2.14 Example: Neuman-Stubblebine Protocol

- Formalization of a concrete application:
  Neuman-Stubblebine key exchange protocol.
- State-of-the-art in automated theorem proving.
- Proof by refutation:
  inconsistency ⇒ intruder can break the protocol.
- Proof by consistency:
  consistency ⇒ no unsafe states exist.
- Termination requires elimination of redundancy.

The Problem

Automatic Analysis of Security Protocols using SPASS: An Automated Theorem Prover for First-Order Logic with Equality

by Christoph Weidenbach

The growing importance of the internet causes a growing need for security protocols that protect transactions and communication. It turns out that the design of such protocols is highly error-prone. Therefore, there is a need for tools that automatically detect flaws like, e. g., attacks by an intruder. Here we show that our automated theorem prover SPASS can successfully be used to analyze the Neuman-Stubblebine key exchange protocol [1]. To this end the protocol is formalized in logic and then the security properties are automatically analyzed by SPASS. A detailed description of the analysis can be found in [2].

The animation successively shows two runs of the Neuman-Stubblebine [1] key exchange protocol. The first run works the way the protocol is designed to do, i. e., it establishes a secure key between Alice and Bob.

The second run shows a potential problem of the protocol. An intruder may intercept the final message sent from Alice to Bob, replace it with a different message and may eventually own a key that Bob believes to be a secure key with Alice. The initial situation for the protocol is that the two participants Alice and Bob want to establish a secure key for communication among them. They do so with the help of a trusted server Trust where both already have a secure key for communication with Trust. The below picture shows a sequence of four message exchanges that eventually establishes the key.

Neuman-Stubblebine: A Regular Run

1. $A \xrightarrow{A, N_a} B$
2. $B \xrightarrow{B, N_b, E_{K_{bt}}(A, N_a, T_b)} T$
3. $T \xrightarrow{E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b} A$
4. $A \xrightarrow{E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)} B$

What Can Happen?

How can an intruder now break this protocol? The key K_{ab} is only transmitted inside encrypted parts of messages and we assume that an intruder cannot break any keys nor does he know any of the initial keys K_{at} or K_{bt}. Here is the solution:

Breaking Neuman-Stubblebine

1. $A \xrightarrow{A, N_a} B$
2. $B \xrightarrow{B, N_b, E_{K_{bt}}(A, N_a, T_b)} T$
3. $T \xrightarrow{E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b} A$
4. $A \xrightarrow{E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)} B$

3. $I \xrightarrow{E_{K_{bt}}(A, N_a, T_b), E_{N_a}(N_b)} B$

The Formalization

The key idea of the formalization is to describe the set of sent messages. This is done by introducing a monadic predicate M in first-order logic. Furthermore, every participant holds its set of known keys, represented by the predicates Ak for Alice's keys, Bk for Bob's keys, Tk for Trust's keys and Ik for the keys the intruder knows. The rest of the used symbols is introduced and explained with the first appearance in a formula. Then the four messages can be translated into the following formulae:

Step 1) A, Na

$$Ak(key(at, t)) \tag{1}$$
$$M(sent(a, b, pair(a, na))) \tag{2}$$

The two formulae express that initially Alice holds the key at for communication with t (for Trust) and that she sends the first message. In order to formalize messages we employ a three place function sent where the first argument is the sender, the second the receiver and the third the content of the message. So the constant a represents Alice, b Bob, t Trust and i Intruder. The functions *pair* (*triple*, *quadr*) simply form sequences of messages of the indicated length.

