Deduction by Combining Semantic Tableaux and Integer Programming*

Bernhard Beckert and Reiner Hähnle
University of Karlsruhe, Institute for Logic, Complexity and Deduction Systems, 76128 Karlsruhe, Germany. Phone: +49-721-608-4324, FAX: +49-721-608-4329, Email: {beckert,haehnle}@ira.uka.de, *WWW:* http://i12www.ira.uka.de/

Abstract. In this paper we propose to extend the current capabilities of automated reasoning systems by making use of techniques from integer programming. We describe the architecture of an automated reasoning system based on a Herbrand procedure (enumeration of formula instances) on clauses. The input are arbitrary sentences of first-order logic. The translation into clauses is done incrementally and is controlled by a semantic tableau procedure using unification. This amounts to an incremental polynomial CNF transformation which at the same time encodes part of the tableau structure and, therefore, tableau-specific refinements that reduce the search space. Checking propositional unsatisfiability of the resulting sequence of clauses can either be done with a symbolic inference system such as the Davis-Putnam procedure or it can be done using integer programming. If the latter is used a number of advantages become apparent.

Introduction

In this paper we propose to extend the current capabilities of automated reasoning (AR) systems by combining the inference procedure *semantic tableaux* with integer program (IP) solvers. We show that the resulting system has properties which are interesting for such applications as formal program verification. In Section 1 we summarize some facts on semantic tableaux in order to make the paper reasonably self-contained. In Section 2 we give a tableau-based polynomial time translation from propositional logic into IPs. This translation will be lifted to full first-order logic in Section 3. With an extended example we illustrate how the system is supposed to work (Section 4) and in Section 5 we summarize the possible synergy effects from marrying AR and OR in the way suggested. Finally we mention related and ongoing work. We had to omit all proofs due to limited space.

1 Semantic Tableaux

First we state some standard notions of computational logic that will be used in the following; consult (Fitting, 1990) for details. Let us fix a first-order language whose terms and formulae are built up from countable sets of predicate symbols, function symbols, constant symbols and object variables in the usual manner (for each arity there are countably many function and predicate symbols). We use the logical connectives $\wedge$ (conjunction), $\vee$ (disjunction), $\supset$ (implication) and $\neg$ (negation),

* This research was supported by Deutsche Forschungsgemeinschaft within the Schwerpunktprogramm *Deduktion*.

and the quantifier symbols $\forall$ and $\exists$. An *atom* is a formula of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol and $t_1, \ldots, t_n$ are terms. Atoms and their negations are called *literals*. A *clause* is a disjunction of literals. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. A variable is *free* if it is not bound by a quantifier ($\forall$ or $\exists$). A *sentence* is a formula not containing any free variables. We use the standard notions of *satisfiability* and *model*. A sentence is called a *tautology* if it is true in all models, i.e., if its negation is unsatisfiable. *Substitutions* are mappings from variables to terms and are extended to formulae as usual. We denote a substitution by $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$, where $\{x_1, \ldots, x_n\}$ are the variables that occur in the term it is applied to. The application of σ to a term t is denoted by $t\sigma$.

Semantic (or analytic) tableaux are a sound and complete calculus for doing logical inferences in full first-order logic. They were developed in the 1950s from Gentzen systems. For an introduction which covers the material needed here, see (Fitting, 1990). Following Fitting we divide the set of formulae of into four classes: α for formulae of conjunctive type, β for formulae of disjunctive type, γ for quantified formulae of universal type and finally δ for quantified formulae of existential type. This is called *uniform notation*; it simplifies presentation and proofs considerably. The classification is motivated by the *tableau expansion rules* which are associated with each formula. The rules characterize the assertion of a truth value to a formula by means of asserting truth values to its direct subformulae. For example, $\phi \wedge \psi$ holds if and only if ϕ and ψ hold. In the upper part of Table I the rule schemata for the various formula types are given. Premises and conclusions are separated by a horizontal bar, while vertical bars in the conclusion denote different *extensions* which are to be thought as disjunctions. In the lower part of Table I the correspondence between formulae and formula types is shown.

$$\frac{\alpha}{\alpha_1 \\ \alpha_2} \qquad \frac{\beta}{\beta_1 \mid \beta_2} \qquad \frac{\gamma}{\gamma_1(y)} \qquad \frac{\delta}{\delta_1(f(x_1, \ldots, x_n))}$$

where y is a new free variable.

where f is a new (Skolem) function symbol, and $x_1, \ldots, x_n$ are the free variables occurring in δ.

