Accelerating lemma learning using joins - DPLL($\sqcup$)

Nikolaj Bjørner
Microsoft Research

Bruno Dutertre
SRI International

Leonardo de Moura
Microsoft Research

Abstract

State-of-the-art satisfiability modulo theory solvers use a combination of the Davis-Putnam-Logemann-Loveland (DPLL) procedure for performing Boolean search and an integration of theory solvers for identifying theory conflicts. Theory conflicts are presented as clauses over the propositional vocabulary that prune the DPLL search. This combination is often highly effective, as propositional reasoning is handled by state-of-the-art methods for propositional satisfiability, while theory solvers can be invoked incrementally as the DPLL core asserts literals. However, there are several cases where this integration misses short proofs if the short proofs require additional literals that are not part of the input. We present a method based on joins for identifying a sufficient basis of additional literals and lemmas that can speed up proof search for DPLL with theories exponentially. We then compare variants of the proposed methods with proof systems based on superposition and resolution. The theoretical result is that general formulations of joins are equivalent in succinctness to superposition and resolution.

1 Introduction

Abstract interpretation and theorem proving are both used in program verification but they traditionally approach the problem from different perspectives. Abstract interpretation focuses on automatically *generating* program invariants [1] whereas theorem proving is used to *verify* that given assertions are invariant. However, the concept of *logical interpretation* [5] shows that deductive methods based on theorem proving can be used to build abstract interpreters. In this paper, we examine the opposite issue, namely, the use of abstraction techniques in automated theorem proving. More specifically, our goal is to improve performance of Satisfiability Modulo Theory (SMT) solvers by generating useful lemmas using abstraction.

SMT solvers decide the satisfiability of formulas in logical theories such as linear arithmetic, the theory of arrays, and bitvectors. Most SMT solvers use the DPLL(T) architecture. They combine a Boolean satisfiability solver based on the Davis-Putnam-Logemann-Loveland procedure (DPLL) with a theory solver that can decide satisfiability of conjunctions of atoms in a specific theory T [2]. In the last few years, specialized theory solvers and the development of new integration methods have led to dramatic performance improvement in SMT solving. Still, there are “easy” formulas that cannot be solved efficiently using the standard DPLL(T) model, because the literals that are necessary for a short proof are not present in the original formula. This problem has been recognized in the setting of difference logic constraints [6], where a solution based on adding atoms based on transtivity of inequality was investigated. We present a method based on abstraction for cheaply discovering additional literals and lemmas. Essentially, the method discovers atomic facts that are implied by both side of a disjunction $\Phi_1 \vee \Phi_2$, which can drastically reduce the search space by avoiding extraneous case splits. Formula (1) motivated some of the techniques presented here. It is an abstraction of a pattern seen in verification conditions from program verification tools. The pattern corresponds roughly to propagating weakest preconditions over branch statements.

2 DPLL(T) as a Non-deterministic Transition System

The DPLL(T) procedure for satisfiability modulo theories is a combination of the DPLL algorithm for Boolean satisfiability and a theory solver for a theory T. In this paper, we focus on the quantifier-free

