Constrained Hyper Tableaux

Jan van Eijck

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

Abstract. Hyper tableau reasoning is a version of clausal form tableau reasoning where all negative literals in a clause are resolved away in a single inference step. Constrained hyper tableaux are a generalization of hyper tableaux, where branch closing substitutions, from the point of view of model generation, give rise to constraints on satisfying assignments for the branch. These variable constraints eliminate the need for the awkward 'purifying substitutions' of hyper tableaux. The paper presents a non-destructive and proof confluent calculus for constrained hyper tableaux, together with a soundness and completeness proof, with completeness based on a new way to generate models from open tableaux. It is pointed out that the variable constraint approach applies to free variable tableau reasoning in general.

1 Introduction

Hyper tableau reasoning was introduced in [2]; like (positive) hyper resolution [9] it resolves away all negative literals of a clause in a single inference step, but it combines this with the notion of a tableau style search for counterexamples. Hyper tableau reasoning, in the improved version proposed in [1], allows local universally quantified variables. The key element in hyper tableau reasoning, the use of purifying substitutions to get rid of variable distribution over different head literals (or, in the improved version, the generation of proper clause instantiations by means of a *Link* rule) is replaced in constrained hyper tableau reasoning by the generation of constraints on the interpretation of the variables that get distributed. Constrained hyper tableaux solve the problem of model generation from open tableaux with free variables in a general way.

2 Basic Definitions

Language. Let Σ be a first order signature. A $\mathcal{L}_\Sigma$ literal is an $\mathcal{L}_\Sigma$ atom or its negation, and an $\mathcal{L}_\Sigma$ clause is a multiset of $\mathcal{L}_\Sigma$ literals, written as $\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n$ $(m, n \geq 0)$. If $m, n > 0$ the clause is *mixed*; if $m = 0, n > 0$ the clause is *positive*; if $m > 0, n = 0$ the clause is *negative*, and if $m = n = 0$ the clause is *empty*. A mixed clause $\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n$ may be written as $A_1 \wedge \cdots \wedge A_m \Rightarrow B_1 \vee \cdots \vee B_n$, and a negative clause as $\neg(A_1 \wedge \cdots \wedge A_m)$. The empty clause is written as $\bot$. We write $\top$ for the formula that is always true.

Substitutions. A *substitution* σ is a function $V \to T_\Sigma$ that makes only a finite number of changes, i.e., σ has the property that dom $(\sigma) = \{v \in V \mid \sigma(v) \neq v\}$ is finite. We use ϵ for the substitution with domain $\emptyset$ (the identity substitution). We represent a substitution σ in the standard way, as a list $\{v_1 \mapsto \sigma(v_1), \ldots, v_n \mapsto \sigma(v_n)\}$, where $\{v_1, \ldots, v_n\}$ is dom (σ). Write substitution application in post-fix notation, and write $\sigma\theta$ for 'θ after σ'.

If σ, θ are substitutions, then $\sigma \preceq \theta$ if σ is less general than θ, i.e., if there is a ρ with $\sigma = \theta\rho$. The relation $\preceq$ is a pre-order (transitive and reflexive), and its poset reflection is a partial order. For this, put $\sigma \sim \theta$ if $\sigma \preceq \theta$ and $\theta \preceq \sigma$, and consider substitutions modulo renaming, i.e., put $|\sigma| = \{\theta \mid \sigma \sim \theta\}$, and put $|\sigma| \sqsubseteq |\theta|$ if $\sigma \preceq \theta$. A renaming is a substitution that is a bijection on the set of variables. For convenience we continue to write σ for $|\sigma|$.

Extend the set of substitutions (modulo renaming) with the improper substitution $\perp\!\!\!\perp$, the substitution with the property that $\perp\!\!\!\perp \sqsubseteq \sigma$ for every substitution σ. Now for every pair of substitutions σ and θ, $\sigma \sqcap \theta$, the greatest common instance of σ and θ, and $\sigma \sqcup \theta$, the least common generalization of σ and θ, exist. If $\sigma \sqcap \theta = \perp\!\!\!\perp$ we say that σ and θ do not unify. We get that ϵ, the substitution that is more general than any, is the top of the lattice given by $\sqsubseteq$, and $\perp\!\!\!\perp$ its bottom. The grounding substitutions are the least general proper substitutions; In the lattice of substitutions, they are just above $\perp\!\!\!\perp$. Note that this hinges on the fact that substitutions have finite domains. If $\sigma \sqsubseteq \rho$, and $\sigma \neq \perp\!\!\!\perp$, we call σ an instance of ρ. A clause ϕ is a proper instance of a clause ψ if for some substitution σ that is not a renaming it is the case that $\phi = \psi\sigma$.

A variable map is a function in $V \to T_\Sigma$ (i.e., we drop the finite domain restriction of substitutions). Variable maps modulo renaming form a *complete lattice* under the 'less general than' ordering. A grounding is a variable map that maps every variable to a closed term.

