Forcing in DPLL Satisfiability Solving

Ján Kľuka and Paul J. Voda

{kluka,voda}@fmph.uniba.sk
Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava
Mlynská dolina
842 48 Bratislava, Slovakia

Abstract. Many state-of-the-art satisfiability (SAT) solvers are based on the DPLL proof procedure [3]. This paper was inspired by a recent elegant characterization of the procedure by Nieuwenhuis, Oliveras, and Tinelli [5] by a *state transition system*. This, in turn, was motivated by (what may be termed) a *proof-theoretical* characterization of Tinelli [7]. The partial satisfaction relation $M \vDash A$ is defined in [5] with M an assignment of truth values to some atoms of a formula A in the conjunctive normal form (cnf). The key observation of this paper is that such $M \vDash A$ coincides on the fragment of cnf-formulas with a suitably chosen forcing relation $M \Vdash A$ for the full classical propositional logic. This helped us to streamline and generalize the characterization of DPLL-based procedures by a *model-theoretical* description of traversal strategies for the universe of possible worlds (as given by the forcing conditions M). Namely, for a given forcing condition M and arbitrary propositional formula A we obtain either an extension N of M such that $N \Vdash A$ (i.e. $V \vDash A$ in all (total) models V of N) or we conclude $M \Vdash \neg A$ (i.e. no model of M satisfies A). The forcing explication of DPLL SAT solving smoothly extends to the solving modulo theories.

1 Introduction

We do not repeat the abstract here and recall only that forcing was introduced in 1963 by Paul Cohen in order to obtain his spectacular independence results in the set theory. The path-breaking innovation was that in a framework where we are either unable or unwilling to construct full models, forcing permits to make assertions about the future (or possible) models.

With only a slight metaphor we can say that Cohen used forcing as an extremely theoretical device exploring a Platonic world inhabited by vastly immense entities far removed from any practical reality. On the other hand, forcing applied in this paper caters to the extremely practical needs of VLSI design because this is where the SAT solvers are used with no less spectacular results.

In Sect. 2 we define and discuss the standard forcing for the classical propositional logic. In Sect. 3 we characterize the DPLL satisfiability procedure.

2 Classical Propositional Forcing

Since forcing is relatively unknown outside a small group of logicians, we will be in the following self-contained so the reader does not have to go to the literature for the definitions. For this reason we also include the description of the (standard) syntax and semantics of classical propositional logic. For additional applications of forcing we refer the reader to [2, 1, 6] with the references listed in the order of increasing difficulty.

2.1 The language and semantics of propositional logic. A *first-order language* $\mathcal{L}$ is given by a denumerable set of function and predicate symbols with at least one constant symbol (a function symbol of arity 0). *Closed terms* of $\mathcal{L}$ are of the form $f(s_1, \ldots, s_n)$ where f is of arity $n \geq 0$ and $s_1,\ldots, s_n$ are closed terms. *Closed propositional formulas* of $\mathcal{L}$ are formed from *atoms* $s_1 = s_2$ and $P(s_1, \ldots, s_n)$ with P a predicate symbol of arity $n \geq 0$ by the connectives $\neg$, $\vee$, $\wedge$, and $\rightarrow$. We will deal below exclusively with closed propositional formulas (sentences) and we will henceforth call them just formulas.

Literals L are formulas of a form A or $\neg A$ with A and atom. In the former case we will designate by $\bar{L}$ the literal $\neg A$ and in the later case the literal A.

Formulas will be designated by meta-variables A, B, C and literals by L, both possibly subscribed. In contexts where the language $\mathcal{L}$ is known we will tacitly understand that the meta-variables range over the corresponding objects from $\mathcal{L}$. For sets of formulas T_1, T_2 we will write T_1, T_2 instead of $T_1 \cup T_2$ and T_1, A instead of $T_1 \cup \{A\}$.

