# Model Generation from Constrained Free Variable Tableaux

Jan van Eijck[⋆]

CWI and ILLC, Amsterdam, Uil-OTS, Utrecht; jve@cwi.nl

**Abstract.** The tableau substitution rule in free variable tableau reasoning is destructive, for in general, $\boldsymbol{T}$ has consequences that $\boldsymbol{T}\sigma$ lacks. We show how this destructive feature can be eliminated in favour of a set-up that replaces tableau substitution with the generation and incremental merge of variable constraints on tableau branches. The approach differs from other constraint based techniques in tableau reasoning in that we constrain tableau branches rather than clauses, and use disunification constraints rather than unification constraints. We prove soundness and completeness, with the completeness proof based on a new way to generate models from open tableaux.

## 1 Basic Definitions

*Language.* Let $\Sigma$ be a first order signature, and $F_{\mathbf{sko}}$ an infinite set of skolem functions, with $F_{\mathbf{sko}} \cap F_{\Sigma} = \emptyset$. Call the extended signature $\Sigma^*$. If $v$ ranges over variables, $f$ over function symbols and $P$ over predicate symbols, then the extended language $\mathcal{L}_{\Sigma^*}$ is given by:

$$\begin{aligned} t &::= v \mid f(t_1, \dots, t_n) \text{ where } f \text{ n-ary} \\ S &::= \top \mid \bot \mid v_1 \approx t_1 \wedge \dots \wedge v_n \approx t_n \mid S_1 \wedge S_2 \\ C &::= \top \mid \bot \mid v_1 \not\approx t_1 \vee \dots \vee v_n \not\approx t_n \mid C_1 \vee C_2 \\ \phi &::= \top \mid \bot \mid P(t_1, \dots, t_n) \text{ where } P \text{ n-ary} \mid \neg\phi \mid \\ &\quad\ \phi_1 \wedge \dots \wedge \phi_n \mid \phi_1 \vee \dots \vee \phi_n \mid \forall v\phi \mid \exists v\phi \end{aligned}$$

An expression of the form $S$ is called a *substitution expression*, an expression of the form $C$ a *variable constraint*. The definitions of literals and clauses are as usual.

*Substitutions.* A *substitution* $\sigma$ is a function $V \to T_{\Sigma^*}$ that makes only a finite number of changes, i.e., $\sigma$ has the property that $\mathrm{dom}\,(\sigma) = \{v \in V \mid v\sigma \neq v\}$ is finite. We use $\epsilon$ for the substitution with domain $\emptyset$ (the identity substitution), and we represent a substitution $\sigma$ in the standard way, as a list

$$\{v_1 \mapsto v_1\sigma,\ \dots, v_n \mapsto v_n\sigma\},$$

where $\{v_1, \dots, v_n\}$ is $\mathrm{dom}\,(\sigma)$. Alternatively, substitutions can be viewed as conjunctions of equalities, i.e., $\sigma$ corresponds to the conjunction $v_1 \approx v_1\sigma \wedge \dots \wedge v_n \approx v_n\sigma$. We can interpret such conjunctions in the initial term algebra $T(\Sigma^*)$ (the algebra of all closed $\Sigma^*$ terms), by means of:

$$[\![\sigma]\!] := \{\alpha \in {}^{V}T(\Sigma^*) \mid T(\Sigma^*) \models_{\alpha} \sigma\}.$$

[⋆] Many thanks to the members of the *Dynamo team*: Balder ten Cate, Juan Heguiabehere and Breanndán Ó Nualláin.

It is not hard to see that this gives: $[\![\epsilon]\!] = [\![\top]\!] = {}^{V}T(\Sigma^*)$, and $[\![\sigma \wedge \rho]\!] = [\![\sigma]\!] \cap [\![\rho]\!]$. If $\sigma, \rho$ are substitutions, then $\sigma \preceq \rho$ if $\rho$ is less general than $\sigma$, i.e., if there is a $\theta$ with $\sigma = \rho\theta$. Note that $\sigma \preceq \rho$ iff $[\![\sigma]\!] \subseteq [\![\rho]\!]$.[1] The relation $\preceq$ is a pre-order (transitive and reflexive), and its poset reflection is a partial order. For this, put $\sigma \sim \rho$ if $\sigma \preceq \rho$ and $\rho \preceq \sigma$, and consider substitutions modulo renaming. This immediately gives: $\sigma \sim \rho$ iff $[\![\sigma]\!] = [\![\rho]\!]$.

The interpretations of substitution expressions in the initial term algebra under $\subseteq$ form a meet semi-lattice, with conjunction interpreted as $\cap$ (lattice meet), i.e., we have $[\![S_1 \wedge S_2]\!] = [\![S_1]\!] \cap [\![S_2]\!]$. If $[\![S_1 \wedge S_2]\!] = [\![\bot]\!] = \emptyset$ we say that $S_1$ and $S_2$ do not unify. In the other case, $S_1 \wedge S_2$ is (interpreted as) a most general unifier for $S_1$ and $S_2$. We get that $[\![\top]\!]$ (or $[\![\epsilon]\!]$), the set of all variable maps in the ground term algebra, is the top of the lattice, and $[\![\bot]\!] = \emptyset$ its bottom.

