FDPLL — A First-Order Davis-Putnam-Logeman-Loveland Procedure

Peter Baumgartner

Institut für Informatik, Universität Koblenz-Landau, D-56073 Koblenz, Germany
Net: `peter@uni-koblenz.de` , `http://www.uni-koblenz.de/~peter/`

Abstract. FDPLL is a directly lifted version of the well-known Davis-Putnam-Logeman-Loveland (DPLL) procedure. While DPLL is based on a splitting rule for case analysis wrt. ground and complementary literals, FDPLL uses a lifted splitting rule, i.e. the case analysis is made wrt. non-ground and complementary literals now.
The motivation for this lifting is to bring together successful first-order techniques like unification and subsumption to the propositionally successful DPLL procedure.
At the heart of the method is a new technique to represent first-order interpretations, where a literal specifies truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values. Based on this idea, the FDPLL calculus is developed and proven as strongly complete.

1 Introduction[1]

The well-known *Davis-Putnam* procedure, as it is usually called, was brought forward in the early 60s by the researchers mentioned in the title [DP60,DLL62,D63]. Nowadays, the procedure is most successfully applied to decide propositional problems, although it was originally conceived as a method for first-order theorem proving. To this end, successively increased sets of ground instances of first-order clauses are enumerated and fed into the propositional part of the procedure. This latter part is referred to as "propositional DPLL" in the sequel.

With the advent of the resolution calculus, the lifting of inference rules to the first-order level is standard in virtually all calculi and efficient proof procedures for first-order logic — except for Davis-Putnam-Logeman-Loveland methods. Thus, the purpose of this paper is to present a lifted version that fills this gap.

On an abstract level, the advantage of the "lifted" methods compared to the "propositional" methods stems from two sources: first, it is possible with a lifted method to finitely represent infinitely many inferences of the corresponding propositional methods, and, second, much more powerful redundancy elimination techniques are possible, e.g. based on subsumption. The motivation is to bring these advantages to DPLL. The other way round, FDPLL instantiates to propositional DPLL when applied to propositional logic.

[1] For a long version of the paper see `http://www.uni-koblenz.de/fb4/publikationen/gelbereihe/`.

Brief description of FDPLL. In order to describe the main idea of FDPLL, it is helpful to refer to the widely-used presentation of propositional DPLL as a calculus with a single *splitting* rule, which carries out a case analysis wrt. a propositional variable A. More exactly, the current clause set $\mathcal{S}$ splits into two cases: the one where A is "true", and the other where A is "false", which give rise to simplifications based on the new information. DPLL proceeds by considering different cases until success (some case is a model for $\mathcal{S}$) or failure (each considered case contradicts a clause in $\mathcal{S}$).

The idea in FDPLL is to lift this splitting to the first-order level, i.e. to split with complementary *non-ground* literals like $P(x, y)$ and $\neg P(x, y)$. The difficulty here is that the "usual" way of reading the literals as universally quantified (i.e. $\forall x, y\ P(x, y)$ and $\forall x, y\ \neg P(x, y)$) immediately leads to an *unsound* calculus. Hence, a different reading is adopted: a bit simplified, a literal, say $P(x, y)$, stands *by default* for all its ground instances, say, $P(a, a)$,$P(a, b)$,$P(b, a)$ and $P(b, b)$ (suppose here that only constants a and b are present). However, the presence of a strictly more specific literal (wrt. the instantiation order) than $P(x, y)$ with complementary sign, say $\neg P(x, b)$, gives rise to exceptions of the default reading of $P(x, y)$ by excluding all instances of $P(x, b)$. Symmetrically, $\neg P(x, b)$ stands by default for all its ground instances (with the possibility to have exceptions again). So, the two literals $P(x, y)$ and $\neg P(x, b)$ together stand for $P(a, a)$,$\neg P(a, b)$,$P(b, a)$ and $\neg P(b, b)$, which in turn can be understood as an interpretation $\mathcal{I}$ in the obvious way.

Now, a "case" in FDPLL is just a set of possibly non-ground literals, such that an interpretation can be associated to, as just sketched. Based on this idea, the purpose of the splitting rule of FDPLL can be explained as follows: suppose there is an instance $C\sigma$ of a clause C that is "false" in the interpretation $\mathcal{I}$ associated to the current case (σ is computed by most general unification). Then, a split is attempted with a literal $L \in C\sigma$ in order to "repair" $\mathcal{I}$ towards an interpretation that assigns "true" to L, and hence to $C\sigma$ as well[2]. If this is not possible because of some elementary contradiction between $C\sigma$ and the current case, the current case is refuted ("closed"). Otherwise, two new cases come up, the one extending the current case with L, and the other with $\overline{L}$.

Continuing the example above, suppose the current case is $\{P(x, y),\ \neg P(x, b)\}$, hence $\mathcal{I} = \{P(a, a),\ \neg P(a, b),\ P(b, a), \neg P(b, b)\}$, and suppose that there is a clause $C = P(x, y) \vee \neg P(x, a)$. The clause instance $C\sigma = P(x, b) \vee \neg P(x, a)$ is "false" in $\mathcal{I}$, where $\sigma = \{y/b\}$ is computed by most general unification of the literals of C and complements of literals of $\mathcal{I}$. Regarding the two literals $P(x, b)$ and $\neg P(x, a)$ in $C\sigma$, only $\neg P(x, a)$ is a candidate for splitting, because $P(x, b)$ is an elementary contradiction to $\neg P(x, b)$ of the current case.

The procedure repeatedly carries out splits in this way and stops if every case is refuted (and reports "unsatisfiable"), or if no clause instance $C\sigma$ of the mentioned kind exists (and reports the current case as a model representation)[3].

[2] Actually, this is a bit simplified, but it serves well to illustrate the idea.

[3] There is a second variant of the splitting rule called "Commit" with the purpose to achieve that $\mathcal{I}$ is indeed consistent.

An improvement of this "basic" procedure recovers for certain branch literals the above mentioned universally quantified reading (cf. Section 5). It lifts to the first-order level the well-known propositional DPLL rule for propagating unit clauses. In resolution terminology, the improvement realizes unit-resulting resolution. Fortunately, only minor modifications of the "basic" calculus are necessary for this.

Properties of FDPLL. Propositional DPLL has certain desirable features: its conceptual simplicity, space efficiency ("one branch at a time"), few inference rules (one is sufficient), efficient and adaptable implementations (the most efficient systematical propositional methods are based on DPLL, e.g. NTAB [CA96] and SATO [Zha97]), existence of non-clausal versions [BBOS98], and, the possibility to immediately extract a model in case that no refutation exists. A goal of this work is to keep these features for the lifted version FDPLL.

FDPLL is in particular space efficient, proof confluent and convergent (i.e. a strong completeness theorem holds). While these properties go without a saying for propositional DPLL, they *are* an issue for certain first-order methods, e.g. tableau and connection calculi (but see [BEF99] for a proof confluent strongly complete connection calculus). Beyond this, FDPLL is known to be a decision procedure for the Bernays-Schönfinkel class, i.e. clause logic without function symbols but constants. This is a non-trivial class, in the sense that most resolution and tableau systems cannot decide it, except in a trivial way by using the finite set of ground clauses.

Structure of the paper. After stating some preliminaries, the model representation technique is introduced. Based on it, the calculus is developed. Then soundness and and completeness is turned to, followed by a sketch of the mentioned "universal literal" improvement. The subsequent proof procedure proves the existence of a concrete, fair strategy. Finally, some conclusions are drawn, including related work.

2 Preliminaries

The usual notions of first-order logic are applied in a way consistent to [CL73]. A *literal* is an atom or a negated atom. The letters K and L are reserved to denote literals. The *complement* of a literal L is $\overline{L} = A$, if $L = \neg A$ for some atom A, or else $\overline{L} = \neg L$; by $|L|$ the atom of L is denoted, i.e. $|A| = A$ and $|\neg A| = A$ for any atom A. A *clause* is a finite, possibly empty multiset $\{L_1, \ldots, L_n\}$ of literals, usually written as a disjunction $L_1 \vee \cdots \vee L_n$. By a *clause set* always a finite set of clauses is meant. The letters C and D are reserved to denote clauses.

