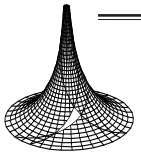

5/1 (2007), 109–118
tmcs@inf.unideb.hu
http://tmcs.math.klte.hu

Teaching
Mathematics and
Computer Science

Linear Clause Generation by Tableaux and DAGs

Gergely Kovásznai

Abstract. Clause generation is a preliminary step in theorem proving since most of the state-of-the-art theorem proving methods act on clause sets. Several clause generating algorithms are known. Most of them rewrite a formula according to well-known logical equivalences, thus they are quite complicated and produce not very understandable information on their functioning for humans. There are other methods that can be considered as ones based on tableaux, but only in propositional logic. In this paper, we propose a new method for clause generation in first-order logic. Since it inherits rules from analytic tableaux, analytic dual tableaux, and free-variable tableaux, this method is called *clause generating tableaux* (CGT). All of the known clause generating algorithms are exponential, so is CGT. However, by switching to *directed acyclic graphs* (DAGs) from trees, we propose a *linear* CGT method. Another advantageous feature is the detection of valid clauses only by the closing of CGT branches. Last but not least, CGT generates a graph as output, which is visual and easy-to-understand. Thus, CGT can also be used in teaching logic and theorem proving.

Key words and phrases: discrete mathematics, logic, graph theory, proof methods, artificial intelligence (theorem proving).

ZDM Subject Classification: N75, E35, K35, E55, R45.

1. Introduction

Most of the state-of-the-art theorem proving methods act on clause sets. Some of them are resolution-based, like binary resolution [10] and hyper-resolution [11], but there are other ones that are based on tableaux, like clause tableaux [4], hyper tableaux [1], and multi-hyper (hyperS) tableaux [5]. Therefore, techniques

for generating clauses from arbitrary first-order formulas are required. Most of the clause generating algorithms rewrite a formula according to well-known logical equivalences, like the de Morgan rules, the rules for converting implication to disjunction, and the rules of distributivity. A survey of such algorithms can be found in [6]. As rare exceptions, there are other alternative methods. Some of them are based on BDDs (Binary Decision Diagrams), also known as Shannon graphs [6, 9]. These algorithms use BDDs as intermediate forms, i.e., a formula is converted to a BDD at first, and then this latter one is converted to a set of clauses. BDDs can basically be used in propositional logic, but they increasingly attract attention in first-order logic in these days. Logicians often draw a parallel between BDDs and tableaux [9], they are compared in clause generation and in theorem proving.

Similarly, a parallel can be drawn between tableaux and clause generating algorithms based on generalized conjunction and disjunction, the dual forms of formulas, and embedded lists [3, 7]. However, these methods can only be used in propositional logic, either. Such an algorithm can convert a formula to either disjunctive normal form (DNF) or conjunctive normal form (CNF); this latter one is called clause normal form as well. For instance, in the case of DNF it generates a list $[Q_1, \ldots, Q_m]$ where each Q_i is a list $\langle L_1, \ldots, L_k \rangle$ where each L_j is a literal. That is, a list $\langle \ldots \rangle$ represents the conjunction of its elements, and a list $[\ldots]$ represents the disjunction of its elements. Consider that the list $[\langle \ldots \rangle, \ldots, \langle \ldots \rangle]$ constructed as DNF can be considered as an analytic tableau (AT) [13]. In the case of CNF, a list $\langle [\ldots], \ldots, [\ldots] \rangle$ is constructed and can be considered as an analytic dual tableau (ADT)[1]. The connection between tableaux and normal forms is well-known in literature, but only in propositional logic.

In this paper, we propose a novel algorithm for clause generation in first-order logic, in Section 2. That algorithm is based on tableaux, and is exponential. In Section 3, that algorithm is linearized by the use of DAGs (Directed Acyclic Graphs). In Section 4, a simple technique is introduced for simplifying the clause set generated by the latter method. In the followings, we assume that the reader is familiar with the basic concepts of first-order logic.

