Restricted Δ-trees and Reduction Theorems in Multiple-Valued Logics*

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

Dept. Matemática Aplicada
Universidad de Málaga
{aciego,guzman,a_valverde}@ctima.uma.es

Abstract. In this paper we continue the theoretical study of the possible applications of the Δ-tree data structure for multiple-valued logics, specifically, to be applied to signed propositional formulas. The Δ-trees allow a compact representation for signed formulas as well as for a number of reduction strategies in order to consider only those occurrences of literals which are relevant for the satisfiability of the input formula. New and improved versions of reduction theorems for finite-valued propositional logics are introduced, and a satisfiability algorithm is provided which further generalise the TAS method [1, 5].

1 Introduction

Automated deduction in multiple-valued logic has been based on the notions of *sign* and *signed formula*, which allow one to apply classical methods in the analysis of multiple-valued logics. The main idea is to apply the following bivalued metalinguistic interpretation of multiple-valued sentences: For example, in a 3-valued logic with truth-values $\{0, 1/2, 1\}$ and with $\{1\}$ as the designated value, the satisfiability of a formula φ can be expressed as: *Is it possible to evaluate φ in $\{1\}$?* In the same way, the unsatisfiability of φ is expressed by: *Is it possible to evaluate φ in $\{0, 1/2\}$?* These questions correspond to the study of validity of the signed formulas ${}_{\{1\}}{:}\varphi$ and ${}_{\{0,1/2\}}{:}\varphi$, which are evaluated on the set $\{0, 1\}$. In other words, the formulas in a signed logic are constructions of the form $S{:}\varphi$, where S is a set of truth-values of the multiple-valued logic, called the *sign*, and φ is a formula of that logic.

Although there are interesting works on automated deduction for infinitely-valued logics [2, 8], we will only be concerned with n-valued logics. The reason for focussing only on the finitely-valued case is that "fuzzy" truth-values (or human preferences) are usually described in a granulated way, by steps in the degree of perception. This is connected to the well-known fact that people can only distinguish finitely many degrees of quality (closeness, cheapness, ...) or quantity in control.

The first works to provide a systematic treatment of sets of truth-values as signs were due to Hähnle in [7] and Murray and Rosenthal in [9]. There, the notion of *signed formula* is formally introduced. In [7] these tools are used in the framework of truth

* Research partially supported by Spanish DGI project BFM2000-1054-C02-02 and Junta de Andalucía project TIC-115.

tables, while in [9] they are used to develop another nonclausal proof method *dissolution*. As a result of these works, the use of signed formulas in the field of automated deduction has been extended, and has led to significant advances in this method; therefore, efficient representations for signed formulas are necessary in order to describe and implement efficient algorithms on this kind of formulas.

An approach to the efficient handling of signed formulas that one finds in the literature is the clause form, which allow the extension of classical techniques such as resolution, or Davis-Putnam procedures. Another approach is that of Multiple-Valued Decision Diagrams (MDDs) and its variants [3], but they are not useful for the study of satisfiability because, although they make straightforward its testing, the construction of a restricted MDD for a given formula requires exponential space in the worst case.

Our approach to automated deduction for signed logics follows the TAS methodology [1, 5], that is, the application of as much reduction theorems with low complexity as possible before applying a branching rule. The main aim of the paper is of theoretical nature, to provide a TAS-based satisfiability algorithm for signed formulas.

To work with signed formulas, we will follow the approach introduced in [4, 6], interpreting signed formulas by means of Δ-trees, that is, trees of clauses and cubes. In this paper, we will be concerned with the metatheory of multiple-valued Δ-trees, not with implementation issues.

2 Reduced signed logics and multiple-valued Δ-trees

The notion of *reduced signed logic* is a generalisation of previous approaches, and it is developed in a general propositional framework without reference either to an initially given multiple-valued logic or to a specific algorithm, ie. the definition is completely independent of the particular application at hand. The generalisation consists in introducing a *possible truth values function*, denoted ω, to restrict the truth values for each variable. These restrictions can be motivated by the specific application and they can be managed dynamically by the algorithms. For example, in [10] are used to characterize non-monotonic reasoning systems.

