Multiple-Valued Tableaux with Δ-reductions

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

Dept. Matemática Aplicada. Universidad de Málaga.
P.O. Box 4114, E-29080 Málaga, Spain

Abstract

We introduce the reduced signed logics *which generalize previous approaches to signed logics in the sense that each variable is allowed to have its own set of semantic values.* Reductions *on both signed logics and signed formulas are used to describe improvements in tableau provers for MVLs. A labelled deductive system allows to use the implicit information in the formulas to describe improved expansion rules based on these reductions.*

Keywords: automated deduction, signed logics

1 Introduction

Signed logics are a fundamental tool in the development of automated theorem provers for Multiple-Valued Logics [8]. The reason is that, for any finite-valued logic $\boldsymbol{L}$, it is possible to define a transformation Φ with the following property [7, 8]:

A is valid in $\boldsymbol{L}$ if and only if the signed formula $\Phi(A)$ is unsatisfiable

This way, testing validity in a finite-valued logic reduces to testing satisfiability of signed formulas.

The expansion rules of tableau systems for MVLs are essentially based in the conversion of the input formula into a signed formula; and the closure tests of the tableau system study the satisfiability of this signed formula. We will consider the input formula to be already signed, and we will focus on the development of a satisfiability tester for *signed logics.*

The basic idea of a tableau method is the generation of a tree out of an input formula, to be tested for satisfiability, by the following operations: Extending the tree by applying an α- or a β-rule, and after each extension, checking the set of literals of each branch for unsatisfiability (the branch is said to be *closed* whenever it is unsatisfiable).

The exponential behaviour of (propositional) tableau methods is due to applications of the β-rule, for it increases the number of branches to be analysed. The improved versions of tableau methods focus on decreasing the number of applications of β-rules; the main strategies used being the application of simplifications as a preprocessing step to either avoid the generation of subsumed branches in the proof [3] or to decrease the internal links in the goal formula [9]; or the application of tests during the proof [4] (subsumption checks, factoring, lemmata, etc).

We have developed a tableau-based satisfiability tester for signed logics with the following improvements:

(i) We make simplifications, generically called Δ-reductions, during the analysis of each branch. The complexity of our simplifications is usually linear, or at most quadratic, whereas most of the simplifications described in the literature have an excessive computational cost.

(ii) A remarkable feature of the Δ-reductions in a multiple-valued framework is that they, dynamically, *reduce* (in the sense of Definition 2) the logic we are working with; therefore, it is possible that we are working with different logics in different branches.

(iii) The β-rule is improved; the basic β-rule is replaced by a Davis-Putnam based β-rule improved with Δ-reductions.

(iv) For efficiency reasons, most tableau systems use the atomic closure of a branch to check its unsatisfiability; our method allows a finer test, with no extra computational cost.

2 Reduced signed logics

Our approach to signed logics, by using an abstract construction in the framework of propositional logics, has no reference to either an initial multiple-valued logic or an specific algorithm, i.e. our definition is completely independent from the application. We also introduce the concept of *reduced signed logics*, which generalizes the signed logics in [7, 8] in the sense that each variable is allowed to have its own set of semantic values (these restrictions can be done in any multiple-valued logic in order to improve the specification of problems in which some variables are known not to be valued in some subset of $\boldsymbol{n}$). We introduce the *reductions of logics* in order to improve satisfiability testers for signed logics; specifically, these reductions are integrated in the algorithm, in such a way that the underlying logic is dynamically reduced in each branch; which is a powerful way of integrating the simplifications described in this work in a tableau algorithm.

We will assume that every propositional logic is built on the same countable set of propositional variables, $\boldsymbol{V}$.

Definition 1: Let $\omega\colon \boldsymbol{V} \to 2^{\boldsymbol{n}} \smallsetminus \varnothing$ be a function, called the *possible truth values function*, we define

1. The set of *ω-signed literals* as follows:

$$\textsc{lit}_\omega = \{S{:}p \mid S \subseteq \omega(p), p \in \mathcal{V}\} \cup \{\bot, \top\}$$

   In a literal $\ell = S{:}p$, the set S is called the *sign of ℓ* and p is the *variable of ℓ*.