An *assignment* (valuation) V *for* $\mathcal{L}$ is a subset of atoms of $\mathcal{L}$. For formulas A of $\mathcal{L}$ we define the relation V *satisfies* A, in writing $V \vDash A$, to satisfy:

- if A is an atom then $V \vDash A$ iff $A \in V$,
- $V \vDash \neg A$ iff $V \nvDash A$,
- $V \vDash A \vee B$ iff $V \vDash A$ or $V \vDash B$,
- $V \vDash A \wedge B$ iff $V \vDash A$ and $V \vDash B$,
- $V \vDash A \rightarrow B$ iff $V \vDash B$ whenever $V \vDash A$.

A (propositional) *theory in* $\mathcal{L}$ is a set T of formulas of $\mathcal{L}$. We write $V \vDash T$ if $V \vDash A$ for all $A \in T$. T has a *model* V if $V \vDash T$. T is *consistent* if it has a model. We clearly have L, T inconsistent iff $T \vDash \bar{L}$.

For a theory T in $\mathcal{L}$ and a formula A we say that A *is a propositional consequence of* T, in writing $T \vDash A$, if $V \vDash A$ for all models V of T. We again write $T_1 \vDash T_2$ when $T_1 \vDash A$ for all $A \in T_2$. We say that T *decides* A if $T \vDash A$ or $T \vDash \neg A$.

We say that two assignments V_1 *and* V_2 *coincide on a set* T if for all atoms occurring in A we have $V_1 \vDash A$ iff $V_2 \vDash A$. This property then naturally extends to all subformulas B of formulas in T and we have $V_1 \vDash B$ iff $V_2 \vDash B$.

2.2 Propositional forcing. The pair $\langle P, \leq\rangle$ is a *poset* if P is partially ordered by $\leq$. A *Kripke structure for* $\mathcal{L}$ is a triple $\langle P, \leq, \Vdash\rangle$ where $\langle P, \leq\rangle$ is a poset with $p \in P$ *forcing conditions* and where the *forcing* relation $p \Vdash A$, read as *p forces*

A, is over conditions p and atoms A of $\mathcal{L}$. The forcing is *monotone* in the sense that for all $q \leq p$ we get from the above $q \Vdash A$ (cf. [1]).

The forcing conditions $p \in P$ can viewed as possible and partially described worlds to be completed in the future to the full assignments for $\mathcal{L}$. For $q, p \in P$ we say that q *extends* p if $q \leq p$ holds. This can be viewed as expressing the fact that the possible world q contains more information than p. We use p, q, r as meta-variables for forcing conditions and in contexts where a Kripke structure is implied we will tacitly assume that the meta-variables range over the forcing conditions of the structure.

We extend the forcing relation $\Vdash$ from atoms to all formulas of $\mathcal{L}$ as a minimal relation satisfying:

- $p \Vdash \neg A$ iff $q \nVdash A$ for all $q \leq p$,
- $p \Vdash A \vee B$ iff $p \Vdash A$ or $p \Vdash B$,
- $p \Vdash A \wedge B$ iff $p \Vdash A$ and $p \Vdash B$,
- $p \Vdash A \rightarrow B$ iff $q \Vdash B$ for all $q \leq p$ such that $q \Vdash A$.

Directly from the negation clause we get the *consistency* of forcing, i.e. for no A both $p \vDash A$ and $p \vDash \neg A$ can hold at the same time. More interesting property is that the extension of the forcing relation to all formulas preserves the monotonicity.

2.3 Theorem (Monotonicity of forcing). *For a Kripke structure for $\mathcal{L}$ $\langle P, \leq, \Vdash \rangle$ and all p, q, A such that $p \Vdash A$ and $q \leq p$ we have $q \Vdash A$.*

Proof. This is a standard proof by straightforward induction on the structure of A. For instance, if $p \Vdash A \rightarrow B$ and $q \leq p$ then in any $r \leq q$ such that $r \Vdash A$ we have $r \Vdash A$ by IH, since $r \leq p$. Hence $r \Vdash B$ and so $r \Vdash A \rightarrow B$. □

2.4 Kripke structures for consistent theories. For a consistent theory T in $\mathcal{L}$ we define the *Kripke structure for* T as $\langle P_T, \leq_T, \Vdash_T \rangle$ where P_T consists of **finite** sets p of literals for which the theory T, p is consistent, $q \leq_T p$ holds iff $p \subseteq q$, and for an atom A we define $p \Vdash_T A$ to hold if $T, p \vDash A$.

