First-Order Theorem Proving and Vampire

Laura Kovács

Outline

Inference Systems

Selection Functions

Saturation Algorithms

Redundancy

Inference System

- inference has the form

$$\frac{F_1 \quad \ldots \quad F_n}{G},$$

  where $n \geq 0$ and $F_1, \ldots, F_n, G$ are formulas.
- The formula G is called the conclusion of the inference;
- The formulas $F_1, \ldots, F_n$ are called its premises.
- An Inference system $\mathbb{I}$ is a set of inference rules.
- Axiom: inference rule with no premises.

Inference System

- inference has the form

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

  where $n \geq 0$ and $F_1, \dots, F_n, G$ are formulas.
- The formula G is called the conclusion of the inference;
- The formulas $F_1, \dots, F_n$ are called its premises.
- An Inference system $\mathbb{I}$ is a set of inference rules.
- Axiom: inference rule with no premises.

Derivation, Proof

- Derivation in an inference system $\mathbb{I}$: a tree built from inferences in $\mathbb{I}$.
- If the root of this derivation is E, then we say it is a derivation of E.
- Proof of E: a finite derivation whose leaves are axioms.

Arbitrary First-Order Formulas

- A first-order signature (vocabulary): function symbols (including constants), predicate symbols. Equality is part of the language.
- A set of variables.
- Terms are buit using variables and function symbols. For example, $f(x) + g(x)$.
- Atoms, or atomic formulas are obtained by applying a predicate symbol to a sequence of terms. For example, $p(a, x)$ or $f(x) + g(x) \geq 2$.
- Formulas: built from atoms using logical connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ and quantifiers $\forall$, $\exists$. For example, $(\forall x)x = 0 \vee (\exists y)y > x$.

Clauses

- Literal: either an atom A or its negation $\neg A$.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$ of literals, where $n \geq 0$.

Clauses

- Literal: either an atom A or its negation $\neg A$.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$ of literals, where $n \geq 0$.
- Empty clause, denoted by $\square$: clause with 0 literals, that is, when $n = 0$.

Clauses

- Literal: either an atom A or its negation $\neg A$.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$ of literals, where $n \geq 0$.
- Empty clause, denoted by $\square$: clause with 0 literals, that is, when $n = 0$.
- A formula in Clausal Normal Form (CNF): a conjunction of clauses.

Clauses

- Literal: either an atom A or its negation $\neg A$.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$ of literals, where $n \geq 0$.
- Empty clause, denoted by $\square$: clause with 0 literals, that is, when $n = 0$.
- A formula in Clausal Normal Form (CNF): a conjunction of clauses.
- From now onwards: We only consider clauses.

Clauses

- Literal: either an atom A or its negation $\neg A$.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$ of literals, where $n \geq 0$.
- Empty clause, denoted by $\square$: clause with 0 literals, that is, when $n = 0$.
- A formula in Clausal Normal Form (CNF): a conjunction of clauses.
- From now onwards: We only consider clauses.
- A clause is ground if it contains no variables.
- If a clause contains variables, we assume that it implicitly universally quantified. That is, we treat $p(x) \vee q(x)$ as $\forall x(p(x) \vee q(x))$.

Binary Resolution Inference System

The binary resolution inference system, denoted by $\mathbb{BR}$ is an inference system on propositional clauses (or ground clauses). It consists of two inference rules:

- Binary resolution, denoted by BR:

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

- Factoring, denoted by Fact:

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

Soundness

- An inference is sound if the conclusion of this inference is a logical consequence of its premises.
- An inference system is sound if every inference rule in this system is sound.

Soundness

- An inference is sound if the conclusion of this inference is a logical consequence of its premises.
- An inference system is sound if every inference rule in this system is sound.

$\mathbb{BR}$ is sound.

Soundness

- An inference is sound if the conclusion of this inference is a logical consequence of its premises.
- An inference system is sound if every inference rule in this system is sound.

$\mathbb{BR}$ is sound.

Consequence of soundness: let S be a set of clauses. If $\square$ can be derived from S in $\mathbb{BR}$, then S is unsatisfiable.

