Automated Deduction

Laura Kovács

Outline

Subsumption and Tautology Deletion

A clause is a propositional tautology if it is of the form $p \vee \neg p \vee C$, that is, it contains a pair of complementary literals.
There are also equational tautologies, for example
$a \neq b \vee b \neq c \vee f(c, c) = f(a, a)$.

Subsumption and Tautology Deletion

A clause is a propositional tautology if it is of the form $p \vee \neg p \vee C$, that is, it contains a pair of complementary literals.
There are also equational tautologies, for example
$a \neq b \vee b \neq c \vee f(c, c) = f(a, a)$.

A clause C subsumes any clause $C \vee D$, where D is non-empty.

Subsumption and Tautology Deletion

A clause is a propositional tautology if it is of the form $p \vee \neg p \vee C$, that is, it contains a pair of complementary literals.
There are also equational tautologies, for example
$a \neq b \vee b \neq c \vee f(c, c) = f(a, a)$.

A clause C subsumes any clause $C \vee D$, where D is non-empty.

It was known since 1965 that subsumed clauses and propositional tautologies can be removed from the search space.

Problem

How can we prove that completeness is preserved if we remove subsumed clauses and tautologies from the search space?

Problem

How can we prove that completeness is preserved if we remove subsumed clauses and tautologies from the search space?

Solution: general theory of redundancy.

Bag Extension of an Ordering

Bag = finite multiset.
Let $>$ be any (strict) ordering on a set X. The bag extension of $>$ is a binary relation $>^{bag}$, on bags over X, defined as the smallest transitive relation on bags such that

$$\{x, y_1, \ldots, y_n\} >^{bag} \{x_1, \ldots, x_m, y_1, \ldots, y_n\}$$
$$\text{if } x > x_i \text{ for all } i \in \{1 \ldots m\},$$

where $m \geq 0$.

Bag Extension of an Ordering

Bag = finite multiset.
Let $>$ be any (strict) ordering on a set X. The bag extension of $>$ is a binary relation $>^{bag}$, on bags over X, defined as the smallest transitive relation on bags such that

$$\{x, y_1, \ldots, y_n\} >^{bag} \{x_1, \ldots, x_m, y_1, \ldots, y_n\}$$
$$\text{if } x > x_i \text{ for all } i \in \{1 \ldots m\},$$

where $m \geq 0$.
Idea: a bag becomes smaller if we replace an element by any finite number of smaller elements.

Bag Extension of an Ordering

Bag = finite multiset.
Let $>$ be any (strict) ordering on a set X. The bag extension of $>$ is a binary relation $>^{bag}$, on bags over X, defined as the smallest transitive relation on bags such that

$$\{x, y_1, \ldots, y_n\} >^{bag} \{x_1, \ldots, x_m, y_1, \ldots, y_n\}$$
$$\text{if } x > x_i \text{ for all } i \in \{1 \ldots m\},$$

where $m \geq 0$.
Idea: a bag becomes smaller if we replace an element by any finite number of smaller elements.
The following results are known about the bag extensions of orderings:

1. $>^{bag}$ is an ordering;
2. If $>$ is total, then so is $>^{bag}$;
3. If $>$ is well-founded, then so is $>^{bag}$.

Clause Orderings

From now on consider clauses also as bags of literals. Note:

- we have an ordering $\succ$ for comparing literals;
- a clause is a bag of literals.

Clause Orderings

From now on consider clauses also as bags of literals. Note:

- we have an ordering $\succ$ for comparing literals;
- a clause is a bag of literals.

Hence

- we can compare clauses using the bag extension $\succ^{bag}$ of $\succ$.

Clause Orderings

From now on consider clauses also as bags of literals. Note:

- we have an ordering $\succ$ for comparing literals;
- a clause is a bag of literals.

Hence

- we can compare clauses using the bag extension $\succ^{bag}$ of $\succ$.

For simpicity we denote the multiset ordering also by $\succ$.

Example

Let $\succ$ be a total well-founded ordering on the ground atoms $p_1, \ldots, p_6$ such that $p_6 \succ p_5 \succ p_4 \succ p_3 \succ p_2 \succ p_1$. Consider the bag extension of $\succ$; for simplicity, denote the bag extension of $\succ$ also by $\succ$.