Step 2) $B, E(Kbt, A, Na, Tb), Nb$

$$Bk(key(bt, t)) \tag{3}$$
$$\begin{aligned}\forall xa, xna\ [&M(sent(xa, b, pair(xa, xna))) \\ &\to M(sent(b, t, triple(b, nb(xna), \\ &\qquad encr(triple(xa, xna, tb(xna)), bt)))))]\end{aligned} \tag{4}$$

Bob holds the key bt for secure communication with Trust and whenever he receives a message of the form of message 1 (formula (2)), he sends a key request to Trust according to message 2. Note that encryption is formalized by the two place function *encr* where the first argument is the date and the second argument the key. Every lowercase symbol starting with an x denotes a variable. The functions nb and tb generate, respectively, a new nonce and time span out of xa's (Alice's) request represented by her nonce xna.

Step 3) $E(Kat, B, Na, Kab, Tb),\ E(Kbt, A, Kab, Tb),\ Nb$

$$Tk(key(at, a))) \wedge Tk(key(bt, b)) \tag{5}$$
$$\begin{aligned}&\forall xb, xnb, xa, xna, xbet, xbt, xat, xk \\ &\quad [\ (M(sent(xb, t, triple(xb, xnb, encr(triple(xa, xna, xbet), xbt)))) \\ &\quad\ \wedge Tk(key(xbt, xb)) \\ &\quad\ \wedge Tk(key(xat, xa))) \\ &\quad\ \to M(sent(t, xa, triple(encr(quadr(xb, xna, kt(xna), xbet), xat), \\ &\qquad\quad encr(triple(xa, kt(xna), xbet), xbt), xnb)))\ ]\end{aligned} \tag{6}$$

Trust holds the keys for Alice and Bob and answers appropriately to a message in the format of message 2. Note that decryption is formalized by unification with an appropriate term structure where it is checked that the necessary keys are known to Trust. The server generates the key by applying his key generation function kt to the nonce xna.

Step 4) $E(Kbt, A, Kab, Tb),\ \ E(Kab, Nb)$

$$
\begin{aligned}
&\forall xnb, xbet, xk, xm, xb, xna \\
&\quad [\ M(sent(t, a, triple(encr(quadr(xb, xna, xk, xbet), at), xm, xnb)) \\
&\quad\ \ \rightarrow (M(sent(a, xb, pair(xm, encr(xnb, xk)))) \wedge Ak(key(xk, xb)))\ ]
\end{aligned} \tag{7}
$$

$$
\begin{aligned}
&\forall xbet, xk, xnb, xa, xna \\
&\quad [\ M(sent(xa, b, pair(encr(triple(xa, xk, tb(xna)), bt), \\
&\qquad\qquad\qquad encr(nb(xna), xk))) \rightarrow Bk(key(xk, xa))]
\end{aligned} \tag{8}
$$

Finally, Alice answers according to the protocol to message 3 and stores the generated key for communication, formula (7). Formula (8) describes Bob's behaviour when he receives Alice's message. Bob decodes this message and stores the new key as well.

A's Formalization Part I

$$A \xrightarrow{A, N_a} B$$

$$
\begin{aligned}
&P(a) \\
&Ak(key(at, t)) \\
&M(sent(a, b, pair(a, na))) \\
&Sa(pair(b, na))
\end{aligned}
$$

Sa is Alice's local store that will eventually be used to verify the nonce when it is sent back to her in Step (3).

A's Formalization Part II

$$
\begin{aligned}
T &\xrightarrow{E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b} A \\
A &\xrightarrow{E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)} B
\end{aligned}
$$

$$
\begin{aligned}
&\forall xb, xna, xnb, xk, xbet, xm \\
&\quad [\,M(sent(t, a, triple(encr(quadr(xb, xna, xk, xbet), at), xm, xnb))) \\
&\quad\ \ \wedge Sa(pair(xb, xna)) \\
&\quad\ \ \rightarrow \\
&\quad\ M(sent(a, xb, pair(xm, encr(xnb, xk)))) \\
&\quad\ \ \wedge Ak(key(xk, xb))\,]
\end{aligned}
$$

The Intruder

The Intruder is modeled as an exhaustive hacker. He records all messages, decomposes the messages as far as possible and generates all possible new compositions. Furthermore, any object he has at hand is considered as a key and tried to used for encryption as well as for decryption. All these messages are posted. The set of messages the intruder has available is represented by the predicate Im.

