Proof Procedure based on Modified Analytic Tableaux

Hajime SAWAMURA*, Takashi MAEDA**
and Michiaki KAWAGUCHI*

(Received June 30, 1976)

Abstract

A formal, mechanical approach to theorem-proving appears to be the most promising from a standpoint of realizing the need for inference in the field of deductive science.

In this paper, we have described an effective and flexible proof procedure in mechanical theorem-proving. The modified analytic tableaux is a variant of the "analytic tableaux" of Smullyan. This method is compatible with the resolution method, which forms the basis for probably all contemporary theorem-provers for predicate calculus. Theoretically, the proposed proof procedure can be considered as an extension of the resolution method.

Also, it was proved that this proof procedure is complete, and several applications of this method are discussed.

1. Introduction

A formal, mechanical approach to theorem-proving appears to be the most promising from a standpoint of realizing the need for inference in the field of deductive science. For, many problems which need inference, i. e., deduction, induction, etc., for example, deductive question answering, problem solving, program analysis, program synthesis, and many others, can be formulated as theorem-proving. And furthermore, the method of mechanical theorem-proving may be used as a mathematician's assistant, in other words, the proof checking techniques and proof finding techniques in mechanical theorem-proving will allow the mathematician to try out his ideas for proofs that are still quite vague, and may speed up mathematical work.

Since 1960s or thereabouts, the proof procedures in mechanical theorem-proving based on the formal system, which is called first-order predicate calculus (for short, first-order logic), has been studied theoretically, and has been implemented by digital computers. The different effective proof procedures for first-order logic exist because it has been most widely explored in mathematical logic. These procedures may be divided into two different types. One is to use techniques developed within the framework of the Herbrand theorem which was

* Department of Information Engineering, Graduate School of Engineering, Hokkaido University, Sapporo, Japan.

** Department of Engineering Science, Faculty of Engineering, Hokkaido University, Sapporo, Japan.

proved by Herbrand in 1930. Given axioms $\{A_1, \cdots, A_n\}$ and a formula B, rather than trying to find a proof of B from $\{A_1, \cdots, A_n\}$, we try to show that there can be no interpretation of $\{A_1, \cdots, A_n, \sim B\}$. Methodologically although the essential idea is to reduce the framework of the search from formulas in the first-order logic to formulas which can be treated to belong to propositional logic, in other words, it suggests a way to avoid the impossibility of enumerating all models, if we are to use the Herbrand theorem directly, we would face two immediate difficulties. The first is how to choose the formulas which can be treated to belong to propositional logic, in such a way as to lead as quickly as possible to a contradiction. The second is the size of the truth tables that must be computed. Numerous investigations in mechanical theorem-proving from a Herbrand point of view have been attempted to overcome these two difficulties[1]. A major breakthrough was made by J. A. Robinson[2] in 1965, who introduced the resolution principle, a single inference rule to which the syllogism principle of propositional logic, which was called resolution, and the instantiation principle of first-order logic, which was called factoring, were combined. Since the introduction of resolution principle, many refinements for the resolution principle have been made in an attempt to further increase its efficiency[1]. The other is to use the Gentzen-like formal system which is based on NK (Natürlicher Kalkül) or LK (Logistischer Kalkül)[3]. In Gentzen-type proof procedure, we actually try to produce proof, namely we try to derive proof by backward reasoning to axioms from the theorem to be proved. It is also used to describe the semantics for programming language in the logic of programs because of its natural, effective notation.

At present, the mechanizations of inferences are studied in various directions, using higher-order logic, modal logic, whose intent is to provide a more natural language for the expression of problems, extending the expressive power of logical language, and special-purpose deduction systems that appear quite promising for program verification.

In this paper, our purpose is to formulate a machine-oriented proof procedure for proving theorems which will answer some reasonably complicated questions before overstepping the bounds of available time and memory space as a certain possibility of proof procedure in mechanical theorem-proving.

