Two simulations about DPLL(T)
Mahfuza Farooque, Stéphane Lengrand, Assia Mahboubi

To cite this version:
Mahfuza Farooque, Stéphane Lengrand, Assia Mahboubi. Two simulations about DPLL(T). 2012.<hal-00690044>

HAL Id: hal-00690044
https://hal.archives-ouvertes.fr/hal-00690044
Submitted on 23 Apr 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Two simulations about DPLL(\(T\))

Mahfuza Farooque\(^1\), Stéphane Lengrand\(^1,2\) and Assia Mahboubi\(^3\)

\(^1\) CNRS
\(^2\) Ecole Polytechnique
\(^3\) Microsoft Research - INRIA Joint Centre

Project PSI: “Proof Search control in Interaction with domain-specific methods”
ANR-09-JCJC-0006

23rd April 2012

Abstract

In this paper we relate different formulations of the DPLL(\(T\)) procedure.

The first formulation is that of [NOT06] based on a system of rewrite rules, which we denote DPLL(\(T\)).

The second formulation is an inference system of [Tin02], which we denote LK_{DPLL}(\(T\)).

The third formulation is the application of a standard proof-search mechanism in a sequent calculus LK'\(^p\)(\(T\)) introduced here.

We formalise an encoding from DPLL(\(T\)) to LK_{DPLL}(\(T\)) that was, to our knowledge, never explicitly given and, in the case where DPLL(\(T\)) is extended with backjumping and Lemma learning, never even implicitly given.

We also formalise an encoding from LK_{DPLL}(\(T\)) to LK'\(^p\)(\(T\)), building on Ivan Gazeau’s previous work: we extend his work in that we handle the “-modulo-Theory” aspect of SAT-modulo-theory, by extending the sequent calculus to allow calls to a theory solver (seen as a blackbox). We also extend his work in that we handle advanced features of DPLL such as backjumping and Lemma learning, etc.

Finally, we refine the approach by starting to formalise quantitative aspects of the simulations: the complexity is preserved (number of steps to build complete proofs). Other aspects remain to be formalised (non-determinism of the search / width of search space).

Contents

1 Encoding DPLL(\(T\)) in LK_{DPLL}(\(T\)) 2
1.1 Preliminaries: LK_{DPLL}(\(T\)) and its properties ................................. 2
1.2 DPLL(\(T\)) with backtracking ...................................................... 5
1.3 DPLL(\(T\)) with backjumping and Lemma learning ............................ 8

2 Encoding LK_{DPLL}(\(T\)) in LK'\(^p\)(\(T\)) 11
2.1 Preliminaries: System LK'\(^p\)(\(T\)) ............................................. 11
2.2 Simulation .................................................................. 12
1 Encoding DPLL(\(T\)) in LK_{DPLL}(\(T\))

In this section we encode DPLL(\(T\)) in LK_{DPLL}(\(T\)).

Note that there exist different variants of DPLL(\(T\)). We first consider the basic version which is equipped with backtracking. This formalises ideas presented in [Tin02]. Then we enhance the encoding to the enhanced version of DPLL(\(T\)) with backjumping, a generalised version of backtracking.

The main gap between DPLL(\(T\)) and an inference system such as LK_{DPLL}(\(T\)) is the fact that a (successful) DPLL(\(T\)) run is a rewrite sequence finishing with the state UNSAT, while a (successful) proof-search run is (/ produces) a proof tree. Roughly speaking, the DPLL(\(T\)) procedure implements the depth-first search of the corresponding tree.

1.1 Preliminaries: LK_{DPLL}(\(T\)) and its properties

Definition 1 (The system LK_{DPLL}(\(T\))) Clauses are finite disjunctions of literals considered up to commutativity and associativity. We will denote them \(C, C_0, C_1\) etc.; the empty clause will be denoted by \(\bot\). The cardinality of a clause \(C\) is denoted |\(C\)|.

Finite sets of clauses, e.g. \(\{C_1, \ldots, C_n\}\), will be denoted \(\phi, \phi_0\), etc. By |\(\phi\)| we denote the sum of the sizes of the clauses in \(\phi\). By \(\text{lit}(\phi)\) we denote the set of literals that appear in \(\phi\) or whose negations appear in \(\phi\).

Given a theory \(T\) the system LK_{DPLL}(\(T\)), given in Figure 1, is an inference system on sequences of the form \(\Delta, \phi \vdash \tau\), where \(\Delta\) is a set of literals (e.g. \(\{l_1, \ldots, l_n\}\)).

\[
\frac{\Delta, l \vdash \phi \quad \Delta, l \not\vdash \tau}{\Delta \vdash \phi \not\vdash \tau} \quad \text{(Split) where } l \in \text{lit}(\phi), \Delta, l \not\vdash \tau \text{ and } \Delta, l \not\vdash \tau
\]

\[
\frac{\Delta, \phi \vdash \bot}{\Delta \vdash \phi \not\vdash \tau} \quad \text{(Empty)}
\]

\[
\frac{\Delta, \phi \vdash \bot}{\Delta \vdash \phi \not\vdash \tau} \quad \text{(Assert) where } \Delta, l \not\vdash \tau \text{ and } \Delta, l \not\vdash \tau
\]

\[
\frac{\Delta, \phi \vdash \bot}{\Delta \vdash \phi \not\vdash \tau} \quad \text{(Subsume) where } \Delta, l \vdash \tau \quad \frac{\Delta, \phi \vdash \bot}{\Delta \vdash \phi \not\vdash \tau} \quad \text{(Resolve) where } \Delta, l \vdash \tau
\]

Figure 1: System LK_{DPLL}(\(T\))

The Assert rule models the fact that every literal occurring as a unit clause in the current clause set must be satisfied for the whole clause set to be satisfied. The Split is mainly used to branch the proof tree from the DPLL rewrite sequence system, This rule corresponds to the decomposition in smaller subproblems of the DPLL method. This rule is the only don’t know non—deterministic rule of the calculus. The Resolve rule removes from a clause all literals whose complement has been asserted (which corresponds to generating the simplified clause by unit resolution and the discarding the clause by backward subsumption). The Subsume rule removes from the clauses that contain an asserted literal (because all of these clause will be satisfied in any model in which the asserted literal is true). To close the branch of a proof tree we use the empty rule is in the calculus just for convenience and could be removed with no loss of completeness. It models the fact that a derivation can be terminated as soon as the empty clause is derived. We do not consider that the model is consistent and satisfiable.

Definition 2 (Semantical entailment) \(\Delta \models_T C\) is a semantical notion of entailment for a particular theory \(T\), i.e. every \(T\)-model of \(\Delta\) is a \(T\)-model of \(C\). A theory lemma is a clause \(C\) such that \(\emptyset \models_T C\).