$$
\begin{array}{lll}
M \parallel F \implies M\ell^d \parallel F & \textbf{if} \begin{cases} \ell \text{ or } \overline{\ell} \text{ occurs in } F \\ \ell \text{ unassigned in } M \end{cases} & \textsf{(Decide)} \\[1em]
M \parallel F, C \vee \ell \implies M\ell^{C\vee\ell} \parallel F & \textbf{if} \begin{cases} \ell \text{ unassigned in } M \\ M \Vdash \neg C \end{cases} & \textsf{(UnitPropagate)} \\[1em]
M \parallel F \implies M\ell^{C\vee\ell} \parallel F & \textbf{if} \begin{cases} \ell \text{ unassigned in } M \\ \ell \text{ or } \overline{\ell} \text{ occurs in } F \\ T \vdash C \vee \ell \\ M \Vdash \neg C \end{cases} & \textsf{(}T\textsf{-Propagate)} \\[1em]
M \parallel F, C \implies M \parallel F, C \parallel C & \textbf{if } M \Vdash \neg C & \textsf{(Conflict)} \\[1em]
M \parallel F \implies M \parallel F, C \parallel C & \textbf{if} \begin{cases} T \vdash C \\ M \Vdash \neg C \end{cases} & \textsf{(}T\textsf{-Conflict)} \\[1em]
M \parallel F \parallel C' \vee \overline{\ell} \implies M \parallel F \parallel C \vee C' & \textbf{if } \ell^{C\vee\ell} \in M & \textsf{(Resolve)} \\[1em]
M\ell_0^d M' \parallel F \parallel C \vee \ell \implies M\ell^{C\vee\ell} \parallel F, C \vee \ell & \textbf{if } M \Vdash \neg C & \textsf{(Backjump)}
\end{array}
$$

Figure 1: Abstract DPLL(T) Procedure

theory of pure equalities (called E).

Given a quantifier-free formula ϕ, we denote that ϕ is valid in T by $T \vdash \phi$. A theory solver for T is an algorithm for deciding the satisfiability of conjunction of ground literals of T. Dually, a theory solver can decide whether $T \vdash \ell_1 \vee \ldots \vee \ell_n$ holds, where $\ell_1, \ldots, \ell_n$ are ground literals.

The DPLL(T) procedure starts with a formula ϕ written in conjunctive normal form. It searches for a truth assignment that satisfies all the clauses of ϕ and is consistent with respect to theory T. The search can be described by the transition system of Figure 1. The system states are of the form $M \parallel F$ or $M \parallel F \parallel C$ where M is a partial truth assignment, F is a set of clause, and C is a clause.

The assignment M is represented as a finite sequence of the form $\ell_1^{e_1} \ldots \ell_n^{e_n}$, where ℓ_i is a literal and e_i is an *explanation*. For every i, the explanation is either the symbol d, in which case ℓ_i is a *decision literal*, or a clause C that explains why ℓ_i must be assigned. The explanation clause is used during conflict resolution. The assignment is implicitly divided in segments of successive decision levels, where the decision level of a literal ℓ_i is the number of decisions in M prior to its occurrence. In states of the form $M \parallel F$, the procedure attempts to extend the current truth assignment by using the unit and theory propagation rules or the decision rule. A conflict is detected when the assignment M falsifies a clause C of F (rule Conflict) or when M is not consistent with respect to the theory (rule T-Conflict). In both cases, the system moves to a conflict state of the form $M \parallel F \parallel C$. In any such state, it can be shown that the clause C is false in M (written $M \Vdash \neg C$). The rules Resolve and Backjump correspond to *conflict-driven clause learning* employed by modern SAT solvers. Resolve constructs a new conflict clause $C \vee C'$ by applying resolution. Backjump is applicable when the conflict clause has a unique literal ℓ of maximal decision level. The conflict clause is then added to F, backtracking is performed (i.e., literal assignments are undone), then ℓ is assigned as implied by $C \vee l$ and the search can continue from a consistent state.

DPLL(T) terminates when none of the rules of Figure 1 is applicable. This can happen in a state $M \parallel F$ where all literals of F are assigned. In such a case, M is a full assignment that satisfies all the

$$\textsf{Sup}\ \frac{C \vee a \simeq b \qquad D[a]}{C \vee D[b]} \quad \textsf{E-Res}\ \frac{C \vee a \not\simeq a}{C} \quad \textsf{E-Fact}\ \frac{C \vee a \simeq b \vee a \simeq c}{C \vee a \simeq b \vee b \not\simeq c}$$

$$\textsf{Res}\ \frac{C \vee \ell \qquad D \vee \neg\ell}{C \vee D} \quad \textsf{Fact}\ \frac{C \vee \ell \vee \ell}{C \vee \ell}$$

Figure 2: The $\mathcal{SP}(E)$ calculus

clauses of F and is consistent with respect to T. In other words, the initial formula ϕ is satisfiable. The other terminal states are of the form $M \parallel F \parallel \Box$ where $\Box$ is the empty clause. In such a case, ϕ is unsatisfiable. Proof of termination and details can be found in [2].

