Representing Boolean formulas by using trees of implicants and implicates

G. Gutiérrez, I.P. de Guzmán, J. Martínez, M. Ojeda-Aciego, A.Valverde

Dept. Matemática Aplicada. Universidad de Málaga.
P.O. Box 4114. E-29080 Málaga, Spain. Email: aciego@ctima.uma.es

Abstract— **A new tree-based representation for propositional formulas is introduced. This representation, the Δ-trees, allows a compact representation for well-formed formulas as well as a number of reduction strategies in order to consider only those occurrences of literals which are relevant for the satisfiability of the input formula.**

Keywords: automated deduction, propositional formula representation, implicates/implicants

I. Introduction

The ability to reason on specifications written in a language as close as possible to natural language is important for information sciences; thus, reasoning efficiently on nnf (negation normal form) is interesting because these formulas are easier to obtain from specifications given in natural language. Obviously, there is a need of efficient representations of nnfs so that, in a similar manner as with cnfs (conjunctive normal forms), we are able to describe and implement efficient algorithms on this kind of formulas.

Formulas in cnf are usually interpreted as lists of clauses, and formulas in dnf are interpreted as lists of cubes; these interpretations allow efficient descriptions and implementations of algorithms to study satisfiability (e.g. linear ordered resolution). In this work we focus on the generalization of these interpretations to nnfs. The proposed generalization will use *trees* of clauses and cubes.

Specifically, nnfs are represented as trees of clauses and cubes such that each clause-node in the tree is an implicant of the formula represented by its scope and, similarly, each cube-node is an implicate of the formula represented by its scope. The new representation is named Δ-tree because its nodes are built up from Δ-lists [1]. After defining the notion of Δ-tree, we define operators `Norm` and Δ-`Tree` which, respectively, associate a nnf to each Δ-tree and vice versa. In addition, it can be shown that this correspondence preserves equivalence and, therefore, we can easily extend the concepts of validity and satisfiability to Δ-trees.

We introduce the concept of restricted Δ-tree (generalizing the well-known concept of restricted cnf in which clauses with repeated or contradictory literals are not allowed and subsumed clauses are omitted), which involves only restricted clauses and cubes in the representation and, in addition, prohibits that a single literal is both an implicant and an implicate of the same subformula.

The transformation into restricted Δ-tree is given by a number of meaning-preserving transformations, with at most quadratic complexity, which eliminate the conclusive or simple nodes and usually reduces the size of the input Δ-tree. Roughly speaking a conclusive node in a Δ-tree is one which can be substituted by a logical constant preserving the meaning of the whole tree, and a simple node in a Δ-tree satisfies that the subformula it represents is equivalent to a literal; these concepts are generalized from the classical case [2].

Technically, the transformation into restricted form is made by the operators $\Phi_\bot$ and $\Phi_\top$, and Φ_ℓ which eliminate, respectively, the conclusive and the simple nodes in a Δ-tree.

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

II. 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 $\{\neg, \wedge, \vee\}$, the semantics for this language being the standard one. The symbol $\equiv$ denotes logical equivalence, and $\models$ denotes logical consequence. We will use the usual notions of literal, clause, cube, and negation normal form:

In this paper, we will always use cubes and clauses ordered by the lexicographic order in the set of literals $\mathcal{V}^{\pm}$.

- 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 $A \models \ell$.

We will use finite lists written in juxtaposition, with the standard notation, `nil`, for the empty list. If $\lambda = \ell_1\ell_2\ldots\ell_n$ is a list of literals, then $\overline{\lambda} = \overline{\ell_1}\,\overline{\ell_2}\ldots\overline{\ell_n}$

In the next section we present a short summary of Δ-lists. These were firstly introduced in [2], and have been recently used in the development of a large set of reduction strategies for studying the satisfiability of non-clausal propositional formulas [1] which can be extended to non-classical logics as well [3], [4], thus allowing a uniform metatheory of reduction of

formulas extremely useful in the field of automated reasoning.

