Journal of Universal Computer Science, vol. 5, no. 3 (1999), 135-155
submitted: 1/11/98, accepted: 2/2/99, appeared: 28/3/99 © Springer Pub. Co.

Loop-Detection in Hyper-Tableaux by Powerful Model Generation

Frieder Stolzenburg
Universität Koblenz · Institut für Informatik
Rheinau 1 · D–56075 Koblenz · Germany
stolzen@uni-koblenz.de

Abstract: Automated reasoning systems often suffer from redundancy: similar parts of derivations are repeated again and again. This leads us to the problem of loop-detection, which clearly is undecidable in general. Nevertheless, we tackle this problem by extending the hyper-tableau calculus as proposed in [Baumgartner, 1998] by generalized terms with exponents, that can be computed by means of computer algebra systems. Although the proposed loop-detection rule is incomplete, the overall calculus remains complete, because loop-detection is only used as an additional, optional mechanism. In summary, this work combines approaches from tableau-based theorem proving, model generation, and integrates computer algebra systems in the theorem proving process.

Key Words: Hyper-tableau; loop-detection; terms with exponents; computer algebra systems; model generation.

1 Introduction

Hyper-tableau [Baumgartner *et al.*, 1996] is a sound and complete calculus for first-order clausal logic wrt. computing answers. It is confluent and offers a method for the generation of models that can be stated by first-order terms produced by tableaux clauses [Baumgartner, 1998]. However, the restriction to first-order terms is a disadvantage, because only few loops can be expressed with them. Therefore, we introduce an inference rule for loop-detection in hyper-tableaux, making use of generalized terms with exponents and mathematical techniques from the theory of generating functions. Although the new rule is incomplete, the overall calculus remains (refutationally) complete, because loop-detection is only used as an additional, optional mechanism.

This research aims at generalizing known techniques for model generation (e.g. [Peltier, 1997a]). It combines approaches from theorem proving, model generation, and integrates computer algebra systems in the theorem proving process. While usually theorem provers work as assistants of computer algebra systems (e.g. [Buchberger, 1997]), here it is the other way round: computer algebra systems are employed as assistants of theorem provers, in order to make automated deduction more effective. In summary, the new calculus comprises several advantages:

1. It provides *powerful loop-detection*, subsuming other known techniques (e.g. some cases of regularity).
2. The use of a computer algebra system allows us to cope with *terms with generalized integer exponents* where, in general, the exponents need not be linear expressions.

3. The new calculus is *confluent*, this means backtracking is not necessary. This property is especially important here, because it avoids repeating computations of computer algebra systems that are expensive steps.

In the following, we first review the hyper-tableau calculus of [Baumgartner, 1998] in Sect. 2. After that, we state the new inference rule for loop-detection and its merits for automated theorem proving and model generation on some examples in Sect. 3. Then, in Sect. 4, we treat the calculus and its properties more formally. Sect. 5 discusses related approaches from different fields that are relevant for the problems of loop-detection and model generation in (tableau-based) automated reasoning. These works were unconnected so far. On the one hand, for loop-detection, regularity tests in tableaux and resolution of cyclic clauses were proposed. On the other hand, the generation of finite and infinite models for resolution-like calculi was investigated many times in the literature by means of terms with exponents and formal grammars. We conclude with an outlook on future work in Sect. 6.

2 Hyper-Tableaux Revisited

Hyper-tableau is a sound and complete calculus for clausal first-order logic. It has been established in [Baumgartner *et al.*, 1996] as an improvement over other model generating and case-splitting calculi such as e.g. SATCHMO [Manthey and Bry, 1988]. It has model building capabilities; but only such models are representable that can be described by simple first-order terms and their (non-)
instances [Baumgartner, 1998]. In this paper, the focus is on improving this situation. But beforehand, we briefly review the hyper-tableau calculus.

2.1 Reviewing the Calculus

In the following, we assume the reader to be familiar with the usual notions of first-order logic (see e.g. [Chang and Lee, 1973]). Our primary interest however is in clausal tableaux, similar to those in [Letz *et al.*, 1994]. A *clause* is a multiset (not an ordinary set) of literals, written as a disjunction $A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n$, where $m, n \geq 0$ and the As and Bs are atoms, or as $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ in implication-style, or $\mathcal{A} \leftarrow \mathcal{B}$ where $\mathcal{A} = \{A_1, \ldots, A_m\}$ and $\mathcal{B} = \{B_1, \ldots, B_n\}$. The literals in $\mathcal{A}$ are called *head literals* and the literals in $\mathcal{B}$ are called *body literals*.

A (Herbrand) interpretation I (for a given language) is represented as a (possibly infinite) set of ground atoms (i.e. atoms without any occurrences of variables), such that an atom A is true in I iff $A \in$ I. As usual, I $\models X$ means X is true in I where X is a sentence or a set of sentences (interpreted conjunctively). We write $X \models X'$ for sentences X and X' (or sets thereof) iff I $\models X$ implies I $\models X'$ for all suitable interpretations I.

We consider literal trees (i.e. finite, ordered trees, where all nodes—except the root—are labeled with a literal). They are also called *tableaux*. A tableau is *closed* iff each of its branches is closed; otherwise it is called *open*. A literal tree is represented as the set of its branches; branch sets are denoted by the letters $\mathcal{P}$, $\mathcal{P}'$ etc. We write $\mathcal{P}, \mathcal{P}'$ and mean $\mathcal{P} \cup \mathcal{P}'$. The *extension* of p with a clause $C = L_1 \vee \cdots \vee L_n$, written as $p \circ C$, is the branch set $p.[L_1], \ldots, p.[L_n]$ where the dot denotes concatenation of branches. Equivalently, in tree view this operation extends the branch p by n new nodes $N_1, \ldots, N_n$ that are labeled with the respective literals from C.

The calculus stated in [Baumgartner, 1998] removes one of the major weaknesses of hyper-tableau (introduced in [Baumgartner *et al.*, 1996]), namely guessing ground-instances of clauses sometimes. This is replaced by a unification-driven technique. The calculus is analytical in the sense that only input clauses and instances thereof are used in the tableaux. It is similar to hyper-resolution, because all body literals of a clause have to be solved simultaneously. In addition, the calculus is *confluent*. This means, proof procedures never have to undo inferences. There are three inference rules in the calculus:

1. extension steps for building tableaux and closing branches,
2. link steps for creating new instances of clauses, and
3. redundancy criteria for finishing derivations that are useful for model generation.

2.2 Extension and Link Steps

Let us now review the *hyper-tableau* calculus as stated in [Baumgartner, 1998]. A hyper-tableau refutation for a (possibly non-ground) clause set is the construction of a closed clausal tableau (i.e. a tableau where every branch is labeled as closed), starting with the tableau which consists of the root node only. Tableaux are equipped with a branch selection function: for every open tableau exactly one open branch is *selected* (arbitrarily), and inferences may be applied to this selected branch only. Selection is indicated by underlining all literals in the respective branch. The tableau construction must be *fair* wrt. the application of the two inference rules *extension* and *link* modulo some redundancy criteria. As usual, fairness means that every possible application of an inference rule must be carried out eventually unless shown to be redundant.