This makes the empty set $\emptyset$ the maximal forcing condition. For infinite $\mathcal{L}$ and T there are generally no minimal conditions. For the proof of the monotonicity for atoms A we assume $p \Vdash A$, $q \leq p$, and take any model V of T, q. We have $V \vDash p$ and so, since $T, p \vDash A$, we obtain $V \vDash A$. This proves $T, q \vDash A$ and hence $q \Vdash A$.

To every condition $p \in P_T$ we define an assignment $V(p) = \{A \mid A \text{ atom}, A \in p\}$. $V(p)$ is clearly a model of p and for all $q \leq p$ the assignments $V(p)$ and $V(q)$ coincide on the atoms of p.

When the theory T will be known from the context we will usually drop the indices T from the symbols designating the components of the Kripke structure for T.

2.5 Theorem (Some properties of forcing). *For any Kripke structure $\langle P, \leq, \Vdash \rangle$ for T in $\mathcal{L}$, any p, L, and A we have*

1. *$p \Vdash L$ iff $T, p \vDash L$,*
2. *if L, p is a forcing condition then $L, p \Vdash \neg A$ iff $p \Vdash A \to \bar{L}$,*
3. *if p decides all atoms in A then the following are equivalent: a) $T, p \vDash A$, b) $V(p) \vDash A$, c) $p \Vdash A$.*

Proof. (1): When L is an atom the property holds from the definition of the Kripke structure. When L is $\neg B$ with B an atom then in the direction $(\to)$ assume $T, p \nvDash \neg B$. Thus T, p, B is consistent and we get $p, B \leq p$. Since $T, p, B \vDash B$, we have $p, B \Vdash B$ and hence $T, p \nVdash \neg B$. In the direction $(\to)$ assume $T, p \vDash \neg B$ and take any $q \leq p$. We have $T, q \vDash \neg B$ and since T, q is consistent, we get $T, q \nvDash B$, i.e. $q \nVdash B$.

(2): Assume that L, p is a condition. In the direction $(\to)$ assume $L, p \Vdash \neg A$, take any $q \leq p$ s.t. $q \Vdash A$. It cannot be the case that T, L, q is consistent because then $L, q \leq L, p$ and hence $L, q \nVdash A$ contradicting $L, p \Vdash \neg A$. Thus $T, q \vDash \bar{L}$ and we get $q \Vdash \bar{L}$ by (1). In the direction $(\leftarrow)$ assume $L, p \nVdash \neg A$. Thus for some $q \leq L, p$ we have $q \Vdash A$. Now, $L \in q$, T, q is consistent, and so $T, q \nvDash \bar{L}$. Hence $q \nVdash \bar{L}$ by (1). But this proves $q \nVdash A \to \bar{L}$.

(3): We assume $T, p \vDash A$ and show that b) holds. Since p is a condition, T, p is consistent and has a model V. Thus $V \vDash A$ and hence $V(p) \vDash A$ by coincidence. We now assume $V(p) \vDash A$, and in order to get a), we take any model V of T, p. We have $V \vDash A$ by coincidence again.

The equivalence of b) with c) immediately follows from an auxiliary property:

> for all subformulas B of A and all extensions $q \leq p$ we have $V(q) \vDash B$ iff $q \Vdash B$.

The property is proved by induction on the structure of B for any $q \leq p$. If B is an atom then, since p decides B, also q does and we have either $B \in q$ and then $V(q) \vDash B$ from which we get $q \Vdash B$, or else $\neg B \in q$ and then $V(q) \nvDash B$, and since $T, q \vDash \neg B$, we have $q \Vdash \neg B$ by (1) and hence $q \nVdash B$ by consistency. If B is of a form $B_1 \vee B_2$ or $B_1 \wedge B_2$, we get the property directly from the induction hypotheses.

If B is $\neg C$ then $V(q) \vDash \neg C$ iff $V(q) \nvDash C$ iff $V(r) \nvDash C$ for all $r \leq q$ iff by IH $r \nVdash C$ for all $r \leq q$ iff $q \Vdash \neg C$.