Example

Consider the following set of clauses

$$\{\neg p \vee \neg q,\ \neg p \vee q,\ p \vee \neg q,\ p \vee q\}.$$

The following derivation derives the empty clause from this set:

$$\dfrac{\dfrac{\dfrac{p \vee q \quad p \vee \neg q}{p \vee p}\ \text{(BR)}}{p}\ \text{(Fact)} \qquad \dfrac{\dfrac{\neg p \vee q \quad \neg p \vee \neg q}{\neg p \vee \neg p}\ \text{(BR)}}{\neg p}\ \text{(Fact)}}{\Box}\ \text{(BR)}$$

Hence, this set of clauses is unsatisfiable.

Can this be used for checking (un)satisfiability

1. What happens when $\square$ cannot be derived from S?
2. How can one search for possible derivations of $\square$?

Can this be used for checking (un)satisfiability

1. Completeness.
   Let S be an unsatisfiable set of clauses. Then there exists a derivation of $\square$ from S in $\mathbb{BR}$.

Can this be used for checking (un)satisfiability

1. Completeness.
   Let S be an unsatisfiable set of clauses. Then there exists a derivation of $\square$ from S in $\mathbb{BR}$.

2. We have to formalize search for derivations.

However, before doing this we will introduce a slightly more refined inference system.

Outline

Selection Function

A literal selection function selects literals in a clause.

- If C is non-empty, then at least one literal is selected in C.

Selection Function

A literal selection function selects literals in a clause.

- If C is non-empty, then at least one literal is selected in C.

We denote selected literals by underlining them, e.g.,

$$\underline{p} \vee \neg q$$

Selection Function

A literal selection function selects literals in a clause.

- If C is non-empty, then at least one literal is selected in C.

We denote selected literals by <u>underlining them</u>, e.g.,

$$\underline{p} \vee \neg q$$

Note: selection function does not have to be a function. It can be any oracle that selects literals.

Binary Resolution with Selection

We introduce a family of inference systems, parametrised by a literal selection function σ.

The binary resolution inference system, denoted by $\mathbb{BR}_\sigma$, consists of two inference rules:

- Binary resolution, denoted by BR

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

Binary Resolution with Selection

We introduce a family of inference systems, parametrised by a literal selection function σ.
The binary resolution inference system, denoted by $\mathbb{BR}_\sigma$, consists of two inference rules:

- Binary resolution, denoted by BR

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

- Positive factoring, denoted by Fact:

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

Completeness?

Binary resolution with selection may be incomplete.

Completeness?

Binary resolution with selection may be incomplete.

Consider this set of clauses:

$$\begin{array}{ll}
(1) & \neg q \vee r \\
(2) & \neg p \vee q \\
(3) & \neg r \vee \neg q \\
(4) & \neg q \vee \neg p \\
(5) & \neg p \vee \neg r \\
(6) & \neg r \vee p \\
(7) & r \vee q \vee p
\end{array}$$

Completeness?

Binary resolution with selection may be incomplete.

Consider this set of clauses:

$$\begin{array}{ll}
(1) & \neg q \lor r \\
(2) & \neg p \lor q \\
(3) & \neg r \lor \neg q \\
(4) & \neg q \lor \neg p \\
(5) & \neg p \lor \neg r \\
(6) & \neg r \lor p \\
(7) & r \lor q \lor p
\end{array}$$

It is unsatisfiable:

$$\begin{array}{lll}
(8) & q \lor p & (6,7) \\
(9) & q & (2,8) \\
(10) & r & (1,9) \\
(11) & \neg q & (3,10) \\
(12) & \square & (9,11)
\end{array}$$

Completeness?

Binary resolution with selection may be incomplete.

Consider this set of clauses:

$$
\begin{array}{ll}
(1) & \neg q \vee \underline{r} \\
(2) & \neg p \vee \underline{q} \\
(3) & \neg r \vee \underline{\neg q} \\
(4) & \neg q \vee \underline{\neg p} \\
(5) & \neg p \vee \underline{\neg r} \\
(6) & \neg r \vee \underline{p} \\
(7) & r \vee q \vee \underline{p}
\end{array}
$$

It is unsatisfiable:

$$
\begin{array}{lll}
(8) & q \vee p & (6,7) \\
(9) & q & (2,8) \\
(10) & r & (1,9) \\
(11) & \neg q & (3,10) \\
(12) & \square & (9,11)
\end{array}
$$

However, any inference with selection applied to set of clauses $(1) - (7)$ give either a clause in this set, or a clause containing a clause in this set.

Completeness?

Binary resolution with selection may be incomplete.