Substitutions as Formulas; Variable Constraints. Associate with a substitution

$$\sigma = \{v_1 \mapsto \sigma(v_1), \ldots, v_n \mapsto \sigma(v_n)\}$$

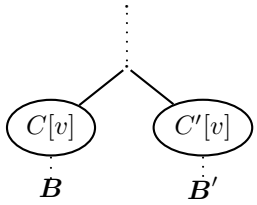
the formula $v_1 \approx \sigma(v_1) \wedge \cdots \wedge v_n \approx \sigma(v_n)$. We can then say what it means that assignment α satisfies substitution σ in model $\mathcal{M}$ in the usual way. Notation $\mathcal{M} \models_\alpha \sigma$. A *variable constraint* is the negation of a substitution as formula, i.e., a variable constraint is a multiset of inequalities $v \not\approx t$, with $t \in T_\Sigma$, written as $v_1 \not\approx t_1 \vee \cdots \vee v_n \not\approx t_n$. From a substitution σ we derive a variable constraint $\overline{\sigma}$ by complementation, as follows:

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

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 $\overline{\epsilon} = \bot$.

Tableaux, Branches. A hyper tableau over Σ is a finitely branching tree with nodes labeled by positive $\mathcal{L}_\Sigma$ literals, or by variable constraints. A branch in a tableau $\boldsymbol{T}$ is a maximal path in $\boldsymbol{T}$. We occasionally identify a branch $\boldsymbol{B}$ with

the set of its atomic facts and constraints. The variables of a tableau branch are the variables that occur in a literal or a constraint along the branch. A variable v distributes over branches $\boldsymbol{B}, \boldsymbol{B}'$ if v occurs in constraints or literals on both sides of a split point, as follows:



The rigid variables of a branch $\boldsymbol{B}$ are the variables of $\boldsymbol{B}$ that are distributed over $\boldsymbol{B}$ and some other branch. The tableau construction rules will ensure that every rigid variable in a tableau has a unique split point (highest point where it gets distributed).

A hyper tableau for a set Φ of $\mathcal{L}_\Sigma$ formulas in clause form is a finite or infinite tree grown according to the following instructions.

Initialize. Put $\top$ at the root node of the tableau.

Expand. Branches of a hyper tableau for clause set Φ are expanded by the only inference rule of Constrained Hyper Tableau (CHT) reasoning, the rule *Expand*, in the following manner.

$$\frac{C_1, \ldots, C_m, \quad \neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n}{B_1\sigma \quad | \quad \cdots \quad | \quad B_n\sigma \quad | \quad \overline{\theta}},$$

where

- $\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n$ is fresh copy of a clause in Φ (fresh with respect to the tableau),
- the C_i are positive literals from the current branch,
- σ is a most general substitution such that $A_i\sigma = C_i\sigma \quad (1 \leq i \leq m)$, and, moreover, σ *does not rename any rigid branch variables*,
- θ is the restriction of σ to the *rigid* variables of the branch.

An application of *Expand* to a branch expands the branch with an instance of a literal from the list $B_1, \ldots, B_n$, or with a variable constraint.

Remark. It is convenient to use mgu's σ in *Expand* that do not rename any rigid branch variables. Suppose Px is a positive literal on a branch, with x rigid. Then a match with the rule $Py \Rightarrow Qy$ can rename either x or y. If x is renamed, the application of *Expand* branches, and two leafs are created, one with constraint $x \not\approx y$, the other with literal Qy. If x is not renamed, only a single leaf Qx is created, for in this case the constraint leaf extension carries constraint $\overline{\epsilon}$, and can be suppressed.

Here is an example application of *Expand*. Here and below, 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. In the example, it is assumed that x is rigid and y is not:

$$\frac{Pxy, \quad Qb, \quad \neg Paz \vee \neg Qz \vee Raz}{Rab \quad | \quad x \not\approx a}.$$

Note that in the case of a positive clause, no branch literals are involved, and the substitution that is produced is ϵ, with corresponding constraint $\overline{\epsilon}$, i.e., $\bot$. In this case the rule boils down to:

$$\frac{B_1 \vee \cdots \vee B_n}{B_1 \mid \quad \cdots \quad \mid B_n}.$$

If there are no positive clauses $B_1 \vee \cdots \vee B_n$ in the clause set Φ, the set Φ cannot be refuted since in this case we can always build a model for Φ from just negative facts.

In case *Expand* is applied with a negative clause, the rule boils down to the following:

$$\frac{C_1, \ldots, C_m, \neg(A_1 \wedge \cdots \wedge A_m)}{\overline{\theta}},$$

where the C_i are as before, there is a most general σ such that $A_i\sigma = C_i\sigma \ \ (1 \leq i \leq m)$ and no rigid variables get renamed, and θ is the restriction of σ to the rigid variables of the branch.

