Introduction to Logic in Computer Science: Autumn 2007

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

Tableaux for First-order Logic

The next part of the course will be an introduction to analytic tableaux for classical first-order logic:

- Quick review of syntax and semantics of first-order logic
- Quantifier rules for Smullyan-style and KE-style tableaux
- Soundness and completeness proofs
- Automatic generation of countermodels
- Discussion of efficiency issues, undecidability
- Free-variable tableaux to increase efficiency
- Tableaux for first-order logic with equality
- Clause tableaux for input in CNF

Syntax of FOL

The *syntax* of a language defines the way in which basic elements of the language may be put together to form clauses of that language. In the case of FOL, the basic ingredients are (besides the logic operators): *variables*, *function symbols*, and *predicate symbols*. Each function and predicate symbol is associated with an *arity* $n \geq 0$.

Definition 1 (Terms) *We inductively define the set of terms as the smallest set such that:*

(1) every variable is a term;

(2) if f is a function symbol of arity k and $t_1, \ldots, t_k$ are terms, then $f(t_1, \ldots, t_k)$ is also a term.

Function symbols of arity 0 are better known as *constants*.

Syntax of FOL (2)

Definition 2 (Formulas) *We inductively define the set of formulas as the smallest set such that:*

(1) if P is a predicate symbol of arity k and $t_1 \ldots, t_k$ are terms, then $P(t_1, \ldots, t_k)$ is a formula;

(2) if φ and ψ are formulas, so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\varphi \rightarrow \psi$;

(3) if x is a variable and φ is a formula, then $(\forall x)\varphi$ and $(\exists x)\varphi$ are also formulas.

Syntactic sugar: $\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$; $\top \equiv P \vee \neg P$ (for an arbitrary 0-place predicate symbol P); $\bot \equiv \neg\top$.

Also recall: *atoms*, *literals*, *ground terms*, *bound* and *free variables*, *closed formulas* (aka *sentences*), ...

Semantics of FOL

The *semantics* of a language defines the meaning of clauses in that language. In the case of FOL, we do this through the notion of models (and variable assignments).

Definition 3 (Models) *A model is a pair $\mathcal{M} = (\mathcal{D}, \mathcal{I})$, where $\mathcal{D}$ (the domain) is a non-empty set of objects and $\mathcal{I}$ (the interpretation function) is mapping each n-place function symbol f to some n-ary function $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ and each n-place predicate symbol P to some n-ary relation $P^{\mathcal{I}} : \mathcal{D}^n \to \{true, false\}$.*

Note that this definition also covers the cases of 0-place function symbols (constants) and predicate symbols.

Semantics of FOL (2)

Definition 4 (Assignments) *A variable assignment over a domain $\mathcal{D}$ is a function g from the set of variables to $\mathcal{D}$.*

Definition 5 (Valuation of terms) *We define a valuation function $val_{\mathcal{I},g}$ over terms as follows:*

$$\begin{array}{rcl} val_{\mathcal{I},g}(x) & = & g(x) \quad \text{for variables } x \\ val_{\mathcal{I},g}(f(t_1,\ldots,t_n)) & = & f^{\mathcal{I}}(val_{\mathcal{I},g}(t_1),\ldots,val_{\mathcal{I},g}(t_n)) \end{array}$$

Definition 6 (Assignment variants) *Let g and g' be assignments over $\mathcal{D}$ and let x be a variable, Then g' is called an x-variant of g iff $g(y) = g'(y)$ for all variables $y \neq x$.*

Semantics of FOL (3)

Definition 7 (Satisfaction relation) *We write $\mathcal{M}, g \models \varphi$ to say that the formula φ is satisfied in the model $\mathcal{M} = (\mathcal{I}, \mathcal{D})$ under the assignment g. The relation $\models$ is defined inductively as follows:*

(1) $\mathcal{M}, g \models P(t_1,\ldots,t_n)$ iff $P^{\mathcal{I}}(val_{\mathcal{I},g}(t_1),\ldots,val_{\mathcal{I},g}(t_n)) = true$;

(2) $\mathcal{M}, g \models \neg\varphi$ iff not $\mathcal{M}, g \models \varphi$;

