# Automated Theorem Proving

## An Introduction

Laura Kovács
TU Vienna

# First-Order Logic

- A first-order signature: function (including constant) and 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 are built from atoms using logical connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ and quantifiers $\forall$, $\exists$. For example, $(\forall x)x = 0 \vee (\exists y)y > x$.

# Exercises

Is is true that:

$$\exists x\ \forall y\ p(x, y) \ \rightarrow\ \forall\ y\ \exists x\ p(x, y)$$

# Exercises

Is is true that:

$$\exists x\, \forall y\; p(x,y) \;\rightarrow\; \forall\, y\, \exists x\; p(x,y)$$

Prove it with our theorem prover: VAMPIRE (vprover.org)

# Exercises

Is is true that:

$$\exists x \, \forall y \; p(x, y) \; \rightarrow \; \forall \, y \, \exists x \; p(x, y)$$

Prove it with our theorem prover: VAMPIRE (vprover.org)
and thank it to yourself!

# Proof by Refutation

### Exercise

$$\boxed{\exists x\, \forall y\, p(x, y) \;\rightarrow\; \forall\, y\, \exists x\, p(x, y)}$$

# Proof by Refutation

## Exercise

$$\boxed{\exists x\ \forall y\ p(x,y)\ \rightarrow\ \forall\ y\ \exists x\ p(x,y)}$$

Assume: $\exists x\ \forall y\ p(x,y)$

Prove: $\forall\ y\ \exists x\ p(x,y)$

# Proof by Refutation

## Exercise: Proof by contradiction

$$\boxed{\exists x\ \forall y\ p(x, y)\ \rightarrow\ \forall\ y\ \exists x\ p(x, y)}$$

Assume: $\exists x\ \forall y\ p(x, y)$

Assume: $\neg\ \forall\ y\ \exists x\ p(x, y)$

Prove a contradiction!

# Proof by Refutation

Exercise: Proof by contradiction $\iff$ Proof by refutation

$$\boxed{\exists x\, \forall y\, p(x,y) \;\rightarrow\; \forall\, y\, \exists x\, p(x,y)}$$

Assume: $\exists x\, \forall y\, p(x,y)$

Assume: $\neg\, \forall\, y\, \exists x\, p(x,y)$

Prove a contradiction!

# Proof by Refutation

Given a problem with assumptions $F_1, \ldots, F_n$ 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.

Exercise: Proof by contradiction $\Longleftrightarrow$ Proof by refutation

$$\boxed{\exists x\, \forall y\, p(x,y) \;\rightarrow\; \forall\, y\, \exists x\, p(x,y)}$$

Assume: $\exists x\, \forall y\, p(x,y)$

Assume: $\neg\, \forall\, y\, \exists x\, p(x,y)$

Prove a contradiction!

# What an Automatic Theorem Prover is Expected to Do

Input:

- a set of assumptions and axioms (first order-formulas);
- a conjecture (first-order formula).

Output:

- proof (hopefully).

# What an Automatic Theorem Prover is Expected to Do

Input:

- a set of assumptions and axioms (first order-formulas);
- a conjecture (first-order formula).

Output:

- proof (hopefully).

*Note:*

Once an automatic theorem prover started a proof attempt,
it can only be interrupted by terminating the process.

# General Proving Scheme (simplified)

- Read a problem;
- Try to derive *false*.
- If *false* is derived, report the result, maybe including a refutation.

# General Proving Scheme (simplified)

- Read a problem;
- Try to derive *false*.
  - What are the proving rules?
  - How to use the proving rules?
- If *false* is derived, report the result, maybe including a refutation.

# General Proving Scheme (simplified)

- Read a problem;
- Try to derive *false*.
  - What are the proving rules? INFERENCE SYSTEM
  - How to use the proving rules? SATURATION ALGORITHM
- If *false* is derived, report the result, maybe including a refutation.

# General Proving Scheme (simplified)

- Read a problem;
- Try to derive *false*.
  - What are the proving rules? INFERENCE SYSTEM
  - How to use the proving rules? SATURATION ALGORITHM
- If *false* is derived, report the result, maybe including a refutation.

Notation: We will use $\square$ to denote *false* (the formula which is always false).

# Outline

# Example from Algebra

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.

# Example from Algebra

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.
More formally: in a group "assuming that $x^2 = 1$ for all $x$ prove that $x \cdot y = y \cdot x$ holds for all $x, y$."

# Example from Algebra

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.
More formally: in a group "assuming that $x^2 = 1$ for all $x$ prove that $x \cdot y = y \cdot x$ holds for all $x, y$."
What is implicit: axioms of the group theory.

$$\forall x(1 \cdot x = x)$$
$$\forall x(x^{-1} \cdot x = 1)$$
$$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

# Formulation in First-Order Logic with Equality

| | |
|---|---|
| | $\forall x(1 \cdot x = x)$ |
| Axioms (of group theory): | $\forall x(x^{-1} \cdot x = 1)$ |
| | $\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$ |
| Assumptions: | $\forall x(x \cdot x = 1)$ |
| Conjecture: | $\forall x \forall y(x \cdot y = y \cdot x)$ |

