2$  are the clauses in the SAT-solver
- $\varphi$  is the **learned clause** to be added to the SAT-solver.

*Proof of soundness*

$$\begin{array}{c} \lambda = (a \rightarrow b) \rightarrow c \\ \frac{\frac{R_1, X_1, A \vdash_i b}{R_1, X_1, A \setminus \{a\} \vdash_i a \rightarrow b} \quad R \rightarrow \quad \frac{}{\lambda, a \rightarrow b \vdash_i c} \text{MP}}{\frac{R_1, X_1, \lambda, A \setminus \{a\} \vdash_i c}{R_1, X_1, \lambda \vdash_i \varphi} \quad R \rightarrow} \text{cut} \\ \frac{R_1, X_1, \lambda \vdash_i \varphi \quad \varphi, R_2, X_2, \lambda \vdash_i g}{R_1, R_2, X_1, X_2, \lambda \vdash_i g} \text{cut} \end{array}$$

# The calculus $\text{LJT}_{\text{SAT}}$

$$\frac{R \vdash_c g}{R, X \Rightarrow g} \text{cpl}_0$$

$$\frac{R_1, X_1, A \Rightarrow b \quad \varphi, R_2, X_2, (a \rightarrow b) \rightarrow c \Rightarrow g}{R_1, R_2, X_1, X_2, (a \rightarrow b) \rightarrow c \Rightarrow g} \text{ljt} \quad \varphi = \bigwedge(A \setminus \{a\}) \rightarrow c$$

We also need a **cut rule**

$$\frac{R_1, X_1 \vdash_i \varphi \quad \varphi, R_2, X_2 \Rightarrow q}{R_1, R_2, X_1, X_2 \Rightarrow q} \text{cut}$$

In [Tableuax 2019], we formalize the **intuit** decision procedure so that, given an r-sequent  $\sigma = R, X \Rightarrow g$ , it outputs either a derivation of  $\sigma$  in  $\text{LJT}_{\text{SAT}}$  or a countermodel for  $\sigma$ .

**The end of the story?**

# Beyond intuit

We have enhanced the Haskell `intuit` code by implementing the derivation/countermodel extraction procedures

We experimented some unexpected and weird phenomena:

- derivations are often convoluted and contain applications of the cut rule which cannot be trivially eliminated.
- countermodels have lots of redundancies.

To overcome these issues:

- we introduce the sequent calculus  $C^{\rightarrow}$ , a lightweight variant of  $LJT_{SAT}$
- we redesign the `intuit` decision procedure, using  $C^{\rightarrow}$  instead of  $LJT_{SAT}$

We call the new prover `intuitR` (`intuit` with `Restart`)

# The calculus $C^{\rightarrow}$

The calculus  $C^{\rightarrow}$  only consists of two rules:

- Axiom rule

Same axiom rule as in  $LJT_{SAT}$

$$\frac{R \vdash_c g}{R, X \Rightarrow g} \text{cpl}_0$$

- Left implication

A simplified version of the rule  $ljt$  of  $LJT_{SAT}$  (rule  $\text{cpl}_1$  in next slide).

There is no need for cut rule.

# The calculus $C^{\rightarrow}$ : rule for left $\rightarrow$

Let us consider the **additive** variant of rule ljt:

$$\frac{R, X, A \Rightarrow b \quad \varphi, R, X, (a \rightarrow b) \rightarrow c \Rightarrow g}{R, X, (a \rightarrow b) \rightarrow c \Rightarrow g} \quad \varphi := \bigwedge (A \setminus \{a\}) \rightarrow c$$

We require that the left premise has a trivial proof:

$$\frac{\frac{R, A \vdash_c b}{R, X, A \Rightarrow b} \text{cpl}_0 \quad \varphi, R, X, (a \rightarrow b) \rightarrow c \Rightarrow g}{R, X, (a \rightarrow b) \rightarrow c \Rightarrow g}$$