The remainder of this work is divided into three chapters. Chapter 2 describes the rules for the construction of modified analytic tableaux, containing the description of the language to be used in a proof procedure of first-order logic. Chapter 3 contains the proof procedure based on modified analytic tableaux and its completeness. In chapter 4, we discuss the main features of this method in relation to other proof procedures and describe the possibilities of application to the interesting problems with examples.

2. Formal System

Our formal system consists of the following; first-order language L, inter-

pretation of L, rules for the construction of modified analytic tableaux.

2.1 First-order language

2.1.1 A first-order language has the following symbols:

1) Variables
   a) bound variables $x, y, z, x_1, y_1, z_1, \cdots$;
   b) free variables $a, b, c, a_1, b_1, c_1, \cdots$.
2) For each n, the n-ary function symbols and the n-ary predicate symbols (0-ary function symbol is called a constant).
3) The symbols $\sim, \vee, \&, \rightarrow, \forall, \exists, (,), \{,\}, [,], ,.$

We use syntactical variables[4]. They vary through the expressions of the language being discussed. Thus a syntactical variables may mean any expression of the language. We shall use German letters as syntactical variables. In particular, $\mathfrak{u}$, $\mathfrak{v}$ will be syntactical variables which vary through all expression, and we use as syntactical variables:

1) $\mathfrak{x}, \mathfrak{y}, \mathfrak{z}, \cdots$, which vary through bound variables;
2) $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \cdots$, which vary through free variables;
3) $\mathfrak{f}, \mathfrak{g}, \mathfrak{h}, \cdots$, which vary through function symbols;
4) $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \cdots$, which vary through predicate symbols;
5) $\mathfrak{e}$, which varies through constants.

2.1.2 Terms are defined recursively as follows:

1) A variable is a term.
2) If $\mathfrak{u}_1, \cdots, \mathfrak{u}_n$ are terms and $\mathfrak{f}$ is n-ary, then $\mathfrak{f}(\mathfrak{u}_1, \cdots, \mathfrak{u}_n)$ is a term (a constant is a term).
3) The terms consist exactly of those expressions generated by 1) and 2).

We use $\mathfrak{t}_1, \cdots, \mathfrak{t}_n$ as syntactical variables which vary through terms.

2.1.3 An atomic formula is an expression of the form $\mathfrak{P}(\mathfrak{t}_1, \cdots, \mathfrak{t}_n)$, where $\mathfrak{P}$ is n-ary.

2.1.4 Well-formed formulas (wff), or formulas for short are defined recursively as follows:

1) An atomic formula is a formula.
2) If $\mathfrak{u}$, $\mathfrak{v}$ are formulas, then $\sim\mathfrak{u}$, $\mathfrak{u}\vee\mathfrak{v}$, $\mathfrak{u}\&\mathfrak{v}$, $\mathfrak{u}\rightarrow\mathfrak{v}$ are formulas.
3) If $\mathfrak{u}$ is a formula, then $(\forall\mathfrak{x})\mathfrak{u}$ and $(\exists\mathfrak{x})\mathfrak{u}$ are formulas.
4) The formulas consist exactly of those expressions generated by 1), 2), and 3).

A first-order language is now defined to be a language in which the symbols and formulas are described above. Next, we let $\mathfrak{b}_{\mathfrak{x}_1,\cdots,\mathfrak{x}_n}(\mathfrak{t}_1, \cdots, \mathfrak{t}_n)$ designate the term obtained from $\mathfrak{b}$ by replacing all free occurrences of $\mathfrak{x}_1, \cdots, \mathfrak{x}_n$ by $\mathfrak{t}_1, \cdots, \mathfrak{t}_n$ respectively, and we let $\mathfrak{A}_{\mathfrak{x}_1,\cdots,\mathfrak{x}_n}(\mathfrak{t}_1, \cdots, \mathfrak{t}_n)$ designate the formula $\mathfrak{A}$ by replacing all free occurrences of $\mathfrak{x}_1, \cdots, \mathfrak{x}_n$ by $\mathfrak{t}_1, \cdots, \mathfrak{t}_n$, but we shall omit the subscripts $\mathfrak{x}_1, \cdots, \mathfrak{x}_n$ when they are immaterial or clear from the context.