Using $\succ$, compare and order the following three clauses:

$$p_6 \vee \neg p_6\,, \qquad \neg p_2 \vee p_4 \vee p_5\,, \qquad p_2 \vee p_3\,.$$

Redundancy

A clause $C \in S$ is called redundant in S if it is a logical consequence of clauses in S strictly smaller than C.

Examples

A tautology $p \vee \neg p \vee C$ is a logical consequence of the empty set of formulas:

$$\models p \vee \neg p \vee C,$$

therefore it is redundant.

Examples

A tautology $p \lor \neg p \lor C$ is a logical consequence of the empty set of formulas:

$$\models p \lor \neg p \lor C,$$

therefore it is redundant.
We know that C subsumes $C \lor D$. Note

$$C \lor D \succ C$$
$$C \models C \lor D$$

therefore subsumed clauses are redundant.

Examples

A tautology $p \vee \neg p \vee C$ is a logical consequence of the empty set of formulas:

$$\models p \vee \neg p \vee C,$$

therefore it is redundant.
We know that C subsumes $C \vee D$. Note

$$C \vee D \succ C$$
$$C \models C \vee D$$

therefore subsumed clauses are redundant.

If $\Box \in S$, then all non-empty other clauses in S are redundant.

Redundant Clauses Can be Removed

In $\mathbb{BR}_\sigma$ (and in all calculi we will consider later) redundant clauses can be removed from the search space.

Redundant Clauses Can be Removed

In $\mathbb{BR}_\sigma$ (and in all calculi we will consider later) redundant clauses can be removed from the search space.

Inference Process with Redundancy

Let $\mathbb{I}$ be an inference system. Consider an inference process with two kinds of step $S_i \Rightarrow S_{i+1}$:

1. Adding the conclusion of an $\mathbb{I}$-inference with premises in S_i.
2. Deletion of a clause redundant in S_i, that is

$$S_{i+1} = S_i - \{C\},$$

where C is redundant in S_i.

Fairness: Persistent Clauses and Limit

Consider an inference process

$$S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$$

A clause C is called persistent if

$$\exists i \forall j \geq i (C \in S_j).$$

The limit S_∞ of the inference process is the set of all persistent clauses:

$$S_\infty = \bigcup_{i=0,1,\ldots} \bigcap_{j \geq i} S_j.$$

Fairness

The process is called $\mathbb{I}$-fair if every inference with persistent premises in S_∞ has been applied, that is, if

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

is an inference in $\mathbb{I}$ and $\{C_1, \ldots, C_n\} \subseteq S_\infty$, then $C \in S_i$ for some i.

Completeness of $\mathbb{BR}_\sigma$

Completeness Theorem. Let $\succ$ be a well-founded ordering and σ a well-behaved selection function. Let also

1. S_0 be a set of clauses;
2. $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ be a fair $\mathbb{BR}_\sigma$-inference process.

Then S_0 is unsatisfiable if and only if $\square \in S_i$ for some i.

Saturation up to Redundancy

A set S of clauses is called saturated up to redundancy if for every $\mathbb{I}$-inference

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

with premises in S, either

1. $C \in S$; or
2. C is redundant w.r.t. S, that is, $S_{\prec C} \models C$.

Saturation up to Redundancy and Satisfiability Checking

Lemma. A set S of clauses saturated up to redundancy is unsatisfiable if and only if $\square \in S$.

Saturation up to Redundancy and Satisfiability Checking

Lemma. A set S of clauses saturated up to redundancy is unsatisfiable if and only if $\square \in S$.

Therefore, if we built a set saturated up to redundancy, then the initial set S_0 is satisfiable. This is a powerful way of checking redundancy: one can even check satisfiability of formulas having only infinite models.

Saturation up to Redundancy and Satisfiability Checking

Lemma. A set S of clauses saturated up to redundancy is unsatisfiable if and only if $\Box \in S$.

Therefore, if we built a set saturated up to redundancy, then the initial set S_0 is satisfiable. This is a powerful way of checking redundancy: one can even check satisfiability of formulas having only infinite models.

The only problem with this characterisation is that there is no obvious way to build a model of S_0 out of a saturated set.

Binary Resolution with Selection

One of the key properties to satisfy this lemma is the following: the conclusion of every rule is strictly smaller that the rightmost premise of this rule.