Lemma 1 (Weakening 1) The following rule is size-preserving admissible in LK_{DPLL}(\(T\))

\[
\frac{\Delta, \phi \vdash \tau}{\Delta, \phi, C \vdash \tau}
\]

Proof: By induction on \(\Delta, \phi \vdash \tau\). □
Definition 3 (Consequences) For every set $\Delta$ of literals $l$, let $\text{Sat}(\Delta) = \{ l | \Delta \models \tau \}$ and $\text{Sat}_\phi(\Delta) = \text{Sat}(\Delta) \cap \text{lit}(\phi)$.

Remark 2 If $\text{Sat}(\Delta) = \text{Sat}(\Delta')$ then $\Delta \models \tau$ iff $\Delta' \models \tau$.

Lemma 3 (Weakening 2) The following rule is size-preserving admissible in $\text{LK}_{\text{DPLL}}(T)$

\[
\frac{\Delta, \phi \vdash \tau}{\Delta', \phi \vdash \tau} \quad \text{Sat}_\phi(\Delta) \subseteq \text{Sat}_\phi(\Delta')
\]

Proof: By induction on the derivation of $\Delta, \phi \vdash \tau$:

Resolve

\[
\frac{\Delta, \phi, C \vdash \tau}{\Delta, l \vdash \tau}
\]

We assume $\text{Sat}_{\phi, lVC}(\Delta) \subseteq \text{Sat}_{\phi, lVC}(\Delta')$

from which we get $\text{Sat}_{\phi, C}(\Delta) \subseteq \text{Sat}_{\phi, C}(\Delta')$, so we can apply the induction hypothesis to construct

\[
\frac{\Delta', \phi, C \vdash \tau}{\Delta', l \vdash \tau}
\]

The side-condition is a consequence of the assumption $\text{Sat}_{\phi, lVC}(\Delta) \subseteq \text{Sat}_{\phi, lVC}(\Delta')$.

Subsume

\[
\frac{\Delta, l, \phi, l \vdash \tau}{\Delta, \phi, l \vdash \tau}
\]

We assume $\text{Sat}_{\phi, lVC}(\Delta) \subseteq \text{Sat}_{\phi, lVC}(\Delta')$

from which we get $\text{Sat}_{\phi, l}(\Delta) \subseteq \text{Sat}_{\phi, l}(\Delta')$, so we can apply the induction hypothesis to construct

\[
\frac{\Delta', \phi, l \vdash \tau}{\Delta', l \vdash \tau}
\]

The side-condition is a consequence of the assumption $\text{Sat}_{\phi, lVC}(\Delta) \subseteq \text{Sat}_{\phi, lVC}(\Delta')$.

Assert

\[
\frac{\Delta, l, \phi, l \vdash \tau}{\Delta, l, \phi \vdash \tau}
\]

We assume $\text{Sat}_{\phi, l}(\Delta) \subseteq \text{Sat}_{\phi, l}(\Delta')$

from which we get $\text{Sat}_{\phi, l}(\Delta, l) \subseteq \text{Sat}_{\phi, l}(\Delta', l)$.

- If $\Delta' \models \tau$, then $\text{Sat}(\Delta', l) = \text{Sat}(\Delta')$, so we have $\text{Sat}_{\phi, l}(\Delta, l) \subseteq \text{Sat}_{\phi, l}(\Delta')$. The induction hypothesis then gives $\Delta', \phi, l \vdash \tau$.
- If $\Delta' \models \tau$, then we construct

\[
\begin{array}{l}
\Delta', \phi, \bot \vdash \tau \\
\frac{\Delta', \phi, \bot \vdash \tau}{\Delta', \phi, l \vdash \tau} \quad \text{Empty}
\end{array}
\]

Split

\[
\frac{\Delta, l, \phi, l \vdash \tau}{\Delta, \phi, l \vdash \tau}
\]

We assume $\text{Sat}_{\phi, lVC}(\Delta) \subseteq \text{Sat}_{\phi, lVC}(\Delta')$ from which we get both

\[
\text{Sat}_{\phi, lVC}(\Delta, l) \subseteq \text{Sat}_{\phi, lVC}(\Delta', l) \quad \text{and} \quad \text{Sat}_{\phi, lVC}(\Delta, l^+) \subseteq \text{Sat}_{\phi, lVC}(\Delta', l^+).
\]

- If $\Delta' \models \tau$, then $\text{Sat}(\Delta') = \text{Sat}(\Delta, l)$, so we have $\text{Sat}_{\phi, lVC}(\Delta, l) \subseteq \text{Sat}_{\phi, lVC}(\Delta')$. The induction hypothesis then gives $\Delta', \phi, l \vdash \tau$.
- If $\Delta' \models \tau$, then $\text{Sat}(\Delta') = \text{Sat}(\Delta', l^+)$, so we have $\text{Sat}_{\phi, lVC}(\Delta, l^+) \subseteq \text{Sat}_{\phi, lVC}(\Delta')$. The induction hypothesis then gives $\Delta', \phi, l \vdash \tau$.
- If $\Delta' \not\models_T l$ and $\Delta' \not\models_T l^\perp$: the induction hypothesis on both premises gives $\Delta', l\phi, l \lor C \models_T$ and $\Delta', l^\perp \phi, l \lor C \models_T$, and we can conclude 
\[
\frac{\Delta', l^\perp \phi, l \lor C \models_T \quad \Delta', l \phi, l \lor C \models_T}{\Delta' \models_T l \text{ and } \Delta' \not\models_T l^\perp}
\]

Empty Straightforward.

\[\square\]

Lemma 4 (Invertibility of Resolve) \enspace \textit{Resolve is size-preserving invertible in LK\textsubscript{DPLL}(T).}

\textbf{Proof:} By induction on the derivation of $\Delta; \phi, C \lor l \models_T$ we prove $\Delta; \phi, C \models_T$ (with the assumption $\Delta, l \models_T$):

\textbf{Resolve} easily permutes with other instances of \textit{Resolve} and with instances of \textit{Subsume}.

\textbf{Assert} The side-condition of the rule guarantees that the literal added to the model, say $l'$, is different from $l$:

\[
\frac{\Delta, l'; \phi', l', C \lor l \models_T}{\Delta; \phi', l', C \lor l \models_T} \Delta, l^\perp \models_T \text{ and } \Delta, l' \not\models_T
\]

We can construct

\[
\frac{\Delta, l^\perp \phi, l', C \models_T}{\Delta; \phi', l', C \models_T} \Delta, l' \models_T \not\models_T \text{ and } \Delta, l' \not\models_T
\]

whose premise is proved by the induction hypothesis.

\textbf{Split} $\Delta, l^\perp \phi, C \lor l \models_T \Delta, l^\perp \phi, C \lor l \models_T \Delta, l' \models_T$ $l' \in \text{lit}(\phi, C \lor l)$ and $\Delta, l^\perp \models_T$ and $\Delta, l' \not\models_T$

We can construct

\[
\frac{\Delta, l^\perp \phi, C \lor l \models_T \Delta, l^\perp \phi, C \lor l \models_T}{\Delta; \phi, C \models_T} l' \in \text{lit}(\phi, C) \text{ and } \Delta, l^\perp \not\models_T \text{ and } \Delta, l' \not\models_T
\]

whose branches are closed by using the induction hypothesis. The side-condition $l' \in \text{lit}(\phi, C)$ is satisfied because $l \neq l'$.