2.1 A Superposition Calculus $\mathcal{SP}(E)$

Figure 2 summarizes basic superposition inference rules for the theory of pure ground equalities. It is a simple instance of more general and complete superposition calculi for the first-order theory of equality, but in this paper we will only consider equalities between constants, and we omit ordering constraints in side conditions on the rules ($\mathcal{SP}(E)$ is finitely saturating without orderings). By $D[a]$ we refer to the 0 or more, but not necessarily all, of the a positions in D, these selected as are replaced by b in $D[b]$.

3 A Hard Formula for DPLL(E)

Consider the unsatisfiable formula (1) (and illustrated in Figure 3) also used in [3], and present in the 2008 SMT competition for the EUF division (`http://www.smtcomp.org`).

$$a_1 \not\simeq a_{50} \;\wedge\; \bigwedge_{i=1}^{49} [(a_i \simeq b_i \wedge b_i \simeq a_{i+1}) \;\vee\; (a_i \simeq c_i \wedge c_i \simeq a_{i+1})] \tag{1}$$

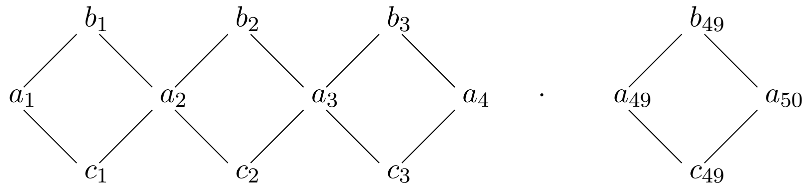

Figure 3: Diamond equalities

The formula is unsatisfiable because in every diamond, it is the case that $a_i \simeq a_{i+1}$ because either $a_i \simeq b_i \wedge b_i \simeq a_{i+1}$ or $a_i \simeq c_i \wedge c_i \simeq a_{i+1}$. Therefore, by repeating this argument for every i, we end up with the implied equality $a_1 \simeq a_{50}$. This contradicts the disequality $a_1 \not\simeq a_{50}$. A proof search method directly based on DPLL(E) is not able to produce a succinct proof like the informal justification just given. In a propositional abstraction of the problem, each of the equalities $a_i \simeq b_i$, $b_i \simeq a_{i+1}$, $a_i \simeq c_i$, $c_i \simeq a_{i+1}$ and $a_1 \simeq a_{50}$ is treated as a propositional variable. Because the atoms $a_i \simeq a_{i+1}$ are not present DPLL assigns truth values to the propositional variables, and a decision procedure for equalities detects a contradiction only when for every $i = 1, \ldots, 49$ $a_i \simeq a_{i+1}$ follows from either $a_i \simeq b_i \wedge b_i \simeq a_{i+1}$ or $a_i \simeq c_i \wedge c_i \simeq a_{i+1}$. There are 2^{49} different such equality conflicts, none of which subsumes the other. There is no short unsatisfiability proof that uses only the original atoms.

On the other hand, the formula has a short proof in $\mathcal{SP}(E)$. More, generally, every proof in DPLL(E) can be simulated by a proof of equal length in $\mathcal{SP}(E)$, but not conversely. We write $F_1 \preceq F_2$ if every proof in the formal system F_2 can be reduced to a proof in the formal system F_1 using at most a polymomial overhead; $\equiv$ is used if reduction is possible in both directions and $\prec$ holds if the reduction only holds in one direction. To summarize:

Theorem 3.1. $\mathcal{SP}(E) \prec$ *DPLL(E).*

4 A Sufficient Basis of Literals

There is a very simple way of augmenting DPLL(T) to allow it to simulate $\mathcal{SP}(E)$: First create the set Δ consisting of all literals of the form $a \simeq b$, where a and b are constants in the original formula F. Then allow these literals to participate in the Decide and propagation rules. For reference, we call the resulting system DPLL($E + \Delta$).