[1]The rules of ADT can be formed by transposing the rules of AT: exchanging the α-rule and the β-rule, and exchanging the γ-rule and the δ-rule.

2. Clause Generation by Tableaux

Unfortunately, ADT cannot directly be used for generating CNF (i.e., clauses) in first-order logic. It is prevented by the γ-rule and the δ-rule. As compared to propositional logic, the definition of satisfiability is supplemented with the concept of assignment, which prevents us from considering satisfiability (in AT) to be the negation of unsatisfiability (in ADT) only by taking model generation for basis. Thus, the adequate tableau rules for clause generation:

- the γ-rule and the δ-rule of AT, and
- the α-rule and the β-rule of ADT.

Besides, existential quantifiers must be eliminated by introducing Skolem-functions [12]. Furthermore, universal quantifiers must be taken to the beginning of the formula by introducing new parameters, like in free-variable tableaux [3]. These transcriptions have to be done on the fly. The required tableau rules can be seen in Figure 1.

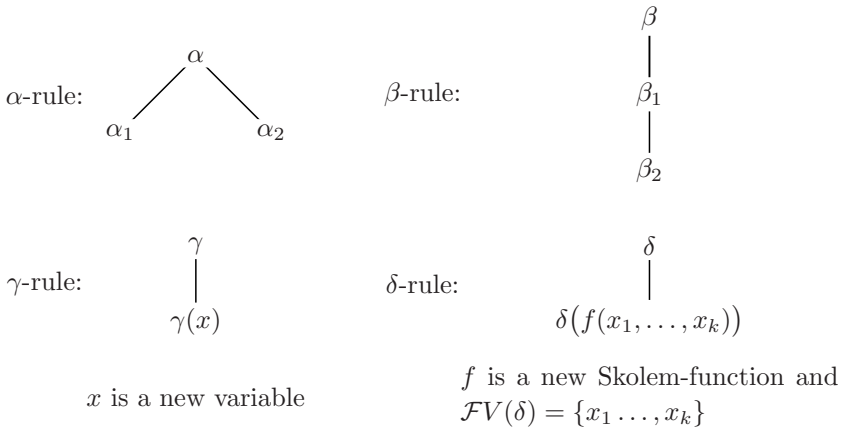

Figure 1. Clause generating tableau – Rules

Definition 1. Let A be a formula. A *clause generating tableau* (CGT) for A is defined inductively as follows:

(1) A tableau consisting of one single node labeled by A is a CGT for A.

(2) If $\mathcal{T}$ is a CGT for A then so is $\mathcal{T}'$ where $\mathcal{T}'$ is the result of applying any rule in Figure 1 to $\mathcal{T}$.

In contrast with tableaux in theorem proving, it is sufficient to apply a rule only once to the same node[2]. In order to force this, erase each label to which a rule has already been applied. A CGT is called *finished* iff all of its labeled nodes are labeled by literals. It should be proven that the clause set specified by a CGT for a formula A is satisfiable iff so is A. The following definition and lemma pave the way for that proof.

DEFINITION 2. Let us define the function $cnf()$ on branches and tableaux, as follows:

(1) If $\mathcal{B}$ is a branch consisting of the formulas $A_1, \ldots, A_k$ $(k \geq 1)$ then

$$cnf(\mathcal{B}) \quad = \quad (A_1 \vee \ldots \vee A_k)$$

(2) If $\mathcal{T}$ is a tableau consisting of the branches $\mathcal{B}_1, \ldots, \mathcal{B}_k$ $(k \geq 1)$ then

$$cnf(\mathcal{T}) \quad = \quad cnf(\mathcal{B}_1) \wedge \ldots \wedge cnf(\mathcal{B}_k)$$

LEMMA 3. *Let $\mathcal{T}$ be a CGT. $cnf(\mathcal{T})$ is satisfiable iff $cnf(\mathcal{T}')$ is also satisfiable where $\mathcal{T}'$ is the result of applying a CGT rule to $\mathcal{T}$.*

PROOF. Let the proof divide in two according to the rule applied.