- Binary resolution,

$$\frac{\underline{p} \vee C_1 \quad \underline{\neg p} \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

- Positive factoring,

$$\frac{\underline{p} \vee \underline{p} \vee C}{p \vee C} \text{ (Fact).}$$

Outline

Redundancy Elimination

Equality

First-order logic with equality

- Equality predicate: $=$.
- Equality: $l = r$.

The order of literals in equalities does not matter, that is, we consider an equality $l = r$ as a multiset consisting of two terms l, r, and so consider $l = r$ and $r = l$ equal.

Equality. An Axiomatisation (Recap)

- reflexivity axiom: $x = x$;
- symmetry axiom: $x = y \rightarrow y = x$;
- transitivity axiom: $x = y \wedge y = z \rightarrow x = z$;
- function substitution (congruence) axioms: $x_1 = y_1 \wedge \ldots \wedge x_n = y_n \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$, for every function symbol f;
- predicate substitution (congruence) axioms: $x_1 = y_1 \wedge \ldots \wedge x_n = y_n \wedge P(x_1, \ldots, x_n) \rightarrow P(y_1, \ldots, y_n)$ for every predicate symbol P.

Inference systems for logic with equality

We will define a resolution and superposition inference system. This system is complete. One can eliminate redundancy.

Inference systems for logic with equality

We will define a resolution and superposition inference system. This system is complete. One can eliminate redundancy.

We will first define it only for ground clauses. On the theoretical side,

- Completeness is first proved for ground clauses only.
- It is then "lifted" to arbitrary first-order clauses using a technique called lifting.
- Moreover, this way some notions (ordering, selection function) can first be defined for ground clauses only and then it is relatively easy to see how to generalise them for non-ground clauses.

Simple Ground Superposition Inference System

Superposition: (right and left)

$$\frac{l = r \vee C \quad s[l] = t \vee D}{s[r] = t \vee C \vee D}\ \text{(Sup)}, \quad \frac{l = r \vee C \quad s[l] \neq t \vee D}{s[r] \neq t \vee C \vee D}\ \text{(Sup)},$$

Simple Ground Superposition Inference System

Superposition: (right and left)

$$\frac{l = r \vee C \quad s[l] = t \vee D}{s[r] = t \vee C \vee D}\ \text{(Sup)}, \quad \frac{l = r \vee C \quad s[l] \neq t \vee D}{s[r] \neq t \vee C \vee D}\ \text{(Sup)},$$

Equality Resolution:

$$\frac{s \neq s \vee C}{C}\ \text{(ER)},$$

Simple Ground Superposition Inference System

Superposition: (right and left)

$$\frac{l = r \lor C \quad s[l] = t \lor D}{s[r] = t \lor C \lor D}\ (\mathrm{Sup}), \quad \frac{l = r \lor C \quad s[l] \neq t \lor D}{s[r] \neq t \lor C \lor D}\ (\mathrm{Sup}),$$

Equality Resolution:

$$\frac{s \neq s \lor C}{C}\ (\mathrm{ER}),$$

Equality Factoring:

$$\frac{s = t \lor s = t' \lor C}{s = t \lor t \neq t' \lor C}\ (\mathrm{EF}),$$

Example

$$f(a) = a \vee g(a) = a$$
$$f(f(a)) = a \vee g(g(a)) \neq a$$
$$f(f(a)) \neq a$$

Can this system be used for efficient theorem proving?

Not really. It has too many inferences. For example, from the clause $f(a) = a$ we can derive any clause of the form

$$f^m(a) = f^n(a)$$

where $m, n \geq 0$.

Can this system be used for efficient theorem proving?

Not really. It has too many inferences. For example, from the clause $f(a) = a$ we can derive any clause of the form

$$f^m(a) = f^n(a)$$

where $m, n \geq 0$.
Worst of all, the derived clauses can be much larger than the original clause $f(a) = a$.

Can this system be used for efficient theorem proving?

Not really. It has too many inferences. For example, from the clause $f(a) = a$ we can derive any clause of the form

$$f^m(a) = f^n(a)$$

where $m, n \geq 0$.
Worst of all, the derived clauses can be much larger than the original clause $f(a) = a$.
The recipe is to use the previously introduced ingredients:

1. Ordering;
2. Literal selection;
3. Redundancy elimination.