α	α_1	α_2
$\phi \wedge \psi$	ϕ	ψ
$\neg(\phi \vee \psi)$	$\neg\phi$	$\neg\psi$
$\neg(\phi \supset \psi)$	ϕ	$\neg\psi$
$\neg\neg\phi$	ϕ	ϕ

β	β_1	β_2
$(\phi \vee \psi)$	ϕ	ψ
$\neg(\phi \wedge \psi)$	$\neg\phi$	$\neg\psi$
$(\phi \supset \psi)$	$\neg\phi$	ψ

γ	$\gamma_1(y)$
$(\forall x)\phi(x)$	$\phi(y)$
$\neg(\exists x)\phi(x)$	$\neg\phi(y)$

δ	$\delta_1(f(x_1, \ldots, x_n))$
$\neg(\forall x)\phi(x)$	$\neg\phi(f(x_1, \ldots, x_n))$
$(\exists x)\phi(x)$	$\phi(f(x_1, \ldots, x_n))$

TABLE I
Formula types and tableau rule schemata.

We use *free variable* quantifier rules (Fitting, 1990; Hähnle and Schmitt, 1994). Instead of "guessing" ground terms that are instantiated for universally quantified variables, a new *free* variable is introduced, that is instantiated later "on demand"

with a term that is useful.

For our purposes it is sufficient to visualize a *tableau* as a finite binary tree, whose nodes are first-order formulae, constructed as follows:

1. A finite linear tree whose nodes are formulae taken from a set Φ of formulae is a tableau for Φ.

2. If T is a tableau for Φ and ϕ is a node from T then a new tableau T' for Φ is constructed by extending a branch of T that contains ϕ by as many new linear subtrees as the rule[1] corresponding to ϕ has extensions, the nodes of the new subtrees being labelled with the formulae in the extensions.[2]

A branch B of T is a maximal path in T. It is often identified with the set of formulae it contains. A tableau *branch* is *closed* iff it contains a pair of complementary formulae, i.e., formulae of the form ϕ and $\neg\phi$.[3] A *tableau* is *closed* (under σ) iff there is a substitution σ such that all branches $B\sigma$ of $T\sigma$ are closed.

To prove tautologyhood of a formula ϕ we begin with a tree whose single node is labelled by $\neg\phi$, that is we assume that ϕ is false in some model. A tableau proof represents a systematic search for such a model. Every tableau branch corresponds to a partial possible model in which the formulae on the branch are valid. Therefore, a complementary pair of formulae, and thus a closed branch, denotes an explicit contradiction, since in no model both a formula and its negation can be true.

A proof of the following theorem, that states soundness and completeness of semantic tableaux, can be found in (Fitting, 1990) (completeness part) and (Hähnle and Schmitt, 1994) (soundness part).

THEOREM 1. Let ϕ be any first-order sentence. Then there is a closed tableau for $\{\neg\phi\}$ iff ϕ is a first-order tautology.

Using the deduction theorem for first order logic[4], an immediate corollary of Theorem 1 is that for all sentences $\phi_1, \ldots, \phi_n, \phi$: $\{\phi_1, \ldots, \phi_n\} \models \phi$ iff there is a closed tableau for $\{\phi_1, \ldots, \phi_n, \neg\phi\}$.

Tableau construction for a set of formulae Φ is a highly non-deterministic procedure. We did not specify, for example, in which order the tableau rules should be applied to the formulae on a branch, or how a closing substitution should be searched for.

[1] It is obtained by looking up the subformulae corresponding to ϕ and instantiating the matching rule schema (Table I).