An *interpretation* $\mathcal{I}$ for a given signature Σ is a set of ground Σ-literals such that either $A \in \mathcal{I}$ or $\neg A \in \mathcal{I}$ for every ground Σ-atom A. The signature Σ is always given implicitly by the input clause set under consideration, and the prefix "$\Sigma-$" usually is not written. All the results below hold wrt. such Herbrand interpretations; it only has to be assumed that Σ contains at least one constant symbol (if none is there, some constant a is added artificially).

A ground literal L and a ground clause C is evaluated wrt. an interpretation $\mathcal{I}$ as expected, i.e. $\mathcal{I}(L) = \mathit{true}$ iff $L \in \mathcal{I}$ and $\mathcal{I}(C) = \mathit{true}$ iff $\mathcal{I}(L) = \mathit{true}$ for some $L \in C$. Furthermore, as expected, for a non-ground clause C define $\mathcal{I}(C) = \mathit{true}$ iff $\mathcal{I}(C') = \mathit{true}$ for every ground instance C' of C. As usual, $\mathcal{I} \models X$ means $\mathcal{I}(X) = \mathit{true}$ where X is a literal, clause or clause set (interpreted conjunctively).

A *unifier* for a set Q of terms (or literals) is a substitution δ such that $Q\delta$ is a singleton. The notion of *most general unifier (MGU)* is used in the usual sense [CL73, e.g.], and a respective unification algorithm *unify* is assumed as given. The notation $\sigma = \mathit{unify}(Q)$ means that an MGU σ of Q exists and is computed by *unify* applied to Q.

Quite frequently, a *simultaneous unifier for a set* $\{Q_1, \ldots, Q_n\}$ of unification problems is to be computed, which is a substitution δ that is a unifier for every $Q_1, \ldots, Q_n$. The notion of a *most general* unifier can be defined in the standard way in the simultaneous case as well. Further, a simultaneous most general unifier (simply called MGU as well) can be computed by iterative application of *unify* to $Q_1, \ldots, Q_n$. See [Ede85] for a thorough treatment. Thus, we may suppose as given a simultaneous unification algorithm *s-unify* and write $\sigma = \mathit{s\text{-}unify}(\{Q_1, \ldots, Q_n\})$ in analogy to $\sigma = \mathit{unify}(Q)$ above.

For literals K and L define $K \gtrsim L$, K *is more general than* L, iff there is a substitution σ_K such that $K\sigma_K = L$; K and L are *variants*, written as $K \sim L$, iff $K \gtrsim L$ and $L \gtrsim K$; K is *strictly more general* than L, $K > L$, iff $K \gtrsim L$ and not $K \sim L$. L is also said to be a *strict*, or *proper* instance of K then. If neither $K \gtrsim L$ nor $L \gtrsim K$ then K and L are *incomparable*. Finally, define $L \in^{\sim} N$ iff $L \sim K$ for some $K \in N$, where N is a set of literals.

3 Basic Concepts Related to Literal Sets

As mentioned, interpretations shall be represented by literal sets. This section contains the respective definitions. In the sequel N always denotes a possibly infinite literal set.

Definition 1 (Most Specific Generalization). *A literal K is called a* most specific generalization (MSG) of a literal L wrt. N *iff $K \gtrsim L$ and there is no $K' \in N$ such that $K > K' \gtrsim L$.*

Notice that nothing is said whether $K, L \in N$ or not.

Example 1. Consider[4] $N_1 = \{P(a, y, u), P(x, b, u)\}$. Then both $P(a, y, u)$ and $P(x, b, u)$ are MSGs of $P(a, b, c)$ wrt. N. This shows that MSGs need not be unique. The literal $P(x, y)$ is not a MSG of $P(y, f(x))$ wrt. $\{P(x, f(y))\}$, because $P(x, y) > P(x, f(y)) \gtrsim P(y, f(x))$.

An MSG $K \in N$ of L wrt. N is a "potential reason" for L to be *true* in the interpretation associated to N, because $K \gtrsim L$ (as said in the introduction).

[4] Here and below, the letters $P, Q, R, \ldots$ denote predicate symbols, $a, b, c, \ldots$ denote constants, $f, g, h, \ldots$ denote non-constant function symbols, and $x, y, z, \ldots$ denote variables.

For efficiency reasons in FDPLL it is desirable to have as few such "reasons" as possible. Therefore, *most specific* generalizations are used.

Definition 2 (Productivity). *A literal K* produces L wrt. N *iff K is a MSG of L wrt. N and there is no $K' \in N$ such that $K > \overline{K'} \gtrsim L$. For a clause C,* K produces C wrt. N *iff K produces some literal $L \in C$ wrt. N. The set N* produces L *(resp. C) iff some $K \in N$ produces L (resp. C) wrt. N.*

Referring again to the introduction and above, this definition realizes the possibility to prevent an MSG K of L wrt. N to assign *true* to L, if there is a complementary literal in between (wrt. $\gtrsim$) K and L, as stated. An equivalent, more compact definition of "K produces L wrt. N" is that $K \gtrsim L$ and there is no literal $K' \in N$ such that $|K| > |K'| \gtrsim |L|$.

Example 2. Let $N_2 = \{P(a, y, u),\ \neg P(x, b, u),\ P(a, b, u)\}$. Then, $P(a, b, u)$ produces the literal $P(a, b, f(u))$ wrt. N_2. However, $P(a, y, u)$ does not produce $P(a, b, f(u))$ wrt. N_2 because $P(a, y, u)$ is not an MSG of $P(a, b, f(u))$ wrt. N_2. (since $P(a, b, u) < P(a, y, u)$ is an MSG of $P(a, b, f(u))$ wrt. N_2). The literal $\neg P(x, b, u)$ produces $\neg P(b, b, f(u))$ wrt. N_2 but does not produce $\neg P(a, b, f(u))$ (since $\neg P(x, b, u) > \overline{K'} \gtrsim \neg P(a, b, f(u))$, where $K' = P(a, b, u) \in N_2$).

Definition 3 (Ground expansion). *Define the* ground expansion of N *as* $[\![N]\!] = \{L \mid L$ *is a ground literal and* N *produces* $L\}$

The plan is to identify for a literal set N constructed by FDPLL its ground expansion $[\![N]\!]$ with an interpretation $\mathcal{I}$. Recall from Section 2 that an interpretation is a set of ground literals such that either $A \in \mathcal{I}$ or $\neg A \in \mathcal{I}$ for every ground atom A. However, there is in general no reason for $[\![N]\!]$ to be an interpretation. For instance:

Example 3. The set $N_3 = \{P(a, y, u),\ \neg P(x, b, u)\}$ produces both $P(a, b, c)$ and $\neg P(a, b, c)$. Hence, $[\![N_3]\!]$ is not an interpretation.

Note 1 (Completeness of N). Beyond this *inconsistency* problem, a *completeness* problem arises as well: for instance, $N = \{\}$ does not produce a single literal. The completeness problem can be solved by adding to N an expression $\neg x$, where x is a variable; the "literal" $\neg x \in N$ then acts as a default case to assign *false* to positive literals. Thus, for instance, $[\![\{\neg x, P(a)\}]\!]$ produces every negative literal except $\neg P(a)$ and thus assigns *false* to every positive literal, except $P(a)$[5].

The following definition formalizes these concepts.

Definition 4 (Contradictory, Consistent, Complete). *A literal set N is called* contradictory *iff there are literals $L, K \in N$ such that $L \sim \overline{K}$. The term "non-contradictory" means "not contradictory". N is called* consistent wrt. a literal L *iff N does not produce both L and $\overline{L}$; N is called* consistent *iff N is*

[5] Of course, instead of "$\neg x$", "x" could be taken as well, which would emphasize the use of negative clauses ("goals") in the calculus.

consistent wrt. every literal L. The term inconsistent *means "not consistent". N is called* complete *iff for every literal L, N produces L or $\overline{L}$. The term* incomplete *means "not complete".*

Example 4. For instance, N_3 from Example 3 is non-contradictory and inconsistent wrt. $P(a, b, u)$ (and hence wrt. $\neg P(a, b, u)$ as well). Adding either $P(a, b, u)$ or $\neg P(a, b, u)$ renders the set consistent (and hence non-contradictory, as is easily seen), and adding both renders the set contradictory and inconsistent wrt. $P(a, b, u)$ again. Each of these sets is incomplete, and adding $\neg x$ achieves completeness.

