Reductions for Non-Clausal Theorem Proving

G. Aguilera, I.P. de Guzmán, M. Ojeda-Aciego, A.Valverde

Dept. Matemática Aplicada. Universidad de Málaga.

Aptdo. de Correos 4114. E-29080 Málaga, Spain

Abstract

This paper presents the TAS methodology [1] as a new framework for generating non-clausal Automated Theorem Provers. We present a complete description of the ATP for Classical Propositional Logic, named TAS-D, but the ideas, which make use of implicants and implicates can be extended in a natural manner to first-order logic, and non-classical logics. The method is based on the application of a number of reduction strategies on subformulas, in a rewrite-system style, in order to reduce the complexity of the formula as much as possible before branching. Specifically, we introduce the concept of complete reduction, and extensions of the pure literal rule and of the collapsibility theorems; these strategies allow the size of the search space. In addition, it is a syntactical countermodel construction.

As an example of the power of TAS-D we study a class of formulas which has linear proofs (in the number of branchings) when either resolution or dissolution with factoring is applied. When applying our method to these formulas we get proofs without branching.

Key words: Non-clausal theorem proving, prime implicates/implicants, SAT problem.

[1] TAS stands for *Transformaciones de Árboles Sintácticos*, Spanish translation of Syntactic Trees Transformations.

⋆ Partially supported by CICYT project number TIC97-0579-C02-02.

1 Introduction

Much research in automated theorem proving has been focused on developing satisfiability testers for sets of clauses. However, experience has pointed out a number of disadvantages of this approach: it is not natural to specify a real-world problem in clause form, the translation into clause form is not easy to handle and, although there a number of efficient translation methods, models usually are not preserved under the translation. In addition, clausal methods are not easy to extend to non-classical logics, partly because no standard clause form can be defined in this wider setting.

There are different obstacles that have to be overcome when clausal normal forms are to be used in a non-classical context: normal forms can be exponentially long wrt the length of the input (although this is not a problem in classical logic where linear transformations exist [**?**,**?**]); the normalized input bears no resemblance to the original formula; non-classical logics can be found that fail to have *internal* normal forms.

Non-clausal theorem proving research has been mainly focused on either tableaux methods or matrix-based methods; also some ideas based on the data structure of BDDs have been used in this context. Recently, path dissolution [**?**] has been introduced as a generalisation of analytic tableaux, allowing tableaux deductions to be substantially speeded up.

The central point for efficiency of any satisfiability tester is the control over the branching, and our approach is focussed on the reduction of the formula to be analysed as much as possible before actually branching. Specifically, we introduce the concept of complete reduction, and extensions of the pure literal rule and of the collapsibility theorems. On the other hand, another interesting point in the design of ATPs is the capability of building models provided the input formula is satisfiable.

A non-clausal algorithm for satisfiability testing in the classical propositional calculus, named TAS-D, is described. The input to the algorithm need not be in conjunctive normal form or any normal form. The output is either "Unsatisfiable", or "Satisfiable" and in the latter case also a model of the formula is given.

To determine satisfiability for a given formula, firstly we reduce the size of the formula by applying satisfiability-preserving transformations, then choose a variable to branch and recursively repeat the process on each generated task.

This feature allows us:

- to obtain useful information from the original structure of the formula.
- to make clearer proofs.
- to extend the method to non-classical logics which do not have a widely accepted normal form.

Although our intention in this paper is to introduce the required metatheory, TAS-D is currently being tested and we are obtaining very promising results. In our opinion, the results of these tests indicate the TAS framework to be a reliable approach to automated theorem proving. The ideas of TAS are widely applicable because they apply to different types of logics; flexible because they provide a uniform way to prove soundness and completeness; and, in addition, easily adaptable because switching to a different logic is possible without having to redesign the whole prover. In fact, it has been already extended to Classical First Order Logic [**?**], Temporal Logic [**?**] and Multiple-Valued Logic [**?**,**?**].

The structure of the paper is as follows:

(1) Firstly, the necessary definitions and theorems which support the reduction strategy are introduced in Section 2.

(2) Later, the algorithm TAS-D is described in Section 3.

(3) Finally, a comparative example is included in Section 4, which shows a class of formulas which has linear proofs (in the number of branchings) when either resolution or dissolution with factoring is applied [**?**,**?**]. When applying TAS-D to these formulas we get proofs without branching.

1.1 Overview of TAS-D

TAS-D is a satisfiability tester for classical propositional logic; therefore it can be used as a refutational ATP method and, like tableaux methods, it is a syntactical model construction.

Reduction strategies are the main novelty of our method with respect to other non-clausal ATPs; like these methods, TAS-D is based on the disjunctive normal form. Its power is based not only on the intrinsically parallel design of

the involved transformations, but also on the fact that these transformations are not just applied one after the other, but guided by some syntax-directed criteria, described in Sections 2.2 and 2.4, whose complexity is polynomial. These criteria allow us:

(1) To detect subformulas which are either valid, or unsatisfiable, or equivalent to literals.

(2) To detect literals ℓ such that it is possible to obtain a equisatisfiable formula in which ℓ appears at most once. Therefore, we can decrease the size of the problem as much as possible before branching.

By checking these criteria we give ourselves the opportunity to reduce the size of the problem while creating just *one* subproblem; in addition, such reductions do not contribute to exponential growth. However, the most important feature of the reductions is that they enable the exponential growth rate to be limited.

As an ATP, TAS-D is sound and complete and, furthermore, as a model building method, it generates countermodels in a natural manner.

1.2 Preliminary Concepts and Definitions

Throughout the rest of the paper, we will work with a classical propositional language over a denumerable set of propositional variables, $\mathcal{V}$, and connectives $\{\neg, \wedge, \vee, \rightarrow\}$; the semantics for this language is the standard one:

- An *assignment* I is an application from the set of propositional variables $\mathcal{V}$ to $\{0, 1\}$; the domain of an assignment is uniquely extended to the whole language with the usual definition of the classical connectives.

- A formula A is said to be *satisfiable* if there exists an assignment I such that $I(A) = 1$; in this case I is said to be a *model* for A.

- Two formulas A and B are said to be *equisatisfiable* if *A is satisfiable iff B is satisfiable.*

- Two formulas A and B are said to be *equivalent*, denoted $A \equiv B$, if $I(A) = I(B)$ for all assignment I.

- A formula A is a logical consequence of a set of formulas Ω, denoted $\Omega \models A$,

if $I(A) = 1$ for all model I of Ω.

- We use the symbols $\top$ and $\bot$ to denote truth and falsity.

We will also use the usual notions of literals, clauses, cubes, implicants, implicates and negation normal form (nnf):

- A literal is either a propositional variable or the negation of a propositional variable; $\mathcal{V}^{\pm}$ denotes the set of literals.

- If ℓ is a literal then $\overline{\ell}$ is its opposite literal, that is: $\overline{p} = \neg p$ and $\overline{\neg p} = p$ for all $p \in \mathcal{V}$.

- A *clause* is a disjunction of literals $\ell_1 \vee \ldots \vee \ell_n$. A clause is said to be *restricted* if it has no pairs of opposite literals and has no repeated literals.

- A *cube* is a conjunction of literals $\ell_1 \wedge \ldots \wedge \ell_m$. A cube is said to be *restricted* if it has no pairs of opposite literals and has no repeated literals.

- A formula A is said to be in *negation normal form (nnf)*, if it has no occurrences of the connective $\rightarrow$ and the negations are only in the literals.

- $S \sqsubseteq A$ denotes that S is a subformula of A, and $S \sqsubset A$ denotes that S is a proper subformula of A.

- A literal ℓ is an *(unitary) implicant* of a formula A if $\ell \models A$.

- A literal ℓ is an *(unitary) implicate* of a formula A if $A \models \ell$.

The transformation of a formula into nnf is linear (by repeated application of De Morgan rules, the double negation rule and the equivalence $A \rightarrow B \equiv \neg A \vee B$), so in the following we will only consider formulas in nnf.

We will use the standard notion of tree and address of a node in a tree; ε will denote the address of the root node. In addition, we will identify the formulas with its syntactic tree. An address, η, in a formula A (i.e. in its syntactic tree) means, when no confusion arises, a subformula of A: the scope of the node with address η in A. Similarly, when we say a subformula B of A we mean an occurrence of B in A; if $B \sqsubseteq A$, then η_B denotes the address of the node corresponding to B in A; in particular, $\eta_A = \varepsilon$.

If $\alpha = \{\ell_1, \ell_2, \ldots, \ell_n\}$ is a set of literals, then $\overline{\alpha} = \{\overline{\ell_1}, \overline{\ell_2}, \ldots, \overline{\ell_n}\}$.

If $\alpha = \{\ell_1, \ell_2, \ldots, \ell_n\}$ is a set of literals in A and $* \in \{\top, \bot\}$, then the expression $A[\alpha/*]$ denotes the formula obtained after substituting in A, for all $\ell \in \alpha$, every occurrence of ℓ by $*$, and $\overline{\ell}$ by $\overline{*}$.

If A, B and C are formulas and $B \sqsubseteq A$, then $A[B/C]$ denotes the result of substituting in A any occurrence of B by C. If $\{\ell_1, \ldots, \ell_n\}$ is a set of literals in A and C_i are formulas, then the expression $A[\ell_1/C_1, \ldots, \ell_n/C_n]$ denotes the formula obtained after substituting in A, for all i, every occurrence of ℓ_i by C_i.

If η is an address in A and C is a nnf, then the expression $A[\eta/C]$ is the formula obtained after substituting in A the subtree rooted in η by C.

2 Adding Information to the Tree: Δ-lists and $\widehat{\Delta}$-sets

The idea underlying the reduction strategy is the use of information given by partial assignments (extensively used in Quine's method [**?**]) just for unitary assignments but, as we will show, in a powerful manner.

We associate to each nnf A two lists[2] of literals denoted $\Delta_0(A)$ and $\Delta_1(A)$ (the associated Δ-lists of A) and two sets, denoted $\widehat{\Delta_0}(A)$ and $\widehat{\Delta_1}(A)$, whose elements are obtained out of the associated Δ-lists of the subformulas of A.

The Δ-lists and the $\widehat{\Delta}$-sets are the key tools of our method to reduce the size of the formula being analysed for satisfiability.

2.1 The Δ-lists

In a nutshell, $\Delta_0(A)$ and $\Delta_1(A)$ are, respectively, lists of prime implicates and implicants (with length at most one) of A. The purpose of these lists is two-fold: firstly, to transform the formula A into an equivalent and smaller-sized one (Section 2.2), and secondly, by means of the $\widehat{\Delta}$-sets (Sections 2.3 and 2.4), to get an equisatisfiable and smaller-sized one. Their formal definition is the following:

Definition 1 *Given an nnf A, $\Delta_0(A)$ and $\Delta_1(A)$ are recursively defined as*

[2] We use lists in lexicographic order just to facilitate the presentation of the examples. The reader can interpret them as sets.

follows:

$$\begin{array}{ll}
\Delta_0(\bot) = \bot & \Delta_1(\bot) = \texttt{nil} \\
\Delta_0(\top) = \texttt{nil} & \Delta_1(\top) = \top \\
\Delta_0(\ell) = \ell & \Delta_1(\ell) = \ell \\
\Delta_0\left(\bigwedge_{i=1}^n A_i\right) = \bigcup_{i=1}^n \Delta_0(A_i) & \Delta_1\left(\bigwedge_{i=1}^n A_i\right) = \bigcap_{i=1}^n \Delta_1(A_i) \\
\Delta_0\left(\bigvee_{i=1}^n A_i\right) = \bigcap_{i=1}^n \Delta_0(A_i) & \Delta_1\left(\bigvee_{i=1}^n A_i\right) = \bigcup_{i=1}^n \Delta_1(A_i)
\end{array}$$

In addition, elements in a Δ_0-list are considered to be conjunctively connected (as implicates are disjunctions a collection of them is a conjunction) and elements in a Δ_0-list are considered to be disjunctively connected. The disjunctive/conjunctive nature of the Δ-lists allows some simplifications to be applied; namely, if $\{\ell, \overline{\ell}\} \subset \Delta_0(A)$, then $\Delta_0(A)$ is simplified to $\bot$, and if $\{\ell, \overline{\ell}\} \subset \Delta_1(A)$, then $\Delta_1(A)$ is simplified to $\top$.