We get the rule **cpl**<sub>1</sub>:

$$\frac{R, A \vdash_c b \quad \varphi, R, X, (a \rightarrow b) \rightarrow c \Rightarrow g}{R, X, (a \rightarrow b) \rightarrow c \Rightarrow g} \text{cpl}_1 \quad \varphi := \bigwedge (A \setminus \{a\}) \rightarrow c$$

# The calculus $C^\rightarrow$ : derivations

Derivations of  $C^\rightarrow$  have a plain linear structure (one branch):

$$\begin{array}{c}
 \frac{R_{m-1}, A_{m-1} \vdash_c b_{m-1} \quad \frac{R_m \vdash_c g}{R_m, X \Rightarrow g}}{R_{m-1}, X \Rightarrow g} \lambda_{m-1} \\
 \vdots \\
 \frac{R_1, A_1 \vdash_c b_1 \quad \frac{\overbrace{\varphi_1, R_1}^{R_2}, X \Rightarrow g}{\lambda_1}}{\lambda_0} \\
 \frac{R_0, A_0 \vdash_c b_0 \quad \frac{\overbrace{\varphi_0, R_0}^{R_1}, X \Rightarrow g}{\lambda_0}}{R_0, X \Rightarrow g} \lambda_0
 \end{array}$$

$$\lambda_k := (a_k \rightarrow b_k) \rightarrow c_k \in X$$

$$\varphi_k := \bigwedge (A_k \setminus \{a_k\}) \rightarrow c_k$$

$$R_{k+1} := R_k \cup \{\varphi_k\}$$

Rule names are omitted, we display the main formulas  $\lambda_k$  of  $\text{cpl}_1$  applications.



# The calculus $C^{\rightarrow}$ : derivations

$$\begin{array}{c}
 \frac{R_{m-1}, A_{m-1} \vdash_c b_{m-1} \quad \frac{R_m \vdash_c g}{R_m, X \Rightarrow g}}{R_{m-1}, X \Rightarrow g} \lambda_{m-1} \\
 \vdots \\
 \frac{R_0, A_0 \vdash_c b_0 \quad \frac{R_1, A_1 \vdash_c b_1 \quad R_2, X \Rightarrow g}{R_1, X \Rightarrow g} \lambda_1}{R_0, X \Rightarrow g} \lambda_0
 \end{array}$$

Note that the sets  $R_k$  are increasing

$$R_0 \subseteq R_1 \subseteq R_2 \subseteq \dots \subseteq R_m$$

Accordingly, to check the conditions

$$R_0, A_0 \vdash_c b_0, \quad R_1, A_1 \vdash_c b_1, \quad \dots \quad R_m \vdash_c g$$

we can use an **incremental SAT-solver**

- $R_0, R_1, \dots$ : clauses stored in the SAT-solver
- $A_0, A_1, \dots, b_0, b_1, \dots$ : local variables

# Bottom-up proof search

## Goal

Search for a derivation of  $R_0, X \Rightarrow g$  in  $C^\rightarrow$ .

Bottom-up proof search procedure by exploiting an incremental SAT-solver to check classical validity.

- (1) Add the clauses  $R_0$  to the SAT-solver and check:

$$R_0 \vdash_c g?$$

- If  $R_0 \vdash_c g$ , then build the derivation:

$$\frac{R_0 \vdash_c g}{R_0, X \Rightarrow g} \text{cpl}_0$$

- Otherwise, choose  $\langle \lambda_0, A_0 \rangle$  such that:

$$\lambda_0 = (a_0 \rightarrow b_0) \rightarrow c_0 \in X \qquad R_0, A_0 \vdash_c b_0$$

and apply:

$$\frac{R_0, A_0 \vdash_c b_0 \quad \overbrace{\varphi_0, R_0}^{R_1}, X \Rightarrow g}{R_0, X \Rightarrow g} \lambda_0 \qquad \varphi_0 = \bigwedge (A_0 \setminus \{a_0\}) \rightarrow c_0$$