2.2 Interpretation

We now turn to a description of the semantics of the first-order language.

2. 2. 1 An interpretation I for L consists of the following:

1) A nonempty set U, called the universe of I. The elements of U are called the individuals of I.

2) For each n-ary function symbol $\mathfrak{f}$ of L, an n-ary function $\mathfrak{f}_I$ from U^n to U (in particular, for each constant $\mathfrak{e}$ of L, $\mathfrak{e}_I$ is an individual of I).

3) For each n-ary predicate symbol $\mathfrak{P}$ of L, an n-ary predicate $\mathfrak{P}_I$ in U^n.

We note that any formula containing free variables cannot be evaluated by the above interpretation. Thus, we shall assume either that formulas do not contain free variables (the closed formula), or that free variables are treated as constants.

2. 2. 2 Validity and Satisfiability

Given a well-formed formula and an interpretation I, we can determine whether a formula is true or not by using the usual truth tables for $\sim$, $\vee$, &, and $\rightarrow$, and interpreting the universally and existentially quantified variables in the standard way.

A formula is called satisfiable if it is true in at least one interpretation in at least one universe, otherwise unsatisfiable. A formula is called valid if it is true under every interpretation (in every universe). Clearly, $\mathfrak{A}$ is valid if and only if $\sim\mathfrak{A}$ is unsatisfiable.

Here, we must mention the validity problem of the first-order logic. Actually, finding a general decision procedure to verify the validity (unsatisfiability) was considered long ago. But, Church and Turing independently showed that there is no general decision procedure (algorithm) to check the validity of formulas of the first-order logic, i. e., the validity problem of the first-order logic is undecidable. However, there are proof procedures which can verify that a formula is valid if indeed it is valid, i. e., the validity problem of the first-order logic is semidecidable. For satisfiable formulas, these procedures in general will never terminate. In view of the result of Church and Turing, this is the best we can expect to obtain from a proof procedure.

2. 3 Construction rules of tableaux

We restrict the form of formula to be proved to the prenex normal form with no existential quantifiers, without loss of generality. That is, it is a well-known result of quantification theory that any formula can be put into prenex normal form with no existential quantifiers by standardizing variables, eliminating existential quantifiers by the introduction of Skolem functions, moving all universal quantifiers to the front of the formula. Strictly speaking, this is guaranteed by the following theorem:

Theorem 2–1 (Skolem's theorem). Every wff $\mathfrak{A}$ can be transformed into a wff $\mathfrak{A}'$ in prenex normal form and that existential quantifiers in the prenex formula can be replaced by function symbols (Skolem function) in such a way that $\mathfrak{A}$ is satisfiable if and only if $\mathfrak{A}'$ is satisfiable.

After all, our rules for the construction of tableaux are applied only to a set of formulas of the form

$$(\forall \mathfrak{x}_1) \cdots (\forall \mathfrak{x}_n)\ \mathfrak{A}\,,$$

where $\mathfrak{x}_i \neq \mathfrak{x}_j$, for $i \neq j$, and $\mathfrak{A}$ is a quantifier free formula (matrix).

A modified analytic tableaux for aformula to be proved is an ordered dyadic tree, whose points are formulas, which is constructed by the following rules:

1) $\dfrac{\sim\sim\mathfrak{A}}{\mathfrak{A}}$

2) $\dfrac{\mathfrak{A}\,\&\,\mathfrak{B}}{\begin{array}{l}\mathfrak{A}\\ \mathfrak{B}\end{array}}$ $\qquad$ $\dfrac{\sim(\mathfrak{A}\,\&\,\mathfrak{B})}{\sim\mathfrak{A}\mid\sim\mathfrak{B}}$