The intuition behind the definition is easy to explain, since in $\Delta_0\left(\bigwedge_{i=1}^n A_i\right)$ we intend to calculate implicates *(for it is Δ_0), and since the union of the implicates of each conjunct is a set of implicates of the conjunction, then we use $\bigcup$, and so on.*

Example 2 *(1)* $\Delta_0(\overline{q} \vee r \vee p) = \texttt{nil}$
$\Delta_1(\overline{q} \vee r \vee p) = p\overline{q}r$

(2) $\Delta_0(p \wedge \overline{q} \wedge \overline{p} \wedge r) = \bot$
$\Delta_1(p \wedge \overline{q} \wedge \overline{p} \wedge r) = \texttt{nil}$

(3) $\Delta_0((s \wedge p) \vee (s \wedge \overline{t})) = ps \cap s\overline{t} = s$
$\Delta_1((s \wedge p) \vee (s \wedge \overline{t})) = \texttt{nil} \cup \texttt{nil} = \texttt{nil}$

(4) $\Delta_0((s \wedge p) \vee s) = ps \cap s = s$
$\Delta_1((s \wedge p) \vee s) = \texttt{nil} \cup s = s$

Note that for cubes and clauses, the lists Δ_0 and Δ_1, respectively, give their representation as sets of literals.

2.2 Information in the Δ-lists

In this section we study the information contained in the Δ-lists of a given formula. Our first theorem states that elements of $\Delta_0(A)$ are implicates of A,

and elements of $\Delta_1(A)$ are implicants of A, and follows easily by structural induction from the definition of Δ_b-lists.

Theorem 3 *Let A be a nnf and ℓ be a literal in A then:*

(1) If $\ell \in \Delta_0(A)$, then $A \models \ell$ and, equivalently, $A \equiv \ell \wedge A$.

(2) If $\ell \in \Delta_1(A)$, then $\ell \models A$ and, equivalently, $A \equiv \ell \vee A$.

As an immediate corollary of the previous theorem we have the following result on the structure of the Δ-lists:

Corollary 4 *For every nnf A we have one and only one of the following possibilities:*

- *There is $b \in \{0, 1\}$ such that $\Delta_b(A) = \texttt{nil}$,*
- *$\Delta_1(A) = \Delta_0(A) = \ell$, and then $A \equiv \ell$.*

The following corollary states a condition on the Δ_1-lists which directly implies the satisfiability of a formula.

Corollary 5 *If $\Delta_1(A) \neq \texttt{nil}$, then A is satisfiable, and if $\ell \in \Delta_1(A)$, then any assignment I such that $I(\ell) = 1$ is a model for A.*

On the other hand, the following result states conditions on the Δ-lists assuring the validity or unsatisfiability of a formula.

Corollary 6 *Let A be a nnf, then*

(1) If $A = \bigwedge_{i=1}^{n} A_i$ in which a conjunct A_{i_0} is a clause such that $\overline{\Delta_1(A_{i_0})} \subseteq \Delta_0(A)$, then $A \equiv \bot$.

(2) If $A = \bigvee_{i=1}^{n} A_i$ in which a disjunct A_{i_0} is a cube such that $\overline{\Delta_0(A_{i_0})} \subseteq \Delta_1(A)$, then $A \equiv \top$.

PROOF.

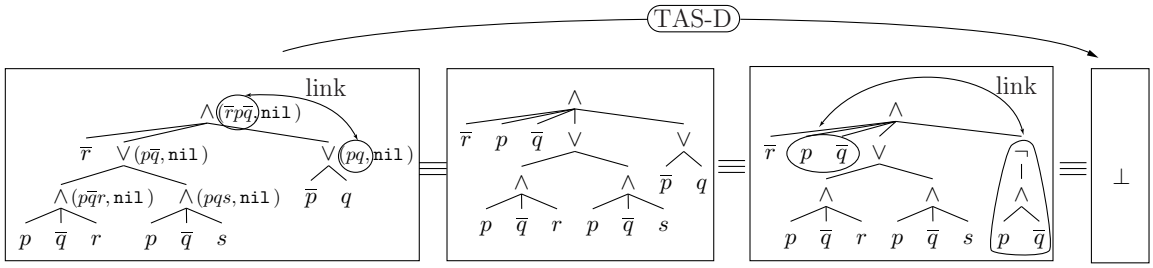
(1) Let $A = \bigwedge_{i=1}^{n} A_i$, using the results of Theorem 3, if $A_{i_0} = \bigvee_{j=1}^{m} \ell_j$ with,

$\overline{\ell_1} \ldots \overline{\ell_m} \subseteq \Delta_0(A)$, then $A \equiv \overline{\ell_j} \wedge A$ for all $j \in \{1, \ldots, m\}$. Therefore:

$$A \equiv A_{i_0} \wedge A = \bigvee_{j=1}^{m} \ell_j \wedge A \equiv \bigvee_{j=1}^{m} \ell_j \wedge (\bigwedge_{j=1}^{m} \overline{\ell_j} \wedge A) \equiv \bot$$

(2) It is similar to the previous one.

The example below shows the transformation stated in the previous corollary; the analysis of the Δ-lists detects a non-trivial link (in the sense of [**?**], i.e. a pair of opposite literals) and conclude the unsatisfiability of the formula.



Definition 7 *If A is a nnf, to* Δ-label *A means to associate to each node η in A the ordered pair $(\Delta_0(\eta), \Delta_1(\eta))$.*

Let us name those formulas whose Δ-lists allow to determine either its validity or its (un)satisfiability.

Definition 8 *A nnf A is said to be*

- finalizable *if one of the following conditions holds:*

(1) $\Delta_1(A) \neq \mathtt{nil}$.

(2) $\Delta_0(A) = \bot$.

This definition will be applicable to the current formula when it is detected to be (un)satisfiable. The following three definitions are referred to subformulas of the current formula which are detected to be either valid, or unsatisfiable, or equivalent to a literal.

- Δ_1-conclusive *if one of the following conditions holds:*

(1) $\Delta_1(A) = \top$.

(2) $A = \bigvee_{i=1}^{n} A_i$ *and a disjunct* A_{i_0} *is a cube such that* $\overline{\Delta_0(A_{i_0})} \subseteq \Delta_1(A)$.

- Δ_0-conclusive *if one of the following conditions holds:*

(1) $\Delta_0(A) = \bot$.

(2) $A = \bigwedge_{i=1}^{n} A_i$ *and a conjunct* A_{i_0} *is a clause such that* $\overline{\Delta_1(A_{i_0})} \subseteq \Delta_0(A)$.

- ℓ-simple *if* A *is not a literal and* $\ell = \Delta_0(A) = \Delta_1(A)$.

The previous results state the amount of information in the Δ-lists which is enough to detect (un)satisfiability; when all these results are applied to a given formula, the resulting one is said to be Δ-*restricted*, and its formal definition is the following:

Definition 9 *Let* A *be an nnf, then it is said that* A *is* Δ-restricted *if either* A *is a logical constant or it satisfies the following conditions:*

- *it has no subtree which is either* Δ_0*-conclusive, or* Δ_1*-conclusive, or* ℓ*-simple,*

- *it has neither* $\top$ *nor* $\bot$ *leaves.*[3]

Remark 10 *From the previous results we can state that if* A *is a nnf, then by repeatedly application of the following sequence of steps we get a* Δ*-restricted formula:*

(1) Δ*-label.*

(2) Substitute subformulas $B \sqsubseteq A$ *by either* $\top$ *(if* B *is* Δ_1*-conclusive), or* $\bot$ *(if* B *is* Δ_0*-conclusive), or a literal* ℓ *(if* B *is* ℓ*-simple).*

(3) Simplify logical constants ($\top$ *or* $\bot$*), as soon as introduced, by using the 0-1-laws.*

This transformation can be made in just one reverse depth-first traverse of the formula, in which the content of the labels is read at most twice. Therefore, the conversion into Δ*-restricted form is linear.*

[3] Although the input formula is supposed not to contain occurrences of logical constants, they can be introduced by the reductions as we will see.

Example 11 *Given the formula*

$$(((q \to r) \wedge (p \vee (r \to t))) \wedge (q \vee (t \to s))) \to ((p \wedge \neg(q \to \neg t)) \vee (r \to ((q \to (s \vee r)) \wedge s)))$$

a linear transformation allows to get a nnf which is equivalent to its negation, A, depicted in Fig. 1 (up).

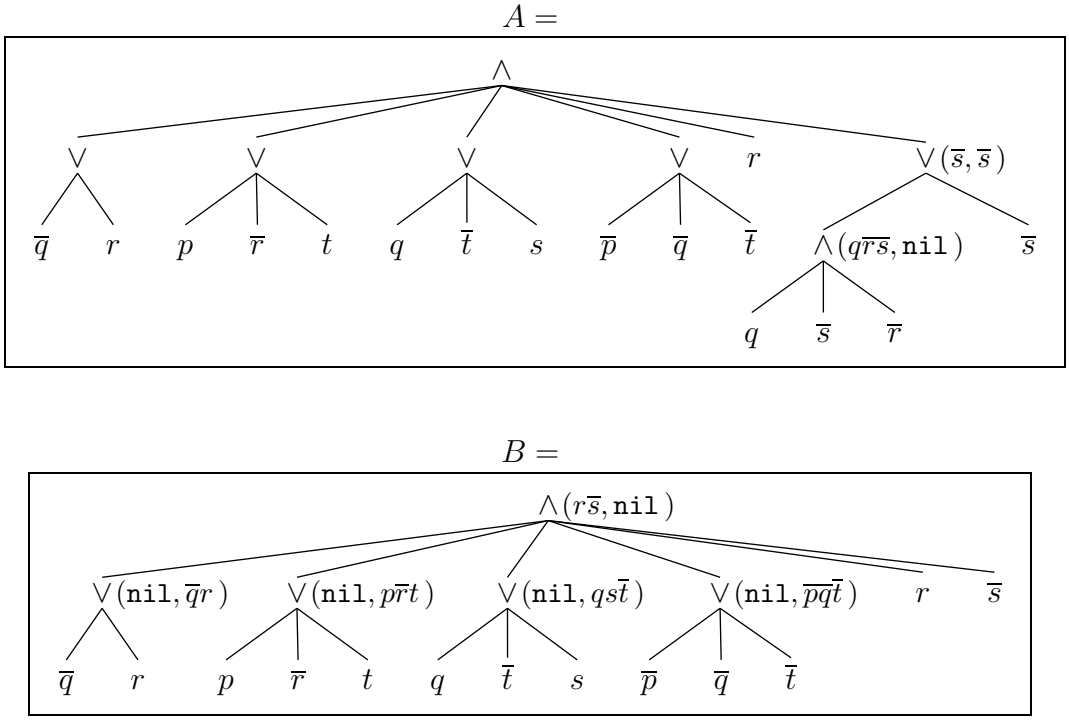


Fig. 1. The formula B is Δ-restricted and equivalent to A

When Δ-labelling A, as stated in Remark 10, node 6 (the right-most branch) is detected to be $\overline{s}$-simple; then the subtree is substituted. The Δ-label process finishes with no more transformations and the Δ-labelled tree B, in the Fig. 1 (down), is obtained.

New applications of the Δ-lists to get information (up to equivalence) of a formula A are given by the following theorem and its corollary.

Theorem 12 *Let A be a nnf and ℓ be a literal in A, then:*

(1) If $\ell \in \Delta_0(A)$, then $A \equiv \ell \wedge A[\ell/\top, \overline{\ell}/\bot]$

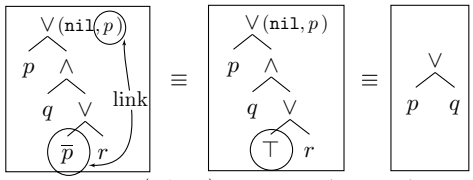
(2) If $\ell \in \Delta_1(A)$, then $A \equiv \ell \vee A[\ell/\bot, \overline{\ell}/\top]$

PROOF. 1. Let I be an assignment; we have to prove that $I(A) = I(\ell \wedge A[\ell/\top, \overline{\ell}/\bot])$:

- If $I(\ell) = 0$ then, by Theorem 3 (item 1) since $\ell \in \Delta_0(A)$, we have that $A \equiv \ell \wedge A$, therefore $I(A) = 0$. Now the result is obvious.
- If $I(\ell) = 1$ then $I(A) = I(A[\ell/\top, \overline{\ell}/\bot]) = I(\ell \wedge A[\ell/\top, \overline{\ell}/\bot])$.

The second item is proved similarly.

The following example shows an application of the theorem above:



(Th. 2) (Δ-label)

The theorem states how to simplify a formula when a link between an element of a Δ-list and a literal is found.

As an immediate consequence of the previous theorem, the following satisfiability-preserving result can be stated, which will be used later:

Corollary 13 *Let A be a nnf. If $\ell \in \Delta_0(A)$, then A and $A[\ell/\top, \overline{\ell}/\bot]$ are equisatisfiable. Furthermore, if I is a model of $A[\ell/\top, \overline{\ell}/\bot]$, then the extension I' of I such that $I'(\ell) = 1$ is a model of A.*

The following theorem allows to substitute a whole subformula C of A (not just literals as in Theorem 12) by a logical constant.