[2] From the two formulae in the conclusion of a double negation only one copy needs to be kept. Moreover, it is is sufficient for completeness to apply α-, β- and δ-rules only once to every formula in each branch. Consequently, formulae of these types may be deleted locally to the current branch after rule application. Note, however, that γ-formulae must be used repeatedly sometimes and hence may not be removed.

[3] It is sufficient merely to consider complementary pairs of *atomic* formulae.

[4] For all sentences $\phi_1, \ldots, \phi_n, \phi$: $\{\phi_1, \ldots, \phi_n\} \models \phi$ iff $(\phi_1 \wedge \ldots \wedge \phi_n) \supset \phi$ is a tautology (where $\models$ denotes the logical consequence relation).

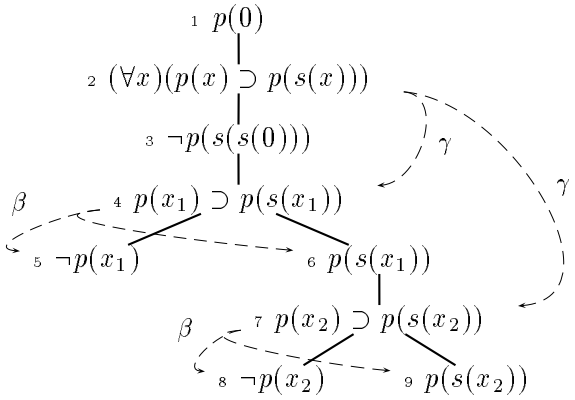

Fig. 1. The tableau proof described in Example 1.

EXAMPLE 1. The tableau shown in Figure 1 proves that $p(s(s(x)))$ is a logical consequence of $\{p(0),\ (\forall x)(p(x) \supset p(s(x)))\}$. Formulae (1)–(3) are put on the tableau initially. Formula (4) is derived from (2) by applying the γ-rule, and then (5) and (6) are added by applying the β-rule to (4). Now, the left branch is closed under the substitution $\{x_1 \leftarrow 0\}$ by (1) and (5). The right branch of the tableau is not closed under $\{x_1 \leftarrow 0\}$; thus, the γ-rule has to be applied a second time to (2) to derive (7), and then (8) and (9) from (7). At that point the whole tableau is closed under the substitution $\{x_1 \leftarrow 0, x_2 \leftarrow s(0)\}$, the middle branch by (6) and (8), and the right branch by (3) and (9). The middle branch could have been closed under the substitution $\{x_2 \leftarrow 0\}$ as well (using (1) and (8)); this, however, would have been useless and does not close the branch on the right. There is a refinement of semantic tableaux called regularity (Letz *et al.*, 1992) that can avoid such closures: it is not allowed to put two identical formulae on a branch. This condition would be violated under the substitution $\{x_2 \leftarrow 0\}$, because (9) and (6) would then become identical.

2 Translating Semantic Tableaux into Integer Programs

In this section we describe a method using semantic tableaux for translating a *propositional*[5] formula ϕ (which *needs not* to be in any normal form) into a 0-1-IP $\mathcal{C}$ such that ϕ is satisfiable iff $\mathcal{C}$ is feasible. Tableau rules are used to split and transform ϕ, whereas IP methods are used to check whether the resulting tableau is closed.

For propositional CNF formulae there is a well-known standard translation into 0-1-IPs: Each clause of the form $p_1 \vee \ldots \vee p_k \vee \neg p_{k+1} \vee \ldots \vee \neg p_m$ $(1 \leq k \leq m)$

[5] Lifting of this method to full first-order logic is described in Section 3.

$$\frac{\boxed{\geq i}\ \alpha}{\begin{array}{l}\boxed{\geq i}\ \alpha_1\\ \boxed{\geq i}\ \alpha_2\end{array}} \qquad \frac{\boxed{\geq i}\ \beta}{\begin{array}{l}\boxed{\geq i-j}\ \beta_1\\ \boxed{\geq i+j-1}\ \beta_2\end{array}} \qquad \frac{\boxed{\geq i}\ p}{p \geq i} \qquad \frac{\boxed{\geq i}\ \neg p}{1-p \geq i}$$

where j is a new IP variable.