*Variable Constraints.* From a substitution $\sigma$, derive a variable constraint $\overline{\sigma}$ by complementation, as follows:

$$\overline{\sigma} = \bigvee\{v \not\approx \sigma(v) \mid v \in \mathrm{dom}\,(\sigma)\}.$$

E.g., the complement $\overline{\sigma}$ of $\sigma = \{x \mapsto a, y \mapsto b\}$ is $x \not\approx a \vee y \not\approx b$. Note that $\overline{\epsilon} = \bot$.

The interpretations of variable constraints in the initial term algebra under $\subseteq$ form a join semi-lattice, with disjunction interpreted as $\cup$ (lattice join), i.e., we have $[\![C_1 \vee C_2]\!] = [\![C_1]\!] \cup [\![C_2]\!]$. Again, the set of all variable maps in the ground term algebra is the top of the semi-lattice, and $[\![\bot]\!] = \emptyset$ its bottom. If $[\![C_1 \vee C_2]\!] = [\![\top]\!] = {}^{V}T(\Sigma^*)$ we say that $C_1$ and $C_2$ merge, or that $C_1 \vee C_2$ is universally satisfiable. We get that $[\![\bot]\!]$ (or $[\![\overline{\epsilon}]\!]$) equals $\emptyset$, so $\overline{\epsilon}$ only merges with $\top$. Call two constraints $C_1$ and $C_2$ *equivalent* if $[\![C_1]\!] = [\![C_2]\!]$.

*Tableaux, Closed and Open Branches.* A tableau over $\Sigma$ is a finitely branching tree with nodes labeled by $\mathcal{L}_{\Sigma^*}$ literals, or by variable constraints. A branch in a tableau $\boldsymbol{T}$ is a maximal path in $\boldsymbol{T}$. A branch $B$ in a tableau $\boldsymbol{T}$ is *closed* if $\bot$ occurs on $B$, otherwise $B$ is *open*. To explain how a tableau for a set $\Phi$ of $\mathcal{L}_{\Sigma^*}$ formulas is constructed we assume the usual Smullyan [8] typology:

| $\alpha$ | $\rightsquigarrow$ | $\alpha_1, \ldots, \alpha_n$ |
|---|---|---|
| $\phi_1 \wedge \cdots \wedge \phi_n$ | $\rightsquigarrow$ | $\phi_1, \ldots, \phi_n$ |
| $\neg(\phi_1 \vee \cdots \vee \phi_n)$ | $\rightsquigarrow$ | $\neg\phi_1, \ldots, \neg\phi_n$ |
| $\neg\neg\phi$ | $\rightsquigarrow$ | $\phi$ |
| $\neg\top$ | $\rightsquigarrow$ | $\bot$ |
| $\neg\bot$ | $\rightsquigarrow$ | $\top$ |

| $\beta$ | $\rightsquigarrow$ | $\beta_1, \ldots, \beta_n$ |
|---|---|---|
| $\phi_1 \vee \cdots \vee \phi_n$ | $\rightsquigarrow$ | $\phi_1, \ldots, \phi_n$ |
| $\neg(\phi_1 \wedge \cdots \wedge \phi_n)$ | $\rightsquigarrow$ | $\neg\phi_1, \ldots, \neg\phi_n$ |
| $\phi \Rightarrow \psi$ | $\rightsquigarrow$ | $\neg\phi, \psi$ |

| $\gamma$ | $\rightsquigarrow$ | $\gamma_1$ |
|---|---|---|
| $\forall v\phi(v)$ | $\rightsquigarrow$ | $\phi(v)$ |
| $\neg\exists v\phi(v)$ | $\rightsquigarrow$ | $\neg\phi(v)$ |

| $\delta$ | $\rightsquigarrow$ | $\delta_1$ |
|---|---|---|
| $\neg\forall v\phi(v)$ | $\rightsquigarrow$ | $\neg\phi(v)$ |
| $\exists v\phi(v)$ | $\rightsquigarrow$ | $\phi(v)$ |

The formulas decompose as follows: to decompose an $\alpha$ formula on a branch, extend the branch with $\alpha_1, \ldots, \alpha_n$; to decompose a $\beta$ formula on a branch, grow $n$ new leafs

[1] We write $\preceq$ for 'less general than' rather than the other way around, to get this natural correspondence with interpretations in the initial term model.

$\beta_1, \ldots, \beta_n$; to decompose a $\gamma$ formula, extend the branch with $\gamma(w)$, where $w$ is a variable that is fresh to the tableau; to decompose a $\delta$ formula, extend the branch with $\delta_1(\text{sko}_\delta(v_1, \ldots, v_n))$, where $v_1, \ldots, v_n$ are the free variables in $\delta$, and $\text{sko}_\delta$ is a skolem function for $\delta$. See [5].