3) $\dfrac{\mathfrak{A}\vee\mathfrak{B}}{\mathfrak{A}\mid\mathfrak{B}}$ $\qquad$ $\dfrac{\sim(\mathfrak{A}\vee\mathfrak{B})}{\begin{array}{l}\sim\mathfrak{A}\\ \sim\mathfrak{B}\end{array}}$

4) $\dfrac{\mathfrak{A}\rightarrow\mathfrak{B}}{\sim\mathfrak{A}\mid\mathfrak{B}}$ $\qquad$ $\dfrac{\sim(\mathfrak{A}\rightarrow\mathfrak{B})}{\begin{array}{l}\mathfrak{A}\\ \sim\mathfrak{B}\end{array}}$

5) $\dfrac{(\forall \mathfrak{x})\,\mathfrak{A}}{\mathfrak{A}(\mathfrak{t})}$, where t is any term.

3. Modified Analytic Tableaux

In this chapter, we give a combined proof procedure of resolution method and analytic tableaux method, which is called modified analytic tableaux[5], and prove its completeness. Tableaux method, which R. M. Smullyan terms[6] "analytic tableaux", is a variant of the "semantic tableaux" of Beth, or of methods of Hintikka. Ultimately, the whole idea is derived from Gentzen and Herbrand.

3.1 Proof procedure based on modified analytic tableaux

In order to construct a proof procedure based on modified analytic tableaux, we describe some definitions and theorem without proof. We use the letter "α" to stand for any formula of conjunctive type -i. e. one of the four forms $\mathfrak{A}\,\&\,\mathfrak{B}$, $\sim(\mathfrak{A}\vee\mathfrak{B})$, $\sim(\mathfrak{A}\rightarrow\mathfrak{B})$, $\sim\sim\mathfrak{A}$, the letter "β" to stand for any formula of disjunctive type -i. e. one of the three forms $\mathfrak{A}\vee\mathfrak{B}$, $\sim(\mathfrak{A}\,\&\,\mathfrak{B})$, $\mathfrak{A}\rightarrow\mathfrak{B}$. And we refer to α_1 as the first component of α and α_2 as the second component of α, similarly for β. We use "γ" to denote any formula of the form $(\forall \mathfrak{x})\,\mathfrak{A}$ and for any term t, by $\gamma(\mathfrak{t})$ we mean $\mathfrak{A}_{\mathfrak{x}}(\mathfrak{t})$.

We now classify the rules in section 2. 3 into three groups for simplicity.

Rule A: $\dfrac{\alpha}{\begin{array}{l}\alpha_1\\ \alpha_2\end{array}}$

Rule B: $\dfrac{\beta}{\beta_1 \mid \beta_2}$

Rule C: $\dfrac{\gamma}{\gamma(t)}$, with proviso.

3. 1. 1 A tableaux is called closed if every branch is an ordered dyadic tree which has two formulas, one of which is the negation of the other, otherwise it is open.

Theorem 3-1 (Unification theorem). Let S be any finite nonempty set of well-formed formulas. If S is unifiable, then S is most generally unifiable with most general unifier σ; moreover, for any unifier μ of S there is a substitution λ such that $\mu = \sigma\lambda$.

Proof procedure: We start the tableaux by placing the formulas to be proved at the origin. Next, our tree generation procedure is divided into two phases. The first phase is to use only the quantification rule C because our initial formula is prenex normal form with no existential quantifiers. Although universally quantified variables may be replaced by any term, in order to apply the unification algorithm after having generated the tree to some extent, it is necessary to rename all bound variables occurring in a set of formulas to be proved so that no bound variables occurring in any two formulas of the set have the same name, and we simply eliminate the universal quantifiers prefixed to the formulas. Furthermore, we consider the resulting set of formulas as if each element of the set were saturated by any term. In this phase, the tableaux has only one branch since we do not use rule B. The second phase is to use the rule A, B. Assuming that we have concluded the n-th stage. Then our next act is determined as follows. If the tableaux already is closed by applying the unification algorithm to the set of formulas on the each branch of generated tree (note that all bound variables with same name on every branch of the tree should be replaced by the same term simultaneously), or every open branch of the tableaux has been used, then we stop. If neither, then we pick a point X which has not yet been used and extend the tableaux as follows:
We take every open branch θ passing through the point X, and

1) If X is an α, we extend θ to the branch $(\theta, \alpha_1, \alpha_2)$.
2) If X is a β, then we simultaneously extend θ to the two branches (θ, β_1) and (θ, β_2).