2. The *signed logic valued in $\boldsymbol{n}$ by ω*, denoted $\mathbf{S}_\omega = (\mathcal{S}_\omega, \mathcal{M})$, is the logic defined as follows:

(a) Its (propositional) language $\mathcal{S}_\omega$ is the words algebra $(\textsc{form}_\omega, \bot, \top, \vee, \wedge)$ generated by $\textsc{lit}_\omega$ where $\bot$ and $\top$, the logical constants and boolean conjunction and disjunction can work with any finite arity. The elements of $\textsc{form}_\omega$ are called *ω-signed formulas.*

(b) Its generalized matrix, $\mathcal{M} = (\mathcal{N}, D, \mathcal{I})$ is given by:

   b.1 $\mathcal{N} = (\{0,1\}, 0, 1, \max, \min)$

   b.2 $D = \{1\}$

   b.3 The elements of $\mathcal{I}$ are *ω-assignments*, that is, homomorphisms $I\colon \mathcal{S}_\omega \to \mathcal{N}$ defined as the unique extension of a function $\imath\colon \textsc{lit}_\omega \to \{0,1\}$ verifying:

   i. For every $p \in \mathcal{V}$ there exists a unique $j \in \omega(p)$ such that $\imath(\{j\}{:}p) = 1$.

   ii. $\imath(S{:}p) = 1$ if and only if there exists $j \in S$ such that $\imath(\{j\}{:}p) = 1$.

If $\omega(p) = \boldsymbol{n}$ for all $p \in \boldsymbol{V}$, we have the usual $\boldsymbol{n}$-valued signed logic, otherwise the logics $\mathbf{S}_\omega$ are called *reduced signed logics.*

Validity and satisfiability are defined in $\mathbf{S}_\omega$ in the usual manner: An ω-signed formula, A, is *satisfiable* if there exists an ω-assignment I such that $I(A) = 1$; in this case, I is a *model* for A; A, is *valid* if every ω-assignment I is a model of A. Two ω-signed formulas are *equivalent*, $A \equiv B$, if $I(A) = I(B)$ for every ω-assignment I. An ω-signed formula A is a *consequence* of the set of ω-signed formulas Γ if every model of Γ is a model of A. Given a literal $S{:}p$, its *conjugate* $(\omega(p)\smallsetminus S){:}p$ will be denoted $\overline{S{:}p}$. Similarly, we say that $\bot$ is the conjugate of $\top$ and vice versa.

The Δ-reductions on a given logic, which will be introduced in the following section, restrict the possible truth-values for one or more variables, the obtained logic is said to be a *reduction* of the initial logic:

Definition 2: Let $\mathbf{S}_{\omega_1}$ and $\mathbf{S}_{\omega_2}$ be logics valued in n. We say that $\mathbf{S}_{\omega_1}$ is a *reduction* of $\mathbf{S}_{\omega_2}$ if $\omega_1(p) \subseteq \omega_2(p)$ for all p. The study of the satisfiability in the reduced logics can be easily lifted to the initial logic, as stated in the proposition below:

Proposition 1 *If $\mathbf{S}_{\omega_1}$ and $\mathbf{S}_{\omega_2}$ are two logics valued in $\boldsymbol{n}$ and $\mathbf{S}_{\omega_1}$ is a reduction of $\mathbf{S}_{\omega_2}$, then:*

1. *Every formula of $\mathbf{S}_{\omega_1}$ is a formula in $\mathbf{S}_{\omega_2}$; that is*
$$\text{FORM}_{\omega_1} \subseteq \text{FORM}_{\omega_2}$$
2. *If I is a ω_1-signed assignment, then there is a unique ω_2-signed assignment extending I.*

3 The Δ-reductions

We introduce here some simplifications to be applied to sets of formulas $\Omega = \{A_1, \dots, A_m\}$. We call Δ-reductions to these strategies, which use sets of unitary implicants and implicates of the formulas to generalize purity properties and dynamically modify the signed logic we are working in. The prefix Δ is introduced because the sets of implicants and implicates we will use are the Δ-sets, introduced in [1].