Consider this set of clauses:

$$\begin{array}{ll}
(1) & \neg q \lor \underline{r} \\
(2) & \neg p \lor \underline{q} \\
(3) & \neg r \lor \underline{\neg q} \\
(4) & \neg q \lor \underline{\neg p} \\
(5) & \neg p \lor \underline{\neg r} \\
(6) & \neg r \lor \underline{p} \\
(7) & r \lor q \lor \underline{p}
\end{array}$$

It is unsatisfiable:

$$\begin{array}{lll}
(8) & q \lor p & (6,7) \\
(9) & q & (2,8) \\
(10) & r & (1,9) \\
(11) & \neg q & (3,10) \\
(12) & \square & (9,11)
\end{array}$$

However, any inference with selection applied to set of clauses (1) – (7) give either a clause in this set, or a clause containing a clause in this set.

For example, (8) cannot be derived from (1) – (7) with selection.

Literal Orderings

Take any well-founded ordering $\succ$ on atoms, that is, an ordering such that there is no infinite decreasing chain of atoms:

$$A_0 \succ A_1 \succ A_2 \succ \cdots$$

In the sequel $\succ$ will always denote a well-founded ordering.

Literal Orderings

Take any well-founded ordering $\succ$ on atoms, that is, an ordering such that there is no infinite decreasing chain of atoms:

$$A_0 \succ A_1 \succ A_2 \succ \cdots$$

In the sequel $\succ$ will always denote a well-founded ordering.

Extend it to an ordering on literals by:

- If $p \succ q$, then $p \succ \neg q$ and $\neg p \succ q$;
- $\neg p \succ p$.

Literal Orderings

Take any well-founded ordering $\succ$ on atoms, that is, an ordering such that there is no infinite decreasing chain of atoms:

$$A_0 \succ A_1 \succ A_2 \succ \cdots$$

In the sequel $\succ$ will always denote a well-founded ordering.

Extend it to an ordering on literals by:

- If $p \succ q$, then $p \succ \neg q$ and $\neg p \succ q$;
- $\neg p \succ p$.

Exercise: prove that the induced ordering on literals is well-founded too.

Orderings and Well-Behaved Selections

Fix an ordering $\succ$. A literal selection function is well-behaved if

- either a negative literal is selected,
  or all maximal literals (w.r.t. $\succ$) must be selected in C.

Orderings and Well-Behaved Selections

Fix an ordering $\succ$. A literal selection function is well-behaved if

- either a negative literal is selected,
  or all maximal literals (w.r.t. $\succ$) must be selected in C.

To be well-behaved, we sometimes must select more than one different literal in a clause. Example: $p \vee p$ or $p(x) \vee p(y)$.

Completeness of Binary Resolution with Selection

Binary resolution with selection is complete for every well-behaved selection function.

Completeness of Binary Resolution with Selection

Binary resolution with selection is complete for every well-behaved selection function.

Consider our previous example:

$$\begin{array}{ll}
(1) & \neg q \vee \underline{r} \\
(2) & \neg p \vee \underline{q} \\
(3) & \neg r \vee \underline{\neg q} \\
(4) & \neg q \vee \underline{\neg p} \\
(5) & \neg p \vee \underline{\neg r} \\
(6) & \neg r \vee \underline{p} \\
(7) & r \vee q \vee \underline{p}
\end{array}$$

A well-behave selection function must satisfy:

1. $r \succ q$, because of (1)
2. $q \succ p$, because of (2)
3. $p \succ r$, because of (6)

There is no ordering that satisfies these conditions.

Outline

How to Establish Unsatisfiability?

Completeness is formulated in terms of derivability of the empty clause $\square$ from a set S_0 of clauses in an inference system $\mathbb{I}$. However, this formulations gives no hint on how to search for such a derivation.

How to Establish Unsatisfiability?

Completeness is formulated in terms of derivability of the empty clause $\square$ from a set S_0 of clauses in an inference system $\mathbb{I}$. However, this formulations gives no hint on how to search for such a derivation.

Idea:

- Take a set of clauses S (the search space), initially $S = S_0$.

  Repeatedly apply inferences in $\mathbb{I}$ to clauses in S and add their conclusions to S, unless these conclusions are already in S.

  .

- If, at any stage, we obtain $\square$, we terminate and report unsatisfiability of S_0.

How to Establish Unsatisfiability?

Completeness is formulated in terms of derivability of the empty clause $\square$ from a set S_0 of clauses in an inference system $\mathbb{I}$. However, this formulations gives no hint on how to search for such a derivation.