Empty Straightforward.

\[\square\]

We now introduce a new system $\text{LK}_{\text{DPLL}^+}(T)$ which is an extended version of $\text{LK}_{\text{DPLL}}(T)$ with $\textit{Weakening1}$, $\textit{Weakening2}$ and the $\textit{Inverted Resolve}$. By the previous lemmas, a sequent derivable in $\text{LK}_{\text{DPLL}^+}(T)$ is derivable in $\text{LK}_{\text{DPLL}}(T)$.

\begin{center}
\begin{array}{ccc}
\Delta; \phi \models_T & \Delta; \phi \models_T & \Delta; \phi, C \models_T \\
\Delta; \phi, C \models_T & \Delta; \phi \models_T & \Delta; \phi, l \lor C \models_T \\
\end{array}
\end{center}

Figure 2: System $\text{LK}_{\text{DPLL}^+}(T)$

Definition 4 (Size of proof-trees in $\text{LK}_{\text{DPLL}^+}(T)$) The size of proof-trees in $\text{LK}_{\text{DPLL}^+}(T)$ is defined as the size of trees in the usual sense, but not counting the occurrences of $\textit{Weakening1}$, $\textit{Weakening2}$ or the $\textit{Inverted Resolve}$ rules.\footnote{For that reason, dashed lines will be used for the occurrences of those inference rules.}

Remark 5 The size-preserving admissibility results of those three rules in $\text{LK}_{\text{DPLL}}(T)$ entails that a proof-tree in $\text{LK}_{\text{DPLL}^+}(T)$ of size $n$, can be transformed into a proof-tree in $\text{LK}_{\text{DPLL}}(T)$ of size at most $n$.

Lemma 6 If $\Delta \models_T \neg C$ then there is a proof-tree concluding $\Delta; C, \phi \models_T$ of size at most $|\phi| + 1$. 
Remark 7

In Fig. 4 we define the interpretation of a model as a.

\[ \Delta \vdash l \text{ for } l \in \text{lit}(\Delta) \]

We can therefore construct

\[
\begin{align*}
\text{Empty} & \quad \Delta; l, \phi \vdash_T \\
\text{Resolve} & \quad \Delta; C, \phi \vdash_T
\end{align*}
\]

1.2 \textit{DPLL}(T) with backtracking

In this section we describe the basic \textit{DPLL}(T) procedure [NOT06], and its encoding into \textit{LKDPLL}(T).

Definition 5 (Basic \textit{DPLL}(T)) Models are defined by the following grammar:

\[ \Delta ::= () \mid \Delta, l^d \mid \Delta, l \]

where \( l \) ranges over literals, and \( l^d \) is an annotated literal called decision literal.

The basic \textit{DPLL}(T) procedure rewrites \textit{states} of the form \( \Delta [\phi] \), with the following rewriting rules:

- **Fail:**
  \( \Delta [\phi], C \Rightarrow \text{UNSAT} \), with \( |\Delta| \vdash \neg C \) and there is no decision literal in \( \Delta \).

- **Decide:**
  \( \Delta [\phi] \Rightarrow \Delta, l^d [\phi] \)
  \( \triangleq (l \not\in \Delta, l^d \not\in \Delta, l \not\in \phi \text{ or } l^d \not\in \phi) \).

- **Backtrack:**
  \( \Delta_1, l^d, \Delta_2 [\phi], C \Rightarrow \Delta_1, l^d [\phi], C \)
  \( \triangleq (|\Delta_1, l, \Delta_2| \vdash \neg C \text{ and no decision literal is in } \Delta_2) \).

- **Unit propagation:**
  \( \Delta [\phi], C \cup l \Rightarrow \Delta, l [\phi], C \cup l \)
  \( \triangleq (|\Delta| \vdash \neg C, l \not\in \Delta, l^d \not\in \Delta) \).

- **Theory Propagate:**
  \( \Delta [\phi] \Rightarrow \Delta, l [\phi] \)
  \( \triangleq (|\Delta| \vdash_T l, l \in \text{lit}(\phi) \text{ and } l \not\in \Delta, l^d \not\in \Delta) \).

where \( |\Delta| \) denotes the result of erasing the annotations on decision literals, an operation defined in Fig. 3.

\[
\begin{array}{c|c|c|c}
|\Delta| & := & () \\
|\Delta, l| & := & |\Delta|, l \\
|\Delta, l^d| & := & |\Delta|, l^d \\
\end{array}
\]

Figure 3: Erasing annotations

We now proceed with the encoding of the basic \textit{DPLL}(T) procedure as the construction of a derivation tree in System \textit{LKDPLL}(T). The simulation could be be stated as follows:

If \( \Delta [\phi] \Rightarrow^{*} \text{UNSAT} \) then there is a \textit{LKDPLL}(T) proof of \( |\Delta| [\phi] \vdash_T \) (i.e. there is no \( T \)-model of \( \phi \) extending \( \Delta \)).

This is true; however, there is more information in \( \Delta [\phi] \Rightarrow^{*} \text{UNSAT} \) than in \( |\Delta| [\phi] \vdash_T \), because the \textit{DPLL}(T) sequence leading to \text{UNSAT} also backtracks on decision literals. This means that not only there is no \( T \)-model of \( \phi \) extending \( |\Delta| \), but no matter how decision literals of \( \Delta \) are changed, there is still no \( T \)-model of \( \phi \) that can be constructed. This notion is formalised by collecting the backtrack models as follows:

Definition 6 (Backtrack models) In Fig. 4 we define the interpretation of a model as a collection (formally, a multiset) of sets of literals.

Remark 7 We have \( |\Delta| \in [\Delta] \) and \( [\Delta] \subseteq |\Delta| \).

We consider a notion of a partial proof-tree to step-by-step simulate \textit{DPLL}(T) runs.
A partial proof-tree that has no open leaf is isomorphic to a derivation in $\text{LK}_{DPLL^+}(T)$ is a tree labelled with sequents, whose leaves are tagged as either open or closed, and such that every node that is not an open leaf is an instance of the $\text{LK}_{DPLL^+}(T)$ rules.\footnote{A partial proof-tree that has no open leaf is isomorphic to a derivation in $\text{LK}_{DPLL^+}(T)$.}

A complete proof-tree is a partial proof-tree whose leaves are all closed.

A partial proof-tree $\pi'$ is an $n$-extension of $\pi$ if $\pi'$ is $\pi$ or if $\pi'$ is obtained from $\pi$ by replacing one of its open leaves by a partial proof-tree of size at most $n$ and whose conclusion has the same label as that leaf.