Literals ${\scriptstyle\{j\}}{:}p$ are fundamental in the description of the Δ-reductions, so we introduce a simplified notation for them:

$$pj \stackrel{def}{=} {\scriptstyle\{j\}}{:}p$$

these literals and their conjugated will be called, respectively, *positive* and *negative* literals.

Definition 3: If A is a signed formula and $pj \models A$, we say that pj is an *unitary implicant* of A; if $A \models \overline{pj}$, then we say that $\overline{pj}$ is an *unitary implicated* of A.

The following definitions are needed in order to define the Δ-reductions. Specifically, we have two types of reductions, namely, reductions on the logic we are working in and reductions on the formulas.

Definition 4: Let $\mathbf{S}_\omega$ be a signed logic valued in $\boldsymbol{n}$, p a propositional variable and $j \in \omega(p)$; the *reductions associated to the mappings $\omega[p \neq j]$ and $\omega[p = j]$* are defined as follows:

- $\omega[p \neq j](p) = \omega(v)$ if $v \neq p$ and $\omega[p \neq j](p) = \omega(p) \smallsetminus \{j\}$.
- $\omega[p = j](v) = \omega(v)$ if $v \neq p$ and $\omega[p = j](p) = \{j\}$.

Definition 5: 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 ${\scriptstyle\{j\}}{:}p$ by $\perp$, $\overline{{\scriptstyle\{j\}}{:}p}$ by $\top$ and ${\scriptstyle S}{:}p$ by ${\scriptstyle (S\smallsetminus\{j\})}{:}p$; 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 ${\scriptstyle S}{:}p$ with $j \in S$ by $\top$ and every literal ${\scriptstyle S}{:}p$ with $j \notin S$ by $\perp$; in addition, the constants are deleted using the 0-1-laws.

3.1 α_{CR}-rule: complete reduction

The first Δ-reduction we introduce, called *complete reduction* [1], is associated to unitary implicants of the formulas; the complete reduction generates as its output equisatisfiable and smaller-sized formulas in a reduction of the initial logic.

Theorem 1 *Let A be a formula in $\mathbf{S}_\omega$ such that $A \models \overline{pj}$, then A is satisfiable in $\mathbf{S}_\omega$ if and only if $A[p \neq j]$ is satisfiable in $\mathbf{S}_{\omega[p\neq j]}$. In addition, every model of $A[p \neq j]$ in $\mathbf{S}_{\omega[p\neq j]}$ is a model of A in $\mathbf{S}_\omega$.*

This theorem will be used as an α-like rule in the tableau prover, the *α_{CR}-rule*.

3.2 α_{PL}-rule: pure literals

The definition of pure literal in signed logics that can be found in the literature corresponds to our *pure positive literals*. The possibility of reducing the logic also allows to exploit the *pure negative literals*. These concepts are introduced below:

Definition 6: Let $A \in \mathbf{S}_\omega$ and $p \in \mathcal{V}$.

1. A positive literal pj is called *pure in A* if $j \in S$ for every literal ${}_S{:}p$ in A.
2. A negative literal $\overline{pj}$ is called *pure in A* if $j \notin S$ for every literal ${}_S{:}p$ in A.

Theorem 2 *Let A be a formula in $\mathbf{S}_\omega$*

1. *If pj is pure in A, then A is satisfiable in $\mathbf{S}_\omega$ if and only if $A[p = j]$ is satisfiable in $\mathbf{S}_{\omega[p=j]}$. Furthermore, every model of $A[p = j]$ in $\mathbf{S}_{\omega[p=j]}$ is a model of A in $\mathbf{S}_\omega$.*
2. *If $\overline{pj}$ is pure in A, then A is satisfiable in $\mathbf{S}_\omega$ if and only if $A[p \neq j]$ is satisfiable in $\mathbf{S}_{\omega[p\neq j]}$. Furthermore, every model of $A[p \neq j]$ in $\mathbf{S}_{\omega[p\neq j]}$ is a model of A in $\mathbf{S}_\omega$.*

This theorem will be used as an α-like rule in the tableau prover, the α_{PL}*-rule*.