This concludes the stage $n+1$ of our procedure.

3. 1. 2 By a finished tableaux we mean a modified analytic tableaux which is either infinite, or else finite but can not be extended further by continuing the above procedure.

3. 2 Completeness theorem

In this section, we prove the completeness theorem of the proof procedure based on the modified analytic tableaux.

We begin by noting that under any interpretation the following three facts

hold. Assuming that S is any set of formulas.

1) If S is satisfiable and $\alpha \in S$, then (S, α_1, α_2) is satisfiable.

2) If S is satisfiable and $\beta \in S$, then at least one of the two sets (S, β_1), (S, β_2) is satisfiable.

3) If S is satisfiable and $\gamma \in S$, then for every term t, the set $(S, \gamma(t))$ is satisfiable.

We should note that the fact 3) is evaluated by substituting individuals under any interpretation to the bound variables occurring in the terms. In what follows, we define a version of Hintikka set and prove a version of Hintikka's lemma for the purpose of our formulation of proof procedure.

3. 2. 1 By Hintikka set we mean a set of H such that the following conditions hold for every α, β, γ.

1) No atomic element and its negation are both in H.

2) If $\alpha \in H$, then α_1, α_2 are both in H.

3) If $\beta \in H$, then $\beta_1 \in H$ or $\beta_2 \in H$.

4) If $\gamma \in H$, then for every term t, $\gamma(t) \in H$.

Lemma (Hintikka's lemma). Every Hintikka set H is satisfiable.

Proof. We must find an interpretation in which all elements of H are true. For every atomic formula $\mathfrak{P}(t_1, \cdots, t_n)$, give it the value "true" if $\mathfrak{P}(t_1, \cdots, t_n)$ is an element of H, "false" if $\sim\mathfrak{P}(t_1, \cdots, t_n)$ is an element of H. We must show that each element $\mathfrak{A}$ of H is true under this atomic interpretation. We do this by induction on the degree of $\mathfrak{A}$. If $\mathfrak{A}$ is of degree 0, it is immediate. Assuming that $\mathfrak{A}$ is of positive degree and that every element of H of lower degree is true. Since $\mathfrak{A}$ is not of degree 0, then it is either α, β or γ. If it is an α or β, then it is true since if it is an α, then α_1, α_2 are both in H, hence it would be true by induction hypothesis, hence α is true; if it is a β, then at least one of β_1, β_2 is in H, and hence true, so β is true. Assuming that it is a γ, then for every term t, $\gamma(t) \in H$ by 4), but every $\gamma(t)$ is of a lower degree than γ, hence true by induction hypothesis. Hence γ must be true.

As easily seen, for any finished tableaux, every open branch is a Hintikka sequence (for denumerable set of terms). From this and Hintikka's lemma, the following theorem holds[6]:

Theorem 3–2. In any finished tableaux T, every open branch is simultaneously satisfiable.

Finally we obtain:

Theorem 3–3 (Completeness theorem). If $\mathfrak{A}$ is valid, then $\mathfrak{A}$ is provable, i. e. there exists a closed tableaux for $\sim\mathfrak{A}$ by the proof procedure based on the modified analytic tableaux.

Proof. Assuming that $\mathfrak{A}$ is valid. Let T be the finished tableaux starting $\sim\mathfrak{A}$. If an open branch θ of T can not be closed by the unification algorithm, the open branch θ satisfies the conditions for the Hintikka set from the construction of the modified analytic tableaux, hence θ would be satisfiable, contrary to the hypothesis that $\mathfrak{A}$ is valid.