With these definitions, the intuition so far can be made precise:

Proposition 1 (Interpretation). *If $\neg x \in N$ then N is complete. If N is consistent and complete, then $[\![N]\!]$ is an interpretation.*

It can be noted that "productivity" and "modelship" are *not* related on the non-ground level. For instance, take $N = \{\neg x,\ \neg P(a),\ P(b),\ Q(a),\ \neg Q(b)\}$ and $C = P(x) \vee Q(x)$. Then $[\![N]\!] \models C$ but there is no $L \in N$ such that L produces C wrt. N. Conversely, $N = \{\neg x,\ P(a)\}$ produces $\neg P(x)$ but $[\![N]\!] \not\models \neg P(x)$.

As mentioned in the introduction, model candidates shall be given up when being "elementary contradictory" to an (instance of) an input clause. The following definition makes this precise (preliminarily):

Definition 5 (Closed, open). *A literal set N is* closed by *a clause C and substitution δ iff $\overline{L} \in^{\sim} N$ for every $L \in C\delta$; N is* closed by *C iff N is closed by C and some substitution δ. Finally, N is closed by a clause set $\mathcal{S}$ iff N is closed by some clause $C \in \mathcal{S}$. The term "open" means "not closed", and "N is open wrt. $\mathcal{S}$" means "N is not closed by $\mathcal{S}$".*

For instance, $N = \{P(x, y),\ Q(a)\}$ is closed by $C = \neg P(x, y) \vee \neg P(y, x) \vee \neg Q(z)$ and $\delta = \{z/a\}$, because for every literal in $C\delta$ there is a complementary variant in N.

Again, as mentioned in the introduction, substitutions shall be computed by FDPLL as most general substitutions. This holds in particular for the substitutions δ that allow to close a literal set. This motivates the following definition.

Definition 6 (Branch unifier). *Let $C = L_1 \vee \cdots \vee L_n$ be a clause. A substitution σ is called a* branch unifier *of C against N iff there are pairwise variable disjoint literals $K_1, \ldots, K_n \in^{\sim} N$, each variable disjoint from C, and such that the following holds:*

(i) $\sigma = \text{s-unify}(\{\{\overline{L_1}, K_1\}, \ldots, \{\overline{L_n}, K_n\}\})$, and
(ii) K_i produces $\overline{L_i\sigma}$ wrt. N, for $1 \leq i \leq n$.

If N is closed by C and σ, then σ is called a closing *branch unifier, otherwise σ is called a* falsifying *branch unifier.*

Item (i) realizes that substitutions are computed at a most general level. Item (ii) acts as a further relevancy filter, by excluding those branch literals K_i that unify with $\overline{L_i}$ but do not produce $\overline{L_i\sigma}$.

Example 5. Let $N_4 = \{P(a,y,u),\ \neg P(x,b,u),\ P(a,b,u)\}$ and $C = \neg P(a,c,z) \vee P(z,v,z)$. Take $K_1 = P(a,y,u) \in^{\sim} N_4$ and $K_2 = \neg P(x,b,u') \in^{\sim} N_4$. Observe that $\sigma = \{u/z,\ u'/z,\ v/b,\ x/z,\ y/c\}$ is a branch unifier of C against N_4, since σ is a simultaneous MGU for $\{\{P(a,c,z), P(a,y,u)\},\ \{\neg P(z,v,z), \neg P(x,b,u')\}\}$, and $K_1 = P(a,y,u)$ produces $P(a,c,z)\sigma = P(a,c,z)$ wrt. N_4, and furthermore $K_2 = \neg P(x,b,u')$ produces $\neg P(z,v,z)\sigma = \neg P(z,b,z)$. Further observe that σ is a falsifying branch unifier (i.e. N_4 is not closed by C and σ).

As a negative example, there is no branch unifier of $P(a,b,c)$ against N_4, because although item (i) in Def. 6 is satisfied by taking $K_1 = \neg P(x,b,u)$ and $\sigma = \{x/a,\ u/c\}$, item (ii) is violated, because $\neg P(x,b,u)$ does not produce $\neg P(a,b,c)$ wrt. N_4.

Branch unifiers are a purely syntactical concept, and existence of branch unifiers for *finite* literal sets N obviously is decidable. The following lemma then, read in the contrapositive direction, guarantees that N is a model for the given clause if no branch unifier exists (provided that N is consistent).

Lemma 1. *Let N be consistent and complete, and C be a clause. If $[\![N]\!] \not\models C$ then there is a branch unifier σ of C against N.*

Unfortunately, the converse of Lemma 1 does not hold: take $N = \{\neg x,\ P(a)\}$, and observe that $\sigma = \{x/P(y)\}$ is a branch unifier of $P(y)$ against N, but $[\![N]\!] \models P(y)$ (supposing that a is the sole ground term). As a consequence, the calculus occasionally computes branch unifiers, although there is no need to.

In order to take advantage of the previous lemma, consistency has to be achieved (there is a respective inference rule in FDPLL). Fortunately, consistency is also a syntactical property, and is decidable in the finite case as well. For the purposes here, the following lemma is sufficient (it can be strengthened):

Lemma 2. *Let N be a non-contradictory literal set. If N is inconsistent then there is a pair of variable disjoint, non-comparable literals $K, L \in^{\sim} N$ with opposite sign (i.e. neither $K \gtrsim \overline{L}$ nor $\overline{L} \gtrsim K$) such that (i) K and $\overline{L}$ are unifiable, i.e. $\sigma = unify(\{K, \overline{L}\})$ exists, and (ii) neither $K\sigma \in^{\sim} N$ nor $L\sigma \in^{\sim} N$.*

4 The FDPLL Calculus

In this section the inference rules based on branch unifiers are introduced. Recall from the previous section that branch unifiers are either "falsifying" or "closing". However, to close literal sets earlier, hence find shorter refutations, the FDPLL calculus uses a different notion of "closure":

Definition 7 (^{a}-closed, ^{a}-open). *Let a be any constant from the signature under consideration (or a "new" constant if none is supplied). By N^a denote the literal set obtained from N by replacing in every literal every occurrence of every variable by a. More formally*[6]*, $L^a = L\gamma$, where $\gamma = \{x/a \mid x \in var(L)\}$, and $N^a = \{L^a \mid L \in N\}$.*

[6] The function *var* returns the set of variables occurring in its argument.

The literal set N is ^{a}-closed by C and δ *iff* $\overline{L} \in N^a$*, for every* $L \in C\delta$*. The derived forms, as well as the term "*^{a}*-open" are defined analogously to "closed" (Def. 5).*

So, determing whether N is ^{a}-closed by C is a question of simultaneously matching all literals of C to complementary literals in N^a.

Note 2 ("^{a}*-closed" closes more branches).* It is not too difficult to see that whenever N is closed by C then N is ^{a}-closed by C as well. The converse, however, is not true: for instance, $\{\neg P(x,y)\}$ is ^{a}-closed by $P(x,y) \vee P(x,x)$, but not closed by $P(x,y) \vee P(x,x)$ (because $\neg P(x,x)$ cannot be instantiated to a variant of $\neg P(x,y)$).

Thus, by contraposition, if N is ^{a}-open wrt. C, N is open wrt. C as well.

In the sequel, $\mathcal{S}$ always denotes a finite clause set.

Definition 8 (Branch, branch set, selection functions). *A* branch p *is a possibly empty, finite set of literals. A* branch set $\mathcal{P}$ *consists of a finite set of branches. The branch set* $\mathcal{P}$ *is* closed *(by* C*, by* $\mathcal{S}$*) iff every* $p \in \mathcal{P}$ *is closed (by* C*, by* $\mathcal{S}$*). The term "open" means "not closed". Define* $\mathcal{P}$ *as* ^{a}-closed *(*^{a}-open*) in the expected way by using the* ^{a}*-versions instead.*

Assume as given a branch selection function *sel which maps any* ^{a}*-open branch set* $\mathcal{P}$ *wrt.* $\mathcal{S}$ *to one of its* ^{a}*-open branches. This branch is referred to as the* selected branch in $\mathcal{P}$*. On* ^{a}*-closed branch sets, sel may be undefined.*

Finally, assume as given a literal selection function *litsel*(C,p) *that maps a clause* C *and a branch* p *that is* ^{a}*-open wrt.* C *to some literal* $L \in C$ *such that neither* $L \in^{\sim} p$ *nor* $\overline{L} \in^{\sim} p$*, provided such a literal exists, and may be undefined otherwise.*

Notice that the empty branch set is closed (and hence ^{a}-closed) wrt. every $\mathcal{S}$, and that the branch set $\{\{\}\}$ is ^{a}-open (and hence open) wrt. $\mathcal{S}$, unless $\mathcal{S}$ contains the empty clause. The same holds for $\{\{\neg x\}\}$, as no clause contains a "literal" x.