# Bottom-up proof search

(2) We continue with the sequent

$$R_1, X \Rightarrow g \quad R_1 = R_0 \cup \{\varphi_0\}$$

Add  $\varphi_0$  to the SAT-solver and check

$$R_1 \vdash_c g?$$

- If  $R_1 \vdash_c g$ , then:

$$\frac{R_0, A_0 \vdash_c b_0 \quad \frac{R_1 \vdash_c g}{R_1, X \Rightarrow g}}{R_0, X \Rightarrow g} \lambda_0$$

- Otherwise, choose  $\langle \lambda_1, A_1 \rangle$  such that

$$\lambda_1 = (a_1 \rightarrow b_1) \rightarrow c_1 \in X \quad R_1, A_1 \vdash_c b_1$$

$$\frac{R_0, A_0 \vdash_c b_0 \quad \frac{R_1, A_1 \vdash_c b_1 \quad \overbrace{\varphi_1, R_1, X \Rightarrow g}^{R_2}}{R_1, X \Rightarrow g} \lambda_1}{R_0, X \Rightarrow g} \lambda_0$$

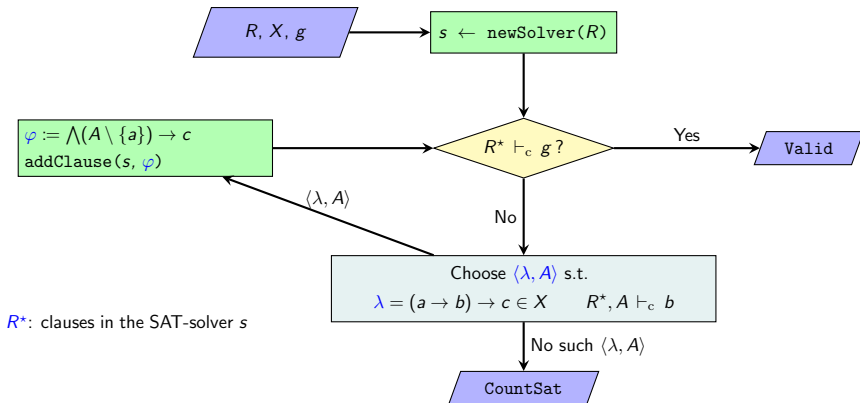
# Bottom-up proof search

## Input Assumptions

$R$  : set of flat clauses  $\varphi = \bigwedge A_1 \rightarrow \bigvee A_2$   
 $X$  : set of impl. clauses  $\lambda = (a \rightarrow b) \rightarrow c$   
 $g$  : atom

## Output Properties

Valid      implies  $R, X \vdash_i g$   
CountSat   implies  $R, X \not\vdash_i g$



# Bottom-up proof search

## Problem

A blind choice of  $\langle \lambda, A \rangle$  might lead to non-termination

## Example

Current sequent:

$$b \rightarrow c, (a \rightarrow b) \rightarrow c \Rightarrow g$$

Selected  $\langle \lambda, A \rangle$ :

$$\lambda = (a \rightarrow b) \rightarrow c \qquad A = \{b\}$$

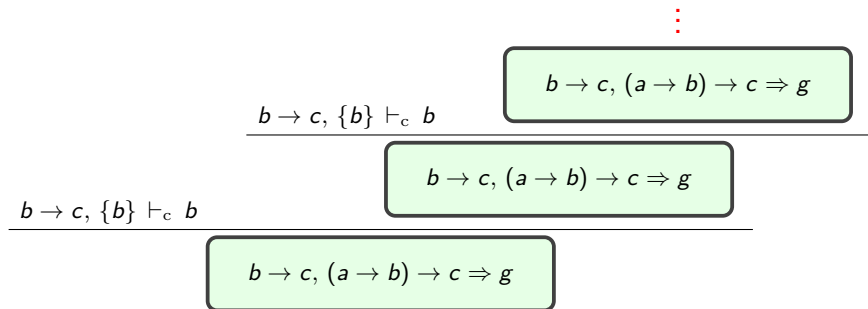
Application of rule  $\text{cpl}_1$ :

$$\frac{b \rightarrow c, \overbrace{\{b\}}^A \vdash_c b \quad \varphi, b \rightarrow c, (a \rightarrow b) \rightarrow c \Rightarrow g}{b \rightarrow c, (a \rightarrow b) \rightarrow c \Rightarrow g}$$

$$\varphi = \bigwedge (A \setminus \{a\}) \rightarrow c = b \rightarrow c$$

# Bottom-up proof search

We get a **non-terminating loop!**



Can we get an informed choice of  $\langle \lambda, A \rangle$ ?