Theorem 14 *Let A be a nnf, $C \sqsubset A$ then:*

(1) If $\ell \in \Delta_1(A)$ and $\ell \in \Delta_0(C)$, then $A \equiv \ell \vee A[C/\bot]$

(2) If $\ell \in \Delta_1(A)$ and $\overline{\ell} \in \Delta_1(C)$, then $A \equiv \ell \vee A[C/\top]$

(3) If $\ell \in \Delta_0(A)$ and $\overline{\ell} \in \Delta_0(C)$, then $A \equiv \ell \wedge A[C/\bot]$

(4) If $\ell \in \Delta_0(A)$ and $\ell \in \Delta_1(C)$, then $A \equiv \ell \wedge A[C/\top]$

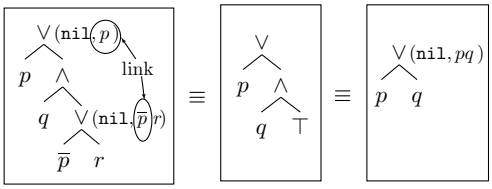
PROOF.

(1) By Theorem 3, we have $A \equiv \ell \vee A$ and $C \equiv \ell \wedge C$. Let I be an interpretation, then

- If $I(\ell) = 1$, then $I(A) = 1 = I(\ell \vee A[C/\bot])$.
- If $I(\ell) = 0$, then $I(C) = 0$ and $I(A) = I(A[C/\bot]) = I(\ell \vee A[C/\bot])$

The rest of the items are proved similarly.

This theorem improves Theorem 12 in that it defines transformations from links between elements of two Δ-lists; although the relevance of the theorem from the TAS standpoint will be shown in the introduction of the $\widehat{\Delta}$-sets.



(Th. 3) (Δ-label)

2.3 The $\widehat{\Delta}$-sets

In the previous section, the information in the Δ-lists has been used *locally*, that is, the information in $\Delta_b(\eta)$ has been used to reduce node η, by using Theorem 3. In this section, the purpose of defining a new structure, the $\widehat{\Delta}$-sets, is to allow the globalisation of the information, in that the information in $\Delta_b(\eta)$ can be *refined* by the information in its ancestors.

Given a Δ-restricted nnf A, we define the sets $\widehat{\Delta_b}(A)$, for $b \in \{0, 1\}$, whose elements are pairs (α, η) where α is a *filtered* Δ_b-list associated to a subformula B of A, and η is the address of B in A. In Section 2.4 we will see how to transform the formula A into an equisatisfiable and smaller sized one by using these sets.

The definition of the $\widehat{\Delta}$-sets is based on an operator (`Filter`) which *filters* information in the Δ-lists according to Theorems 12 and 14. These theorems describe substitutions applicable to a node with respect to the *dominant* literals, i.e. those occurring in the labels of some ancestor. Specifically, some literals in the Δ-lists can allow substitutions of subformulas by either $\top$ or $\bot$ as a consequence of Theorem 14; on the other hand, when this theorem is not applicable, it is still possible to delete the dominated literals, as an application

of Theorem 12. In the definition of the Δ-sets, these dominated literals will not be deleted but *framed*, therefore we are allowed to assume that a framed literal does or does not occur in the formula according to our necessities.

Definition 15 *Given a Δ-restricted nnf A and $B \sqsubseteq A$ we have:*

- $\texttt{Filter}(\Delta_0(B))$ *is:*

(1) $\bot$*, if there is a literal $\ell \in \Delta_0(B)$ such that either $\ell \in \Delta_1(B')$ or $\overline{\ell} \in \Delta_0(B')$ where $B \sqsubset B' \sqsubseteq A$.*

This is a consequence of Theorem 14, items 1 and 3.

(2) The result of framing any literal $\ell \in \Delta_0(B)$ satisfying either $\ell \in \Delta_0(B')$ or $\overline{\ell} \in \Delta_1(B')$ where $B \sqsubset B' \sqsubseteq A$.

This is a consequence of Theorem 12, items 1 and 2.

- $\texttt{Filter}(\Delta_1(B))$ *is*

(1) $\top$*, if there is a literal $\ell \in \Delta_1(B)$ such that either $\ell \in \Delta_0(B')$ or $\overline{\ell} \in \Delta_1(B')$ where $B \sqsubset B' \sqsubseteq A$.*

This is a consequence of Theorem 14, items 2 and 4.

(2) The result of framing any literal $\ell \in \Delta_1(B)$ satisfying either $\ell \in \Delta_1(B')$ or $\overline{\ell} \in \Delta_0(B')$ where $B \sqsubset B' \sqsubseteq A$.

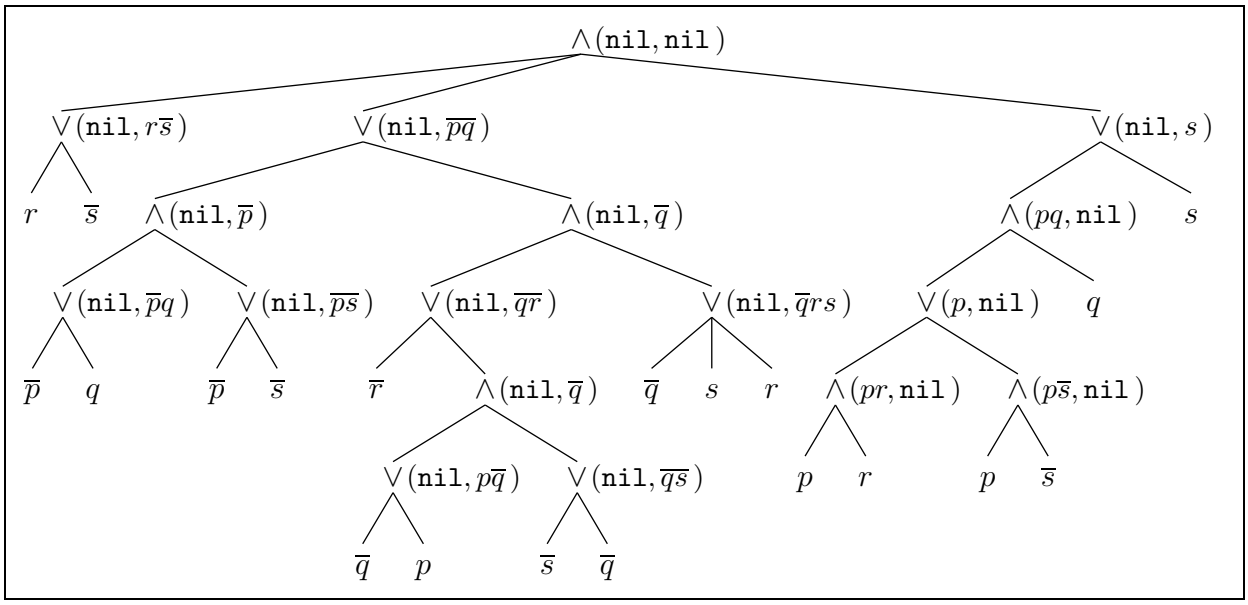
This is a consequence of Theorem 12, items 2 and 1.

Definition 16 *Let A be a Δ-restricted formula. For $b \in \{0, 1\}$, the set $\widehat{\Delta_b}(A)$ is recursively defined as follows:*

- *If ℓ is a literal, then $\widehat{\Delta_0}(\ell) = \widehat{\Delta_1}(\ell) = \varnothing$*
- *Otherwise, $\widehat{\Delta_b}(A) = \{(\texttt{Filter}(\Delta_b(B)), \eta_B) \mid B$ is a subformula of A and $\Delta_b(B) \neq \texttt{nil}\}$*

In the following example we present a step-by-step calculation of $\widehat{\Delta_b}$-sets.

Example 17 *Consider the formula A whose Δ-labelled tree appears below,*



For this tree, we have

$$\widehat{\Delta_0}(A) = \{(\boxed{p}r, 3111), (\boxed{p}\,\boxed{\bar{s}}, 3112), (\boxed{p}, 311), (pq, 31)\}$$

Literal p in nodes 3111, 3112 and 311 is framed because of its occurrence in $\Delta_0(31) = pq$; and literal $\bar{s}$ in node 3112 is framed because of the occurrence of s in $\Delta_1(3) = s$.

On the other hand, the $\widehat{\Delta_1}$-set for A is the following:

$$\begin{aligned}\widehat{\Delta_1}(A) = \{&(r\bar{s}, 1), (\top, 211), (\boxed{\bar{p}}\bar{s}, 212), (\boxed{\bar{p}}, 21), (\top, 22121), (\boxed{\bar{q}}\bar{s}, 22122),\\ &(\boxed{\bar{q}}, 2212), (\boxed{\bar{q}}\bar{r}, 221), (\boxed{\bar{q}}rs, 222), (\boxed{\bar{q}}, 22), (\bar{p}\bar{q}, 2), (s, 3)\}\end{aligned}$$

Node 211 is substituted by $\top$ because of the occurrences of $\bar{q}$ in $\Delta_1(2)$ and q in $\Delta_1(211)$; and node 22121 is substituted by $\top$ because of the occurrences of p in $\Delta_1(2)$ and $\bar{p}$ in $\Delta_1(22121)$. Finally, the occurrences of $\bar{p}$ are framed because of the occurrence of $\bar{p}$ in $\Delta_1(2)$, and the occurrences of $\bar{q}$ are framed because of the occurrence of $\bar{q}$ in $\Delta_1(2)$.

2.4 $\widehat{\Delta}$-sets and meaning-preserving results

In this section we study the information which can be extracted from the $\widehat{\Delta}$-sets. This is stated in Theorem 18, and in its proof we will use the following facts about the $\widehat{\Delta}$-sets of a given Δ-restricted nnf A:

- No element in $\widehat{\Delta_1}(A)$ is (α, ε), since A is a Δ-restricted nnf and cannot be finalizable.

- If $(\alpha, \eta) \in \widehat{\Delta_b}(A)$, then η is not the address of a leaf of A (since $\widehat{\Delta_0}(\ell) = \widehat{\Delta_1}(\ell) = \varnothing$ for all literal ℓ).

- If $\alpha = \Delta_0(A)$, then $(\alpha, \varepsilon) \in \widehat{\Delta_0}(A)$ and the list α does not have framed literals. (Just note that a literal $\ell \in \alpha$ is framed in (α, η) from the information in the Δ-lists of its ancestors).

The following theorem states that, similar to the Δ-labels, the $\widehat{\Delta}$-labels also allow substituting subformulas in A by either $\top$ or $\bot$.

Theorem 18 *Let A be a Δ-restricted nnf then*

(1) If $(\bot, \eta) \in \widehat{\Delta_0}(A)$, then $A \equiv A[\eta/\bot]$.

(2) If $(\top, \eta) \in \widehat{\Delta_1}(A)$, then $A \equiv A[\eta/\top]$.

PROOF.

(1) Suppose $(\bot, \eta) \in \widehat{\Delta_0}(A)$ and let C be the subformula at address η. By the definition of $\widehat{\Delta_0}(A)$ there exist a formula B such that $C \sqsubset B \sqsubseteq A$ and a literal ℓ satisfying

$$\ell \in \Delta_0(C) \qquad \text{and} \qquad \text{either } \ell \in \Delta_1(B) \text{ or } \overline{\ell} \in \Delta_0(B) \tag{1}$$

By Corollary 4, using that $\ell \in \Delta_0(C)$, that the address η cannot correspond to a leaf, and that A is a Δ-restricted nnf (specifically, A does not have ℓ-simple subformulas), we get that $\Delta_1(C) = \texttt{nil}$.

Note that, clearly, it is enough to prove that $B \equiv B[C/\bot]$.

Firstly, we will prove that, under these hypotheses:

(a) If $\ell \in \Delta_1(B)$, then $\ell \in \Delta_1(B[C/\bot])$.

(b) If $\overline{\ell} \in \Delta_0(B)$, then $\overline{\ell} \in \Delta_0(B[C/\bot])$.

Proof of (a): By induction on the depth of η in B, denoted $d_B(\eta)$.

(i) If $d_B(\eta) = 1$, then either $B = C \wedge D$ or $B = C \vee D$ (up to commutativity and associativity).

It cannot be the case that $B = C \wedge D$, since $\ell \in \Delta_1(B)$ and, as $\Delta_1(B) = \Delta_1(C) \cap \Delta_1(D)$, we would have $\ell \in \Delta_1(C)$, which contradicts the fact that $\Delta_1(C) = \mathtt{nil}$.

Therefore, we must have $B = C \vee D$ and, consequently, $B[C/\bot] = \bot \vee D$. Now, using again $\Delta_1(C) = \mathtt{nil}$, and $\Delta_1(B) = \Delta_1(C) \cup \Delta_1(D)$, we have that $\ell \in \Delta_1(D)$ and, therefore, $\ell \in \Delta_1(\bot \vee D) = \Delta_1(B[C/\bot])$.