Theorem 4.1. $\mathcal{SP}(E) \equiv$ *DPLL($E + \Delta$). In particular, any superposition inference can be simulated by DPLL($E + \Delta$).*

The set Δ is quadratic in the size of the input, so additional techniques are needed to make this approach efficient, but then neither does $\mathcal{SP}(E)$ provide any built-in guidance.

5 A Solution Based on Joins

Our approach to solving such problems efficiently is based on ideas from abstract interpretation. It is based on the availability of a *join* operator on constraints maintained by theory solvers to discover atomic facts that are implied by both sides of a disjunction or case split.

To describe the basic procedure in the context of DPLL(T), consider a state $M \parallel F$ where M does not contain decision literals (literals annotated as ℓ^d). We can then choose an unassigned propositional variable p; first assign it to `true`, perform UnitPropagate and T-Propagate to derive all consequences of p to obtain the context M_1, second assign p to `false` and perform the same propagation to obtain the context M_2. We then use an operator $\sqcup$ such that $M_1 \sqcup M_2$ is a set of literals that are implied by M_1 and M_2 to compute a joint set of implied literals. The rule can be formulated in the context of the abstract transition system for DPLL(T) as an inference rule $\sqcup_1^1$:

$$\frac{\begin{array}{c} Mp^d \parallel F \Longrightarrow M_1 \parallel F \qquad M\neg p^d \parallel F \Longrightarrow M_2 \parallel F \\ p \text{ is the only decision variable in } M_1, M_2 \end{array}}{M \parallel F \Longrightarrow M_1 \sqcup M_2 \parallel F} \sqcup_1^1$$

For the propositional case, the resulting system is reminiscent of Stålmarck's method [4], except, that method also allows learning equivalences between literals. The rule allows some proof-acceleration in formulas like (1), but it is also limited as we have:

Theorem 5.1. $\mathcal{SP}(E) \prec$ *DPLL($E + \sqcup_1^1$) $\prec$ DPLL(E).*

5.1 Joining Equalities

Let E be the equivalence classes of a set of constants at state M. So for every $e, e' \in E$ if $e \neq e'$ then $e \cap e' = \emptyset$, and $\bigcup E$ consists of all the constants in M. For a given constant t, associate $E(t)$ as the class in E such that $t \in E(t) \in E$. We can characterize the join of two partitions as:

$$E_1 \sqcup E_2 \quad := \quad \{E_1(t) \cap E_2(t) \mid t \in \bigcup E\} \tag{2}$$

Also, the set of equalities associated with a partition is then just a spanning tree of equalities per equivalence class.

5.2 Generalized Join

There is an obvious limitation to the rule $\sqcup_1^1$: It can only be applied when M does not contain decision literals. Consequently, it allows only learning units facts. The limitation is on purpose: the rule requires at most a quadratic number of applications (based on the number of atoms in F) to either assign all literals, or saturate. The more generic formulation of the inference rule is to allow it being applied at any level and add new literals to M without these being unit facts. For reference, we will call this system DPLL($E + \sqcup^\omega$). The definition of joins will then have to be adjusted so that explanations are tracked correctly when literals are joined. We will not give the full details of DPLL($E + \sqcup^\omega$), instead we will arrive at a system that is equally succinct as the one just sketched. But we do so the *hard* way to examine the limitations of the more conservative liftings of DPLL($E + \sqcup_1^1$).