History Conditions on Expand. To avoid superfluous applications of *Expand*, a *history list* is kept of all clause instances that were applied to a branch. For this we need a preliminary definition. We say that a literal B reaches a k-fold in clause set Φ if either there is a clause in Φ in which the predicate of B has at least k negative occurrences, or there is a clause $\ldots B \ldots \Rightarrow \ldots C \ldots$ in Φ, and C reaches a k-fold in Φ. E.g., if $Qa \Rightarrow Pa, Px \wedge Py \Rightarrow Rxy$ in Φ, then Qx reaches a 2-fold in Φ. If Qx is also in Φ, we should generate two copies Qx', Qx'', which in turn will yield two copies of Pa, so that Raa can be derived.

If a clause is applied with substitution σ, the conditions on the application, in a tableau for clause set Φ, are:

1. $\neg A_1\sigma \vee \cdots \vee \neg A_m\sigma \vee B_1\sigma \vee \cdots \vee B_n\sigma$ is not a proper instance of any of the instances of $\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n$ that were applied to the branch before;
2. if $\neg A_1\sigma \vee \cdots \vee \neg A_m\sigma \vee B_1\sigma \vee \cdots \vee B_n\sigma$ is the k-th variant of any of the instances of $\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n$ that were applied to the branch before, then at least one of the B_i must reach a k-fold in Φ.

If these two conditions are fulfilled, we say that the instance of the clause is *fresh to the branch*. All clause instances used on a branch are kept in a branch history list. The *history conditions* on *Expand* are fair, for application of proper instances of previously applied clause instances to a branch is spurious, and generation of

alphabetic variants only makes sense if they (eventually) lead to the generation of alphabetic variants that can be matched simultaneously against a single clause in Φ.

Constraint Merge for Closure. To check a tableau consisting of n branches for closure, apply the following *constraint merge for closure*. It is assumed that the $\overline{\sigma_i}$ are constraints on the different branches.

$$\frac{\overline{\sigma_1}, \quad \cdots \quad , \overline{\sigma_n}}{\text{closure by: } \sigma_1 \sqcap \cdots \sqcap \sigma_n} \sigma_1 \sqcap \cdots \sqcap \sigma_n \neq \perp\!\!\!\perp.$$

The idea of the constraint merge for closure is that if σ, θ each close a branch and can be unified, then $\sigma \sqcap \theta$ closes both branches, and so on, until the whole tableau is closed.

Open and Closed Tableaux. A hyper 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 there is no way to pick constraints from individual branches and merge their corresponding substitutions into a single substitution (in the sense of: pick a finite initial stage $\boldsymbol{T}$, and pick $\overline{\sigma_i}$ on each B_i of $\boldsymbol{T}$ such that $\sigma_1 \sqcap \cdots \sqcap \sigma_n \neq \perp\!\!\!\perp$).

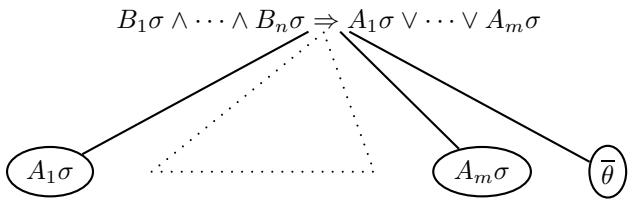
Fair Tableaux. A hyper tableau $\boldsymbol{T}$ for clause set Φ is *fair* if on every open branch B of $\boldsymbol{T}$, *Expand* is applied to each clause in Φ as many times as is compatible with the history conditions on the branch.

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. Since connectedness is symmetric, the reflexive transitive closure of this relation (connected*) is an equivalence. A tableau *bundle* is an equivalence class of connected* branches.

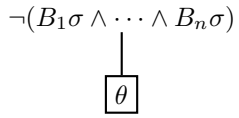
We will consider term models built from Herbrand universes of ground terms. The Herbrand universe of a bundle $\mathcal{B}$ in a tableau is the set of terms built from the skolem constants and functions that occur in $\mathcal{B}$, or, if no skolem constants are present, the set of terms built from the 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 σ with $\mathrm{dom}\,(\sigma) = \mathrm{vars}(\mathcal{B})$ and $\mathrm{rng}\,(\sigma) \subseteq H_{\mathcal{B}}$ 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 $\mathrm{vars}(\mathcal{B})$ may be infinite.

3 Refutation Proof Examples

Let us agree on some conventions for tableau representation. To represent an application of extension in the tableau, we just have to write the rule instance $B_1\sigma \wedge \cdots \wedge B_n\sigma \Rightarrow A_1\sigma \vee \cdots \vee A_m\sigma$, and the branch extensions with the list of daughters $A_1\sigma$, ..., $A_m\sigma$, $\overline{\theta}$, as follows:



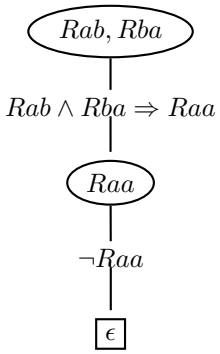
In case the constraint $\overline{\theta}$ that is generated is $\bot$, we suppress that leaf, unless it is the single leaf that closes the branch. If a constraint gives rise to a substitution that closes the whole tableau, then the substitution will be put in a box, like this (note that $\boxed{\theta}$ should be read as $\overline{\theta}$):



Reasoning about Relations. To prove that every transitive and irreflexive relation is asymmetric, we refute the clause form of its negation:

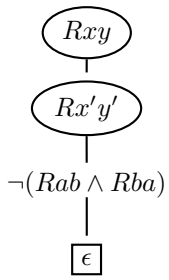
$$\{Rxy \wedge Ryz \Rightarrow Rxz, \neg Ruu, Rab, Rba\},$$

where the Rab, Rba provide the witnesses of non-asymmetry.

