Automated Reasoning

Brian Kuhns, Chris Lynch, Dohan Kim, Gabe Tucker,
Graham Northup, Josh Gordon, Samuel Heater, Thomas Johnson

April 2019

1 Introduction

The algorithm is an extension of ordered resolution, which attempts to find finite representations of infinite clause families, and proceed with ordered resolution on these families.

2 Formalities

Let all clauses be denoted by $\mathbf{C}$, and let:

- $\mathbf{F}_n$ be the set of all functions with arity n. The set $\mathbf{F}_0$ refers specifically to constants (nilary functions). Hereafter, by convention, $a, b, d, e, \ldots$ represent the constants $\mathbf{F}_0$ (c commonly denotes any clause), while $f, g, h, \ldots$ represent the non-constant functions $\mathbf{F} - \mathbf{F}_0$.
- $\mathbf{T}$ is the set of all terms. Terms are either variables or functions of other terms. Hereafter, variables are usually denoted $x, y, z, \ldots$, with subscripts as appropriate.
- $\mathbf{P}_n : \mathbf{T}^n \to \{\top, \bot\}$ be the set of all predicates on n terms.
- $\mathbf{P}$ is the set of all predicates applied to the appropriate number of terms. $\mathbf{P} = \{p(t_1 \ldots t_n) : p \in \mathbf{P}_n \wedge t_1 \ldots t_n \in \mathbf{T}\}$
- $\mathbf{L}$ denotes the set of all literals $\mathbf{L} = \mathbf{P} \cup \neg\mathbf{P}$
- Let $\mathbf{D}_n$ be the set of all disjunctions over predicates of size n:

  $$\forall l_0, \ldots, l_{n-1} \in \mathbf{L} : \bigvee_{0 \le i < n} l_i \in \mathbf{D_n};$$

  then $\mathbf{C} = \bigcup_{n \in \mathbb{N}} \mathbf{D}_n$.
- An assignment is an ordered pair of a variable and a term indicating that that variable is assigned to that term.

- A substitution is a set of assignments which don't conflict.

Let $\models$ indicate the *proves* relation, a relation from $\mathcal{P}(\mathbf{C})$ to $\mathbf{C}$. This represents that a clause logically follows from the conjunction of some set of clauses. It has the following useful properties: $((B \models c) \wedge (\forall b \in B, A \models b)) \Rightarrow A \models c$ and $\forall a \in A, A \models a$

3 Anatomy of a Scheme

Define a scheme as the pair (I, B) with $B \subseteq \mathbf{C}$ being the set of base clauses in this scheme, and I being the set of implications. Define an implication as $H \to c$ with $H \subseteq \mathbf{C}$ being the hypotheses of this implication, and $c \in \mathbf{C}$ being the conclusion.

The application of an implication to a set of clauses is possible when the conjunction of the set of clauses can be unified with the conjunction of the hypotheses of that implication, the result of which is the conclusion with the unifier of the set of clauses and the hypotheses applied to it. Symbolically S the set of clauses in the scheme, $S' \subseteq S$, (H, c) is an implication, and σ is the most general substitution of S' and H.

$$(S'\sigma \sqsubseteq H\sigma) \Rightarrow c\sigma \in S \tag{1}$$

An implication is syntactically valid if each hypothesis clause subsumes the conclusion. This ensures that an implication can be applied iteratively, as any clause derived will be subsumed by the conclusion and thus the hypothesis.

A scheme is syntactically valid if each implication is valid and each hypothesis of each implication subsumes all of the base clauses.

A clause is in a scheme if it is a base clause or can be derived by applying the implications clauses in the scheme.

A scheme is said to contain a clause iff it can be derived by applying some finite sequence of the implications to the base clauses.

4 Learning Implications

Learning implications is done in two phases: finding candidates and constructing implications from them.

4.1 Candidates

A candidate is a tree whose nodes are clauses and whose leafs are clauses. Suppositions are also clauses but they are handled differently to construct implications.