If B is $B_1 \to B_2$ then we have $V(q) \vDash B_1 \to B_2$ iff $V(r) \vDash B_1 \to B_2$ for all $r \leq q$ iff for all $r \leq q$ such that $V(r) \vDash B_1$ also $V(r) \vDash B_2$ iff by IH for all $r \leq q$ such that $r \Vdash B_1$ also $r \Vdash B_2$ iff $q \Vdash B_1 \to B_2$. ☐

3 Forcing Explication of DPLL

The SAT-problem for a formula A modulo a (consistent) theory T (possibly empty) can be viewed as a constructive decision of the forcing relation $\emptyset \Vdash_T \neg A$. If the relation does not hold then this is witnessed by a condition $p \leq \emptyset$ deciding all atoms of A and such that $p \Vdash_T A$. By Thm. 2.5(3) we will then have the desired satisfaction $T, p \vDash A$. Hence, if $\emptyset \Vdash_T \neg A$ holds then no model of T satisfies A. Our procedures for the solution of the $\emptyset \Vdash_T \neg A$ will need to solve the subproblems $p \Vdash_T \neg A$ for conditions $p \in P_T$.

3.1 Forced literals. For a consistent theory T in $\mathcal{L}$ and its Kripke structure $\langle P, \leq, \Vdash\rangle$ we say that L *is forced by* A *in* p if $p \Vdash A \rightarrow L$. We denote by $F_{T,A}(p)$ the set of forced literals. The literals not in $F_{T,A}(p)$ are *free for* A *in* p.

The forced literals do not affect the satisfiability of A (although they do affect the efficiency of the solving procedure) because for any finite $F \subseteq F_{T,A}(p)$ such that F, p is a forcing condition, i.e. $F, p \leq p$, we have

$$p \Vdash \neg A \quad \text{iff} \quad F, p \Vdash \neg A. \tag{1}$$

The direction $(\rightarrow)$ follows by monotonicity. The direction $(\leftarrow)$ is proved by induction on the cardinality of F. When $F = \emptyset$ there is nothing to prove. When $F = L, F'$ with $L \notin F'$ then we assume $L, F', p \Vdash \neg A$ and obtain $F', p \Vdash A \rightarrow \bar{L}$ by Thm. 2.5(2). Since $L \in F_{T,A}(p)$, we have $p \Vdash A \rightarrow L$ and hence $F', p \Vdash A \rightarrow L$ by monotonicity. We now take any $q \leq F', p$. It cannot be the case that $q \Vdash A$ because we then get a contradiction $q \Vdash \bar{L}$ and $q \Vdash L$. Thus $F', p \Vdash \neg A$ and we get $p \Vdash \neg A$ by IH.

3.2 Approximations. For a fixed consistent theory T in $\mathcal{L}$, its Kripke structure $\langle P, \leq, \Vdash\rangle$, and a formula A, the problem of deciding $\emptyset \Vdash \neg A$ in the style of DPLL calls for the incremental and simultaneous collection of two kinds of approximations to the desired satisfying condition q deciding all atoms of A and such that $V(q) \vDash A$, which is equivalent to $T, q \vDash A$ by Thm. 2.5(3). The first kind of approximations are conditions p which will be incrementally extended with literals free for A in p in the hope that q is an extension of p. The second kind approximations are for each such p finite subsets F of forced literals $F_{T,A}(p)$.

We will thus incrementally collect finite sequences of pairs $\langle L, F\rangle$ with L a literal whose atom is among the atoms of A and with F a finite subset of such literals. In order to increase the readability we abbreviate the pairs $\langle L, F\rangle$ to $\genfrac{}{}{0pt}{}{L}{F}$.