5.3 k-lookaheads

The rule $\sqcup_1^1$ allows for splitting on a single atom p. The implied consequences of the different cases for p are then combined. We say that this approach uses *one lookahead*. One lookahead is not always sufficient for learning the right implied facts. Consider a simple extension of the diamond problem given in equation (3), and illustrated in Figure 4.

$$a_1 \not\simeq a_{50} \;\wedge\; \bigwedge_{i=1}^{49} \left[\begin{array}{ll} & (a_i \simeq b_i \wedge b_i \simeq a_{i+1}) \\ \vee & (a_i \simeq c_i \wedge c_i \simeq a_{i+1}) \\ \vee & (a_i \simeq d_i \wedge d_i \simeq a_{i+1}) \end{array} \right] \tag{3}$$

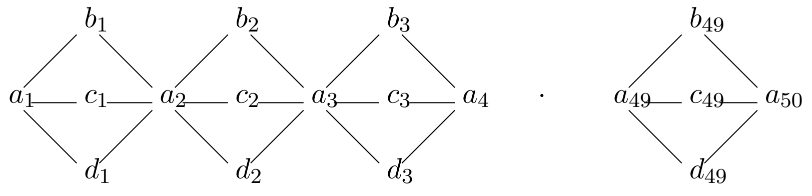

Figure 4: Double diamond equalities

In order to learn that $a_0 \simeq a_1$ we now need two splits. The obvious generalization of rule $\sqcup_1^1$ is to combine multiple branches in a join. We call the resulting system DPLL($\sqcup_k^1$) where k are the number of lookaheads admitted. Note that k lookaheads produce up to 2^k branches. It admits short proofs for formulas such as (3), but it can still be simulated by $\mathcal{SP}(E)$.

Theorem 5.2. *$\mathcal{SP}(E) \prec$ DPLL($E + \sqcup_k^1$) $\prec$ DPLL($E + \sqcup_1^1$) $\prec$ DPLL(E).*

5.4 m-disjunctions

The inequality in Theorem 5.2 is strict, as can be seen from the formula in (4) and Figure 5.

$$a_1 \not\simeq a_{50} \;\wedge\; \bigwedge_{i=1}^{49}(a_i \simeq a_{i+1} \;\vee\; a_i \simeq b_{i+1}) \;\wedge\; \bigwedge_{i=2}^{49}(b_i \simeq a_{i+1} \;\vee\; b_i \simeq b_{i+1}) \;\wedge\; b_{50} \simeq a_{50} \tag{4}$$

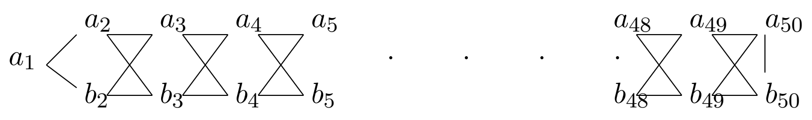

Figure 5: Butterfly equalities

Let DPLL($E + \sqcup_k^m$) be the extension of DPLL($E + \sqcup_k^1$) where join may return not only units but disjunctions with up to m literals. This is also known as *disjunctive join*. Instead of adding non-units to M the resulting (non-unit) clauses are added to F. We also don't need to examine 2^k branches because we can trade additional disjunctions for explored branches. Finally, the resulting system is equivalent to $\mathcal{SP}(E)$ in succinctness:

Theorem 5.3. $\mathcal{SP}(E) \equiv$ *DPLL($E + \sqcup_k^m$).*

6 Conclusions

We have examined the following equivalently succinct systems:

$$\mathcal{SP}(E) \;\equiv\; \text{DPLL}(E + \Delta) \;\equiv\; \text{DPLL}(E + \sqcup_k^m) \;\equiv\; \text{DPLL}(E + \sqcup^\omega)$$

so what is the difference in practice? The advantages of DPLL(T) have been the availability of space-efficient and adaptive search techniques developed in the context of SAT solvers. The advantage of using $\sqcup$ was that we could combine results from different branches into unit facts or lemmas. In future work we examine the more general problem of the quantifier-free theory of uninterpreted functions with equality, as well as describe applying the framework on selected theories, such as the theory of arrays. There are inherent theoretical limitations in the approaches studied so far. For example, the pigeon hole principle can be encoded as:

$$\left(\bigwedge_{i \le m} \bigvee_{j < m} d_i \simeq r_j \right) \;\wedge\; \bigwedge_{i < j \le m} d_i \not\simeq d_j \tag{5}$$