The formulas in the reduced signed logic $\mathbf{S}_\omega$, the *signed logic valued in* $\mathbf{n}$ *by* ω, are built by using the connectives $\wedge$ and $\vee$ on ω*-signed literals* (or simply, literals): if $\mathbf{n} = \{1, \ldots, n\}$ is a finite set of truth-values, $\mathcal{V}$ is the set of propositional variables and $\omega\colon \mathcal{V} \to (2^{\mathbf{n}} \setminus \varnothing)$ is a mapping, called the *possible truth-values function*, then the set of ω*-signed literals* is $\textsc{lit}_\omega = \{S{:}p \mid S \subseteq \omega(p), p \in \mathcal{V}\} \cup \{\bot, \top\}$.

In a literal $\ell = S{:}p$, the set S is called the *sign of* ℓ and p is the *variable of* ℓ. The complement of a signed literal $S{:}p$ is $(\omega(p) \setminus S){:}p$ and will be denoted $\overline{S{:}p}$.

The semantics of $\mathbf{S}_\omega$ is defined using the ω*-assignments*. The ω-assignments are mappings from the language into the set $\{0, 1\}$ that interpret $\vee$ as maximum, $\wedge$ as minimum, $\bot$ as falsity, $\top$ as truth and, in addition, satisfy:

1. For every p there exists a unique $j \in S$ such that $I(\{j\}{:}p) = 1$
2. $I(S{:}p) = 1$ if and only if there exists $j \in S$ such that $I(\{j\}{:}p) = 1$

These conditions arise from the objective for which signed logics were created: the ω-assignment I over $S{:}p$ is 1 if the variable p is assigned a value in S; this value must be

unique for every multiple-valued assignment and thus unique for every ω-assignment. This is why we sometimes will write $I(\{j\}{:}p) = 1$ as $I(p) = j$.

An important operation in the sequel will be the *reduction* of a signed logic. This operation decreases the possible truth-values set for one or more propositional variables. The reduction will be forced during the application of an algorithm but it can also help us to specify a problem using signed formulas. Specifically, we will use two basic reductions: to prohibit a specific value for a given variable, $[p \neq j]$, and to force a specific value for a given variable, $[p = j]$: If ω is a possible truth-values function, then the possible truth-values functions $\omega[p \neq j]$ and $\omega[p = j]$ are defined as follows:

$$\omega[p \neq j](v) = \begin{cases} \omega(p) \smallsetminus \{j\} & \text{if } v = p \\ \omega(v) & \text{otherwise} \end{cases} \qquad \omega[p = j](v) = \begin{cases} \{j\} & \text{if } v = p \\ \omega(v) & \text{otherwise} \end{cases}$$

If A is a formula in $\mathbf{S}_\omega$, we define the following substitutions:

- $A[p \neq j]$ is a formula in $\mathbf{S}_{\omega[p \neq j]}$ obtained from A by replacing $\{j\}{:}p$ by $\bot$, $\overline{\{j\}{:}p}$ by $\top$, $S{:}p$ by $(S \smallsetminus \{j\}){:}p$ and, in addition, the constants are deleted using the 0-1-laws.
- $A[p = j]$ is a formula in $\mathbf{S}_{\omega[p=j]}$ obtained from A by replacing every literal $S{:}p$ satisfying $j \in S$ by $\top$ and every literal $S{:}p$ satisfying $j \notin S$ by $\bot$; in addition, the constants are deleted using the 0-1-laws.

An immediate consequence is the following: if I is a model of A in $\mathbf{S}_\omega$ and $I(p) \neq j$, then (the restriction of) I is also a model of $A[p \neq j]$ in $\mathbf{S}_{\omega[p \neq j]}$; if I is a model of A in $\mathbf{S}_\omega$ and $I(p) = j$, then I is a model of $A[p = j]$ in $\mathbf{S}_{\omega[p=j]}$.

Throughout the rest of the paper, we will use the following standard definitions. A signed formula A in $\mathbf{S}_\omega$ is said to be *satisfiable* if there is an ω-assignment I such that $I(A) = 1$; in this case I is said to be a *model* for A. Two signed formulas A and B are said to be *equisatisfiable*, denoted $A \approx B$, 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 ω-assignment I. We will also use the usual notions of clause (disjunction of literals) and cube (conjunction of literals). Given a set of formulas Ω, the notation $\Omega \models A$ means that all models for Ω are also models for A. A literal ℓ is an *implicant* of a formula A if $\ell \models A$. A literal ℓ is an *implicate* of a formula A if $A \models \ell$.