# Bottom-up proof search

Current goal: prove the sequent  $\sigma_k = R_k, X \Rightarrow g$

$$\frac{\frac{R_{k-1}, A_{k-1} \vdash_c b_{k-1} \quad R_k, X \Rightarrow g}{R_{k-1}, X \Rightarrow g} \lambda_{k-1} \quad \vdots}{\frac{R_0, A_0 \vdash_c b_0 \quad R_1, X \Rightarrow g}{R_0, X \Rightarrow g} \lambda_0} R_0 \subseteq R_1 \subseteq \dots \subseteq R_k$$

We search for a **countermodel**  $\mathcal{K}$  for  $\sigma_k$ .

- If we find  $\mathcal{K}$  then:

$\mathcal{K}$  is a countermodel for  $R_0, X \Rightarrow g$  (indeed,  $R_0 \subseteq R_k$ )

We conclude **CountSat** (namely,  $R_0, X \not\models_i g$ )

- Otherwise

From the failure, we **learn** the proper choice of  $\langle \lambda_k, A_k \rangle$

# Countermodels

A **Kripke model** can be seen as a **set of interpretations**.

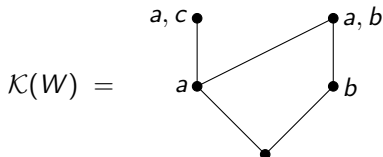
Let  $W$  be a finite set of interpretations with minimum  $M_0$  (namely:  $M_0 \subseteq M$ , for every  $M \in W$ ).

Then,  $\mathcal{K}(W)$  is the Kripke model such that:

- The set of worlds is  $W$ ;
- $M_1 \leq M_2$  (in  $\mathcal{K}(W)$ ) iff  $M_1 \subseteq M_2$ ;
- the root of  $\mathcal{K}(W)$  is  $M_0$ ;
- $M \Vdash p$  iff  $p \in M$ .

## Example

$$W = \{ \emptyset, \{a\}, \{b\}, \{a, c\}, \{a, b\} \}$$





# Countermodels

Let  $W$  be a finite set of interpretations.

We introduce the following **realizability relation**  $\triangleright_W$  ( $M \in W$ ):

$$M \triangleright_W (a \rightarrow b) \rightarrow c \quad \text{iff} \quad a \in M \text{ or } b \in M \text{ or } c \in M \text{ or} \\ \exists M' \in W \text{ s.t.} \\ (M \subset M') \text{ and } a \in M' \text{ and } b \notin M'$$

## Main property of $\triangleright_W$

Let  $W$  be a finite set of interpretations with minimum  $M_0$ .

Then,  $\mathcal{K}(W)$  is a **countermodel** for  $R, X \Rightarrow g$  iff:

- $g \notin M_0$ ;
- for every  $M \in W$ :

$$\begin{array}{ll} M \models R & \text{namely: } M \models \varphi, \forall \varphi \in R \\ M \triangleright_W X & \text{namely: } M \triangleright_W \lambda, \forall \lambda \in X \end{array}$$

We use the property to build countermodels.

# Countermodels

Let  $R, X \Rightarrow g$  be the current sequent to be proved in the main loop.

- If  $R \vdash_c g$ , then there exists a derivation of  $R, X \Rightarrow g$ .
- Otherwise, the SAT-solver yields a model  $M$  s.t.  $M \models R$  and  $M \not\models g$ .