The purpose of the two selection functions will become clear soon. Next, the two inference rules of FDPLL are defined.

Definition 9 (Split Inference Rule). *The following inference rule* Split *transforms a branch* p*, a clause* C *such that* p *is* ^{a}*-open wrt.* C*, and a substitution* σ *into two new branches:*

$$\mathsf{Split}(C,\sigma)\ \frac{p}{p \cup \{L\} \qquad p \cup \{\overline{L}\}} \quad \textit{if} \begin{cases} \textit{(i)} & \sigma \textit{ is a branch-unifier of } C \textit{ against } p, \textit{ and} \\ \textit{(ii)} & \textit{for some } L \in C\sigma, \textit{ neither } L \in^{\sim} p \textit{ nor} \\ & \overline{L} \in^{\sim} p, \textit{ and} \\ \textit{(iii)} & L = \textit{litsel}(C\sigma, p) \end{cases}$$

If for given p*,* C *and* σ *the conditions (i) and (ii) hold, it is said that the* Split *inference rule is* applicable to p, C and σ*, and the result as the set* $\{p \cup \{L\},\ p \cup \{\overline{L}\}\}$ *is denoted by* $\mathsf{Split}(p,C,\sigma)$*. The literal* L *in (iii) is called the literal* split on.

Note 3 (Purpose of Split*).* Assume that whenever Split is applied to a branch p, then p is consistent (that this can be achieved is argued for below). By Proposition 1 then, $[\![p]\!]$ is an interpretation. Now, the intuition behind Split is to find a clause C and a branch unifier σ of C against p, such that at least one ground instance of $C\sigma$ is *false* in $[\![p]\!]$. If no such σ exists, by Lemma 1 we can be sure that with $[\![p]\!]$ a model for the clause set has been found. Otherwise, Split is applicable to p, C and some branch unifier σ by the following line of reasoning: since p is ^{a}-open, hence open (cf. Note 2), σ must be a *falsifying* branch unifier. But then, condition (ii) must be satisfied. For, if (ii) would not be satisfied, for every literal $L \in C\sigma$ it would hold (a) $L \in^\sim p$ or (b) $\overline{L} \in^\sim p$. It is impossible that $L \in^\sim p$ for any $L \in C\sigma$ because then p would produce both L (because $L \in^\sim p$) and $\overline{L}$ (because σ is a branch unifier of C against p, and so p produces $\overline{L}$), and thus p would be inconsistent. Hence case (b) applies for every $L \in C\sigma$, and so σ would be a *closing* branch unifier of C against p, but we know that σ must be a *falsifying* branch unifier. Hence, with this contradiction condition (ii) holds. As a consequence, the literal selection function *litsel* is defined on $C\sigma$ and returns some arbitrarily (i.e. don't-care nondeterministically) selected literal from $C\sigma$ which is used for splitting p into the two new cases as stated.

Since the use of branch unifiers is insisted upon, Split is applicable in a very restricted way only. For instance, referring back to Example 5, Split is *not* applicable to the branch N_4 and the clause $P(a, b, c)$. Non-applicability of Split realizes a search-space reduction.

As said at the beginning of Note 3, it has to be made sure that p is consistent before Split is applied. This is the purpose of the following Commit inference rule (as the branch N_3 in Example 4 shows, consistency does not hold automatically).

Definition 10 (Commit). *The following inference rule* Commit *transforms a branch p, a literal L from p and a substitution σ into two new branches:*

$$\mathsf{Commit}(L,\sigma)\ \frac{p}{p \cup \{L\sigma\} \qquad p \cup \{\overline{L\sigma}\}} \quad \textit{if} \begin{cases} \textit{(i)} & L \in p, \textit{ and} \\ \textit{(ii)} & \sigma = \textit{unify}(\{L, \overline{K}\}), \textit{ for some } K \in^\sim p, \textit{ variable disjoint from } L, \textit{ and} \\ \textit{(iii)} & \textit{neither } L\sigma \in^\sim p \textit{ nor } \overline{L\sigma} \in^\sim p. \end{cases}$$

If for given p, L and σ the conditions (i) – (iii) hold, it is said that the Commit *inference rule is* applicable to p, L and σ*, and the result as the set $\{p \cup \{L\sigma\}, \{p \cup \{\overline{L\sigma}\}\}$ is denoted by* Commit(p, L, σ)*. The literal $L\sigma$ is called the literal* split on.

Note 4 (Purpose of Commit*).* Lemma 2 states (almost directly) that Commit is applicable to p, for some L and σ, whenever p is inconsistent. Thus, by the contrapositive direction, by repeated application of Commit one arrives at a consistent branch eventually.

The converse of Lemma 2 is *not* true: $\{P(x, a, u),\ \neg P(b, y, a),\ \neg P(b, a, u)\}$ *is* consistent but Commit is applicable (consider the first two literals). This shows that Commit is possibly applied more often than necessary. As an improvement, condition (iii) in Commit can be replaced by "L produces $L\sigma$ wrt. p and K produces $\overline{L\sigma}$ wrt. p."

Definition 11 (Derivation). *A* derivation $\mathcal{D}$ (from a clause set $\mathcal{S}$) *is a (possibly infinite) sequence of branch sets* $\mathcal{D} = (\mathcal{P}_0 = \{\{\neg x\}\}), \mathcal{P}_1, \ldots, \mathcal{P}_n, \ldots$ *, such that for* $i \geq 0$,

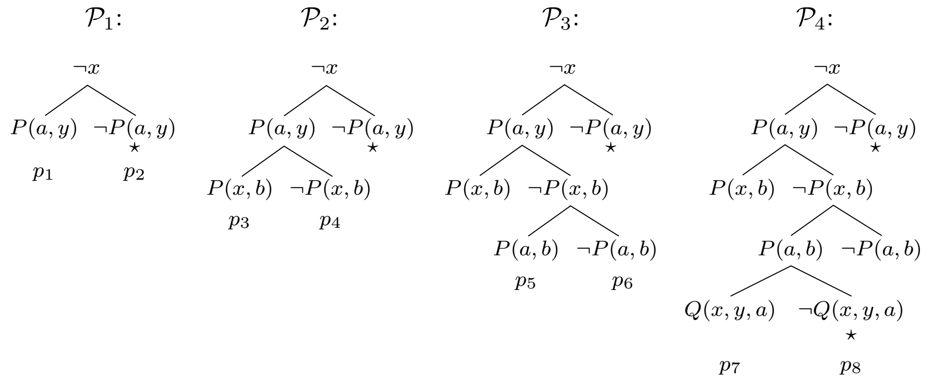
(i) $\mathcal{P}_{i+1} = (\mathcal{P}_i \setminus \{p_i\}) \cup \mathsf{Split}(p_i, C, \sigma)$ *for some clause* $C \in \mathcal{S}$ *and substitution* σ*, or*

(ii) $\mathcal{P}_{i+1} = (\mathcal{P}_i \setminus \{p_i\}) \cup \mathsf{Commit}(p_i, L, \sigma)$ *for some literal* L *and substitution* σ*,*

where in both cases $\mathcal{P}_i$ *is* ^{a}*-open wrt.* $\mathcal{S}$ *and* $p_i = sel(\mathcal{P}_i)$ *is the selected (hence* ^{a}*-open) branch in* $\mathcal{P}_i$*. A derivation is called a* refutation (of $\mathcal{S}$) *iff some* $\mathcal{P}_i$ *is* ^{a}*-closed (by* $\mathcal{S}$*). A derivation* of $\mathcal{P}_n$ *is a finite derivation that ends in* $\mathcal{P}_n$*.*

Both Split and Commit are applied to ^{a}-open branches only. Thus, if some $\mathcal{P}_i$ is ^{a}-closed, then $\mathcal{P}_i$ contains no single ^{a}-open branch and the derivation stops as a refutation.