There are no short superposition proofs of unsatisfiability for this formula, but there are short proofs in Frege systems, which amounts to short proofs if arbitrary literals and definitions (cuts) can be introduced.

References

[1] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL-14*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[2] R. Niewenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[3] M. Rozanov and O. Strichman. Generating minimum transitivity constraints in P-time for deciding equality logic. In *SMT 2007*, volume 198 of *ENTCS*, pages 3–17, 2007.

[4] Mary Sheeran and Gunnar Stålmarck. A Tutorial on Stålmarck's Proof Procedure for Propositional Logic. *Formal Methods in System Design*, 16(1):23–58, 2000.

[5] A. Tiwari and S. Gulwani. Logical interpretation: Static program analysis using theorem proving. In F. Pfenning, editor, *CADE-21*, volume 4603 of *LNAI*, pages 147–166. Springer, 2007.

[6] Chao Wang, Aarti Gupta, and Malay Ganai. Predicate learning and selective theory deduction for a difference logic solver. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 235–240, New York, NY, USA, 2006. ACM.

A Proof Outlines

Sketch proof of Theorem 3.1. The theorem states that $\mathcal{SP}(E) \prec$ DPLL(E). Formula (1) shows that DPLL(E) $\not\preceq \mathcal{SP}(E)$, so it suffices to establish that $\mathcal{SP}(E) \preceq$ DPLL(E). Thus, every DPLL(E) proof can be directly simulated in $\mathcal{SP}(E)$. First notice that DPLL induces a propositional resolution proof. In fact the conflict resolution steps derive the conflict clause using a sequence of resolution steps based on clauses that annotate the literals in the context M. These clauses are either extracted from the original formula F or obtained from conflict resolution. Second, let us examine T-Propagate and T-Conflict. These rules supply additional T-lemmas (clauses) into the produced proofs. $\mathcal{SP}(E)$ cannot directly derive T-lemmas, so we cannot just replace these lemmas by superposition steps. Instead, consider a supoerposition proof-tree that contains T-lemmas. In the theory of equality all T lemmas are of the form $a_1 \simeq a_k \vee \bigvee_{i=1}^{k-1} a_i \not\simeq a_{i+1}$. We will show how to eliminate T-lemmas from a proof tree, starting from the lower-most occurrences of T-lemmas. So suppose that $a \simeq c \vee a \not\simeq b \vee b \not\simeq c$ is a lower-most T-lemma in a proof-tree. Then there are nodes labeled by $C \vee a \simeq b$, $D \vee b \simeq c$, and $E \vee a \not\simeq c$ that resolve with the literals from the T-lemma (and there are other nodes that resolve with literals in C, D and E). Apply rule Sup to $C \vee a \simeq b$, $D \vee b \simeq c$ to obtain the clause $C \vee D \vee a \simeq c$. Then apply Sup on the result and $E \vee a \not\simeq c$, to obtain $C \vee D \vee E \vee a \not\simeq a$. Use E-Fact to remove the last disequality. The remaining literals can be resolved using the same clauses that were used in the original proof. By repeating this argument, we can eliminate all T-lemmas. ☐

Remark 1. Note that we are giving $\mathcal{SP}(E)$ some flexibility. In particular, we do not refer to any term orderings in the side-conditions. The succinctness results for $\mathcal{SP}(E)$ would not work if one requires a total ordering on all constants and that the superposition rules respect these. For example, create the disjunction of formula (1) and another copy of it, but swap a_i and b_i in the second copy. We claim that a good ordering for the (1) is a bad ordering for the second copy, and vice versa. So the disjunction does not have a short proof if a total ordering on ground constants is required.