Idea:

- Take a set of clauses S (the search space), initially $S = S_0$.

  Repeatedly apply inferences in $\mathbb{I}$ to clauses in S and add their conclusions to S, unless these conclusions are already in S.
  If an inference in $\mathbb{I}$ can be applied, eventually it has to be applied (fairness).

- If, at any stage, we obtain $\square$, we terminate and report unsatisfiability of S_0.

How to Establish Satisfiability?

When can we report satisfiability?

How to Establish Satisfiability?

When can we report satisfiability?

When we build a set S such that any inference applied to clauses in S is already a member of S. Any such set of clauses is called saturated (with respect to $\mathbb{I}$).

How to Establish Satisfiability?

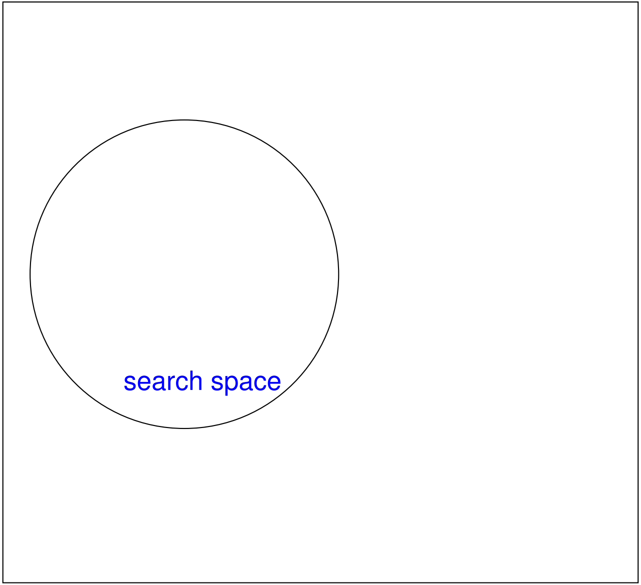
When can we report satisfiability?

When we build a set S such that any inference applied to clauses in S is already a member of S. Any such set of clauses is called saturated (with respect to $\mathbb{I}$).

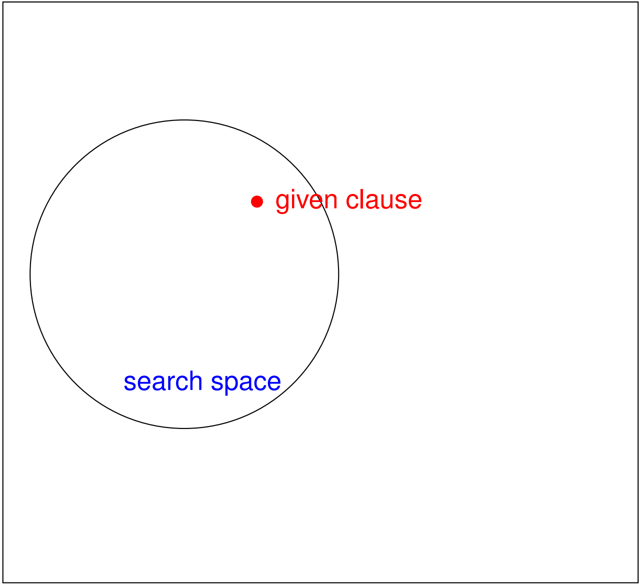
In first-order logic it is often the case that all saturated sets are infinite (due to undecidability), so in practice we can never build a saturated set.

The process of trying to build one is referred to as saturation.

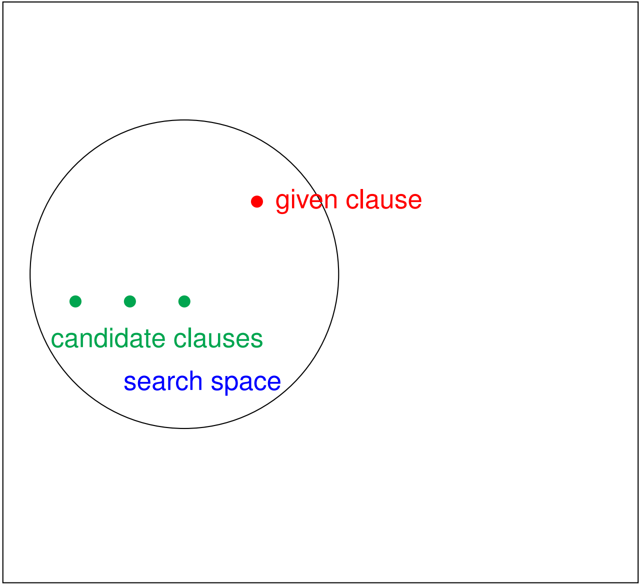
Saturation Algorithms: Inference Selection by Clause Selection