TABLE II

Propositional rules for signed α-formulae, β-formulae, and literals (p is an atomic formula).

corresponds to the constraint

$$p_1 + \cdots + p_k + (1 - p_{k+1}) + \cdots + (1 - p_m) \geq 1 \ .$$

The question whether a single tableau branch B is closed can as well be easily transformed into a 0-1-IP: B is closed iff the set of constraints

$$\{p \geq 1 \ : \ p \in B,\, p \text{ an atom}\} \cup \{p \leq 0 \ : \ \neg p \in B,\, p \text{ an atom}\}$$

is infeasible. Using this translation, the question whether a whole tableau T is closed results in a disjunctive programming problem: T is closed iff there is a solution to one of the IPs constructed for each of its branches; that way, nothing is gained by using IPs, because the transformation does not make use of their expressiveness.

Instead, we use techniques similar to that of disjunctive programming to encode a whole tableau, including its structure, into a single 0-1-IP. This translation makes use of *signed* formulae[6]. A signed formula is a string of the form $\boxed{\geq i}\,\phi$, where ϕ is a (propositional or first-order) formula and i is a linear expression (for example $1-j_1+j_2$). The sign associates a logical truth value with the formula. For example, $\boxed{\geq 1}\,\phi$ means that ϕ is true. One could add signs of the form $\boxed{\leq i}\,\phi$ to express "ϕ is false" by $\boxed{\leq 0}\,\phi$; this, however, is not necessary as we may use $\boxed{\geq 1}\,\neg\phi$ instead. By employing signed formulae, tableau rules that are linear for β-formulae (in contrast to the rule in Table I) can be defined, see the second rule in Table II. To generate a 0-1-IP, two additional rules are needed that translate literals into constraints, see the two rules on the right of Table II.

There is, of course, a price to be paid for the linearity of the disjunctive β-rules. New variables are introduced by their application, that we call *branching variables*. Each assignment of 0/1-values to the branching variables in the resulting IP corresponds to one of the tableau branches and, thus, to a partial model. If, for example, by assigning values to $j_1, \ldots, j_k$, a linear expression $i = i(j_1, \ldots, j_k)$ evaluates to 1, then $\boxed{\geq i}\,\phi$ means that ϕ is part of the branch B corresponding to that assignment and is valid in the partial model associated with B.

[6] Signed formulae (with different types of signs) are frequently used in semantic tableaux for non-classical logics, e.g. multiple-valued logics (Hähnle, 1994a).

The rules from Table II can be used to step by step transform a set of signed formulae into an IP:

DEFINITION 1. Let $\Phi = \{\phi_1, \ldots, \phi_k\}$ be a set of propositional formulae, and let the sequence $\mathcal{C}_0, \ldots, \mathcal{C}_n$ be formed according to the following rules:

1. $\mathcal{C}_0 = \{\boxed{\geq 1}\,\phi_1, \ldots, \boxed{\geq 1}\,\phi_k\}$,
2. $\mathcal{C}_m$ is derived from $\mathcal{C}_{m-1}$ by applying one of the tableau rules from Table II to $\psi \in \mathcal{C}_{m-1}$ and replacing ψ by the result of the transformation ($1 \leq m \leq n$).
3. $\mathcal{C}_n$ consists only of constraints (i.e., there are no a signed formulae left).

Then $\mathcal{C}_n$ is a **0-1-IP associated with** Φ.

The following soundness and completeness theorem holds:

THEOREM 2. If $\mathcal{C}$ is a 0-1-IP associated with a set Φ of propositional formulae (Def. 1), then:

$$\mathcal{C} \text{ is infeasible iff } \Phi \text{ is unsatisfiable.}$$

Theorem 2 implies that a propositional formula ϕ is a tautology iff the IP(s) associated with $\{\neg\phi\}$ are infeasible.