We first consider the *extension* rule; its application can be described as follows: let p be the selected branch; take a clause $\mathcal{A} \leftarrow \mathcal{B}$ from the current clause set $\mathcal{C}$ (which is initialized with the given input clause set), and apply to p the usual β-rule for tableaux (see e.g. [Fitting, 1996]) with $\mathcal{A} \leftarrow \mathcal{B}$ (i.e. we split the clause below the leaf of p). But this is done only if there is a most general substitution σ such that every element $B\sigma \in \mathcal{B}\sigma$ is equal to a variant of a literal L from p. Then, all new branches with leaf $\neg B\sigma$ where $B\sigma \in \mathcal{B}\sigma$ are labeled as closed; the new branches (if any) with leafs from $\mathcal{A}\sigma$ are labeled as open. If there is an open branch in the resulting tableau, we select one. Let us consider an example now:

Example 1 (indirect successor loop).

$$P(g(z)) \leftarrow \tag{1}$$
$$Q(f(x)) \leftarrow P(x) \tag{2}$$
$$P(f(y)) \leftarrow Q(y) \tag{3}$$
$$\leftarrow P(f(f(g(0)))) \tag{4}$$

For this example, a hyper-tableau derivation is shown in Fig. 1. Underlining is used to indicate the selected branch and the selected literals in the tableau clauses. Please note that the same variable names are used in several different tableau clauses for convenience; actually each clause is quantified individually. We start the derivation by extending the initial empty tableau with $\underline{P(g(z))}$.

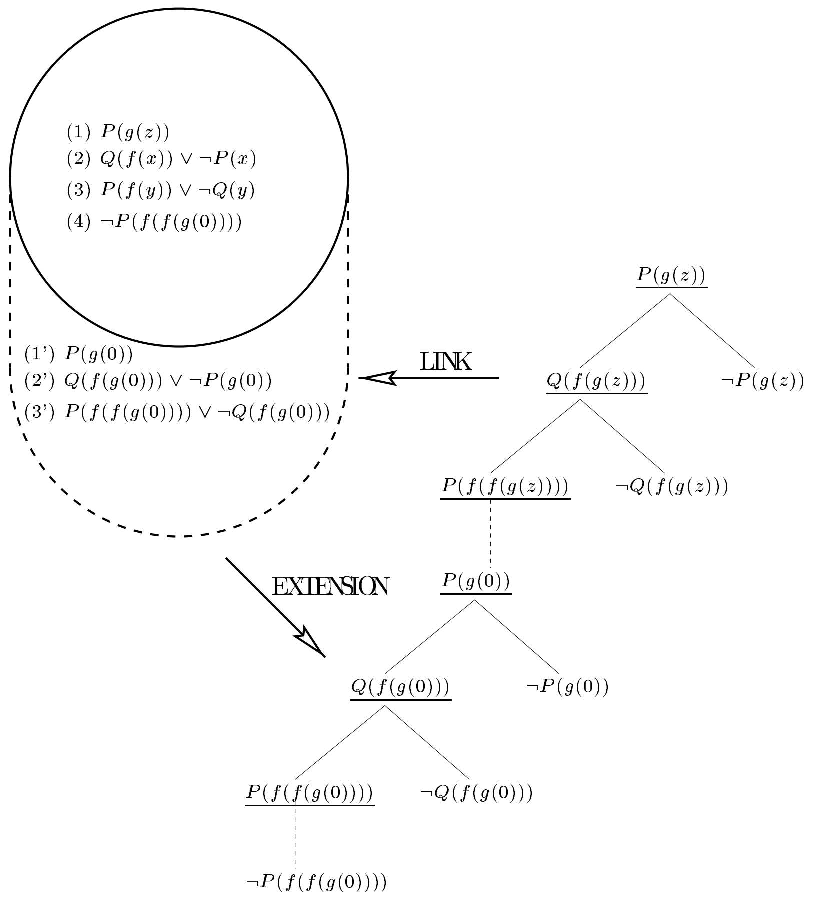


Figure 1: Hyper-tableau derivation for Ex. 1.

After three extension steps with instances from the clauses (1), (2) and (3), the extension rule need no longer be applied (although it could). At this point, we take new instances of already used clauses, in order to arrive at a closed tableau. They are generated by *link* steps, resulting in the (proper) instances (1'), (2') and (3') of the clauses (1), (2) and (3), respectively, shown in the extended clause set. For example, we obtain (3') by linking clause (3) with (4); we get the substitution $[y = f(g(0))]$. Linking (3') with (2) gives (2'), and linking (2') with (1) gives (1'). These instances are added to the current clause set. Now, after extension with (1'), the extension rule becomes applicable again. Finally, an extension step with (4) closes all branches. For details of the calculus, including the redundancy criteria, the reader is referred to [Baumgartner, 1998].

3 Loop-Detection by Examples

3.1 Detecting Simple Loops

Since, at the end, all branches are closed in Fig. 1, we obtain a refutation (i.e. the given clause set is unsatisfiable). However, if we do it without clause (4), then we run into a loop. We can repeat the block of extension (and link) steps between the dashed lines in Fig. 1 again and again, getting more and more complex terms. Unfortunately, the redundancy criteria in [Baumgartner, 1998] do not help us here. Therefore, in order to overcome this problem, we try identifying the loop involving clauses (2) and (3), similarly to related approaches. Resolving these clauses yields the cycle clause $P(x) \to P(f(f(x)))$ and hence the (non-idempotent) substitution $\lambda = [x = f(f(x))]$. In addition, the literal $(\neg)P(x)$ in (2) is unified with $P(g(z))$ from (1) by the substitution $\mu = [x = g(z)]$. This is shown in Fig. 2. There, the symbol # marks the only open branch, which has become finite now, in contrast to the situation in Fig. 1. We will introduce the formal details on how to compute λ and μ in Sect. 4.4.

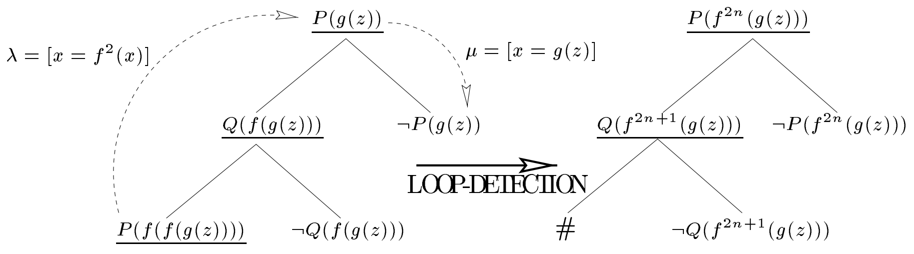


Figure 2: Loop-detection for Ex. 1.

Now we introduce the *loop-detection* rule informally. If we find a cyclic unifier λ of the form $[x_1 = e_1, \ldots, x_k = e_k]$ where only the variable x_i occurs in each e_i—no other variables—, then we can comprise the infinite derivation sequence by means of substitutions with terms with exponents. We have to express the substitution $\kappa = \lambda^*\mu$; it is the composition of the cyclic unifier λ (applied zero, one or more times) with the ordinary unifier μ. In Ex. 1, it is $\lambda^* = [x = f^{2n}(x)]$, $\mu = [x = g(z)]$, and thus $\kappa = [x = f^{2n}(g(z))]$, where only the substitution of the variable x from (2) is shown. In this context, we make use of the well-known notation for terms with exponents (see e.g. [Comon, 1995, Salzer, 1992]), which will be introduced formally in Def. 1.

After applying the substitutions on the part of the tableau which is involved in the loop, we arrive at a situation where any further extension is redundant, neglecting clause (4) here, since only instances of literals that are already on the branch can be created by further extension steps. From the only open branch (on the right side of Fig. 2), we read off the model for Ex. 1 without clause (4), namely:

$$\mathrm{I} = \{P(f^{2n}(g(z))),\ Q(f^{2n+1}(g(z)) \mid n \geq 0\}$$

3.2 Making Use of Computer Algebra Systems

Let us consider another, more complicated example now:

Example 2 (sum of naturals).

$$\begin{aligned} Add(t,0,t) &\leftarrow \\ Add(x,f(y),f(z)) &\leftarrow Add(x,y,z) \\ Sum(0,0) &\leftarrow \\ Sum(f(x),f(z)) &\leftarrow Sum(x,y), Add(x,y,z) \end{aligned}$$

The *Add* predicate realizes the addition in successor algebra, and the meaning of the *Sum* predicate is computing the sum of the first n natural numbers. Looking at Fig. 3, we notice that in fact the intended model

$$\mathrm{I} = \{Add(x,f^m(0),f^m(x)),\ Sum(f^n(0),f^{n(n+1)/2}(0)) \mid m,n \geq 0\}$$

can be computed. Please note that we omitted some (obsolete) side branches in Fig. 3 after loop-detection steps. In this example, the semantics of the *Add* predicate is computed quite similarly to the previous example during the first application of the loop-detection rule in this case. The loop-detection rule is applied twice here.