The participants are Alice, Bob, Trust and Intruder:

$$P(a) \wedge P(b) \wedge P(t) \wedge P(i) \tag{9}$$

The intruder records all messages:

$$\forall xa, xb, xm\ [M(sent(xa, xb, xm)) \rightarrow Im(xm)] \tag{10}$$

He decomposes and decrypts all messages he owns the key for:

$$\forall u, v\ [Im(pair(u, v)) \rightarrow Im(u) \wedge Im(v)] \tag{11}$$
$$\forall u, v, w\ [Im(triple(u, v, w)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w)] \tag{12}$$
$$\forall u, v, w, z\ [Im(quadr(u, v, w, z)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z)] \tag{13}$$
$$\forall u, v, w\ [Im(encr(u, v)) \wedge Ik(key(v, w)) \rightarrow Im(u)] \tag{14}$$

He composes all possible messages:

$$\forall u, v\ [Im(u) \wedge Im(v) \rightarrow Im(pair(u, v))] \tag{15}$$
$$\forall u, v, w\ [Im(u) \wedge Im(v) \wedge Im(w) \rightarrow Im(triple(u, v, w))] \tag{16}$$
$$\forall u, v, w, x\ [Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(x) \rightarrow Im(quadr(u, v, w, x))] \tag{17}$$

He considers every item to be a key and uses it for encryption:

$$\forall v, w\ [Im(v) \wedge P(w) \rightarrow Ik(key(v, w))] \tag{18}$$
$$\forall u, v, w\ [Im(u) \wedge Ik(key(v, w)) \wedge P(w) \rightarrow Im(encr(u, v))] \tag{19}$$

He sends everything:

$$\forall x, y, u\ [P(x) \wedge P(y) \wedge Im(u) \rightarrow M(sent(x, y, u))] \tag{20}$$

Finally we must formalize the insecurity requirement. Intruder must not have any key for communication with Bob that Bob believes to be a secure key for Alice:

$$\exists x\ [Ik(key(x, b)) \wedge Bk(key(x, a))]$$

SPASS Solves the Problem

Now the protocol formulae together with the intruder formulae (9)-(20) and the insecurity formula above can be given to SPASS. Then SPASS automatically proves that this formula holds and that the problematic key is the nonce Na. The protocol can be repaired by putting type checks on the keys, such that keys can no longer be confused with nonces. This can be added to the SPASS first-order logic formalization. *Then SPASS disproves the insecurity formula above*. This capability is currently unique to SPASS. Although some other provers might be able to prove that the insecurity formula holds in the formalization without type checks, we are currently not aware of any prover that can disprove the insecurity formula in the formalization with type checking. Further details can be found in [2], below. The experiment is available in full detail from the SPASS home page in the download area.

References:

[1] Neuman, B. C. and Stubblebine, S. G., 1993, A note on the use of timestamps as nonces, ACM SIGOPS, Operating Systems Review, 27(2), 10-14.

[2] Weidenbach, C., 1999, Towards an automatic analysis of security protocols in first-order logic, in 16th International Conference on Automated Deduction, CADE-16, Vol. 1632 of LNAI, Springer, pp. 378-382.

2.15 Summary: Resolution Theorem Proving

- Resolution is a machine calculus.
- Subtle interleaving of enumerating ground instances and proving inconsistency through the use of unification.
- Parameters: atom ordering $\succ$ and selection function S. On the non-ground level, ordering constraints can (only) be solved approximatively.
- Completeness proof by constructing candidate interpretations from productive clauses $C \vee A$, $A \succ C$; inferences with those reduce counterexamples.
- *Local* restrictions of inferences via $\succ$ and S
  $\Rightarrow$ fewer proof variants.
- *Global* restrictions of the search space via elimination of redundancy
  $\Rightarrow$ computing with "smaller" clause sets;
  $\Rightarrow$ termination on many decidable fragments.
- However: not good enough for dealing with orderings, equality and more specific algebraic theories (lattices, abelian groups, rings, fields)
  $\Rightarrow$ further specialization of inference systems required.