EXAMPLE 2. Let $\Phi = \{p \vee \neg p\}$; then $\mathcal{C}_0 = \{\boxed{\geq 1}\,(p \vee \neg p)\}$. By applying the β-rule we obtain $\mathcal{C}_1 = \{\boxed{\geq 1-j}\,p, \boxed{\geq 1+j-1}\,\neg p\}$; the literal rules are applied to derive the 0-1-IP

$$\mathcal{C}_3 = \{\quad p \geq 1 - j\ ,\quad 1 - p \geq 1 + j - 1\}$$

that is associated with Φ. Since $\mathcal{C}_3$ is feasible, $p \vee \neg p$ has to be satisfiable (which is, of course, true).

The two possible assignments of values (0 or 1) to the branching variable j correspond two the two branches of the semantic tableau for $p \vee \neg p$, and thus to the two possible models, in which p is either true or false.

In case β_1 or β_2 is a literal, the β rule can be optimized inasmuch as the introduction of an additional variable can be avoided; the variable is simply replaced by the literal itself, which then becomes part of the constraint (Table III).

Using this optimization, the formula from Example 2 is transformed into the single constraint $1 - p \geq 1 - p$ whose feasibility (for all values of p) can be seen immediately. Taking this optimization into account our translation collapses into the standard translation mentioned at the beginning of this section in the case of CNF input.

$$\frac{\boxed{\geq^i}\,\beta}{\boxed{\geq^{i-p}}\,\beta_2} \qquad \frac{\boxed{\geq^i}\,\beta}{\boxed{\geq^{i-p}}\,\beta_1} \qquad \frac{\boxed{\geq^i}\,\beta}{\boxed{\geq^{i-(1-p)}}\,\beta_2} \qquad \frac{\boxed{\geq^i}\,\beta}{\boxed{\geq^{i-(1-p)}}\,\beta_1}$$

if β_1 is a literal p. if β_2 is a literal p. if β_1 is a literal $\neg p$. if β_2 is a literal $\neg p$.

TABLE III
Optimized β-rules in the case when β_1 or β_2 is a literal.

$$\frac{\boxed{\geq^i}\,\beta}{\begin{array}{l}\boxed{\geq^{i-j(x_1,\ldots,x_n)}}\,\beta_1\\ \boxed{\geq^{i+j(x_1,\ldots,x_n)-1}}\,\beta_2\end{array}}$$

where j is a new n-ary predicate symbol, and $x_1, \ldots, x_n$ are the free variables occurring both in β_1 and β_2.

$$\frac{\boxed{\geq^i}\,\gamma}{\boxed{\geq^i}\,\gamma_1(y)}$$

where y is a new free variable.

$$\frac{\boxed{\geq^i}\,\delta}{\boxed{\geq^i}\,\delta_1(f(x_1,\ldots,x_n))}$$

where f is a new (Skolem) function symbol, and $x_1, \ldots, x_n$ are the free variables occurring in δ.

TABLE IV
First-order constraint rules for β-, γ-, and δ-formulae.

3 Lifting to First-Order Logic

Our lifting of the method described in the previous section to first-order logic is based on Herbrand's Theorem[7]. A set Φ of first-order sentences is first transformed into an IP containing free variables[8]. Then, new instances of the parts of the IP that correspond to universally quantified (sub-)formulae are added to the problem until it becomes unsatisfiable (if Φ is satisfiable this process does, in general, not terminate, because satisfiability of first-order sentences is undecidable).

The transformation rules for quantified formulae (γ- and δ-rules) from Table I can be adapted to signed formulae straightforwardly. The α-rules and the rules for literals (Table II) remain unchanged for first-order logic. The β-formulae, however, become slightly more complicated. It is necessary to parameterize the branching variables with some of the free variables. The first-order rules are shown in Table IV.