A. The Δ-lists

We associate to each nnf A a pair of lists of literals denoted $\Delta_0(A)$ and $\Delta_1(A)$ (so-called associated Δ-lists of A).

In a nutshell, $\Delta_0(A)$ and $\Delta_1(A)$ are, respectively, lists of implicates and implicants of A.

Definition 1: Given a nnf A, $\Delta_0(A)$ and $\Delta_1(A)$ are elements of $\mathtt{List}(\mathcal{V}^{\pm}) \cup \{\top, \bot\}$ called Δ-lists associated with A, recursively defined as follows:

$$\begin{array}{ll} \Delta_0(\ell) = \ell & \Delta_1(\ell) = \ell \\ \Delta_0(\bot) = \bot & \Delta_1(\bot) = \mathtt{nil} \\ \Delta_0(\top) = \mathtt{nil} & \Delta_1(\top) = \top \\ \Delta_0\left(\bigwedge_i A_i\right) = \bigotimes_i \Delta_0(A_i); & \Delta_1\left(\bigwedge_i A_i\right) = \bigcap_i \Delta_1(A_i) \\ \Delta_0\left(\bigvee_i A_i\right) = \bigcap_i \Delta_0(A_i); & \Delta_1\left(\bigvee_i A_i\right) = \bigoplus_i \Delta_0(A_i) \end{array}$$

In the definition above there are two versions of the union operator, and this can be explained because of the intended interpretation of these sets together with Theorem 1 below:
1. Elements in Δ_0 are considered to be conjunctively connected. Namely, if ℓ and $\overline{\ell} \in \Delta_0(A)$, then $\Delta_0(A)$ simplifies to $\bot$. This way, we obtain a set of implicates which can be thought of as a cube.
2. Elements in Δ_1 are considered to be disjunctively connected. Namely, if ℓ and $\overline{\ell} \in \Delta_1(A)$, then $\Delta_1(A)$ simplifies to $\top$. This way, we obtain a set of implicants which can be thought of as a clause.

B. Information in the Δ-lists

The next theorem states that elements of $\Delta_0(A)$ are implicates of A, and that elements of $\Delta_1(A)$ are implicants of A. It follows easily by structural induction from the definition of Δ-lists.

Theorem 1: Let A be a nnf and ℓ be a literal in A then:
1. If $\ell \in \Delta_0(A)$, then $A \models \ell$, that is, $A \equiv \ell \wedge A$.
2. If $\ell \in \Delta_1(A)$, then $\ell \models A$, that is, $A \equiv \ell \vee A$.

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

Corollary 1: For all nnf A we have one and only one of the following possibilities:
- There is $b \in \{0, 1\}$ such that $\Delta_b(A) = \mathtt{nil}$,
- $\Delta_1(A) = \Delta_0(A) = \ell$, and then $A \equiv \ell$.

Finally, the following corollary defines a meaning-preserving substitution for a formula A whose result contains only one occurrence of any literal in the Δ-lists of A.

Corollary 2: Let A a nnf and ℓ a literal in A. Then:
1. If $\ell \in \Delta_0(A)$, then $A \equiv A[\ell/\top, \overline{\ell}/\bot] \wedge \ell$.
2. If $\ell \in \Delta_1(A)$, then $A \equiv A[\ell/\bot, \overline{\ell}/\top] \vee \ell$.

Remark 1: The substitution defined in the corollary above never increases the size of A; actually, the size is always decreased but in the following cases:
- If A is a conjunctive formula such that $\ell \in \Delta_0(A)$, and there is only one occurrence of ℓ.
- If A is a disjunctive formula such that $\ell \in \Delta_1(A)$, and there is only one occurrence of ℓ.