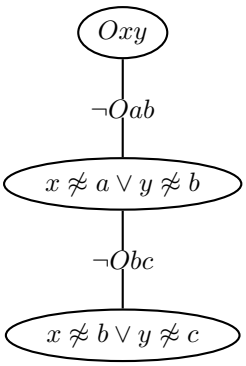


To apply the negative clause $\neg Ruu$, we use the substitution $\{u \mapsto a\}$. The restriction of that substitution to the rigid tableau variables is ϵ, so ϵ is the closing substitution of the tableau.

Closure by Renaming. To refute the clause set $\{Rxy, \neg Rab \vee \neg Rba\}$, two applications of *Expand* to the clause Rxy are needed. The second application uses fresh variables. Since none of the variables is distributed in the tableau, the closing substitution is ϵ.

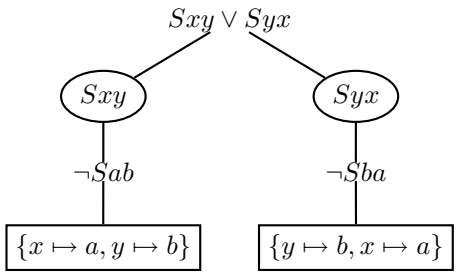


Generation of Multiple Closing Substitutions. If we try to refute the clause set $\{Oxy, \neg Oab, \neg Obc\}$, we can close the tableau in two ways, but since no variable is distributed, the closing substitution is ϵ in both cases. If the clauses are used to expand a tableau branch in which x and y are distributed, the following two constraints are generated on the branch.



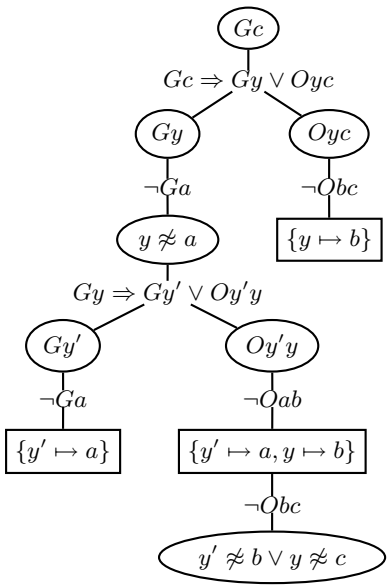
The order in which the constraints are generated does not matter. Both substitutions $\{x \mapsto a, y \mapsto b\}, \{x \mapsto b, y \mapsto c\}$ are candidates for use in the merge check for closure of the whole tree. If the branch is part of an open tableau, then both constraints act as constraints on branch satisfaction.

Closure by Merge. A hyper tableau for the clause set $\{Sxy \vee Syx, \neg Sab, \neg Sba\}$ has x, y rigid, so these variables occur in the constraints that are generated.



The substitutions unify (are, in fact, identical), so the tableau closes.

Fig. 1. Tableau for AI Puzzle.



An AI Puzzle. If a is green, a is on top of b, b is on top of c, and c is not green, then there is a green object on top of an object that is not green. For a hyper tableau refutation proof, refute the clause form of the negation of this: $\{Ga, \neg Gc, Oab, Obc, Gx \wedge Oxy \Rightarrow Gy\}$. To make for a more interesting example, we swap positive and negative literals, as follows.

$$\{\neg Ga, Gc, \neg Oab, \neg Obc, Gx \Rightarrow Gy \vee Oyx\}.$$

This is not in the Horn fragment of FOL, so beyond Prolog (except through Horn renaming; for the present example, a Horn renaming is a swap of O and $\neg O$, and of G and $\neg G$). In the tableau for this example, in Fig. 1, note that when the rule $Gx \Rightarrow Gy \vee Oyx$ is used for the second time, its variables are first renamed. The variable y gets distributed at the first tableau split, the variable y' at the second split. The tableau of Figure 1 closes, for the substitution $\{y' \mapsto a, y \mapsto b\}$ closes every branch. This closing substitution is found by an attempt to merge closing substitutions of the individual branches. The branch constraints for which this works are boxed. Other possibilities fail. In particular, the substitution $\{y' \mapsto b, y \mapsto c\}$ closes the middle branch all right, but it clashes with both substitutions that close the left hand branch, either on the y or on

the y' value, and with the substitution that closes the rightmost branch on the y value.