Sketch proof of Theorem 4.1. By case analysis, where we consider the rules specific to $\mathcal{SP}(E)$:

$$\textsf{Sup}\ \frac{C \ \vee\ a \simeq b \qquad D[a]}{C \vee D[b]}$$

It can be simulated in DPLL($E + \Delta$) in the following way: Use Decide to build the context with $\overline{C}$ and $\overline{D}[b]$. By unit propagation deduce $a \simeq b$. By congruence deduce $\overline{D}[a]$ (conflict with clause $D[a]$). Use all decided conflict resolution strategy to get $(C \vee D[b])$.

The all decided conflict resolution strategy consists in applying (Resolve) until the clause C in$M \parallel F \parallel C$ contains only decided literals.

$$\textsf{E-Res}\ \frac{C \ \vee\ a \not\simeq a}{C}$$

DPLL(T) keeps the clauses fully simplified.

$$\textsf{E-Fact}\ \frac{C \ \vee\ a \simeq b \ \vee\ a \simeq c}{C \ \vee\ a \simeq b \ \vee\ b \not\simeq c}$$

Use Decide to build the context with $\overline{C}, a \not\simeq b, b \simeq c$. By unit propagation deduce $a \simeq c$. From $a \not\simeq b$ and $a \simeq c$ deduce $b \not\simeq c$ (conflict) Use all decided conflict resolution strategy to get $C \vee a \simeq b \vee b \not\simeq c$.

So, any $\mathcal{SP}(E)$ proof can be simulated by DPLL($E + \Delta$).

The converse direction, that any DPLL($E + \Delta$) proof can be simulated by $\mathcal{SP}(E)$, follows from the proof of Theorem 3.1.

$\square$

Sketch proof of Theorem 5.1. Example (1) has a linear size proof in $\mathrm{DPLL}(E+\sqcup_1^1)$ but not in DPLL(E). Using Tseitsin's translation into clausal form, the sub-formula $a_1 \simeq b_1 \wedge b_1 \simeq a_2$ is associated with a predicate p_1, and similarly, the subformula $a_1 \simeq c_1 \wedge c_1 \simeq a_2$ is associated with a fresh predicate q_1, and the clauses $(p_1 \vee q_1)$, $(\neg p_1 \vee a_1 \simeq b_1)$, $(\neg p_1 \vee b_1 \simeq a_2)$, $(\neg q_1 \vee a_1 \simeq c_1)$, $(\neg q_1 \vee c_1 \simeq a_2)$ are added. Similarly, all the other conjunctions are represented using proxies. The proof in $\mathrm{DPLL}(E + \sqcup_1^1)$ is obtained by first splitting on p_1. In the branch where p_1 is asserted, both $a_1 \simeq b_1$ and $b_1 \simeq a_2$ are asserted. From these two equalities it follows that $a_1 \simeq a_2$. In the branch where $\neg p_1$ is asserted, unit-propagation over the clause $p_1 \vee q_1$ ensures that q_1 is asserted. Similarly $a_1 \simeq c_1$ and $c_1 \simeq a_2$ get asserted and therefore also $a_1 \simeq a_2$ is learned. It therefore follows that $\mathrm{DPLL}(E + \sqcup_1^1) \prec$ DPLL(E).

Formula (3) shows that $\mathrm{DPLL}(E + \sqcup_1^1) \not\preceq \mathcal{SP}(E)$ because the corresponding clausification of the formula produces instead of $(p_1 \vee q_1)$ the clause $(p_1 \vee q_1 \vee r_1)$, and adds $\neg r_1 \vee a_1 \simeq d_1$, $\neg r_1 \vee d_1 \simeq a_2$. Splitting on any of $\neg p_1$, $\neg q_1$ or $\neg r_1$ does not allow propagating any equalities because these assignments don't imply any equalities directly and the clause $(p_1 \vee q_1 \vee r_1)$ cannot yet be used for unit-propagation. Two splits are required to learn any equalities, and in particular learn that $a_1 \simeq a_2$.