III. The Δ-trees

Definition 2: A *Δ-tree* T is a labeled tree in

$$\mathcal{H} = \{[\gamma]\lambda \mid \gamma \in \{\alpha, \beta\} \text{ and } \lambda \in \mathtt{List}(\mathcal{V}^{\pm}) \cup \{\top, \bot\}\}$$

inductively defined by the three properties below:
1. The leaves in a Δ-tree are elements in $\mathcal{H}$.
2. Let $T_1, \ldots, T_m$ be Δ-trees whose roots are $[\beta]\lambda_1, \cdots, [\beta]\lambda_m$ and $[\alpha]\lambda \in \mathcal{H}$, then $\dfrac{[\alpha]\lambda}{T_1 \quad \ldots \quad T_m}$ is a (conjunctive) Δ-tree.
3. Let $T_1, \ldots, T_m$ be Δ-trees whose roots are $[\alpha]\lambda_1, \cdots, [\alpha]\lambda_m$ and $[\beta]\lambda \in \mathcal{H}$, then $\dfrac{[\beta]\lambda}{T_1 \quad \ldots \quad T_m}$ is a (disjunctive) Δ-tree.

Every Δ-tree T can be interpreted as a propositional formula A in nnf. This interpretation also allows to identify the subtrees of T with subformulas of A.

Definition 3: Given a Δ-tree T, we can generate a nnf by using the operator $\mathtt{Norm}$, recursively defined as follows:
1. $\mathtt{Norm}([\alpha]\lambda) = \bigwedge_{\ell \in \lambda} \ell$, and $\mathtt{Norm}([\alpha]\mathtt{nil}) = \top$
2. $\mathtt{Norm}([\beta]\lambda) = \bigvee_{\ell \in \lambda} \ell$, and $\mathtt{Norm}([\beta]\mathtt{nil}) = \bot$
3. If T is a conjunctive Δ-tree, $T = \dfrac{[\alpha]\lambda}{T_1 \quad \ldots \quad T_m}$ then $\mathtt{Norm}(T) = \mathtt{Norm}([\alpha]\lambda) \wedge \bigwedge_{i=1}^{m} \mathtt{Norm}(T_i)$
4. If T is a disjunctive Δ-tree, $T = \dfrac{[\beta]\lambda}{T_1 \quad \ldots \quad T_m}$ then $\mathtt{Norm}(T) = \mathtt{Norm}([\beta]\lambda) \vee \bigvee_{i=1}^{m} \mathtt{Norm}(T_i)$

Conversely, given a nnf A we can generate a Δ-tree, whose nodes are the Δ-lists associated to A.

Definition 4: Let A be a nnf, we can generate a Δ-tree by using the operator $\Delta\text{-}\mathtt{Tree}$, recursively defined as follows:
1. Let A be a clause, $A \neq \bot$, then $\Delta\text{-}\mathtt{Tree}(A) = [\beta]\Delta_1(A)$.
2. Let A be a non-literal cube such that $A \neq \top$ and A is not a literal, then $\Delta\text{-}\mathtt{Tree}(A) = [\alpha]\Delta_0(A)$.

3. Let A be a disjunctive nnf, and let $A_1, \ldots, A_n$, with $n \geq 1$, be the non-literal disjuncts of A, then

$$\Delta\text{-}\mathtt{Tree}(A) = \frac{[\beta]\Delta_1(A)}{\Delta\text{-}\mathtt{Tree}(A_1) \quad \ldots \quad \Delta\text{-}\mathtt{Tree}(A_n)}$$

4. Let A be a conjunctive nnf, and let $A_1, \ldots, A_n$, with $n \geq 1$, be the non-literal conjuncts of A, then

$$\Delta\text{-}\mathtt{Tree}(A) = \frac{[\alpha]\Delta_0(A)}{\Delta\text{-}\mathtt{Tree}(A_1) \quad \ldots \quad \Delta\text{-}\mathtt{Tree}(A_n)}$$

It is remarkable the idea that, in some sense, the structure of Δ-tree allows to substitute reasoning with literals by reasoning on clauses and cubes.

Note that for the example above $\mathtt{Norm}(\Delta\text{-}\mathtt{Tree}(A))$ is *not* equal to A, for a new literal q is attached as an immediate successor of the root node, making explicit that q is an implicate of the formula.