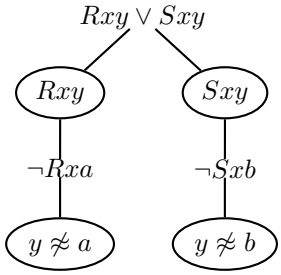
4 Model Generation Examples

No Positive Clause Present. Every clause set that contains no positive clauses is satisfiable in a model with a single object satisfying no atomic predicates. Example: a model for transitivity and irreflexivity.

$$\{Rxy \wedge Ryz \Rightarrow Rxy, \neg Ruu\}.$$

No hyper tableau rule is applicable to such a clause set, so we get no further than the top node $\top$. Since there are no skolem constants, we generate the Herbrand universe from c. This gives a single object with no properties.

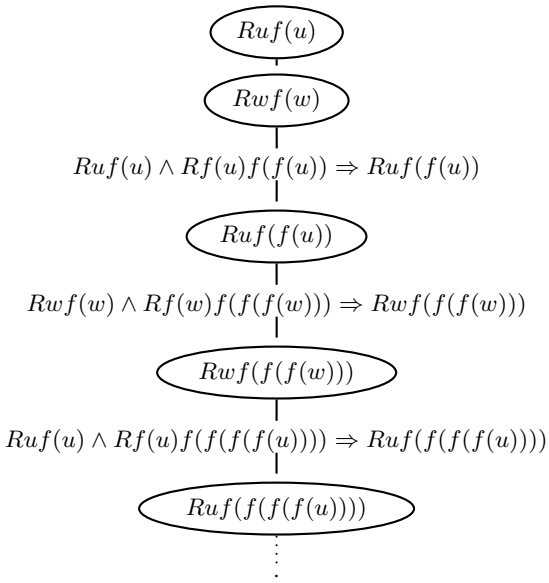
Disjunctively Satisfiable Constraints. Here is a tableau for the clause set $\{Rxy \vee Sxy, \neg Rza, \neg Sub\}$:



This tableau does not close, for the two substitutions disagree on the value for y. Or, put differently, the two constraints can be satisfied disjunctively. There are no further rule applications, so we have an open tableau. A model for the clause set is not generated by a single branch in this case, as the two branches share a constrained variable. 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 $\theta_1 = \{x \mapsto a, y \mapsto a\}$, $\theta_2 = \{x \mapsto a, y \mapsto b\}$, $\theta_3 = \{x \mapsto b, y \mapsto a\}$, $\theta_4 = \{x \mapsto b, y \mapsto b\}$. θ_1 satisfies only the right branch, so it generates the fact Saa. θ_2 satisfies only the left branch, so it generates the fact Rab. θ_3 satisfies only the right branch, so it generates the fact Sba. Finally, θ_4 satisfies only the left branch, so it generates the fact Rbb. The model is given by the set of facts $\{Saa, Rab, Sba, Rbb\}$.

Infinitary Tableau Development. There are relations that are transitive and serial. The attempt to refute this combination of properties should lead to an open hyper tableau. In fact, the model that is generated for the clause set $\{Rxy \wedge Ryz \Rightarrow Rxz, Ruf(u)\}$ is infinite. The step from $Ruf(u)$ to $Rwf(w)$, in Fig. 2, is an application of *Expand* that generates an alphabetic variant. This agrees with the history condition, since there is a clause in the clause set with

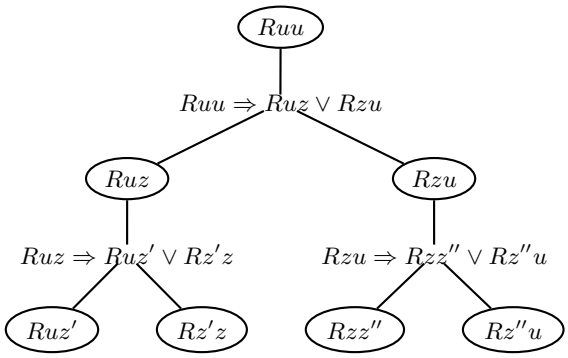
Fig. 2. Infinitary Tableau Development



two negative R occurrences. The tableau will not close, and tableau development will not be stopped by the check on instantiations, for new instances of the rule $Rxy \wedge Ryz \Rightarrow Rxz$ will keep turning up. The corresponding model is isomorphic to $\mathbb{N}, <$. Although finite models for the clause set exist (a single reflexive point also constitutes a model for this example) the calculus needs to be modified to generate them. For finite model generation, we need a slightly more sophisticated treatment of literals that introduce new skolem terms to a branch. This is beyond the scope of the present paper.

Open Tableau; No Further Rules Applicable. In the tableau for clause set $\{Rxy \Rightarrow Rxz \vee Rzy, Ruu\}$, given in Fig. 3, no further branch extensions are generated, as on all branches the next instance of $Rxy \Rightarrow Rxz \vee Rzy$ is a variant of an instance that has already been used on the branch. Generation of variants of the R predicate on a branch is spurious, because no clause in the clause set has more that a single negative occurrence of the R predicate. Note that variables are renamed in the second and the third application of the rule $Rxy \Rightarrow Rxz \vee Rzy$. Since there are no skolem constants, we generate the Herbrand model from a fresh c. This gives a model consisting of a single reflexive point, from any of the branches.