We define the set $X_{T,A}$ of *approximations* to consist of all finite sequences

$$\left\langle \genfrac{}{}{0pt}{}{L_n}{F_n}, \ldots, \genfrac{}{}{0pt}{}{L_1}{F_1}, \genfrac{}{}{0pt}{}{\bullet}{F_0} \right\rangle \tag{1}$$

where $n \leq 0$, $\bullet$ is a *dummy* literal, the atoms of all literals are in A, the *free* literals $L_n, \ldots, L_1$ are consistent and pairwise different. The the sets of *forced* literals $F_n, \ldots, F_1, F_0$ must be consistent and pairwise disjoint. Moreover, the free and forced literals must also be disjoint. Finally, we require that $F_n, L_n, \ldots, F_1, L_1, F_0$ be a forcing condition and that the sets F_i consist of forced literals, i.e. that $F_i \subseteq F_{T,A}(L_i, \ldots, L_1)$ for all $i \leq n$.

With the approximation (1) we will be trying to solve the problem $L_n, \ldots, L_1 \Vdash \neg A$ which is equivalent to $F_n, L_n, \ldots, F_1, L_1, F_0 \Vdash \neg A$ by 3.1(1). We will start with the approximation $\left\langle \genfrac{}{}{0pt}{}{\bullet}{\emptyset} \right\rangle$ and with its problem $\emptyset \Vdash \neg A$.

We denote by $\#(T)$ the cardinality of the set T increased by 1 and assign to M the number of atoms in A increased by 2. We assign a *weight* to the approximation (1) as

$$\#(F_0) \cdot M^{M-0} + \#(F_1) \cdot M^{M-1} + \cdots + \#(F_{n-1}) \cdot M^{M-(n-1)} + \#(F_n) \cdot M^{M-n}$$

Note that the weight is always less than M^{M+1}.

3.3 Approximation trees. Suppose that we are given a set $X_{T,A}$ of approximations and a binary relation $\prec$ over $X_{T,A}$ which *increases the weight of approximations*, i.e. if $U \prec V$ then the weight of U is less than the weight of V.

We define the set $Tr_{T,A,\prec}$ to consist of all finite sequences

$$\langle U_0, \ldots, U_k \rangle \tag{1}$$

where U_0, U_1, …, U_k are from $X_{T,A}$, $U_0 = \left\langle {\bullet \atop \emptyset} \right\rangle$, and $U_i \prec U_{i+1}$ for all $i < k$.

We call $Tr_{T,A,\prec}$ the *tree of approximations* because its definition makes it a tree in the set-theoretical sense with $\left\langle {\bullet \atop \emptyset} \right\rangle$ its *root*. The sequences $\langle U_0, \ldots, U_k, U_{k+1} \rangle$ where $U_k \prec U_{k+1}$ are *sons* of (1). A *path* through the tree $Tr_{T,A,\prec}$ is a possibly infinite sequence $\langle U_0, \ldots, U_k, \ldots \rangle$ where $\langle U_0, \ldots, U_k \rangle \in Tr_{T,A,\prec}$ for all k. Because the weight of approximations cannot increase beyond M^{M+1}, every path in the tree must be finite and be thus an element of the tree.

3.4 Approximation trees for non-lazy DPLL SAT-solving. For a given consistent theory T in $\mathcal{L}$ and its Kripke structure $\langle P, \leq, \Vdash \rangle$, we will now define the relation $\prec_A$ for every formula A from $\mathcal{L}$. The relation is defined over the set of approximations $X_{T,A}$ to satisfy two clauses. The first clause specifies *free extensions*:

$$\left\langle {L_n \atop F_n}, \ldots, {L_1 \atop F_1}, {\bullet \atop F_0} \right\rangle \prec_A \left\langle {L_{n+1} \atop \emptyset}, {L_n \atop F_n}, \ldots, {L_1 \atop F_1}, {\bullet \atop F_0} \right\rangle$$

where the atom of the literal L_{n+1} does not occur on the left.

The second clause specifies *forced contractions*:

$$\left\langle {L_n \atop F_n}, \ldots, {L_i \atop F_i}, \ldots, {L_1 \atop F_1}, {\bullet \atop F_0} \right\rangle \prec_A \left\langle {L_i \atop L, F_i}, \ldots, {L_1 \atop F_1}, {\bullet \atop F_0} \right\rangle$$

where $i \leq n$, $L \notin F_i$, and $L_i, \ldots, L_1 \Vdash_T A \to L$.