Multiple-valued Δ-trees The satisfiability algorithm we will describe is based on the structure of multiple-valued Δ-trees. In the classical case, nodes in the Δ-trees correspond to lists of literals; in the multiple-valued case we will exploit a duality in the representation of signed literals in terms of literals whose sign is a singleton. To better understand this duality, let us consider the literal $\{1,4\}{:}p$ in the signed logic $\mathbf{S}_\omega$ where $\omega(p) = \{1, 2, 4, 5\}$, then: $\{1,4\}{:}p \equiv \{1\}{:}p \vee \{4\}{:}p$ and $\{1,4\}{:}p \equiv \overline{\{2\}{:}p} \wedge \overline{\{5\}{:}p}$. This way, we have both a disjunctive and a conjunctive representation of signed literals using the literals $\{j\}{:}p$ and $\overline{\{j\}{:}p}$, which are called *basic literals*. In the sequel, we will use a simpler representation for these literals: $pj =_{def} \{j\}{:}p$ and $\overline{pj} =_{def} \overline{\{j\}{:}p}$.

The basic literals pj are the *positive literals* and their complements, $\overline{pj}$, are the *negative literals*. In the Δ-tree representation we work with lists of positive literals.

Definition 1.

1. *A list/set of positive literals, λ, is* saturated *for the variable p if $pj \in \lambda$ for all $j \in \omega(p)$. (This kind of lists/sets will be interpreted as logical constants.)*
2. *A Δ-*list *is either the symbol $\sharp$ or a list of positive literals such that it does not have repeated literals and it is non-saturated for any propositional variable.*
3. *A Δ-*tree *T is a tree with labels in the set of Δ-lists.*

In order to define the operator `sgf` which interprets a Δ-tree as a signed formula, we should keep in mind that:

1. The empty list, `nil`, has different conjunctive and disjunctive interpretations, since it is well-known the identification of the empty clause with $\bot$ and the empty cube with $\top$; but anyway it corresponds to the neutral element for the corresponding interpretation. Similarly, we will use a unique symbol, $\sharp$, to represent the absorbent elements, $\bot$ and $\top$, under conjunctive and disjunctive interpretation, respectively.
2. A Δ-tree will always represent a conjunctive signed formula, however, its subtrees are alternatively interpreted as either conjunctive or disjunctive signed formulas, i.e. the immediate subtrees of a conjunctive Δ-tree are disjunctive, and vice versa.

Definition 2. *The operator* `sgf` *over the set of Δ-trees is defined as follows:*

1. $\mathtt{sgf}(\mathtt{nil}) = \top$, $\mathtt{sgf}(\sharp) = \bot$, $\mathtt{sgf}(\ell_1 \dots \ell_n) = \overline{\ell_1} \wedge \dots \wedge \overline{\ell_n}$
2. $\mathtt{sgf}\left(\begin{array}{c}\lambda\\ \diagup\quad\diagdown\\ T_1\ \dots\ T_m\end{array}\right) = \mathtt{sgf}(\lambda) \wedge \mathtt{dsgf}(T_1) \wedge \dots \wedge \mathtt{dsgf}(T_m)$

where the auxiliary operator `dsgf` *is defined as follow:*

1. $\mathtt{dsgf}(\mathtt{nil}) = \bot$, $\mathtt{dsgf}(\sharp) = \top$, $\mathtt{dsgf}(\ell_1 \dots \ell_n) = \ell_1 \vee \dots \vee \ell_n$
2. $\mathtt{dsgf}\left(\begin{array}{c}\lambda\\ \diagup\quad\diagdown\\ T_1\ \dots\ T_m\end{array}\right) = \mathtt{dsgf}(\lambda) \vee \mathtt{sgf}(T_1) \vee \dots \vee \mathtt{sgf}(T_m)$

In short, we will write $\hat{T} = \mathtt{sgf}(T)$ and $\overset{\vee}{T} = \mathtt{dsgf}(T)$; in particular, if $T = \lambda = \ell_1 \dots \ell_n$ we have: $\overset{\wedge}{\lambda} = \ell_1 \wedge \dots \wedge \ell_n$ and $\overset{\vee}{\lambda} = \ell_1 \vee \dots \vee \ell_n$.

An important feature of the structure of Δ-tree is that it gives us a means to calculate implicants and implicates, to be used in the reduction transformations below.