Example 6 (Derivation). The figure below shows in tree notation a sample derivation from the clause set $\mathcal{S}$ consisting of the clauses $C_1 = P(a, y)$ and $C_2 = P(x, b) \vee \neg P(z, y) \vee Q(x, y, z)$.



The branch set $\mathcal{P}_0 = \{\{\neg x\}\}$ is not depicted; ^{a}-closed branches are marked with a "$\star$". $\mathcal{P}_1$ is obtained from $\mathcal{P}_0$ by a applying Split to $\{\{\neg x\}\}$ and C_1 (and the empty substitution); the branch p_2 is ^{a}-closed (even closed) due to C_1; $\mathcal{P}_2$ is obtained from $\mathcal{P}_1$ by applying Split to p_1 and C_2 (and some σ not made explicit). Neither Split nor Commit is applicable to p_3, so $[\![p_3]\!]$ is an interpretation (cf. Note 4) and $[\![p_3]\!] \models \mathcal{S}$ (cf. Note 3). To continue the example suppose that the branch p_4 is selected in $\mathcal{P}_2$. The Commit rule is applicable to p_4, which derives the branches p_5 and p_6. Applying Split to p_5 and C_2 yields p_7 and p_8 (as branch literals in the computation of the branch unifier use variants of $P(a, y)$ and $\neg P(x, b)$).The branch p_8 is ^{a}-closed by C_2, and to p_7, neither Commit nor Split is applicable (in particular, the instance $P(x, b) \vee \neg P(a, b) \vee Q(x, b, a)$ of C_2 which can be obtained by simultaneous unifying away the P-literals of C_2

against $P(a,b)$ and $\neg P(x',b)$ is produced by $Q(x,y,a)$). Thus, $[\![p_7]\!] \models \mathcal{S}$. The derivation can continue with $\mathcal{P}_4$ at the branch called p_6 in $\mathcal{P}_3$, which is not shown here.

Note 5 (Regularity). For both inference rules, when applied to a branch p, the literal L split on is "new" to p in the sense that neither L nor $\overline{L}$ is contained in p, not even as a variant. In Split, a respective condition in *litsel*, and in Commit condition (iii) is responsible for this. In other words, a stronger form of "regularity" than the identity-based one used in rigid variable calculi holds; also, it is impossible to derive a contradictory branch (cf. Def 4). Thus, from the completeness of FDPLL (Theorem 2) it follows immediately that FDPLL is a decision procedure for Bernays-Schönfinkel logic (no function symbols except constants), a class that cannot be decided easily by resolution or tableau methods.

This section is concluded with optimizations concerning Split, in particular the literal selection function *litsel* (Def. 8). The basis is the following lemma:

Lemma 3 (Open Branch Literal Selection). *Let p be an ^{a}-open branch wrt. $\mathcal{S}$, $C \in \mathcal{S}$ be a clause and σ be a branch unifier of C against p. Then, all of the following hold:*

(i) Split *is applicable to p, C and σ.*
(ii) The set $\mathcal{L} = \{L \in C\sigma \mid \overline{L^a} \notin p^a\}$ is non-empty.
(iii) $\overline{L} \notin^\sim p$, for every $L \in \mathcal{L}$.
(iv) If p is consistent, then $L \notin^\sim p$, for every $L \in C\sigma$.

It is clear from the definition of "derivation" that Split is applied to a branch p only if p is ^{a}-open. So, this precondition of the lemma is satisfied whenever Split is attempted. Now assume that C and σ exist as required in the lemma statement, and thus that items (i) – (iv) hold.

Item (i) summarizes what was sketched at the beginning of Note 3.

Suppose that Commit applications are preferred to Split applications (as is realized in the proof procedure in Section 7 below). Then, the branch p is consistent (cf. Note 4), and item (iv) shows that the condition (ii) in the Split inference rule may be equivalently replaced by "for some $L \in C\sigma$, $\overline{L} \notin^\sim p$" (since by item (iv) $L \notin^\sim p$ holds for all literals $L \in C\sigma$). Item (iv) is proven as follows: we are given that σ is a branch-unifier of C against p. This means in particular that each literal $\overline{L}$, where $L \in C\sigma$, is produced by some literal $K \in^\sim p$. Now, if $L \in^\sim p$ would hold, then p would produce L as well. By consistency, however, p cannot produce L.

Next, sensible literal selection by *litsel* is turned to. Concretely, *litsel*$(C\sigma, p)$ should return an element L from the (non-empty) set $\mathcal{L}$ defined in item (ii). Observe that splitting on this literal L is indeed possible, because by item (iii) the modified applicability condition of Split explained in the previous paragraph is satisfied for L. Now, item (ii) expresses that one need not select a literal from $C\sigma$ that is solved in the sense that it contributes to ^{a}-close p. For instance, if $p = \{\neg x,\ P(x,y)\}$, $C = \neg P(x,x) \vee \neg P(x,b)$ and $\sigma = \epsilon$, then $\mathcal{L} = \{\neg P(x,b)\}$.

It is more sensible to select $\neg P(x,b)$ for splitting, because the thus upcoming branch $\{\neg x,\ P(x,y),\ P(x,b)\}$ is ^{a}-closed, whereas splitting with $\neg P(x,x)$ leaves the upcoming branch $\{\neg x,\ P(x,y),\ P(x,x)\}$ ^{a}-open; with respect to closing branches, $P(x,y)$ and $P(x,x)$ are the same. Selecting literals from $\mathcal{L}$ yields shorter refutations.

5 Universal Literals

In this section, an optional inference rule for an improved treatment of unit clauses is discussed. The improvement allows to read, under certain circumstances, a literal in a branch, say, $P(x,y)$, as universally quantified, i.e. as $\forall x,y\ P(x,y)$. In the terminology used here, such a universally quantified literal $\forall L$ occuring in a branch p then produces all instances of L wrt. p – without exception –, and any extension of p also containing a literal K with $L > \overline{K}$ can be considered as closed, without any explicit refutation. Thus, many inferences can be saved by closing branches earlier.

One application of universal literals is to simulate resolution strategies like unit-resulting resolution and hyper-resolution for Horn clause sets (with subsumption pruning). To this end, the possibility to split, on the basis of variable-disjointness, a clause in one of its literals and the rest clause is exploited. For instance, suppose that the current clause set $\mathcal{S}$ contains the clause $P(x) \vee Q(y) \vee R(y)$. Then, $P(x)$ and $Q(y) \vee R(y)$ are variable disjoint and $\mathcal{S}$ is split into the two cases $\mathcal{S} \cup \{P(x)\}$ and $\mathcal{S} \cup \{Q(y) \vee R(y)\}$.

While the inference rules underlying these strategies could certainly be added to the FDPLL calculus, there is a reason not to do so: they *add* clauses to the current clause set. The control mechanism of a proof procedure would be more complicated then; a "current clause set" would have to be maintained, which would be enlarged by the inference rules and shrunk again when backtracking to an open case after the current case has been refuted.

Fortunately, the reasoning with split off literals can be integrated in FDPLL in a much smoother way by the following *atomic cut* inference rule[7]:

$$\mathsf{Cut}(L)\ \frac{p}{p \cup \{\forall L\} \qquad p \cup \{\overline{L\delta}\}} \quad \text{if} \begin{cases} \text{(i)} & \delta \text{ is a Skolemizing substitution for } L \text{ (see text), and} \\ \text{(ii)} & \text{no literal in } p \text{ stems from a previous } \mathsf{Cut} \text{ application of the form } \mathsf{Cut}(K)\text{, where } K \sim L \end{cases}$$

Here, $\forall L$ denotes the universal closure of the literal L, and $\forall L$ is called a *universal literal in* $p \cup \{\forall L\}$. The Skolemizing substitution δ in condition (i) is any substitution of the form $\delta = \{x_1/c_1, \ldots, x_n/c_n\}$, where the x_i's are all the variables occuring in L, and the c_i's are pairwise different and new constants (wrt. the constants occuring in p). Observe that Cut just branches on the tautology $\forall L \vee \neg\forall L$, followed by Skolemizing the right branch.