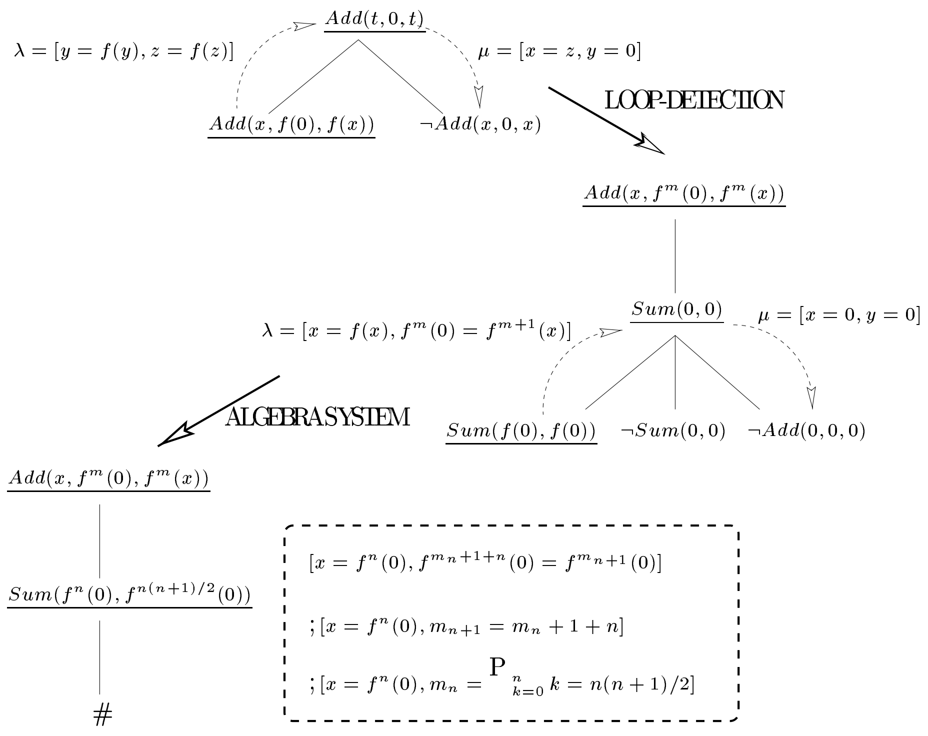
Again, parts of the problem can be solved by known (cyclic) unification procedures (see also Sect. 3.3). In this example, the cyclic unification of x with $f(x)$ and 0 yields $[x = f^n(0)]$, where n is a new variable introduced at this stage. However, we need some additional technique from computer algebra, since we have to face a unification problem of the form $s^{\varphi}(t) = s^{\psi}(t)$ here, namely $f^{m+1+n}(0) = f^m(0)$. We realize immediately, that the ms at both sides of this unification problem must be different. They denote successive instances, dependent on the new parameter n. This leads us to a recursive equation:

$$m_{n+1} = m_n + 1 + n \text{ where } m_0 = 0 \tag{5}$$

For computing the solution of (5), we can use a *computer algebra* system, which (in this context) serves as a tool for solving theorem proving problems. We immediately realize:

$$m_n = \sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

This sum can be resolved by a computer algebra system. However, recursive equations may be more complicated. In such cases, we can apply techniques from generating functions (see e.g. [Wilf, 1990, Graham *et al.*, 1994]). In Ex. 2, we then have to proceed as follows: the infinite series m_n is associated with the



$Add(t,0,t)$
$\lambda = [y = f(y), z = f(z)]$
$\mu = [x = z, y = 0]$
$Add(x, f(0), f(x))$
$\neg Add(x,0,x)$
LOOP-DETECTION
$Add(x, f^m(0), f^m(x))$
$Sum(0,0)$
$\lambda = [x = f(x), f^m(0) = f^{m+1}(x)]$
$\mu = [x = 0, y = 0]$
$Sum(f(0), f(0))$
$\neg Sum(0,0)$
$\neg Add(0,0,0)$
ALGEBRASYSTEM
$Add(x, f^m(0), f^m(x))$
$Sum(f^n(0), f^{n(n+1)/2}(0))$
#
$[x = f^n(0), f^{m_n+1+n}(0) = f^{m_{n+1}}(0)]$
$;[x = f^n(0), m_{n+1} = m_n + 1 + n]$
$;[x = f^n(0), m_n = \sum_{k=0}^{n} k = n(n+1)/2]$

Figure 3: Loop-detection for Ex. 2.

generating function $G(z) = \sum_{n=0}^{\infty} m_n z^n$, and the recursive equation (5) (from above) is transformed into:

$$G(z)/z = G(z) + \sum_{n=0}^{\infty}(1+n)z^n = G(z) + \frac{1}{(1-z)^2}$$

This is equivalent with $G(z) = z/(1-z)^3$. The Taylor series expansion of $G(z)$ then yields $m_n = n(n+1)/2$ as expected. The stated equations may be solved by computer algebra systems like *Mathematica* [Wolfram, 1996]. There, the procedure with generating functions just sketched is already available in the standard add-on package *DiscreteMath'RSolve'* [Martin, 1996]. Fig. 4 shows a Mathematica session, in which the closed form for m_n is computed.

Since we consider polynomial expressions as exponents here, we expect that the overall problem of unifying such generalized terms with exponents is undecidable, because we are able to express Hilbert's 10th problem. But since the loop-detection rule is only optional and we can mimic every step of the plain hyper-tableau calculus within our new calculus—please note that always not only the terms with exponents but also the simple terms in their denotations are available for extension and link steps of the extended calculus—, the overall procedure remains complete. We just have to guarantee that loop-detection is

```
In[1]:= <<DiscreteMath`RSolve`

In[2]:= RSolve[ { m[n+1] == m[n]+1+n, m[0] == 0 }, m[n], n ]

                               (1 + n) (2 + n)
Out[2]= {{m[n] -> -1 - n + ---------------}}
                                      2

In[3]:= Simplify[%]

                     n (1 + n)
Out[3]= {{m[n] -> ---------}}
                         2
```

Figure 4: Mathematica session for Ex. 2.

not applied in an unfair manner (i.e. the application of other rules is not deferred infinitely long). In summary, the use of terms with generalized integer exponents makes powerful loop-detection and model generation possible; we will discuss model generation in greater detail in Ex. 3.

3.3 Background

The techniques for computing cyclic substitutions are not new (see e.g. [Comon, 1995, Salzer, 1992, Socher-Ambrosius, 1993, Klingenbeck, 1997, Peltier, 1997a]). However, their application to hyper-tableau (and to tableaux in general) is new. Nevertheless, [Klingenbeck, 1997] makes use of terms with exponents in a tableau calculus. But there it is not possible to handle loops involving more than one clause, only (binary) cyclic (self-resolvent) clauses are considered as in [Bibel *et al.*, 1992, Ohlbach, 1998]. In contrast to this, Ex. 1 includes an indirect loop, involving the two clauses (2) and (3), which can be handled by our loop-detection rule. The loop-detection rule may even handle in some cases what is called regularity in tableau calculi [Letz *et al.*, 1994]. Then we have the case that the literals causing the loop are identical, thus it holds $\lambda = \epsilon$ (i.e. the empty substitution).