The next theorem shows that the operators introduced in Definitions 3 and 4 are inverse, up to equivalence.

Theorem 2: Let A be a nnf. Then $A \equiv \mathtt{Norm}(\Delta\text{-}\mathtt{Tree}(A))$.

IV. Towards a restricted form for Δ-trees

In this section, meaning-preserving transformations are introduced which allow to reduce the size of a Δ-tree and get a restricted form for them. These transformations extend to Δ-trees the definitions of Δ_0-*conclusive*, Δ_1-*conclusive* and ℓ-*simple* given for nnfs in [1].

A. Subformulas which can be substituted by constants

The result of Corollary 2 is extended to Δ-trees, in that not only literals, but also subformulas can be substituted by the constants $\top$ or $\bot$. We also introduce the operators $\Phi_\bot$ and $\Phi_\top$ on Δ-trees which reduce a Δ-tree by deleting its *redundant* nodes, that is, those nodes which can be substituted by logical constants in a meaning-preserving way.

Definition 5: Let η be a node of a Δ-tree T is said to be 0-conclusive if it satisfies any of the following conditions:

- It is labeled with $[\alpha]\bot$.
- It is a leaf labeled with $[\beta]\mathtt{nil}$.
- It is a monary node labeled with $[\beta]\mathtt{nil}$.
- It is labeled with $[\alpha]\lambda$, it has an immediate successor $[\beta]\lambda'$ which is a leaf and $\overline{\lambda'} \subseteq \lambda$.
- It is labeled with $[\alpha]\lambda$, its immediate ancestor is labeled with $[\beta]\lambda'$ and $\lambda \cap \lambda' \neq \emptyset$.

Intuitively, the previous definition detects those nodes in the Δ-tree which, in some sense, can be substituted by $\bot$ without affecting the meaning. The effective deletion of those nodes is made by the operator $\Phi_\bot$ which, in addition, reduces the Δ-tree according to the 0-1 laws.

Theorem 3: Let T be a Δ-tree, then $\Phi_\bot(T)$ has no 0-conclusive nodes and, in addition, $T \equiv \Phi_\bot(T)$, where the operator $\Phi_\bot$ is defined as follows:

1. If T is a leaf Δ-tree, then:
   (a) $\Phi_\bot(T) = \bot$, if $T = [\beta]\mathtt{nil}$.
   (b) $\Phi_\bot(T) = T$ otherwise.

If T is not a leaf Δ-tree, then $\Phi_\bot$ traverses T in a reverse depth-first order, that is from the leaves to the root, and from right to left; for every visited node η, the following transformations are applied:

2. If η is labeled with $[\alpha]\bot$, then:
   (a) If $\eta = \varepsilon$, then $\Phi_\bot(T) = \bot$.
   (b) If $\eta \neq \varepsilon$, then the subtree rooted at η is erased.
3. If η is a leaf labeled with $[\beta]\mathtt{nil}$, and η' is the immediate ancestor of η, then the subtree rooted at η' is substituted by $\bot$.
4. If η is a monary node labeled with $[\beta]\mathtt{nil}$, and its only successor σ is labeled with $[\alpha]\lambda$, then:
   (a) If $\eta = \varepsilon$, then the node η is erased.
   (b) Otherwise, if η' is the immediate ancestor of η and it is labeled with $[\alpha]\lambda'$, then nodes η and σ are erased and η' is re-labeled with $\bigl\lfloor\bigwedge\bigr\rfloor\{\lambda, \lambda'\}$, that is, the union of λ and λ' considered conjunctively connected.
5. If η is labeled with $[\alpha]\lambda$ and it has an immediate successor which is a leaf, if this leaf is labeled with $[\beta]\lambda'$ and $\overline{\lambda'} \subseteq \lambda$, then the subtree rooted at η is substituted by $\bot$.
6. If η is labeled with $[\beta]\lambda$ and it has an immediate successor η' which is labeled with $[\alpha]\lambda'$, if $\lambda \cap \lambda' \neq \emptyset$, then the subtree rooted at η' is erased.