We finally show that $\mathcal{SP}(E) \preceq \mathrm{DPLL}(E+\sqcup_1^1)$. Thus, we need to simulate proofs in $\mathrm{DPLL}(E+\sqcup_1^1)$ using $\mathcal{SP}(E)$. The new proof rule $\sqcup_1^1$is simulated by using the decision variable as the selected literal for resolution. The literals learned in one branch correspond to the clauses $p \vee \ell_i$, for $i = 1, \ldots, k$ for some k. The literals learned in the other branch correspond to clauses $\neg p \vee \ell'_j$, for $j = 1, \ldots, m$. All binary clauses in the cross-product can therefore be derived as well. Suppose $\ell \in M_1 \sqcup M_2$. Then, ℓ is already in M or there is a sequence of super-position steps from one of the binary clauses $\ell_i \vee \ell'_j$ such that factoring applies to produce a single learned literal.

$\square$

Sketch proof of Theorem 5.2. Formula (3) has a linear size proof in DPLL($E + \sqcup_k^1$) but not in $\mathrm{DPLL}(E + \sqcup_1^1)$. This establishes that DPLL($E + \sqcup_k^1$) $\prec$ $\mathrm{DPLL}(E + \sqcup_1^1)$. Formula (4) can be used to establish that DPLL($E + \sqcup_k^1$) $\not\preceq \mathcal{SP}(E)$.

Establishing that $\mathcal{SP}(E) \preceq$ DPLL($E + \sqcup_k^1$) is a direct extension of the argument for $\mathcal{SP}(E) \preceq$ $\mathrm{DPLL}(E + \sqcup_1^1)$: Consider the pairwise join of branches that have all but one assignment to a decision literal in common. The argument from Theorem 5.1 can be used in this case to derive a clause that contains the joined literal and all other decision variables. The clauses produced in this way can be resolved with each-other leaving just the new literals. $\square$

Sketch proof of Theorem 5.3. Establishing $\mathcal{SP}(E) \preceq$ DPLL($E + \sqcup_m^k$) follows by extending the arguments from the sketch proofs of Theorems 5.1 and 5.2. The difference is that we don't necessarily need to apply factoring to produce a unit literal. Similar to the proof of Theorem 4.1, we show that DPLL($E + \sqcup_m^k$) $\preceq \mathcal{SP}(E)$ examining each rule of $\mathcal{SP}(E)$.

Sup: To simulate Sup we guess first $\overline{C}^d$. This causes $a \simeq b$ to be added using UnitPropagate. The context is thus $\overline{C} a \simeq b \,\|\, F, C \vee a \simeq b, D[a]$. Then guess all variants of the literals in $D[a]$. The corresponding copies of $D[b]$ are implied by the equality $a \simeq b$. The resulting joined clause is the desired resolvent $C \vee D[b]$.

E-Fact: First guess $\overline{C}^d$, then guess $(a \not\simeq b)^d$. This causes $a \simeq c$ to be derived using UnitPropagate. The context is thus $\overline{C}^d, (a \not\simeq b)^d, a \simeq c \,\|\, F, C \vee a \simeq b \vee a \simeq c$. Then consider the other branch $\overline{C}^d, a \simeq b \,\|\, F, C \vee a \simeq b \vee a \simeq c$. This branch is consistent with the clause, but the join of their difference: $((a \not\simeq b)^d \wedge a \simeq c) \sqcup a \simeq b$ is the disjunction $a \simeq b \ \vee \ b \not\simeq c$ which we need for the result of E-Fact. The other literals from C are added as we only consider $\overline{C}^d$. ☐