It should be clear that the relation $\prec_A$ increases the weight of approximations and thus $Tr_{T,A,\prec_A}$ is a tree of approximations.

3.5 DPLL procedures for non-lazy SAT solving. We call a satisfiability solving procedure a *non-lazy DPLL SAT-solving procedure* if for a given T in an $\mathcal{L}$ the procedure on input A goes through states which can be successively mapped on the approximations from a path of the approximation tree $Tr_{T,A,\prec_A}$. Although the internal states of the procedure can be arbitrary, it suffices that the computation of the procedure can be identified with a path in the tree.

We will see below that when T is empty then our DPLL procedures are comparable with the *basic DPLL procedures* from [5]. With T non-empty our procedures can be compared with the *DPLL procedures with non-exhaustive propagation.*

3.6 Theorem (Characterization of non-lazy DPLL SAT solving). *For any tree $Tr_{T,A,\prec_A}$ of approximations and any path*

$$\left\langle \left\langle {\bullet \atop \emptyset} \right\rangle, \ldots, \left\langle {L_n \atop F_n}, \ldots, {L_1 \atop F_1}, {\bullet \atop F_0} \right\rangle \right\rangle$$

through it, the forcing condition $p = F_n, L_n, \ldots, F_1, L_1, F_0$ decides all atoms of A and we have

1. *if $V(p) \vDash A$ then $T, p \vDash A$,*
2. *if $V(p) \nvDash A$ then $n = 0$ and no model of T satisfies A.*

Proof. Call the last approximation of the path U. We prove first that p decides all atoms of A. By way of contradiction assume otherwise. Thus there is an atom B in A such that neither B nor $\neg B$ are in p and we consider two cases. If T, B, p is consistent then $B, p < p$ and the free extension would apply to U with $L_{n+1} = B$. If T, B, p is inconsistent then $T, p \vDash \neg B$ and hence $p \Vdash \neg B$ by Thm. 2.5(1). But then also $p \Vdash A \to \neg B$ and forced contraction would apply to U with $i = n$ and $L = \neg B$.

The property (1) follows trivially from Thm. 2.5(3).

For the proof of the property (2) we assume $V(p) \nvDash A$. Then $V(p) \vDash \neg A$ and we get $p \Vdash \neg A$ by Thm. 2.5(3), by monotonicity we have $F_i \subseteq F_{T,A}(L_n, \ldots, L_1)$ for all $i \leq n$, so we get $L_n, \ldots, L_1 \Vdash \neg A$ by 3.1(1). We now consider two cases. If $n > 0$ then $L_{n-1}, \ldots, L_1 \Vdash A \to \bar{L}_n$ by Thm. 2.5(2). But this would mean that forced contraction would apply to the approximation U with $i = n - 1$ and $L = \overline{L}_n$. Therefore it must be the case that $n = 0$ and we have $\emptyset \Vdash \neg A$. We point out to the reader familiar with [5] that we are now in the *fail* state. We take any model V of T and form the finite set:

$$q = \{L \mid \text{the atom of the literal } L \text{ occurs in } A \text{ and } V \vDash L\} .$$

Since $V \vDash q$, V is also a model of T, q and so q is a forcing condition deciding all atoms of A and such that $q \Vdash \neg A$ by the monotonicity of forcing. Thus $V(q) \vDash \neg A$ by Thm. 2.5(3) and hence $V \nvDash A$ by coincidence of V and $V(q)$. □

3.7 Discussion. The reader will note that in the Characterization theorem the only constraint on the theory T is consistency. Neither do we put any restriction on the formula A. However, in order to have computable non-lazy DPLL procedures we must require that T be decidable in the sense that for any finite set of literals p we can test whether T, p is consistent or not. In this way we can test whether p is a forcing condition and this is sufficient for the DPLL procedures to be computable. The exponential complexity of our DPLL procedures (as can been from the weight of approximations) cannot be escaped unless, of course, $P = NP$ and then we probably would have a fundamentally different procedure which would not need to be presented in the DPLL style of operation. In the following discussion we use the terminology of [5].