Condition (ii) is needed to control Cut. Any precaution to avoid unbound Cut applications along a branch in the same way would do.

[7] Well-known from the tableaux world.

The modifications to reason with universal literals are sketched next [8].

The starting point is a strictly stronger condition (than ^{a}-closed) to close branches: assume that a set of universal literals in p, $\mathit{Univ}(p) \subseteq p$, has already been determined. It is reasonable (and sound) to include in $\mathit{Univ}(p)$ the ground literals of p, too. Now let C be a clause. The branch p is said to be *closed** *by* C iff C can be partitioned as $C = C_1 \cup C_2$ such that (i) there is a simultaneous most general unifier σ of every $L \in C_1$ with some literal K s.t. $\overline{K} \in^\sim \mathit{Univ}(p)$ (use a new variant), and (ii) there is a substitution δ such that $\overline{L\delta} \in (p \setminus \mathit{Univ}(p))^a$ for every $L \in C_2\sigma$ (in particular, if $C_2\sigma = \{\}$ then $\delta = \epsilon$ exists trivially). In other words, condition (ii) is the same as saying that $p \setminus \mathit{Univ}(p)$ has to be ^{a}-closed by $C_2\sigma$.

Now, if (i) holds but (ii) does not hold for the considered clause $C = C_1 \cup C_2$ and branch p, then, as said, $p \setminus \mathit{Univ}(p)$ is not ^{a}-closed by $C_2\sigma$, and there is no *closing* branch unifier of $C_2\sigma$ against $p \setminus \mathit{Univ}(p)$. But it is still possible that there is a *falsifying* branch unifier σ' of $C_2\sigma$ against $p \setminus \mathit{Univ}(p)$. In order to describe how universal literals are derived, assume that this is the case. If furthermore there is a literal $L \in C_2\sigma\sigma'$ such that neither $L \in\ p$ nor $\overline{L} \in\ p$ (as is also required in the Split rule) and $\mathit{var}(L) \cap \mathit{var}(C_2\sigma\sigma' \setminus \{L\}) = \{\}$, then $\mathsf{Cut}(L)$ is applied to p (instead of Split, which is also applicable).

Carrying out $\mathsf{Cut}(L)$ just in case that $C_2\sigma\sigma = \{L\}$ (i.e. all but one literal in C are resolved away by universal or ground branch literals) suffices to simulate the abovementioned resolution strategies (observe that the right branch closes immediately then).

It should be emphasized that the model construction technique is not changed: to determine the interpretation $[\![p]\!]$ for a branch p containing universal literals, each universal literal $\forall L$ in p is taken as if it were not universally quantified, i.e. the quantifier is forgotten, and hence Definition 3 applies as without universal literals. However, possibly a non-Herbrand interpretation results due to Skolemization.

Fortunately, adding the described reasoning with universal literals requires only *minor* modifications. It is enough to attach boolean labels to the literals in branches to indicate their “universal” status, and to generalize “^{a}-closed” towards “closed*” as described above. In particular, Split need not be changed, and further improvements are expressed as a sharpened selection function *litsel*. For instance, it can be achieved that if Split is applicable to an open* branch p, C and σ, always a literal $L \in C\sigma$ exists such that neither L nor $\overline{L}$ is subsumed by a universal literal in p (Formally, Lemma 3 can be strictly strengthened). In other words, instances of universal literals (or their complements) need never be added by a Split. This mirrors “subsumption by a unit clause” in resolution.

6 Soundness and Completeness

Theorem 1 (Soundness of FDPLL). *Let $\mathcal{S}$ be a clause set and $\mathcal{D}$ be a refutation of $\mathcal{S}$. Then $\mathcal{S}$ is unsatisfiable.*

[8] The long version of the paper contains a full account.

Proof sketch: consider the last element $\mathcal{P}$ in $\mathcal{D}$, every branch of which is ^{a}-closed. Its ground instantiation $\mathcal{P}^a = \{p^a \mid p \in \mathcal{P}\}$ can be seen as a usual semantic tree $\mathcal{T}$ made up of splits with complementary ground literals (cf. [CL73]). Furthermore, the (finite) set of all those (ground) clauses $C\delta$ that are used for closing the branches in $\mathcal{P}^a$ show that every leaf in $\mathcal{T}$ is a failure node. Now apply the soundness result for usual semantic trees.

Next we turn to completeness. The FDPLL calculus proceeds by further modifying one single branch set, and never "backtracks" to a previously derived branch set. This notion of derivation indicates that to obtain a completeness result, fairness is to be defined as an exhaustive process (up to redundancy) of inference rule applications.

Before going into the details, a general note on this topic can be made: the notion of "derivation" in Def. 11 can be adapted to virtually every confluent rigid variable method. For these methods, the real challenge is to define fairness as just mentioned in such a way that it can be turned into an effective proof procedure. Only few attempts in this direction have been made [BEF99,Bec98]. Coming from the FDPLL side, it seems possible to bring the technique here to e.g. clausal tableaux calculi (by branching on clauses instead of complementary literals).

Definition 12 (Path). *Let $\mathcal{D}$ be a derivation that is not a refutation, written as in Definition 11. Let $\mathcal{P}^\infty = \bigcup_{j \geq 0} \mathcal{P}_j$, the set of all branches ever constructed in $\mathcal{D}$. A* path (of $\mathcal{D}$) *is a possibly infinite sequence $I = (p_0 = \{\neg x\}) \subset p_1 \subset \cdots \subset p_m \subset \cdots$ of branches in $\mathcal{P}^\infty$ such that for every $i \geq 0$*

(i) *$p_i = sel(\mathcal{P}_{s_i})$ is the selected (^{a}-open) branch in some branch set $\mathcal{P}_{s_i}$ in $\mathcal{D}$, and*
(ii) *$p_{i+1} = p_i \cup \{L_i\}$, for some literal L_i, and*
(iii) *if in $\mathcal{D}$ the* Commit *or the* Split *inference rule is applied to p_i, then I contains a successor element p_{i+1}.*

Finally, define the chain limit $\cup I = \bigcup_{i \geq 0} p_i$.

The limit $\cup I$ thus is an "infinitely long" branch, obtained by tracing some branch that is extended infinitely often and remains ^{a}-open. As an important property, $\cup I$ is ^{a}-open, because if $\cup I$ were ^{a}-closed, there is a clause C ^{a}-closing $\cup I$. Since clauses are finite and $\cup I$ is the limit of a chain, some finite $p_j \subset \cup I$ would be ^{a}-closed by C, contradicting item (i) in the definition. It is only noted here without proof that for every derivation $\mathcal{D}$ that is not a refutation a path of $\mathcal{D}$ exists.

Definition 13 (Finishedness, Fairness). *Let $\mathcal{D}$ be a derivation that is not a refutation and I be a path of $\mathcal{D}$, written as in Definition 12. The path I is* finished *iff for every $i \geq 0$ the following holds:*

(i) *if the* Split *inference rule is applicable to p_i and some clause $C \in \mathcal{S}$ and substitution σ, then $C\sigma$ is produced by p_j (Def. 2) for some $j \geq i$.*

(ii) if the Commit *inference rule is applicable to p_i and some literal $L \in p_i$ and substitution σ, then p_j is consistent wrt. $L\sigma$, for some $j \geq i$.*

$\mathcal{D}$ is fair *iff (i) $\mathcal{D}$ is a refutation or (ii) some path of $\mathcal{D}$ is finished.*

The purpose of condition (ii) in Definition 13 is to achieve that $\cup I$ is consistent wrt. any literal $L\sigma$ identified by a possible Commit application at time point i; similarly, the purpose of condition (i) in Definition 13 is to achieve that $\cup I$ produces the clause instance $C\sigma$ identified via a possible Split application at time point i.

That these purposes can be satisfied is a consequence of having a $\cup I$ as a *chain* limit (roughly, compactness wrt. the required properties holds then) and that Split and Commit applications achieve (i) and (ii), respectively, whenever violated (cf. Notes 4 and 3 again). However, *actually* carrying out the inferences is only the last resort: it suffices that their *effect* is achieved, namely (in case of Split, e.g.) that the clause $C\sigma$ is produced eventually. Indeed, a Split application might cause a former possible Split application to be impossible. For instance, when a clause $C_3 = \neg P(a,b) \vee R(a)$ is added to the clause set in Example 6, then Split is applicable to p_1 and C_3 (with $\sigma = \{x/R(a),\ y/b\}$), but Split is no longer applicable to C_3 and p_6 (due to $\neg P(a,b) \in p_6$).