(3) $\mathcal{M}, g \models \varphi \wedge \psi$ iff $\mathcal{M}, g \models \varphi$ and $\mathcal{M}, g \models \psi$;

(4) $\mathcal{M}, g \models \varphi \vee \psi$ iff $\mathcal{M}, g \models \varphi$ or $\mathcal{M}, g \models \psi$;

(5) $\mathcal{M}, g \models \varphi \to \psi$ iff not $\mathcal{M}, g \models \varphi$ or $\mathcal{M}, g \models \psi$;

(6) $\mathcal{M}, g \models (\forall x)\varphi$ iff $\mathcal{M}, g' \models \varphi$ for all x-variants g' of g; and

(7) $\mathcal{M}, g \models (\exists x)\varphi$ iff $\mathcal{M}, g' \models \varphi$ for some x-variant g' of g.

Semantics of FOL (4)

Observe that in the case of *closed* formulas φ the variable assignment g does not matter (we just write $\mathcal{M} \models \varphi$).

Satisfiability. A closed formula φ is called *satisfiable* iff it has a model, *i.e.* there exists a model $\mathcal{M}$ with $\mathcal{M} \models \varphi$.

Validity. A closed formula φ is called *valid* iff for every model $\mathcal{M}$ we have $\mathcal{M} \models \varphi$. We write $\models \varphi$.

Consequence relation. Let φ be a closed formula and let Δ be a set of closed formulas. We write $\Delta \models \varphi$ iff whenever $\mathcal{M} \models \psi$ holds for all $\psi \in \Delta$ then also $\mathcal{M} \models \varphi$ holds.

Quantifier Rules

Both the KE-style and the Smullyan-style tableau method for propositional logic can be extended with the following rules.

Gamma Rules:

$$\frac{(\forall x)A}{A[t/x]} \qquad \frac{\neg(\exists x)A}{\neg A[t/x]}$$

Delta Rules:

$$\frac{(\exists x)A}{A[c/x]} \qquad \frac{\neg(\forall x)A}{\neg A[c/x]}$$

Here, t is an *arbitrary ground term* and c is a *constant symbol* that is *new* to the branch.

Unlike all other rules, the gamma rule may have to be applied more than once to the same formula on the same branch.

Substitution. $\varphi[t/x]$ denotes the formula we get by replacing each free occurrence of the variable x in the formula φ by the term t.

Smullyan's Uniform Notation

Formulas of universal (γ) and existential (δ) type:

γ	$\gamma_1(u)$
$(\forall x)A$	$A[u/x]$
$\neg(\exists x)A$	$\neg A[u/x]$

δ	$\delta_1(u)$
$(\exists x)A$	$A[u/x]$
$\neg(\forall x)A$	$\neg A[u/x]$

We can now state gamma and delta rules as follows:

$$\frac{\gamma}{\gamma_1(t)} \qquad \frac{\delta}{\delta_1(c)}$$

where:

- t is an arbitrary ground term
- c is a constant symbol new to the branch

Exercises

Give Smullyan-style or KE-style tableau proofs for the following two arguments:

- $(\forall x)P(x) \vee (\forall x)Q(x) \models \neg(\exists x)(\neg P(x) \wedge \neg Q(x))$
- $\models (\exists x)(P(x) \vee Q(x)) \leftrightarrow (\exists x)P(x) \vee (\exists x)Q(x)$

Soundness and Completeness

Let φ be a first-order formula and Δ a set of such formulas. We write $\Delta \vdash \varphi$ to say that there exists a closed tableau for $\Delta \cup \{\neg\varphi\}$.

Theorem 1 (Soundness) *If* $\Delta \vdash \varphi$ *then* $\Delta \models \varphi$.

Theorem 2 (Completeness) *If* $\Delta \models \varphi$ *then* $\Delta \vdash \varphi$.

We shall prove soundness and completeness only for Smullyan-style tableaux (but it's almost the same for KE-style tableaux).

Important note: The mere *existence* of a closed tableau does not mean that we have an effective method of *finding* it! Concretely: we don't know how often we need to apply the gamma rule and what terms to use for the substitutions.

Proof of Soundness

This works exactly as in the propositional case ($\rightsquigarrow$ last week).