**Definition 8 (Correspondence between $\text{DPLL}(T)$ states and partial proof-trees)** A partial proof-tree $\pi$ corresponds to a $\text{DPLL}(T)$ state $\Delta \parallel \phi$ if the sequents labelling its open leaves form a sub-set of $\{ \Delta'; \phi \vdash T \mid \Delta' \in [\Delta] \}$. A partial proof-tree $\pi$ corresponds to UNSAT if it has no open leaf.

The $\text{DPLL}(T)$ procedure starts from an initial state i.e. $\emptyset \parallel \phi$, to which corresponds the partial proof-tree consisting of one node (both a root and a leaf) labelled with the sequent $\emptyset \vdash T$.

Note that, different partial proof-trees might correspond to the same $\text{DPLL}(T)$ state, as different $\text{DPLL}(T)$ runs can lead to that state from various initial $\text{DPLL}(T)$ states. The simulation theorem below expresses the fact that, when $\text{DPLL}(T)$ rewrites one state to another state, any partial proof-tree corresponding to the formal state can be extended into a partial proof-tree corresponding to the latter state.

**Theorem 8** If $\Delta \parallel \phi \Rightarrow \mathcal{S}_2$ is a rewrite step of $\text{DPLL}(T)$ and if $\pi_1$ corresponds to $\Delta \parallel \phi$ then there is, in $\text{LK}_{DPLL^+}(T)$, a $|\phi| + 1$-extension $\pi_2$ of $\pi_1$ corresponding to $\mathcal{S}_2$.

**Proof:** By case analysis:

- **Fail:** $\Delta \parallel \phi, C \Rightarrow^* \text{UNSAT}$ with $|\Delta| \models \neg C$ and there is no decision literal in $\Delta$. Let $\pi_1$ be a partial proof-tree corresponding to $\Delta \parallel \phi, C$. Since there are no decision literals in $\Delta$, $\pi_1$ can have at most one open leaf, labelled by $|\Delta|; \phi \vdash T$. We $|\phi, C| + 1$-extend $\pi_1$ into $\pi_2$ by replacing that leaf by a complete tree deriving $|\Delta|; \phi \vdash T$. We obtain that tree by applying Lemma 6 on the hypothesis $|\Delta| \models \neg C$. The new tree $\pi_2$ is complete and therefore corresponds to the UNSAT state of the $\text{DPLL}(T)$ run.

- **Decide:** $\Delta \parallel \phi \Rightarrow T \parallel \phi \parallel \phi$ where $l \not\in \Delta$, $t \not\in \Delta$, $l \in \phi$ or $t \in \phi$.

Let $\pi_1$ be a partial proof-tree corresponding to $\Delta \parallel \phi$. We $1$-extend it into $\pi_2$ by replacing the open leaf labelled with $|\Delta|; \phi \vdash T$ (if there is such a leaf) by one of three proof-trees:

- **If** $|\Delta|, l \models T$, we have $\text{Sat}(|\Delta|) = \text{Sat}(|\Delta|, l^t)$ and we take:

\[
\frac{|\Delta|, l^t; \phi \models T}{|\Delta|; \phi \models T} \quad \text{Weakening 2}
\]

The new open leaves form a sub-set of $\{ |\Delta|, l^t; \phi \vdash T \} \cup \{ \Delta'; \phi \vdash T \mid \Delta' \in [\Delta] \}$ (since $|\Delta|, l^t = |\Delta|, l^t \in [\Delta], l^t \in [\Delta], l^t \subseteq [\Delta], l^t \}$ and therefore $\pi_2$ corresponds to $\Delta, l^t \parallel \phi$.

- **If** $|\Delta|, l^t \models T$, we have $\text{Sat}(|\Delta|) = \text{Sat}(|\Delta|, l)$ and we take:

\[
\frac{|\Delta|, l; \phi \models T}{|\Delta|; \phi \models T} \quad \text{Weakening 2}
\]

The formal definition of the $\text{DPLL}(T)$ state, any partial proof-tree corresponding to the formal state can be extended into a partial proof-tree corresponding to the latter state.

**Theorem 8** If $\Delta \parallel \phi \Rightarrow \mathcal{S}_2$ is a rewrite step of $\text{DPLL}(T)$ and if $\pi_1$ corresponds to $\Delta \parallel \phi$ then there is, in $\text{LK}_{DPLL^+}(T)$, a $|\phi| + 1$-extension $\pi_2$ of $\pi_1$ corresponding to $\mathcal{S}_2$.

**Proof:** By case analysis:

- **Fail:** $\Delta \parallel \phi, C \Rightarrow^* \text{UNSAT}$ with $|\Delta| \models \neg C$ and there is no decision literal in $\Delta$. Let $\pi_1$ be a partial proof-tree corresponding to $\Delta \parallel \phi, C$. Since there are no decision literals in $\Delta$, $\pi_1$ can have at most one open leaf, labelled by $|\Delta|; \phi \vdash T$. We $|\phi, C| + 1$-extend $\pi_1$ into $\pi_2$ by replacing that leaf by a complete tree deriving $|\Delta|; \phi \vdash T$. We obtain that tree by applying Lemma 6 on the hypothesis $|\Delta| \models \neg C$. The new tree $\pi_2$ is complete and therefore corresponds to the UNSAT state of the $\text{DPLL}(T)$ run.

- **Decide:** $\Delta \parallel \phi \Rightarrow T \parallel \phi \parallel \phi$ where $l \not\in \Delta$, $t \not\in \Delta$, $l \in \phi$ or $t \in \phi$.

Let $\pi_1$ be a partial proof-tree corresponding to $\Delta \parallel \phi$. We $1$-extend it into $\pi_2$ by replacing the open leaf labelled with $|\Delta|; \phi \vdash T$ (if there is such a leaf) by one of three proof-trees:

- **If** $|\Delta|, l \models T$, we have $\text{Sat}(|\Delta|) = \text{Sat}(|\Delta|, l^t)$ and we take:

\[
\frac{|\Delta|, l^t; \phi \models T}{|\Delta|; \phi \models T} \quad \text{Weakening 2}
\]

The new open leaves form a sub-set of $\{ |\Delta|, l^t; \phi \vdash T \} \cup \{ \Delta'; \phi \vdash T \mid \Delta' \in [\Delta] \}$ (since $|\Delta|, l^t = |\Delta|, l^t \in [\Delta], l^t \in [\Delta], l^t \subseteq [\Delta], l^t \}$ and therefore $\pi_2$ corresponds to $\Delta, l^t \parallel \phi$.

- **If** $|\Delta|, l^t \models T$, we have $\text{Sat}(|\Delta|) = \text{Sat}(|\Delta|, l)$ and we take:

\[
\frac{|\Delta|, l; \phi \models T}{|\Delta|; \phi \models T} \quad \text{Weakening 2}
\]
The new open leaves form a sub-set of \( \{ \Delta, l; \phi \vdash \tau \} \cup \{ \Delta'; \phi \vdash \tau \mid \Delta' \in [\Delta] \} \subseteq \{ \Delta', \phi \vdash \tau \mid \Delta' \in [\Delta', l''] \} \) (since \( |\Delta|, l = |\Delta, l| \in [\Delta, l''] \)) and therefore \( \pi_2 \) corresponds to \( \Delta, l''\|\phi \).