2.16 Semantic Tableaux

Literature:

M. Fitting: First-Order Logic and Automated Theorem Proving, Springer-Verlag, New York, 1996, chapters 3, 6, 7.

R. M. Smullyan: First-Order Logic, Dover Publ., New York, 1968, revised 1995.

Like resolution, semantic tableaux were developed in the sixties, by R. M. Smullyan on the basis of work by Gentzen in the 30s and of Beth in the 50s. (According to Fitting, semantic tableaux were first proposed by the Polish scientist Z. Lis in a paper in Studia Logica 10, 1960 that was only recently rediscovered.)

Idea

Idea (for the propositional case):

A set $\{F \wedge G\} \cup N$ of formulas has a model if and only if $\{F \wedge G,\ F,\ G\} \cup N$ has a model.

A set $\{F \vee G\} \cup N$ of formulas has a model if and only if $\{F \vee G,\ F\} \cup N$ or $\{F \vee G,\ G\} \cup N$ has a model.

(and similarly for other connectives).

To avoid duplication, represent sets as paths of a tree.

Continue splitting until two complementary formulas are found $\Rightarrow$ inconsistency detected.

A Tableau for $\{P \wedge \neg(Q \vee \neg R),\ \neg Q \vee \neg R\}$

1. $P \wedge \neg(Q \vee \neg R)$
2. $\neg Q \vee \neg R$

Left branch	Right branch
3. $\neg Q$	4. $\neg R$
5. P	10. P
6. $\neg(Q \vee \neg R)$	11. $\neg(Q \vee \neg R)$
7. $\neg Q$	
8. $\neg\neg R$	
9. R	

This tableau is not "maximal", however the first "path" is. This path is not "closed", hence the set $\{1, 2\}$ is satisfiable. (These notions will all be defined below.)

Properties

Properties of tableau calculi:

analytic: inferences according to the logical content of the symbols.

goal oriented: inferences operate directly on the goal to be proved (unlike, e. g., ordered resolution).

global: some inferences affect the entire proof state (set of formulas), as we will see later.

Propositional Expansion Rules

Expansion rules are applied to the formulas in a tableau and expand the tableau at a leaf. We append the conclusions of a rule (horizontally or vertically) at a *leaf*, whenever the premise of the expansion rule matches a formula appearing *anywhere* on the path from the root to that leaf.

Negation Elimination

$$\frac{\neg\neg F}{F} \qquad \frac{\neg\top}{\bot} \qquad \frac{\neg\bot}{\top}$$

α-Expansion

(for formulas that are essentially conjunctions: append subformulas α_1 and α_2 one on top of the other)

$$\frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}}$$

β-Expansion

(for formulas that are essentially disjunctions:
append β_1 and β_2 horizontally, i. e., branch into β_1 and β_2)

$$\frac{\beta}{\beta_1 \mid \beta_2}$$

Classification of Formulas

conjunctive			disjunctive		
α	α_1	α_2	β	β_1	β_2
$X \wedge Y$	X	Y	$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$\neg(X \vee Y)$	$\neg X$	$\neg Y$	$X \vee Y$	X	Y
$\neg(X \rightarrow Y)$	X	$\neg Y$	$X \rightarrow Y$	$\neg X$	Y

We assume that the binary connective $\leftrightarrow$ has been eliminated in advance.

Tableaux: Notions

A *semantic tableau* is a marked (by formulas), finite, unordered tree and inductively defined as follows: Let $\{F_1, \ldots, F_n\}$ be a set of formulas.

(i) The tree consisting of a single path

$$\begin{array}{l} F_1 \\ \vdots \\ F_n \end{array}$$

is a tableau for $\{F_1, \ldots, F_n\}$. (We do not draw edges if nodes have only one successor.)

