Saturation-based Theorem Proving and ML

Course Machine Learning and Reasoning 2020

MLR 2020[1]

[1]Czech Technical Univeristy in Prague (CIIRC)

April 3, 2020

Outline

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 built 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.
- 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 $\forall x(p(x) \vee q(x))$.

What an Automatic Theorem Prover is Expected to Do

Input:

- a set of axioms (first order formulas) or clauses $\mathcal{A}$
- a conjecture (first-order formula or set of clauses) G

Question:

- Does G logically follow from $\mathcal{A}$?

$$\mathcal{A} \overset{?}{\models} G$$

What an Automatic Theorem Prover is Expected to Do

Input:

- a set of axioms (first order formulas) or clauses $\mathcal{A}$
- a conjecture (first-order formula or set of clauses) G

Question:

- Does G logically follow from $\mathcal{A}$?

$$\mathcal{A} \overset{?}{\models} G$$

Output:

- Either yes and a proof,
- or . . . ?

Proof by Refutation

Given a problem with axioms and assumptions $\mathcal{A} = F_1, \ldots, F_n$ and conjecture G,

1. negate the conjecture;
2. establish unsatisfiability of the set of formulas $F_1, \ldots, F_n, \neg G$.

Proof by Refutation

Given a problem with axioms and assumptions $\mathcal{A} = F_1, \ldots, F_n$ and conjecture G,

1. negate the conjecture;
2. establish unsatisfiability of the set of formulas $F_1, \ldots, F_n, \neg G$.

Thus, we reduce the theorem proving problem to the problem of checking unsatisfiability.

General Scheme in One Slide

- Read a problem P
- Preprocess the problem: $P \Longrightarrow P'$
- Convert P' into Clause Normal Form N
  - replacing connectives, formula naming, distributive laws
  - Skolemisation
- Run a saturation algorithm on it, try to derive $\square$.
  - computes a closure of N with respect to an inference system
  - logical calculus: resolution + superposition
- If $\square$ is derived, report the result, maybe including a refutation.

General Scheme in One Slide

- Read a problem P
- Preprocess the problem: $P \Longrightarrow P'$
- Convert P' into Clause Normal Form N
  - replacing connectives, formula naming, distributive laws
  - Skolemisation
- Run a saturation algorithm on it, try to derive $\square$.
  - computes a closure of N with respect to an inference system
  - logical calculus: resolution + superposition
- If $\square$ is derived, report the result, maybe including a refutation.

Trying to derive $\square$ using a saturation algorithm is the hardest part, which in practice may not terminate or run out of memory.

A Bit More on the CNF Transformation

- replacing unwanted connectives:

$$
\begin{array}{lcl}
A \leftrightarrow B & \Longrightarrow & (A \rightarrow B) \wedge (B \rightarrow A) \\
A \rightarrow B & \Longrightarrow & \neg A \vee B \\
\neg(A \vee B) & \Longrightarrow & \neg A \wedge \neg B \\
 & \ldots &
\end{array}
$$

A Bit More on the CNF Transformation

- replacing unwanted connectives:

$$
\begin{array}{lcl}
A \leftrightarrow B & \Longrightarrow & (A \rightarrow B) \wedge (B \rightarrow A) \\
A \rightarrow B & \Longrightarrow & \neg A \vee B \\
\neg(A \vee B) & \Longrightarrow & \neg A \wedge \neg B \\
 & \cdots &
\end{array}
$$

- distributive laws:

$$(A \wedge B) \vee (C \wedge D) \Longrightarrow (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

A Bit More on the CNF Transformation

- replacing unwanted connectives:

$$\begin{array}{lcl} A \leftrightarrow B & \Longrightarrow & (A \rightarrow B) \land (B \rightarrow A) \\ A \rightarrow B & \Longrightarrow & \neg A \lor B \\ \neg(A \lor B) & \Longrightarrow & \neg A \land \neg B \\ & \cdots & \end{array}$$

- distributive laws:

$$(A \land B) \lor (C \land D) \Longrightarrow (A \lor C) \land (A \lor D) \land (B \lor C) \land (B \lor D)$$

- formula naming (Tseitin / Pleisted-Greenbaum):

$$(A \land B) \lor (C \land D) \Longrightarrow (F_{AB} \lor (C \land D)) \land (F_{AB} \rightarrow A) \land (F_{AB} \rightarrow B)$$

