Introduction to Logic in Computer Science: Autumn 2006

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

Plan for Today

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

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 1 (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 2 (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:

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

$\uparrow$

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 $\mu := \mu \circ [x/t]$;
        }
    }
}
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:

$$\frac{\gamma}{\gamma_1(y)}$$

Delta Rules:

$$\frac{\delta}{\delta_1(f(x_1,\ldots,x_n))}$$

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 use 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:

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

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. $\leadsto$ 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"*). $\leadsto$ 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: $(\forall x_1) \cdots (\forall x_n)(\forall y_1) \cdots (\forall y_n)[(x_1 \approx y_1) \wedge \cdots \wedge (x_n \approx y_n) \rightarrow f(x_1, \ldots, x_n) \approx f(y_1, \ldots, y_n)]$
- Replacement axiom for each n-place predicate symbol P: $(\forall x_1) \cdots (\forall x_n)(\forall y_1) \cdots (\forall y_n)[(x_1 \approx y_1) \wedge \cdots \wedge (x_n \approx y_n) \rightarrow (P(x_1, \ldots, x_n) \rightarrow P(y_1, \ldots, y_n))]$

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:

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

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

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:

$$\frac{A(t) \quad t' \approx s}{[A(s)]\mu} \qquad \frac{A(t) \quad s \approx t'}{[A(s)]\mu} \qquad \frac{\neg(t \approx t')}{\times\mu}$$

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:

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

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").

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

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

Skolemisation (cont.)

Definition 3 (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 1 (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 written 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).

Summary

- 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 what we have done today 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.