We set

$$W = \{M\}$$

We try to turn  $\mathcal{K}(W)$  into a countermodel for  $R, X \Rightarrow g$  by running a **saturation process**:

- ✓ we add to  $W$  the worlds needed to fulfill the main property (inner loop).

## Key point

- Suppose that there exists a pair  $\langle w, \lambda \rangle$  such that

$$w \in W \quad \lambda = (a \rightarrow b) \rightarrow c \in X \quad w \not\models_W \lambda$$

Then, we search for an interpretation  $w'$  s.t.:

$$w \subseteq w' \quad \text{and} \quad w' \models R \quad \text{and} \quad a \in w' \quad \text{and} \quad b \notin w'$$

- ✓ If such a  $w'$  exists, we add  $w'$  to  $W$  and we continue to saturate
- ✓ Otherwise, there is no countermodel for  $R, X \Rightarrow g$ .

# Countermodels

How can we search for an interpretation  $w'$  s.t.:

$$w \subseteq w' \quad \text{and} \quad w' \models R \quad \text{and} \quad a \in w' \quad \text{and} \quad b \notin w' ?$$

Ask to the SAT-solver:

$$R, w, a \not\models_c b ?$$

Possible outcomes

- Yes(A)

This means that

$$A \subseteq w \cup \{a\} \quad \text{and} \quad R, A \vdash_c b$$

Thus,  $R, w, a \vdash_c b$  and such a  $w'$  does not exist.

Accordingly, the construction of the countermodel fails.

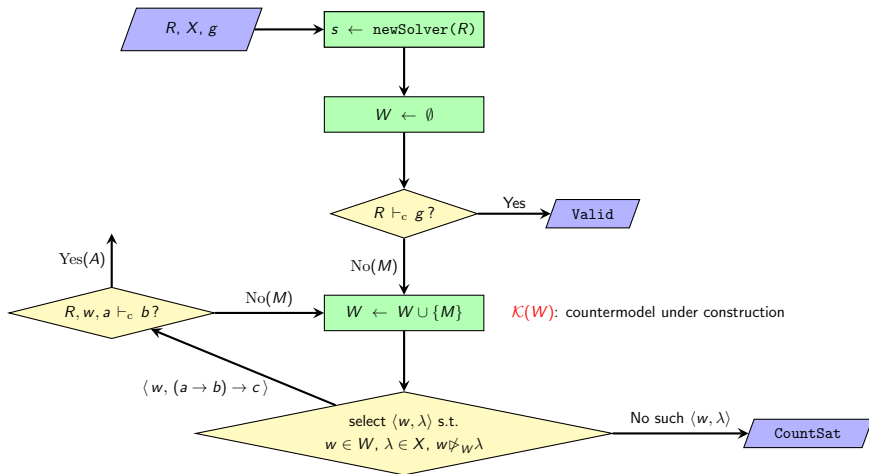
- No( $M'$ )

This means that

$$M' \models R \cup w \cup \{a\} \quad \text{and} \quad M' \not\models b$$

We set  $w' = M'$ .

# Countermodels



# Countermodels

Suppose that, after having chosen the pair  $\langle w, \lambda \rangle$ , the inner loop ends with  $\text{Yes}(A)$ , meaning that  $R, A \vdash_c b$

Then,  $\lambda$  and  $A$  are the main formula and the local assumptions to be used:

$$\frac{R, A \vdash_c b \quad \varphi, R, X \Rightarrow g}{R, X \Rightarrow g} \quad \lambda \quad \begin{array}{l} \lambda \in X \\ \varphi = \bigwedge (A \setminus \{a\}) \rightarrow c \end{array}$$

- The learned clause  $\varphi$  is added to the SAT-solver
- We empty the set  $W$  (namely, we discard the current countermodel) and we perform a new iteration of the main loop (**Restart**)
  - ✓ At each restart, we execute the procedure from scratch.

However, at each restart the SAT-solver is more powerful (we have added a new learned clause  $\varphi$  to it).

We call the obtained prover **intuitR** (intuit with Restart).