# Proof by Vampire (Slightly Modified)

```
Refutation found. Thanks to Tanya!
203. $false [subsumption resolution 202,14]
202. sP1(mult(sK,sK0)) [backward demodulation 188,15]
188. mult(X8,X9) = mult(X9,X8) [superposition 22,87]
87. mult(X2,mult(X1,X2)) = X1 [forward demodulation 71,27]
71. mult(inverse(X1),e) = mult(X2,mult(X1,X2)) [superposition 23,20]
27. mult(inverse(X2),e) = X2 [superposition 22,10]
23. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 18,9]
22. mult(X0,mult(X0,X1)) = X1 [forward demodulation 16,9]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 11,12]
18. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 11,10]
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
15. sP1(mult(sK0,sK)) [inequality splitting 13,14]
14. ~sP1(mult(sK,sK0)) [inequality splitting name introduction]
13. mult(sK,sK0) != mult(sK0,sK) [cnf transformation 8]
12. e = mult(X0,X0) (0:5) [cnf transformation 4]
11. mult(mult(X0,X1),X2)=mult(X0,mult(X1,X2))[cnf transformation 3]
10. e = mult(inverse(X0),X0) [cnf transformation 2]
9. mult(e,X0) = X0 [cnf transformation 1]
8. mult(sK,sK0) != mult(sK0,sK) [skolemisation 7]
7. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ! [X0,X1] : mult(X0,X1) = mult(X1,X0) [input]
4. ! [X0] : e = mult(X0,X0)[input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))[input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
1. ! [X0] : mult(e,X0) = X0 [input]
```

# Proof by Vampire (Slightly Modified)

```
Refutation found. Thanks to Tanya!
203. $false [subsumption resolution 202,14]
202. sP1(mult(sK,sK0)) [backward demodulation 188,15]
188. mult(X8,X9) = mult(X9,X8) [superposition 22,87]
87. mult(X2,mult(X1,X2)) = X1 [forward demodulation 71,27]
71. mult(inverse(X1),e) = mult(X2,mult(X1,X2)) [superposition 23,20]
27. mult(inverse(X2),e) = X2 [superposition 22,10]
23. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 18,9]
22. mult(X0,mult(X0,X1)) = X1 [forward demodulation 16,9]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 11,12]
18. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 11,10]
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
15. sP1(mult(sK0,sK)) [inequality splitting 13,14]
14. ~sP1(mult(sK,sK0)) [inequality splitting name introduction]
13. mult(sK,sK0) != mult(sK0,sK) [cnf transformation 8]
12. e = mult(X0,X0) (0:5) [cnf transformation 4]
11. mult(mult(X0,X1),X2)=mult(X0,mult(X1,X2))[cnf transformation 3]
10. e = mult(inverse(X0),X0) [cnf transformation 2]
9. mult(e,X0) = X0 [cnf transformation 1]
8. mult(sK,sK0) != mult(sK0,sK) [skolemisation 7]
7. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ! [X0,X1] : mult(X0,X1) = mult(X1,X0) [input]
4. ! [X0] : e = mult(X0,X0)[input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))[input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
1. ! [X0] : mult(e,X0) = X0 [input]
```

- Proof by refutation;

# Proof by Vampire (Slightly Modified)

```
Refutation found. Thanks to Tanya!
203. $false [subsumption resolution 202,14]
202. sP1(mult(sK,sK0)) [backward demodulation 188,15]
188. mult(X8,X9) = mult(X9,X8) [superposition 22,87]
87. mult(X2,mult(X1,X2)) = X1 [forward demodulation 71,27]
71. mult(inverse(X1),e) = mult(X2,mult(X1,X2)) [superposition 23,20]
27. mult(inverse(X2),e) = X2 [superposition 22,10]
23. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 18,9]
22. mult(X0,mult(X0,X1)) = X1 [forward demodulation 16,9]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 11,12]
18. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 11,10]
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
15. sP1(mult(sK0,sK)) [inequality splitting 13,14]
14. ~sP1(mult(sK,sK0)) [inequality splitting name introduction]
13. mult(sK,sK0) != mult(sK0,sK) [cnf transformation 8]
12. e = mult(X0,X0) (0:5) [cnf transformation 4]
11. mult(mult(X0,X1),X2)=mult(X0,mult(X1,X2))[cnf transformation 3]
10. e = mult(inverse(X0),X0) [cnf transformation 2]
9. mult(e,X0) = X0 [cnf transformation 1]
8. mult(sK,sK0) != mult(sK0,sK) [skolemisation 7]
7. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ! [X0,X1] : mult(X0,X1) = mult(X1,X0) [input]
4. ! [X0] : e = mult(X0,X0)[input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))[input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
1. ! [X0] : mult(e,X0) = X0 [input]
```

- Inference rules of the superposition calculus;

# Proof by Vampire (Slightly Modified)

```
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
```

```
12. e = mult(X0,X0)
11. mult(mult(X0,X0),X1)=mult(X0,mult(X0,X1))
```

- Inference rules of the superposition calculus;

# Proof by Vampire (Slightly Modified)

```
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
```

```
12. e = mult(X0,X0)
11. mult(mult(X0,X0),X1)=mult(X0,mult(X0,X1))
```

- Inference rules of the superposition calculus;

# Proof by Vampire (Slightly Modified)

```
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
```

```
12. e = mult(X0,X0)
11. mult(mult(X0,X0),X1)=mult(X0,mult(X0,X1))
```

- Inference rules of the superposition calculus;