*Initialization.* Put $\top$ at the root node of the tableau.

*Expansion.* Branches of a tableau for a set of formulas $\Phi$ are expanded according to the Smullyan decomposition recipe.

*Rigid Variables.* The rigid variables of a branch are defined in terms of the rigid variables of a node. The root node $R$ has $r(R) = \emptyset$. The nodes that are created by an $\alpha, \gamma$ or $\delta$ rule have the same rigid variables as their parent node. If an application of a $\beta$ rule creates daughter nodes $N_1, \ldots, N_n$, then $r(N_i)$ is given by:

$$\begin{aligned} r(N_i) &:= r(N) \cup \\ &\qquad (\text{var }(\beta_i) \cap \bigcup \{\text{var }(\beta_j) \mid 1 \leq j \leq n, i \neq j, \beta_i \text{ occurs on an open branch }\}). \end{aligned}$$

The set of rigid variables of a branch $\boldsymbol{B}$ is the rigid variable set at the end node of $\boldsymbol{B}$ if $\boldsymbol{B}$ is finite, or the set $\bigcup_{N \in \boldsymbol{B}} r(N)$ otherwise. This definition is extended to sets of branches in an obvious way.

*Constraint Generation.* Here comes the new element:

$$\frac{L,\ \overline{L'}}{\overline{\theta}},$$

where $L$ and $\overline{L'}$ are literals with opposite sign on the current branch, and $\theta$ is the restriction to the *rigid* variables of the current node of a most general substitution $\sigma$ with $L\sigma = L'\sigma$, where $\sigma$ has the property that it *does not rename any rigid variables of the branch.*

*Remark.* It is convenient to use substitions $\sigma$ in the constraint generation rule that *do not rename any rigid branch variables.* Suppose $Px$ is a positive literal on a branch, with $x$ rigid, and suppose that $y$ is not rigid on the branch. Then a match with $\neg Py$ can rename either $x$ or $y$. If $x$ is renamed, a constraint $x \not\approx y$ is generated. If $y$ is renamed, the constraint $\perp$ is generated, because we can take the constraint based on $\epsilon$, the restriction of the unifying substitution $\{y \mapsto x\}$ of $Px, \neg Py$ to the rigid variables on the branch. This closes the branch without further ado.

*Locked Variables.* A rigid variable $v$ is *locked* in a tableau if there is a tableau subtree $\boldsymbol{T}$ with

- $\gamma_1(v)$ at the root of $\boldsymbol{T}$,
- $\boldsymbol{T}$ has different open branches $B, B'$, with constraints $C$ on $B$, $C'$ on $B'$, and with $v \not\approx t \in C, v \not\approx t' \in C'$, $t$ and $t'$ different,
- the part of $B$ starting at $C$ does not contain $\gamma_1(w)$ for any $w$.

A locked variable is *freed* by a reapplication of an appropriate $\gamma$ rule, either to $B$ or to $B'$, in such a way that this does not generate an alphabetic variant of $\boldsymbol{T}$.

*Computation Rules; Fairness* A tableau computation rule $F$ is a function that for any tableau $\boldsymbol{T}$ for $\Phi$ computes the next rule to be applied to $\boldsymbol{T}$. This defines a partial order on the set of tableaux for $\Phi$, with the successor of $\boldsymbol{T}$ given by $F$. Then there is a (possibly infinite) sequence of tableaux for $\Phi$ starting from the initial tableau, and with supremum $\boldsymbol{T}_\infty$. A computation rule $F$ is fair if the following holds for all branches $B$ in $\boldsymbol{T}_\infty$:

1. All formulas of type $\alpha, \beta, \gamma, \delta$ occurring on $B$ or in $\Phi$ were used to expand $B$,
2. All locked variables occurring on $B$ were freed by a reapplication of an appropriate $\gamma$ rule (either to $B$ or to the other branch involved in the lock).

Note that it is thanks to the presence of constraints that we can restrict repetition of the $\gamma$ rule, and that infinite expansion of $\gamma$ formulas is not required in general. (If the $\gamma$ rule reapplication that frees a locked variable creates an alphabetic variant, the reapplication is spurious.)