Proposition 1. *If T is rooted with λ and $pj \in \lambda$, then:*

$$\mathtt{sgf}(T) \models \overline{pj} \qquad \textit{and} \qquad pj \models \mathtt{dsgf}(T)$$

The notions of validity, satisfiability, equivalence, equisatisfiability or model are defined on Δ-trees by means of the `sgf` operator; for example, a Δ-tree, T is satisfiable if and only if $\mathtt{sgf}(T)$ is satisfiable and the models of T are the models of $\mathtt{sgf}(T)$.

In [6] we formally introduced operators to define the converse translation: specifically, operators `c`Δ`List`, `d`Δ`List` and Δ`Tree` are defined. The first two are auxiliary operators (the inverse of the base cases of `sgf` and `dsgf`) and Δ`Tree` constructs the Δ-tree associated to a general signed formula.

Example 1. In the logic $\mathbf{S}_\omega$ with $\omega(p) = \{1,2,4,5\}$, $\omega(q) = \{1,2,3\}$, $\omega(r) = \{2,5\}$.

- $\mathtt{d}\Delta\mathtt{List}(\{1,4\}{:}p \vee \{1,2\}{:}q) = p1\, p4\, q1\, q2$
- $\mathtt{c}\Delta\mathtt{List}(\{1,4\}{:}p \wedge \{1,2\}{:}q) = p2\, p5\, q3$
- $\mathtt{d}\Delta\mathtt{List}(\{1,4\}{:}p \vee \{2\}{:}r \vee \{2,4,5\}{:}p) = \sharp$, for $\{p1, p2, p4, p5, r2\}$ is saturated for p.
- $\mathtt{c}\Delta\mathtt{List}(\{1\}{:}q \wedge \{1,2,4\}{:}p \wedge \{2\}{:}q) = \sharp$, for $\{p5, q1, q2, q3\}$ is saturated for q.

Recall that, as established in [6], a Δ-tree will always be interpreted as a conjunctive signed formula and arbitrary signed formulas are represented by means of lists of Δ-trees;[1] this way, the study of satisfiability can be performed in parallel.

Example 2. The following examples are from $\mathbf{S}_3$, where 3 denotes the constant mapping defined as $3(p) = \mathbf{3}$ for all p.

$$\Delta\mathtt{Tree}\big((\{1,2\}{:}p \vee \{2\}{:}q) \wedge (\{2,3\}{:}p \vee \{1,3\}{:}r)\big) = \left[\begin{array}{c} \mathtt{nil} \\ \hline p1p2q2 \quad p2p3r1r3 \end{array}\right]$$

$$\Delta\mathtt{Tree}\big(\{2,3\}{:}q \vee (\{1,2\}{:}p \wedge (\{1,2\}{:}q \vee \{2,3\}{:}p) \wedge (\{3\}{:}q \vee \{1\}{:}p))\big) = \left[q1,\ \begin{array}{c} p3 \\ \hline p2p3q1q2 \quad p1q3 \end{array}\right]$$

It is interesting to recall the intrinsic parallelism between the usual representation of cnfs as lists of clauses and our representation of signed formulas as lists of Δ-trees.

3 Restricted Δ-trees

In multiple-valued logic there is not a notion which captures the well-known definition of restricted clauses of classical logic, in which complementary literals and logical constants are not allowed. We can say that restricted Δ-trees are Δ-trees without *trivially* redundant information. In this section we give a suitable generalisation built on the notion of restricted multiple-valued Δ-tree which is built from its classical counterpart [4].

Definition 3. *The operators* $\mathtt{Uni}$ *and* $\mathtt{Int}$ *are defined on the set of Δ-lists as follows. If $\lambda_1, \ldots, \lambda_n$ are Δ-lists then:*

1. $\mathtt{Uni}(\lambda_1, \ldots, \lambda_n) = \sharp$ *if either there exists i such that $\lambda_i = \sharp$ or $\bigcup_{i=1}^n \lambda_i$ is saturated for some variable p. Otherwise,* $\mathtt{Uni}(\lambda_1, \ldots, \lambda_n) = \bigcup_{i=1}^n \lambda_i$.
2. $\mathtt{Int}(\lambda_1, \ldots, \lambda_n) = \sharp$ *if $\lambda_i = \sharp$ for all i.*
   Otherwise, $\mathtt{Int}(\lambda_1, \ldots, \lambda_n) = \bigcap_{\lambda_i \neq \sharp} \lambda_i$.