(ii) Assume the result for $d_X(\eta) \leq k - 1$ and let us prove it for $d_B(\eta) = k$:

· If $B = B_1 \vee B_2$, $C \sqsubset B_1$ and $\ell \in \Delta_1(B_2)$, then the result is obvious.

· If $B = B_1 \vee B_2$, $C \sqsubset B_1$ and $\ell \in \Delta_1(B_1)$, then by the induction hypothesis we have that $\ell \in \Delta_1(B_1[C/\bot]) \subseteq \Delta_1(B[C/\bot])$.

· If $B = B_1 \wedge B_2$ and $C \sqsubset B_1$, then $\ell \in \Delta_1(B_1)$ and $\ell \in \Delta_1(B_2)$. Now, by the induction hypothesis, $\ell \in \Delta_1(B_1[C/\bot]) \cap \Delta_1(B_2) \subseteq \Delta_1(B[C/\bot])$.

· The cases $C \sqsubset B_2$ are similar.

The proof of (b) is obtained by duality.

Finally, we prove $B \equiv B[C/\bot]$ by considering the two possibilities in (1) above:

- If $\ell \in \Delta_0(C)$ and $\ell \in \Delta_1(B)$, then

$$\begin{array}{rcll} B & \equiv & \ell \vee B[C/\bot] & \text{by Theorem 14} \\ & \equiv & B[C/\bot] & \text{by (a) and Theorem 3} \end{array}$$

- If $\ell \in \Delta_0(C)$ and $\overline{\ell} \in \Delta_0(B)$, then

$$\begin{array}{rcll} B & \equiv & \overline{\ell} \wedge B[C/\bot] & \text{by Theorem 14} \\ & \equiv & B[C/\bot] & \text{by (b) and Theorem 3} \end{array}$$

(2) The proof is similar.

Note that this theorem introduces a meaning-preserving transformation which allows substituting a subformula by a constant. The information given by the Δ-lists substitutes subformulas which are equivalent to either $\top$ (Δ_1-conclusive) or $\bot$ (Δ_0-conclusive); however, under the hypotheses of this theorem, it need not be true that η is equivalent to either $\top$ or $\bot$.

Definition 19 *Let A be an nnf then it is said that A is* restricted *if it is Δ-restricted and satisfies the following:*

- *There are no elements $(\bot, \eta)$ in $\widehat{\Delta_0}(A)$.*
- *There are no elements $(\top, \eta)$ in $\widehat{\Delta_1}(A)$.*

If A is a nnf, to label *A means Δ-label and associate to the root of A the ordered pair $(\widehat{\Delta_0}(A), \widehat{\Delta_1}(A))$.*

Note that given a Δ-restricted nnf, A, after calculating $(\widehat{\Delta_0}(A), \widehat{\Delta_1}(A))$ we get either the (un)satisfiability of A or an equivalent and restricted nnf by means of the substitutions determined by Theorem 18, and the 0-1*-laws.*

2.5 $\widehat{\Delta}$-sets and satisfiability-preserving results

The following results will allow, by using the information in the $\widehat{\Delta}$-sets, substitution of a nnf A by an equisatisfiable and smaller sized A' with no occurrences of some literals occurring in A.

2.5.1 A complete reduction theorem

To begin with, Corollary 13 can be stated in terms of the $\widehat{\Delta}$-sets as follows:

Theorem 20 (Complete reduction) *Let A be a nnf such that $(\alpha, \varepsilon) \in \widehat{\Delta_0}(A)$, then A is satisfiable if and only if $A[\alpha/\top, \overline{\alpha}/\bot]$ is satisfiable. Furthermore, if I is a model of $A[\alpha/\top, \overline{\alpha}/\bot]$, then the extension I' of I such that $I'(\ell) = 1$ for all $\ell \in \alpha$ is a model of A.*

Note that this result allows elimination of all the occurrences of all the literals appearing in α, that is why it is named complete reduction. Its usefulness will be shown in the examples.

2.5.2 Generalised pure literal rule

The introduction of the $\widehat{\Delta}$-sets allows a generalisation of the well-known pure literal rule for sets of clauses. Firstly, recall the standard definition and result for a formula in nnf:

Definition 21 *Let ℓ be a literal occurring in a nnf A. Literal ℓ is said to be* pure *in A if $\overline{\ell}$ does not occur in A.*

Lemma 22 *Let A be a nnf and ℓ a pure literal in A then A is satisfiable iff $A[\ell/\top]$ is satisfiable. Furthermore, if I is a model of $A[\ell/\top]$, then the extension I' of I such that $I'(\ell) = 1$ is a model of A.*

Our $\widehat{\Delta}$-sets allow to generalise the definition of pure literal and, as a consequence, to get an extension of the lemma above.

Definition 23 *Let A be a nnf. A literal ℓ is said to be* $\widehat{\Delta}$-pure *in A if it satisfies the following conditions:*

(1) ℓ occurs in $\widehat{\Delta_0}(A) \cup \widehat{\Delta_1}(A)$.

(2) All the occurrences of $\overline{\ell}$ in $\widehat{\Delta_0}(A) \cup \widehat{\Delta_1}(A)$ are framed.

The next theorem is a proper extension of Lemma 22, for it can be applied even when ℓ and $\overline{\ell}$ occur in A. The idea here is that framed literals can be deleted, however we need not to delete them before studying the existence of pure literals. The associated transformation can be easily defined as stated in the theorem below.

Theorem 24 (Generalised pure literal rule) *Let A be a nnf, let ℓ be a $\widehat{\Delta}$-pure literal in A and let B be the formula obtained from A by the following substitutions:*

(i) If $(\alpha, \eta) \in \widehat{\Delta_0}(A)$ with $\ell \in \alpha$, then node η in A is substituted by $\eta[\ell/\top, \overline{\ell}/\bot]$.

(ii) If $(\alpha, \eta) \in \widehat{\Delta_1}(A)$ with $\ell \in \alpha$, then node η in A is substituted by $\top$.

Then A is satisfiable if and only if B is satisfiable. Furthermore, if I is a model of B, then the extension I' of I such that $I'(\ell) = 1$ is a model of A.

PROOF. By Theorem 12, and the definition of the $\widehat{\Delta}$-sets, we have that

(a) If $(\alpha, \eta) \in \widehat{\Delta_0}(A)$, then $\alpha = \Delta_0(\eta)$; and if, in addition, we have $\ell \in \alpha$, then

$$A \equiv A[\eta/(\ell \wedge \eta[\ell/\top, \overline{\ell}/\bot])]$$

(b) If $(\alpha, \eta) \in \widehat{\Delta_1}(A)$, then $\alpha = \Delta_1(\eta)$; and if, in addition, we have $\ell \in \alpha$, then

$$A \equiv A[\eta/(\ell \vee \eta[\ell/\bot, \overline{\ell}/\top])]$$

Therefore, if we consider the formula A', obtained when applying the equivalences of items (a) and (b), we get that literal ℓ is pure in A'. Now, by an application of Lemma 22 to A' we get the formula B, which completes the proof.

In the rest of the section we introduce the necessary definitions to extend the collapsibility results introduced in [**?**].

2.5.3 Collapsibility theorems

Definition 25 *Let A be a nnf and ℓ_1 and ℓ_2 literals in A. Literals ℓ_1 and ℓ_2 are* 0-1-bound *if the following conditions are satisfied:*

(1) There are no occurrences[4] *of either $\overline{\ell_1}$ or $\overline{\ell_2}$ in $\widehat{\Delta_0}(A)$, and for all $(\alpha, \eta) \in \widehat{\Delta_0}(A)$ we have that $\ell_1 \in \alpha$ iff $\ell_2 \in \alpha$.*

(2) There are no occurrences of either ℓ_1 or ℓ_2 in $\widehat{\Delta_1}(A)$, and for all $(\alpha, \eta) \in \widehat{\Delta_1}(A)$ we have that $\overline{\ell_1} \in \alpha$ iff $\overline{\ell_2} \in \alpha$.

From the definition of $\widehat{\Delta}$-sets we have that:

Remark 26 *If ℓ_1 and ℓ_2 are* 0-1*-bound in A, then every leaf in A with a literal in $\{\ell_1, \ell_2, \overline{\ell_1}, \overline{\ell_2}\}$ has an ancestor η in A which is maximal in the sense that its associated Δ-lists satisfy one of the following conditions but its ancestors' Δ-lists do not:*

- *$\ell_1, \ell_2 \in \Delta_0(\eta)$ and if η' is an ancestor of η, then none of the literals $\ell_1, \ell_2, \overline{\ell_1}, \overline{\ell_2}$ occur in the Δ-lists associated with η'.*

[4] In this section, when we say an occurrence of ℓ, we mean an *unframed* occurrence of ℓ.

- $\overline{\ell_1}, \overline{\ell_2} \in \Delta_1(\eta)$ *and if* η' *is an ancestor of* η*, then none of the literals* $\ell_1, \ell_2, \overline{\ell_1}, \overline{\ell_2}$ *occur in the* Δ*-lists associated with* η'*.*

We will use the following notation in the proof of the collapsibility results, where ℓ_i are literals, and $b \in \{0, 1\}$:

$$\mathcal{S}_{\ell_1 \cdots \ell_n, b}(A) = \{S \sqsubseteq A \mid \ell_1, \ldots, \ell_n \in \Delta_b(S)\}$$

Theorem 27 (Collapsibility) *Let* A *be a nnf and let* ℓ_1 *and* ℓ_2 *be literals in* A*. If* ℓ_1 *and* ℓ_2 *are* 0-1*-bound, then* A *is satisfiable if and only if* $A[\ell_1/\top, \overline{\ell_1}/\bot]$ *is satisfiable. Furthermore, if* I *is a model of* $A[\ell_1/\top, \overline{\ell_1}/\bot]$*, then any extension* I' *of* I *such that* $I'(\ell_1) = 1$ *is a model of* A*.*

PROOF. The *if* part is immediate. For the *only if* part, let us suppose A is satisfiable.

Let I be a satisfying assignment for A. If $I(\ell_1) = 1$, there is nothing to prove; so, let us consider $I(\ell_1) = 0$ and prove that A is also satisfied by an assignment I' such that $I'(\ell_1) = 1$.

From Remark 26, A can be considered as a formula in the language with the following set of atoms

$$\mathcal{S}_{\ell_1 \ell_2, 0}(A) \cup \mathcal{S}_{\overline{\ell_1 \ell_2}, 1}(A) \cup (\mathcal{V}^{\pm} \setminus \{\ell_1, \ell_2, \overline{\ell_1}, \overline{\ell_2}\})$$

that is, in A every leaf is either a formula in $\mathcal{S}_{\ell_1 \ell_2, 0}(A) \cup \mathcal{S}_{\overline{\ell_1 \ell_2}, 1}(A)$ or a literal $\ell \notin \{\ell_1, \ell_2, \overline{\ell_1}, \overline{\ell_2}\}$.

Note that if $S_1 \in \mathcal{S}_{\ell_1 \ell_2, 0}(A)$, then we have that $I(S_1) = 0$, and if $S_2 \in \mathcal{S}_{\overline{\ell_1 \ell_2}, 1}(A)$, then $I(S_2) = 1$.

Let I' be the assignment obtained from I by changing the values on ℓ_i as follows:

$$I'(\ell_1) = 1 \text{ and } I'(\ell_2) = 0$$

This assignment satisfies $I'(S_1) = 0 = I(S_1)$ and $I'(S_2) = 1 = I(S_2)$, and coincides with I in the rest of leaves. Therefore I' is a satisfying assignment for A with $I'(\ell_1) = 1$.

This result is a generalisation of van Gelder's collapsibility lemma, which treats the case in which all the occurrences of $\ell_1, \ell_2, \overline{\ell_1}, \overline{\ell_2}$ are bound as $\ell_1 \wedge \ell_2$ and $\overline{\ell_1} \vee \overline{\ell_2}$; so that ℓ_1 and ℓ_2 can be represented by a single literal with $\ell = \ell_1 \wedge \ell_2$, see [**?**] for the details. Our result drops the requirement that all the occurrences in the defining subset of ℓ_1 and ℓ_2 have to be children of a $\wedge$ node and the occurrences in the defining subset of $\{\overline{\ell_1}, \overline{\ell_2}\}$ have to be children of a $\vee$ node.

Obviously, the previous result can be easily extended to the case of n literals which can be collapsed into one.