The central step is to show that each of the expansion rules preserves satisfiability:

- If a non-branching rule is applied to a satisfiable branch, the result is another satisfiable branch.
- If a branching rule is applied to a satisfiable branch, at least one of the resulting branches is also satisfiable.

Proof of Soundness (cont.)

Gamma rule: If γ appears on a branch, you may add $\gamma_1(t)$ for any ground term t to the same branch.

<u>Proof:</u> suppose branch B with $\gamma \equiv (\forall x)\gamma_1(x) \in B$ is satisfiable
$\Rightarrow$ there exists $\mathcal{M} = (\mathcal{D}, \mathcal{I})$ s.t. $\mathcal{M} \models B$ and hence $\mathcal{M} \models (\forall x)\gamma_1(x)$
$\Rightarrow$ for all assignments g: $\mathcal{M}, g \models \gamma_1(x)$; choose g' s.t. $g'(x) = t^{\mathcal{I}}$
$\Rightarrow \mathcal{M}, g' \models \gamma_1(x) \;\Rightarrow\; \mathcal{M} \models \gamma_1(t) \;\Rightarrow\; \mathcal{M} \models B \cup \{\gamma_1(t)\}$ ✓

Delta rule: If δ appears on a branch, you may add $\delta_1(c)$ for any new constant symbol c to the same branch.

<u>Proof:</u> suppose branch B with $\delta \equiv (\exists x)\delta_1(x) \in B$ is satisfiable
$\Rightarrow$ there exists $\mathcal{M} = (\mathcal{D}, \mathcal{I})$ s.t. $\mathcal{M} \models B$ and hence $\mathcal{M} \models (\exists x)\delta_1(x)$
$\Rightarrow$ there exists a variable assignment g s.t. $\mathcal{M}, g \models \delta_1(x)$
now suppose $g(x) = d \in \mathcal{D}$; define new model $\mathcal{M}' = (\mathcal{D}, \mathcal{I}')$ with $\mathcal{I}'$ like $\mathcal{I}$ but additionally $c^{\mathcal{I}'} = d$ (this is possible, because c is *new*)
$\Rightarrow \mathcal{M}' \models \delta_1(c)$ and $\mathcal{M}' \models B \;\Rightarrow\; \mathcal{M}' \models B \cup \{\delta_1(c)\}$ ✓

Hintikka's Lemma

Definition 8 (Hintikka set) *A set of first-order formulas H is called a Hintikka set provided the following hold:*

(1) not both $P \in H$ and $\neg P \in H$ for atomic formulas P;

(2) if $\neg\neg\varphi \in H$ then $\varphi \in H$ for all formulas φ;

(3) if $\alpha \in H$ then $\alpha_1 \in H$ and $\alpha_2 \in H$ for alpha formulas α;

(4) if $\beta \in H$ then $\beta_1 \in H$ or $\beta_2 \in H$ for beta formulas β.

(5) for all terms t built from function symbols in H (at least one constant symbol): if $\gamma \in H$ then $\gamma_1(t)$ for gamma formulas γ;

(6) if $\delta \in H$ then $\delta_1(t) \in H$ for some term t, for delta formulas δ.

Lemma 1 (Hintikka) *Every Hintikka set is satisfiable.*

Proof of Hintikka's Lemma

Construct a model $\mathcal{M} = (\mathcal{D}, \mathcal{I})$ from a given Hintikka set H:

- $\mathcal{D}$: set of terms constructible from function symbols appearing in H (add one constant symbol in case there are none)
- $\mathcal{I}$: (1) function symbols are being interpreted "as themselves": $f^{\mathcal{I}}(d_1, \ldots, d_n) = f(d_1, \ldots, d_n)$; (2) predicate symbols: $P^{\mathcal{I}}(d_1, \ldots, d_n) = \mathit{true}$ iff $P(d_1, \ldots, d_n) \in H$

<u>Claim:</u> $\varphi \in H$ entails $\mathcal{M} \models \varphi$.

<u>Proof:</u> By structural induction. [...] ✓

Proof of Completeness

Fairness. We call a tableau proof *fair* iff every non-literal gets *eventually* analysed on every branch and, additionally, every gamma formula gets *eventually* instantiated with every term constructible from the function symbols appearing on a branch.