A Bit More on the CNF Transformation

- replacing unwanted connectives:

$$\begin{array}{lcl} A \leftrightarrow B & \Longrightarrow & (A \to B) \land (B \to A) \\ A \to B & \Longrightarrow & \neg A \lor B \\ \neg(A \lor B) & \Longrightarrow & \neg A \land \neg B \\ & \cdots & \end{array}$$

- distributive laws:

$$(A \land B) \lor (C \land D) \Longrightarrow (A \lor C) \land (A \lor D) \land (B \lor C) \land (B \lor D)$$

- formula naming (Tseitin / Pleisted-Greenbaum):

$$(A \land B) \lor (C \land D) \Longrightarrow (F_{AB} \lor (C \land D)) \land (F_{AB} \to A) \land (F_{AB} \to B)$$

- Skolemisation on an example

$$\forall x[x \neq 0 \to \exists y(x \cdot y = 1)] \quad \Longrightarrow \quad x \neq 0 \to x \cdot sk_y(x) = 1$$

The Premise Selection Task

The set of clauses $F_1, \ldots, F_n, \neg G$ to be passed to the saturation may be too large to process efficiently

- common sense reasoning tasks (big ontologies)
- automatic support for interactive provers
  - e.g. Mizar, Isabelle, HOL, and Coq
  - large background libraries of already formalized math

Premise Selection:

- heuristically pick a subset $\mathcal{A}' \subset \mathcal{A} = F_1, \ldots, F_n$ such that $\mathcal{A}', \neg G$ is (likely) still unsatisfiable

Approaches to Premise Selection

"Traditional" – SInE:

- The SUMO Inference Engine
- signature based relatedness to the conjucture

Approaches to Premise Selection

"Traditional" – SInE:

- The SUMO Inference Engine
- signature based relatedness to the conjucture

Machine Learning approaches:

- Premise Selection for Mathematics by Corpus Analysis and Kernel Methods. J. Autom. Reasoning (2014)
- DeepMath - Deep Sequence Models for Premise Selection. NIPS 2016
- ATPboost: Learning Premise Selection in Binary Setting with ATP Feedback. IJCAR 2018

Approaches to Premise Selection

"Traditional" – SInE:

- The SUMO Inference Engine
- signature based relatedness to the conjucture

Machine Learning approaches:

- Premise Selection for Mathematics by Corpus Analysis and Kernel Methods. J. Autom. Reasoning (2014)
- DeepMath - Deep Sequence Models for Premise Selection. NIPS 2016
- ATPboost: Learning Premise Selection in Binary Setting with ATP Feedback. IJCAR 2018

Learning from previously discovered proofs

Outline

1. First-order Logic and Theorem Proving
2. Saturation-based Proving
3. Further Tuning and the Role of Strategies
4. Summary

Overview

Saturation-based proving

- the most prominent technology for proving in FOL
  - provers: E, Vampire, Spass, iProver, ...
- alternatives:
  - the tableaux approach: e.g. LeanCop
  - Satisfiability Modul Theories (SMT): Z3, CVC4, . . .

Overview

Saturation-based proving

- the most prominent technology for proving in FOL
  - provers: E, Vampire, Spass, iProver, ...
- alternatives:
  - the tableaux approach: e.g. LeanCop
  - Satisfiability Modul Theories (SMT): Z3, CVC4, . . .

Topics:

- A Static View: Inferences, Soundness, and Completeness
- A Dynamic View: The Saturation Loop
- Making It Fast in Practice

Inference System

- An 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 (clauses).
- The formula G is called the conclusion of the inference;
- The formulas $F_1, \dots, F_n$ are called its premises.

Inference System

- An 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 (clauses).

- The formula G is called the conclusion of the inference;
- The formulas $F_1, \dots, F_n$ are called its premises.

- An inference rule R is a set of inferences.
- Every inference $I \in R$ is called an instance of R.
- An Inference system / calculus $\mathbb{I}$ is a set of inference rules.

Derivation, Proof

- Derivation in an inference system $\mathbb{I}$:
  a DAG built from inferences in $\mathbb{I}$.
- Derivation of E from $E_1, \ldots, E_m$: a finite derivation of E whose every leaf is one of the expressions $E_1, \ldots, E_m$ and the root of which is is E.
- A refutation is a derivation of the empty clause $\square$.