*Variable Constraint Reduction.* To check a tableau consisting of $n$ branches for closure, we apply *constraint merge for closure*, to be defined in terms of joins of variable constraints. Computing joins of variable constraints is nothing but syntactic term unification under a different guise. It is clear that $\overline{\sigma} \vee \overline{\rho}$ reduces to $\top$ iff $\sigma$ and $\rho$ do not unify. Moreover, we have that $\overline{\sigma} \vee \overline{\rho}$ reduces to $\top$ iff $\overline{\sigma} \vee \overline{\rho}$ is universally satisfiable:

**Theorem 1.** *$\overline{\sigma} \vee \overline{\rho}$ reduces to $\top$ iff $[\![\overline{\sigma} \vee \overline{\rho}]\!] = [\![\top]\!]$.*

*Constraint Merge for Closure Check.* If $\boldsymbol{B}_1, \dots, \boldsymbol{B}_n$ are tableau branches, and for all $i \in [1..n]$, $\overline{\sigma_i}$ is a constraint on $\boldsymbol{B}_i$, then the *Constraint Merge for Closure Check* goes like this:

$$\frac{\overline{\sigma_1}, \quad \cdots \quad , \overline{\sigma_n}}{\text{closure by: } \sigma_1 \wedge \cdots \wedge \sigma_n} \overline{\sigma_1} \vee \cdots \vee \overline{\sigma_n} \not\equiv \top.$$

The idea of constraint merge for closure is that if the disjunction of $\overline{\sigma}$ and $\overline{\rho}$ is not universally satisfiable, then this means there exists an assignment that satisfies both $\sigma$ and $\rho$, which means in turn that $\sigma$ and $\rho$ can be unified, and that (a substitution corresponding to) $\sigma \wedge \rho$ closes both branches. Next, we try to extend the substitution $\sigma \wedge \rho$ to a closing substitution for other branches, until we get at a substitution that closes the whole tableau.

*Open and Closed Tableaux.* A tableau is *open* if one of the following two conditions holds, otherwise it is *closed*:

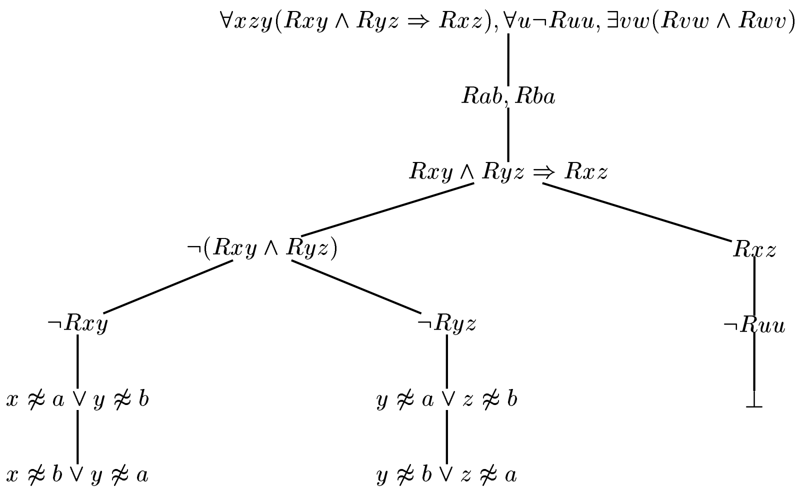
- some branch in the tableau carries no constraint,
- all branches in the tableau carry constraints, but all ways of picking constraints from different branches and merge them yield $\top$ (if one picks an $\overline{\sigma_i}$ on each $\boldsymbol{B}_i$, then always $\overline{\sigma_1} \vee \cdots \vee \overline{\sigma_n} \equiv \top$).

*Tableau Bundles; Herbrand Universes for Open Tableaux.* A pair of different branches in a tableau is *connected* if some variable distributes over the two branches (i.e., some rigid variable occurs on both branches). Since connectedness is symmetric, the reflexive transitive closure of this relation is an equivalence. A tableau *bundle* is an equivalence class of connected* branches. We will consider term models built from Herbrand universes of ground terms. The Herbrand universe of a bundle $\mathcal{B}$ in a tableau is the set of terms built from the skolem constants and functions that occur in $\mathcal{B}$, or, if no skolem constants are present, the set of terms built from the constant $c$ and the skolem functions that occur in $\mathcal{B}$. If $\mathcal{B}$ contains no skolem functions and $\mathcal{B}$ is finite, the Herbrand universe of $\mathcal{B}$ is finite; if $\mathcal{B}$ contains skolem functions it is infinite. The models over such a Herbrand universe are completely specified by a set of ground positive literals. We use $H_{\mathcal{B}}$ for the Herbrand universe of $\mathcal{B}$, and we call a variable map $\sigma$ with dom $(\sigma) = \text{vars}(\mathcal{B})$ and rng $(\sigma) \subseteq H_{\mathcal{B}}$ a *grounding* for $\mathcal{B}$ in $H_{\mathcal{B}}$, and a ground instance of a clause under a grounding for $\mathcal{B}$ in $H_{\mathcal{B}}$ an $H_{\mathcal{B}}$ instance. Note that a grounding need not be a substitution, as the set vars($\mathcal{B}$) may be infinite.

## 2 Examples

Uppercase characters are used for predicates, $x, y, z, u, \ldots$ for variables, $a, b, c, \ldots$ for individual constants (skolem constants), $f, g \ldots$ for skolem functions.