(1) *If either the α-rule, the β-rule, or the γ-rule is applied:*
Suppose $cnf(\mathcal{T})$ to be satisfiable. Since there is a model $\mathcal{M}$ such that $\mathcal{M} \models cnf(\mathcal{T})$, for all branches $\mathcal{B}$ of $\mathcal{T}$ $\mathcal{M} \models cnf(\mathcal{B})$. When applying a rule, only one branch changes, but the other ones remain unchanged. It is sufficient to focus on the branch that changes, which is denoted by $\mathcal{B}$ and regarded as a set of its formulas in the followings. Similarly, let $\mathcal{T}$ be regarded as a set of its branches.
   (a) If $\mathcal{B} = X \cup \{(A \vee B)\}$, i.e., $\mathcal{M} \models cnf(X) \vee (A \vee B)$:
   The β-rule is applied to $(A \vee B)$, which results in branch $\mathcal{B}' = X \cup \{A, B\}$.
   Since $cnf(\mathcal{B}') = cnf(X) \vee A \vee B$,

$$\mathcal{M} \models cnf(\mathcal{B}) \text{ iff } \mathcal{M} \models cnf(\mathcal{B}').$$

   (b) If $\mathcal{B} = X \cup \{(A \wedge B)\}$, i.e., $\mathcal{M} \models cnf(X) \vee (A \wedge B)$:

[2]For a complete calculus in first order logic, it must be allowed to apply the γ-rule of AT and the δ-rule of ADT several times. The method of CGT is not a calculus.

The α-rule is applied to $(A \wedge B)$, which results in branches $\mathcal{B}'_1 = X \cup \{A\}$ and $\mathcal{B}'_2 = X \cup \{B\}$.
Since $cnf(\mathcal{B}'_1) = cnf(X) \vee A$ and $cnf(\mathcal{B}'_2) = cnf(X) \vee B$,

$$\mathcal{M} \models cnf(\mathcal{B}) \text{ iff } \mathcal{M} \models cnf(\mathcal{B}'_1) \wedge cnf(\mathcal{B}'_2).$$

(c) If $\mathcal{B} = X \cup \{\forall x A(x)\}$, i.e., $\mathcal{M} \models cnf(X) \vee \forall x A(x)$:
The γ-rule is applied to $\forall x A(x)$, which results in branch $\mathcal{B}' = X \cup \{A(y)\}$.
The variable y is new, i.e., $y \notin \mathcal{FV}(\mathcal{T})$. Thus, $y \notin \mathcal{B}$ and $y \notin X$.
Since $cnf(\mathcal{B}') = cnf(X) \vee A(y)$,

$$\mathcal{M} \models cnf(\mathcal{B}) \text{ iff } \mathcal{M} \models cnf(\mathcal{B}').$$

(2) *If the δ-rule is applied,* i.e., there is a branch $\mathcal{B} \in \mathcal{T}$ such that $\mathcal{B} = X \cup \{\exists x A(x)\}$:
The δ-rule is applied to $\exists x A(x)$, which results in branch $\mathcal{B}' = X \cup \{A(f(x_1, \ldots, x_k))\}$. According to Skolem [12], a formula is satisfiable iff it remains also satisfiable after eliminating one of its existential quantifiers by introducing a new Skolem-function. Thus, $cnf(\mathcal{T})$ is satisfiable iff so is $cnf(\mathcal{T}')$.

□

THEOREM 4. *Let A be a formula, and let $\mathcal{T}$ be a CGT for A. A is satisfiable iff so is $\mathcal{C} = \{cnf(\mathcal{B}) \mid \mathcal{B} \in \mathcal{T}\}$.*

PROOF. By Lemma 3, A is satisfiable iff so is $cnf(\mathcal{T})$. And by the definition of $\mathcal{C}$, $cnf(\mathcal{T})$ is satisfiable iff so is $\mathcal{C}$. □

Consider that if $\mathcal{T}$ is finished then $\mathcal{C}$ is a clause set. I.e., the branches of a finished CGT represent the required clauses.