- If \( |\Delta|, l \not\vdash \tau \) and \( |\Delta|, l^2 \not\vdash \tau \), we take
  \[
  |\Delta|, l; \phi \vdash \tau \quad |\Delta|, l^2; \phi \vdash \tau
  \]
  \[
  \text{Split}
  \]
  The new open leaves form a sub-set of \( \{ |\Delta|, l; \phi \vdash \tau \} \cup \{ |\Delta|, l^2; \phi \vdash \tau \} \cup \{ \Delta'; \phi \vdash \tau \mid \Delta' \in [\Delta] \} \subseteq \{ \Delta', \phi \vdash \tau \mid \Delta' \in [\Delta, l'] \} \) and therefore \( \pi_2 \) corresponds to \( \Delta, l''\|\phi \).

- Backtrack: \( |\Delta|, l, l^2; \phi, C \Rightarrow |\Delta|, l^i; \phi, C \)
  if \( |\Delta|, l, l^2 \models \neg C \) and no decision literal is in \( l_2 \).

Let \( \pi_1 \) be a partial proof-tree corresponding to \( \Delta, l^i, \Delta_2; \phi, C \). Since there are no decision literal in \( \Delta_2, \pi_1 \) can have at most one open leaf, labelled with \( |\Delta|, l^i, \Delta_2; \phi, C \vdash \tau \).

We \( |\phi, C|+1 \)-extend \( \pi_1 \) into \( \pi_2 \) by replacing that leaf by a complete tree deriving \( |\Delta|, l^i, \Delta_2; \phi, C \vdash \tau \).

We obtain that partial proof-tree by applying lemma 6 on the assumption \( |\Delta|, l^i, \Delta_2 | \models \neg C \).

The new open leaves form a sub-set of \( \{ |\Delta|, l; \phi \vdash \tau \} \cup \{ |\Delta; \phi \vdash \tau \mid \Delta' \in [\Delta] \} \subseteq \{ |\Delta'; \phi \vdash \tau \mid \Delta' \in [\Delta, l'] \} \) (since \( |\Delta|, l^i = |\Delta, l'| \in [\Delta, l'] \)) and therefore \( \pi_2 \) corresponds to \( |\Delta|, l^{i+1}; \phi, C \) state of the DPLL(\( T \)) run.

- Unit propagation: \( \Delta; \phi, C \land l \Rightarrow |\Delta|, l; \phi, C \lor l \) where \( |\Delta| \models \neg C, l \not\in \Delta, l^i \not\in \Delta \).

Let \( \pi_1 \) be a partial proof-tree corresponding to \( \Delta; \phi, C \land l \). We \( |\phi, C \land l|+1 \)-extend it into \( \pi_2 \) by replacing the open leaf labelled with \( |\Delta|; \phi, C \land l \vdash \tau \) (if there is such a leaf) by one of three proof-trees:

- If \( |\Delta|, l^i \not\vdash \tau \), we have \( \text{Sat}(|\Delta|) = \text{Sat}(|\Delta|, l) \) and we take:
  \[
  |\Delta|, l; \phi \vdash \tau
  \]
  The new open leaves form a sub-set of \( \{ |\Delta|, l; \phi \vdash \tau \} \cup \{ |\Delta'; \phi \vdash \tau \mid \Delta' \in [\Delta] \} \subseteq \{ |\Delta'; \phi \vdash \tau \mid \Delta' \in [\Delta, l'] \} \) (since \( |\Delta|, l = |\Delta, l| \in [\Delta, l'] \)) and therefore \( \pi_2 \) corresponds to \( |\Delta, l'; \phi, C \lor l \) where the side-conditions of \( \text{Resolve} \) are provided by the hypothesis \( \Delta'' \models \neg C \).

The new open leaves form a sub-set of \( \{ |\Delta|, l; \phi \vdash \tau \} \cup \{ |\Delta'; \phi \vdash \tau \mid \Delta' \in [\Delta] \} \subseteq \{ |\Delta'; \phi \vdash \tau \mid \Delta' \in [\Delta, l'] \} \) (since \( |\Delta|, l = |\Delta, l| \in [\Delta, l'] \)) and therefore \( \pi_2 \) corresponds to \( |\Delta, l'; \phi, C \land l \) where \( |\Delta| \models \tau, l \in \text{lit}(\phi) \) and \( l \not\in \Delta, l^i \not\in \Delta \).

- Theory Propagate: \( \Delta; \phi \Rightarrow |\Delta|, l; \phi \)

Let \( \pi_1 \) be a partial proof-tree corresponding to \( \Delta; \phi \). We \( 1 \)-extend it into \( \pi_2 \) by replacing the open leaf labelled with \( |\Delta|; \phi \vdash \tau \) by the following proof-tree:

\[
|\Delta|, l; \phi \vdash \tau
\]

\[
|\Delta|; \phi \vdash \tau
\]

The new open leaves form a sub-set of \( \{ |\Delta|, l; \phi \vdash \tau \} \cup \{ |\Delta'; \phi \vdash \tau \mid \Delta' \in [\Delta] \} \subseteq \{ |\Delta'; \phi \vdash \tau \mid \Delta' \in [\Delta, l'] \} \) (since \( |\Delta|, l = |\Delta, l| \in [\Delta, l'] \)) and therefore \( \pi_2 \) corresponds to \( |\Delta, l'; \phi \)
Corollary 9 \( LK_{\text{DPLL}}(T) \) is complete, i.e. if \( \phi \models_T \) then \( \phi \vdash_T \).

Proof: By completeness of basic \( \text{DPLL}(T) \) and Theorem 8. \( \square \)

1.3 \( \text{DPLL}(T) \) with backjumping and Lemma learning

We now consider a more advanced version of \( \text{DPLL}(T) \), which involves backjumping and lemma learning features, and which we denote \( \text{DPLL}_{bj}(T) \). \( \text{DPLL}_{bj}(T) \) extends basic \( \text{DPLL}(T) \) with the rules known as \( T \)-Backjump, \( T \)-Learn, \( T \)-Forget, and Restart [NOT06]. Those rules drastically increase the efficiency of SMT-solvers.

\( T \)-Backjump: \( \Delta_1, l^1, \Delta_2 \models \phi, C \Rightarrow \Delta_1, l_b \models \phi, C \) with

1. \( |\Delta_1, l^1, \Delta_2| \models C'. \)
2. \( |\Delta_1| \models \neg C'. \)
3. \( \phi, C \models_T C' \lor b_j \)
4. \( l_b, \Delta_1, l^1_b \not\models \Delta_1 \) and \( b_j \in \text{lit}(\phi, \Delta_1, l^1, \Delta_2) \).

for some clause \( C' \) such that \( \text{lit}(C') \subseteq \text{lit}(\phi, C) \).