Note that in this context, in addition, we exploit computer algebra systems for (Herbrand) model generation. This is a rather new idea. So far, the connection of theorem proving with computer algebra systems usually is the other way round: theorem provers work as assistants of computer algebra systems. For example, the *Theorema* project (undertaken in Linz, Austria, headed by Buchberger) aims at integrating proving support into computer algebra systems. The emphasis is on proof generation for routine parts of proofs of theorems from analytical mathematics [Buchberger, 1997]. But in this paper, computer algebra systems work as assistants of theorem proving systems, such that searching for refutational proofs becomes more efficient and model generation is possible. Of course, a combination of both perspectives is thinkable and may also be useful.

4 Inference Rules for the Enhanced Calculus

Now is the time to give a more formal treatment of the hyper-tableau calculus enhanced by terms with exponents and hence more powerful loop-detection. We will do this by adapting the notions and notations from [Baumgartner, 1998]. Therefore, we will have modified versions of the extension and link rules in our calculus (see Defs. 3 and 4). First of all, we have to extend the definition of a clause. The major difference is that, instead of simple terms (without exponents), we may also have terms with general integer exponents occurring in some clause literals.

Definition 1 (terms with general exponents). The set T of *terms with general exponents* is the smallest set satisfying the following properties:

1. All Herbrand variables $x, y, z, \ldots$ are in T.
2. If the terms $s_1, \ldots, s_n$ are in T, then also $f(s_1, \ldots, s_n)$, where f is an n-ary function symbol. For $n = 0$, the symbol f denotes a *constant*.
3. The *hole* symbol $\diamond$ is also in T. However, the hole is only allowed to appear in a context term (whose definition follows next).
4. If s and t are in T where s—called *context* or context term here—contains at least one occurrence of $\diamond$ but is not identical with it, and φ is a general integer expression (see its definition below), then $s^{\varphi}(t)$—called *term with exponent*—is in T.

A *general integer expression* φ is an arithmetic expression denoting a total function mapping several natural number *parameters* $l, m, n, \ldots$ to a natural number (including zero).

Let us consider the term with exponents $t = h(f(\diamond)^n(a), g(\diamond)^{2n+1}(b))$ as an example. It contains two contexts, namely $f(\diamond)$ and $g(\diamond)$ with the corresponding exponents n and $2n+1$. There is only one integer parameter (namely n) occurring in both exponents. For terms with exponents of the form $f(\diamond)^{\varphi}(s)$ (i.e. with unary function symbols f), we write also $f^{\varphi}(s)$ for short. Thus, the term t from above can also be stated as $h(f^n(a), g^{2n+1}(b))$. We already made use of this notation earlier.

What is the meaning of terms with exponents? Generally speaking, each term denotes a set of terms without exponents. Let us state this more formally now. For natural numbers k, we define:

$$s^k(t) = \begin{cases} t & \text{if } k = 0, \\ s[\diamond = s^{k-1}(t)] & \text{otherwise} \end{cases}$$

As expected, the *denotation* of a term with exponents $s^{\varphi}(t)$ is the set of all $s^k(t)$ where k is the value of φ for some parameter instantiation. The generalization to terms in general, literals or even clauses is straightforward. For example, the denotation of the term t from above—written $\|t\|$—consists of the terms $h(a, g(b)), h(f(a), g(g(g(b)))), h(f^2(a), g^5(b)), \ldots$ for $n = 0, 1, 2, \ldots$ Note that n is inserted simultaneously in both exponents. We also extend the notion *interpretation* on terms with exponents as expected: a sentence X (possibly containing terms with exponents) is true in an interpretation I iff I $\models \chi$ for all $\chi \in \|X\|$.

4.1 Initialization

In the following, we consider literal trees equipped with a *branch selection function* which assigns to every open literal tree one of its open branches. We write $p, \mathcal{P}$ to indicate that p is selected in the branch set $p \cup \mathcal{P}$. Furthermore, every open branch p is labeled with a finite set of clauses, which is denoted by $\mathcal{C}^-(p)$. Intentionally, $\mathcal{C}^-(p)$ provides the current clause set so to speak, whose members can be used for extension steps. Alternatively, we will also write $p : \mathcal{C}^-$ and mean the branch p with $\mathcal{C}^-(p) = \mathcal{C}^-$.

There is another set $\mathcal{C}^+(p)$ of tableau clauses of p, namely those clauses which were used in extension steps to construct p. Since p can be understood as a branch through $\mathcal{C}^+(p)$, it is natural that p determines a respective selection of head literals of the clauses in $\mathcal{C}^+(p)$. A *clause with selection* is a program clause (i.e. with at least one head literal) where one of its head literals L is labeled (in some distinguished way). L is called the *selected literal*, it is also denoted by $\underline{C}$.

Definition 2 (initialization rule).

$$\frac{}{[\,] : \mathcal{C}}$$

for given finite clause set $\mathcal{C}$ without selection. Here, $[\,]$ denotes the empty branch.

4.2 The Extension and the Link Rule

We are now ready to state the extension and link rules for the enhanced calculus. They are very similar to the ones in [Baumgartner, 1998]. But there are in fact differences, that are, however, somewhat hidden. First of all, simple term unification is replaced by unification of terms with general exponents, called *exponential unification*. As a consequence of this, the current clause set $||\mathcal{C}^-(p)||$ for some branch p may contain infinitely many simple terms, after a loop-detection step has been performed. Nevertheless, each branch has only finite length, but it may contain terms with generalized exponents.

In the following, we need the notion of a *variant*. Since we consider terms with exponents here, the usual definition for simple terms without exponents has to be extended. s and t are called variants—written $s \sim t$—iff the ground instantiations of $||s||$ and $||t||$ are identical. However in practice, we may restrict to a simpler notion: $s \sim t$ iff there is a variable renaming ρ, that is a bijection substituting term variables by term variables and integer variables by integer variables, such that $s\rho$ and $t\rho$ become syntactically identical. Obviously, both versions of the definition reduce to the usual definition of variant, if we restrict our attention to simple terms only. So, let us now adapt the inference rules from [Baumgartner, 1998].

Definition 3 (extension rule).

$$\frac{p : \mathcal{C}^-, \mathcal{P} \qquad \mathcal{A} \leftarrow \mathcal{B}}{p \circ (\mathcal{A} \leftarrow \mathcal{B})\sigma, \mathcal{P}}$$

where

1. $p, \mathcal{P}$ is a branch set with selected branch p, and
2. $(\mathcal{A} \leftarrow \mathcal{B})$ is fresh copy of a clause in $\mathcal{C}^-(p)$ or in its denotation with new variables, and
3. there are new variants $C_1, \ldots, C_k$ from clauses in $\mathcal{C}^+(p)$ or in its denotation with no variables in common, and
4. σ is a most general multi-set unifier of $\mathcal{B}$ and $\{\underline{C_1}, \ldots, \underline{C_k}\}$, and
5. $C_i \sim C_i\sigma$, for every i, $1 \leq i \leq k$, and
6. every new branch $p.[\neg B\sigma] \in p \circ (\mathcal{A} \leftarrow \mathcal{B})\sigma$ where $B \in \mathcal{B}$, is closed, and
7. every new branch $p.[A\sigma] \in (\mathcal{A} \leftarrow \mathcal{B})\sigma$ where $A \in \mathcal{A}$ is open, and $\mathcal{C}^-(p.[A\sigma]) = \mathcal{C}^-(p)$.

If the extension rule is applied to a tableau as just stated, then we say that there is a *link* from the literals $B \in \mathcal{B}$ to $L \in \{\underline{C_1}, \ldots, \underline{C_k}\}$ iff σ unifies the literals B and L (disregarding their polarity). In this context, σ is also called *linking substitution*. These notions should not be mixed with the following link rule.