**Fig. 1.** Refutation Proof Example.


$\forall xzy(Rxy \wedge Ryz \Rightarrow Rxz), \forall u \neg Ruu, \exists vw(Rvw \wedge Rwv)$
$Rab, Rba$
$Rxy \wedge Ryz \Rightarrow Rxz$
$\neg(Rxy \wedge Ryz)$
$Rxz$
$\neg Rxy$
$\neg Ryz$
$\neg Ruu$
$x \not\approx a \vee y \not\approx b$
$y \not\approx a \vee z \not\approx b$
$\perp$
$x \not\approx b \vee y \not\approx a$
$y \not\approx b \vee z \not\approx a$

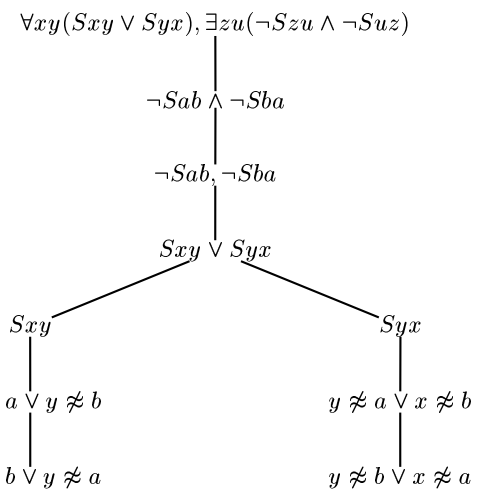
*Refutation Proof Example 1.* To prove that every transitive and irreflexive relation is asymmetric, we refute the following conjunction:

$$\forall xzy(Rxy \wedge Ryz \Rightarrow Rxz) \wedge \forall u \neg Ruu \wedge \exists vw(Rvw \wedge Rwv).$$

The tableau is given in Fig. 1. On the rightmost branch, we take care to use the constraint based on the substitution $\{u \mapsto x, u \mapsto y\}$ that does not rename the rigid variables $x, y$. This immediately leads to the constraint $\overline{\epsilon} = \bot$, for $u$ is not a rigid variable of the branch.

Constraint merge of $x \not\approx a \vee y \not\approx b \vee y \not\approx a \vee z \not\approx b$ yields $\top$, and so does constraint merge of $x \not\approx b \vee y \not\approx a \vee y \not\approx b \vee z \not\approx a$, so these combinations will never lead to a refutation of universal satisfiability. Constraint merge of $x \not\approx a \vee y \not\approx b \vee y \not\approx b \vee z \not\approx a$ yields the constraint $x \not\approx a \vee y \not\approx b \vee z \not\approx a$. Constraint merge of $x \not\approx b \vee y \not\approx a \vee y \not\approx a \vee z \not\approx b$ yields the constraint $x \not\approx b \vee y \not\approx a \vee z \not\approx b$. Both of these yield a non-trivial constraint when merged with $\bot$. Thus, the substitutions $\{x \mapsto a, y \mapsto b, z \mapsto a\}$, $\{x \mapsto b, y \mapsto a, z \mapsto b\}$ close the tableau.

**Fig. 2.** Another Refutation Proof Example.



*Refutation Proof Example 2.* A tableau for the sentence

$$\forall xy(Sxy \vee Syx) \wedge \exists zu(\neg Szu \wedge \neg Suz)$$

is given in Fig. 2. The constraint merges that lead to non-trivial constraints, and thus to closing substitutions, are: $x \not\approx a \vee y \not\approx b \vee y \not\approx b \vee x \not\approx a$ and $x \not\approx b \vee y \not\approx a \vee y \not\approx a \vee x \not\approx b$.

*Closure Through Reapplication of $\gamma$ Rule.* In Figure 3, a case is shown where reapplication of a $\gamma$ rule is crucial to achieve closure.

*Model Generation Example.* Consider a tableau for the sentence

$$\forall xy(Rxy \vee Sxy) \wedge \neg\exists zRza \wedge \neg\exists uSub,$$

**Fig. 3.** Closure Through Reapplication of a $\gamma$ Rule.

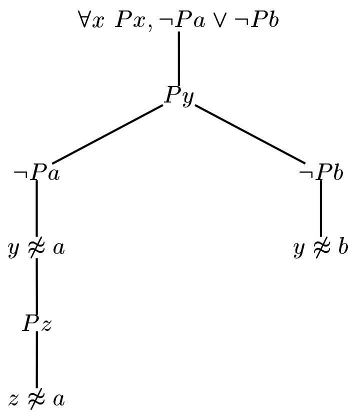
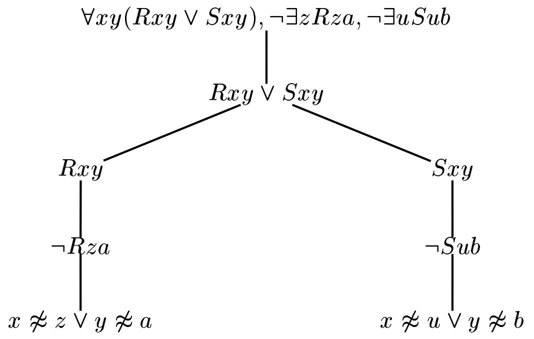


**Fig. 4.** Model Generation Example.



given in Fig. 4. This tableau does not close, for the disjunction of the two constraints is universally satisfiable, as it witnessed by the fact that $x \not\approx z \vee y \not\approx a \vee x \not\approx u \vee y \not\approx b$ reduces to $\top$.