Proof sketch. We will show the contrapositive: assume $\Delta \nvdash \varphi$ and try to conclude $\Delta \not\models \varphi$.

If there is no proof for $\Delta \cup \{\neg\varphi\}$ (assumption), then there can also be no *fair* proof. Observe that any fairly constructed non-closable branch enumerates the elements of a Hintikka set H.

H is satisfiable (Hintikka's Lemma) and we have $\Delta \cup \{\neg\varphi\} \subseteq H$.

So there is a model for $\Delta \cup \{\neg\varphi\}$, *i.e.* we get $\Delta \not\models \varphi$. ✓

Summary: Basic Tableaux Systems for FOL

- Two tableau methods for first-order logic: Smullyan-style (syntactic branching) and KE-style (semantic branching)
- Soundness and completeness
- Undecidability: gamma rule is the culprit

Automatic Generation of Countermodels

Besides deduction and theorem proving, another important application of automated reasoning is *model generation*.

Using tableaux, we sometimes get termination for failed proofs and can extract a counterexample (particularly nice for KE).

Saturated Branches

An open branch is called *saturated* iff every non-literal has been analysed at least once and, additionally, every gamma formula has been instantiated with every term we can construct using the function symbols on the branch.

Failing proofs. A tableau with an open saturated branch can never be closed, *i.e.* we can stop an declare the proof a failure.

The solution? This only helps us in special cases though.
(A single 1-place function symbol together with a constant is already enough to construct infinitely many terms ...)

Propositional logic. In propositional logic (where we have no gamma formulas), after a limited number of steps, every branch will be either closed or saturated. This gives us a decision procedure.

Countermodels

If a KE proof fails with a *saturated open branch*, you can use it to help you define a model $\mathcal{M}$ for all the formulas on that branch:

- domain: set of all terms we can construct using the function symbols appearing on the branch (so-called *Herbrand universe*)
- terms are interpreted as themselves
- interpretation of predicate symbols: see literals on branch

In particular, $\mathcal{M}$ will be a model for the premises Δ and the negated conclusion $\neg\varphi$, *i.e.* a *counterexample* for $\Delta \models \varphi$.

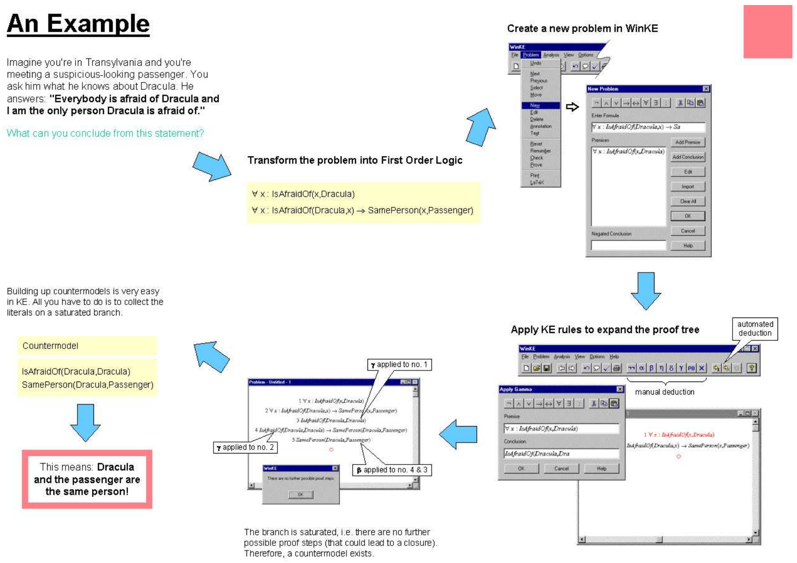
You can do the same with Smullyan-style tableaux, but for KE distinct open branches always generate distinct models.

Take care: There's a bug in WinKE—sometimes, what is presented as a countermodel is in fact only *part* of a countermodel (but it can always be extended to an actual model).