3. Linear algorithm by DAGs

All the known clause generating algorithms are exponential, which is caused by the rewritings according to distributivity. The worst case is the one when the original formula is in DNF. In this case, the number of the generated clauses is l^c at most where c is the number of disjuncts in the DNF, and l is the maximal number of literals within those disjuncts. Of course, that number cannot be reduced. Nevertheless, the number of steps of the generation can be.

CGT is exponential, too. The problem is that some formulas are duplicated in the CGT, so they must be processed separately, although they are the same. This problem originates in distributivity and manifests itself in the α-rule. It is easy to consider that the tree as data structure causes the problem since a tree cannot contain a node with a degree greater than 1. This latter restriction should be eliminated, i.e., directed acyclic graphs (DAGs) are needed instead of trees. DAGs are well-known for logicians. Robinson's exponential unification algorithm was linearized by Paterson and Wegman in the 70s, by the use of DAGs [8]. The key idea was the replacing of all occurrences of a variable by references to the original variable. I.e., a variable is stored and substituted only in one spot, and not in all of its occurrences. DAGs are widely used in logical applications where duplications are to be prevented. Moreover, DAGs are used not only for the sake of efficiency: e.g., the relation between worlds in Kripke's modal semantics can be represented quite naturally by DAGs, as considered by Castilho et al. [2], who constructed their own modal tableaux based on DAGs.

In the case of CGT, the introduction of DAGs is motivated for the sake of efficiency, and the fact that the resulting CGT produces a more expressive output. As Paterson and Wegman linearized the unification algorithm by changing the data structure only, so do we in the case of CGT. As mentioned above, the problematic α-rule has to be changed, as shown in Figure 2. It can be seen that every junction is immediately closed in an unlabeled node.

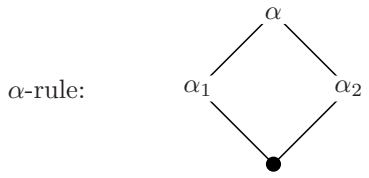

Figure 2. Clause generating DAG – α-rule changed

The way that the rules are applied is also important. The traditional way of applying tableau rules to a node N is that the new formulas are appended at the end of each branch containing N. Now, in the case of CGT they get inserted right below N. Consider that such a CGT is really a DAG and always contains exactly one leaf. This algorithm is obviously linear in terms of the number of logical connectives in the original formula.

In Figure 3, a finished CGT as a DAG can be seen which has been generated for the formula in the root.