Definition 28 *Let A be a nnf and let $\ell_1, \ldots, \ell_n$ be literals in A, literals $\ell_1, \ldots, \ell_n$ are* 0-1-bound *if the following conditions are satisfied:*

(1) In $\widehat{\Delta_0}(A)$ there are no occurrences of $\overline{\ell_1}, \ldots, \overline{\ell_n}$ and if $(\alpha, \eta) \in \widehat{\Delta_0}(A)$ for all $i, j \in \{1, \ldots, n\}$, then we have that $\ell_i \in \alpha$ iff $\ell_j \in \alpha$.

(2) In $\widehat{\Delta_1}(A)$ there are no occurrences of $\ell_1, \ldots, \ell_n$ and if $(\alpha, \eta) \in \widehat{\Delta_1}(A)$ for all $i, j \in \{1, \ldots, n\}$, then we have that $\overline{\ell_i} \in \alpha$ iff $\overline{\ell_j} \in \alpha$.

Corollary 29 (Generalised collapsibility) *Let A be a nnf, and let $\ell_1, \ldots, \ell_n$ be literals* 0-1-*bound in A then A is satisfiable iff $A[\ell_1/\top, \ldots, \ell_{n-1}/\top, \overline{\ell_1}/\bot, \ldots, \overline{\ell_{n-1}}/\bot]$ is satisfiable. Furthermore, if I is a model of $A[\ell_1/\top, \ldots, \ell_{n-1}/\top, \overline{\ell_1}/\bot, \ldots, \overline{\ell_{n-1}}/\bot]$, then any extension I' of I such that $I'(\ell_j) = 1$ for all $j \in \{1, \ldots, n-1\}$ is a model of A.*

Example 30 *van Gelder's reduction lemmas cannot be applied to the formula in Example 17, but it is collapsible in the sense of Theorem 27. We had the following $\widehat{\Delta}$-sets:*

$$\widehat{\Delta_0}(A) = \{(\boxed{p}r, 3111), (\boxed{p}\,\boxed{\bar{s}}, 3112), (\boxed{p}, 311), (pq, 31)\}$$
$$\widehat{\Delta_1}(A) = \{(r\bar{s}, 1), (\top, 211), (\boxed{\bar{p}}\bar{s}, 212), (\boxed{\bar{p}}, 21), (\top, 22121), (\boxed{\bar{q}}\bar{s}, 22122),$$
$$(\boxed{\bar{q}}, 2212), (\boxed{\bar{q}}\bar{r}, 221), (\boxed{\bar{q}}rs, 222), (\boxed{\bar{q}}, 22), (\bar{p}\bar{q}, 2), (s, 3)\}$$

Specifically, p and q are 0-1-bound.

In order to state the generalisation of mixed collapsibility we need the following definition:

Definition 31 *Let A be a nnf, $b \in \{0, 1\}$ and let ℓ_1 and ℓ_2 be literals in A.*

Literal ℓ_1 *is* b-bounded to ℓ_2 *if the following conditions are satisfied*

(1) In $\widehat{\Delta_{\overline{b}}}(A)$ *there are no occurrences (neither framed nor unframed) of either* ℓ_1 *or* $\overline{\ell_1}$.

(2) If $(\alpha, \eta) \in \widehat{\Delta_b}(A)$, *then we have that*

- *If* $\ell_1 \in \alpha$, *then* $\ell_2 \in \alpha$.
- *If* $\overline{\ell_1} \in \alpha$ *then* $\overline{\ell_2} \in \alpha$.

By this definition if ℓ_1 is b-bound to ℓ_2 in a formula A, then every leaf of A belonging to $\{\ell_1, \overline{\ell_1}\}$ has an ancestor η in $\mathcal{S}_{\ell_1\ell_2,b}(A) \cup \mathcal{S}_{\overline{\ell_1\ell_2},b}(A)$.

Theorem 32 (Mixed collapsibility) *Let* A *be a nnf and* ℓ_1, ℓ_2 *literals in* A,

(1) If ℓ_1 *is 0-bound to* ℓ_2, *and* A' *is the formula obtained from* A *by applying the following substitutions*

- *If* $(\alpha, \eta) \in \widehat{\Delta_0}(A)$ *and* $\ell_1, \ell_2 \in \alpha$, *then* η *is substituted in* A *by* $\eta[\ell_1/\top, \overline{\ell_1}/\bot]$.
- *If* $(\alpha, \eta) \in \widehat{\Delta_0}(A)$ *and* $\overline{\ell_1}, \overline{\ell_2} \in \alpha$, *then* η *is substituted in* A *by* $\eta[\ell_1/\bot, \overline{\ell_1}/\top]$.

then A *is satisfiable if and only if* A' *is satisfiable. In addition, if* I *is a satisfying assignment of* A', *then any extension* I'*of* I *such that* $I(\ell_1) = I(\ell_2)$ *is a satisfying assignment for* A.

(2) If ℓ_1 *is 1-bound to* ℓ_2, *and* A' *is the formula obtained from* A *by applying the following substitutions*

- *If* $(\alpha, \eta) \in \widehat{\Delta_1}(A)$ *and either* $\ell_1, \ell_2 \in \alpha$ *or* $\overline{\ell_1}, \overline{\ell_2} \in \alpha$, *then* η *is substituted in* A *by* $\top$.

then A *is satisfiable if and only if* A' *is satisfiable. In addition, if* I *is a satisfying assignment of* A', *then any extension* I' *of* I *such that* $I(\ell_1) = I(\overline{\ell_2})$ *is a satisfying assignment for* A.

PROOF. 1. Note that A can be considered as a formula in the language with set of atoms

$$\mathcal{S}_{\ell_1\ell_2,0}(A) \cup \mathcal{S}_{\overline{\ell_1\ell_2},0}(A) \cup (\mathcal{V}^{\pm} \setminus \{\ell_1, \overline{\ell_1}\})$$

that is, in A every leaf is either a formula in $\mathcal{S}_{\ell_1\ell_2,0}(A) \cup \mathcal{S}_{\overline{\ell_1\ell_2},0}(A)$ or is a literal $\ell \notin \{\ell_1, \overline{\ell_1}\}$.

Let I be a satisfying assignment for A:

(1) if $I(\ell_1) = I(\ell_2)$, then for every leaf S in $\mathcal{S}_{\ell_1\ell_2,0}(A)$ we have, by Theorem 12, that $I(S) = I(S[\ell_1/\top, \overline{\ell_1}/\bot])$ and for every leaf S in $\mathcal{S}_{\overline{\ell_1\ell_2},0}(A)$ we have, again by Theorem 12, $I(S) = I(S[\ell_1/\bot, \overline{\ell_1}/\top])$. Therefore, $I(A) = 1 = I(A')$.

(2) If $I(\ell_1) \neq I(\ell_2)$, then for every leaf S in $\mathcal{S}_{\ell_1\ell_2,0}(A) \cup \mathcal{S}_{\overline{\ell_1\ell_2},0}(A)$ we have, by Theorem 12, $I(S) = 0$. Consider the assignment $I^\star$, obtained from I by changing only its value on ℓ_1; obviously, we have $I(S) \leq I^\star(S)$.

By monotonicity of boolean conjunction and disjunction, we have $1 = I(A) \leq I^\star(A) = I^\star(A')$. That is, A' is satisfiable.

Conversely, let I be a satisfying assignment for A' and let $I^\star$ any extension of I such that $I^\star(\ell_1) = I(\ell_2)$. By Theorem 12, for every leaf S in $\mathcal{S}_{\ell_1\ell_2,0}(A)$ we have $I(S) = I(S[\ell_1/\top, \overline{\ell_1}/\bot])$ and for every leaf S in $\mathcal{S}_{\overline{\ell_1\ell_2},0}(A)$ we have $I(S) = I(S[\ell_1/\bot, \overline{\ell_1}/\top])$. Thus, $I^\star(A) = I(A') = 1$.

2. The proof is similar.

Example 33 *Following with the formula in Example 11, for the formula B in figure 1, we had*

$$\widehat{\Delta_0}(B) = \{(r\overline{s}, \varepsilon)\} \qquad \textit{and} \qquad \widehat{\Delta_1}(B) = \{(\top, 1), (p\boxed{\overline{r}}t, 2), (q\boxed{s}\overline{t}, 3), (\overline{pq}\overline{t}, 4)\}$$

therefore the first subtree can be pruned, obtaining the tree in Fig. 2.

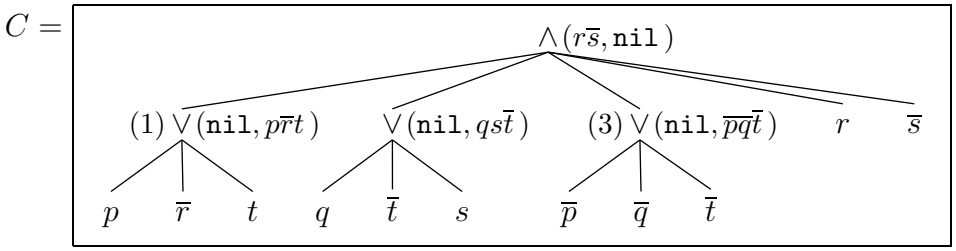


Fig. 2. The tree C.

Now, variables r and s can be deleted by Theorem 20 of complete reduction (for $(r\overline{s}, \varepsilon) \in \widehat{\Delta_0}(B)$), storing the information $(r = 1)$ and $(\overline{s} = 1)$ to be able

to generate a model (if it exists) of the input formula.

In addition, p is 1-bounded to t; therefore, by Theorem 32 of mixed collapsibility, (1) *and* (3) *can be substituted by* $\top$ *and the information* $(p = \overline{t})$ *is stored.*

The resulting formula is $q \vee \overline{t}$*, which is finalizable (for* $\Delta_1(q \vee \overline{t}) = q\overline{t} \neq \texttt{nil}$*). Specifically, it is satisfiable and a model is* $(q = 1)$*.*

We can deduce that the input formula in Example 11 is non-valid (for its negation is satisfiable); by collecting the stored information we get the following countermodel $I(r) = 1$*,* $I(s) = 0$*,* $I(q) = 1$ *and two possibilities:* $I(p) = 1$ *and* $I(t) = 0$ *(or* $I(p) = 0$ *and* $I(t) = 1$*) by the information* $(p = \overline{t})$ *.*

2.5.4 Splitting a formula

We finish the section by introducing a satisfiability-preserving result which prevents a branching when suitable hypotheses hold. The *splitting*, as we call it, results as a consequence of the following well-known theorem.

Theorem 34 (Quine) *A is satisfiable if and only if $A[p/\top] \vee A[p/\bot]$ is satisfiable. Furthermore, if I is a model of $A[p/\top]$, the extension of I with the assignment $I(p) = 1$ is a model of A; similarly, if I is a model of $A[p/\bot]$, the extension of I with the assignment $I(p) = 0$ is a model of A*

If no satisfiability-preserving reduction can be applied to a restricted conjunctive nnf, then we would have to branch. The following definition states a situation in which the formula need not cause a branch but instead can be split.

Definition 35 *Let $A = \bigwedge_{i \in J} A_i$ be a restricted nnf; A is said to be* p-splittable *if $J_p \cup J_{\overline{p}} = J$ where*

$$J_p = \{i \in J \mid p \in \Delta_1(A_i)\} \qquad J_{\overline{p}} = \{i \in J \mid \overline{p} \in \Delta_1(A_i)\}$$

Corollary 36 *Let $A = \bigwedge_{i \in J} A_i$ be a restricted and p-splittable nnf. Then A is satisfiable if and only if $\left(\bigwedge_{i \in J_p} A_i[p/\bot]\right) \vee \left(\bigwedge_{i \in J_{\overline{p}}} A_i[p/\top]\right)$ is satisfiable. Furthermore, if I is a model of $\bigwedge_{i \in J_{\overline{p}}} A_i[p/\top]$, then the extension of I with the assignment $I(p) = 1$ is a model of A; similarly, if I is a model of $\bigwedge_{i \in J_p} A_i[p/\bot]$, then the extension of I with the assignment $I(p) = 0$ is a model of A.*

This result can be seen as a generalisation of the Davis-Putnam rule with the following advantages:

- It can be shifted to non-classical logics.
- Branching is essentially a doubling in size, whereas splitting results in two subproblems whose joint size is at most the size of the original.
- Its interactions with the reduction strategies turn out to be extremely efficient.

Now, we can describe the algorithm of the prover following the steps we have applied in the previous examples.

3 The TAS-D algorithm

In this section we describe the algorithm TAS-D and its soundness and completeness are proved. The flowchart of the algorithm appears in Figure 3; we have to keep in mind that:

- TAS-D determines the (un)satisfiability of the input formula. Therefore, it can be viewed as a refutational ATP.
- The data flow of the algorithm is a pair $(B, \mathcal{M})$ where B is a nnf, and $\mathcal{M}$ is a set of expressions $(\ell = b)$ or $(\ell = \ell')$, where $b \in \{0, 1\}$ and ℓ is a literal not occurring in B.
- The elements in $\mathcal{M}$ define a partial interpretation for the input formula, which is used by `CollectInfo`, if necessary. This interpretation is defined as follows:

$$I(\ell) = b \quad \text{if } (\ell = b) \in \mathcal{M}$$
$$I(\ell) = I(\ell') \text{ if } (\ell = \ell') \in \mathcal{M}$$

  In general, due to the second condition, $\mathcal{M}$ might define more than one interpretation, depending on the choosing of $I(\ell')$.