The following definition gathers the specific situations that will not be allowed in a restricted form: nodes in the Δ-tree which, in some sense, can be substituted by either $\bot$ or $\top$ without affecting the meaning, and also leaves with only one propositional variable; in addition, our restricted trees must have explicitly the implicants and implicates of every subtree in order to perform the reductions based in these objects (see [5]).

[1] To help the reading, we will write these lists with the elements separated by commas and using square brackets as delimiters. This way, for example, $p_1 j_1 \ldots p_n j_n$ is a Δ-list, and $[p_1 j_1, \ldots, p_n j_n]$ is a list of Δ-trees (in which each Δ-tree is a leaf, which turns out to be a singleton Δ-list).

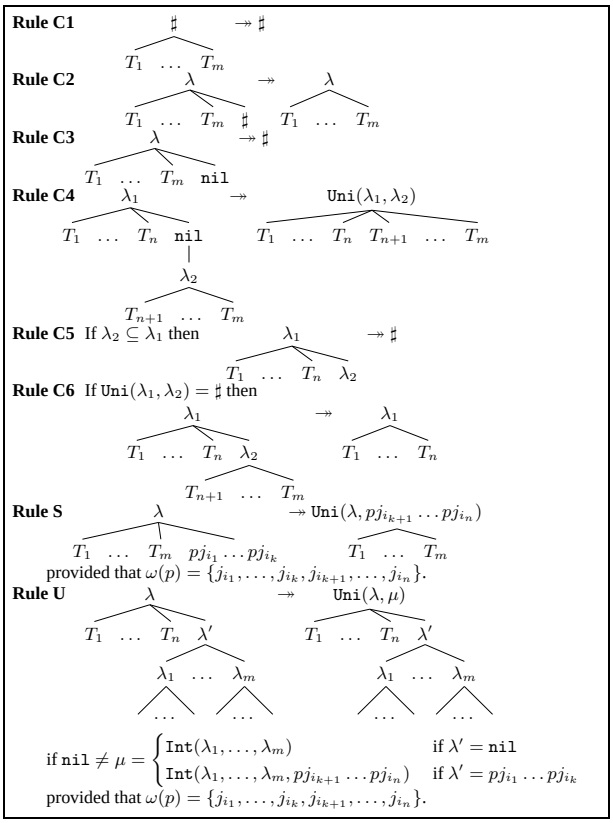


Fig. 1. Rewriting rules to obtain the restricted form

Definition 4. *Let T be a Δ-tree.*

1. *A node of T is said to be* conclusive *if it satisfies any of the following conditions:*
   - *It is labelled with $\sharp$, provided that $T \neq \sharp$.*
   - *It is either a leaf or a monary node labelled with* `nil`*, provided that it is not the root node.*
   - *It is labelled with λ, it has an immediate successor λ' which is a leaf and $\lambda' \subseteq \lambda$.*
   - *It is labelled with λ and $\mathtt{Uni}(\lambda, \lambda') = \sharp$, where λ' is the label of its predecessor.*
2. *A leaf in T is said to be* simple *if the literals in its label share a common propositional variable.*
3. *Let λ be the label of a node of T; let λ' be the label of one immediate successor of λ and let $\lambda_1, \ldots, \lambda_n$ be the labels of the immediate successors of λ'. We say that λ* can be updated *if it satisfies some of the following conditions:*
   - *$\lambda' = \mathtt{nil}$ and $\mathtt{Int}(\lambda_1, \ldots, \lambda_m) \not\subset \lambda$.*

– $\lambda' = pj_{i_1} \dots pj_{i_k}$ *and* $\mathtt{Int}(\lambda_1, \dots, \lambda_m, pj_{i_{k+1}} \dots pj_{i_n}) \not\subset \lambda$*, provided that* $\omega(p) = \{j_{i_1}, \dots, j_{i_k}, j_{i_{k+1}}, \dots, j_{i_n}\}$.