The definition of IPs associated with a set of formulae has to be adapted. Since more than one instance of universally quantified (sub-)formulae may be needed, a mechanism has to be added that allows to duplicate and instantiate parts of the IP (Rule 2(b) in the definition):

DEFINITION 2. Let $\Phi = \{\phi_1, \ldots, \phi_k\}$ be a set of first-order sentences and let the sequence $\mathcal{C}_0, \ldots, \mathcal{C}_n$ be formed according to the following rules:

[7] A set Φ of clauses is unsatisfiable iff there is an unsatisfiable *finite* set of *ground* (i.e. variable-free) instances of clauses from Φ.

[8] These free variables should not be confused with IP variables in constraints (e.g. branching variables). IP variables correspond to atomic formulae and, thus, might *contain* free variables.

1. $\mathcal{C}_0 = \{\boxed{\geq 1}\,\phi_1, \ldots, \boxed{\geq 1}\,\phi_k\}$,
2. a) $\mathcal{C}_m$ is derived from $\mathcal{C}_{m-1}$ by applying the α- or the literal rules from Table II, or the β, γ-, or δ-rules from Table IV to $\psi \in \mathcal{C}_{m-1}$ and replacing ψ by the result of the transformation $(1 \leq m \leq n)$;[9] or
   b) there is a substitution σ such that $\mathcal{C}_m = \mathcal{C}_{m-1} \cup (\mathcal{C}_{m-1}\sigma)$.
3. $\mathcal{C}_n$ consists only of constraints (that is no signed formulae are left).

Then $\mathcal{C}_n$ is a **(first-order) 0-1-IP associated with Φ**.

Optimized versions of the β-rule in case when β_1 or β_2 is a literal (similar to those in Table III) can still be used.

The following soundness and completeness theorem for first-order logic holds; note, that in general not every IP associated with an unsatisfiable set of formulae is infeasible (in contrast to the propositional case, Theorem 2).

THEOREM 3. A finite set Φ of first-order sentences is unsatisfiable iff at least one of the first-order 0-1-IPs associated with Φ is infeasible.

This theorem implies soundness and completeness of the following procedure that can be used to prove a first-order formula ϕ to be a tautology:

1. Apply α-, β-, γ-, δ-, and literal rules as long as possible to derive from $\mathcal{C}_0 = \{\boxed{\geq 1}\,\neg\phi\}$ the 0-1-IP $\mathcal{C}$
2. **if** the 0-1-IP $\mathcal{C}$ is *infeasible*
   then STOP ($\neg\phi$ is unsatisfiable; ϕ is a tautology)
3. Choose a solution L of $\mathcal{C}$, L : Atoms($\mathcal{C}$) $\rightarrow \{0, 1\}$
4. **if** there are σ, p, q, such that $\sigma(p) = \sigma(q)$ but $L(p) \neq L(q)$
   then $\mathcal{C} := \mathcal{C} \cup \sigma(\mathcal{C})$; GOTO 3
   else STOP ($\neg\phi$ satisfiable; ϕ is not a tautology)

Note, that the choice of the solution L is indeterministic; for completeness backtracking has to be used or fairness strategies have to be employed. Since the substitutions σ, that are applied to generate new instances, are computed by analyzing the solutions of the IPs, and since this analysis is global (and is not restricted to single tableau branches), the search space is much smaller than that for semantic tableaux.

The pairs of atoms p, q that can be used to remove a solution are called *links*. It is a good heuristic to prefer links that involve an atom (p or q) that is part of as few links as possible. This heuristic can be encoded into a minimization problem and integrated into the IP.

[9] Note, that γ-rules, too, are removed and replaced by γ_1

4 Example

As an example, we use the procedure described above to prove (again) the formula from Example 1 to be a tautology, i.e., that $\Phi = \{p(0),\ (\forall x)(p(x) \supset p(s(x))),\ \neg p(s(s(x)))\}$ is unsatisfiable. We initialize

$$\mathcal{C}_0 = \{\boxed{\geq 1}\, p(0),\ \boxed{\geq 1}\, (\forall x)(p(x) \supset p(s(x))),\ \boxed{\geq 1}\, \neg p(s(s(x)))\}\ .$$