Fig. 3. No Further Applicable Rules



5 Soundness, Model Generation, Completeness

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

Theorem 1. *$\mathcal{M} \models_\alpha \overline{\sigma}$ iff there is a $v \in \mathrm{dom}\,(\sigma)$ with $\alpha(v) \neq [\![v\sigma]\!]^{\mathcal{M}}_\alpha$.*

The idea of the constraints is to *forbid* certain variable interpretations!

An assignment α satisfies a branch $\boldsymbol{B}$ of a tableau $\boldsymbol{T}$ in a model $\mathcal{M}$ if α 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 α satisfies a tableau $\boldsymbol{T}$ in a model $\mathcal{M}$ if α satisfies a branch of $\boldsymbol{T}$. Notation: $\mathcal{M} \models_\alpha \boldsymbol{T}$. A tableau $\boldsymbol{T}$ is satisfiable if for some model $\mathcal{M}$ it is the case that all assignments α for $\mathcal{M}$ satisfy $\boldsymbol{T}$ in $\mathcal{M}$. Notation: $\mathcal{M} \models \boldsymbol{T}$.

Theorem 2 (Satisfiability). *If Φ is a satisfiable set of clauses, then any tableau for Φ is satisfiable.*

Proof. Let $\boldsymbol{T}$ be a tableau for Φ. Since a tableau for Φ is a any tree grown from the seed $\top$ with the rule *Expand*, either there is a finite tableau sequence $\boldsymbol{T}_1, \ldots, \boldsymbol{T}_n = \boldsymbol{T}$, or there is an infinite sequence $\boldsymbol{T}_1, \ldots$, 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 *Expand*. To prove by induction on n that a finite $\boldsymbol{T}$ is satisfiable, we have to check that satisfiability is preserved by each of these steps. 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 *Expand* to $\boldsymbol{T}_i$. Assume the branch to which *Expand* is applied is $\boldsymbol{B}$, the clause is $B_1 \wedge \cdots \wedge B_k \Rightarrow A_1 \vee \cdots \vee A_m$, the branch literals used in the rule are $C_1, \ldots, C_k$, the matching substitution is σ, and the restriction of σ to the rigid branch variables is θ.

Consider an assignment α that satisfies $\boldsymbol{T}_i$ in $\mathcal{M}$. In case α satisfies a branch different from $\boldsymbol{B}$ then the application of *Expand* will not affect this, and α will satisfy $\boldsymbol{T}_{i+1}$ in $\mathcal{M}$. Suppose, therefore, that α satisfies only $\boldsymbol{B}$. We have to show

that α satisfies at least one of the branch extensions, with $A_1\sigma$, with $\ldots$, with $A_m\sigma$, or with $\overline{\theta}$. From $\mathcal{M} \models \Phi$ we get that

$$\mathcal{M} \models B_1 \wedge \cdots \wedge B_k \Rightarrow A_1 \vee \cdots \vee A_m,$$

and therefore, since $B_i\sigma = C_i\sigma$,

$$\mathcal{M} \models C_1\sigma \wedge \cdots \wedge C_k\sigma \Rightarrow A_1\sigma \vee \cdots \vee A_m\sigma,$$

so in particular

$$\mathcal{M} \models_\alpha C_1\sigma \wedge \cdots \wedge C_k\sigma \Rightarrow A_1\sigma \vee \cdots \vee A_m\sigma.$$

In case $\mathcal{M} \models_\alpha C_1\sigma \wedge \cdots \wedge C_k\sigma$, it follows from the above that $\mathcal{M} \models_\alpha A_1\sigma \vee \cdots \vee A_m\sigma$, and we are done. In case $\mathcal{M} \not\models_\alpha C_1\sigma \wedge \cdots \wedge C_k\sigma$ we have to show that $\mathcal{M} \models_\alpha \overline{\theta}$. In this case, there is an i with $\mathcal{M} \models_\alpha C_i$ and $\mathcal{M} \not\models_\alpha C_i\sigma$. Let assignment α' be given by $\alpha'(v) = [\![v\sigma]\!]^{\mathcal{M}}_\alpha$. Then $\mathcal{M} \models_\alpha C_i$ and $\mathcal{M} \not\models_{\alpha'} C_i$. Thus $\mathcal{M} \not\models_{\alpha'} \boldsymbol{B}$, and by the satisfiability of $\boldsymbol{T}_i$, there has to be a $\boldsymbol{B}'$ with $\mathcal{M} \models_{\alpha'} \boldsymbol{B}'$. Since $\boldsymbol{B}$ is 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 α does meet $\overline{\theta}$, i.e., $\mathcal{M} \models_\alpha \overline{\theta}$.