\( T \)-Learn: \( \Delta \models \phi \Rightarrow \Delta \models \phi, C \) if \( \text{lit}(C) \subseteq \text{lit}(\phi, \Delta) \) and \( \phi \models_T C \).

\( T \)-Forget: \( \Delta \models \phi, C \Rightarrow \Delta \models \phi \) if \( \phi \models_T C \).

Restart: \( \Delta \models \phi \Rightarrow \emptyset \models \phi \).

In order to simulate those extra rules in \( LK_{\text{DPLL}}(T) \), we need to extend \( LK_{\text{DPLL}}(T) \) with a cut rule as follows:

\[ \frac{\Delta \models \phi, C \models_T \Delta \models \phi, C}{\Delta \models \phi \models_T} \]  \( \text{Cut} \)

We define the size of proof-trees in \( LK_{\text{DPLL}}(T) \) as we did for \( LK_{\text{DPLL}+}(T) \) (ignoring Weakening1, Weakening2 or the Inverted Resolve), but also ignoring the left-branch of the cut-rules.\(^3\)

Definition 10 (\( n, \phi, S \)-sync action) \( \pi_{\phi} \) is a \( n, \phi, S \)-sync action if it is a function that maps every model \( \Delta \in S \) to a partial proof-tree of size at most \( n \) and concluding \( \Delta \models \phi \models_T \).

Definition 11 (Parallel \( n \)-extension of partial proof-trees) \( \pi_2 \) is a parallel \( n \)-extension of \( \pi_1 \) according to \( \pi_\phi \) if \( \pi_\phi \) is a \( n, \phi, S \)-sync action and if \( \pi_2 \) is obtained from \( \pi_1 \) by replacing all the open leaves of \( \pi_1 \) labelled by sequents of the form \( \Delta \models \phi \models_T \) (where \( \Delta \in S \)) by \( \pi_\phi(\Delta) \).

Theorem 10 If \( \Delta \models \phi \Rightarrow \text{DPLL}_{bj}(T) \), \( S_2 \) and \( \pi_1 \) corresponds to \( \Delta \models \phi \), there is parallel \( \phi \models +3 \)-extension \( \pi_2 \) of \( \pi_1 \) (according to some \( \pi_\phi \)) such that \( \pi_2 \) corresponds to \( S_2 \).

Proof: Since \( LK_{\text{DPLL}}(T) \) is a sub-system of \( LK_{\text{DPLL}}(T) \), we only need to simulate (in \( LK_{\text{DPLL}}(T) \)) the new rules.

\( T \)-Backjump: \( \Delta_1, l^1, \Delta_2 \models \phi, C \Rightarrow \Delta_1, l_b \models \phi, C \) with

1. \( |\Delta_1, l^1, \Delta_2| \models C'. \)
2. \( |\Delta_1| \models \neg C'. \)
3. \( \phi, C \models_T C' \lor b_j \)
4. \( l_b \not\models \Delta_1, l^1_b \not\models \Delta_1 \) and \( b_j \in \text{lit}(\phi, \Delta_1, l^1, \Delta_2) \).

\(^3\)As we shall see in the simulation theorem, this definition mimics the fact that the length of \( \text{DPLL}(T) \) sequences is a complexity measure that ignores the cost of checking the side-conditions.
Let \( \pi_1 \) be a partial proof-tree corresponding to \( \Delta_1, l_{b_1} \parallel \phi, C \). We have to build a \( \pi_2 \) that corresponds to \( \Delta_1, l_{b_2} \parallel \phi, C \) in the \( \text{DPLL}_{b_1}(T) \) run. This means that the open leaves of \( \pi_2 \) should be labelled with sequents of the form \( \Delta', \phi, C \vdash \top \) where \( \Delta' \in [\Delta_1, l_{b_2}] \).

Let \( S = [\Delta_1, l_{b_2}] \parallel [\Delta_1] \) and \( \pi_0 \) be the \( \{\phi, C\}+3, \phi, C, S \)-sync action that maps every \( \Delta \in S \) to

\[
\begin{align*}
|\Delta_1|, l_{b_2} ; \phi, C \vdash &\top & \text{Subsume} \\
|\Delta_1| ; \phi, C, l_{b_2} \vdash &\top & \text{Assert} \\
|\Delta_1| ; \phi, C \vdash &\top & \text{Resolve} \\
|\Delta_1| ; \phi, C, C' \lor l_{b_2} \vdash &\top & \text{cut}
\end{align*}
\]

It is a valid partial proof-tree because \( \Delta \in S \) entails \( |\Delta_1| \subseteq \Delta \) and therefore \( \text{Sat}_B(|\Delta_1|) \subseteq \text{Sat}_B(\Delta) \). The left branch is closed by assumption (3) and the completeness of \( \text{LKDPDLL}(T) \) on \( \phi, C \vdash \neg C' \lor l_{b_2} \vdash \top \) (Corollary 9). We cannot anticipate the size of the proof-tree closing that branch, and we therefore ignore that proof-tree to compute the size of the whole tree, just as the length of the \( \text{DPLL}(T) \) run ignores the cost of checking \( C' \lor l_{b_2} \).

Let \( \pi_2 \) be the parallel \( \{\phi, C\}+3 \)-extension of \( \pi_1 \) according to \( \pi_0 \). The new open leaves form a sub-set of \( \{[\Delta_1], l_{b_2} ; \phi, C \vdash \top \} \cup \{[\Delta', \phi, C \vdash \top \mid \Delta' \in [\Delta_1]\} \subseteq \{[\Delta'; \phi, C \vdash \top \mid \Delta' \in [\Delta_1, l_{b_2}]\} \)

(since \( [\Delta_1], l_{b_2} = [\Delta_1, l_{b_2}] \in [\Delta_1, l_{b_2}] \) and \( [\Delta_1, l_{b_2}] = [\Delta_1] \)) and therefore \( \pi_2 \) corresponds to \( \Delta_1, l_{b_2} \parallel \phi, C \).

**T-Learn:** \( \Delta \parallel \phi \Rightarrow \Delta \parallel \phi, C \) if each atom of \( C \) occurs in \( \phi \) or in \( \Delta \) and \( \phi \vdash \top \).

Let \( \pi_1 \) be a partial proof-tree corresponding to \( \Delta \parallel \phi \). We have to build a \( \pi_2 \) that corresponds to \( \Delta \parallel \phi, C \) in the \( \text{DPLL}_{b_1}(T) \) run. This means that the open leaves of \( \pi_2 \) should be labelled with sequents of the form \( \Delta', \phi, C \vdash \top \) where \( \Delta' \in [\Delta] \).

Let \( S = [\Delta] \) and \( \pi_0 \) be the \( \{\phi, C, S\} \)-sync action that maps every \( \Delta \in S \) to:

\[
\begin{align*}
|\Delta| ; \phi, C \vdash &\top & \text{Subsume} \\
|\Delta| ; \phi, C, C' \lor l_{b_2} \vdash &\top & \text{Resolve} \\
|\Delta| ; \phi, C, l_{b_2} \vdash &\top & \text{cut}
\end{align*}
\]

The left branch of the cut is closed by assumption and completeness of \( \text{LKDPDLL}(T) \) on \( \phi \vdash C \lor l_{b_2} \vdash \top \) (Corollary 9). We cannot anticipate the size of the proof-tree closing that branch, and we therefore ignore that proof-tree to compute the size of the whole tree, just as the length of the \( \text{DPLL}(T) \) run ignores the cost of checking \( C \lor l_{b_2} \).