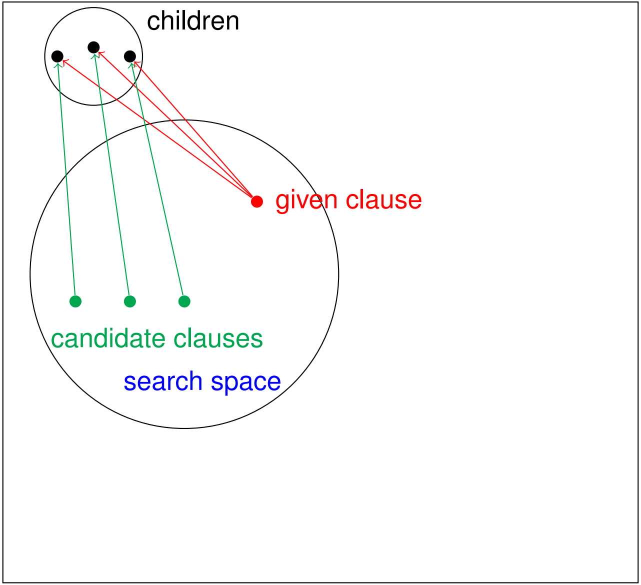
Saturation Algorithms: Inference Selection by Clause Selection



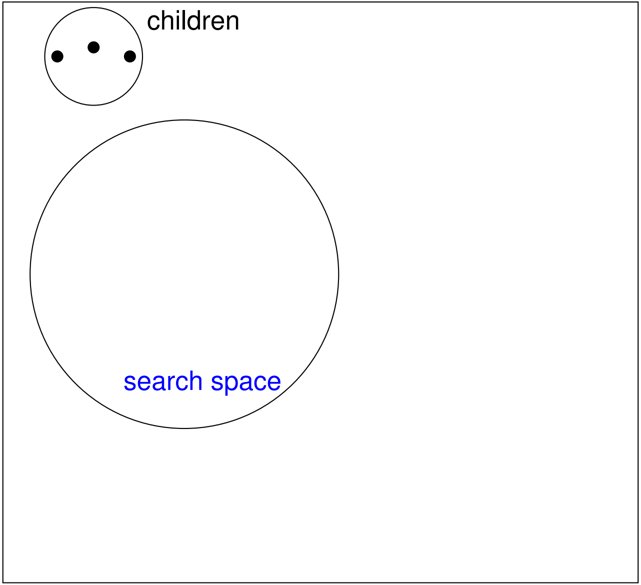
Saturation Algorithms: Inference Selection by Clause Selection



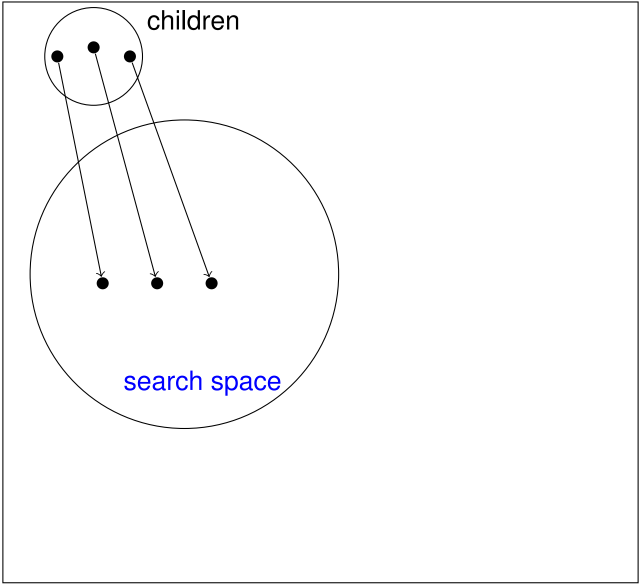
Saturation Algorithms: Inference Selection by Clause Selection