Example. Consider the following valid formula;

$$(\forall u)(\exists v)(\forall w)(\forall x)(\exists y)[P(v, w)\vee P(v, x)\rightarrow P(u, y)].$$

The initial formula to be proved is the unsatisfiable formula of prenex normal form with no existential quantifiers which are eliminated by 0-ary Skolem function a and 1-ary Skolem function symbols f, g as follows;

$$(\forall v)(\forall y)[\sim(P(v, f(v))\vee P(v, g(v))\rightarrow P(a, y))].$$

We illustrate the example of the modified analytic tableaux in Fig. 1, corresponding to that of resolution method in Fig. 2.

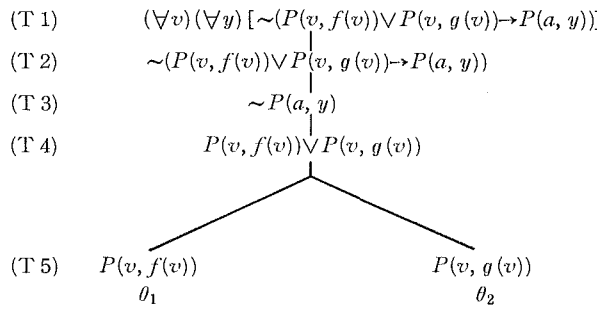

Fig. 1. Example of modified analytic tableaux

(R 1)	$P(v, f(v))\vee P(v, g(v))$	from (T 2)
(R 2)	$\sim P(a, y)$	from (T 2)
(R 3)	$P(a, g(a))$	a resolvent of (R 1) and (R 2) by substitution $(a/v, f(a)/y)$
(R 4)	$\square$ (null clause)	a resolvent of (R 2) and (R 3) by substitution $(g(a)/y)$

Fig. 2. Example of resolution method

Now, we can easily see that an open branch θ_1 to be closed by substitution a/v and $f(a)/y$ in Fig. 1 corresponds to a resolvent (R 3) in Fig. 2, θ_2 closed by $g(a)/y$ corresponds to a null resolvent (R 4).

4. Discussions and Applications

In this chapter, we discuss informally the main features of our proof procedure in relation to the resolution method and Gentzen-type proof procedure[5,7], and describe the applications to program synthesis and hypothesis-generation.

4.1 Some advantageous features

We have obtained the proof procedure based on the modified analytic tableaux by the natural combination of aspect of resolution, i. e. unification and that of analytic tableaux, i. e. tree structure. These methods are closely similar to each other in that their proof procedures are based on refutation procedure, in other words, reductio ad absurdum. Accordingly, they suggest a way to avoid the impossibility of enumerating all models. On the one hand, there is a very

interesting aspect between resolution method and Gentzen-type formal system. In the resolution method, we use only one inference rule, resolution principle. That is, we infer from two clauses $\mathfrak{C}_1$ and $\mathfrak{C}_2$ the resolvent:

$$(\mathfrak{C}_1-\mathfrak{L}_1)\,\sigma\cup(\mathfrak{C}_2-\mathfrak{L}_2)\,\sigma$$

by the most general unifying substitution σ so that $\mathfrak{L}_1\sigma$ and $\mathfrak{L}_2\sigma$ are complementary literals, where $\mathfrak{L}_1\in\mathfrak{C}_1$ and $\mathfrak{L}_2\in\mathfrak{C}_2$. In Gentzen-type formal system, we can prove theorem without inference figure, cut rule[3]:

$$\frac{\Gamma\rightarrow\Delta,\mathfrak{A}\qquad\mathfrak{A},\Pi\rightarrow\Lambda}{\Gamma,\Pi\rightarrow\Delta,\Lambda}\,,$$

which may be essentially considered as the same form as the resolution principle. On the other hand, as readily seen, from the form of the rules of inference, our rules for constructing the modified analytic tableaux are as natural a form as Gentzen-type inference rule. They are also goal-oriented, that is, a formula to be proved is decomposed into subgoals which correspond to lemmas in mathematics by the rules of inference. In Gentzen-type method, we test whether those subgoals are axioms simultaneously, and in our method, whether they include contradictions, i. e., formulas and their negations simultaneously. Thus, from the above considerations we can see that our method has main features of both resolution method and Gentzen-type procedure.