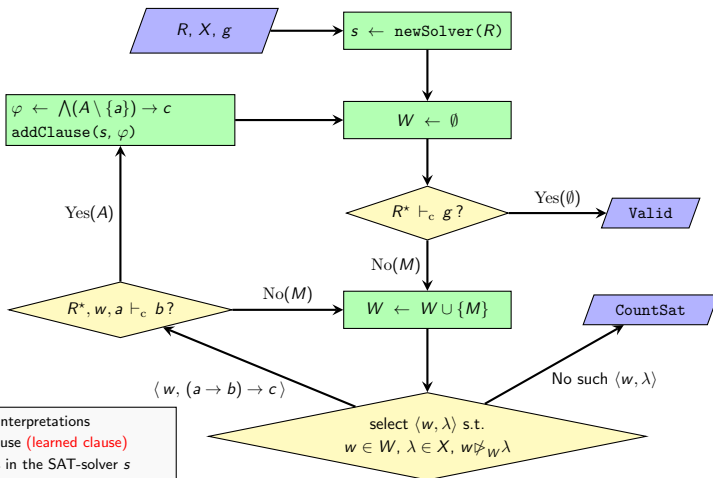
# intuitR

## Input Assumptions

$R$ : set of flat clauses  $\varphi = \bigwedge A_1 \rightarrow \bigvee A_2$   
 $X$ : set of impl. clauses  $\lambda = (a \rightarrow b) \rightarrow c$   
 $g$ : atom

## Output Properties

**Valid** implies  $R, X \vdash_1 g$   
**CountSat** implies  $R, X \not\vdash_1 g$



# intuitR implementation

- We have implemented `intuitR` in Haskell on the top of `intuit`; we have added some useful features (e.g., trace of computations, construction of derivations/countermodels).
- As in `intuit`, we exploit the module `MiniSat`, a Haskell bundle of the MiniSat SAT-solver (but in principle we can use any incremental SAT-solver).
- The `intuitR` implementation can be downloaded at

<https://github.com/cfiorentini/intuitR>.

## intuitR vs intuit

- The proof search procedure of `intuitR` has a plain and intuitive presentation, consisting of two nested loops.
- Derivations have a linear structure, formalized by the calculus  $C^{\rightarrow}$ . Basically, a derivation in  $C^{\rightarrow}$  is a cut-free derivation in  $LJT_{SAT}$  (the calculus of `intuit`) having only one branch.
- The countermodels obtained by `intuitR` are in general smaller than the ones obtained by `intuit`, since at every restart the model is reset.
- We have replicated the experiments performed for `intuit`: `intuitR` outperforms `intuit`.



# intuit vs intuitR: derivations

## Example 1 (Valid formula)

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

$$\eta_{ij} = p_i \leftrightarrow p_j \quad \gamma = p_1 \wedge p_2 \wedge p_3$$

first instance of problem class SYJ201 from the [ILTP library](#)

## Clausification

$R_0$ : 17 flat clauses,  $\chi$ : 6 implication clauses

## intuitR

14 call to the SAT-solver (6 Yes, 8 No), 6 Restart

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

# intuit vs intuitR: derivations

intuit

14 calls to the SAT-solver (7 Yes, 6 No)

$$\begin{array}{c}
 \frac{R_0, \dots \vdash_c p_1}{R_0, \dots \Rightarrow p_1} \quad \frac{\frac{R_1, \dots \vdash_c p_2}{R_1, \dots \Rightarrow p_2} \quad \frac{R_2, \dots \vdash_c p_2}{R_2, \dots \Rightarrow p_2}}{R_1, \dots \Rightarrow p_2} \lambda_3 \quad \frac{\frac{R_3, \dots \vdash_c p_1}{R_3, \dots \Rightarrow p_1} \quad \frac{\frac{\frac{R_4, \dots \vdash_c p_3}{R_4, \dots \Rightarrow p_3} \quad \frac{R_5, \dots \vdash_c p_3}{R_5, \dots \Rightarrow p_3}}{R_4, \dots \Rightarrow p_3} \lambda_4 \quad \frac{R_6 \vdash_c \tilde{g}}{R_6, \dots \Rightarrow \tilde{g}} \lambda_2}{R_3, \dots \Rightarrow \tilde{g}} \lambda_1 \quad \frac{R_3, \dots \Rightarrow p_1 \quad R_3, \dots \Rightarrow \tilde{g}}{R_3, \dots \Rightarrow \tilde{g}} \lambda_0 \\
 \frac{R_0, \dots \Rightarrow p_2 \quad R_3, \dots \Rightarrow \tilde{g}}{R_0, \varphi_0, \varphi_1, \varphi_4, X \Rightarrow \tilde{g}} \text{cut (3 times)} \\
 \hline
 R_0, X \Rightarrow \tilde{g}
 \end{array}$$