# Proof by Vampire (Slightly Modified)

```
Refutation found. Thanks to Tanya!
203. $false [subsumption resolution 202,14]
202. sP1(mult(sK,sK0)) [backward demodulation 188,15]
188. mult(X8,X9) = mult(X9,X8) [superposition 22,87]
87. mult(X2,mult(X1,X2)) = X1 [forward demodulation 71,27]
71. mult(inverse(X1),e) = mult(X2,mult(X1,X2)) [superposition 23,20]
27. mult(inverse(X2),e) = X2 [superposition 22,10]
23. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 18,9]
22. mult(X0,mult(X0,X1)) = X1 [forward demodulation 16,9]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 11,12]
18. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 11,10]
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
15. sP1(mult(sK0,sK)) [inequality splitting 13,14]
14. ~sP1(mult(sK,sK0)) [inequality splitting name introduction]
13. mult(sK,sK0) != mult(sK0,sK) [cnf transformation 8]
12. e = mult(X0,X0) (0:5) [cnf transformation 4]
11. mult(mult(X0,X1),X2)=mult(X0,mult(X1,X2))[cnf transformation 3]
10. e = mult(inverse(X0),X0) [cnf transformation 2]
9. mult(e,X0) = X0 [cnf transformation 1]
8. mult(sK,sK0) != mult(sK0,sK) [skolemisation 7]
7. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ! [X0,X1] : mult(X0,X1) = mult(X1,X0) [input]
4. ! [X0] : e = mult(X0,X0)[input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))[input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
1. ! [X0] : mult(e,X0) = X0 [input]
```

- Proof by refutation;
- Inference rules of the superposition calculus;
- Each inference derives a new formula;

# Proof by Vampire (Slightly Modified)

```
Refutation found. Thanks to Tanya!
203. $false [subsumption resolution 202,14]
202. sP1(mult(sK,sK0)) [backward demodulation 188,15]
188. mult(X8,X9) = mult(X9,X8) [superposition 22,87]
87. mult(X2,mult(X1,X2)) = X1 [forward demodulation 71,27]
71. mult(inverse(X1),e) = mult(X2,mult(X1,X2)) [superposition 23,20]
27. mult(inverse(X2),e) = X2 [superposition 22,10]
23. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 18,9]
22. mult(X0,mult(X0,X1)) = X1 [forward demodulation 16,9]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 11,12]
18. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 11,10]
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
15. sP1(mult(sK0,sK)) [inequality splitting 13,14]
14. ~sP1(mult(sK,sK0)) [inequality splitting name introduction]
13. mult(sK,sK0) != mult(sK0,sK) [cnf transformation 8]
12. e = mult(X0,X0) (0:5) [cnf transformation 4]
11. mult(mult(X0,X1),X2)=mult(X0,mult(X1,X2))[cnf transformation 3]
10. e = mult(inverse(X0),X0) [cnf transformation 2]
9. mult(e,X0) = X0 [cnf transformation 1]
8. mult(sK,sK0) != mult(sK0,sK) [skolemisation 7]
7. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ! [X0,X1] : mult(X0,X1) = mult(X1,X0) [input]
4. ! [X0] : e = mult(X0,X0)[input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))[input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
1. ! [X0] : mult(e,X0) = X0 [input]
```

- Proof by refutation;
- Inference rules of the superposition calculus;
- Each inference derives a new formula;
- Generating and simplifying inferences.

# Proof by Vampire (Slightly Modified)

```
Refutation found. Thanks to Tanya!
203. $false [subsumption resolution 202,14]
202. sP1(mult(sK,sK0)) [backward demodulation 188,15]
188. mult(X8,X9) = mult(X9,X8) [superposition 22,87]
87. mult(X2,mult(X1,X2)) = X1 [forward demodulation 71,27]
71. mult(inverse(X1),e) = mult(X2,mult(X1,X2)) [superposition 23,20]
27. mult(inverse(X2),e) = X2 [superposition 22,10]
23. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 18,9]
22. mult(X0,mult(X0,X1)) = X1 [forward demodulation 16,9]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 11,12]
18. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 11,10]
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
15. sP1(mult(sK0,sK)) [inequality splitting 13,14]
14. ~sP1(mult(sK,sK0)) [inequality splitting name introduction]
13. mult(sK,sK0) != mult(sK0,sK) [cnf transformation 8]
12. e = mult(X0,X0) (0:5) [cnf transformation 4]
11. mult(mult(X0,X1),X2)=mult(X0,mult(X1,X2))[cnf transformation 3]
10. e = mult(inverse(X0),X0) [cnf transformation 2]
9. mult(e,X0) = X0 [cnf transformation 1]
8. mult(sK,sK0) != mult(sK0,sK) [skolemisation 7]
7. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ! [X0,X1] : mult(X0,X1) = mult(X1,X0) [input]
4. ! [X0] : e = mult(X0,X0)[input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))[input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
1. ! [X0] : mult(e,X0) = X0 [input]
```

- Proof by refutation;
- Inference rules of the superposition calculus;
- Each inference derives a new formula;
- Generating and simplifying inferences.

} Saturation algorithm

# Inference System

- inference rule 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 rule 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 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 $E$.

# The Superposition Inference System - An Inference System for Logic with Equality

We will define it only for propositional formulas (or ground formulas).

Notation: $s[l]$ denotes the term $s$ such that $l$ is a subterm of $s$.

# The Superposition Inference System - An Inference System for Logic with Equality

The ground superposition inference system $\mathbb{SRF}$ consists of three inference rules:

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:

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

Equality Factoring:

$$\frac{s = t \vee s = t' \vee C}{s = t \vee t \neq t' \vee C} \text{ (EF)},$$

# 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.

$\mathbb{SRF}$ is sound.