Reapplication of the $\gamma$ rule to $\forall xy(Rxy \vee Sxy)$ does give a subtree that is an alphabetic variant of the tree we started out with, so such reapplications are spurious.

Other than that, there are no further rule applications, so we have an open tableau. A model for the sentence is not generated by a single branch in this case, as the two branches share constrained variables. The domain of a model generated from this tableau is the set of closed terms of the tableau, i.e., the set $\{a, b\}$. The set of groundings in this domain consists of $\theta_1 = \{x \mapsto a, y \mapsto a\}$, $\theta_2 = \{x \mapsto a, y \mapsto b\}$, $\theta_3 = \{x \mapsto b, y \mapsto a\}$, $\theta_4 = \{x \mapsto b, y \mapsto b\}$. $\theta_1$ satisfies only the right branch, so it generates the fact $Saa$. $\theta_2$ satisfies only the left branch, so it generates the fact $Rab$. $\theta_3$ satisfies only the right branch, so it generates the fact $Sba$. Finally, $\theta_4$ satisfies only the left branch, so it generates the fact $Rbb$. The model is given by the set of facts $\{Saa, Rab, Sba, Rbb\}$.

## 3 Soundness, Model Generation, Completeness

An assignment $\alpha$ in a model $\mathcal{M}$ meets a constraint $\overline{\sigma}$ if $\mathcal{M} \models_\alpha \overline{\sigma}$. Let $[\![\cdot]\!]_\alpha^\mathcal{M}$ give the term interpretation in the model with respect to $\alpha$. Then we have:

**Theorem 2.** *$\mathcal{M} \models_\alpha \overline{\sigma}$ iff there is a $v \in \mathrm{dom}\,(\sigma)$ with $\alpha(v) \neq [\![v\sigma]\!]_\alpha^\mathcal{M}$.*

An assignment $\alpha$ satisfies a branch $\boldsymbol{B}$ of a tableau $\boldsymbol{T}$ in a model $\mathcal{M}$ if $\alpha$ meets all constraints on $\boldsymbol{B}$, and $\mathcal{M} \models_\alpha L$ for all positive literals $L$ on $\boldsymbol{B}$. Notation: $\mathcal{M} \models_\alpha \boldsymbol{B}$. An assignment $\alpha$ satisfies a tableau $\boldsymbol{T}$ in a model $\mathcal{M}$ if $\alpha$ satisfies a branch of $\boldsymbol{T}$. Notation: $\mathcal{M} \models_\alpha \boldsymbol{T}$. A tableau $\boldsymbol{T}$ is (universally) satisfiable if for some model $\mathcal{M}$ it is the case that all assignments $\alpha$ for $\mathcal{M}$ satisfy $\boldsymbol{T}$ in $\mathcal{M}$. Notation: $\mathcal{M} \models \boldsymbol{T}$.

**Theorem 3 (Satisfiability).** *If $\Phi$ is a satisfiable first order sentence, then any tableau for $\Phi$ is satisfiable.*

*Proof.* Let $\boldsymbol{T}$ be a tableau for $\Phi$. Then either there is a finite tableau sequence $\boldsymbol{T}_1, \dots, \boldsymbol{T}_n = \boldsymbol{T}$, or there is an infinite sequence $\boldsymbol{T}_1, \dots$, with $\boldsymbol{T} = \bigcup_{i=1}^\infty \boldsymbol{T}_i$. In any case, $\boldsymbol{T}_1$ consists of a single node $\top$, and $\boldsymbol{T}_{i+1}$ is constructed from $\boldsymbol{T}_i$ by an application of one of the tableau expansion rules, or by an application of the constraint generation rule. To prove by induction on $n$ that $\boldsymbol{T}$ is satisfiable, we have to check that satisfiability is preserved by each of these steps, and by the process of taking limits. The only non-standard case is that of constraint generation.

Take some $\mathcal{M}$ with $\mathcal{M} \models \Phi$. Assume that $\mathcal{M} \models \boldsymbol{T}_i$, and $\boldsymbol{T}_{i+1}$ is the result of applying constraint generation to $\boldsymbol{T}_i$. Assume the branch to which *Constraint Generation* is applied is $\boldsymbol{B}$, the branch literals used in the rule are $L, \overline{L'}$, the unifying substitution is $\sigma$, and the restriction of $\sigma$ to the rigid branch variables is $\theta$.