3.3 $\boldsymbol{\beta_{DP}}$-rule

The Δ-reductions provide an alternative form for the β-rule which can be interpreted as a generalized version of the Davis-Putnam procedure. The following result justifies the correctness of the expansion rule which will be denoted β_{DP}-rule.

Theorem 3 *Let p be a variable occurring in a formula A in $\mathbf{S}_\omega$ and consider $j \in \omega(p)$. Then, A is satisfiable if and only if some of the following conditions hold:*

1. *$A[p = j]$ is satisfiable in $\mathbf{S}_{\omega[p=j]}$, and a model for $A[p = j]$ is a model for A.*
2. *$A[p \neq j]$ is satisfiable in $\mathbf{S}_{\omega[p\neq j]}$, and a model for $A[p \neq j]$ is a model for A.*

3.4 $\boldsymbol{\alpha_{DP}}$-rule

The choosing of an adequate literal p to branch in Theorem 3 might not *branch* the tableau, but *split* it. The following corollary of Theorem 3, justifies the correctness of a new α-like rule, called the α_{DP}*-rule*.

Corollary 1 *Let $\Omega = \{A_1, \dots, A_m\}$ be a set of formulas in $\mathbf{S}_\omega$ such that*

1. *$pj \models A_i$ for all $i \in \{1 \leq i \leq k-1\}$*
2. *$pj' \models A_i$ for every $j' \in \omega(p) \smallsetminus \{j\}$ and for all $i \in \{k \leq i \leq m\}$*

Then Ω is satisfiable if and only if one of the following conditions hold:

1. *$\Omega[p \neq j] = \{A_1[p \neq j], \dots, A_{k-1}[p \neq j]\}$ is satisfiable in $\mathbf{S}_{\omega[p\neq j]}$, in this case a model for $\Omega[p \neq j]$ is a model for Ω.*
2. *$\Omega[p = j] = \{A_k[p = j], \dots, A_m[p = j]\}$ is satisfiable in $\mathbf{S}_{\omega[p=j]}$, in this case a model for $\Omega[p = j]$ is a model for Ω.*

4 A labelled deductive system

In this section we use the Δ-reductions introduced above to develop a refinement of the basic tableaux system stated in the introduction. The system is described as a labelled deductive system [5] where the nodes of the execution tree (which are sets of formulas), are labelled with $(\omega, \Delta_0, \Delta_1)$ where ω determines the logic we are using in the branch, Δ_0 is a set of unitary implicates of the formulas and Δ_1 is a set of implicants of the formulas. Since the calculation of the set of all the unitary implicants and implicates is very complex, we will only use those implicants/implicates which can be determined by means of a linear complexity algorithm.

Definition 7: Let A be a ω-signed formula, $\Delta_0(A)$ is either $\perp$ or a set of negative literals, and $\Delta_1(A)$ is either $\top$ or a set of positive literals. The recursive definition of these sets is given by the following rules:

$\Delta_0(\bot) = \bot, \qquad \Delta_1(\bot) = \varnothing$
$\Delta_0(\top) = \varnothing, \quad \Delta_1(\top) = \top$
$\Delta_0(S{:}p) = \overline{pj_1} \dots \overline{pj_m},$
  if $\varnothing \neq S \neq \omega(p)$, $\omega(p) \smallsetminus S = \{j_1, \dots, j_m\}$
$\Delta_1(S{:}p) = pj_1 \dots pj_m,$
  if $\varnothing \neq S \neq \omega(p)$, $S = \{j_1, \dots, j_m\}$
$\Delta_0\left(\bigwedge_{i=1}^n A_i\right) = \mathtt{Uni}(\Delta_0(A_1), \dots, \Delta_0(A_n))$
$\Delta_0\left(\bigvee_{i=1}^n A_i\right) = \mathtt{Int}(\Delta_0(A_1), \dots, \Delta_0(A_n))$
$\Delta_1\left(\bigwedge_{i=1}^n A_i\right) = \mathtt{Int}(\Delta_1(A_1), \dots, \Delta_1(A_n))$
$\Delta_1\left(\bigvee_{i=1}^n A_i\right) = \mathtt{Uni}(\Delta_1(A_1), \dots, \Delta_1(A_n))$

The operator `Uni` calculates the union of its arguments, and also identifies the result with $\bot$ (resp. $\top$) if it contains all the negative (resp. positive) literals with variable p for some variable p.[1] The operator `Int` calculates the intersection of its arguments.