Cuts are needed to drip out the extra learned clauses  $\varphi_0, \varphi_1, \varphi_4$ .

Actually, one can prove that each learned clause  $\varphi_k$  satisfies

$$R_0, X \vdash_i \varphi_k$$

# intuit vs intuitR: countermodels

## Example 2 (CountSat formula)

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

$$\eta_{ij} = p_i \leftrightarrow p_j \quad \gamma = p_1 \wedge p_2 \wedge p_3 \wedge p_4$$

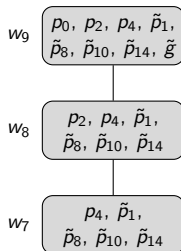
first instance of problem class SYJ207 from the **ILTP library**

## Clausification

$R_0$ : 24 flat clauses, **X**: 9 implication clauses

intuitR

14 call to the SAT-solver (4 Yes, 10 No), 4 Restart

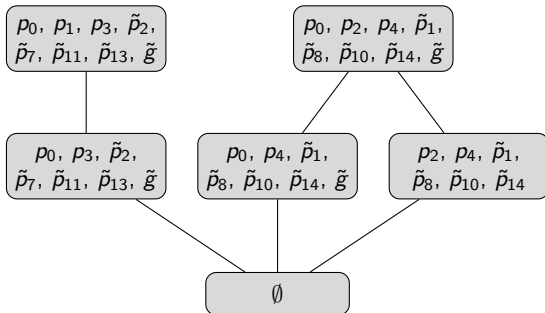


$\mathcal{K}(\{w_7, w_8, w_9\})$

# intuit vs intuitR: countermodels

intuit

31 calls to the SAT-solver (24 No, 7 Yes)



# intuit vs intuitR: experiments

We compare `intuitR` with the state-of-the-art provers for IPL:

- `intuit`
- `fCube` [Ferrari et al. LPAR 2010]  
Standard tableaux calculus with simplification rules
- `intHistGC` [Goré et al., IJCAR 2014]  
Sequent calculus with histories, dependency directed backtracking for global caching

## Benchmarks

- 1200 problems (498 Valid and 702 CountSat)
- timeout: 600 seconds.

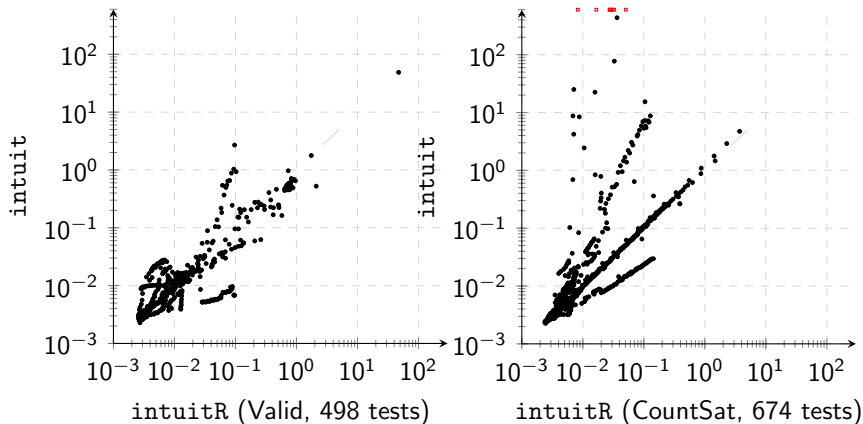
## Outcome

- `intuitR` solve more problems than its competitors
- In families SYJ201 (Valid formulas) and SYJ207 (CountSat formulas) of ILTP library, `intuitR` outperforms its rivals
- In all the other cases (except 3 families), `intuitR` is comparable to the best prover (which is `intuit` in most cases).