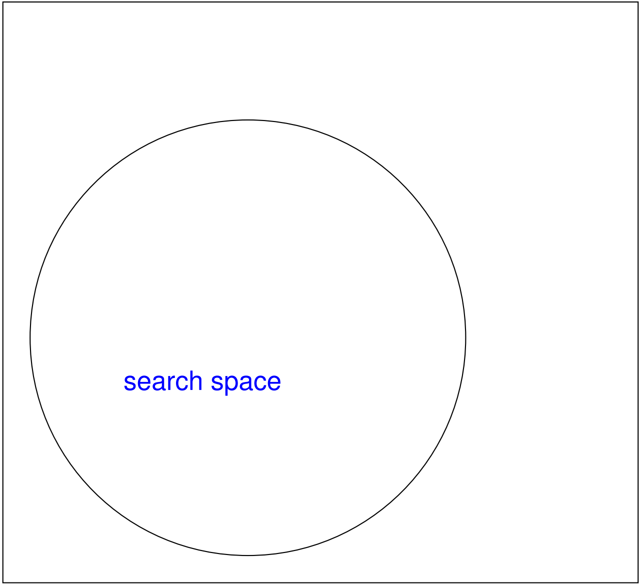
Saturation Algorithms: Inference Selection by Clause Selection



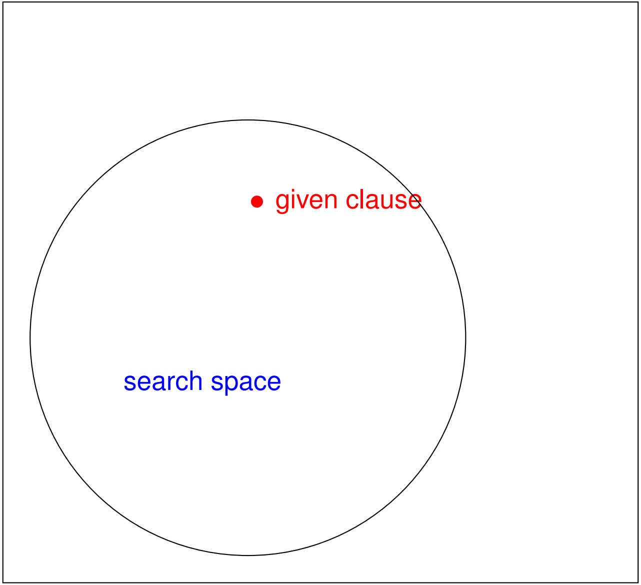
Saturation Algorithms: Inference Selection by Clause Selection



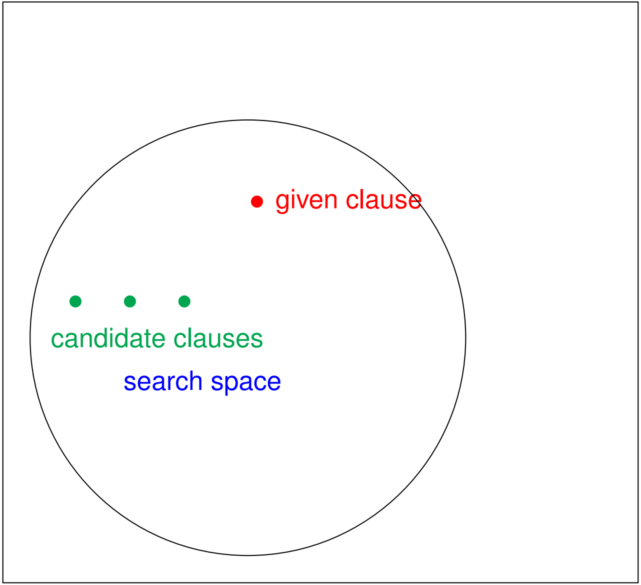
Saturation Algorithms: Inference Selection by Clause Selection