Exercise

Construct a counterexample for the following argument:

- $(\forall x)(P(x) \vee Q(x)) \models^? (\forall x)P(x) \vee (\forall x)Q(x)$



Extensions and Variations

Next we'll be looking into several extensions and variations of the tableau method for first-order logic:

- Free-variable tableaux to increase efficiency
- Tableaux for first-order logic with equality
- Clause tableaux for input in CNF

Efficiency Issues

Due to the undecidability of first-order logic there can be no general method for finding a closed tableau for a given theorem (although its existence is guaranteed by completeness).

Nevertheless, there are some heuristics:

- As in the propositional case, use "deterministic" rules first: propositional rules except PB and the delta rule.
- As in the propositional case, use *beta simplification*.
- Use the gamma rule a "reasonable" number of times (with "promising" substitutions) before attempting to use PB.

  > *Example:* for the automated theorem prover implemented in WinKE you can choose n, the maximum number of applications of the gamma rule on a given branch before PB will be used.

- Use analytic PB only.

A Problem and an Idea

One of the main drawbacks of either variant of the tableau method for FOL, as presented so far, is that for every application of the gamma rule we have to *guess* a good term for the substitution.

And idea to circumvent this problem would be to try to "postpone" the decision of what substitution to choose until we attempt to close branches, at which stage we would have to check whether there are complementary literals that are unifiable.

Instead of substituting with ground terms we will use *free variables*. As this would be cumbersome for KE-style tableaux, we will only present free-variable Smullyan-style tableaux.

But first, we need to speak about *unification* in earnest ...

Unification

Definition 9 (Unification) *A substitution σ (of possibly several variables by terms) is called a unifier of a set of formulas $\Delta = \{\varphi_1, \ldots, \varphi_n\}$ iff $\sigma(\varphi_1) = \cdots = \sigma(\varphi_n)$ holds. We also write $|\sigma(\Delta)| = 1$ and call Δ unifiable.*

Definition 10 (MGU) *A unifier μ of a set of formulas Δ is called a most general unifier (mgu) of Δ iff for every unifier σ of Δ there exists a substitution σ' with $\sigma = \mu \circ \sigma'$.*

(The composition $\mu \circ \sigma'$ is the substitution we get by first applying μ to a formula and then σ'.)

Remark. We also speak of unifiers (and mgus) for sets of *terms*.

Unification Algorithm: Preparation

We shall formulate a unification algorithm for literals only, but it can easily be adapted to work with general formulas (or terms).

Subexpressions. Let φ be a literal. We refer to formulas and terms appearing within φ as the *subexpressions* of φ. If there is a subexpression in φ starting at position i we call it $\varphi^{(i)}$ (otherwise $\varphi^{(i)}$ is undefined; say, if there is a comma at the ith position).

Disagreement pairs. Let φ and ψ be literals with $\varphi \neq \psi$ and let i be the smallest number such that $\varphi^{(i)}$ and $\psi^{(i)}$ are defined and $\varphi^{(i)} \neq \psi^{(i)}$. Then $(\varphi^{(i)}, \psi^{(i)})$ is called the *disagreement pair* of φ and ψ. Example:

$$\begin{array}{rcl} \varphi & = & P(g_1(c), f_1(a, g_1(x), g_2(a, g_1(b))) \\ \psi & = & P(g_1(c), f_1(a, g_1(x), g_2(f_2(x, y), z)) \\ & & \qquad\qquad\qquad\qquad\quad \uparrow \end{array}$$

Disagreement pair: $(a, f_2(x, y))$

Robinson's Unification Algorithm

set $\mu := []$ (empty substitution)
while $|\mu(\Delta)| > 1$ **do** {
  pick a disagreement pair p in $\mu(\Delta)$;
  if *no variable in* p **then** {
    stop and **return** 'not unifiable';
  } **else** {
    let $p = (x, t)$ with x being a variable;
    if x *occurs in* t **then*** {
      stop and **return** 'not unifiable';
    } **else** {
      set $\mu := \mu \circ [t/x]$;
    }
  }
}
return μ;

Input: Δ (set of literals)
Output: μ (mgu of Δ)
or 'not unifiable'

* so-called *occurs-check*

Exercise

Run Robinson's Unification Algorithm to compute the mgu of the following set of literals (assuming x, y and z are the only variables):

$$\Delta \;=\; \{Q(f(x, g(x, a)), z),\; Q(y, h(x)),\; Q(f(b, w), z)\}$$

Free-variable Tableaux

The Smullyan-style tableau method for propositional logic can be extended with the following quantifier rules.