These considerations shall serve as a proof sketch for the first main result:

Theorem 2 (Completeness of FDPLL). *Let $\mathcal{S}$ be a clause set and $\mathcal{D}$ be a fair derivation from $\mathcal{S}$. If $\mathcal{D}$ is not a refutation then $\mathcal{S}$ is satisfiable. More specifically, for every finished path I of $\mathcal{D}$, $[\![\cup I]\!]$ is an interpretation and $[\![\cup I]\!] \models \mathcal{S}$.*

Notice that in the contrapositive direction, the theorem is just a refutational completeness result.

A final remark: the calculus is asymmetric wrt. the rôle of branches. Consider e.g. a branch $p = \{\neg x,\ P(x),\ \neg P(a)\}$. To determine closure, p^a is used, and to extract a model $[\![p]\!]$ is used; inconsistency of p^a (as in the example) is not relevant for model extraction – $P(a)$ is simply *false* in $[\![p]\!]$[9]. For Theorem 2 to hold there is no need to go beyond the Herbrand interpretations as stated in Section 2.

7 Proof Procedure

So far, fair derivations are purely abstract mathematical objects, and it still has to be demonstrated that an effective fair strategy exists. In essence, to guarantee fairness, a *maximal term depth bound* is used, which all literals to be split on in Split applications have to obey; starting with a small natural number, the value of this bound is increased only after having exhausted Split within the current value. This inner loop exhaustion always terminates, essentially because variants of literals already present in a branch are never added. Fortunately, Commit need not be subject to such a term depth bound check — it *finitely* exhausts on any (finite) literal set.

[9] The calculus can be slightly improved by considering p as closed if p^a is contradictory (as in the example), although p is ^{a}-open.

These are the essential ingredients of the following concrete proof procedure.

```
funct $FDPLL(\mathcal{S})$ $\equiv$
  var $\mathcal{I}$;                                   % for the model representation
  funct $Satisfiable(p, bound)$ $\equiv$
    % $p$: the current branch. $bound$: non-negative integer, the maximal term
    % depth admissible in literals for splitting
    if ${}^{a}\text{-}Closed(p, \mathcal{S})$
      then return false
       else % Try a Commit. First, collect all candidate literals in $\mathcal{L}$:
           var $\mathcal{L} := \{L\sigma \mid L \in p,\ \exists K \in p : \sigma = unify(\{L, \overline{new(K)}\}) \neq$
                     undefined $\wedge\, L\sigma \notin^{\sim} p \wedge \overline{L\sigma} \notin^{\sim} p\}$;
           if $\mathcal{L} \neq \{\}$
             then % Commit is applicable
                  var $L_c := \eta\, L \in \mathcal{L} : true$;            % Select any candidate
                  if $Satisfiable(p \cup \{L_c\}, bound)$         % Left branch extension
                    then return true
                     else return $Satisfiable(p \cup \{\overline{L_c}\}, bound)$  % Right branch
                  fi
              else % Commit not applicable - try Split. Collect all candidates:
                  var $\mathcal{L} := \{L \mid \exists C \in \mathcal{S},\ \sigma \in BranchUnify(C, p) :$
                              $L = litsel(C\sigma, p)\}$
                  if $\mathcal{L} = \{\}$
                    then $\mathcal{I} := p$;  % Split is not applicable – got a model in $p$.
                         return true
                     else % $\mathcal{L} \neq \{\}$, so Split is applicable
                         var $L_s := \eta\, L' \in \mathcal{L} : \|L'\| \leq bound$;
                                  % Select any candidate within depth bound.
                                                % However, it might not exist:
                         if $L_s \neq$ undefined
                           then       % Candidate within depth bound exists
                                if $Satisfiable(p \cup \{L_s\}, bound)$
                                  then return true
                                   else return $Satisfiable(p \cup \{\overline{L_s}\}, bound)$
                                fi
                            else % Hit depth bound – try with higher one:
                                return $Satisfiable(p, bound + 1)$
                         fi fi fi fi.

  % Body of $FDPLL$:
  if $Satisfiable(\{\neg x\}, 0)$
    then return $\mathcal{I}$
     else return false
  fi.
```

Some functions remain unspecified: a call of ^{a}*-Closed(p,S)* is supposed to return **true** iff p is ^{a}-closed by $\mathcal{S}$ (cf. Def 5); $new(L)$ is supposed to return a "fresh" variant of L, containing no variables used so far. A call of $BranchUnify(C, p)$ is supposed to return a possibly empty, finite and complete set of branch unifiers of

C against p[10]. Finally, $\|L\|$ denotes the depth of $|L|$ as a tree. All these functions can be effectively implemented.

Some comments on the structure of the procedure: *FDPLL* is a wrapper around *Satisfiable*. The p parameter of *Satisfiable* is the currently selected branch in an implicitly constructed derivation. The branch selection realized in *Satisfiable* is implicitly a left-to-right depth-first strategy.

The counterpart of ^{a}-closed branches in the definition of "derivation" is a return value of **false** in *Satisfiable*; thus, closed branches are not kept in memory, and so *Satisfiable* realizes a space efficient one-branch-at-a-time approach.

If every incarnation of *Satisfiable* returns *false*, then *FDPLL* returns *false* as well, indicating unsatisfiability of $\mathcal{S}$. *FDPLL* returns a model $\mathcal{I}$ only if some incarnation of *Satisfiable* returns **true** on line 25, because a return value of **true** in *Satisfiable* is immediately propagated to its caller. This happens only if neither Commit nor Split are applicable to p. That $\mathcal{I} \models \mathcal{S}$ holds then follows directly from Notes 3 and 4.

Now some more specific comments on *Satisfiable*: the set comprehension formula on line 11 is just the applicability condition of Commit. An η-expression $\eta\, L \in \mathcal{L} :\ \varphi(L)$ (as on lines 15 and 27) returns any $L \in \mathcal{L}$ such that $\varphi(L)$ holds, if such an L exists, and returns **undefined** else. The set $\mathcal{L}$ on line 21 is assigned a finite set of literals such that whenever Split is applicable to some $C \in \mathcal{S}$ and σ then $\mathit{litsel}(C\sigma, p) \in^{\sim} \mathcal{L}$. When reaching line 21, p is known to be ^{a}-open (because Line 9 was not reached) and consistent (Note 3 account for this fact). Hence, all of Lemma 3 is applicable, and the improvements for *litsel* discussed there can be taken advantage of.

The *bound* parameter of *Satisfiable* realizes a depth bound, which all literals to be split on in Split applications have to obey. The fairness of the procedure is guaranteed by only increasing *bound* (on line 37) after exhaustion of Split on the currently given value of *bound*. However, Commit need not be subject to such a depth bound check — it *finitely* exhausts on any finite literal set.

In order to save space and concentrate on the most essential issues to be contributed here, the *FDPLL* procedure just described does not use universal literals (cf. Section 5). The long version of the paper contains a procedure with universal literals in full detail, which also realizes the restrictions of Commit and Split mentioned at the end of Section 5. A further built-in improvement is to remove a split rule application from a derivation if in the left derived branch $p \cup \{L\}$ the literal L is not needed in the refutation of $p \cup \{L\}$. Hence the right branch $p \cup \{\overline{L}\}$ need not be considered. This well-known improvement[11] realizes (but is more powerful than) the *purity* rule of propositional DPLL. The implementation mentioned in Section 1 refers to the version with these improvements.

The procedure is correct: soundness and completeness holds in a similar way as stated for the calculus. Beyond this, the procedure constructs a fair derivation. For space reasons, the proof is omitted here.

[10] I.e. it contains modulo renaming every branch unifier of C against p.

[11] Also known as "dependency directed backtracking", "level cut", "condensing" etc.

8 Conclusions

A directly lifted, confluent and strongly complete version of the DPLL procedure has been presented. As the theoretical concepts (in particular the model representation) are new, emphasis was put on these rather than on experimental studies.