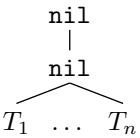
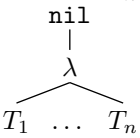
We say that T *is* updated *if it has no nodes that can be updated.*

4. *If* T *is updated and it has neither conclusive nodes nor simple leaves, then it is said to be* restricted*.*

The rewriting rules in Fig. 1 (up to the order of the successors) allow to delete the conclusive nodes and simple leaves of a Δ-tree and in addition, to update the updatable nodes. Note that the rewriting rules have a double meaning; since they need not apply to the root node, the interpretation can be either conjunctive or disjunctive. This is just another efficiency-related feature of Δ-trees: duality of connectives $\wedge$ and $\vee$ gets subsumed in the structure and it is not necessary to determine the conjunctive/disjunctive behaviour to decide the transformation to be applied.

Theorem 1. *If* T *is a* Δ*-tree, there exists a list of restricted* Δ*-trees,* $[T_1, \dots, T_n]$*, such that* $\mathtt{sgf}(T) \equiv \hat{T}_1 \vee \dots \vee \hat{T}_n$.

The proof of the theorem allows to specify a procedure to obtain $[T_1, \dots, T_n]$. Let T' be the Δ-tree obtained from T by exhaustively applying the rules C1, C2, C3, C4, C5, C6, S, and U till none of them can be applied any more, then the list of restricted Δ-trees $[T_1, \dots, T_n]$, denoted by $\mathtt{Restrict}(T)$, is defined as:[2]

1. If $T' =$  then $\mathtt{Restrict}(T) = [T_1, \dots, T_n]$
2. If $T' =$  , and $\mathtt{dsgf}(\lambda) = S_1{:}p_1 \vee \dots \vee S_k{:}p_k$ with $p_i \neq p_j$ for every $i \neq j$, then $\mathtt{Restrict}(T) = [\mathtt{c}\Delta\mathtt{List}(S_1{:}p_1), \dots, \mathtt{c}\Delta\mathtt{List}(S_k{:}p_k), T_1, \dots, T_n]$
3. Otherwise, $\mathtt{Restrict}(T) = [T']$.

4 Reduction of Δ-trees

In this section we introduce the reduction theorems used by the TAS algorithm to be given later, which motto is to apply as much reductions with low complexity as possible before applying a branching rule.

In the statements of the reductions we will use the substitutions $[p = j]$ and $[p \neq j]$, defined on Δ-trees as follows:

Definition 5. *Let* T *be a* Δ*-tree.*

[2] These patterns correspond to the elimination of a conclusive node at the root, which cannot be deleted by rule C4.

1. $[p \neq j]T$ *is the* Δ*-tree in* $\mathbf{S}_{w[p \neq j]}$ *obtained from* T *deleting every occurrence of* pj *in* T *and, in addition, if a node is saturated for some variable, it is substituted by* $\sharp$*.*
2. $[p = j]T$ *is the* Δ*-tree in* $\mathbf{S}_{w[p=j]}$ *obtained from* T *by the applications of the following transformations:*
   (a) If pj *is in the root of* T*, then* $[p = j]T = \sharp$ *(that is,* $\mathtt{sgf}([p = j]T) = \bot$*).*
   (b) Otherwise, every subtree rooted with a list λ *such that* $pj \in \lambda$ *is deleted and any occurrence of a literal* pj' *with* $j \neq j'$ *is also deleted.*

   In addition, if a node is saturated for some variable, it is substituted by $\sharp$*.*

Obviously, these operations on Δ-trees are the same to those on signed formulas:

Lemma 1. *If* T *is a* Δ*-tree, then:*
$\mathtt{sgf}([p = j]T) \equiv \mathtt{sgf}(T)[p = j], \mathtt{sgf}([p \neq j]T) \equiv \mathtt{sgf}(T)[p \neq j]$.

The main result to prove the soundness of the reductions on signed formulas is given below. The theorem allows to drive literals downwards to force either contradictions or tautologies, which can be deleted. In the subsequent corollary we apply the theorem to delete several occurrences of literals; this result is the theoretical support of both the *subreduction* and the *complete reduction*.

Theorem 2. *Let* A *be a signed formula and* η *a subformula of* A*.*

1. *If* $A \models \overline{pj}$*, then* $A \equiv \overline{pj} \wedge A[\eta/\eta \wedge \overline{pj}]$.
2. *If* $pj \models A$*, then* $A \equiv pj \vee A[\eta/\eta \vee pj]$.

Corollary 1. *Let* A *be a signed formula.*

1. *If* $A \models \overline{pj}$*, then* $A \equiv \overline{pj} \wedge A[p \neq j]$*, and also* $A \approx A[p \neq j]$.
2. *If* $pj \models A$*, then* $A \equiv pj \vee A[p = j]$.