The superposition calculus

Resolution **Factoring**

$$\frac{\underline{A} \vee C_1 \quad \underline{\neg A'} \vee C_2}{(C_1 \vee C_2)\theta}, \qquad \frac{\underline{A} \vee A' \vee C}{(A \vee C)\theta},$$

where, for both inferences, $\theta = \mathsf{mgu}(A, A')$ *and* A *is not an equality literal*

Superposition

$$\frac{\underline{l \simeq r} \vee C_1 \quad \underline{L[s]_p} \vee C_2}{(L[r]_p \vee C_1 \vee C_2)\theta} \quad \text{or} \quad \frac{\underline{l \simeq r} \vee C_1 \quad \underline{t[s]_p \otimes t'} \vee C_2}{(t[r]_p \otimes t' \vee C_1 \vee C_2)\theta},$$

where $\theta = \mathsf{mgu}(l, s)$ *and* $r\theta \not\succeq l\theta$ *and, for the left rule* $L[s]$ *is not an equality literal, and for the right rule* $\otimes$ *stands either for* $\simeq$ *or* $\not\simeq$ *and* $t'\theta \not\succeq t[s]\theta$

EqualityResolution **EqualityFactoring**

$$\frac{\underline{s \not\simeq t} \vee C}{C\theta}, \qquad \frac{\underline{s \simeq t} \vee s' \simeq t' \vee C}{(t \not\simeq t' \vee s' \simeq t' \vee C)\theta},$$

where $\theta = \mathsf{mgu}(s, t)$

where $\theta = \mathsf{mgu}(s, s')$, $t\theta \not\succeq s\theta$, *and* $t'\theta \not\succeq s'\theta$

Fig. 1. The rules of the superposition and resolution calculus.

Soundness and Completeness

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.

Consequence of soundness: Let S be a set of clauses. If $\square$ can be derived from S by a sound $\mathbb{I}$ then S is unsatisfiable.

Soundness and Completeness

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.

Consequence of soundness: Let S be a set of clauses. If $\square$ can be derived from S by a sound $\mathbb{I}$ then S is unsatisfiable.

1. What if the empty clause cannot be derived from S?
2. Can we systematically search for possible derivations of $\square$?

Soundness and Completeness

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.

Consequence of soundness: Let S be a set of clauses. If $\square$ can be derived from S by a sound $\mathbb{I}$ then S is unsatisfiable.

1. What if the empty clause cannot be derived from S?
2. Can we systematically search for possible derivations of $\square$?

Completeness

An inference system $\mathbb{I}$ is complete, if for every unsatisfiable set of clauses S, there is a derivation of $\square$ from S using $\mathbb{I}$.

Idea of Saturation

Completess 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 of Saturation

Completess 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.

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:

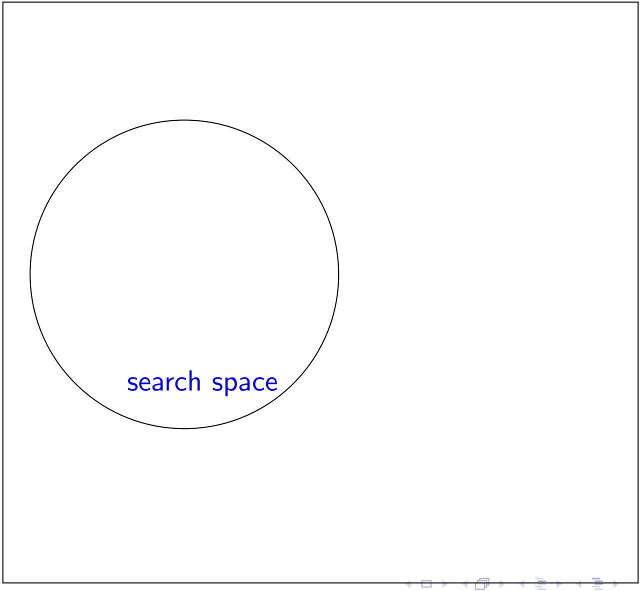
1. At some moment the empty clause $\square$ is generated, in this case the input set of clauses is unsatisfiable.
2. Saturation will terminate without ever generating $\square$, in this case the input set of clauses in satisfiable.
3. Saturation will run forever, but without generating $\square$. In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