A candidate should be derived whenever the history of a clause contains a subtree which subsumes it. A clause-tree subsumes another clause-tree if the graphs are isomorphic the and each node subsumes the corresponding node in the other tree.

Two clauses are of the same form if some clause subsumes both of them.

The important property to have is that, if any set of clauses of some form can derive a new clause of one of that forms. A scheme is derived which subsumes all such resolutions.

4.2 Implications

When given a candidate the important property to ensure is that if it can be resolved indefinitely, then a scheme which subsumes all of these clauses is derived.

First mark all leaf clauses of the same form as the conclusion as suppositions.

The following properties should be vetted.

- There is at least one supposition
- The conclusion can larger than all of the hypothesis
- The conclusion can be a non-tautology
- The conclusion has at least as many predicates as the hypotheses

Once these properties are verified find the most specific form of the suppositions and conclusion.

This can be found by a process much like unification except when two clauses are not unifiable instead of failing replace them with a variable.

Next resolve clauses in the tree, with all the suppositions replaced by the most specific form. The resulting clauses are the conclusions and the suppositions are the implications.

5 Subsumption

5.1 Clause to Scheme

A clause should subsume an implication if all but finitely many things which it can derive are subsumed by the clause. In order to check this if the conclusion and the A clause subsumes an implication if it subsumes the conclusion. Clause to Scheme is not necessary if the implications can be subsumed and the schemes are rebuilt each round.

5.2 Scheme to Clause

A scheme subsumes a clause if it contains any clauses which subsume that clause. To determine this: First check if the target is a base clause. For any conclusion of an implication which subsumes the target apply the unifier of that subsumption to each of the hypotheses of the implication and treat these as the new targets if all of theme are subsumed this is successful. When doing this maintain a state of the list of previous targets on each recursive call and fail to find a target if it is in this list this should prevent some infinite loops without causing unnecessary failure.

5.3 Scheme to Scheme

A scheme subsumes a scheme if all clauses derived by the right scheme are subsumed by a clause derived by the left scheme To determine this: This is equivalent to the right base clause being subsumed by the left scheme so this step can be skipped as long as the schemes are rebuilt every layer.

6 Resolution

6.1 Scheme with Clause

To resolve a scheme against a clause it is necessary to create a list of clauses and implications containing every clause which could be derived by resolving any clause in the scheme with the clause.

To determine this: For each implication in the scheme if the conclusion resolves with the clause the result of this resolution is the new conclusion to get the new hypotheses apply the unifier to each of the hypotheses also resolve the clause against each of the base clauses.

6.2 Scheme with Scheme

To resolve a scheme against a scheme it is necessary to create a list of clauses and schemes containing every clause which could be derived by resolving any clause in one scheme with a clause in the other. In order to do this, for each pair of conclusions (one left one right) that resolve, find the new implications, and the new clauses.

6.2.1 New implications

To find the new implications: For both schemes create a list of predicate-level functions corresponding to how each implication changes the predicate used in the resolution. The predicate-level functions should be stored with the clause-level function that causes that predicate-level change.

Next begin a stateful recursive chase. The state is the previously encountered problems and the list of clause and predicate level functions have been applied. When you encounter a problem you've seen before create an implication from the original resolution to the resolution after applying all the clause level functions

6.2.2 New Clauses

To find a generating base of resolutions unify each base clause of one schema with the other schema. This works because the hypotheses of each implication subsumes the conclusion so if there is a failure to unify nothing derived by it will unify either. If there is a successful unification then the search can stop because any further unifications would be absorbed by the implications.

7 Finding Empty Clauses

As no implication decreases the number of literals in a clause schema only contain the empty clause if it is a base clause. Thus it is sufficient to check clauses for the empty clause.

8 Tautology Removal

I don't know a way to ensure tautology removal in schema. I don't think it is necessary for correctness.