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

# Example

| | | |
|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | (input) |
| (3) | $f(f(a)) \neq a$ | (input) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \vee g(a) = a$ | $(1, 3)$ | (superposition) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \vee g(a) = a$ | $(1, 3)$ | (superposition) |
| (5) | $a \neq a \vee g(a) = a \vee g(a) = a$ | $(1, 4)$ | (superposition) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \vee g(a) = a$ | $(1, 3)$ | (superposition) |
| (5) | $a \neq a \vee g(a) = a \vee g(a) = a$ | $(1, 4)$ | (superposition) |
| (6) | $g(a) = a \vee g(a) = a$ | $(5)$ | (equality resolution) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \vee g(a) = a$ | $(1, 3)$ | (superposition) |
| (5) | $a \neq a \vee g(a) = a \vee g(a) = a$ | $(1, 4)$ | (superposition) |
| (6) | $g(a) = a \vee g(a) = a$ | $(5)$ | (equality resolution) |
| (7) | $g(a) = a \vee a \neq a$ | $(6)$ | (equality factoring) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \vee g(a) = a$ | $(1,3)$ | (superposition) |
| (5) | $a \neq a \vee g(a) = a \vee g(a) = a$ | $(1,4)$ | (superposition) |
| (6) | $g(a) = a \vee g(a) = a$ | $(5)$ | (equality resolution) |
| (7) | $g(a) = a \vee a \neq a$ | $(6)$ | (equality factoring) |
| (8) | $g(a) = a$ | $(7)$ | (equality resolution) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \lor g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \lor g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \lor g(a) = a$ | $(1, 3)$ | (superposition) |
| (5) | $a \neq a \lor g(a) = a \lor g(a) = a$ | $(1, 4)$ | (superposition) |
| (6) | $g(a) = a \lor g(a) = a$ | $(5)$ | (equality resolution) |
| (7) | $g(a) = a \lor a \neq a$ | $(6)$ | (equality factoring) |
| (8) | $g(a) = a$ | $(7)$ | (equality resolution) |
| (9) | $f(f(a)) = a \lor g(a) \neq a$ | $(2, 8)$ | (superposition) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \vee g(a) = a$ | $(1, 3)$ | (superposition) |
| (5) | $a \neq a \vee g(a) = a \vee g(a) = a$ | $(1, 4)$ | (superposition) |
| (6) | $g(a) = a \vee g(a) = a$ | $(5)$ | (equality resolution) |
| (7) | $g(a) = a \vee a \neq a$ | $(6)$ | (equality factoring) |
| (8) | $g(a) = a$ | $(7)$ | (equality resolution) |
| (9) | $f(f(a)) = a \vee g(a) \neq a$ | $(2, 8)$ | (superposition) |
| (10) | $f(f(a)) = a \vee a \neq a$ | $(8, 9)$ | (superposition) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \vee g(a) = a$ | $(1, 3)$ | (superposition) |
| (5) | $a \neq a \vee g(a) = a \vee g(a) = a$ | $(1, 4)$ | (superposition) |
| (6) | $g(a) = a \vee g(a) = a$ | $(5)$ | (equality resolution) |
| (7) | $g(a) = a \vee a \neq a$ | $(6)$ | (equality factoring) |
| (8) | $g(a) = a$ | $(7)$ | (equality resolution) |
| (9) | $f(f(a)) = a \vee g(a) \neq a$ | $(2, 8)$ | (superposition) |
| (10) | $f(f(a)) = a \vee a \neq a$ | $(8, 9)$ | (superposition) |
| (11) | $f(f(a)) = a$ | $(10)$ | (equality resolution) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \vee g(a) = a$ | $(1, 3)$ | (superposition) |
| (5) | $a \neq a \vee g(a) = a \vee g(a) = a$ | $(1, 4)$ | (superposition) |
| (6) | $g(a) = a \vee g(a) = a$ | $(5)$ | (equality resolution) |
| (7) | $g(a) = a \vee a \neq a$ | $(6)$ | (equality factoring) |
| (8) | $g(a) = a$ | $(7)$ | (equality resolution) |
| (9) | $f(f(a)) = a \vee g(a) \neq a$ | $(2, 8)$ | (superposition) |
| (10) | $f(f(a)) = a \vee a \neq a$ | $(8, 9)$ | (superposition) |
| (11) | $f(f(a)) = a$ | $(10)$ | (equality resolution) |
| (12) | $a \neq a$ | $(3, 11)$ | (superposition) |

# Example

| | | | |
|---|---|---|---|
| (1) | $f(a) = a \vee g(a) = a$ | | (input) |
| (2) | $f(f(a)) = a \vee g(g(a)) \neq a$ | | (input) |
| (3) | $f(f(a)) \neq a$ | | (input) |
| (4) | $f(a) \neq a \vee g(a) = a$ | $(1, 3)$ | (superposition) |
| (5) | $a \neq a \vee g(a) = a \vee g(a) = a$ | $(1, 4)$ | (superposition) |
| (6) | $g(a) = a \vee g(a) = a$ | $(5)$ | (equality resolution) |
| (7) | $g(a) = a \vee a \neq a$ | $(6)$ | (equality factoring) |
| (8) | $g(a) = a$ | $(7)$ | (equality resolution) |
| (9) | $f(f(a)) = a \vee g(a) \neq a$ | $(2, 8)$ | (superposition) |
| (10) | $f(f(a)) = a \vee a \neq a$ | $(8, 9)$ | (superposition) |
| (11) | $f(f(a)) = a$ | $(10)$ | (equality resolution) |
| (12) | $a \neq a$ | $(3, 11)$ | (superposition) |
| (13) | $\square$ | $(12)$ | (equality resolution) |

# Soundness be used for Checking (Un)satisfiability!

Completeness.

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

# Soundness be used for Checking (Un)satisfiability!

Completeness.

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

How to find this derivation: using a saturation algorithm.

# Outline

# How to Establish Unsatisfiability?

Idea:

- Take a set of formulas $S$, initially $S = S_0$,

  where $S_0$ is the input set of formulas.

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

# How to Establish Unsatisfiability?

Idea:

- Take a set of formulas $S$, initially $S = S_0$,

  where $S_0$ is the input set of formulas.