(ii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if T' results from T by applying an expansion rule then T' is also a tableau for $\{F_1, \ldots, F_n\}$.

A *path* (from the root to a leaf) in a tableau is called *closed*, if it either contains $\perp$, or else it contains both some formula F and its negation $\neg F$. Otherwise the path is called *open*.

A tableau is called *closed*, if all paths are closed.

A *tableau proof* for F is a closed tableau for $\{\neg F\}$.

A path P in a tableau is called *maximal*, if for each non-atomic formula F on P there exists a node in P at which the expansion rule for F has been applied.

In that case, if F is a formula on P, P also contains:

(i) F_1 and F_2, if F is a α-formula,

(ii) F_1 or F_2, if F is a β-formula, and

(iii) F', if F is a negation formula, and F' the conclusion of the corresponding elimination rule.

A tableau is called *maximal*, if each path is closed or maximal.

A tableau is called *strict*, if for each formula the corresponding expansion rule has been applied at most once on each path containing that formula.

A tableau is called *clausal*, if each of its formulas is a clause.

A Sample Proof

One starts out from the negation of the formula to be proved.

$$
\begin{array}{lcr}
1. & \neg[(P \rightarrow (Q \rightarrow R)) \rightarrow ((P \vee S) \rightarrow ((Q \rightarrow R) \vee S))] & \\
2. & (P \rightarrow (Q \rightarrow R)) & [1_1] \\
3. & \neg((P \vee S) \rightarrow ((Q \rightarrow R) \vee S)) & [1_2] \\
4. & P \vee S & [3_1] \\
5. & \neg((Q \rightarrow R) \vee S)) & [3_2] \\
6. & \neg(Q \rightarrow R) & [5_1] \\
7. & \neg S & [5_2]
\end{array}
$$

$$
\begin{array}{ll}
8.\ \neg P\ [2_1] & 9.\ Q \rightarrow R\ [2_2] \\
10.\ P\ [4_1] \quad 11.\ S\ [4_2] &
\end{array}
$$

There are three paths, each of them closed.

Properties of Propositional Tableaux

We assume that T is a tableau for $\{F_1, \ldots, F_n\}$.

Theorem 2.49 *$\{F_1, \ldots, F_n\}$ satisfiable $\Leftrightarrow$ some path (i. e., the set of its formulas) in T is satisfiable.*

Proof. By induction over the structure of T. □

Corollary 2.50 *T closed $\Rightarrow$ $\{F_1, \ldots, F_n\}$ unsatisfiable*

Theorem 2.51 *Let T be a strict propositional tableau. Then T is finite.*

Proof. New formulas resulting from expansion are either $\bot$, $\top$ or subformulas of the expanded formula. By strictness, on each path a formula can be expanded at most once. Therefore, each path is finite, and a finitely branching tree with finite paths is finite by Lemma 2.19. □

Conclusion: Strict and maximal tableaux can be effectively constructed.

Refutational Completeness

Theorem 2.52 *Let P be a maximal, open path in a tableau. Then set of formulas on P is satisfiable.*

Proof. (We consider only the case of a clausal tableau.)
Let N be the set of formulas on P. As P is open, $\bot$ is not in N. Let $C \vee A$ and $D \vee \neg A$ be two resolvable clauses in N. One of the two subclauses C or D, C say, is not empty, as otherwise P would be closed. Since P is maximal, in P the β-rule was applied on $C \vee A$. Therefore, P (and N) contains a proper subclause of $C \vee A$, and hence $C \vee A$ is redundant w. r. t. N. By the same reasoning, if N contains a clause that can be factored, that clause must be redundant w. r. t. N. In other words, N is saturated up to redundancy w. r. t. Res(olution). Now apply Theorem 2.24 to prove satisfiability of N. ☐

Theorem 2.53 $\{F_1, \dots, F_n\}$ *satisfiable* $\Leftrightarrow$ *there exists no closed strict tableau for* $\{F_1, \dots, F_n\}$.