Definition 4 (link rule).

$$\frac{p : \mathcal{C}^-, \mathcal{P} \qquad \mathcal{A} \leftarrow \mathcal{B}}{p : \mathcal{C}^- \cup \{C_1\sigma, \ldots, C_n\sigma\}, \mathcal{P}}$$

where

1. $p, \mathcal{P}$ is a branch set with selected branch p, and
2. $(\mathcal{A} \leftarrow \mathcal{B})$ is a clause in $\mathcal{C}^-(p)$ or in its denotation, and
3. there are new variants $C_1, \ldots, C_k$ from clauses in $\mathcal{C}^+(p)$ or in its denotation with no variables in common, and
4. σ is a most general multi-set unifier of $\mathcal{B}$ and $\{\underline{C_1}, \ldots, \underline{C_k}\}$, and
5. $C_i\sigma \not\sim C_i$, for some i, $1 \leq i \leq k$.

4.3 Loop-Detection and Computer Algebra

Now we come to the topic of loop-detection by means of computer algebra techniques. First, we will state the loop-detection rule which is able to detect even indirect loops, involving possibly more than one clause which need not necessarily be binary. In this procedure we need a generalized unification procedure, that does not deal only with terms with linear arithmetic expressions in the exponent, but with more general expressions that can be computed by computer algebra systems such as *Mathematica* [Wolfram, 1996].

Definition 5 (loop-detection rule).

$$\frac{p : \mathcal{C}^-, \mathcal{P}}{\tilde{p} : \tilde{\mathcal{C}}^-, \tilde{\mathcal{P}}}$$

where

1. $p = p_1.[L_1].p_2.[L_2]$ is the only open branch in the tableau with prefix p_1 (because otherwise the tableau would become infinitely wide), and

2. $\tilde{p} = p_1.r$ where $r = ([L_1'].p_2')\kappa$ and κ is the exponential unifier of L_1 and L_2, and L_1', and p_2' are the literal and the branch in the tableau $\mathcal{P}'$ (introduced below) corresponding to the respective components of p, and
3. All links between clauses in $\mathcal{C}^+(r)$ lead to different nodes, and for each such pair of literals L_1 and L_2 it holds $L_1 \sim L_2$ (disregarding their polarity).
4. $\tilde{\mathcal{C}}^- = \mathcal{C}^- \cup \{C\kappa \mid C \in \mathcal{C}^-([L_1'].p_2'.[L_2'])\}$ (i.e. both the terms with exponents and parts of their denotations are kept in $\mathcal{C}^-$), and
5. $\tilde{\mathcal{P}}$ is the part of the tableau $\mathcal{P}'$ (introduced below), that corresponds to the parts at the same positions in $\mathcal{P}$.

It remains to explicate the term *exponential unifier* κ and the new tableau $\mathcal{P}'$. We do this in several stages. At first, there must be a link between some literal L_0, which must be a (negative) leaf literal of a closed branch with prefix $p_1.[L_1]$, and the literal L_1, which stems from a clause $C \in \mathcal{C}^-(p_1)$. In general, there may be one or more such literals L_0 to choose from. Clearly, L_0 is an instance of a literal $L \in \mathcal{B}$ of a clause $(\mathcal{A} \leftarrow \mathcal{B}) \in \mathcal{C}^-(q)$ for some prefix q of p. The latter clause must have been added by an extension step with the linking substitution σ. This σ is called μ in this context.

Next, we undo the effect of the substitution μ in the whole tableau, such that there is no link between L_0 and L_1 any more. This means, we consider a tableau $\mathcal{P}'$ which is identical with the given one, except that, in part 4 of the above-mentioned extension step, σ only is a multi-set unifier of $\mathcal{B} \setminus \{L\}$ and $\{\underline{C_1}, \dots, \underline{C_k}\} \setminus \{L_1\}$. Let now L_0', L_1' and L_2' be the literals in the newly constructed tableau $\mathcal{P}'$ which stand at the positions corresponding to the positions of the literals L_0, L_1 and L_2 in the original tableau. In addition, let L_1' be the atom of L_0' (i.e. the negation sign is dropped). Now, all term and integer variables in L_0' are equipped with the index $n+1$, all variables in L_1' are equipped with the index 0, and all variables in L_2' are equipped with the index n, where n is a new integer variable. We denote the resulting terms by $(L_0')_{[n+1]}$, $(L_1')_{[0]}$ and $(L_2')_{[n]}$, respectively.

Before we continue with more formal details, let us give an example for illustration: Fig. 5 (a) shows the tableau $\mathcal{P}$ for Ex. 2 before the second loop-detection step with computer algebra. Links are denoted by thick lines. They constitute the substitution μ. In Fig. 5 (b), the same tableau is shown, but without link between L_0 and L_1; this is the tableau $\mathcal{P}'$. By equating L_0' and L_2' after annotating the indices $n+1$ or n, respectively, we get λ (as shown in the figure).

4.4 Cyclic Unification

How can the exponential unifier be computed exactly? We will not state the complete unification procedure here, because this has been done elsewhere and would require too much space (see e.g. [Comon, 1995, Socher-Ambrosius, 1993, Peltier, 1997a, Klingenbeck, 1997]). But for the sake of completeness and in order to make the paper more self-contained, the simplification rules according to [Socher-Ambrosius, 1993] are shown in Fig. 6, presented as a rule- or constraint-based approach to unification similar to [Jouannaud and Kirchner, 1991], who view unification as solving equations in abstract algebras in a completely declarative manner. The rules of this system have to be applied non-deterministically (i.e. all alternative rule applications have to be explored in order to find all solutions κ). Equations are considered as not oriented pairs here. In the case of

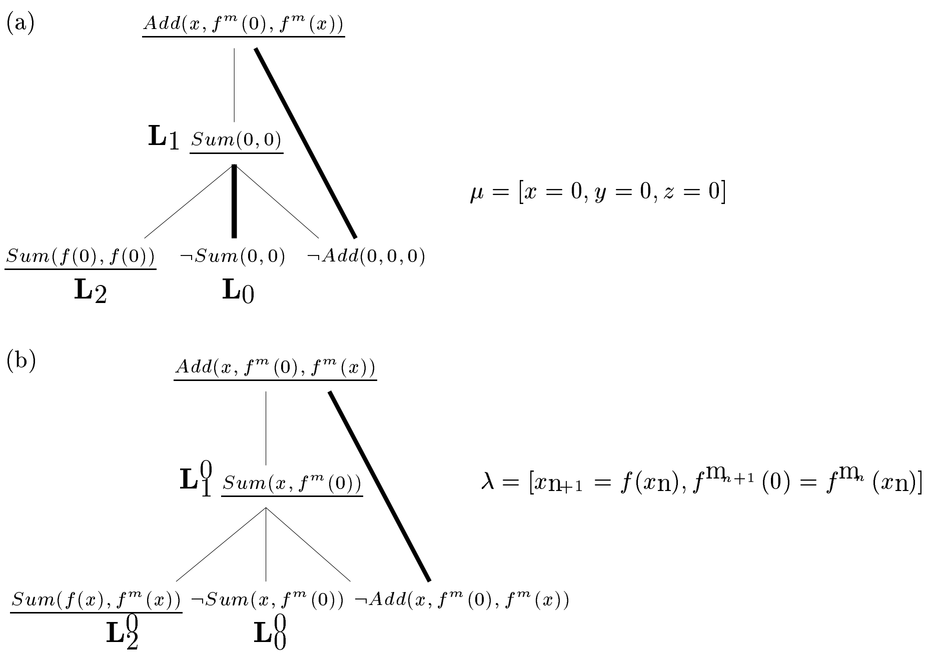


Figure 5: Determining μ and λ for Ex. 2.

multiple solutions, the r in condition 2. of Def. 5 has to be appended several times for each computed κ in the branch $\tilde{p}$.