By restricting the conditions to those consistent with T we postulate that our DPLL procedures be non-lazy in the sense that they will never venture into a world which is not consistent with the theory T. [5] permits conditions which are not consistent with T but by lazy theory learning rules one can always recover from such a condition by a conflict with the learned theory lemma. Actually this is what always happens in practice ([5]).

In the view of the exponential character of DPLL SAT solving, it seemingly cannot matter in what form the formula A is presented to the solver, or in which way the solver tests for literals which are candidates for forced contractions, or even how high the solver back-jumps in forced contractions (pure backtracking with $i = n-1$ would be sufficient). Yet, and this is rather surprising, SAT-solvers are being used in the design and verification of processors with spectacular success. Thus it must pay to optimize.

In the actual practice the formula A is always presented to the solver in a conjunctive normal form as a conjunction of clauses which are disjunctions of literals. It always pays to apply eagerly the unit propagation rule of [5]. In our framework this means that we are trying to solve the problem $p \vDash \neg A$ with $p = L_n, \dots, L_1$ and a literal L in a clause $L \vee C$ of the formula A is forced, i.e. $p \Vdash L \vee C \to L$ holds. This is possible only if $p \Vdash L \vee \neg C$ and this in turn is the case iff for all L' occurring in C we have $p \Vdash \bar{L}'$. The reader will note that from this we have $p \Vdash A \to L$ because in any $q \leq p$ when $q \Vdash A$ we also have $q \Vdash L \vee C$ and hence $q \Vdash L$. Thus the solver should use forced contraction with the literal L and $i = n$.

All practical solvers when doing the above test $p \Vdash \neg C$ will use the full condition $q = F_n, L_n, \dots, F_1, L_1, F_0$ for which $q \Vdash \neg C$ is equivalent to $p \Vdash \neg C$. In the condition q the solver will look whether for all literals L' occurring in C it is the case that $\bar{L}' \in q$. A positive outcome of this test is sufficient to conclude $p \Vdash \neg C$ but it is by no means necessary.

Forced contraction with $i = n$ also applies when $F_n, L_n, \dots, F_1, L_1, F_0 \Vdash L$. This is equivalent to $T, F_n, L_n, \dots, F_1, L_1, F_0 \vDash L$ by Thm. 2.5(1) and so it corresponds to the theory propagate rule of [5]. That the forced expansion is applicable can be see from the fact that $F_n, L_n, \dots, F_1, L_1, F_0 \Vdash L$ is equivalent to $L_n, \dots, L_1 \Vdash L$ by 3.2(1). From the last $L_n, \dots, L_1 \Vdash A \to L$ follows because for any $q \leq L_n, \dots, L_1$ where $q \Vdash A$ we have $q \Vdash L$ by monotonicity.

Forced contraction with $i < n$ can simulate the back-jump rule of [5] which is used when (expressed in our notation) $T \vDash A \to C \vee L$ and $L_i, \dots, L_1 \Vdash \neg C$. We can weaken the first condition to $L_i, \dots, L_1 \Vdash A \to C \vee L$ from which we get similarly as above $L_i, \dots, L_1 \Vdash A \to L$.

3.8 Learned Lemmas. In [5] we read:

> ...the spectacular improvements in the performance of DPLL-based SAT solvers achieved in the last years are due to ... conflict-driven lemma learning...

The clause learning rule in [5], rephrased in our terminology, allows—by the processes of learning and forgetting—to switch equivalently between the solutions of the problems $p \Vdash \neg A$ and $p \Vdash \neg(A \wedge C)$ provided $T \vDash A \to C$.

In our framework we can prove a slightly stronger property where the assumption $T \models A \to C$, which amounts to $\emptyset \Vdash A \to C$, is changed to a weaker $p \Vdash A \to C$ but now with C an arbitrary formula:

if $p \Vdash A \to C$, then $p \Vdash A \to L$ iff $p \Vdash A \wedge C \to L$.