Saturation Algorithm in Practice

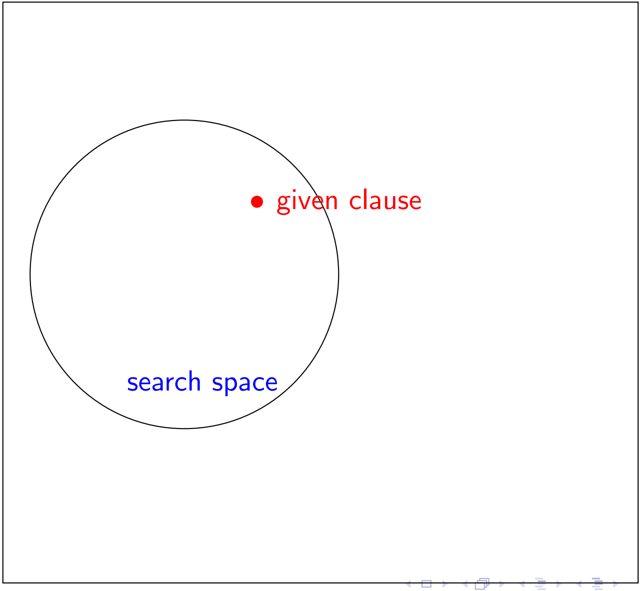
In practice there are three possible scenarios:

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.