In addition to the above features, other characteristic features of our proof procedure may be suggested as follows:

1) The matrix part of the initial formula to be proved is not a conjunctive normal form. In order to transform any formula into conjunctive normal form, we should replace, for example, $(\mathfrak{A}\,\&\,\mathfrak{B})\vee\mathfrak{C}$ by $(\mathfrak{A}\vee\mathfrak{C})\,\&\,(\mathfrak{B}\vee\mathfrak{C})$. This causes the complexity of the formula to increase because of the duplication of formula $\mathfrak{C}$.

2) The main task of our proof procedure is to construct a semantic proof tree, although the major feature of the resolution method is the uniformity of the proof procedure which results in producing many resolvents which are not related to a proof and consuming time and space in a computer.

3) We can add the resolvents constructed by the resolution principle from any two clauses to the set of formulas to be proved if desired, since this is guaranteed by the following theorem[1]:

Theorem 4–1. Given two clauses $\mathfrak{C}_1$ and $\mathfrak{C}_2$, a resolvent $\mathfrak{C}$ of $\mathfrak{C}_1$ and $\mathfrak{C}_2$ is a logical consequence of $\mathfrak{C}_1$ and $\mathfrak{C}_2$.

4.2 Illustrative examples

We can not give a precise formulation to problems to be considered as application of our proof procedure, but can only conceptually describe the possibilities of applying our proof procedure with examples.

Example 4–1. We consider the problem of program synthesis in the same

way as Waldinger and Lee[8] in our formal system. Assuming that we wish to write a program of the characteristic function of the predicate "Atom" in LISP. The axiom we need and program specification can be written as follows;

$$(\forall x)\ \mathrm{Equal}\ (x, x)$$
$$(\forall x)\ (\exists y)\ [(\mathrm{Atom}\ (x)\ \&\ \mathrm{Equal}\ (y, 1))\vee(\sim\mathrm{Atom}\ (x)\ \&\ \mathrm{Equal}\ (y, 0))]\ .$$

Hence, the unsatisfiable set of formulas is

$$\left\{\begin{array}{l}(\forall x)\ \mathrm{Equal}\ (x, x)\ , \\ (\forall y)\ [\sim((\mathrm{Atom}\ (a)\ \&\ \mathrm{Equal}\ (y, 1))\vee(\sim\mathrm{Atom}\ (a)\ \&\ \mathrm{Equal}\ (y, 0)))]\end{array}\right\}\ .$$

Then, the corresponding modified analytic tableaux is given in Fig. 3.

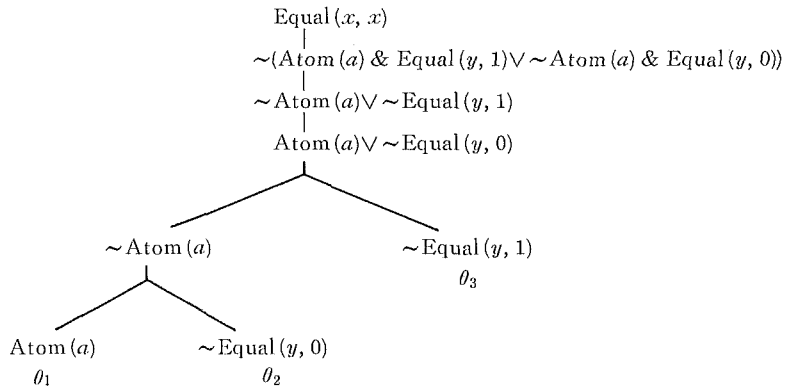

Fig. 3. A tableaux for program synthesis

In Fig. 3, θ_2, θ_3 is closed by $0/y$, $1/y$ respectively. From this modified analytic tableaux, it will be possible to extract the necessary information to construct a program satisfying the given specification as well as the information extracting method of resolution. In this case, the resulting program may be written as, for example,

$$\text{if}\ \mathrm{Atom}\ (x)\ \text{then}\ y \leftarrow 1\ \text{else}\ y \leftarrow 0\ ;\ .$$

