Translating a Satallax Refutation to a Tableau Refutation Encoded in Coq

Bachelor's Thesis - Final Talk

Andreas Teucke

Advisor: Chad Brown
Supervisor: Gert Smolka

Department of Computer Science
Saarland University

May 6, 2011

The Goal

Verifying the result of Satallax

- Satallax reduces higher-order theorem proving to checking unsatisfiability of SAT problems.
- Can we trust the result of Satallax and the SAT-solver?
- Goal: Extract a higher-order proof, where one can easily check correctness.
- Solution: A tableau refutation encoded as a Coq Proof Script.

Outline

1. Introduction
   - Goal
   - Conjecture
2. Recap
   - First Talk
   - Proposal Talk
3. Implementation
   - Search
   - Completion
   - Output

The Conjecture

The result of Satallax defines a finite tableau calculus

- Minisat is able to indirectly prove refutability,
- while only knowing the HO formulae and tableau steps encoded as literals and clauses.
- Conjecture: A tableau calculus restricted to these formulae refutes the HO problem.

Outline

1. Introduction
   - Goal
   - Conjecture
2. Recap
   - First Talk
   - Proposal Talk
3. Implementation
   - Search
   - Completion
   - Output

First Talk

The restriction to a fixed set of formulae creates some obstacles:

- Analytic cut is in some cases required
- The $\exists$ rule cannot introduce arbitrary new variables, but we can enforce an ordering such that the witnesses are fresh.

Proposal Talk

Theorem

If we have an abstract refutation for some problem A
- as a result from Satallax -,
then A is refutable in the restricted tableau calculus $\mathcal{T}$

Abstract Refutation
A formal Definition

Definition (abstract refutation (F, S))

Let A be an open branch, F a finite set of formulae and S a function from variables to terms.
Then we call (F, S) an abstract refutation of A, if

1. $<_S$ is acyclic
2. For every $x \in dom\ S$, x is not free in A
3. For every full expansion B, either
   B is refutable in $\mathcal{T}$ in one step or
   there is an $x \in dom\ S$ such that $\exists t \in B$ and $\neg[tx] \in B$
   where $t = S(x)$

Abstract Refutation

An intuitive Definition.

Definition (abstract refutation (F, S))

Let A be an open branch, F a finite set of formulae and S the log of existential witnesses.
Then we call (F, S) an abstract refutation of A, if

1. Existential witnesses are globally fresh, unique variables.
2. There exists an unsatisfiable set of clauses where each clause is a subset of F and encodes either A or a tableau step.

Proof

A constructive proof that builds a simple refutation.

Theorem

If (F, S) is an abstract refutation of A then A is refutable in $\mathcal{T}$

Proof.

1. Apply Cut on $\exists t$ formulae in chronological order of S and introduce their witnesses with the $\exists$ rule.
2. Apply Cut on all remaining formulae in F.
3. Close branches with single step in $\mathcal{T}$.

$\square$

Outline

1 Introduction
- Goal
- Conjecture

2 Recap
- First Talk
- Proposal Talk

3 Implementation
- Search
- Completion
- Output

SAARLAND UNIVERSITY

First Phase: Search

An automated HO theorem prover

The core of the implementation is like an automated higher-order theorem prover.
Common techniques are implemented to improve the result:

- Back-jumping
- Semantic branching

For the proof script we log the steps the search takes.

First Phase: Search

Difference to a pure automated theorem prover

Using the result of Satallax we know the following in advance:

- The search is guaranteed to succeed eventually
- All steps neccessary for the refutation
- All instantiations for $\forall$ and $\exists$ steps

This will make it a lot easier for us.

Preprocessing Steps

- The clause set in the result of Satallax already encodes all tableau steps necessary for a refutation.
- The steps are extracted once from the clauses in a preprocessing phase saving time during the search.
- Reducing the set to its unsatisfiable core often drastically reduces the number of steps.

Static Sorting Heuristic

- A common practice in SAT-solving is to statically sort literals by number of occurrences in the clause set.
- We sort steps in ascending order by the number of alternatives and secondarily in descending order by occurrences of their formulae in the set of steps.
- This static order replaces a dynamic priority queue.

Fixed Instantiations

- Due to the fixed set of formulae
  $\forall$ and $\exists$ instantiations are fixed as well.
- Especially enumerating over infinite higher-order instantiations is avoided
- The clauses left by the reduction to the UNSAT core determine the relevant instantiations for the encoded steps.

Outline

1 Introduction
- Goal
- Conjecture

2 Recap
- First Talk
- Proposal Talk

3 Implementation
- Search
- Completion
- Output

Second Phase: Completion

Satallax does not solve the original problem
as it rewrites input and intermediate results:

- Logical constants are standardised to $\bot, \rightarrow, \forall$ and $=$,
  e.g., $\exists x.s$ rewritten as $\neg\forall x.\neg s$
- $\beta\eta$-reduction
- Double negations are removed
- $s = t$ and $t = s$ are mapped to the same literal

These operations (except β) have to be made explicit for Coq.

Rewrites

The solution:
Apply the Leibniz property of precomputed equalities $s = t$:

$$\forall p. p\, s \to s = t \to p\, t$$

For this we often need to state p explicitly. For example,

to η-reduce $f\,(\lambda x. g\, x)$ using $\lambda f. \lambda x. f\, x = \lambda f. f$.

we need to state $p := \lambda x. f\,(x\, g)$.