Gamma Rules: $\dfrac{\gamma}{\gamma_1(y)}$

Delta Rules: $\dfrac{\delta}{\delta_1(f(x_1, \ldots, x_n))}$

Here y is a (new) *free variable*, f is a *new function symbol*, and $x_1, \ldots, x_n$ are the *free variables occurring in* δ.

An additional tableau rule is added to the system: an arbitrary *substitution* may be applied to the entire tableau.

The closure rule is being restricted to complementary literals (to avoid dealing with unification for formulas with bound variables).

Closing Branches

There are different ways in which to use the interplay of the substitution rule and the closure rule:

- One approach is to develop the tableau until a single application of the substitution rule produces complementary literals on all branches. Nice in theory, but not that efficient.
- Another approach is to compute mgus of potentially complementary literals to close branches as you go along. This is more goal-directed, but as substitutions carry over to other branches, we may make suboptimal choices.

Exercises

Give free-variable tableaux for the following theorems:

- $\models (\exists x)(P(x) \to (\forall y)P(y))$
- $\models (\exists x)(\forall y)(\forall z)(P(y) \vee Q(z) \to P(x) \vee Q(x))$
- $\models (\exists x)(P(x) \vee Q(x)) \to (\exists x)P(x) \vee (\exists x)Q(x)$

Handling Equality

Three approaches to tableaux for first-order logic with equality:

- Introduce a binary predicate symbol to represent equality and explicitly axiomatise it as part of the premises. This requires no extension to the calculus. $\rightsquigarrow$ Possible, but very inefficient.
- Add expansion and closure rules to your favourite tableau method to handle equality. There are different ways of doing this (we'll look at some of them next).
- For free-variable tableaux, take equalities and inequalities into account when searching for substitutions to close branches (*"E-unification"*). $\rightsquigarrow$ Requires serious work on algorithms for E-unification, but is potentially the best method.

We use the symbol $\approx$ to denote the equality predicate.

Axiomatising Equality

We can use our existing tableau methods for first-order logic with equality if we explicitly axiomatise the (relevant) properties of the special predicate symbol $\approx$ (using infix-notation for readability):

- Reflexivity axiom: $(\forall x)(x \approx x)$
- Replacement axiom for each n-place function symbol f: $(\forall x_1)\cdots(\forall x_n)(\forall y_1)\cdots(\forall y_n)[(x_1 \approx y_1) \wedge \cdots \wedge (x_n \approx y_n) \to f(x_1,\ldots,x_n) \approx f(y_1,\ldots,y_n)]$
- Replacement axiom for each n-place predicate symbol P: $(\forall x_1)\cdots(\forall x_n)(\forall y_1)\cdots(\forall y_n)[(x_1 \approx y_1) \wedge \cdots \wedge (x_n \approx y_n) \to (P(x_1,\ldots,x_n) \to P(y_1,\ldots,y_n))]$

This is taken from Fitting's textbook, where you can also find a proof showing that it works.

Jeffrey's Tableau Rules for Equality

These are the classical tableau rules for handling equality and apply to *ground* tableaux:

$$\frac{\begin{array}{c}A(t)\\ t \approx s\end{array}}{A(s)} \qquad \frac{\begin{array}{c}A(t)\\ s \approx t\end{array}}{A(s)} \qquad \frac{\neg(t \approx t)}{\times}$$

Exercise: Show $\models (a \approx b) \wedge P(a,a) \to P(b,b)$.

For even just slightly more complex examples, these rules quickly give rise to a huge search space ...