The concept of 1-conclusive node and the operator $\Phi_\top$ are defined by duality.

B. Simple leaves

In order to get to a restricted Δ-tree it is also necessary to detect which leaves are redundant, in the sense that are not proper clauses or cubes, but literals.

Definition 6: Let T be a non-leaf Δ-tree, and let η be a leaf in T. We say that η is *simple* if it is labeled with either $[\alpha]\ell$ or $[\beta]\ell$, where $\ell \in \mathcal{V}^{\pm}$.

Theorem 4: Let T be a Δ-tree, then $\Phi_\ell(T)$ is a Δ-tree without simple leaves and, in addition, $T \equiv \Phi_\ell(T)$, where the operator Φ_ℓ is defined as follows:

If the simple leaf η is labeled with $[\Theta]\ell$ and its immediate ancestor is labeled with $[\overline{\Theta}]\lambda$, then the simple leaf is erased, and its predecessor is relabeled as follows; if $\overline{\Theta} = \alpha$, then the new label is $[\overline{\Theta}]\bigl\lfloor\bigwedge\bigr\rfloor\{\lambda, \ell\}$, otherwise it is $[\overline{\Theta}]\bigl\lfloor\bigvee\bigr\rfloor\{\lambda, \ell\}$.

C. Updated Δ-trees

A useful property of the operator Δ-`Tree` is that, given a nnf A, in a Δ-`Tree`(A) the label of each $[\alpha]$ (resp. $[\beta]$) node is the Δ_0- (resp. Δ_1-)list associated to the subformula that it represents. However, this property need not hold when some transformation has already been applied on T.

Definition 7: Let T be a Δ-tree, and let η be a node of T that is neither a leaf nor the root. Let $[\Theta]\lambda$ be the label of the predecessor of η, and let $[\Theta]\lambda_1, \ldots, [\Theta]\lambda_n$ be the labels of its immediate successors. We say that η *can be updated* if it satisfies some of the next conditions:
1. Is labeled with $[\overline{\Theta}]$`nil` and $\bigcap_{i=1}^{n}\{\lambda_1, \ldots, \lambda_n\} \not\subset \lambda$.
2. Is labeled with $[\overline{\Theta}]\ell$ for some $\ell \in \mathcal{V}^{\pm}$ and satisfies both $\ell \notin \lambda$ and $\ell \in \bigcap_{i=1}^{n}\{\lambda_1, \ldots, \lambda_n\}$.

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

From the definition above, in order to obtain an updated Δ-tree, we have to drive upwards all those literals that can be generated by intersections; this operation is done by the operator `Update`.

Theorem 5: If T is a Δ-tree, then `Update`(T) is updated and, in addition, `Update`$(T) \equiv T$, where the operator `Update` is defined as follows:
1. If T is a leaf Δ-tree, then `Update`$(T) = T$.
2. If T is not a leaf Δ-tree, we traverse T in a reverse depth-first ordering (from the leaves to the root, and from right to left) until we get to a node that can be updated. Let η be such a node and let $[\Theta]\lambda$ be the label of its predecessor, and $[\Theta]\lambda_1, \ldots, [\Theta]\lambda_n$ the labels of its immediate successors.
(a) If η is labeled with $[\overline{\Theta}]$`nil`, and $\bigcap_i\{\lambda_1, \ldots, \lambda_n\} = \delta \not\subset \lambda$ then the predecessor of η is re-labeled. This new label is $[\Theta]\bigwedge\{\delta, \lambda\}$ if $\Theta = \alpha$, and $[\Theta]\bigvee\{\delta, \lambda\}$ otherwise.
(b) If η is labeled with $[\overline{\Theta}]\ell$, with $\ell \in \mathcal{V}^{\pm}$, $\ell \notin \lambda$ and $\ell \in \bigcap_{i=1}^{n}\{\lambda_1, \ldots, \lambda_n\}$ then the predecessor of η is re-labeled as follows; the new label is $[\Theta]\bigwedge\{\ell, \lambda\}$ if $\Theta = \alpha$, and with $[\Theta]\bigvee\{\ell, \lambda\}$ otherwise.