Let \( \pi_2 \) be the parallel \( \{\phi\}+3 \)-extension of \( \pi_1 \) according to \( \pi_0 \). The new open leaves form a sub-set of \( \{\Delta', \phi, C \vdash \top \mid \Delta' \in [\Delta]\} \) and therefore \( \pi_2 \) corresponds to \( \Delta \parallel \phi, C \).

**T-Forget:** \( \Delta \parallel \phi \Rightarrow \emptyset \parallel \phi \) if \( \phi \vdash \top \).

Let \( \pi_1 \) be a partial proof-tree corresponding to \( \Delta \parallel \phi \). We have to build a \( \pi_2 \) that corresponds to \( \emptyset \parallel \phi \) in the \( \text{DPLL}_{b_1}(T) \) run. This means that the open leaves of \( \pi_2 \) should be labelled with sequents of the form \( \phi \vdash \top \).
Let $S = [\Delta]$ and $\pi_\phi$ be the 1, 1, $S$-sync action that maps every $\Delta' \in S$ to:

$$
\phi \vdash T \quad Weakening2
$$

Let $\pi_2$ be the parallel 1-extension of $\pi_1$ according to $\pi_\phi$. The new open leaves form a sub-set of $\{; \phi \vdash T\}$ and therefore $\pi_2$ corresponds to $\emptyset\parallel \phi$.

□
2 Encoding \( \text{LK}_{\text{DPLL}}(\mathcal{T}) \) in \( \text{LK}^p(\mathcal{T}) \)

2.1 Preliminaries: System \( \text{LK}^p(\mathcal{T}) \)

In this section we introduce (the propositional fragment of) system \( \text{LK}^p(\mathcal{T}) \).

**Definition 12 (Formulae, negation)** The formulae of \( \text{LK}^p(\mathcal{T}) \) are given by the following grammar:

\[
\text{Formulae } A, B, \ldots ::= l \mid A \lor^+ B \mid A \lor^+ B \mid A \land^- B \mid A \lor^- B
\]

where \( l \) ranges over literals.

Let \( P \) be a set of literals declared to be *positive*, while their negations, required to not be in \( P \), are declared to be *negative*. Given such a set \( P \), we define positive formulae and negative formulae as the formulae generated by the following grammars:

**positive formulae**

\[
P, \ldots ::= p \mid A \lor^+ B \mid A \lor^+ B
\]

**negative formulae**

\[
N, \ldots ::= p\bot \mid A \lor^- B \mid A \lor^- B
\]

where \( p \) ranges over \( P \).

Negation is recursively extended into an involutive map from formulae to formulae as follows:

\[
(\lnot (A \lor^+ B))^\bot := A \lor^- B \lor^+ \lnot \!
(\lnot (A \lor^- B))^\bot := A \lor^+ B \lor^- \lnot \!
(\lnot (A \land^- B))^\bot := A \land^+ B \land^- \lnot \!
(\lnot (A \land^+ B))^\bot := A \land^- B \land^+ \lnot \!
\]

**Definition 13 (System \( \text{LK}^p(\mathcal{T}) \))** The sequent calculus \( \text{LK}^p(\mathcal{T}) \) has two kinds of sequents:

\[
\Gamma \vdash \mathcal{T}[P] \quad \text{where } P \text{ is in the focus of the sequent}
\]

Its rules are given in Figure 5.

\( \mathcal{T}(\Delta) \) is the call to the decision procedure on the conjunction of all atomic formulae within \( \Delta \). It holds if the procedure returns UNSAT.

![Figure 5: System \( \text{LK}^p(\mathcal{T}) \)](image)

We also consider two cut-rules. The analytic cut:

\[
\Gamma, l \vdash \mathcal{T}[P] \quad \text{with the condition that } l \text{ appears in } \Gamma.
\]
2.2 Simulation

We now encode \( \text{LK} \) in \( \text{LK}' \).

The main gap between \( \text{DPLL} \) (or even \( \text{DPLL} \)) and a sequent calculus such as \( \text{LK}' \) is the fact that the structures handled by the former are very flexible (e.g. clauses are multisets of literals), while sequent calculus implements a root-first decomposition of formulae trees.

Clauses in \( \text{DPLL} \) (and in \( \text{DPLL} \)) are disjunctions considered modulo associativity and commutativity. The way we encode them as formulae of sequent calculus is as follows: a clause \( C \) will be represented by a formula \( C' \) which is a disjunctive tree whose leaves contain at least all the literals of \( C \) but also other literals that we can consider as garbage.

Of course, one could fear that the presence of garbage parts within \( C' \) degrades the efficiency of proof-search when simulating \( \text{DPLL} \). This garbage comes from the original clauses at the start of the \( \text{DPLL} \) rewriting sequence, which might have been simplified in later steps of \( \text{DPLL} \) but which remain unchanged in sequent calculus. The size of the garbage is therefore smaller than the size of the original problem. We ensure that the inspection, by the proof-search process, of the garbage in \( C' \) takes no more inference steps than the size of the garbage itself (the waste of time is linear in the size of the garbage). In order to ensure this, we use polarities and the focusing properties of \( \text{LK}' \): the garbage literals in \( C' \) must be negative atoms that are negated in the model/context.

Definition 14 (\( \mathcal{P} \)-correspondence) Let \( \mathcal{P} \) be a multiset of literals.

- A formula \( C' \) \( \mathcal{P} \)-corresponds to a clause \( C \) (in system \( \text{LK} \)) where \( C = \bigvee \mathcal{P} \), if \( C' = \bigvee \mathcal{P} \) with \( \mathcal{P} \subseteq \mathcal{P} \) and for any \( l \in \mathcal{P} \), \( l \in \mathcal{P} \).
- A \( \text{LK}' \) sequent \( \Delta, C_1, \ldots, C_m \vdash \mathcal{P} \) corresponds to a \( \text{LK} \) sequent \( \Delta \vdash C_1, \ldots, C_m \), if \( C_i' \) \( \mathcal{P} \)-corresponds to \( C_i \) and for all \( l \in \mathcal{P} \), \( \Delta \vdash l \).

Lemma 11 If \( C' \) \( \mathcal{P} \)-corresponds to \( C \), then \( C' \) also \( (\mathcal{P}, l) \)-corresponds to \( C \).

Proof: Straightforward.

Theorem 12 Assume \( S \) is a rule of \( \text{LK} \). For every \( \text{LK}' \) sequent \( S' \) that corresponds to \( S \), there exist a partial proof-tree in \( \text{LK}' \)

- whose open leaves \( (S') \) are such that \( \forall i, S'_i \) corresponds to \( S_i \) and
- whose size is smaller than size \( (S') + 4 \).