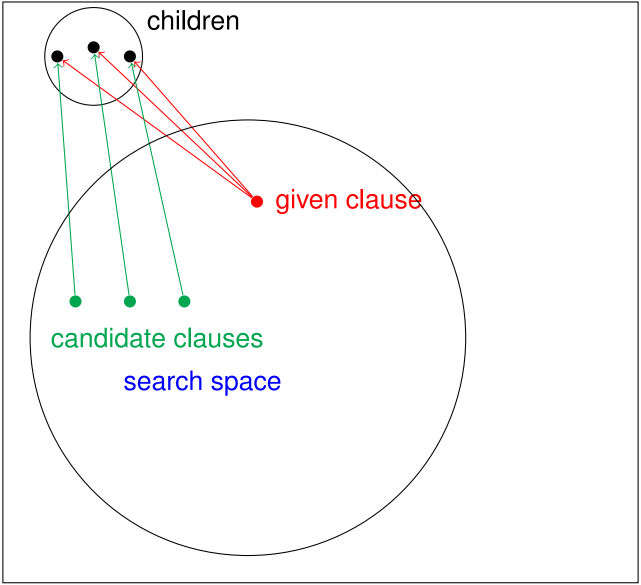
Saturation Algorithms: Inference Selection by Clause Selection



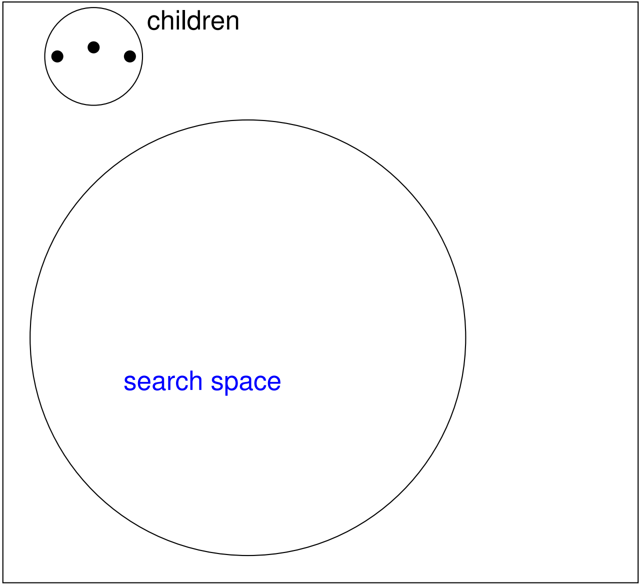
Saturation Algorithms: Inference Selection by Clause Selection



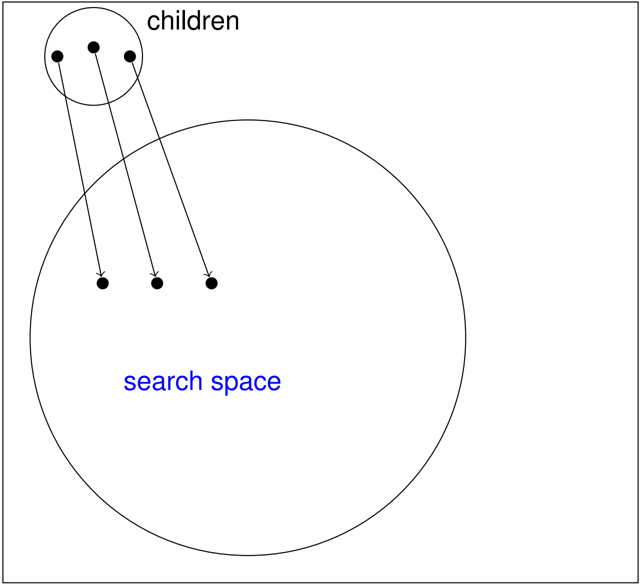
Saturation Algorithms: Inference Selection by Clause Selection



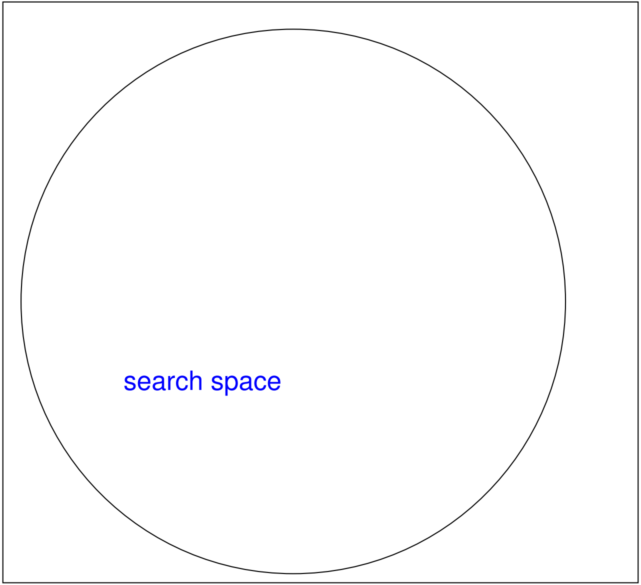
Saturation Algorithms: Inference Selection by Clause Selection