The simplification system is correct, but incomplete and may sometimes not terminate. So, actually, a more complicated procedure is necessary, which is also stated in [Socher-Ambrosius, 1993]. But, in principle, any one of the cyclic unification procedures in the literature can be used for our purpose. Most of them transform a given system of unification equations into a normal form where the exponential unifier can be easily read off. The terms may have exponents, but only linear Diophantine equations are admissible in the cited literature. In this paper, we drop this restriction and extend the cyclic unification algorithm, bringing computer algebra into play.

Now, in order to continue the computation of the exponential unifier, we apply the simplification rules of Fig. 6 as long as possible on the equation system

$$(L_1')_{[0]} = L_1 \ \wedge \ (L_0')_{[n+1]} = (L_2')_{[n]} \tag{6}$$

(disregarding the negative polarity of L_0'). The first equation is called the μ-part, whereas the second one is called the λ-part of the equation system; both together constitute the overall solution κ. The μ-part of this system is needed, in order to provide the base for the recursion. We will arrive at an equation system (or a disjunction thereof) of the form $x_1 = e_1 \wedge \cdots \wedge x_k = e_k \wedge E$ where E is an system of equations between terms that cannot be further simplified by the simplification rules of Fig. 6, because they equate terms with non-linear integer exponents, e.g. $f^{m_{n+1}}(0) = f^{m_n}(x_n)$. Since any loop-detection must be incomplete for theoretical reasons anyway, we only consider the case where

Trivial: $\dfrac{t = t \wedge E}{E}$, where t may be an arbitrary term (possibly with exponents), including variables and simple terms.

Clash: $\dfrac{s = t \wedge E}{\perp}$, if $head(s) \neq head(t)$.

Here, $head(t)$ means the leading function symbol of a non-variable term t, defined by $head(f(t_1, \dots, t_n)) = f$ and $head(s\ (t)) = head(s)$.

Occur-Check: $\dfrac{x = t \wedge E}{\perp}$, if $x \in var(t)$ where t must not be a variable.

With $var(X)$, we denote the set of all variables occurring in the expression X.

Merge: $\dfrac{x = t \wedge E}{x = t \wedge E[x = t]}$, if $x \notin var(t)$ and $x \in var(E)$, and t is a variable implies $t \in var(E)$.

Decompose: $\dfrac{s = t \wedge E}{s{\downarrow_1} = t{\downarrow_1} \wedge \cdots \wedge s{\downarrow_n} = t{\downarrow_n} \wedge E}$, if $head(s) = head(t)$.

Let t be a non-variable term with $head(t) = f$, and f has the arity n. Then, for $1 \leq i \leq n$, it is $t{\downarrow_i} = t_i$, if t is a simple term $f(t_1, \dots, t_n)$. For terms with exponents, due to the lack of space, we will only consider an example: Let $t = f(g(h(\diamond, a)), g(h(\diamond, a)))^{2n}(f(g(b), g(b)))$. At first, t is rewritten into $(f(\diamond, \diamond) \cdot g(\diamond) \cdot h(\diamond, a))^{2n+2=3}(b)$, where $\cdot$ denotes concatenation of contexts, and a fraction (2/3) occurs in the exponent. Then, $t{\downarrow_1} = t{\downarrow_2} = (g(\diamond) \cdot h(\diamond, a) \cdot f(\diamond, \diamond))^{2n+1=3}(b)$. Further details can be found in [Socher-Ambrosius, 1993, Klingenbeck, 1997].

Eliminate: $\dfrac{s\ (t) = u \wedge E}{(t = u \wedge E)\nu}$, where ν is a solution of the integer equation $\varphi = 0$.

Figure 6: Simplification rules for unification of terms with exponents.

1. every context occurring anywhere in the equation system does not contain any variables,
2. no x_i (on the left-hand side of equations) for $1 \leq i \leq k$ has the index n,
3. if an x_i has the index $n+1$, then e_i contains no variables except the variable x_i with index n.

Equations according to condition 3. have the form $x_{n+1} = t$ where t is a term with one or more occurrences of the variable x_n, e.g. $x_{n+1} = f(x_n)$. Now we replace each occurrence of x_n or x_{n+1} by $(t[x_n = \diamond])^n(x_0)$ or $(t[x_n = \diamond])^{n+1}(x_0)$, respectively. After that, further simplification may be possible by the simplification rules (of Fig. 6), such that we arrive at a simplification system $x'_1 = e'_1 \wedge \cdots \wedge x'_k 0 = e'_k 0 \wedge E'$ which is similar to the system from above, but all term variables with index n or $n+1$ are removed.

E' must have the form $s_1^{\varphi_1}(t_1) = s_1'^{\psi_1}(t'_1) \wedge \cdots \wedge s_l^{\varphi_l}(t_l) = s_l'^{\psi_l}(t'_l)$. Again, we only consider a restricted case, namely where $s_j = s'_j$ and $t_j = t'_j$ for $1 \leq j \leq l$. The only remaining task now is to solve the recursive integer equation system E $= (\varphi_1 = \psi_1 \wedge \cdots \wedge \varphi_l = \psi_l)$. This can be done automatically by means of a computer algebra system (as shown in Sect. 3). Provided that the computer algebra system was able to solve E, then we proceed as follows: substitute all integer variables m_n (or m_{n+1}) by the corresponding closed forms from the solution of E. After that, further simplification may be possible resulting in the desired substitution κ.

Let us again illustrate the progress of our method with Ex. 2. We start with the equation system (6) (see above), that is in this context:

$$x_0 = 0 \wedge f^{m_0}(0) = 0 \wedge x_{n+1} = f(x_n) \wedge f^{m_{n+1}}(0) = f^{m_n}(x_n)$$

Let us first draw attention to the equation $x_{n+1} = f(x_n)$. Together with $x_0 = 0$ this means, $x = f^n(0)$. From the remaining equations, we obtain E $= (m_0 = 0 \wedge m_{n+1} = m_n + 1 + n)$ as seen earlier. Finally, we get $\kappa = [x = f^n(0), m = n(n+1)/2]$ (see also Fig. 2). This substitution has to be applied to all literals in $\mathcal{P}'$ which are involved in the loop (i.e. the ones in $\mathcal{C}^+(r)$); corresponding variables linked according to condition 3. of Def. 5 are substituted by the same term.

4.5 Soundness and Completeness

We observe that the procedure for exponential unification may fail for at least two reasons: not only because in some cases unification is impossible, but also because of our restrictions set up in Sect. 4.4 or because the computer algebra system is not able to solve the recurrence equation system E, at least not in some reasonable time limit. In the latter case, the loop-detection rule should be stopped, and the other rules should be applied before trying the loop-detection rule again. This means, the new rule is incomplete, but the overall calculus remains complete, because loop-detection is only used as an additional, optional mechanism. Since the enhanced calculus presented here is very similar to the one stated in [Baumgartner, 1998], the model generation and the soundness and completeness proofs are analogous in both cases. We have the following theorem:

Theorem 6 (soundness and completeness). *Let $\mathcal{C}$ be a first-order clause set. Each fair derivation eventually leads to a closed tableau iff $\mathcal{C}$ is unsatisfiable.*

Proof. For the direction from left to right (soundness), we prove the following proposition: Let $\mathcal{P} = (p_1, \ldots, p_k)$ be a tableau (branch set), built according to the rules from clauses of the clause set $\mathcal{C}$, where each branch p_i (for $1 \leq i \leq k$) is a sequence of literals $(L_{i1} \cdots L_{il_i})$. Then, it holds

$$\mathcal{C} \models \forall((L_{11} \wedge \cdots \wedge L_{1l_1}) \vee \cdots \vee (L_{k1} \wedge \cdots \wedge L_{kl_k})) \qquad (7)$$

where $\forall$ denotes the universal closure of all term variables in the formula. Now, if all branches are closed, they all contain complementary literals by construction. Hence, the formula in (7) is unsatisfiable, and therefore $\mathcal{C}$ must be unsatisfiable, too.