Proof: By case analysis:

- Split:

\[
\Delta, l^+, \phi \vdash \tau \quad \Delta, l, \phi \vdash \tau
\]

where \( l \in \text{lit}(\phi) \), \( \Delta, l^+ \not\vdash \tau \) and \( \Delta, l \not\vdash \tau \)

Assume that \( \Delta, \phi' \vdash \mathcal{P} \) corresponds to \( \Delta; \phi \vdash \tau \) (i.e. \( \phi' = C'_1, \ldots, C'_n \) and \( \phi = C_1, \ldots, C_n \) with \( C'_i \) \( \mathcal{P} \)-corresponding to \( C_i \) for \( i = 1 \ldots n \)).

We build in \( \text{LK}' \) the following derivation that uses an analytic cut:

\[
\Delta, l^+, \phi' \vdash \Delta \quad \Delta, l, \phi' \vdash \Delta
\]

and \( \Delta, l^+, \phi' \vdash \Delta \) \( \mathcal{P} \)-corresponds to \( \Delta, l^+; \phi \vdash \tau \) and \( \Delta, l, \phi' \vdash \Delta \) \( \mathcal{P} \)-corresponds to \( \Delta, l; \phi \vdash \tau \).
• Assert:

\[ \Delta, l \phi, l \vdash \Delta, l \vdash^{(+)T} \quad \text{and} \quad \Delta, l \vdash^{(+)T} \]

Assume that \( \Delta, \phi', C' \vdash^{T} \) corresponds to \( \Delta; \phi, l \vdash^{T} \) (i.e. \( \phi' = C_1, \ldots, C_n \) and \( \phi = C_1, \ldots, C_n \) with \( C_i \vdash^{T} \) corresponds to \( C_i \), for \( i = 1 \ldots n \), and \( C' \vdash^{T} \) corresponds to \( l \), that is to say \( C' = \lor_{i=1}^n l_i \) where \( l = l_{i_0} \) for some \( i_0 \in 1 \ldots n \). We build in \( \text{LK} \) the following derivation:

\[
\begin{array}{c}
\text{T}(\Delta, \phi', C', l_i) \\
\Delta, \phi', C' \vdash^{T} [l_i] \\
\hline
\Delta, \phi', C' \vdash^{T} [l_i] \\
\Delta, \phi', C' \vdash^{T} l_i \\
\Delta, \phi', C' \vdash^{T} [l_i] \\
\hline
\end{array}
\]

For \( i \neq i_0 \), \( l_i^+ \in \Delta_0 \), so it is positive and we can use an axiom (remember that \( \Delta \vdash l_i^+ \)).

• Empty\( \vdash \):

\[ \Delta; \phi, \perp \vdash^{T} \]

Assume that \( \Delta, \phi', C' \vdash^{T} \) corresponds to \( \Delta; \phi, \perp \vdash^{T} \) (i.e. \( C' \vdash^{T} \perp \) corresponds to \( \perp \), \( \phi' = C_1, \ldots, C_n \) and \( \phi = C_1, \ldots, C_n \) with \( C_i \vdash^{T} \) corresponding to \( C_i \), for \( i = 1 \ldots n \)). We build in \( \text{LK} \) the following derivation:

\[
\begin{array}{c}
\text{T}(\Delta, \phi', C', l_i) \\
\Delta, \phi', C' \vdash^{T} [l_i] \\
\hline
\Delta, \phi', C' \vdash^{T} [l_i] \\
\Delta, \phi', C' \vdash^{T} l_i \\
\Delta, \phi', C' \vdash^{T} [l_i] \\
\hline
\end{array}
\]

Again, \( l_i^+ \in \Delta_0 \), so it is positive and we can use an axiom (remember that \( \Delta \vdash l_i^+ \)).

• Resolve:

\[ \Delta; \phi, l \lor C \vdash^{T} \Delta, l \vdash^{(+)T} \]

Assume that \( \Delta, \phi', C' \vdash^{T} \) corresponds to \( \Delta; \phi, l \lor C \vdash^{T} \) (i.e. \( C' \vdash^{T} \) corresponds to \( l \lor C \), \( \phi' = C_1, \ldots, C_n \) and \( \phi = C_1, \ldots, C_n \) with \( C_i \vdash^{T} \) corresponding to \( C_i \) for \( i = 1 \ldots n \)). We build in \( \text{LK}(T) \) the following derivation

\[
\begin{array}{c}
\text{T}(\Delta, \phi', C', l_i) \\
\Delta, \phi', C' \vdash^{T} [l_i] \\
\hline
\Delta, \phi', C' \vdash^{T} l_i \\
\Delta, \phi', C' \vdash^{T} l_i \\
\hline
\end{array}
\]

It suffices to notice that \( \Delta, \phi', C' \vdash^{(+)T} \) corresponds to \( \Delta; \phi, C \vdash^{T} \).

• Subsume:

\[ \Delta; \phi \vdash^{T} \Delta, l \vdash^{T} \]

Assume that \( \Delta, \phi', C' \vdash^{T} \) corresponds to \( \Delta; \phi, l \lor C \vdash^{T} \) (i.e. \( C' \vdash^{T} \) corresponds to \( l \lor C \), \( \phi' = C_1, \ldots, C_n \) and \( \phi = C_1, \ldots, C_n \) with \( C_i \vdash^{T} \) corresponding to \( C_i \) for \( i = 1 \ldots n \)).

• Cut: If we want to simulate \( \text{DPLL}(T) \) with backjump, we need to encode the cut rule of \( \text{LK}_{\text{DPLL}} \).
\[ \Delta; \phi, l_1, \ldots, l_n \vdash_T \quad \Delta; \phi, C \vdash_T \quad C = l_1^+ \lor \ldots \lor l_n^+ \]

Assume that \( \Delta, \phi \vdash_T \) corresponds to \( \Delta; \phi \vdash_T \) (i.e. \( \phi' = C_1', \ldots, C_n' \) and \( \phi = C_1, \ldots, C_n \) with \( C_i' \) \( \mathcal{P} \)-corresponding to \( C_i \) for \( i = 1 \ldots n \)).

We build in \( \mathcal{LKP}(T) \) the following derivation that uses a general cut:

\[
\begin{array}{c}
\Delta; \phi', l_1, \ldots, l_n \vdash_T \\
\Delta, \phi', (l_1^+ \lor \ldots \lor l_n^+) \vdash_T \\
\end{array}
\]

\[
\Delta; \phi' \vdash_T
\]

Clearly, \( \Delta; \phi', l_1, \ldots, l_n \vdash_T \) corresponds to \( \Delta; \phi, l_1, \ldots, l_n \vdash_T \) and \( \Delta, \phi', (l_1^+ \lor \ldots \lor l_n^+) \vdash_T \) corresponds to \( \Delta; \phi, C \vdash_T \).

\[\Box\]

References