The operators involved in the algorithm are described below, the soundness of each one follows from the results in previous sections:

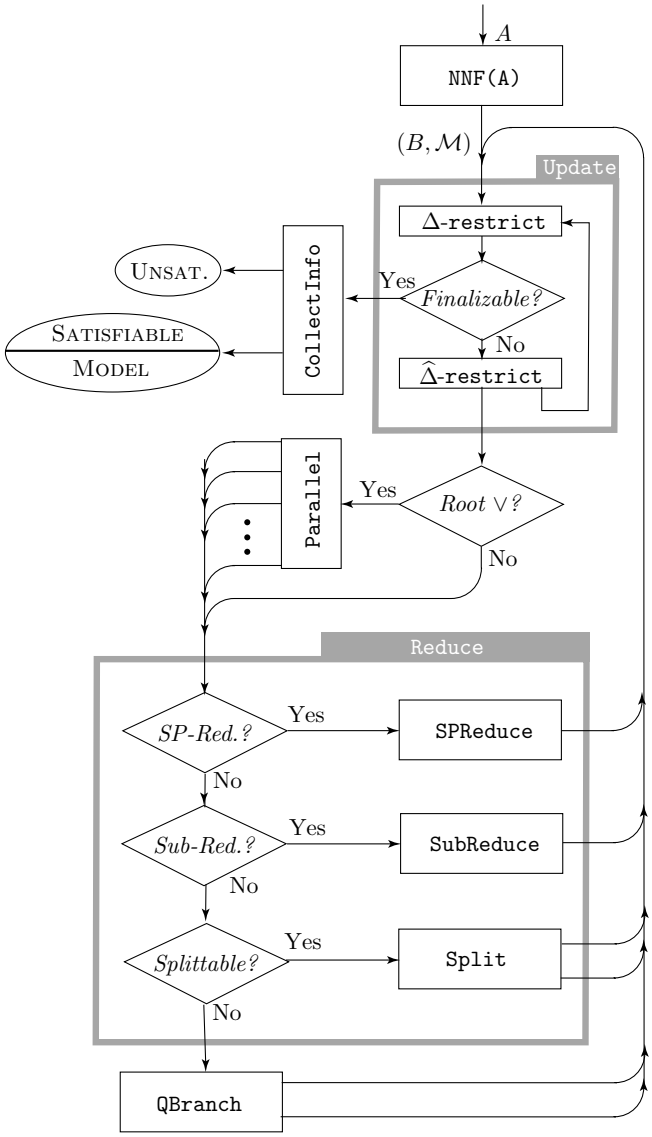


Fig. 3. Flowchart of the TAS-D algorithm.

The initialisation stage: `NNF`

The user's input A is translated into nnf by the operator `NNF`; specifically, $\texttt{NNF}(A) = (B, \varnothing)$ where B is a nnf which is equivalent to A.

The Update module

The different stages in the algorithm transform subtrees of the input tree; in each transformation, the labels of the ancestor nodes (of the transformed node) are deleted; `Update` processes these trees by recalculating the missing

labels and giving as output either a restricted nnf or a finalizable formula. From another point of view, this stage *updates* the formula in the sense that it prunes those subtrees that can be directly deduced to be equivalent either to $\top$, or $\bot$, or a literal.

3.0.5 The $\Delta-$`restrict` *operator*

The input of $\Delta-$`restrict` is a pair $(B, \mathcal{M})$, where B is a partially labelled formula, possibly with logical constants.

Given a nnf B we have that $\Delta-\texttt{restrict}(B, \mathcal{M}) = (C, \mathcal{M})$ where C is the Δ-restricted formula obtained from B as indicated in Definition 10.

3.0.6 The $\widehat{\Delta}-$`restrict` *operator*

The input of $\widehat{\Delta}-$`restrict` is a pair $(B, \mathcal{M})$ where B is a Δ-restricted formula. We have

$$\widehat{\Delta}-\texttt{restrict}(B, \mathcal{M}) = (C, \mathcal{M})$$

where C is the restricted formula obtained from B as indicated in Definition 19.

Parallelization

The input of `Parallel` is a pair $(B, \mathcal{M})$, where B is a restricted formula and $B = \bigvee_{i=1}^{n} B_i$. We have

$$\texttt{Parallel}(\bigvee_{i=1}^{n} B_i, \mathcal{M}) = ((B_1, \mathcal{M}), \ldots, (B_n, \mathcal{M}))$$

Since a disjunction is satisfiable iff some disjunct is satisfiable, each pair $(A_i, \mathcal{M})$ is independently passed to `Reduce`, the following module in the algorithm.

The input of `Reduce` is the labelled syntactic tree of a restricted nnf $B = \bigwedge_{i=1}^{n} B_i$. In this stage we decrease, if possible, the size of B before branching, by using the information provided by the $\widehat{\Delta}$-labels and the Δ-labels. Specifically,

- the $\widehat{\Delta}$-labels of the root node allow, using the `SPReduce` operator, to substitute B by an equisatisfiable formula in which some propositional variables have been eliminated;
- the Δ-labels of a proper subtree X allow, using the `SubReduce` operator, to substitute the subformula X by an equivalent formula in which the symbols in its Δ-lists occur exactly once.

3.0.7 The `SPReduce` *operator*

A restricted nnf B is said to be *SP-reducible* if either it is completely reducible (i.e. there is an element $(\alpha, \varepsilon) \in \widehat{\Delta_0}(B)$), or it has $\widehat{\Delta}$-pure literals, or it has a pair of 0-1 bound literals, or it has a literal b-bound to other literal; for these formulas we have

$$\texttt{SPReduce}(B, \mathcal{M}) = (C, \mathcal{M}')$$

where $(C, \mathcal{M}')$ is obtained by applying the following items:

(1) If $(\alpha, \varepsilon) \in \widehat{\Delta_0}(B)$, then $C = B[\alpha/\top, \overline{\alpha}/\bot]$ and $\mathcal{M}' = \mathcal{M} \cup \{(\ell = 1); \ell \in \alpha\}$, by Theorem 20.

(2) If ℓ is $\widehat{\Delta}$-pure in B, then C is the obtained formula after applying in B the substitutions in Theorem 24, and $\mathcal{M}' = \mathcal{M} \cup \{(\ell = 1)\}$.

(3) If ℓ and ℓ' are 0-1-bound, then C is the obtained formula after applying in B the substitutions in Theorem 27, and $\mathcal{M}' = \mathcal{M} \cup \{(\ell = 1)\}$.

(4) If ℓ is b-bound to ℓ', then C is the obtained formula after applying in B the substitutions in Theorem 32, and

$$\mathcal{M}' = \begin{cases} \mathcal{M} \cup \{(\ell = \ell')\} & \text{if } b = 0 \\ \mathcal{M} \cup \{(\ell = \overline{\ell'})\} & \text{if } b = 1 \end{cases}$$

3.0.8 The `SubReduce` *operator*

The input of `SubReduce` is a restricted, not SP-reducible nnf A; its effect can be described as an application of Theorem 12 up to associativity and commutativity. The formal definition needs some extra terminology, included below:

Definition 37 *Let a $A = \Theta_{i\in I} A_i$ be formula such that is not SP-reducible, and consider $\bigcup_{i\in I} \Delta_b(A_i) = \{\ell_1, \ldots, \ell_n\}$, where $b = 1$ if $\Theta = \bigwedge$ and $b = 0$ if $\Theta = \bigvee$.*

The integers denoted by $m(\ell_j)$, defined below, are associated to A:

$$m(\ell_j) = |\{i \in I \mid \ell_j \in \Delta_b(A_i)\}|$$

where $|\cdot|$ denotes the cardinality of a finite set.

*It is said that A is ℓ-*reducible *if $m(\ell) > 1$ and*

$$m(\ell) = \max\{m(\ell_j) \text{ associated with } A\}$$

*Let A be ℓ-*reducible *and consider $J = \{i \in I \mid \ell \in \Delta_b(A_i)\}$, the formula A_ℓ is defined as follows, by application of Theorem 12*

$$A_\ell = \begin{cases} \Big(\ell \vee \bigwedge\limits_{i\in J} A_i[\ell/\bot, \overline{\ell}/\top]\Big) \wedge \Big(\bigwedge\limits_{i\in I\smallsetminus J} A_i\Big) \text{ if } \Theta = \bigwedge, \\ \\ \Big(\ell \wedge \bigvee\limits_{i\in J} A_i[\ell/\top, \overline{\ell}/\bot]\Big) \vee \Big(\bigvee\limits_{i\in I\smallsetminus J} A_i\Big) \text{ if } \Theta = \bigvee \end{cases}$$

A is subreducible *if it has a subformula B such that one of the following conditions holds:*

- $\Delta_0(B) \neq$ `nil`.
- $\Delta_1(B) \neq$ `nil`.
- *B is ℓ-reducible for some literal ℓ.*

By Theorem 12 we have that the subreduction preserves meaning, therefore

$$\texttt{SubReduce}(A, \mathcal{M}) = (C, \mathcal{M})$$

where C is obtained by traversing the tree A depth-first in order to find the first subtree B indicated above, and

(1) Apply Theorem 12, if either $\Delta_0(B) \neq \texttt{nil}$ or $\Delta_1(B) \neq \texttt{nil}$.

(2) Substitute B by B_ℓ, otherwise.

The interest of using sub-reductions is that they can make possible further reductions. It is this use of reductions before branching one of the main novelties of this method with respect to others; specifically, the unit clause rule of the Davis-Putnam procedure is a special case of SP reduction; also [**?**] uses a weak version of our sub-reductions in his *dominance lemma*, but he only applies the substitutions to the first level of depth of each subformula.

3.0.9 The `Split` *operator*

The input of `Split` is a pair $(B, \mathcal{M})$ where $B = \bigwedge_{i \in I} B_i$ is a restricted and p-splittable nnf which is neither SP-reducible nor subreducible; we have

$$\texttt{Split}(\bigwedge_{i \in J} B_i, \mathcal{M}) = \left(\Big(\bigwedge_{i \in J_p} B_i[p/\bot], \mathcal{M} \cup \{(p = 0)\} \Big), \Big(\bigwedge_{i \in J_{\overline{p}}} B_i[p/\top], \mathcal{M} \cup \{(p = 1)\} \Big) \right)$$

These two tasks are treated independently by the `Update` process.

Branching: the `QBranch` *operator*

The input of `QBranch` is a pair $(B, \mathcal{M})$ where B is a restricted nnf which is neither SP-reducible, nor splittable, nor sub-reducible, nnf. We have:

$$\texttt{QBranch}(B, \mathcal{M}) = ((B[p/\bot], \mathcal{M} \cup \{(p = 0)\}), (B[p/\top], \mathcal{M} \cup \{(p = 1)\}))$$

These two tasks are treated independently by the `Update` process.

Our experimental tests show that the best results are obtained when choosing p as the propositional variable with more occurrences in the formula being analysed (this information can be easily obtained from the $\widehat{\Delta}$-sets).

Collecting partial results: `CollectInfo`

The `CollectInfo` operator collects the outputs of `Update` for each subproblem generated by either `Parallel`, or `Split`, or `QBranch`, and finishes the execution of the algorithm:

- If all the outputs of the subproblems are $\bot$, then `CollectInfo` ends the algorithm with the output UNSATISFIABLE.
- If some of the subproblems outputs $(\top, \mathcal{M})$, then `CollectInfo` ends the algorithm with output SATISFIABLE and a model, which is built from $\mathcal{M}$.
- If some of the subproblems outputs $(A, \mathcal{M})$ satisfying $\Delta_1(A) \neq$ `nil`, then `CollectInfo` ends the algorithm with output SATISFIABLE and a model is built from $\mathcal{M} \cup \{(\ell = 1)\}$, where ℓ is the first element of $\Delta_1(A)$.

3.1 Soundness and completeness of TAS-D

The termination of the algorithm just described is obvious, for each applied process reduces the size and/or the number of propositional variables of the formula. Specifically, in the worst case, in which no reduction can be applied, the only applicable process is `QBranch` which decreases by one the number of propositional variables in the formula.

Now, we can prove the soundness and completeness of TAS-D.