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

$$S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \dots$$

1. there exists an inference

$$\frac{F_1 \quad \dots \quad F_n}{F}$$

   in $\mathbb{I}$ such that $\{F_1, \dots, F_n\} \subseteq S_i$;
2. $S_{i+1} = S_i \cup \{F\}$.

# How to Establish Unsatisfiability?

Idea:

- Take a set of formulas $S$, initially $S = S_0$,

  where $S_0$ is the input set of formulas.

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

$\mathbb{I}$-inference process: $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$

1. there exists an inference

$$\frac{F_1 \quad \ldots \quad F_n}{F}$$

   in $\mathbb{I}$ such that $\{F_1, \ldots, F_n\} \subseteq S_i$;
2. $S_{i+1} = S_i \cup \{F\}$.

# How to Establish Unsatisfiability?

Idea:

- Take a set of formulas $S$, initially $S = S_0$,

  where $S_0$ is the input set of formulas.

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

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

- 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 formulas in $S$ is already a member of $S$. Any such set of formulas is called saturated.

# How to Establish Satisfiability?

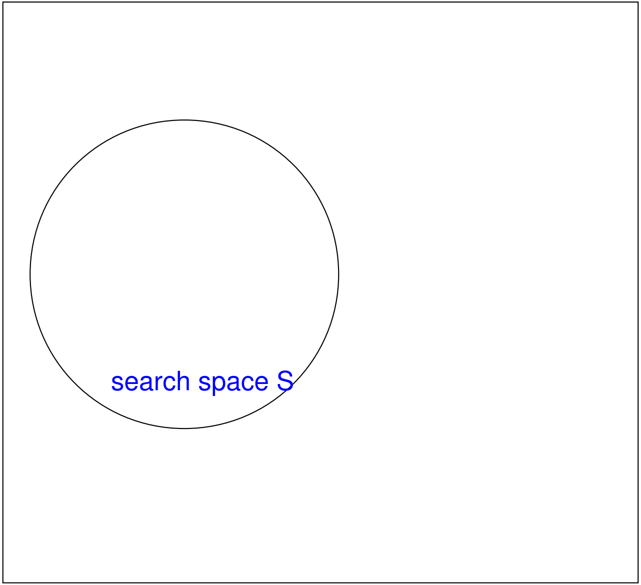
When can we report satisfiability?

When we build a set $S$ such that any inference applied to formulas in $S$ is already a member of $S$. Any such set of formulas is called saturated.

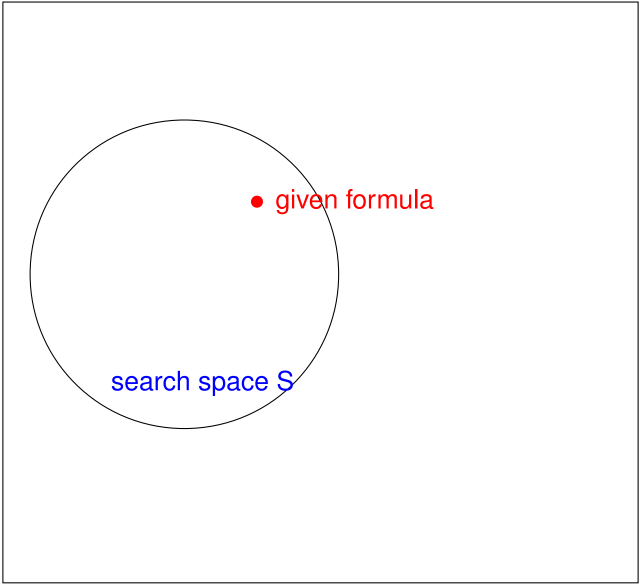
In first-order logic it is often the case that all saturated sets are infinite, so in practice we can almost never build a saturated set.