By applying the literal rules (Table II) to $\boxed{\geq 1}\, p(0)$ and $\boxed{\geq 1}\, \neg p(s(s(x)))$ we derive the constraints

$$p(0) \ \geq\ 1 \tag{1}$$
$$1 - p(s(s(0))) \ \geq\ 1 \tag{2}$$

From $\boxed{\geq 1}\, (\forall x)(p(x) \supset p(s(x)))$ we derive $\boxed{\geq 1}\, (p(x) \supset p(s(x)))$ using the γ-rule (Table IV), then $\boxed{\geq 1-(1-p(x))}\, p(s(x))$ by applying the optimized β-rule[10], and finally using the literal rule $p(s(x)) \geq 1 - (1 - p(x))$, i.e.,

$$(1 - p(x)) + p(s(x)) \ \geq\ 1 \tag{3}$$

The 0-1-IP $\mathcal{C}$ consisting of (1)–(3) is feasible. We (arbitrarily) chose the solution L_1, where $L_1(p(0)) = L_1(p(s(x))) = 1$ and $L_1(p(s(s(0)))) = L_1(p(x)) = 0$. This solution can be removed using the link $p(0), p(x)$, since $L_1(p(0)) \neq L_1(p(x))$, but $\sigma(p(0)) = \sigma(p(x))$, where $\sigma = \{x \leftarrow 0\}$. Thus, we carry on with the IP $\mathcal{C} \cup \mathcal{C}\sigma$, i.e., we add the constraint

$$(1 - p(0)) + p(s(0)) \ \geq\ 1 \tag{4}$$

The new problem (1)–(4) is still feasible. One solution is L_2, where $L_2(p(0)) = L_2(p(s(0))) = L_2(p(s(x))) = 1$ and $L_2(p(s(s(0)))) = L_2(p(x)) = 0$. We remove the solution using the link $p(s(0)), p(x)$ and apply $\sigma = \{x \leftarrow s(0)\}$ to add

$$(1 - p(s(0))) + p(s(s(0))) \ \geq\ 1 \tag{5}$$

The resulting 0-1-IP (1)–(5) is infeasible, which proves Φ to be unsatisfiable.

It is obviously useless to use the link $p(0), p(x)$ to remove the solution L_2, because (4) would be added a second time. In general it is not as easy to recognize useless links; fortunately, it is possible to adapt regularity (described in Example 1) and other strategies known from semantic tableaux to avoid using such links.

[10] Applying the non-optimized rule from Table II results in the two formulae $\boxed{\geq 1-j(x)}\, \neg p(x)$ and $\boxed{\geq 1+j(x)-1}\, p(s(x))$ containing a branching variable, and finally in the constraints $j(x) + (1 - p(x)) \geq 1$ and $(1 - j(x)) + p(s(x)) \geq 1$.

5 Synergy Effects

In this section we list the benefits that can be gained from an interaction between AR and OR techniques as suggested in the previous sections. Due to lack of space we had to leave out concrete examples for many statements.

- The fact that logical formulae and the linear fragment of arithmetic are mapped into the same domain allows an efficient representation of the search space associated with formulae as they typically occur in verification conditions during formal program verification. Arithmetic properties are awkward to define by purely logical means. On the other hand, if a special machinery for dealing with purely arithmetical subproblems is used, tough problems with redundancy and fairness tend to emerge. It is possible to view first-order formulae over linear arithmetic as an *extension* of IP and the presented mechanism as a solver that makes use of AR techniques to gain efficiency.

- As a tableau procedure is used to produce instances of formulae the input is not restricted to any normal form; for the same reason an adaptation of the technique to certain non-classical logics is possible, see (Hähnle, 1994b; Hähnle and Ibens, 1994). Both properties are important for many applications.

- Reductions of the search space (such as the regularity restriction defined above) as they are commonly found in tableau-oriented procedures can be built into the translation. The same holds for polynomial CNF transformation, cf. (Plaisted and Greenbaum, 1986), and for an optimized version of Skolemization (Hähnle and Schmitt, 1994).