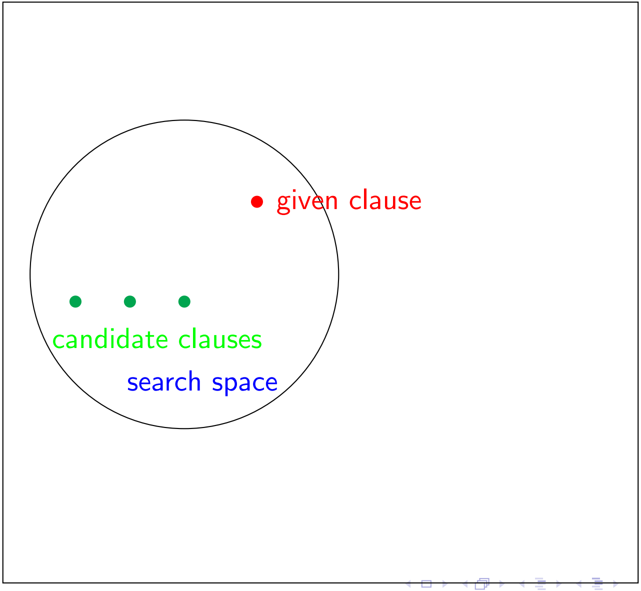
Inference Selection by Clause Selection



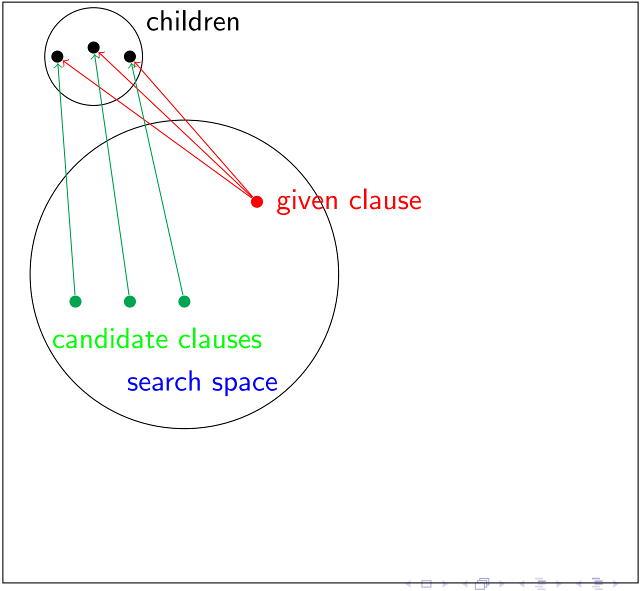
Inference Selection by Clause Selection



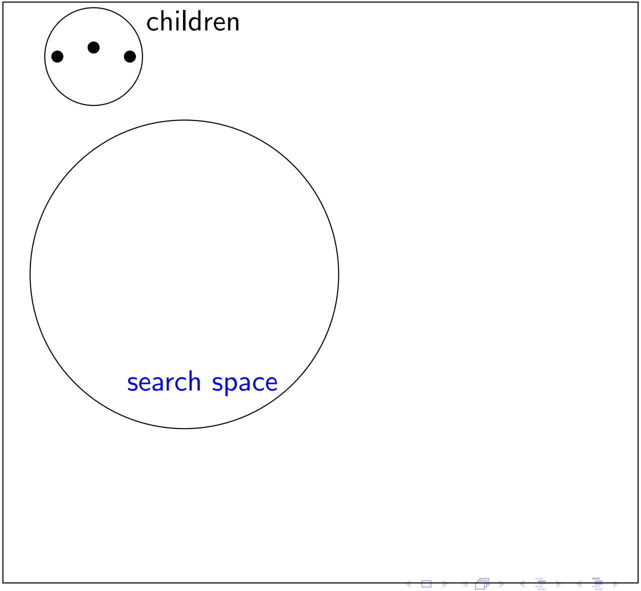
Inference Selection by Clause Selection



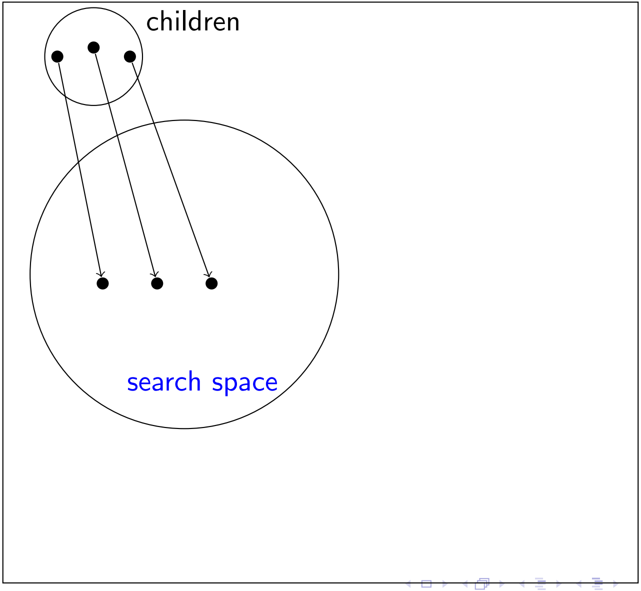
Inference Selection by Clause Selection



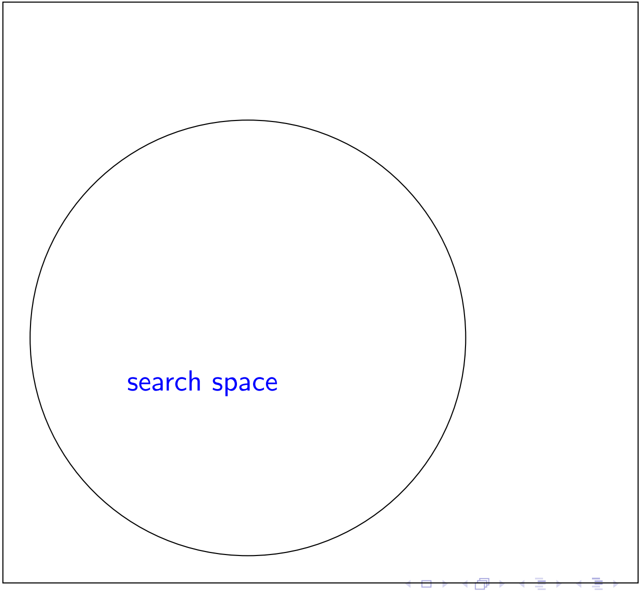
Inference Selection by Clause Selection



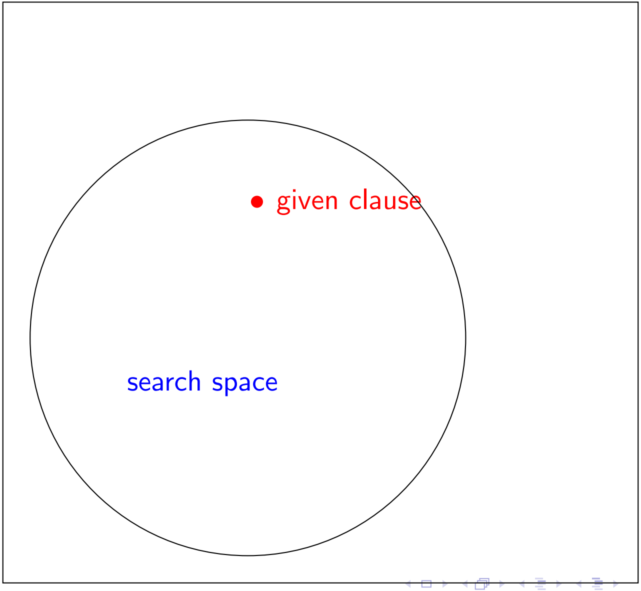
Inference Selection by Clause Selection



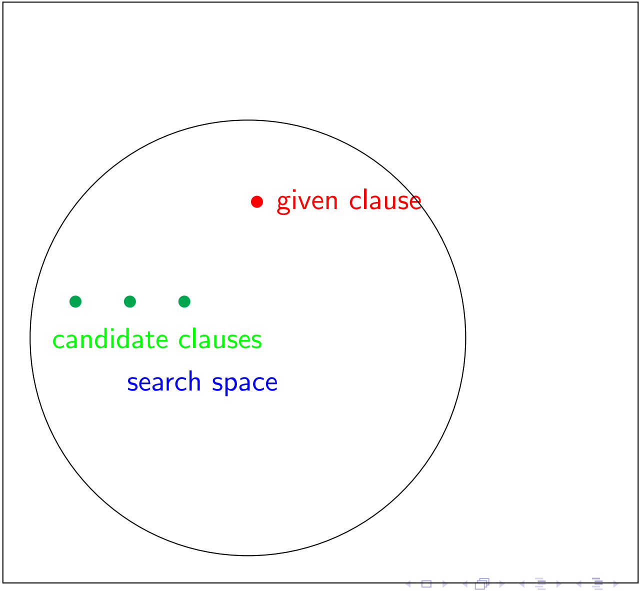
Inference Selection by Clause Selection



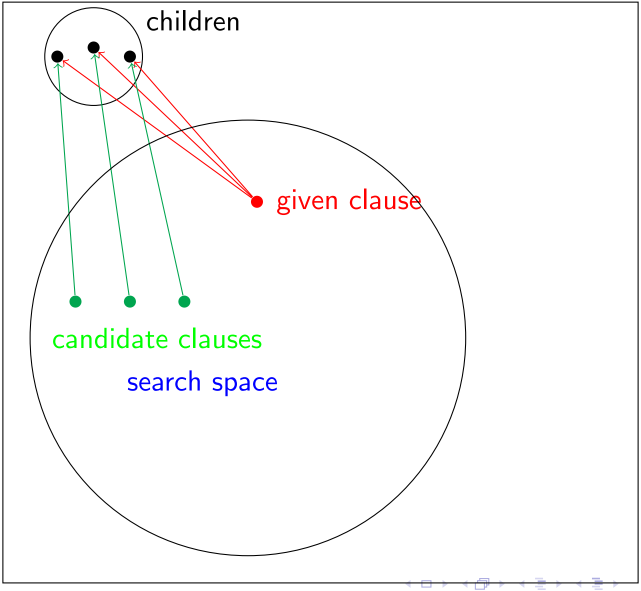
Inference Selection by Clause Selection



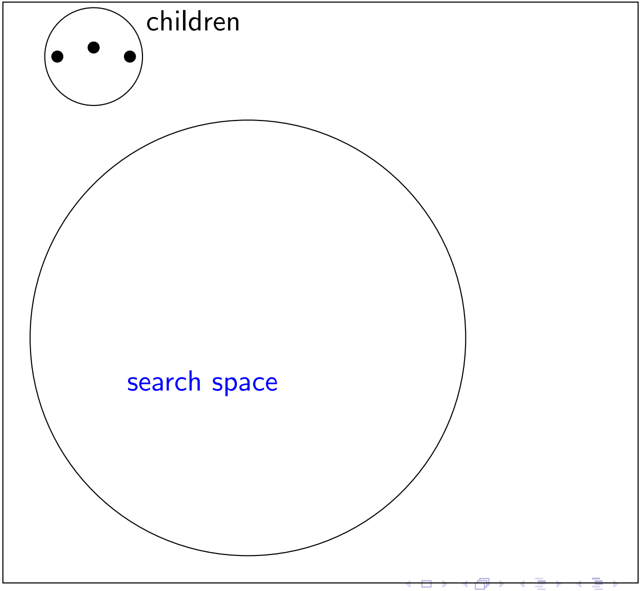
Inference Selection by Clause Selection



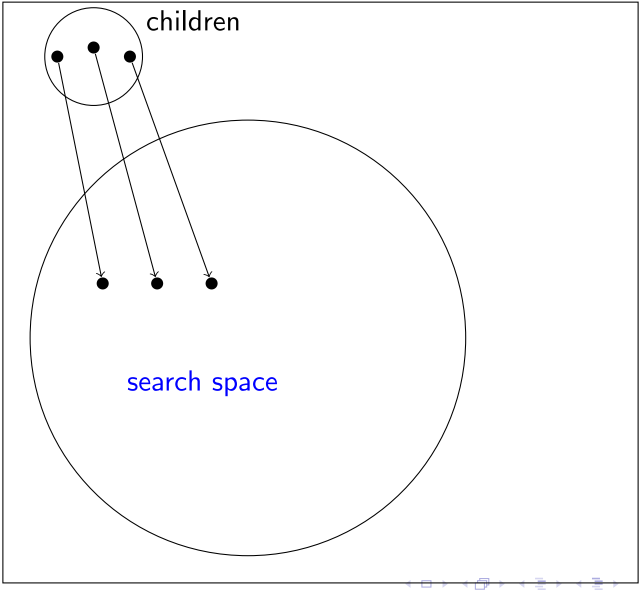
Inference Selection by Clause Selection



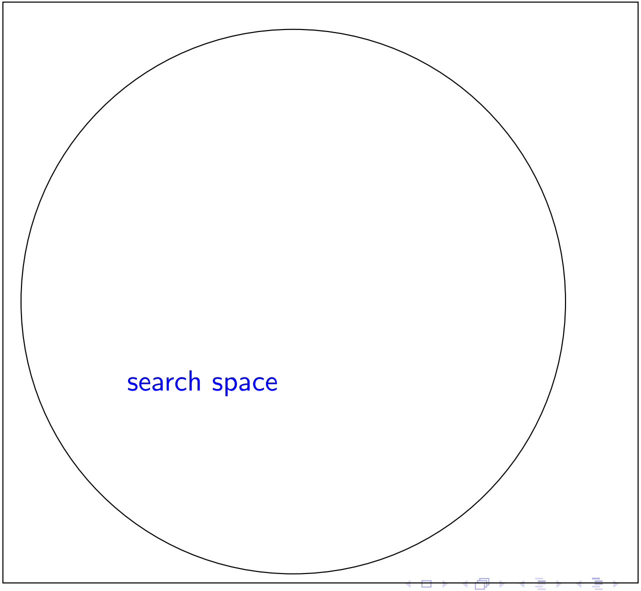
Inference Selection by Clause Selection



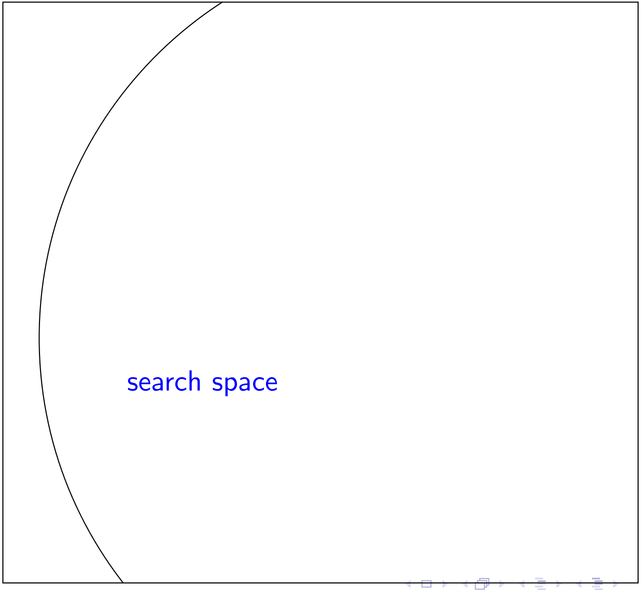
Inference Selection by Clause Selection



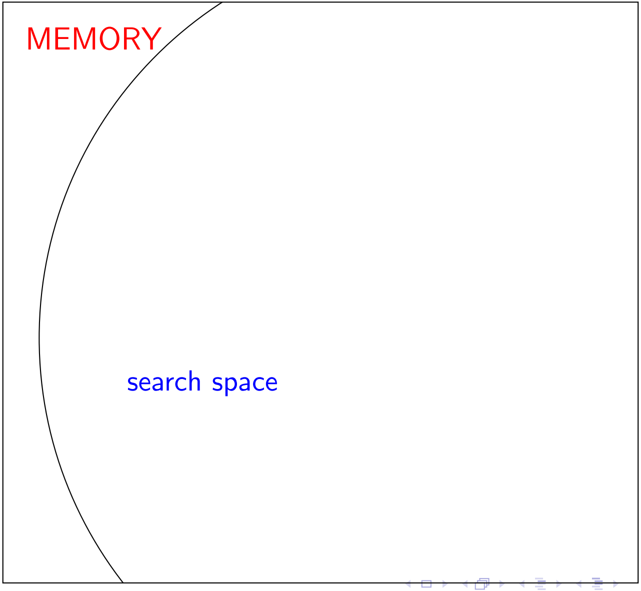
Inference Selection by Clause Selection



Inference Selection by Clause Selection



Inference Selection by Clause Selection

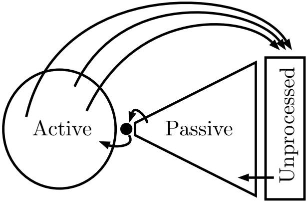