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

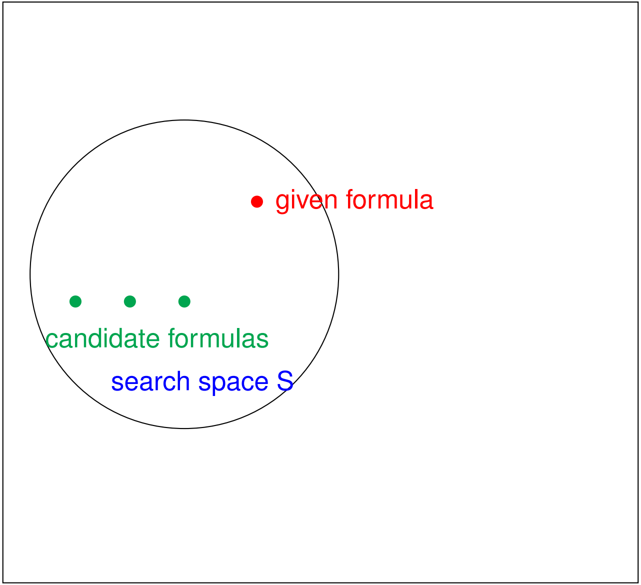
# Saturation Algorithms



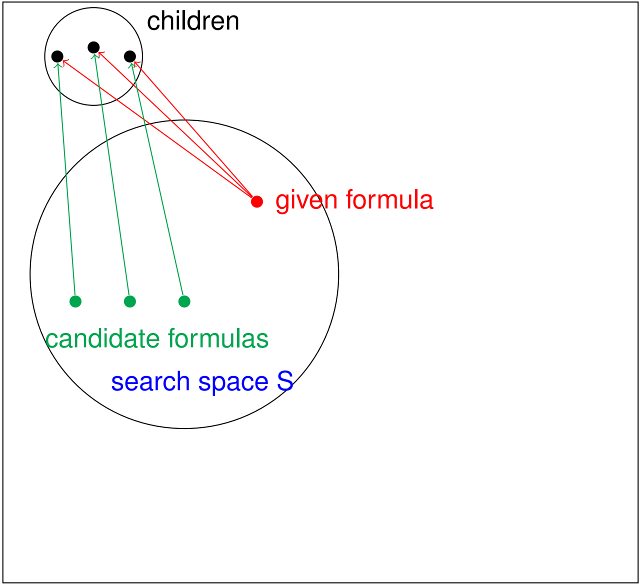
# Saturation Algorithms



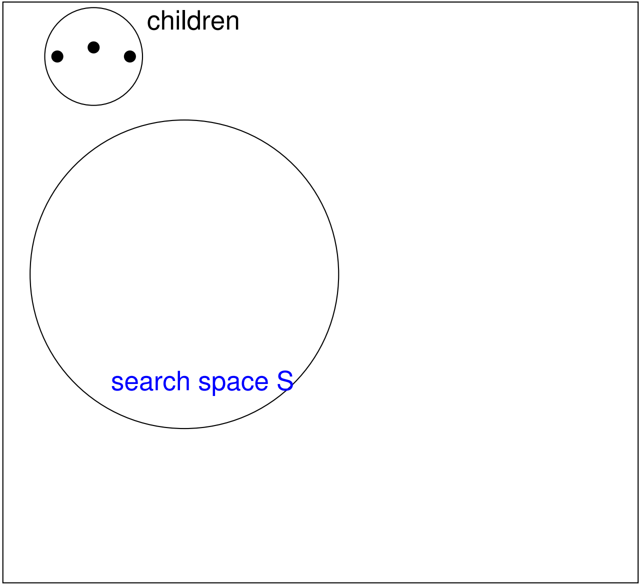
# Saturation Algorithms



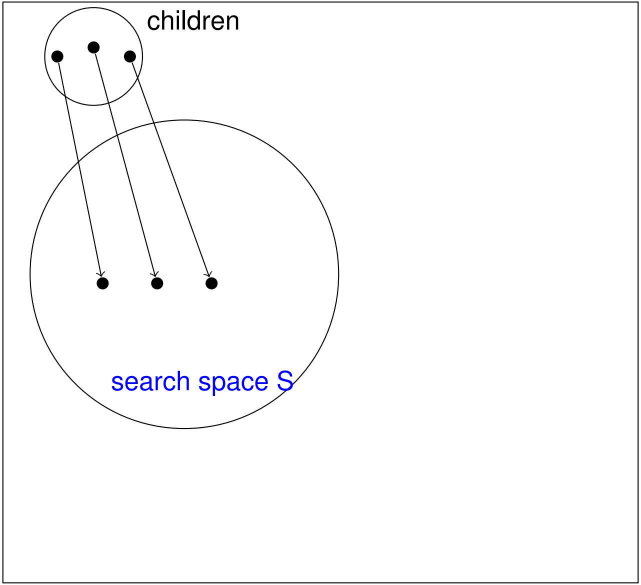
# Saturation Algorithms



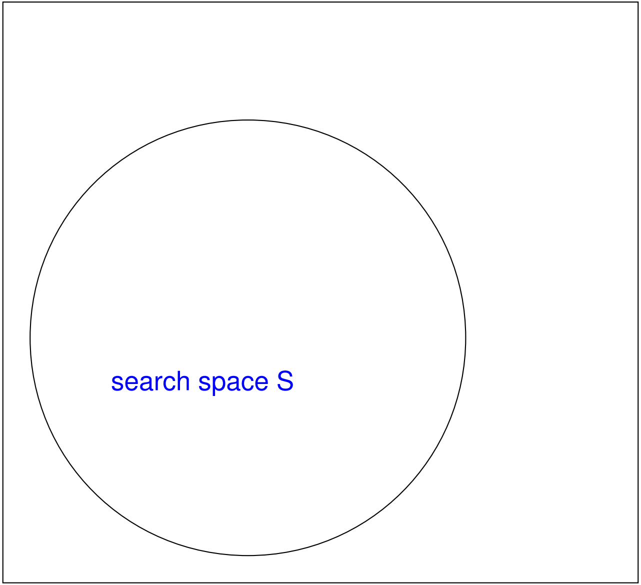
# Saturation Algorithms



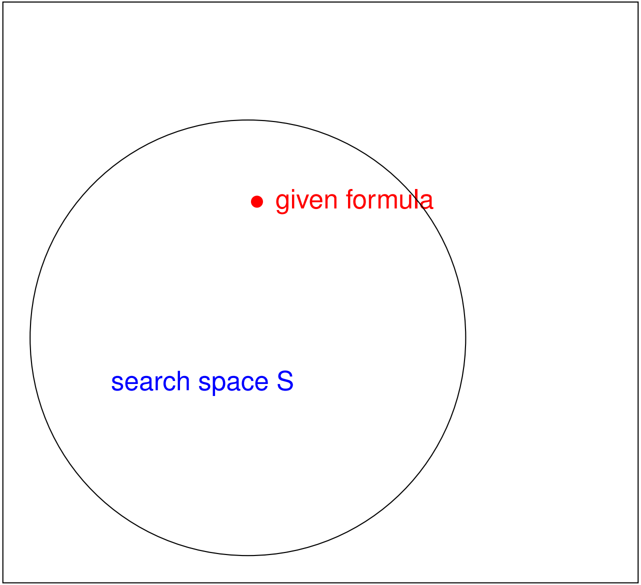
# Saturation Algorithms



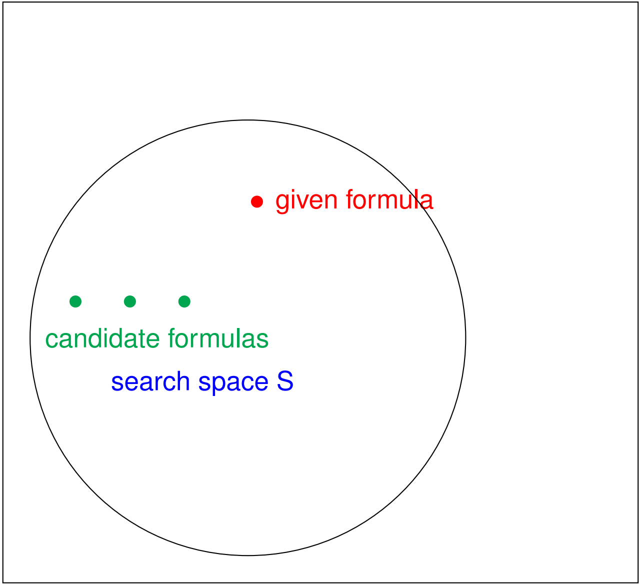
# Saturation Algorithms



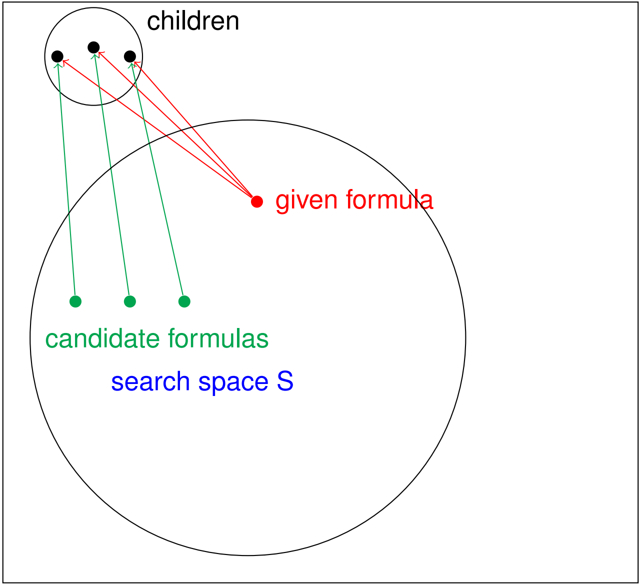
# Saturation Algorithms



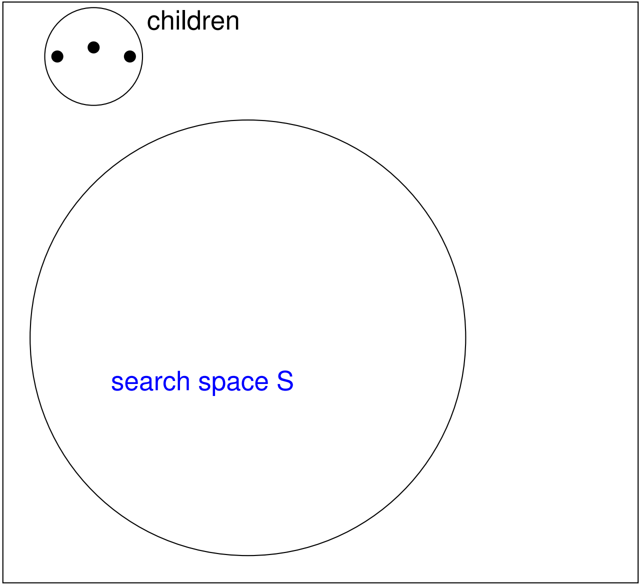
# Saturation Algorithms



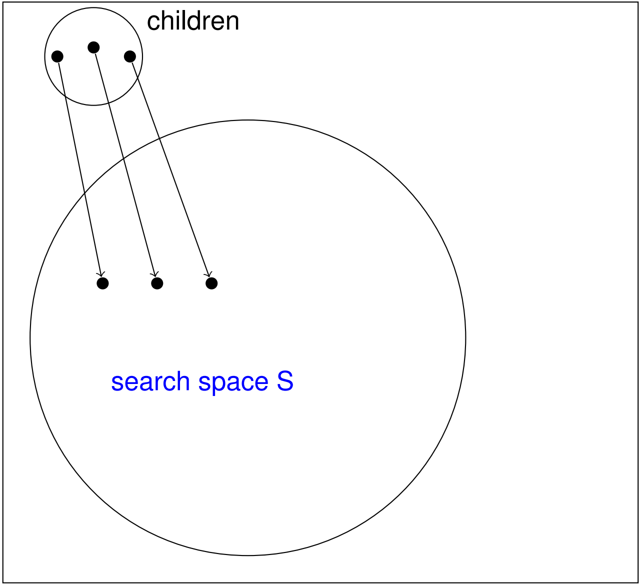
# Saturation Algorithms



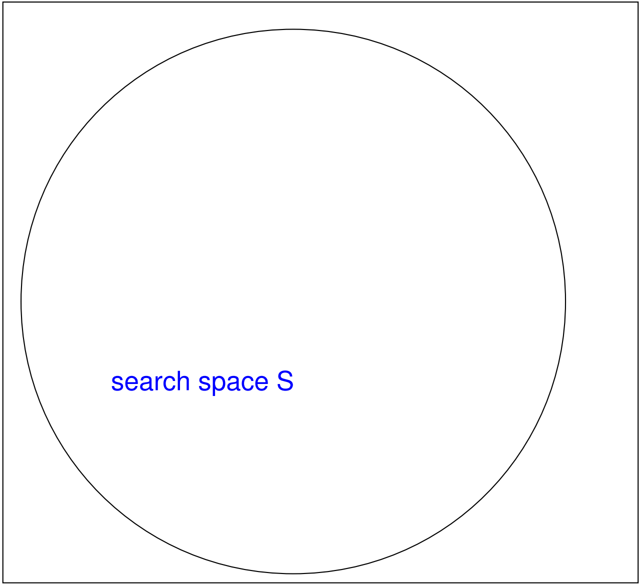
# Saturation Algorithms



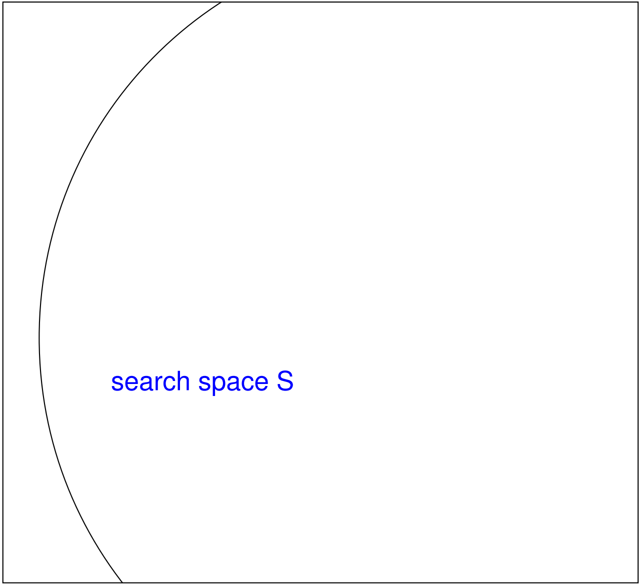
# Saturation Algorithms



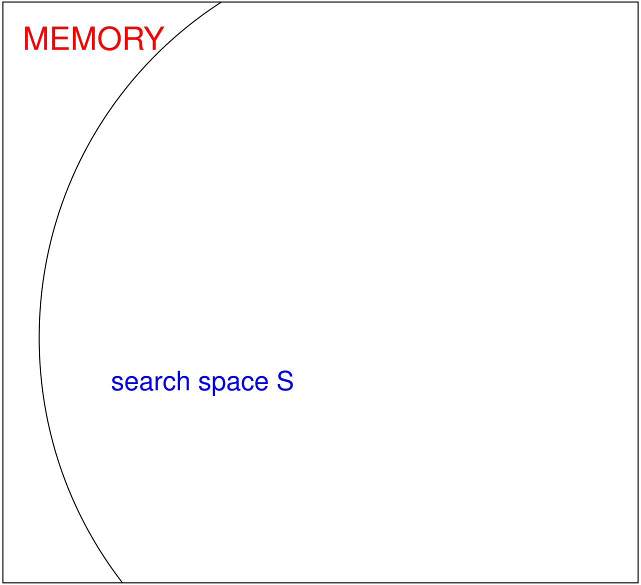
# Saturation Algorithms



# Saturation Algorithms



# Saturation Algorithms



# Saturation Algorithm

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

In theory there are three possible scenarios:

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

# Saturation Algorithm in Practice

In practice there are three possible scenarios:

1. At some moment the empty formula $\square$ is generated, in this case the input set of formulas is unsatisfiable.