Consider an assignment $\alpha$ that satisfies $\boldsymbol{T}_i$ in $\mathcal{M}$. In case $\alpha$ satisfies a branch different from $\boldsymbol{B}$ then the new constraint $\overline{\theta}$ will not affect this, and $\alpha$ will satisfy $\boldsymbol{T}_{i+1}$ in $\mathcal{M}$. Suppose, therefore, that $\alpha$ satisfies only $\boldsymbol{B}$. We have to show that $\alpha$ satisfies $\overline{\theta}$. Since $L, \overline{L'}$ are on branch $\boldsymbol{B}$, we know that $\mathcal{M} \models_\alpha L$ and $\mathcal{M} \not\models_\alpha L'$.

Let assignment $\alpha'$ be given by $\alpha'(v) = [\![v\sigma]\!]_\alpha^\mathcal{M}$. We distinguish two cases. (1) If $\mathcal{M} \models_\alpha L\sigma$, then $\mathcal{M} \models_{\alpha'} L$, and $\mathcal{M} \models_{\alpha'} L'$, so $\alpha'$ does not satisfy $\boldsymbol{B}$. (2) If $\mathcal{M} \not\models_\alpha L\sigma$, then $\mathcal{M} \not\models_{\alpha'} L$, and $\mathcal{M} \not\models_{\alpha'} L'$, so again $\alpha'$ does not satisfy $\boldsymbol{B}$. In both cases, by the universal satisfiability of $\boldsymbol{T}_i$, there has to be a $\boldsymbol{B}'$ with $\mathcal{M} \models_{\alpha'} \boldsymbol{B}'$. Since $\boldsymbol{B}$ is by assumption the only branch with $\mathcal{M} \models_\alpha \boldsymbol{B}$, $\mathcal{M} \not\models_\alpha \boldsymbol{B}'$. So there has to be a variable $v$ that is both on $\boldsymbol{B}$ and $\boldsymbol{B}'$ with the property that $\alpha(v) \neq \alpha'(v)$. But this means that $v \in \mathrm{dom}\,(\sigma)$ and $v$ is rigid in $\boldsymbol{T}_i$. It follows that $v \in \mathrm{dom}\,(\theta)$, and that $\alpha$ does meet $\overline{\theta}$, i.e., $\mathcal{M} \models_\alpha \overline{\theta}$. □

**Theorem 4 (Merge).** *If a tableau $\boldsymbol{T}$ closes by constraint merge, then $\boldsymbol{T}$ is not universally satisfiable.*

*Proof.* If $\boldsymbol{T}$ closes by constraint merge then there is a way to pick constraints $\overline{\sigma_1}$, $\dots$, $\overline{\sigma_n}$, one on each tableau branch, such that $\overline{\sigma_1} \vee \cdots \vee \overline{\sigma_n}$ does not reduce to $\top$. By Theorem 1, this means that $\overline{\sigma_1} \vee \cdots \vee \overline{\sigma_n}$ is not universally satisfiable in the initial term algebra. It follows that $\overline{\sigma_1} \vee \cdots \vee \overline{\sigma_n}$ is not universally satisfiable in any model. □

**Theorem 5 (Soundness).** *If there is a tableau refutation for a sentence $\Phi$, then $\Phi$ is unsatisfiable.*

*Proof.* Immediate from the Satisfiability Theorem and the Merge Theorem. □

A variable map $\theta$ meets a constraint $\overline{\sigma}$ if $\theta \in [\![\overline{\sigma}]\!]$. A variable map $\theta$ is *compatible* with a branch $\boldsymbol{B}$ if $\theta$ meets all constraints $\overline{\sigma}$ on $\boldsymbol{B}$. A variable map $\theta$ is *compatible* with a bundle $\mathcal{B}$ if $\theta$ is compatible with at least one branch $\boldsymbol{B}$ of $\mathcal{B}$.

**Theorem 6 (Compatibility).** *If a tableau bundle $\mathcal{B}$ is open, then every variable map $\theta$ is compatible with $\mathcal{B}$.*

*Proof.* Assume $\mathcal{B}$ consists of branches $\boldsymbol{B}_i\ _{(i \geq 0)}$. We have to show that every variable map is compatible with at least one $\boldsymbol{B}_i$. Suppose $\theta$ is a variable map that is not compatible with any $\boldsymbol{B} \in \mathcal{B}$. Then for each of the $\boldsymbol{B}_i$ there is a constraint $\overline{\sigma_i}$ on $\boldsymbol{B}_i$ such that $\theta \in [\![\sigma_i]\!]$. Since variable maps modulo renaming form a complete lattice under $\preceq$, it follows that $\theta \preceq \bigwedge_{(i \geq 0)} \sigma_i$. Now any constraint is at finite distance from the root of the tableau, so there has to be a *finite* set of constraints $\overline{\sigma_1}, \dots, \overline{\sigma_n}$ with $\forall i \geq 0\ \exists j \leq n$ such that $\overline{\sigma_j}$ occurs on $\boldsymbol{B}_i$. But then $\overline{\sigma_1} \vee \cdots \vee \overline{\sigma_n}$ does not reduce to $\top$, and contradiction with the assumption that $\mathcal{B}$ is open. □

**Theorem 7 (Model Generation).** *Every open tableau is satisfiable.*

*Proof.* Since in a Herbrand universe groundings play the role of assignments, all we have to do to satisfy a tableau $\boldsymbol{T}$ in a Herbrand model is look at all the ground instances of the tableau. To generate a model from an open tableau, proceed as follows. Pick an open bundle $\mathcal{B}$, and consider groundings for $\mathcal{B}$ in $H_{\mathcal{B}}$.