Reeves' Tableau Rules for Equality

These rules, also for ground tableaux, are more "goal-oriented" and hence somewhat reduce the search space (let P be atomic):

$$\frac{\begin{array}{c}P(t_1,\ldots,t_n)\\ \neg P(s_1,\ldots,s_n)\end{array}}{\neg((t_1 \approx s_1)\wedge\cdots\wedge(t_n \approx s_n))} \qquad \frac{\neg(f(t_1,\ldots,t_n)\approx f(s_1,\ldots,s_n))}{\neg((t_1 \approx s_1)\wedge\cdots\wedge(t_n \approx s_n))}$$

We also need a rule for symmetry, and the closure rule from before:

$$\frac{t \approx s}{s \approx t} \qquad \frac{\neg(t \approx t)}{\times}$$

Exercise: Show $\models (\forall x)(\forall y)(\forall z)[(x \approx y)\wedge(y \approx z) \rightarrow (x \approx z)]$.

Fitting's Tableau Rules for Equality

Jeffrey's approach can also be combined with free-variable tableaux, but we need to interleave substitution steps with other steps to make equality rules applicable. Alternatively, equality rules can also be formulated so as to integrate substitution:

$$\frac{\begin{array}{c}A(t)\\ t' \approx s\end{array}}{[A(s)]\mu} \qquad \frac{\begin{array}{c}A(t)\\ s \approx t'\end{array}}{[A(s)]\mu} \qquad \frac{\neg(t \approx t')}{\times\mu}$$

Here μ is an mgu of t and t' and must be applied to the entire tree.

Exercise: Show that the following set of formulas is unsatisfiable:

$$\begin{array}{c}\{\ (\forall x)[(g(x)\approx f(x)) \vee \neg(x \approx a)],\\ (\forall x)(g(f(x)) \approx x),\ b \approx c,\\ P(g(g(a)),b),\ \neg P(a,c)\ \}\end{array}$$

Tableaux and Resolution

The most popular deduction system in automated reasoning is the *resolution* method (to be discussed briefly later on in the course).

Resolution works for formulas in CNF. This restriction to a normal form makes resolution very efficient. Still, the tableau method has several advantages:

- Tableaux proofs are a lot easier to read than resolution proofs.
- Input may not be in CNF and translation may result in an exponential blow-up.
- For some non-classical logic, translation may be impossible.

Nevertheless, people interested in developing powerful theorem provers for FOL (rather than in using tableaux as a more general framework) are often interested in tableaux for CNF, also to allow for better comparison with resolution.

Normal Forms

Recall: Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF) for propositional logic

Prenex Normal Form. A FOL formula φ is said to be in *Prenex Normal Form* iff all its quantifiers (if any) "come first". The quantifier-free part of φ is called the *matrix* of φ.

Every sentence can be transformed into a logically equivalent sentence in Prenex Normal Form.

Transformation into Prenex Normal Form

If necessary, rewrite the formula first to ensure that no two quantifiers bind the same variable and no variable has both a free and a bound occurrence (variables need to be "named apart").

$$
\begin{array}{rclrcl}
\neg(\forall x)A & \equiv & (\exists x)\neg A & \neg(\exists x)A & \equiv & (\forall x)\neg A \\
((\forall x)A) \wedge B & \equiv & (\forall x)(A \wedge B) & ((\exists x)A) \wedge B & \equiv & (\exists x)(A \wedge B) \\
((\forall x)A) \vee B & \equiv & (\forall x)(A \vee B) & ((\exists x)A) \vee B & \equiv & (\exists x)(A \vee B)
\end{array}
$$

etc.

To avoid making mistakes, formulas involving $\rightarrow$ or $\leftrightarrow$ should first be translated into formulas using only $\neg$, $\wedge$ and $\vee$ (and quantifiers).

Skolemisation

Skolemisation is the process of removing existential quantifiers from a formula in Prenex Normal Form (without affecting satisfiability).

Algorithm. Given: a formula in Prenex Normal Form.

(1) If necessary, turn the formula into a sentence by adding $(\forall x)$ in front for every free variable x ("universal closure").

(2) While there are still existential quantifiers, repeat: replace

- $(\forall x_1) \cdots (\forall x_n)(\exists y)\varphi$ with
- $(\forall x_1) \cdots (\forall x_n)\varphi[f(x_1, \ldots, x_n)/y]$,
  where f is a new function symbol.

Skolemisation (cont.)