Theorem 38 *TAS-D*(A)=SATISFIABLE *if and only if* A *is satisfiable.*

PROOF. It suffices to show that all the processes in the algorithm preserve satisfiability. Process `NNF` clearly preserves the meaning, for it is the translation into nnf; all processes in the modules `Update` and `Reduce` preserve either meaning or satisfiability, by the results in Section 2. To finish the proof, one only has to keep in mind that the subproblem generating processes (`Parallel`, `Split`, `QBranch`) are based in the following fact: a disjunction is satisfiable if

and only if a disjunct is satisfiable. So, the process `CollectInfo` preserves satisfiability as well.

3.2 Complexity of TAS

The complexity of each stage of the algorithm is analyzed in this section. We only have to study the complexity of the tests which determine the applicability of each transformation.

- The operator $\Delta-$`restrict` is linear w.r.t. the size of its input formula, for its execution requires at most a double traverse of the formula. In addition, this operator can also reduce the size of the formula, but will never increase it.

- The calculation of the $\widehat{\Delta}$-sets is quadratic, since the determination of the list associated with each node we have to visit its ancestors. We have also to keep in mind that the size of the formula can be reduced, and that the size of $\widehat{\Delta}_0(A) \cup \widehat{\Delta}_1(A)$ is equal to the size of A. This is important because this is the set used in the following stages.

- The stage `SPReduce` is quadratic too, although we have to remark its following features: Firstly, the complete reduction, one of the most powerful reductions in the algorithm, has a linear cost; secondly, in a quadratic process, the set of all Δ-pure literals can be determined, that is, it is not necessary to analyze the variables separately; this remark also applies to the analysis of bounded literals.

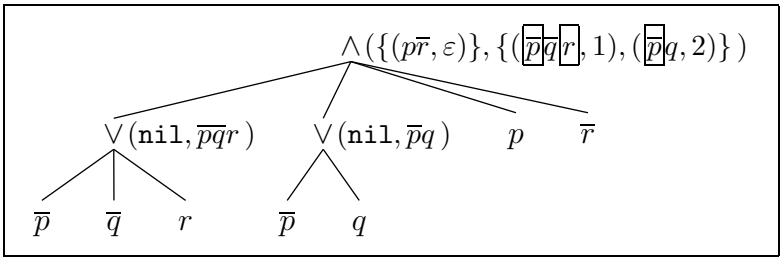
- The modules `SubReduce` and `Split` are also quadratic. For the latter we have to note that, although it requires a quadratic test, this cost does not applies on the size of the tree, but on the lists of the nodes which are immediate successors of the root node.

The computational pay-off of the reductions may seem doubtful, since some time must be spent for scanning the formula and applying the corresponding reduction. It is known that a proof for a formula A of size n is (potentially) of size $O(2^n)$, so if the reduction decreases the size of A at least by 1, then the potential search space would be reduced at least by half. Therefore, we are applying a polynomial processing for an exponential gain.

Example 39 *Consider the formula* $A = (p \to (q \to r)) \to ((p \to q) \to (p \to r))$.

The result of $\texttt{Update}(\texttt{NNF}(\neg A))$ *is* $= (B, \varnothing)$, *where*

$B =$



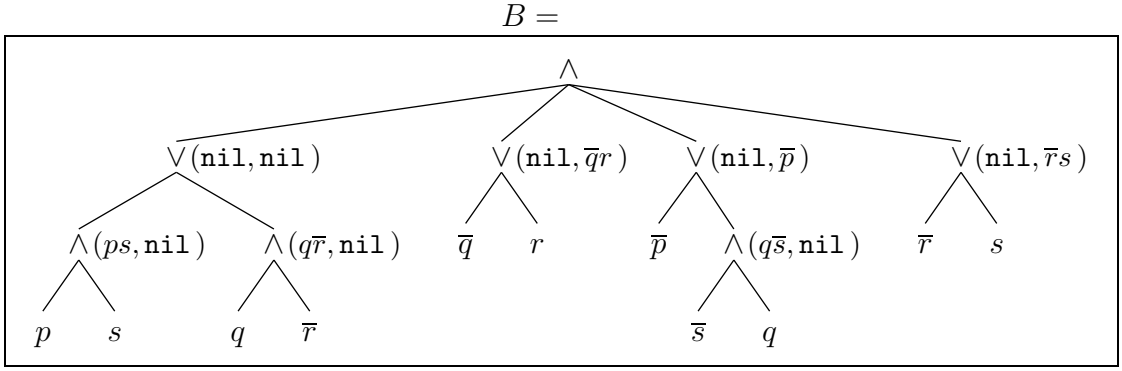
Now, as we have $\widehat{\Delta_0}(B) = \{(p\overline{r}, \varepsilon)\}$, *a complete reduction can be applied w.r.t.* p *and* r*; as a consequence we get* $\texttt{Update}(\texttt{SPReduce}(B, \varnothing)) = (\bot, \{(p = 1), (\overline{r} = 1)\})$, *and then the output is "*$\neg A$ *is* UNSATISFIABLE*", therefore* A *is valid.*

Example 40 *Consider the formula*

$$A = (((p \to \neg s) \to (q \wedge \neg r)) \wedge (\neg q \vee r)) \to ((p \to (\neg s \wedge q)) \to (r \wedge \neg s))$$

we have $\texttt{Update}(\texttt{NNF}(\neg A)) = (B, \varnothing)$, *where*

$B =$



with

$$\widehat{\Delta_0}(B) = \{(ps, 11), (q\overline{r}, 12), (q\overline{s}, 32)\}$$
$$\widehat{\Delta_1}(B) = \{(\overline{q}r, 2), (\overline{p}, 3), (\overline{r}s, 4)\}$$

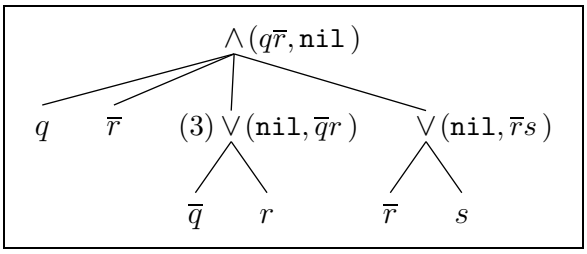
the reduction module does not apply to this tree, that is, $(B, \varnothing)$ *is the input of*

`QBranch`. *We apply* `QBranch` *w.r.t. variable* p *and obtain,*

$$\texttt{QBranch}(B, \varnothing) = ((C_1, \{(p = 0)\}), (C_2, \{(p = 1)\}))$$

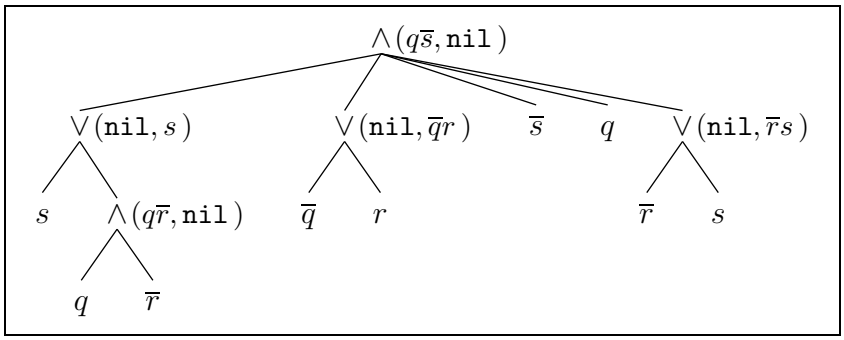
The subproblem C_1 *is studied below:*

After $\Delta-$`restrict` *we get the following tree*

$C_1 =$ 

as $\overline{\Delta_1((3))} = \Delta_0(C_1)$*, then* C_1 *is* Δ_0*-conclusive, and* $\texttt{Update}(C_1) = (\bot, \{(p = 0)\})$.

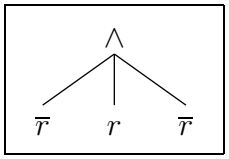
For the second subproblem C_2 *we have*

$C_2 =$ 

for which

$$\widehat{\Delta_0}(C_2) = \{(\boxed{q}\overline{r}, 12), (\overline{qs}, \varepsilon)\}$$
$$\widehat{\Delta_1}(C_2) = \{(\boxed{s}, 1), (\boxed{\overline{q}}r, 2), (\overline{r}\boxed{s}, 5)\}$$

Now, $\widehat{\Delta}-$`restrict`*'s output is fed into* `SPReduce`*; the formula can be completely reduced, for* $(\overline{qs}, \varepsilon) \in \widehat{\Delta_0}(C_2)$*; therefore, by applying substitutions* $[q/\top]$ *and* $[s/\bot]$ *and simplifying the logical constants we get,*

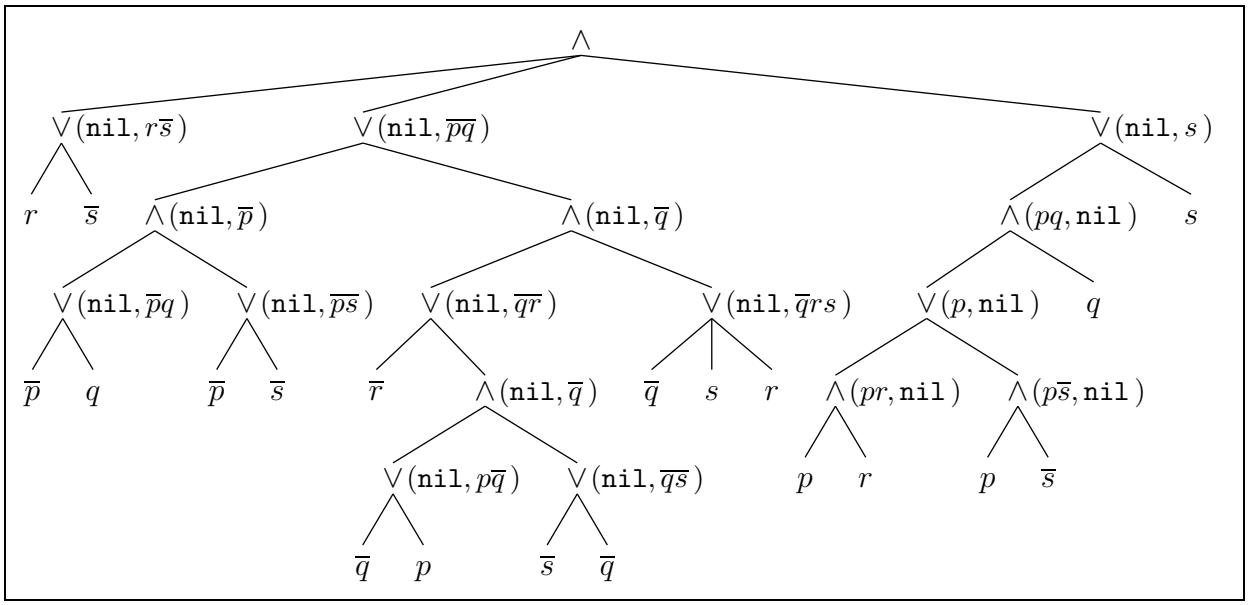


which is Δ_0-conclusive. Therefore, $\texttt{Update}(C_2)) = (\bot, \{(p = 1), (q = 1), (s = 0)\})$.