A real example

`tab_rew_or H358 H359` (fun (x1:o –> o –> o) => ˜ ((forall (x2:i) (x3:i), (forall (x4:i), in' x4 x2 = in' x4 x3) -> x2 = x3) -> (forall (x2:i), ˜ in' x2 emptyset) -> (forall (x2:i) (x3:i) (x4:i), in' x4 (setadjoin x2 x3) = (˜ x4 = x2 -> in' x4 x3)) -> (forall (x2:i) (x3:i), in' x3 (powerset x2) = (forall (x4:i), in' x4 x3 -> in' x4 x2)) -> (forall (x2:i) (x3:i), in' x3 (setunion x2) = (˜ (forall (x4:i), in' x3 x4 -> (˜ in' x4 x2)))) -> in' emptyset omega -> (forall (x2:i), in' x2 omega -> in' (setadjoin x2 x2) omega) -> (forall (x2:i), ˜ (in' emptyset x2 -> (˜ (forall (x3:i), ˜ (in' x3 omega -> (˜ in' x3 x2)) -> in' (setadjoin x3 x3) x2))) -> (forall (x3:i), in' x3 omega -> in' x3 x2)) -> (forall (x2:i –> i –> o) (x3:i), (forall (x4:i), in' x4 x3 -> (˜ (forall (x5:i), x2 x4 x5 -> (˜ (forall (x6:i), x2 x4 x6 -> x5 = x6))))) -> (˜ (forall (x4:i), ˜ (forall (x5:i), in' x5 x4 = (˜ (forall (x6:i), in' x6 x3 -> (˜ x2 x6 x5))))))) -> (forall (x2:i), ˜ (forall (x3:i), ˜ in' x3 x2) ->...

A real example

- This would continue for thirty slides...
- ... for one rewrite in a proof script with over seven hundred rewrites.
- Although this is a worst case example,
  it shows that rewrites should be avoided if possible.

Lazy Rewriting

To achieve this the translation tries to apply workarounds:

- The refutation from the first phase is modified.
- If appropriate we apply alternative rules ,
  e.g., $\mathcal{T}_\vee$ instead of $\mathcal{T}_\rightarrow$ avoids rewriting $s \vee t$ into $\neg s \rightarrow t$.
- If nothing works rewrite will be applied.

$$\mathcal{T}_\rightarrow \; \frac{s \rightarrow t}{\neg s \mid t} \qquad \mathcal{T}_{\neg\rightarrow} \; \frac{\neg(s \rightarrow t)}{s, \neg t}$$

$$\mathcal{T}_\vee \; \frac{s \vee t}{s \mid t} \qquad \mathcal{T}_\wedge \; \frac{s \wedge t}{s, t}$$

Outline

1 Introduction
- Goal
- Conjecture

2 Recap
- First Talk
- Proposal Talk

3 Implementation
- Search
- Completion
- Output

Third Phase: Proof Script

Tableau rules encoded as Coq tactic macros

To encode a tableau rule such as

$$\mathcal{T}\ \frac{s_1, \ldots, s_l}{t_{1,1}, \ldots, t_{1,m} \mid \ldots \mid t_{n,1}, \ldots, t_{n,m}}$$

we proof the corresponding lemma `T`

$$\begin{aligned} s_1 \to \ldots \to s_l \to \quad & (t_{1,1} \to \ldots \to t_{1,m} \to \bot) \quad \to \ldots \\ \ldots \to \quad & (t_{n,1} \to \ldots \to t_{n,m} \to \bot) \quad \to \bot. \end{aligned}$$

and refine it in a tactic macro

```
refine (T s1..sl _.._); intros t1..tm.
```

An example

Boolean extensionality

$$\mathcal{T}_{BE} \quad \frac{s \neq_o t}{s\,,\, \neg t \;\mid\; \neg s\,,\, t}$$

Lemma TBE:

$$\forall s\ t : o.\ (s \neq t) \rightarrow \quad (s \rightarrow \neg t \rightarrow \bot) \quad \rightarrow$$
$$(\neg s \rightarrow t \rightarrow \bot) \quad \rightarrow \bot$$

```
Ltac tab_be H H1 H2 :=
(refine (TBE H _ _) ; intros H1 H2).
```

SAARLAND UNIVERSITY

Summary

- The result of Satallax defines a small finite tableau calculus that can refute the initial problem.
- The implementation uses its own customized higher-order theorem prover to search in this calculus.
- Future work
  - Learning
  - Satisfiability case

References I

Julian Backes and Chad E. Brown.
Analytic tableaux for higher-order logic with choice.
In Reiner Hähnle Jürgen Giesl, editor, *Automated Reasoning: 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010, Proceedings*, volume 6173 of *LNCS/LNAI*, pages 76–90. Springer, 2010.

Chad E. Brown.
Reducing higher-order theorem proving to a sequence of SAT problems.
In *CADE – the 23rd International Conference on Automated Deduction (To Appear)*. LNCS, Feb 2011.
To Appear.

References II

Niklas Eén and Niklas Sörensson.
An extensible SAT-solver.
In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 333–336. Springer Berlin / Heidelberg, 2004.

Yves Bertot and Pierre Castéran.
Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.
Texts in Theoretical Computer Science. Springer Verlag, 2004.