D. Restricted Δ-trees

Definition 8: Let T be a Δ-tree. If T is updated and it has neither 0-conclusive nodes nor 1-conclusive nodes nor simple leaves, then it is said to be *restricted*.

The operators defined in the previous sections allow us to transform every Δ-tree in another equivalent and restricted one.

From Theorems 3–5 we immediately obtain the following result.

Theorem 6: Let T be a Δ-tree, then `Restrict`(T) is restricted and, in addition, $T \equiv$ `Restrict`(T); where the operator `Restrict` traverses T in a reverse depth-first order (from leaves to the root, and from right to left) and in every node it tests whether the node is 0-conclusive, or 1-conclusive, or a simple leaf, or a node that can be updated, and in this case applies the corresponding operator in $\{\Phi_\bot, \Phi_\top, \Phi_\ell, \texttt{Update}\}$.

TABLE I
RUN TIME ON SOME IFIP BENCHMARKS.

Problem	Bea	TAS	Problem	Bea	TAS
`d3`	0.1	0.17	`vg2`	7.0	2.82
`misg`	0.7	0.35	`alu`	7.1	3.98
`ztwaalf1`	0.8	0.80	`x1dn`	7.2	3.37
`mp2d`	1.1	1.03	`z9sym`	9.8	4.07
`dk27`	2.2	0.07	`sqn`	11.2	0.43
`rom2`	2.5	3.03	`add1`	12.2	1.20
`table`	2.8	2.72	`dc2`	12.5	0.40
`dk17`	3.0	0.38	`mul03`	20.1	1.03
`z5xpl`	4.1	0.38	`rd73`	30.4	1.27
`f51m`	5.7	0.48	`root`	33.7	0.67
`pitch`	5.7	2.55	`alupla20`	618.1	31.72

V. Experimental results

Δ-trees have been used to implement an ATP using the reductions described in [2], [1], together with a branching rule based on the Davis-Putnam procedure (without using heuristic or probabilistic criteria for selecting the variable to branch the formula); namely, a formula A is split into two subformulas $A[p/\top]$ and $A[p/\bot]$, where p is the *first* variable occurring in A.

As our method is specially focused on non-cnf formulas we have run our prover, written in Objective CAML, on the IFIP benchmarks for hardware verification. The results obtained, using a Power Macintosh G3, are shown in table I, together with the results reported in [5] where a Sun Super SPARK workstation was used. It is noticeable the important speed-up obtained when using TAS.

References

[1] G. Aguilera, I. P. de Guzmán, M. Ojeda-Aciego, and A. Valverde, "Reductions for non-clausal theorem proving," *Theoretical Computer Science*, 2000. To appear

[2] G. Aguilera, I. P. de Guzmán, and M. Ojeda-Aciego, "Increasing the efficiency of automated theorem proving," *Journal of Applied Non-Classical Logics*, vol. 5, no. 1, pp. 9–29, 1995.

[3] I. P. de Guzmán, M. Ojeda-Aciego, and A. Valverde, "Multiple-valued tableaux with Δ-reductions," in *Proc. of ICAI'99*. 1999, pp. 177–183, C.S.R.E.A.

[4] I. P. de Guzmán, M. Ojeda-Aciego, and A. Valverde, "Implicates and reduction techniques for temporal logics," *Annals of Mathematics and Artificial Intelligence*, 1999, To appear.

[5] Fabio Massacci, "Simplification: a general constraint propagation technique for propositional and modal tableaux," in *Proceedings of Tableaux'98*. Lect. Notes in Artificial Intelligence 1397, 1998, pp. 217–231.