Proof. One direction is clear by Theorem 2.49. For the reverse direction, let T be a strict, maximal tableau for $\{F_1, \dots, F_n\}$ and let P be an open path in T. By the previous theorem, the set of formulas on P, and hence by Theorem 2.49 the set $\{F_1, \dots, F_n\}$, is satisfiable. ☐

Consequences

The validity of a propositional formula F can be established by constructing a strict, maximal tableau for $\{\neg F\}$:

- T closed $\Leftrightarrow$ F valid.
- It suffices to test complementarity of paths w. r. t. atomic formulas (cf. reasoning in the proof of Theorem 2.52).
- Which of the potentially many strict, maximal tableaux one computes does not matter. In other words, tableau expansion rules can be applied don't-care non-deterministically ("*proof confluence*").
- The expansion strategy, however, can have a dramatic impact on tableau size.
- Since it is sufficient to saturate paths w. r. t. ordered resolution (up to redundancy), tableau expansion rules can be even more restricted, in particular by certain ordering constraints.

Semantic Tableaux for First-Order Logic

Additional classification of quantified formulas:

universal		existential	
γ	$\gamma(t)$	δ	$\delta(t)$
$\forall xF$	$F[t/x]$	$\exists xF$	$F[t/x]$
$\neg\exists xF$	$\neg F[t/x]$	$\neg\forall xF$	$\neg F[t/x]$

Moreover we assume that the set of variables X is partitioned into 2 disjoint infinite subsets X_g and X_f, so that bound [free] variables variables can be chosen from X_g [X_f]. (This avoids the variable capturing problem.)

Additional Expansion Rules

γ-expansion

$$\frac{\gamma}{\gamma(x)} \quad \text{where } x \text{ is a variable in } X_f$$

δ-expansion

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

where f is a *new* Skolem function, and the x_i are the free variables in δ

Skolemization becomes part of the calculus and needs not necessarily be applied in a preprocessing step. Of course, one could do Skolemization beforehand, and then the δ-rule would not be needed.

Note that the rules are parametric, instantiated by the choices for x and f, respectively. Strictness here means that only one instance of the rule is applied on each path to any formula on the path.

In this form the rules go back to Hähnle and Schmitt: The liberalized δ-rule in free variable semantic tableaux, J. Automated Reasoning 13,2, 1994, 211–221.

Definition: Free-Variable Tableau

Let $\{F_1, \ldots, F_n\}$ be a set of *closed formulas*.

(i) The tree consisting of a single path

$$\begin{array}{c} F_1 \\ \vdots \\ F_n \end{array}$$

is a tableau for $\{F_1, \ldots, F_n\}$.

(ii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if T' results by applying an expansion rule to T, then T' is also a tableau for $\{F_1, \ldots, F_n\}$.

(iii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if σ is a substitution, then $T\sigma$ is also a tableau for $\{F_1, \ldots, F_n\}$.

The *substitution rule* (iii) may, potentially, modify all the formulas of a tableau. This feature is what is makes the tableau method a *global proof method*. (Resolution, by comparison, is a local method.)

If one took (iii) literally, by repeated application of γ-rule one could enumerate all substitution instances of the universally quantified formulas. That would be a major drawback compared with resolution. Fortunately, we can improve on this.