As all the subproblems generated by `QBranch` *output* $\bot$*, then* `CollectInfo` *produces the output "$\neg A$ is* UNSATISFIABLE*", therefore A is valid.*

Example 41 *Let us study the satisfiability of the formula in Example 30:*

$A =$

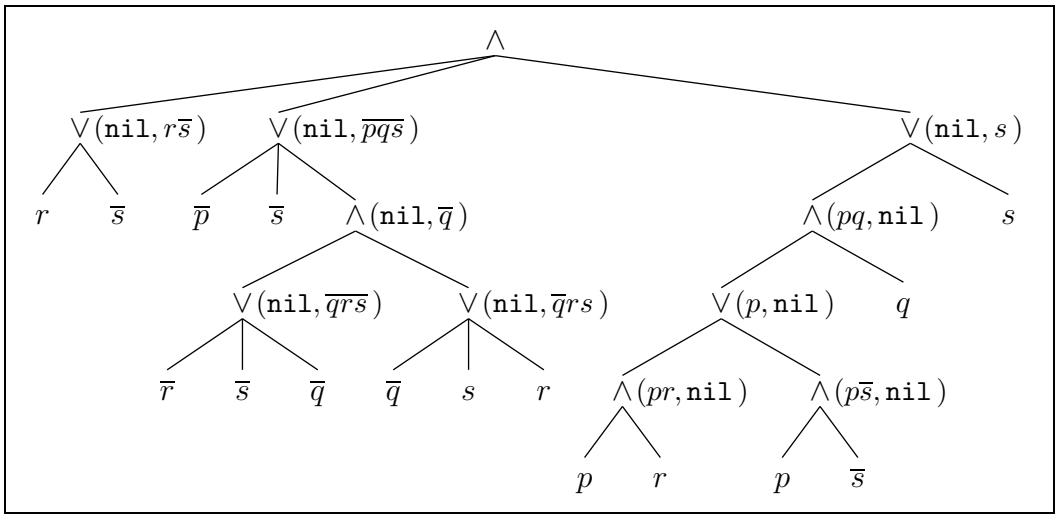


The $\widehat{\Delta_b}$-sets for the previous formula are the following:

$$\widehat{\Delta_0}(A) = \{(\boxed{p}r, 3111), (\boxed{p}\,\boxed{\overline{s}}, 3112), (\boxed{p}, 311), (pq, 31)\}$$
$$\widehat{\Delta_1}(A) = \{(r\overline{s}, 1), (\top, 211), (\boxed{\overline{p}}\overline{s}, 212), (\boxed{\overline{p}}, 21), (\top, 22121), (\boxed{\overline{q}}\overline{s}, 22122),$$
$$(\boxed{\overline{q}}, 2212), (\boxed{\overline{q}}\overline{r}, 221), (\boxed{\overline{q}}rs, 222), (\boxed{\overline{q}}, 22), (\overline{pq}, 2), (s, 3)\}$$

An application of $\widehat{\Delta}-\texttt{restrict}$ *substitutes* (211) *and* (22121) *by* $\top$*, the result is an equivalent formula B:*

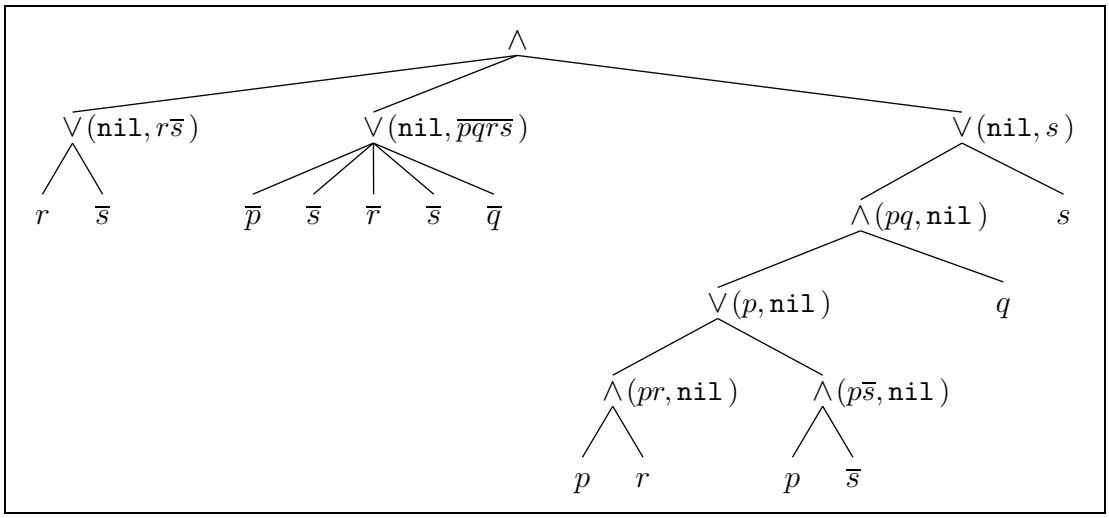
$B =$



$\widehat{\Delta_0}(B) = \{(\boxed{p}r, 3111), (\boxed{p}\,\boxed{\overline{s}}, 3112), (\boxed{p}, 311), (pq, 31)\}$
$\widehat{\Delta_1}(B) = \{(r\overline{s}, 1), (\boxed{\overline{q}}\,\boxed{r}\,\boxed{\overline{s}}, 231), (\top, 232), (\boxed{\overline{q}}, 23), (\overline{pqs}, 2), (s, 3)\}$

Once again, the $\top$ *in* $\widehat{\Delta_1}(B)$ *allows to substitute* (232) *by* $\top$*, obtaining the equivalent formula* C*:*

$C =$

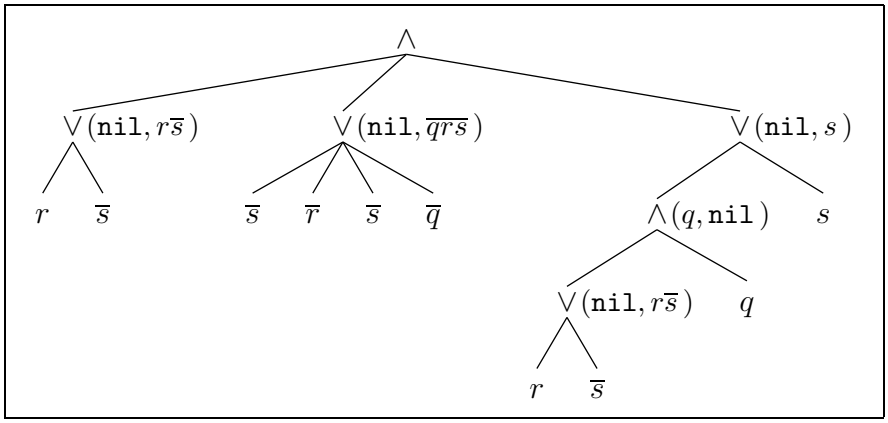


$\widehat{\Delta_0}(C) = \{(\boxed{p}r, 3111), (\boxed{p}\,\boxed{\overline{s}}, 3112), (\boxed{p}, 311), (pq, 31)\}$
$\widehat{\Delta_1}(C) = \{(r\overline{s}, 1), (\overline{pqrs}, 2), (s, 3)\}$

Therefore, $\texttt{Update}(A, \varnothing) = (C, \varnothing)$*. Now* `SPReduce` *can be applied, for literals* p *and* q *are 0-1-bound, we substitute all the occurrences of* p *by* $\top$*,*

i.e. $\Delta{-}\texttt{restrict}(\texttt{SPReduce}(C, \varnothing)) = (D, \{(p = 1)\})$, *where*

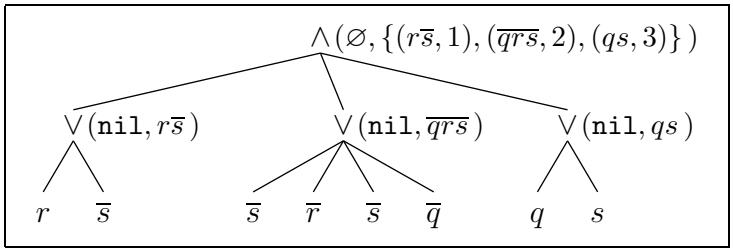
$D =$



$\widehat{\Delta_0}(D) = \{(q, 31)\}$
$\widehat{\Delta_1}(D) = \{(r\overline{s}, 1), (\overline{qrs}, 2), (\top, 311), (s, 3)\}$

After substituting (311) *by* $\top$ *we get :*

$E =$



In this formula q *is 1-bounded to* s *and,* `SPReduce` *substitutes branches at addresses 2 and 3 by* $\top$*; then*

$$\Delta{-}\texttt{restrict}(\texttt{SPReduce}(E, \{(p = 1)\})) = (r \vee \overline{s}, \{(p = 1), (q = \overline{s})\})$$

As $r \vee \overline{s}$ *is finalizable, for its* $\Delta_1 \neq \texttt{nil}$*, the stage* `CollectInfo` *ends the algorithm with output "A is* SATISFIABLE*" and the model determined by* $\{(p = 1), (q = \overline{s}), (r = 1)\}$*, that is, any interpretation* I *such that* $I(p) = I(q) = I(r) = 1$ *and* $I(s) = 0$ *is a model of* A*. Note that* I' *defined as* $I'(p) = I'(r) = I'(s) = 1$ *and* $I'(q) = 0$ *is also a model of* A*.*

4 A comparative example

To put our method in connection with other existent approaches in the literature, we will study the collection $\{T_n\}$ of clausal forms taken from [**?**], we also use their notation for the propositional variables. Consider, for instance, T_3 below:

$$p^1 \vee p^2_+ \vee p^3_{++}$$

$$p^1 \vee p^2_+ \vee \overline{p^3_{++}}$$

$$p^1 \vee \overline{p^2_+} \vee p^3_{+-}$$

$$p^1 \vee \overline{p^2_+} \vee \overline{p^3_{+-}}$$

$$\overline{p^1} \vee p^2_- \vee p^3_{-+}$$

$$\overline{p^1} \vee p^2_- \vee \overline{p^3_{-+}}$$

$$\overline{p^1} \vee \overline{p^2_-} \vee p^3_{--}$$

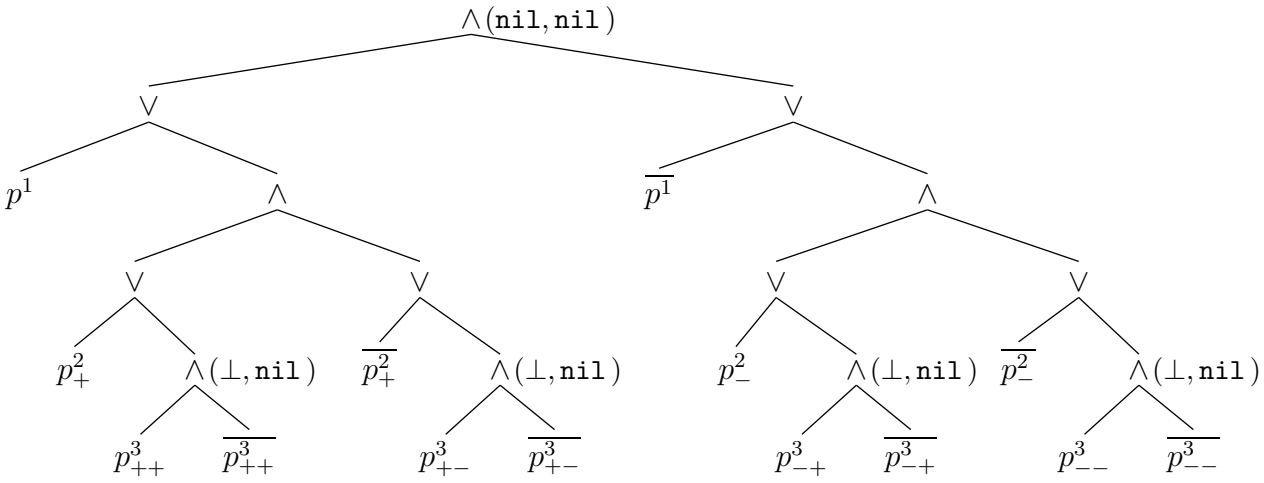
$$\overline{p^1} \vee \overline{p^2_-} \vee \overline{p^3_{--}}$$

each clause contains atoms of the form $p^i_\sharp$, where $\sharp$ is a string of $+$'s and $-$'s. The superscripts in each clause always form the sequence 1,2, ..., n. The subscript of each literal is exactly the sequence of signs of the preceding literals in its clause. When T_n is built from T_{n-1}, each $p^n_\sharp$ is added both positively and negatively. It is easy to see that T_n has $2^n - 1$ distinct propositional variables, 2^n clauses, each of which contains n literals.

In [**?**], Cook and Reckhow described the family $\{T_n, n \geq 1\}$ and showed that the study of its satisfiability is intractable for analytic tableaux but can be handled in linear time by resolution. In [**?**], Murray and Rosenthal showed that dissolution with factoring provides proofs for this class that are linear in the number of input clauses, $|T_n|$.

When we apply TAS-D to test the satisfiability of T_n we get that it is subreducible. For instance, formula $\texttt{Reduce}(T_3)$ can be expressed equivalently as

the formula



Thus, $\Delta-$`restrict` reduces the previous tree, for there are four Δ_0-conclusive subtrees (namely, the conjunctions $p^3_\sharp \wedge \overline{p^3_\sharp}$). When simplifying the four $\bot$ leaves, we get $\bot$. Therefore, when using TAS-D we can detect the unsatisfiability of the formulas T_n with *no branching* at all.

5 Conclusions

We have presented a non-clausal satisfiability tester, named TAS-D, for Classical Propositional Logic. The main novelty of the method, as opposed to other approaches, is that the reductions applied on each formula are dynamically selected, and applied to subformulas like in a rewrite system, following syntax-directed criteria. Specifically, we have introduced extensions of the pure literal rule and of the collapsibility theorems. This fact increases the efficiency, for it decreases branching.

As an example of the power of TAS-D we have studied a class of formulas which has linear proofs (in the number of branchings) when either resolution or dissolution with factoring is applied; on the other hand, when applying our method to these formulas we get proofs without branching.

Acknowledgements

The authors would like to thank José Meseguer and Daniele Mundici and the anonymous referees for their valuable comments on earlier versions of this work.