The Δ-tree representation is very adequate to apply these properties, because the basic literals in the nodes are either implicants or implicates of the corresponding subformula, as stated in Proposition 1. All the transformations performed by operator `Restrict` use just the information of a node and its immediate successors. The next transformation uses "ascending" information, in that nodes are simplified according to information from its ascendants.

Definition 6 (Subreduction). *Let* T *be a restricted* Δ*-tree.* $\mathtt{SubRed}(T)$ *is the* Δ*-tree obtained form* T *performing the following transformations in a depth-first traverse:*

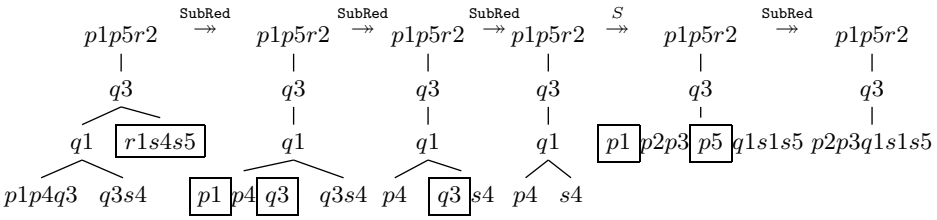
1. *If the union of the ascendant nodes of* η *(including* η *itself) is saturated for a variable* p*, then the subtree rooted at* η *is deleted.*
2. *Otherwise, in a node labelled with* λ *we delete a literal* $pj \in \lambda$ *if* pj *occurs in some proper ascendant of the node.*

Theorem 3. *Let* T *be a* Δ*-tree, then* $\mathtt{SubRed}(T) \equiv T$.

The following proposition, which follows easily from the definition of subreduction, states that only the dominant occurrences of literals are present in a subreduced Δ-tree.

Proposition 2. *Let T be a Δ-tree. In every branch of* $\mathtt{SubRed}(T)$ *there is at most one occurrence of each propositional variable. In particular, if ℓ is a literal in* $\mathtt{SubRed}(T)$*, then there is no occurrence of ℓ under ℓ.*

Example 3. We are going to apply the SubRed operator to the following Δ-tree in $\mathbf{S}_\omega$ with $\omega(p) = \mathbf{5}$, $\omega(q) = \{1,3,5\}$, $\omega(r) = \{1,2\}$, $\omega(s) = \{1,4,5\}$.

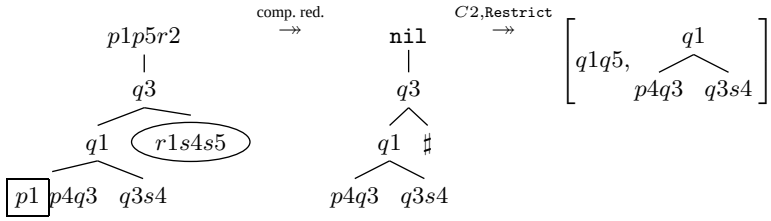


Now we introduce a satisfiability-preserving transformation which, essentially, is a refinement of the subreduction of the Δ-list of the root. Theorem 4 is the Δ-tree formulation of Corollary 1, item 1 (2nd assertion).

Definition 7. *A Δ-tree with non-empty root is said to be* completely reducible.

Theorem 4 (Complete reduction). *If $\lambda \neq \mathtt{nil}$ is the root of T and $pj \in \lambda$, then $T \approx [p \neq j]T$. If I is a model of $[p \neq j]T$ in $\mathbf{S}_{w[p\neq j]}$, then I is a model of T in $\mathbf{S}_\omega$.*

Example 4. Let us consider the initial Δ-tree, T, in Example 3 with the same signed logic. The Δ-tree is completely reducible and thus it is satisfiable iff $[p \neq 1, p \neq 5, r \neq 2]T$ is satisfiable in $\mathbf{S}_{\omega'}$ with $\omega'(p) = \{2,3,4\}$, $\omega'(q) = \{1,3,5\}$, $\omega'(r) = \{1\}$, $\omega'(s) = \{1,4,5\}$. (In fact, it is satisfiable, because the first element in the list is a clause, and $I(q) = 3$ is a model for it).