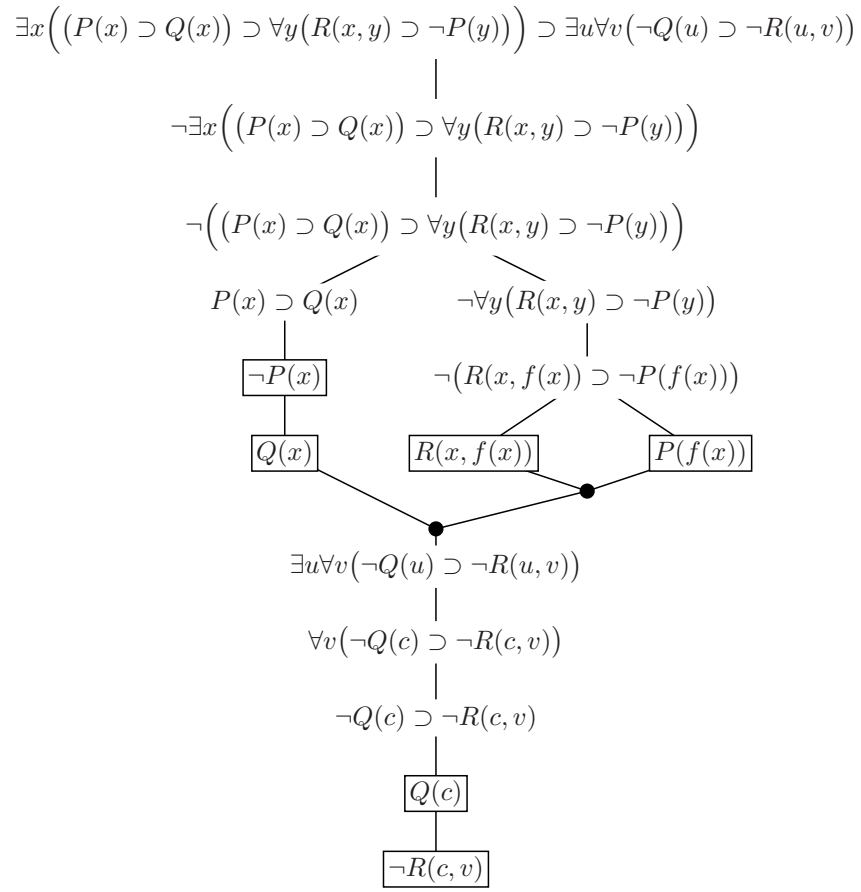

Figure 3. Example for CGT as DAG

Consider that if a formula B was the result of applying a rule to a formula A then B is located right below A. Two Skolem-functions were introduced: f and c. In the figure, we have not erased the compound formulas, but have framed the nodes that are really located in the finished CGT. These latter nodes (literals) form the required clauses:

$$\begin{aligned} \neg P(x) \vee Q(x) \vee Q(c) \vee \neg R(c,v) \\ R(x,f(x)) \vee Q(c) \vee \neg R(c,v) \\ P(f(x)) \vee Q(c) \vee \neg R(c,v) \end{aligned}$$

4. Simplifying Clause Set by Closure

In the generated clause set the clauses that are valid can be eliminated since they do not have an effect on the satisfiability of the clause set. It is quite easy to detect valid clauses in a CGT. In an ADT a branch can be proven to be valid by closure. So can it be in a CGT, since it inherits the α-rule and the β-rule from ADT. If a branch of a finished CGT can be closed, then that branch as a clause does not get in the generated clause set.

The closing of CGT branches can be done as in the MGU Atomic Closure Rule in [3], i.e., by generating most general unifiers (MGUs), as follows:

(1) Looking for an MGU for two oppositely negated literals in the branch.

(2) If such an MGU exists then the given branch (as a clause) does not get in the generated clause set.

In the case of the finished CGT in Figure 3 the clause $R(x, f(x)) \vee Q(c) \vee \neg R(c, v)$ does not get in the clause set, since the branch consisting of its literals can be closed by the MGU $\{x/c, v/f(c)\}$, which is in fact the MGU of the oppositely negated literals $R(x, f(x))$ and $\neg R(c, v)$.

5. Conclusion

In this paper, a novel algorithm for clause generation is proposed and is called clause generating tableaux (CGT). CGT is based on tableaux, namely on both analytic tableaux and analytic dual tableaux. CGT is exponential like all the close generating algorithms, but it can easily be linearized by switching to DAGs from trees. Another advantage of using tableaux is that valid clauses can be eliminated very easily only by detecting the closure of branches. CGT is a very clear, simple, and easy-to-implement algorithm for clause generation, and it is linear in contrast with all the known algorithms for the same purpose.

At the University of Debrecen, theorem proving was introduced to education in 1994. Currently, in the Bologna process, the following courses are held in connection with theorem proving:

- *"The Logical Bases of Informatics"*: In this compulsory and foundation course, students acquire the basics of the classical propositional and first-order logic (syntax and semantics), logical laws, normal forms, etc.