When applying Theorem 1 and Corollary 1 in the satisfiability tester for every branch, we will use the Δ-lists. For this reason we have called Δ-reductions to the simplifications described in this work.

For the sake of simplicity, the Δ-sets will be written as lists whose elements are ordered with the lexicographic order of the variables and the numerical order of truth-values. Thus, the sets $\Delta_0(A)$ and $\Delta_1(A)$ will be sometimes called Δ-lists.

From the definition of Δ-lists the following fundamental theorem arises:

Theorem 4 *Let A be a ω-signed formula:*

1. *If $\overline{pj} \in \Delta_0(A)$, then $A \models \overline{pj}$*
2. *If $\Delta_0(A) = \bot$, then $A \equiv \bot$; in this case, A is called* 0-conclusive.
3. *If $pj \in \Delta_1(A)$, then $pj \models A$; specifically, in this case A is satisfiable and any interpretation satisfying $I(\{j\}{:}p) = 1$ is a model for A.*
4. *If $\Delta_1(A) = \top$, then $A \equiv \top$; in this case, A is called* 1-conclusive.
5. *If $\Delta_1(A) \cup \overline{\Delta_0(A)}$ is the set of all the positive literals with variable p, then $A \equiv \bigvee_{\ell \in \Delta_1(A)} \ell$; in this case, A is called* simple.

[1] If $A \models \overline{pj}$ for every $j \in \omega(p)$, then A is unsatisfiable; dually, if $pj \models A$ for every $j \in \omega(p)$, then A is valid.

Definition 8: Let A be a ω-signed formula, we say that A is Δ-restricted if it has neither conclusive nor simple subformulas.

The calculation of $\Delta_0(A)$ and $\Delta_1(A)$ is linear w.r.t. the size of the formula; furthermore, during the calculation it is possible to eliminate the conclusive subformulas of A and substitute the simple formulas by the corresponding literal. Consequently, the conversion into Δ-restricted form is linear. In the following, we will only work with Δ-restricted formulas.

4.1 Expansion rules

In the tableau system we are describing now the tableau's nodes are sets of signed formulas labelled as follows:

$$^{(\omega,\Delta_0,\Delta_1)}{:}\{A_1, \dots A_m\}$$

where $A_1, \dots, A_m$ are Δ-restricted formulas in $\mathbf{S}_\omega$ and

$$\Delta_0 = \Delta_0(A_1) \cup \dots \cup \Delta_0(A_m)$$
$$\Delta_1 = \Delta_1(A_1) \cap \dots \cap \Delta_1(A_m)$$

All the rules introduced in this section (but the α-rule) require the updating of the Δ-labels in the generated leaves, in order to always obtain Δ-restricted formulas; therefore, the resulting labels (Δ_0', Δ_1') in the newly generated leaves need not to be the same as the previous ones (Δ_0, Δ_1).

The following expansion rules will be considered in our improved tableaux system.

4.1.1 α-rule

$$\begin{array}{c} ^{(\omega,\Delta_0,\Delta_1)}{:}\{\dots, A_1 \wedge \dots \wedge A_m, \dots\} \\ | \\ ^{(\omega,\Delta_0,\Delta_1)}{:}\{\dots, A_1, \dots, A_m, \dots\} \end{array}$$

4.1.2 α_{CR}-rule

$$\begin{array}{c} ^{(\omega,\{\overline{pj}\}\cup\Delta_0,\Delta_1)}{:}\{A_1, \dots, A_m\} \\ | \\ ^{(\omega[p\neq j],\Delta_0',\Delta_1')}{:}\{A_1[p \neq j], \dots, A_m[p \neq j]\} \end{array}$$