It remains to show the proposition (7). This can be done by induction on the number of derivation steps. The base case—an initialization step according to Def. 2—is trivial, since the empty branch set [ ] corresponds to a tautology $\top$, and clearly $\mathcal{C} \models \top$. In the induction step, we make a case distinction:

Extension Step (Def. 3): Extension steps are similar to the β-rule in tableaux and resolution in logic programming. Because of this, the standard soundness result (e.g. in the textbooks [Fitting, 1996, Lloyd, 1987, Chang and Lee, 1973]) can be adapted to this context.

Link Step (Def. 4): Since a link step does not change the actual tableau, but affects only the clause set $\mathcal{C}^-(p)$, the proposition is an immediate consequence of the induction hypothesis.

Loop-Detection (Def. 5): Here, soundness partly follows from the soundness of the simplification rules (see e.g. [Socher-Ambrosius, 1993, Peltier, 1997a]). In addition, since loop-detection can be seen as applying the extension rule (and possibly also the link rule) arbitrarily often, this case can be reduced to the previous ones.

The direction from right to left (completeness) is easy. Since the versions of the link and extension rules we use here (Defs. 3 and 4) contain the corresponding rules in [Baumgartner, 1998] as a special case—recall that always not only the terms with exponents but also the simple terms in their denotations are available for extension and link steps of the extended calculus—, the completeness result in [Baumgartner, 1998, Th. 12] can be simply adopted here without change. □

4.6 Model Generation

As a consequence of the last theorem, we can do model generation similar to [Baumgartner, 1998, Sect. 5]. In particular, if we have a finite open branch p in a tableau that is finished, then we can read off a model of the given clause set that can be built from the literals in p. A branch p is called *finished* iff any further extension or link steps yield only clauses that are already in $\mathcal{C}^-(p)$ or $\mathcal{C}^+(p)$ (or their denotation). Here in addition, the redundancy criteria in [Baumgartner, 1998, Def. 10] could be applied (after adaption to our extended calculus). Let us consider now an example for model generation.

Example 3 (the even or odd example).

$$\begin{aligned} R(x), R(f(x)) &\leftarrow \\ &\leftarrow R(x), R(f(x)) \end{aligned} \tag{8}$$

Fig. 7 shows a derivation for this example. After loop-detection, the selected branch (ending with #) cannot be properly extended further. This means it is finished, and we can read off a model from this branch according to the subsequent Def. 7. Provided that the only constant and function symbols in our language are 0 and f, there are exactly two models for Ex. 3:

$$\mathrm{I}_1 = \{R(f^{2n}(0) \mid n \geq 0\} \text{ and } \mathrm{I}_2 = \{R(f^{2n+1}(0) \mid n \geq 0\}$$

Definition 7 (model generation). Let $\mathcal{P}$ be a hyper-tableau, built according to the rules of our calculus from clauses of a clause set $\mathcal{C}$, with an open but finished branch p. Now, p constitutes a Herbrand model I of $\mathcal{C}$, consisting of all ground atoms that are produced by p. We say a ground atom A is *produced* by the branch p in $\mathcal{P}$ iff the following conditions hold:

1. A is a ground instance of a selected literal L in a clause $C \in \|\mathcal{C}^+(p)\|$ via the (ground) substitution γ (i.e. $A = L\gamma$).

2. There is no clause $C' \in \|\mathcal{C}^-(p)\|$ such that $C \succ C' \precsim C\gamma$.
   Here, for clauses C_1 and C_2, we write $C_1 \precsim C_2$ ($C_1 \succ C_2$) and mean C_1 is *more general* (*strictly more general*) than C_2 iff there is an ordinary substitution δ (which in the strict case must not be a variable renaming, i.e. $C_1 \not\sim C_2$), such that $C_1\delta = C_2$.

The first model I_1 can be determined from the selected branch p, while I_2 can be determined from another branch further right (not shown in Fig. 7). For the construction of I_1, note that the clauses

$$\underline{R(f^{2n}(x))} \vee R(f^{2n+1}(x)) \tag{9}$$

$$\text{and} \quad R(f^{2n+1}(x)) \vee \underline{R(f^{2n+2}(x))} \tag{10}$$

are contained in $\mathcal{C}^-(p)$; both clauses are instances of clause (8). Hence, for example, $R(0)$ is in I_1, because it is produced by the selected literal in (9). However, $R(f(0))$ is not in I_1, although it also is an instance of the selected literal in (9)—we just have to apply the substitution $[x = f(0)]$—, because it is not produced by this literal, but by the unselected literal in (10). Of course, model generation can never be complete, because satisfiability is undecidable for first-order clause sets in general. In addition, there may be uncountably many (minimal) Herbrand models for a clause set; in order to see this, just consider a clause set consisting only of clause (8).

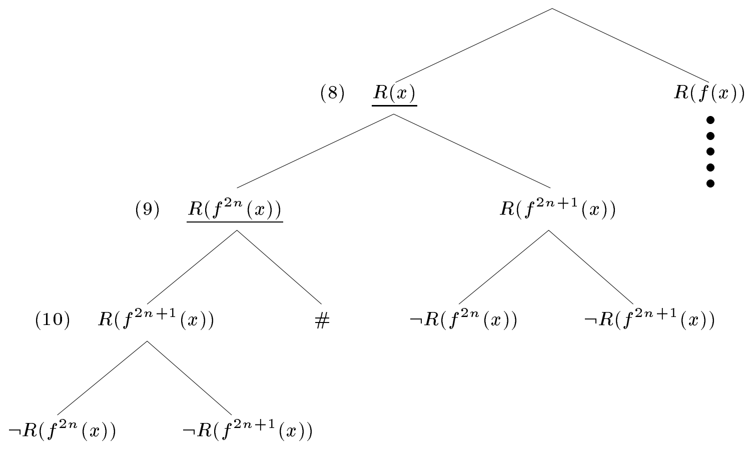


Figure 7: Model generation for Ex. 3.

5 Other Approaches

5.1 Generating Finite and Infinite Models

Since searching for counter examples is as important as the search for proofs, there are numerous works on model generation in theorem proving. For example, [Slaney *et al.*, 1994] presents a finite enumeration method which is able to build finite models. The restriction to finite models is overcome in [Caffera and Zabel, 1992, Caffera and Peltier, 1997]. This approach also allows building infinite models by making use of equational constraints. However, this approach is not able to represent the model of Ex. 1 (disregarding the last clause). Hence, in [Peltier, 1997a] equational constraints are enhanced by terms with exponents. However, there are only linear integer expressions considered as exponents; this restriction is dropped here. In addition, the calculus developed by Caffera *et al.* is not confluent (in contrast to the one presented here), if its rule for generating many pure literals is applied, because it only preserves satisfiability but not equivalence of clauses sets.

5.2 Approaches Based on Formal Languages

There are simple examples that cannot be handled by terms with exponents:

Example 4 (alternative clauses).

$$\begin{aligned} P(0) &\leftarrow \\ P(f(x)) &\leftarrow P(x) \\ P(g(y)) &\leftarrow P(y) \end{aligned}$$

In consequence, [Peltier, 1997b] proposes the use of tree grammars which is also done in [Heintze, 1992]. Nevertheless tree grammars are less expressive than context-free grammars. [Matzinger, 1997] observes that with tree grammars only finite models (not necessarily finite Herbrand models) can be expressed. Other grammar types are proposed, e.g. so-called primal grammars [Salzer, 1994, Hermann and Galbavý, 1997]. Also, indexed grammars of the IO-type may be used. They are more general than context-free grammars and allow to mimic terms with linear integer exponents. The word problem for these class of grammars is tractable [Asveld, 1981]. Nevertheless, intersection and hence unification of terms cannot be decidable. This, of course, is a severe drawback. In summary, each approach has its advantages and disadvantages.