Satisfiability in $\mathcal{M}$ for an infinite $\boldsymbol{T} = \bigcup_{i=1}^{\infty} \boldsymbol{T}_i$ follows from the fact that satisfiability in $\mathcal{M}$ is a universal property (it has the form 'for all literals and all constraints on the branch ...'), and is therefore by standard model-theoretic reasoning preserved under limit constructions. □

Theorem 3 (Merge). *If a hyper tableau $\boldsymbol{T}$ closes by constraint merge, then $\boldsymbol{T}$ is not satisfiable.*

Proof. If $\boldsymbol{T}$ closes by constraint merge then there is a way to pick a finite initial stage $\boldsymbol{T}'$ of $\boldsymbol{T}$, and pick constraints $\overline{\sigma_1}$, $\ldots$, $\overline{\sigma_n}$, one on each tableau branch of $\boldsymbol{T}'$, such that $\sigma_1 \sqcap \cdots \sqcap \sigma_n \neq \bot\!\!\!\bot$. Thus, there is a ground substitution θ with $\theta \sqsubseteq \sigma_1 \sqcap \cdots \sqcap \sigma_n$. Note that we can associate with each ground substitution an assignment in a model, as follows. If α is an assignment for $\mathcal{M}$, and θ is a ground substitution, then the assignment $\theta\alpha$ is given by $\theta\alpha(v) = [\![v\theta]\!]^{\mathcal{M}}_\alpha$. Thus, for any model $\mathcal{M}$ and any assignment α for $\mathcal{M}$ it will be the case that $\mathcal{M} \models_{\theta\alpha} \sigma_1 \sqcap \cdots \sqcap \sigma_n$. So for any $\mathcal{M}$ there is an assignment α' with $\mathcal{M} \models_{\alpha'} \sigma_1$ and ... and $\mathcal{M} \models_{\alpha'} \sigma_n$, i.e., with $\mathcal{M} \not\models_{\alpha'} \overline{\sigma_1}$ and ... and $\mathcal{M} \not\models_{\alpha'} \overline{\sigma_n}$. In other words, for any $\mathcal{M}$ there has to be an assignment that does not meet any of the constraints $\overline{\sigma_1}, \ldots, \overline{\sigma_n}$. □

Theorem 4 (Soundness). *If there is a hyper tableau refutation for a clause set Φ, then Φ is unsatisfiable.*

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

A variable map θ meets a constraint $\overline{\sigma}$ if $\theta \sqcap \sigma = \bot\!\!\!\bot$; θ is *compatible* with a branch $\boldsymbol{B}$ if θ meets all constraints $\overline{\sigma}$ on $\boldsymbol{B}$; θ is *compatible* with a tableau $\boldsymbol{T}$ if θ is compatible with at least one branch $\boldsymbol{B}$ of $\boldsymbol{T}$.

Theorem 5 (Compatibility). *If a tableau $\boldsymbol{T}$ is open, then every ground variable map θ for vars($\boldsymbol{T}$) is compatible with $\boldsymbol{T}$.*

Proof. Assume $\boldsymbol{T}$ consists of (open) branches $\{\boldsymbol{B}_i\}_{i\geq 0}$. We have to show that every grounding is compatible with at least one $\boldsymbol{B}_i$. Suppose θ is a grounding for vars($\boldsymbol{T}$) that is not compatible with any $\boldsymbol{B} \in \boldsymbol{T}$. Note that θ need not be a substitution, as the set vars($\boldsymbol{T}$) may be infinite. Then for each of the $\boldsymbol{B}_i$ there is a constraint $\overline{\sigma_i}$ on $\boldsymbol{B}_i$ such that $\sigma_i \sqcap \theta \neq \perp\!\!\!\perp$. Since θ is grounding, $\sigma_i \sqcap \theta = \theta$, i.e., $\theta \sqsubseteq \sigma_i$. Since variable maps modulo renaming form a complete lattice under $\sqsubseteq$, it follows that $\theta \sqsubseteq \sqcap_{(i\geq 0)}\sigma_i$. Now, since any tableau is finitely branching, and since any constraint is at finite distance from the root of the tableau, by König's lemma there has to be a *finite* set of constraints $\overline{\sigma_1}, \ldots, \overline{\sigma_n}$ with $\forall i \geq 0\ \exists j \leq n$ such that $\overline{\sigma_j}$ occurs on $\boldsymbol{B}_i$. But then $\sigma_1 \sqcap \cdots \sqcap \sigma_n \neq \perp\!\!\!\perp$, and contradiction with the assumption that $\boldsymbol{T}$ is open. □

Theorem 6 (Model Generation). *Every fair open tableau for Φ has a model $\mathcal{M}$ with $\mathcal{M} \models \Phi$.*

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 hyper 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. By the fairness of the tableau construction process, this model also satisfies Φ.
- 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 θ 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 θ. Collect the ground instances of the positive literals of $\boldsymbol{B}$. The union, for all groundings θ, of the sets of ground positive literals collected from branches compatible with θ, constitutes a model for the tableau. Again, by the fairness of the tableau construction process, the model also satisfies Φ. □

Theorem 7 (Completeness). *If a clause set Φ is unsatisfiable, then there exists a hyper tableau refutation for Φ.*

Proof. Immediate from the Model Generation Theorem. □

6 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 sub-tree [6]. The usual versions of free variable tableaux are all destructive. Clearly, the CHT calculus is non-destructive. A tableau calculus is *proof confluent* if every tableau

for an unsatisfiable clause set Φ can be expanded to a closed tableau [6]. Again, it is clear that the CHT calculus is proof-confluent.