4.1.3 α_{PL}-rule

If pj is pure in A_i for all $i \in \{1, \dots, m\}$, then

$$\begin{array}{c} {\scriptstyle (\omega,\Delta_0,\Delta_1):}\{A_1, \dots, A_m\} \\ | \\ {\scriptstyle (\omega[p=j],\Delta_0',\Delta_1'):}\{A_1[p=j], \dots, A_m[p=j]\} \end{array}$$

If $\overline{pj}$ is pure in A_i for all $i \in \{1, \dots, m\}$, then

$$\begin{array}{c} {\scriptstyle (\omega,\Delta_0,\Delta_1):}\{A_1, \dots, A_m\} \\ | \\ {\scriptstyle (\omega[p\neq j],\Delta_0',\Delta_1'):}\{A_1[p \neq j], \dots, A_m[p \neq j]\} \end{array}$$

4.1.4 α_{DP}-rule

If $\{A_1, \dots A_m\}$ is pj-splittable, that is $\{A_1, \dots A_m\} = \{A_{l_1}, \dots A_{l_{k-1}}\} \cup \{A_{l_k}, \dots A_{l_m}\}$ satisfying

1. $pj \models A_{l_i}$ for all $i \in \{1 \le i \le k-1\}$
2. $pj' \models A_{l_i}$ for every $j' \in \omega(p) \smallsetminus \{j\}$ and for all $i \in \{k \le i \le m\}$

$$\begin{array}{c} {\scriptstyle (\omega,\Delta_0,\Delta_1):}\{A_1, \dots, A_m\} \\ {\scriptstyle (\omega[p=j],\Delta_0',\Delta_1'):}\{A_{l_1}[p=j], \dots, A_{l_{k-1}}[p=j]\} \qquad {\scriptstyle (\omega[p\neq j],\Delta_0'',\Delta_1''):}\{A_{l_k}[p \neq j], \dots, A_{l_m}[p \neq j]\} \end{array}$$

4.1.5 β_{DP}-rule

$$\begin{array}{c} {\scriptstyle (\omega,\Delta_0,\Delta_1):}\{A_1, \dots, A_m\} \\ {\scriptstyle (\omega[p=j],\Delta_0',\Delta_1'):}\{A_1[p=j], \dots, A_m[p=j]\} \qquad {\scriptstyle (\omega[p\neq j],\Delta_0'',\Delta_1''):}\{A_1[p \neq j], \dots, A_m[p \neq j]\} \end{array}$$

4.2 The improved tableaux system

We can now describe our improved tableaux system with Δ-labels, as follows:

Definition 9: Let $\Omega = \{A_1, \dots, A_m\}$ be a set of restricted ω-signed formulas

1. The following tree is a tableau for Ω:

   $${\scriptstyle (\omega,\Delta_0,\Delta_1):}\{A_1, \dots A_m\}$$

   where $\Delta_0 = \Delta_0(A_1) \cup \dots \cup \Delta_0(A_m)$, $\Delta_1 = \Delta_1(A_1) \cap \dots \cap \Delta_1(A_m)$;
2. If T is a tableau for Ω, and T^* results from T by the application of any tableau expansion rule, then T^* is a tableau for Ω:

Definition 10: Let T be a tableau for $\{A_1, \dots, A_m\}$

1. A branch of T, with leaf ${\scriptstyle (\omega,\Delta_0,\Delta_1):}\{A_1, \dots A_m\}$, is said to be *closed* if $\Delta_0 = \bot$.
2. A branch of T, with leaf ${\scriptstyle (\omega,\Delta_0,\Delta_1):}\{A_1, \dots A_m\}$, is said to be *open* $\Delta_1 \neq \varnothing$.

Note that our definition of closed and open branch is based on Theorem 4, and it is by no means standard: the condition $\Delta_0 = \bot$ is more general than the atomic closure of a branch, and the condition $\Delta_1 \neq \varnothing$ allows to detect the satisfiability of a branch even when that branch is not still complete.