- The amount of backtracking which normally occurs in tableaux is greatly reduced due to the efficient representation of a whole tableau which still can be checked rapidly for closure (unsatisfiability). This kind of representation makes it also possible to define subsumption within the $\mathcal{C}_i$. Moreover, the cost function of integer programs can be employed to suggest substitutions that lead to a favourable structure of the search space. In addition, a meaningful cost function often improves the behaviour of IP solvers.

- Many IP solvers allow incremental solutions. Moreover, IP solvers tend to find solutions of satisfiable problems quickly. Hence, they promise to be efficient for large, combinatorially not too hard, and mostly satisfiable problems such as they result from large formal specifications. Specific techniques for managing sparse matrices will be of advantage for such problems as well.

- Problem dependent heuristics can often be encoded as arithmetical properties in which case they can be represented at the same level as the problems themselves.

– Some IP techniques such as detection of simple (polynomially solvable) cases, generation of certain strong cuts or various preprocessing aids have no direct logical counterparts. Therefore, it can be hoped that such techniques can solve some problems quickly, where symbolic inference is in trouble.

Conclusion

Related Work The inference procedure as sketched in this paper is reminiscent of the *Primal Partial Instantiation Method* developed by Hooker and Rago (1992; 1994). The latter derives its name from the analogy between the generation of new inequalities in the primal simplex method of Dantzig (1963) for solving linear programs and the generation of new clauses/inequations in the procedure outlined above. Our proposal differs from Hooker and Rago's mainly in the following points: (i) we work with full first-order logic, not only with function-free universal clauses; (ii) our procedure encodes part of the structure of a semantic tableau into the generated inequations; (iii) we take advantage of the optimizing part of IP solvers for computing links (*blocks* in the terminology of Hooker and Rago) with a minimal number of alternatives, whereas Rago (1994) does not consider the use of IPs, but generates sequences of ground (variable-free) clauses.

Further related work is (Kagan *et al.*, 1993) which provides a translation (working as well by partial instantiation) from definite logic programs into linear programs. It is restricted to the area of logic programming and, as the authors concede, linear programming is not specifically exploited and could be substituted by a symbolic inference procedure.

Ongoing and Future Work An implementation of the suggested procedure (implemented in Prolog and C++) is under way. Once a prototype is operational we will start to evaluate various heuristics.

Summary On the meta-level the potential synergy effects of putting together AD and OR can be summarized as follows:

1. A mixed approach can switch implementation paradigms whenever it is of advantage.
2. Some techniques of AD have no OR counterpart and vice versa. A mixed procedure can employ all of them.
3. Finally, an occasional change of the point of view often results in new ideas such as the usage of cost functions to compute substitutions.

References

George B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.
Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, 1990.

Reiner Hähnle and Ortrun Ibens. Improving temporal logic tableaux using integer constraints. In *Proceedings, International Conference on Temporal Logic, Bonn, Germany*, pages 535–539. Springer LNCS 827, 1994.

Reiner Hähnle and Peter H. Schmitt. The liberalized δ-rule in free variable semantic tableaux. *Journal of Automated Reasoning,*, 13(2):211–222, October 1994.

Reiner Hähnle. *Automated Deduction in Multiple-Valued Logics*, volume 10 of *International Series of Monographs on Computer Science*. Oxford University Press, 1994.

Reiner Hähnle. Many-valued logic and mixed integer programming. *Annals of Mathematics and Artificial Intelligence*, 12(3,4):231–264, December 1994.

John N. Hooker. New methods for computing inferences in first order logic. Working Paper, GSIA, CMU Pittsburgh, April 1992.

Vadim Kagan, Anil Nerode, and V. S. Subrahmanian. Computing definite logic programs by partial instantiation and linear programming. Draft Manuscript, 1993.

Reinhold Letz, Johann Schumann, Stephan Bayerl, and Wolfgang Bibel. SETHEO: A high-perfomance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

Gabriella Rago. *Optimization, Hypergraphs and Logical Inference*. PhD thesis, Dipartimento di Informatica, Università di Pisa, March 1994. Available as Tech Report TD-4/94.