Indeed, we assume $p \Vdash A \to C$ and in the direction $(\to)$ also $p \Vdash A \to L$. For any $q \le p$ we assume $q \Vdash A \wedge C$. Thus $q \Vdash A$ and hence $q \Vdash L$. In the direction $(\leftarrow)$ we assume $p \Vdash A \wedge C \to L$ and for any $q \le p$ we assume $q \Vdash A$. Thus $q \Vdash C$ from the main assumption, then $q \Vdash A \wedge C$ and hence $q \Vdash L$.

This property allows any non-lazy DPLL procedure to remember the lemma C in the world when it is trying to solve $p \Vdash \neg A$ (the lemma can be remembered somewhere in the internal data structures) and used in the future, i.e. when the procedure is solving the problem $q \Vdash \neg A$ for $q \le p$, to determine the literals L in forced contractions. The procedure can derive the literal L by using $q \Vdash A \wedge C \to L$ instead of the usual $q \Vdash A \to L$. Note that because of monotonicity, our lemma learning property applies to all $q \le p$. The procedure can decide to forget and relearn the lemma many times but it must be before it leaves the worlds below p. Note that if the procedure wishes to use a lemma C anywhere it suffices that it proves $\emptyset \Vdash A \to C$.

4 Conclusions

We did not attempt to analyze all DPLL strategies presented in [5]. We have analyzed just those which seem to us to be the most elegant to explain by forcing. As it happens, those are exactly the procedures DPLL(T) described in [4]. As we have argued in Paragraphs 3.7 and 3.8, our non-lazy procedures can with $T = \emptyset$ simulate the basic DPLL procedures from [5], and with T non-empty they can simulate those DPLL procedures with non-exhaustive propagation which immediately recover from a theory inconsistent condition by theory learning with a subsequent recovery from the conflict by a fail or a back-jump.

If the reader wishes to admit lazy theory procedures then he should work with the Kripke structure for the empty theory $\emptyset$ where arbitrary, but consistent, conditions are permitted. The two clauses defining the relation $\prec_A$, would then look as follows:
Free extensions:

$$\left\langle {}^{L_n}_{F_n}, \ldots, {}^{L_1}_{F_1}, {}^{\bullet}_{F_0} \right\rangle \prec'_A \left\langle {}^{L_{n+1}}_{\emptyset}, {}^{L_n}_{F_n}, \ldots, {}^{L_1}_{F_1}, {}^{\bullet}_{F_0} \right\rangle$$

Forced contractions:

$$\left\langle {}^{L_n}_{F_n}, \ldots, {}^{L_i}_{F_i}, \ldots, {}^{L_1}_{F_1}, {}^{\bullet}_{F_0} \right\rangle \prec'_A \left\langle {}^{L_i}_{L,F_i}, \ldots, {}^{L_1}_{F_1}, {}^{\bullet}_{F_0} \right\rangle$$

provided $F_n, L_n, \ldots, F_1, L_1, F_0 \Vdash_\emptyset A \to L$ or $T, F_n, L_n, \ldots, F_1, L_1, F_0 \models L$. The approximation trees will be $Tr_{\emptyset,A,\prec'_A}$.

References

1. Jeremy Avigad. *Forcing in Proof Theory.* Bulletin of Symbolic Logic, 10:305–333, 2004.
2. George S. Boolos and Richard C. Jeffrey. *Computability and Logic—2^{nd} ed.* Cambridge University Press, Cambridge, 1980.
3. Martin Davis, George Logemann, and Donald Loveland. *A machine program for theorem proving.* Communications of the ACM, 5(7):394–397, 1962.
4. Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast Decision Procedures. *16th International Conference on Computer Aided Verification (CAV)*, Boston, July 2004.
5. Robert Nieuwenhuis, Alberto Oliveras, and Cesare Tinelli. Abstract DPLL and Abstract DPLL Modulo Theories. *11th International Conference on Logics for Programming, AI and Reasoning (LPAR)*, Montevideo, March 2005.
6. Joseph R. Shoenfield. *Mathematical Logic.* Addison-Wesley, Reading, Massachusetts, 1967.
7. Cesare Tinelli. A DPLL-based calculus for ground satisfiability modulo theories. In *Proceedings of the 8th European Conference on Logic in Artificial Intelligence*, LNAI 2424, pp. 308–318, 2002.