Example 4-2. We consider the problem of hypothesis generation whose object is to find the lacking hypotheses (or premises) by deduction[9]. This problem is very important because it is generally difficult and, error-prone to specify a problem which can be formulated as a proof of theorem in a certain complete logical form. If we wish to find some hypothesis lacking in the following set of formulas,

$$\left\{\begin{array}{l}(\forall x)\ [C\ (x) \rightarrow W\ (x)\ \&\ R\ (x)]\ , \\ (\forall y)\ [\sim D\ (y)\vee\sim R\ (y)]\end{array}\right\}\ .$$

Then, the modified analytic tableaux with which we are concerned is in Fig. 4.

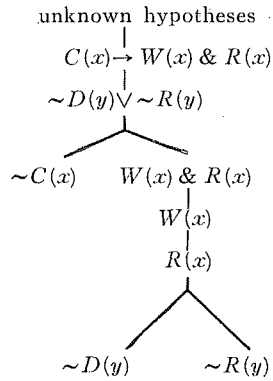

Fig. 4. A tableaux for hypothesis generation

From this tableaux, we have a formula $(\forall x)(C(x) \& D(x))$ as a possible interesting formula. Because, by $(\forall x)(C(x) \& D(x))$ we can close all branches of the tableaux.

Up to this point, we have considered applications without describing the concrete procedures. As future research, we will implement modified analytic tableaux, formulate effective algorithms for these applications and also it will be necessary for us to make further improvements of the modified analytic tableaux in order to apply our procedure to practical problems. In particular, it will be helpful for improvements to study in what manner useful strategies for resolution method affect the tree form of tableaux. Furthermore, it seems to be of considerable interest that the tableaux method may be extended in such a way that the treatment of other logical operators, for example, $\Box$ (necessity) or $\Diamond$ (possibility) in modal logic[10] is possible.

Acknowledgements

The authors wish to express their gratitude to Professor S. Takasu, Associate Professor S. Igarashi, Mr. M. Sato, Mr. K. Hayashi of Research Institute for Mathematical Sciences, Kyoto University and Mr. S. Gotoh of Musashino Electrical Communication Laboratory, N. T. T. for their critical discussions and many helpful suggestions.

References

1) Chang, C. L. and Lee, R. C. T.: Symbolic logic and mechanical theorem proving, Academic Press (1973).
2) Robinson, J. A.: A machine-oriented logic based on the resolution principle, J. ACM **12** (1965), pp. 23–41.
3) Gentzen, G.: Untersuchungen über das logische Schliessen 1, Math. Z. **39** (1935), pp. 176–210.
4) Shoenfield, J. R.: Mathematical logic, Addison-Wesley (1967).
5) Sawamura, H.: On proof procedures in mechanical theorem proving (Master's thesis), Department of Information Engineering, Hokkaido Univ. (1975).

6) Smullyan, R. M.: First-order logic, Springer-Verlag (1971).
7) Sawamura, H. et al.: On a programming of proofs, In a collection of Papers, SICE, held in Sapporo (1974) (in Japanese).
8) Waldinger, R. J. and Lee, R. C. T.: PROW: A step toward automatic program writing, Proc. International Joint Conf. on Artificial Intelligence, Washington, D. C. (1969), pp. 241–252.
9) Maeda, T.: A consideration on the deduction with conditions AL 74–22 (1974) (in Japanese).
10) McCarthy, J. and Hayes, P. J.: Some philosophical problems from the standpoint of artificial intelligence, Machine Intelligence 4 (1969), pp. 463–503.