Example

$$\begin{array}{lll}
1. & \neg[\exists w \forall x\ p(x, w, f(x, w)) \rightarrow \exists w \forall x \exists y\ p(x, w, y)] & \\
2. & \exists w \forall x\ p(x, w, f(x, w)) & 1_1\ [\alpha] \\
3. & \neg \exists w \forall x \exists y\ p(x, w, y) & 1_2\ [\alpha] \\
4. & \forall x\ p(x, c, f(x, c)) & 2(c)\ [\delta] \\
5. & \neg \forall x \exists y\ p(x, v_1, y) & 3(v_1)\ [\gamma] \\
6. & \neg \exists y\ p(b(v_1), v_1, y) & 5(b(v_1))\ [\delta] \\
7. & p(v_2, c, f(v_2, c)) & 4(v_2)\ [\gamma] \\
8. & \neg p(b(v_1), v_1, v_3) & 6(v_3)\ [\gamma]
\end{array}$$

7. and 8. are complementary (modulo unification):

$$v_2 \doteq b(v_1),\ \ c \doteq v_1,\ \ f(v_2, c) \doteq v_3$$

is solvable with an mgu $\sigma = [c/v_1, b(c)/v_2, f(b(c), c)/v_3]$, and hence, $T\sigma$ is a closed (linear) tableau for the formula in 1.

AMGU-Tableaux

Idea: Restrict the substitution rule to unifiers of complementary formulas.

We speak of an A*MGU-Tableau*, whenever the substitution rule is only applied for substitutions σ for which there is a path in T containing two *literals* $\neg A$ and B such that $\sigma = \mathrm{mgu}(A, B)$.

Correctness

Given an signature Σ, by Σ^{sko} we denote the result of adding infinitely many new Skolem function symbols which we may use in the δ-rule.

Let $\mathcal{A}$ be a Σ^{sko}-interpretation, T a tableau, and β a variable assignment over $\mathcal{A}$.

T is called $(\mathcal{A}, \beta)$-*valid*, if there is a path P_β in T such that $\mathcal{A}, \beta \models F$, for each formula F on P_β.

T is called *satisfiable* if there exists a structure $\mathcal{A}$ such that for each assignment β the tableau T is $(\mathcal{A}, \beta)$-valid. (This implies that we may choose P_β depending on β.)

Theorem 2.54 *Let T be a tableau for $\{F_1, \dots, F_n\}$, where the F_i are closed Σ-formulas. Then $\{F_1, \dots, F_n\}$ is satisfiable $\Leftrightarrow$ T is satisfiable.*

Proof. Proof of "$\Rightarrow$" by induction over the depth of T. For δ one needs to reuse the ideas for proving that Skolemization preserves [un-]satisfiability. $\square$

Incompleteness of Strictness

Strictness for γ is incomplete:

$$
\begin{array}{lcr}
1. & \neg[\forall x\; p(x) \rightarrow (p(c) \wedge p(b))] & \\
2. & \forall x\; p(x) & 1_1 \\
3. & \neg(p(c) \wedge p(b)) & 1_2 \\
4. & p(v_1) & 2(v_1)
\end{array}
$$

$$
5.\quad \neg p(c) \quad 3_1 \qquad\qquad 6.\quad \neg p(b) \quad 3_2
$$

If we placed a strictness requirement also on applications of γ, the tableau would only be expandable by the substitution rule. However, there is no substitution (for v_1) that can close both paths simultaneously.

Multiple Application of γ Solves the Problem

$$
\begin{array}{lcc}
1. & \neg[\forall x\, p(x) \rightarrow (p(c) \wedge p(b))] & \\
2. & \forall x\, p(x) & 1_1 \\
3. & \neg(p(c) \wedge p(b)) & 1_2 \\
4. & p(v_1) & 2_{v_1}
\end{array}
$$

$$
\begin{array}{lll}
5. & \neg p(c) & 3_1
\end{array}
\qquad
\begin{array}{lll}
6. & \neg p(b) & 3_2 \\
7. & p(v_2) & 2_{v_2}
\end{array}
$$

The point is that different applications of γ to $\forall x\, p(x)$ may employ different free variables for x.

Now, by two applications of the AMGU-rule, we obtain the substitution $[c/v_1, b/v_2]$ closing the tableau.

Therefore *strictness for* γ should from now on mean that each *instance* of γ (depending on the choice of the free variable) is applied at most once to each γ-formula on any path.