In order to be able to treat examples like Ex. 4, we may introduce disjunctive terms. Its model could be stated e.g. as:

$$\mathrm{I} = \{P((f(\diamond) \sqcup g(\diamond))^n(0)) \mid n \geq 0\}$$

We have to introduce an additional inference rule for such *alternatives*, which detects more than one loop at once. The formal details of a loop-detection rule with alternatives have to be worked out. For this, loop-detection has to be postponed, in order to detect multiple loops via alternative terms. An appropriate control strategy is required for deciding when loop-detection should be applied or not. This is still ongoing work.

6 Conclusions and Future Works

In this paper, we have extended the hyper-tableau calculus by terms with generalized exponents. Although the new inference rule for loop-detection is incomplete in general, the overall method remains complete wrt. computing answers.

Because of the use of methods from computer algebra, we have model generating capabilities that are enhanced compared with other approaches. To the best of my knowledge, the procedure proposed here, is the first one that

1. makes use of an incomplete loop-detection rule,
2. exploits the power of computer algebra systems, and
3. is applicable to tableau methods.

The methods for loop-detection and model generation are also applicable to other calculi than (hyper-)tableau. Future work should aim at making the loop-detection rule more complete. For this, especially unification should remain a tractable operation, regardless what kind of data structures we use for representing models. Another goal, of course, is implementing the calculus, after addressing the question of complexity of the involved procedures, such that the benefits of the combination of theorem proving, model generation, and computer algebra can be exploited in applications such as system diagnosis or debugging of axiomatizations [Furbach *et al.*, 1998].

Acknowledgments

I thank the audience of the workshop *Integration of Deduction Systems*, some anonymous referees, and my colleague *Peter Baumgartner* for helpful discussions and comments.

References

[Asveld, 1981] Peter R. J. Asveld. Time and space complexity of inside-out macro languages. *International Journal of Computer Mathematics*, 10:3–14, 1981.

[Baumgartner *et al.*, 1996] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper tableaux. In José Júlio Alferes, Luís Moniz Pereira, and Ewa Orlowska, editors, *Proceedings of the European Workshop on Logics in Artificial Intelligence*, LNAI 1126, pages 1–17, Èvora, Portugal, 1996. Springer, Berlin, Heidelberg, New York.

[Baumgartner, 1998] Peter Baumgartner. Hyper tableau — the next generation. In Harrie de Swart, editor, *Proceedings of the 7th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 1397, pages 60–76. Springer, Berlin, Heidelberg, New York, 1998.

[Bibel *et al.*, 1992] Wolfgang Bibel, Steffen Hölldobler, and Jörg Würtz. Cycle unification. In Deepak Kapur, editor, *11th International Conference on Automated Deduction*, LNAI 607, pages 94–108, Saratoga Springs, New York, USA, June 15–18, 1992. Springer-Verlag.

[Buchberger, 1997] Bruno Buchberger, editor. *1st International Theorema Workshop*, RISC-Linz Report 97-20, Hagenberg, Austria, 1997. RISC.

[Bundy, 1994] Alan Bundy, editor. *12th International Conference on Automated Deduction*, LNAI 814, Nancy, France, June 26–July 1, 1994. Springer-Verlag.

[Caffera and Peltier, 1997] Ricardo Caffera and Nicolas Peltier. A new technique for verifying and correcting logic programs. *Journal of Automated Reasoning*, 19(3):277–318, 1997.

[Caffera and Zabel, 1992] Ricardo Caffera and Nicolas Zabel. A method for simultaneous search for refutations and models by equational constraint solving. *Journal of Symbolic Computation*, 13:613–641, 1992.

[Chang and Lee, 1973] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, London, 1973.

[Comon, 1995] Hubert Comon. On unification of terms with integer exponents. *Mathematical Systems Theory*, 28(1):67–88, 1995.

[Fitting, 1996] Melvin Fitting. *First-Order Logic and Automated Theorem Proving.* Texts and Monographs in Computer Science. Springer, Berlin, Heidelberg, New York, 2nd edition, 1996.

[Furbach *et al.*, 1998] Ulrich Furbach, Michael Kühn, and Frieder Stolzenburg. Model-guided proof debugging. Fachberichte Informatik 6/98, Universität Koblenz, 1998.

[Graham *et al.*, 1994] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics.* Addison-Wesley, Reading, MA, 2nd edition, 1994.

[Heintze, 1992] Nevin Heintze. Practical aspects of set-based analysis. In Krzysztof R. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 765–779, Cambridge, MA, London, 1992. MIT Press.

[Hermann and Galbavý, 1997] Miki Hermann and Roman Galbavý. Unification of infinite sets of terms schematized by primal grammars. *Theoretical Computer Science*, 176:111–158, 1997.

[Jouannaud and Kirchner, 1991] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 257–321. MIT Press, Cambridge, MA, London, 1991.

[Klingenbeck, 1997] Stefan Klingenbeck. *Counter Examples in Semantic Tableaux.* DISKI 156. infix, Sankt Augustin, 1997. Dissertation.

[Letz *et al.*, 1994] Reinhold Letz, Klaus Mayr, and Christian Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13:297–337, 1994.

[Lloyd, 1987] John W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, Heidelberg, New York, 1987.

[Manthey and Bry, 1988] Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In Ewing Lusk and Ross Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction*, LNCS 310, pages 415–434, Argonne, IL, 1988. Springer, Berlin, Heidelberg, New York.

[Martin, 1996] Emily Martin, editor. *Mathematica 3.0 Standard Add-on Packages.* Wolfram Media, Champaign, IL, 1996.

[Matzinger, 1997] Robert Matzinger. Computational representations of Herbrand models using grammars. In Dirk van Dalen, editor, *Proceedings of the Conference on Computer Science Logic 1996*, LNCS 1258, pages 334–348. Springer, Berlin, Heidelberg, New York, 1997.

[Ohlbach, 1998] Hans Jürgen Ohlbach. Elimination of self-resolving clauses. *Journal of Automated Reasoning*, 20:317–336, 1998.

[Peltier, 1997a] Nicolas Peltier. Increasing model building capabilities by constraint solving on terms with integer exponents. *Journal of Symbolic Computation*, 24:59–101, 1997.

[Peltier, 1997b] Nicolas Peltier. Tree automata and automated model building. *Fundamenta Informatica*, 30:23–41, 1997.

[Salzer, 1992] Gernot Salzer. The unification of infinite sets of terms and its applications. In Andrei Voronkov, editor, *Proceedings of the 3rd International Conference on Logic Programming and Automated Reasoning*, LNAI 624, pages 409–420. Springer, Berlin, Heidelberg, New York, 1992.

[Salzer, 1994] Gernot Salzer. Primal grammars and unification modulo a binary clause. In Bundy [1994], pages 282–295.

[Slaney *et al.*, 1994] John Slaney, Ewing Lusk, and William McCune. SCOTT: Semantically constrained Otter (system description). In Bundy [1994], pages 764–768.

[Socher-Ambrosius, 1993] Rolf Socher-Ambrosius. Unification of terms with integer exponents. Technical Report MPI-I-93-217, Max-Planck-Institut für Informatik, Saarbrücken, 1993.

[Wilf, 1990] Herbert S. Wilf. *generatingfunctionology*. Academic Press, San Diego, London, 1990.

[Wolfram, 1996] Stephen Wolfram. *The Mathematica Book*. Wolfram Media, Champaign, IL, 3rd edition, 1996.