# intuit vs intuitR: experiments

Class (# problems)	intuitR	intuit	fCube	intHistGC
SYJ201(50)	50 (2.259)	50 (11.494)	50 (259.776)	50 (39.466)
SYJ202(38)	10 <sup>*</sup> (49.265)	10 <sup>*</sup> (50.658)	9 <sup>*</sup> (176.984)	6 <sup>*</sup> (324.673)
SYJ203(50)	50 (0.250)	50 (0.335)	50 (1.671)	50 (0.293)
SYJ204(50)	50 (0.442)	50 (0.477)	50 (0.972)	50 (0.203)
SYJ205(50)	50 (0.500)	50 (0.730)	50 (1.317)	50 (4.129)
SYJ206(50)	50 (0.303)	50 (0.348)	50 (0.759)	50 (0.112)
SYJ207(50)	50 (2.291)	50 (109.919)	50 (138.546)	50 (1014.476)
SYJ208(38)	38 (5.225)	38 (5.479)	29 <sup>*</sup> (2.755)	38 (497.715)
SYJ209(50)	50 (0.226)	50 (0.278)	50 (1.690)	50 (0.254)
SYJ210(50)	50 (0.272)	50 (0.252)	50 (0.988)	50 (0.288)
SYJ211(50)	50 (0.462)	50 (1.251)	50 (1.073)	50 (63.686)
SYJ212(50)	50 (0.669)	42 <sup>*</sup> (587.794)	50 (2.698)	50 (1.624)
EC(100)	100 (2.738)	100 (0.821)	100 (6.183)	100 (0.651)
negEC(100)	100 (3.614)	100 (1.116)	100 (13.733)	100 (5.807)
cross(4)	4 (0.100)	4 (0.097)	4 (3.417)	2 <sup>*</sup> (0.005)
jm_cross(4)	4 (0.120)	4 (0.090)	4 (5.404)	3 <sup>*</sup> (4.324)
jm_lift(3)	3 (0.170)	3 (0.133)	3 (6.847)	2 <sup>*</sup> (0.028)
lift(3)	3 (0.119)	3 (0.102)	3 (6.494)	2 <sup>*</sup> (0.012)
mapf(4)	4 (0.187)	4 (0.400)	4 (446.921)	3 <sup>*</sup> (0.043)
portia(100)	100 (32.878)	100 (22.596)	100 (3255.818)	100 (3200.135)
negportia(100)	100 (7.956)	100 (8.309)	98 <sup>*</sup> (3826.011)	100 (28.289)
negportia2(100)	100 (8.081)	100 (8.411)	98 <sup>*</sup> (1264.103)	100 (3212.293)
nishimura2(28)	28 (9.784)	28 (12.285)	27 <sup>*</sup> (141.326)	28 (7.616)
<b>Unsolved</b>	28	36	43	38

## intuit vs intuitR: experiments



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

# Conclusions

- `intuitR` can be extended to deal with some superintuitionistic logics.

## Key idea:

- ✓ if the countermodel under construction is not a model of the logic, the inner loop fails, and we run a new iteration of the main loop.
- ✓ From the failure, we learn new clauses to add to the SAT-solver (corresponding to instances of the axiom schema of the logic).
- Other generalizations suggested in [C&R, LPAR 2015] (modal logics, fragments of first-order logic) seem to be more challenging.
- The `intuitR` implementation and other additional material (e.g., the omitted proofs, a detailed report on experiments) can be downloaded at

<https://github.com/cfiorentini/intuitR>.