Because of its non-destructiveness and proof-confluence, fair computation with constrained hyper tableaux is easy. We give a mere sketch. First apply *Expand* to every all positive clause, on every branch. Next, use the list of positive literals on a branch to select the candidates from the clause set Φ for a match. For a given P on the branch and candidate clause ϕ, determine whether P needs to be copied for a match. If so, apply *Expand* again to generate the appropriate number of alphabetic variants. Next apply *Expand* to the mixed and the negative clauses. As the applications of *Expand* are non-destructive, no backtracking is ever needed in the merge check for closure.

As one of the referees pointed out, there is scope for further redundancy tests. E.g., for the clause set $\{Pa, Pb, Px \Rightarrow Q\}$ we would get Q twice on the branch. This can be avoided by adding a check like 'never expand a branch with a clause instance if it is already true in all models of the branch'. In the same spirit, if a branch is expanded with a literal A, but a proper instance $A\sigma$ is already present, then $A\sigma$ may be deleted from the branch, on condition of course that none of the variables in A are rigid.

We are experimenting with an implementation of CHT reasoning in Haskell [7], with merge checks for closure performed on tableau branches represented as lazy lists. Since the method is essentially breadth-first, space consumption is an issue, and it remains to be seen what the practical merits of the approach are.

7 Related Work

The standard reference for free variable reasoning in first order tableaux is [3]. 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 [4] to do tableau proof search by merging closing substitutions for tableau branches into a closing substitution for the whole tableau. This suggestion is worked out in [5]. The difference between that approach and the present one is that we use disunification constraints rather than unification constraints. In our approach, the negations of the substitutions that close a branch are viewed as constraints on branch satisfiability, and a tableau remains open along as there is *no* way to unify a list of constraints selected from each branch. Since this is done in a setting where open branches only contain positive literals, it is ensured that a constraint can never clash with a branch literal.

The idea to enrich tableau branches with history lists for keeping track of the clause instances used in the construction of the branch is from [1]. As is mentioned there, this bookkeeping stratagem makes hyper tableaux a decision engine for satisfiability of the Bernays Schönfinkel class (relational $\exists^*\forall^*$ sentences without equality). The clause form of such sentences may have skolem constants, but since there are no skolem functions, any clause has only a finite number of

instances. Thus, the history conditions ensure that tableau developments for Bernays Schönfinkel sentences are always finite.

One of the referees drew my attention to [8], an earlier proposal for handling rigid variables in a hyper tableau setting (with no variable constraints involved, however), for which completeness unfortunately remained open. The present paper settles this issue.

As far as I know, the idea to use constraints on the interpretation of rigid tableau variables for model generation from open free variable tableaux is new. This idea, by the way, applies to free variable tableau reasoning in general. Instead of using a closure rule that applies a most general unifying substitution σ for A and $\neg A'$ to a whole tableau, generate the constraint $\overline{\theta}$, where θ is the restriction of σ to the rigid variables of the current branch, and add a constraint merge for closure check: see [10] for details.

Acknowledgments Many thanks to the members of the *Dynamo team* (Balder ten Cate, Juan Heguiabehere and Breanndán Ó Nualláin), to Bernhard Beckert, Martin Giese, Maarten Marx and Yde Venema, and to three anonymous referees.

References

[1] Baumgartner, P. Hyper tableaux — the next generation. In *Proceedings International Conference on Automated Reasoning with Analytic Tableaux and Related Methods* (1998), H. d. Swart, Ed., no. 1397 in Lecture Notes in Computer Science, Springer, pp. 60–76.

[2] Baumgartner, P., Fröhlich, P., and Niemelä, I. Hyper tableaux. In *Proceedings JELIA 96* (1996), Lecture Notes in Artificial Intelligence, Springer.

[3] Fitting, M. *First-order Logic and Automated Theorem Proving; Second Edition*. Springer Verlag, Berlin, 1996.

[4] Giese, M. 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`.

[5] Giese, M. Incremental closure of free variable tableaux. In *IJCAR 2001 Proceedings* (2001).

[6] Hähnle, R. Tableaux and related methods. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Elsevier Science Publishers, to appear, 2001.

[7] Jones, S. P., Hughes, J., et al. Report on the programming language Haskell 98. Available from the Haskell homepage: `http://www.haskell.org`, 1999.

[8] Kühn, M. Rigid hypertableaux. In *KI-97: Advances in Artificial Intelligence* (1997), G. Brewka, C. Habel, and B. Nebel, Eds., pp. 87–98.

[9] Robinson, J. Automated deduction with hyper-resolution. *International Journal of Computer Mathematics 1* (1965), 227–234.

[10] van Eijck, J. Model generation from constrained free variable tableaux. In *IJCAR 2001 – Short Papers* (Siena, 2001), R. Goré, A. Leitsch, and T. Nipkov, Eds., pp. 160–169.