Saturation with the Given-Clause Algorithm

Only apply inferences to the *selected clause and the previously selected clauses*.

Saturation with the Given-Clause Algorithm

Only apply inferences to the *selected clause and the previously selected clauses*.



Thus, the search space is divided in two parts:

- active clauses, that participate in inferences;
- passive clauses, that do not participate in inferences.

Observation: the set of passive clauses is usually considerably larger than the set of active clauses, often by 2-4 orders of magnitude (depending on the saturation algorithm and the problem).

Anatomy of Saturation

Active

- initially empty
- backed up by sophisticated data structures (indexes) to allow efficient processing of inferences

Passive

- initially contains the clausified input
- typically consists of several queues ordering clauses to process by various (heuristical) criteria
- fairness!

Unprocessed:

- a temporary container
- just after generation, simplify before put into passive

The Clause Selection Task

Selecting the given clause is arguably the most important choice point in the implementation of a saturation algorithm

- If we only knew which to select up front . . .
- the standard approach: two queues (age, weight) and a ratio
- a natural spot for applying ML

The Clause Selection Task

Selecting the given clause is arguably the most important choice point in the implementation of a saturation algorithm

- If we only knew which to select up front ...
- the standard approach: two queues (age, weight) and a ratio
- a natural spot for applying ML

Notable attempts so far:

- Deep Network Guided Proof Search. LPAR 2017
- ENIGMA: Efficient Learning-Based Inference Guiding Machine. CICM 2017
- much more work done since (Jan will tell)

Outline

Making It Fast in Practice

- Literal selection and ordering constraints
  - restrict applicability of inference rules
- Redundancy elimination and simplifications
  - tautology deletions, subsumption, demodulation
- Saturation loop variants
  - Otter loop, Discount loop, LRS
- The AVATAR architecture
- Efficient data structures: term sharing, indexing, ...
- Specialised modes and calculi: InstGen, FMB, . . .
- . . .
- Strategy scheduling mode

Options and Strategies

A typical theorem prover has many ways to set up and run the proving process.

A naive idea: leave it up to the user to pick the best option setup, i.e. a strategy, for the problem P at hand.

Options and Strategies

A typical theorem prover has many ways to set up and run the proving process.

A naive idea: leave it up to the user to pick the best option setup, i.e. a strategy, for the problem P at hand.

A more fruitful idea:
Automatically run a full schedule of strategies, ideally selected to have complementary strengths/weaknesses such that they cover the most problems.

- Introduced in Gandalf, (Tammet 1998)
- Vampire's famous *CASC mode*

Two more machine learning tasks:

Automatic Strategy Selection

- Given a problem P pick a strategy most likely to succeed on P
- e.g. MaLeS: A Framework for Automatic Tuning of Automated Theorem Provers. J. Autom. Reasoning 2015

Two more machine learning tasks:

Automatic Strategy Selection

- Given a problem P pick a strategy most likely to succeed on P
- e.g. MaLeS: A Framework for Automatic Tuning of Automated Theorem Provers. J. Autom. Reasoning 2015

Automatic Strategy Invention

- Automatically discover sets of (complementary) strategies that together solve many problems (over a given benchmark)
- BliStr: The Blind Strategymaker. GCAI 2015
- BliStrTune: hierarchical invention of theorem proving strategies. CPP 2017

Outline

Thank you!

Questions?