2. Saturation will terminate without ever generating $\square$, in this case the input set of formulas 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

# From theory to practice

- Preprocessing input problems;
- Normal form transformations of formulas;
- Superposition system;
- Orderings;
- Selection functions;
- Fairness (saturation algorithms);
- Redundancy.

# Our story of success ... http://vprover.org

ver.org/trophies.cgi

Search PDFCreator eBay Amazon Coupons Radio Options

Suche AVIRA Der Browser Schutz ist aktiviert. Aktualisieren Sie Avira

## Our Trophies


Vampire is winning at least one division of the world cup in theorem proving CASC since 1999. All together Vampire won 23 titles: more than any other prover.
We traditionally take part in the following two divisions of the competition:

1. The FOF division: unrestricted first-order problems. This division was ranked second in importance after the MIX division before 2007 and is now recognised as the main competition division.
2. The CNF division: first-order problems in conjunctive normal form. This division was called MIX and recognised as the main competition division before 2007.
3. The LTB division: problems with very large axiomatisations (some of them contain about 3.5 million axioms).

Here is the list of our achievements:

| | FOF | CNF/MIX | LTB |
|---|---|---|---|
| 1999 | | winner | - |
| 2000 | winner | | - |
| 2001 | | winner | - |
| 2002 | winner | winner | - |
| 2003 | winner | winner | - |
| 2004 | winner* | winner | - |
| 2005 | winner | winner* | - |
| 2006 | winner | winner* | - |
| 2007 | winner | winner* | - |
| 2008 | winner | winner* | |
| 2009 | winner | winner* | winner |
| 2010 | winner | winner | winner* |

Note: winner* means that Vampire solved more problems that all other provers in this division and '-' means that t
not exist that year.

# Outline

# Homework Exercises

Problem 1. Establish the unsatisfiability of the following set of four formulas, using the superposition inference system $\mathbb{SRF}$:

$$\begin{array}{ll}(1) & c = d \\ (2) & f(d) \neq d \vee a = b \\ (3) & f(c) = d \\ (4) & g(a, b) \neq g(b, a)\end{array}$$

Problem 2. The limit of an $\mathbb{I}$-inference process $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ is the set of formulas $\bigcup_i S_i$. In other words, the limit is the set of all derived formulas.

Suppose that we have an infinite inference process such that $S_0$ is unsatisfiable and we use the ground superposition inference system $\mathbb{SRF}$.

Question: does completeness of $\mathbb{SRF}$ imply that the limit of the process contains the empty clause? Justify your answer!