Definition 11 (Skolem Normal Form) *A formula φ is said to be in Skolem Normal Form (SNF) iff it is of the following form:*

$$\varphi \quad = \quad (\forall x_1)(\forall x_2) \cdots (\forall x_n)\, \varphi',$$

where φ' is a quantifier-free formula in CNF (with $n \in \mathbb{N}_0$).

Theorem 3 (Skolemisation) *For every formula φ there exists a formula φ_{sk} in SNF such that φ is satisfiable iff φ_{sk} is satisfiable. φ_{sk} can be obtained from φ through the process of Skolemisation.*

Proof: By induction over the sequence of transformation steps in the Skolemisation algorithm [details omitted].

Note that φ and φ_{sk} are *not* (necessarily) equivalent.

Exercise

Compute the Skolem Normal Form of the following formula:

$$(\forall x)(\exists y)[P(x, g(y)) \rightarrow \neg(\forall x)Q(x)]$$

Clauses

Clauses. A *clause* is a set of literals. Logically, it corresponds to the *disjunction* of these literals.

Sets of clauses. A *set of clauses* logically corresponds to the *conjunction* of the clauses in the set.

This means, any formula in Skolem Normal Form can be rewritten as a set of clauses. Variables are understood to be implicitly universally quantified. Example:

$$\{\,\{P(x), Q(y)\},\ \{\neg P(f(y))\}\,\} \ \sim\ (\forall x)(\forall y)[(P(x) \vee Q(y)) \wedge \neg P(f(y))]$$

Clause Tableaux

The input (root of the tree) is a set of clauses. We need a beta rule and a closure rule for literals:

$$\frac{\{L_1, \ldots, L_n\}}{\{L_1\} \mid \cdots \mid \{L_n\}} \qquad \frac{\begin{array}{c}\{L\}\\ \{\neg L\}\end{array}}{\times}$$

We also need a rule that allows us to add any number of *copies* of the input clauses to a branch, with variables being renamed (corresponds to multiple applications of the gamma rule).

The *substitution rule* is the same as before: arbitrary substitutions may be applied to the entire tableau (but will typically be guided by potentially complementary literals).

Clause Tableaux: Alternative Presentation

The presentation on the previous slide attempts to be as close as possible to what we have done before, but the following alternative presentation is more common:

- Initialise the tableau with $\top$ and keep the set of clauses S to be shown unsatisfiable separate.
- Applying an *extension rule* means choosing a branch B and a new instance $\{L_1, \ldots, L_n\}$ of a clause in S, and then appending n children below B and labelling them with $\{L_1\}$ to $\{L_n\}$.
- Close branches (on literals) using suitable mgu's as usual.

Observe how this extension rule combines beta and gamma rules.

Exercises

Give a closed tableau for the following set of clauses:

- $\{\{P(x), Q(x)\}, \{\neg Q(x), \neg R(x)\}, \{\neg P(a)\}, \{R(x)\}\}$

Give a proof using clause tableaux for the following theorem:

- $\models (\exists x)(\forall y)(\forall z)(P(y) \vee Q(z) \rightarrow P(x) \vee Q(x))$

Guiding Proofs

Even for clause tableaux, the search space is generally still huge. A lot of research has gone into finding refinements of the basic procedure to guide proof search. For instance:

- A *connection tableau* is a clause tableau in which every non-leaf node labelled with a literal L has a child labelled with the complement of L.
- A clause tableau is called *regular* iff no branch contains more than one copy of the same literal.

Completeness can still be guaranteed if we restrict search to regular connection tableaux. See the handbook chapter by Hähnle (2001) for a precise statement of this result.

Summary: Extensions and Variations

- Free-variable tableaux: postpone instantiations and close by unification ($\leadsto$ compute mgus with Robinson's algorithm)
- Handling equality: several approaches, including several ways of defining additional expansion rules
- Clause tableaux: simplified system for clauses rather than general formulas ($\leadsto$ requires translation into SNF)
- Much of the material covered can be found in:
  - R. Hähnle. *Tableaux and Related Methods*. In: A. Robinson and A. Voronkov (eds.), *Handbook of Automated Reasoning*, Elsevier Science and MIT Press, 2001.

  The material on handling equality is taken from:
  - B. Beckert. Semantic Tableaux with Equality. *Journal of Logic and Computation*, 7(1):39–58, 1997.