The completeness and correctness of this improved tableaux method is a consequence of Theorems 1–4, and Corollary 1:

Theorem 5 *Let $\Omega = \{A_1, \dots, A_m\}$ be a set of ω-signed formulas*

1. *If Ω has a closed tableau, then it is unsatisfiable.*
2. *If Ω has a tableau with an open branch, then it is satisfiable; in addition, if pj is any element in the Δ_1-list of the leaf ${\scriptstyle (\omega',\Delta_0,\Delta_1):}\{\Omega'\}$, then any model of pj in $\mathbf{S}_{\omega'}$ is a model of Ω in $\mathbf{S}_\omega$.*

4.3 Describing the algorithm

Attending to the complexity of each rule, the following algorithm is given for a set Ω of ω-signed formulas.

Step 1 Generate the one-leaf tableau for Ω.

Step 2

2.1 If every branch is closed, then the initial set is unsatisfiable and the algorithm ends.

2.2 If there exists an open branch, then the initial set is satisfiable, a model is given as in Theorem 5 and the algorithm ends.

Step 3 If the α-rule can be applied on a non-closed branch, then it is applied as many times as possible and go back to step 2.

Step 4 If the Δ_0-label of the leaf of a non-closed branch is non-empty, then the α_{CR}-rule is applied for all the elements in Δ_0 and go back to step 2.

Step 5 If the conjunction of the formulas in the leaf of a non-closed branch have pure literals, then the α_{PL}-rule is applied for each pure literal and go back to step 2.

Step 6 If the α_{DP}-rule can be applied on a non-closed branch, then this rule is applied and go back to step 2.

Step 7 If no rule of type α can be applied on a non-closed branch, then the β_{DP}-rule is applied and go back to step 2.

5 Conclusions

The Δ-reductions introduced above allow the development of refinements of basic tableau systems. These refinements can be described as a labelled deductive system [5], with labels $(\omega, \Delta_0, \Delta_1)$ where ω determines the logic in which the closing property is analysed, and Δ_0 (resp. Δ_1) is the set of implicates (resp. implicants) of the formulas, as defined in [1].

The computational pay-off of the use of reductions may seem doubtful, since some time must be spent for scanning the formula and applying the corresponding reduction. It is known that a tableau 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.

The reductions of formulas applied during the expansion of the tableau have complexity at most quadratic, the given closure tests are more general than the atomic closure test but with a similar cost. Therefore, we are applying a polynomial processing for an exponential gain.

A more general approach to the Δ-reductions in many-valued logics can be seen in [1]. The TAS methodology for signed logics, a rewrite-based version of the reductions, and a more detailed study of the signing transformations can be seen in [10]. Finally, it is worth to note that reductions can be successfully applied to several types of logic [2, 6].

References

[1] G. Aguilera, I. P. de Guzmán, M. Ojeda, and A. Valverde. Reducing signed propositional formulas. *Soft Computing*, 2(4):157–166, 1999.

[2] G. Aguilera, I. P. de Guzmán, M. Ojeda. Increasing the efficiency of automated theorem proving. *Journal of Applied Non-Classical Logics*, 5(1):9–29, 1995.

[3] B. Beckert, R. Hähnle, and G. Escalada-Imaz. Simplification of many-valued logic formulas using anti-links. *Journal of Logic and Computation*, 8(4):569–587, 1998.

[4] W. Bibel, S. Bruening, U. Egly, D. Korn, and T. Rath. Issues in theorem proving based on the connection method. *LNAI* 918, pp. 1–16, 1995.

[5] D. M. Gabbay. *Labelled Deductive Systems*. Oxford University Press, 1996.

[6] I. P. de Guzmán, M. Ojeda, A. Valverde. Implicates and reduction techniques for temporal logics. *LNAI* 1489, pp. 309-323. 1998 (extended version to appear in the *Annals of Mathematics and AI*)

[7] R. Hähnle. *Automated Deduction in Multiple Valued Logics*. Oxford University Press, 1993.

[8] J. J. Lu, N. V. Murray, and E. Rosenthal. A framework for automated reasoning in multiple-valued logics. *Journal of Automated Reasoning*, 21(1):39–67, 1998.

[9] K. Mayr. Link deletion in model elimination. *Lect. Notes in Artif. Intelligence* 918, pp. 169–184, 1995.

[10] A. Valverde. *Δ-trees of implicants and implicates and reductions of signed logics in ATPs.* PhD thesis, Universidad de Málaga, Spain, July 1998.