The TAS Algorithm for Signed Logics: One cannot hope that the reduction strategies are enough to prove the satisfiability of any signed formula. This is only possible after adding a suitable branching strategy, which is based on the Davis-Putnam procedure.

The algorithm handles a list of pairs $[(T_1, \omega_1), \ldots, (T_m, \omega_m)]$, called the *flow*, where the T_i are Δ-trees and ω_i are possible truth values functions. For the satisfiability of a formula A, we set $[T_1, \ldots, T_n] = \Delta\mathtt{Tree}(A)$ and, in the initial list $\omega_i = \mathbf{n}$ for all i.

Given the flow in some instant during the execution of the algorithm, the initial Δ-tree is unsatisfiable iff every T_i is unsatisfiable in $\mathbf{S}_{\omega_i}$, that is $T_i = \sharp$ for all i.[3]

[3] The actual search done by the algorithm is to obtain an element $(\mathtt{nil}, \omega)$ in the list of tasks. In this case, the input formula is satisfiable by any assignment in $\mathbf{S}_\omega$.

1. UPDATING: On the initial list, and after each reduction, the Δ-trees are converted to restricted form.
2. COMPLETE REDUCTION: If some of the elements of the list of tasks is completely reducible, then the transformation is applied and the corresponding logic is reduced.
3. SUBREDUCTION: If no task is completely reducible, then the subreduction transformation is applied.
4. BRANCHING: Finally, if no transformation applies to the list of tasks, then a random task is chosen together with a literal pj to branch on, as follows:

$$[\ldots,(T,\omega),\ldots] \twoheadrightarrow [\ldots,([p=j]T,\omega[p=j]),([p\neq j]T,\omega[p\neq j]),\ldots]$$

5 Conclusions and future work

A multiple-valued extension of the results obtained for classical logic in [4] has been introduced, which can be seen as the refined version of the results in [5]. As a result it is possible to obtain simpler statements of the theorems and, as a consequence, reduction transformations are more adequately described in terms of rewrite rules.

We have introduced Δ-trees for signed formulas. This allows for a compact representation for well-formed formulas as well as for a number of reduction strategies in order to consider only those occurrences of literals which are relevant for the satisfiability of the input formula. The reduction theorems have been complemented by a Davis-Putnam-like branching strategy in order to provide a decision procedure.

References

1. G. Aguilera, I.P. de Guzmán, M. Ojeda-Aciego, and A. Valverde. Reductions for non-clausal theorem proving. *Theoretical Computer Science*, 266(1/2):81–112, 2001.
2. S. Aguzzoli and A. Ciabattoni. Finiteness in infinite-valued Lukasiewicz logic. *Journal of Logic, Language and Information*, 9(1):5–29, 2000.
3. C. Files, R. Drechsler, and M. Perkowski. Functional decomposition of MVL functions using multi-valued decision diagrams. In *Proc. ISMVL'97*, pages 7–32, 1997.
4. G. Gutiérrez, I.P. de Guzmán, J. Martínez, M. Ojeda-Aciego, and A. Valverde. Satisfiability testing for Boolean formulas using Δ-trees. *Studia Logica*, 72:33–60, 2002.
5. I.P. de Guzmán, M. Ojeda-Aciego, and A. Valverde. Reducing signed propositional formulas. *Soft Computing*, 2(4):157–166, 1999.
6. I.P. de Guzmán, M. Ojeda-Aciego, and A. Valverde. Restricted Δ-trees in multiple-valued logics. In *AI - Methodologies, Systems, Applications. AIMSA'02*. Lect. Notes in Computer Science 2443, 2002.
7. R. Hähnle. Uniform notation of tableaux rules for multiple-valued logics. In *Proc. Intl Symp on Multiple-Valued Logic*, pages 238–245. IEEE Press, 1991.
8. D. Mundici and N. Olivetti. Resolution and model building in the infinite-valued calculus of Lukasiewicz. *Theoretical Computer Science*, 200:335–366, 1998.
9. N.V. Murray and E. Rosenthal. Improving tableau deductions in multiple-valued logics. In *Proc. 21st Intl Symp on Multiple-Valued Logic*, pages 230–237. IEEE Press, 1991.
10. D. Pearce, I.P. de Guzmán, and A. Valverde. Computing equilibrium models using signed formulas. In *Proc. 1st Intl Conf on Computational Logic, CL'2000*, Lect. Notes in Artificial Intelligence 1861, pages 688–702, 2000.