Refutational Completeness

Theorem 2.55 *$\{F_1, \ldots, F_n\}$ satisfiable $\Leftrightarrow$ there exists no closed, strict AMGU-Tableau for $\{F_1, \ldots, F_n\}$.*

For the proof one defines a fair tableau expansion process converging against an infinite tableau where on each path each γ-formula is expanded into all its variants (modulo the choice of the free variable).

One may then again show that each path in that tableau is saturated (up to redundancy) by resolution. This requires to apply the lifting lemma for resolution in order to show completeness of the AMGU-restriction.

How Often Do we Have to Apply γ?

Theorem 2.56 *There is no recursive function $f : F_\Sigma \times F_\Sigma \rightarrow \mathbb{N}$ such that, if the closed formula F is unsatisfiable, then there exists a closed tableau for F where to all formulas $\forall x G$ appearing in T the γ-rule is applied at most $f(F, \forall x G)$ times on each path containing $\forall x G$.*

Otherwise unsatisfiability or, respectively, validity for first-order logic would be decidable. In fact, one would be able to enumerate in finite time all tableaux bounded in depth as indicated by f. In other words, free-variable tableaux are not recursively bounded in their depth.

Again $\forall$ is treated like an infinite conjunction. By repeatedly applying γ, together with the substitution rule, one can enumerate all instances $F[t/x]$ vertically, that is, conjunctively, in each path containing $\forall xF$.

Semantic Tableaux vs. Resolution

- Both methods are machine methods on which today's provers are based upon.
- Tableaux: global, goal-oriented, "backward".
- Resolution: local, "forward".
- Goal-orientation is a clear advantage if only a small subset of a large set of formulas is necessary for a proof. (Note that resolution provers saturate also those parts of the clause set that are irrelevant for proving the goal.)
- Like resolution, the tableau method, in order to be useful in practice, must be accompanied by refinements: lemma generation, ordering restrictions, efficient term and proof data structures.
- Resolution can be combined with more powerful redundancy elimination methods.
- Because of its global nature redundancy elimination is more difficult for the tableau method.
- Resolution can be refined to work well with equality (see next chapter) and algebraic structures; for tableaux this seems to be impossible.

2.17 Other Inference Systems

Instantiation-based methods for FOL:

- Partial instantiation;
- Resolution-based instance generation;
- Disconnection calculus.

Further (mainly propositional) proof systems:

- Hilbert calculus;
- Sequent calculus;

- Natural deduction.

Instantiation-Based Methods for FOL

Idea:

Overlaps of complementary literals produce instantiations (as in resolution);

However, contrary to resolution, clauses are not recombined.

Instead: treat remaining variables as constant and use efficient propositional proof methods, such as DPLL.

There are both saturation-based variants, such as partial instantiation [Hooker et al.] or resolution-based instance generation (Inst-Gen) [Ganzinger and Korovin], and tableau-style variants, such as the disconnection calculus [Billon; Letz and Stenz].

Hilbert Calculus

Hilbert calculus:

Direct proof method (proves a theorem from axioms, rather than refuting its negation)

Axiom schemes, e. g.,

$$F \to (G \to F)$$
$$(F \to (G \to H)) \to ((F \to G) \to (F \to H))$$

plus Modus ponens:

$$\frac{F \qquad F \to G}{G}$$

Unsuitable for both humans and machines.

Natural Deduction

Natural deduction (Prawitz):

Models the concept of proofs from assumptions as humans do it (cf. Fitting or Huth/Ryan).

Sequent Calculus

Sequent calculus (Gentzen):

Assumptions internalized into the data structure of sequents

$$F_1, \ldots, F_m \rightarrow G_1, \ldots, G_k$$

meaning

$$F_1 \wedge \cdots \wedge F_m \rightarrow G_1 \vee \cdots \vee G_k$$

A kind of mixture between natural deduction and semantic tableaux.

Perfect symmetry between the handling of assumptions and their consequences.

Can be used both backwards and forwards.