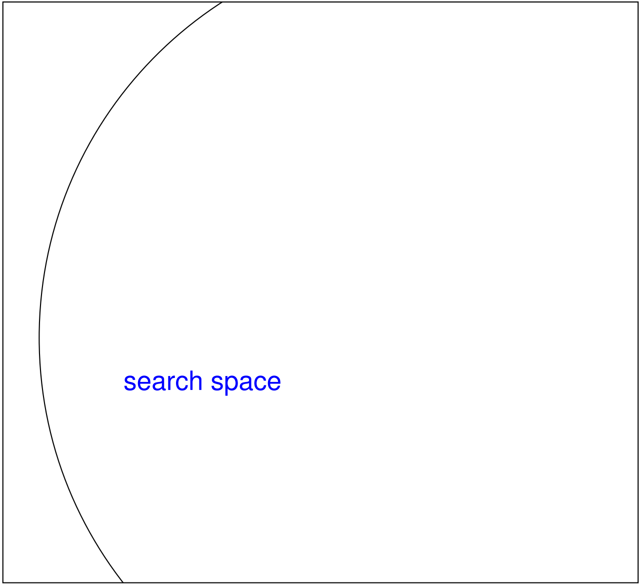
Saturation Algorithms: Inference Selection by Clause Selection



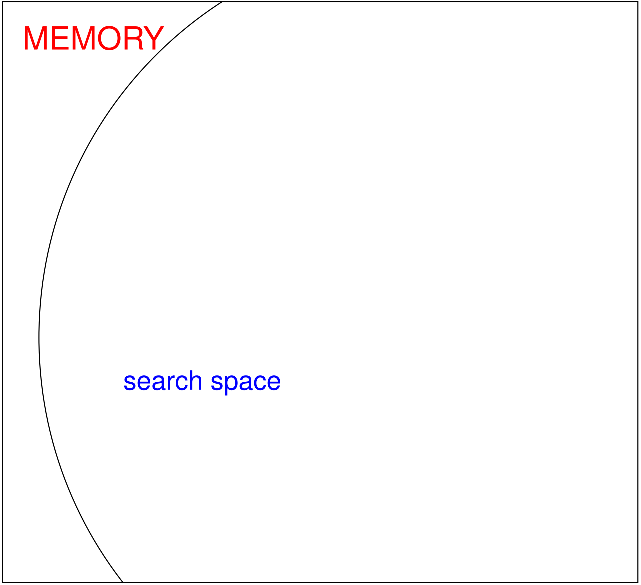
Saturation Algorithms: Inference Selection by Clause Selection



Saturation Algorithms: Inference Selection by Clause Selection



Saturation Algorithms: Inference Selection by Clause Selection



Saturation Algorithm

A saturation algorithm tries to saturate a set of clauses with respect to a given inference system.

In theory there are three possible scenarios:

1. At some moment the empty clause $\square$ is generated, in this case the input set of clauses is unsatisfiable.

2. Saturation will terminate without ever generating $\square$, in this case the input set of clauses in satisfiable.

3. Saturation will run forever, but without generating $\square$. In this case the input set of clauses is satisfiable.

Saturation Algorithm in Practice

In practice there are three possible scenarios:

1. At some moment the empty clause $\square$ is generated, in this case the input set of clauses is unsatisfiable.
2. Saturation will terminate without ever generating $\square$, in this case the input set of clauses in satisfiable.
3. Saturation will run until we run out of resources, but without generating $\square$. In this case it is unknown whether the input set is unsatisfiable.

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 $\succ$.

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 $p \vee \neg p \vee C$ 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.

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

$$\begin{array}{c} C \vee D \succ C \\ C \models C \vee D \end{array}$$

therefore subsumed clauses are redundant.

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

Redundant Clauses Can be Removed

In $\mathbb{BR}_\sigma$ (and in the superposition calculus considered later) redundant clauses can be removed from the search space.

Saturation with Redundancy

Let $\mathbb{I}$ be an inference system. Consider a saturation 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.

Saturation with Redundancy

Let $\mathbb{I}$ be an inference system. Consider a saturation 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.
   [generating inference]
2. Deletion of a clause redundant in S_i, that is

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

   where C is redundant in S_i. [simplifying inference]