- If there is an unconstrained $\boldsymbol{B} \in \mathcal{B}$, the set of all $H_{\mathcal{B}}$ instances of the positive literals along $\boldsymbol{B}$ constitutes a model for the tableau. It is clear that the model satisfies the tableau.
- If all branches in $\mathcal{B}$ are constrained, then generate $H_{\mathcal{B}}$ instances from groundings for $\mathcal{B}$ in $H_{\mathcal{B}}$, as follows. For every grounding $\theta$ for $\mathcal{B}$ in $H_{\mathcal{B}}$, we can pick, according to the Compatibility Theorem, a branch $\boldsymbol{B}$ in $\mathcal{B}$ that is compatible with $\theta$. Collect the ground instances of the positive literals of $\boldsymbol{B}$. The union, for all groundings $\theta$, of the sets of ground positive literals collected from branches compatible with $\theta$, constitutes a model for the tableau. Again, it is clear that the model satisfies the tableau. □

**Theorem 8 (Completeness).** *If a set of formulas $\Phi$ is unsatisfiable, then there exists a tableau refutation for $\Phi$.*

*Proof.* Immediate from the Model Generation Theorem. □

## 4 Fair Computation

A tableau calculus is *non-destructive* if all tableaux that can be constructed with the help of its rules from a given tableau $\boldsymbol{T}$ contain $\boldsymbol{T}$ as an initial subtree [5]. The usual versions of free variable tableaux are all destructive. Clearly, the present calculus is non-destructive. A tableau calculus is *proof confluent* if every tableau for an unsatisfiable set of formulas $\Phi$ can be extended to a closed tableau [5]. Again, it is clear that the present calculus is proof-confluent. Because of its non-destructiveness and proof-confluence, fair computation with constrained tableaux is easy. The usual fairness conditions for tableau expansion are replaced by more sophisticated ones in terms of constraints (freeing of locked variables, cutting off rule applications that lead to alphabetic variants). As the calculus is non-destructive, no backtracking is ever needed in the merge check for closure.

## 5 Related Work

The standard reference for free variable reasoning in first order tableaux is [2], but the idea to use free variables in theorem proving as placeholders to delay instantiation is already present in Prawitz [7]. With the introduction of free variables in tableaux, easy model generation from open tableaux got lost. Working with variable constraints in the manner explained above restores this delightful property of tableau reasoning.

The research for this paper was sparked off by a suggestion from [3] to do tableau proof search by merging closing substitutions for tableau branches into a closing substitution for the whole tableau (see also [4]). From the model generation perspective, this is turned around: the negations of the substitutions that close a branch can be viewed as constraints on branch satisfiability. The perspective of the present paper is worked out more fully in [9] in the context of hyper tableaux [1]. We are experimenting with implementations of constrained (hyper) tableau reasoning in Haskell [6], with merge checks for closure performed on tableau branches represented as lazy lists.

## References

1. P. Baumgartner, P. Fröhlich, and I. Niemelä. Hyper tableaux. In *Proceedings JELIA 96*, Lecture Notes in Artificial Intelligence. Springer, 1996.
2. M. Fitting. *First-order Logic and Automated Theorem Proving; Second Edition.* Springer Verlag, Berlin, 1996.
3. Martin Giese. Proof search without backtracking using instance streams, position paper. In *Proc. Int. Workshop on First-Order Theorem Proving, St. Andrews, Scotland*, 2000. Available online at `http://i12www.ira.uka.de/~key/doc/2000/giese00.ps.gz`.
4. Martin Giese. Incremental closure of free variable tableaux. In *IJCAR 2001 Proceedings*, 2001.
5. R. Hähnle. Tableaux and related methods. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning.* Elsevier Science Publishers, to appear, 2001.
6. S. Peyton Jones, J. Hughes, et al. Report on the programming language Haskell 98. Available from the Haskell homepage: `http://www.haskell.org`, 1999.
7. D. Prawitz. An improved proof procedure. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning. Classical Papers in Computational Logic*, pages 162–201. Springer, 1983. Originally appeared in Theoria in 1960.
8. R. Smullyan. *First-order logic.* Springer, Berlin, 1968.
9. J. van Eijck. Constrained hyper tableaux. In *Proceedings of CSL'01 (to appear)*, 2001. Electronically available ast `http://www.cwi.nl/~jve/papers/01/Eijck01b.ps.gz`.