- *"The Basics of Artificial Intelligence"*: Logical puzzles are represented by graphs, and are solved with numerous graph search methods (e.g., backtracking, deep-first, breadth-first, heuristic).
- *"Automated Theorem Proving"*: Analytic tableaux and resolution are introduced in propositional logic, as well as several resolution strategies. In first-order logic, after useful methods for automating tableaux and resolution are proposed (e.g., unification and skolemization), first-order tableaux and first-order resolution are introduced.
- *"Logic Programming"*: Linear input resolution and Prolog. Several puzzles are represented by clause sets, solved with resolution, and implemented in Prolog by students.

Since clause generation is already introduced in the foundation course "The Logical Bases of Informatics", and is essential in the courses "Automated Theorem Proving" and "Logic Programming", it is worth to show a linear and easy-to-implement algorithm for this problem to students (besides the method in [6]). Especially as students get acquainted with graph representation and graph methods in the course "The Basics of Artificial Intelligence", teaching the CGT method in the courses "Automated Theorem Proving" and/or "Logic Programming" is quite useful, since

- it is a simple and linear algorithm for clause generation (i.e., it performs a preliminary task for resolution);
- it is based on analytic tableaux;
- it checks closure of branches and uses unification in order to eliminate valid clauses;
- and it produces a DAG as output, which is graphic, and therefore very expressive for humans.

References

[1] P. Baumgartner, U. Furbach, I. Niemelä, Hyper Tableaux, *Lecture Notes in Computer Science* **1126** (1996), 1–17.

[2] M. A. Castilho, L. F. del Cerro, O. Gasquet, A. Herzig, Modal Tableaux with Propagation Rules and Structural Rules, *Fundamenta Informaticae* **32** (1997), 281–297.

[3] M. Fitting, *First-Order Logic and Automated Theorem Proving*, Springer-Verlag, 1996.

[4] R. Hähnle, Tableaux and Related Methods, in: *Handbook of Automated Reasoning*, Vol. 1, Chapter 3, (J. A. Robinson and A. Voronkov, eds.), Elsevier and MIT Press, 2001, 100–178.

[5] G. Kovásznai, HyperS Tableaux – Heuristic Hyper Tableaux, *Acta Cybernetica* **17** (2005), 325–338, ZBL#1099.68096, MR#2183822.

[6] A. Nonnengart, C. Weidenbach, Computing Small Clause Normal Forms, in: *Handbook of Automated Reasoning*, Vol. 1, Chapter 6, (J. A. Robinson and A. Voronkov, eds.), Elsevier and MIT Press, 2001, 335–367.

[7] K. Pásztorné Varga, M. Várterész, A Generalized Approach to the Theorem Proving Methods, 5^{th} *International Conference on Applied Informatics*, Hungary (2001), 191–200.

[8] M. S. Paterson, M. N. Wegman, Linear Unification, *Annual ACM Symposium on Theory of Computing* (1976), 181–186.

[9] J. Posegga, P. H. Schmitt, Automated Deduction with Shannon Graphs, *Journal of Logic and Computation* **5** (1995), 697–729.

[10] J. A. Robinson, A Machine-Oriented Logic Based on the Resolution Principle, *Journal of the ACM* **12** (1965), 23–41.

[11] J. A. Robinson, Automated Deduction with Hyper-Resolution, *International Journal of Computer Mathematics* **1** (1965), 227–234.

[12] T. Skolem, Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit matematischer Sätze nebst einem Theoreme über dichte Mengen, *Videnskabsakademiet i Kristiania, Skrifter I*, no. 4 (1919), 1–36.

[13] R. M. Smullyan, *First-Order Logic*, Springer-Verlag, 1968.

GERGELY KOVÁSZNAI
UNIVERSITY OF DEBRECEN
FACULTY OF INFORMATICS
H–4010 DEBRECEN
HUNGARY
P.O. BOX 12

E-mail: kovasz@inf.unideb.hu

(Received November, 2006)