Related work. Already in [CL73] a lifted DPLL calculus can be found. It uses the device of "pseudosemantic trees", which, like FDPLL, realizes splits at the non-ground level. Nethertheless, the pseudosemantic tree method is very different: in sharp contrast to FDPLL, a variable is treated rigidly there, i.e. as a placeholder for a (one) not-yet-known term. As a consequence, like in all rigid variable methods, only a very weak regularity condition can be used (cf. Note 5 below). Furthermore, only a weak completeness result is known, which translates into a heavily backtracking oriented proof procedure only.

When FDPLL reports "satisfiable", a model representation has been computed without further processing. This does neither hold for the mentioned method in [CL73], nor for the resolution based methods for model computation in [FL93,FL96]. Typically, the latter attempt to compute a model by enumerating all true *ground* literals, thereby interleaving this enumeration with calls to the resolution procedure again in order to determine the "next" ground literal.

The probably most advanced first-order tableau system tailored for model computation is RAMCET [Pel99], which is a successor of Peltier's and his coworkers resolution calculus and previous tableaux calculi (see e.g. [CP95]). As a drawback, RAMCET needs additional inference rules for model computation. In particular, the "model explosion" rule seems problematic, as it branches out wrt. the whole signature of the formula set under consideration. FDPLL does not need such a rule. Furthermore, unlike for FDPLL, strong completeness for RAMCET is still unsolved (while proof confluence is trivial), in the sense that no effective fair strategy is known except a trivial one, which needs exponential space — a widespread problem of tableaux calculi.

In the literature several methods are described that are related to FDPLL in the sense that variables are treated in a similar way (cf. "description of FDPLL" in Section 1). FDPLL was influenced and is intended as a successor of the hyper tableau calculus [Bau98] (which in turn is a successor of the calculus in [BFN96], a calculus in the tradition of Satchmo[MB88]). Among other things, FDPLL improves on this calculus by not needing to store instances of clauses as the derivation proceeds – only the "current interpretation" needs to be kept. Beyond this, FDPLL is conceptually different: like any tableaux calculus, hyper tableau branches on (sub-)formulas, whereas FDPLL branches in a binary way on complementary literals, i.e. uses "cut" as the single inference rule. The latter is more general and "builds in" standard improvements like factorization automatically.

What was said about hyper tableau applies equally to the disconnection method [Bil96]. Also, no model computation result was given for this calculus.

Also related are Plaisted's hyper-linking calculi: the semantic hyper-linking calculus (SHL) [LP92] proceeds by searching in a guided way for (not necessarily ground) instances of input clauses, which are tested for unsatisfiability by a propositional DPLL procedure. Much of what was said about hyper tableau above applies to this calculus as well. In particular, unlike SHL, FDPLL does not interleave two processes "clause instance generation" and "propositional DPLL". The former process occurs in FDPLL only "locally" within the splitting rule, and derived clause instances need not be kept. It seems worth to investigate combinations of SHL and FDPLL, e.g. by replacing propositional DPLL in SHL by FDPLL, or picking up the idea of *guided* instance generation in SHL to improve FDPLL. However, this is future work.

Quite different is the ordered semantic hyper linking (OSHL) calculus [PZ97]. OSHL has many interesting features, for instance "semantical guidance". In the intersection with FDPLL, it can be described as a calculus that applies unit-resulting resolution as long as possible, and then splits with a *ground* literal in order to begin the next round. The main motivation for FDPLL however was to get rid of such ground splits. Therefore, it seems realistic to possibly improve on OSHL by bringing in the non-ground splitting technique of FDPLL.

Future work. A part of my FDPLL research plan is an *efficient* implementation. So far, only a slow and *prototypical* implementation in Prolog exists (available from my home page). Although it lacks such crucial features like term indexing, the performance seems promising, in particular for satisfiable or non-Horn problems without equality (there is no built-in equality treatment yet). In the respective subdivisions SAT and NNE of the CASC-16 system competition 1999, FDPLL scored rank 4 of 6 and rank 4 of 10, respectively. From the TPTP library [SSY94], FDPLL can also solve some difficult unsatisfiable problems quite quickly (e.g. `ANA002-4`, the intermediate value theorem, in 3 seconds). The overall success rate is about 40% (Otter: 52%) for a time limit of 10 minutes.

Other sources for future work are combinations of the techniques described here with hyper-linking calculi, equality treatment and improved termination behavior to name a few. On the theoretical level, the relationship between the model representation capabilities in FDPLL and the *atomic model representations* [GP98] used in the resolution and tableau world should be clarified.

Acknowledgments. I am grateful to the members of our group for discussions about FDPLL. David Plaisted, Mark Stickel and Ryuzo Hasegawa read the paper in depth and helped to clarify concepts. Three referees gave valuable comments and suggestions, in particular concerning improvements of the Commit rule.

References

[Bau98] Peter Baumgartner. Hyper Tableaux — The Next Generation. In Harry de Swaart, editor, *Tableaux-98*, LNAI 1397. Springer, 1998.

[Bec98] Bernhard Beckert. *Integration und Uniformierung von Methoden des tableaubasierten Theorembeweisens.* Dissertation, University of Karlsruhe, 1998.

[BBOS98] Wolfgang Bibel, Stefan Brüning, Jens Otten, and Thorsten Schaub. Volume I, Chapter 5: Compressions and Extensions. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction. A Basis for Applications*, pp. 133–179. Kluwer Academic Publishers, 1998.

[BEF99] Peter Baumgartner, Norbert Eisinger, and Ulrich Furbach. A confluent connection calculus. In Harald Ganzinger, editor, *CADE-16*, LNAI 1632, pp. 329–343, Trento, Italy, 1999. Springer.

[BFN96] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *JELIA 96*, LNAI 1126. Springer, 1996.

[Bil96] Jean-Paul Billon. The Disconnection Method. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Tableaux-96*, LNAI 1071, pp. 110–126. Springer, 1996.

[CA96] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in random 3sat. *Artificial Intelligence*, 81, 1996.

[CL73] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

[CP95] Ricardo Caferra and Nicolas Peltier. Decision Procedures using Model Building techniques. In *Computer Science Logic (CSL '95)*, 1995.

[D63] Martin Davis. Eliminating the irrelevant from mechanical proofs. In *Proceedings of Symposia in Applied Mathematics – Experimental Arithmetic, High Speed Computing and Mathematics*, volume XV, pp. 15–30. American Mathematical Society, 1963.

[DLL62] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7), 1962.

[DP60] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7:201–215, 1960.

[Ede85] Elmar Eder. Properties of Substitutions and Unifications. *Journal of Symbolic Computation*, 1(1), March 1985.

[FL93] Christian Fermüller and Alexander Leitsch. Model building by resolution. In E. Börger, G. Jäger, H. Kleine-Büning, S. Martini, and M.M. Richter, editors, *Computer Science Logic – CSL'92*, LNCS 702, pp. 134–148. Springer, 1993.

[FL96] Christian Fermüller and Alexander Leitsch. Hyperresolution and automated model building. *Journal of Logic and Computation*, 6(2):173–230, 1996.

[GP98] Georg Gottlob and Reinhard Pichler. Working with Arms: Complexity Results on Atomic Representations of Herbrand Models. In *Proceedings of the 14th Symposium on Logic in Computer Science*, IEEE, 1998.

[LP92] S.-J. Lee and D. Plaisted. Eliminating Duplicates with the Hyper-Linking Strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.

[MB88] Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In Ewing Lusk and Ross Overbeek, editors, *CADE 9*, LNCS 310, pp. 415–434. Springer, 1988.

[Pel99] N. Peltier. Pruning the search space and extracting more models in tableaux. *Logic Journal of the IGPL*, 7(2):217–251, 1999.

[PZ97] David A. Plaisted and Yunshan Zhu. Ordered Semantic Hyper Linking. In *Proceedings AAAI-97*, 1997.

[SSY94] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In Alan Bundy, editor, *CADE 12*, LNAI 814, pp. 192–206, Nancy, France, June 1994. Springer.

[Zha97] Hantao Zhang. SATO: An Efficient Propositional Theorem Prover. In W. McCune, editor, *CADE 14*, LNAI 1249, pp. 272–275, Springer.