First-Order Theorem Proving and Vampire

Laura Kovács

Schedule – First-Order Theorem Proving and Vampire

- *Session 1 (June 14)*: Getting Started
- *Session 2 (June 15)*: Overview and Theory
- *Session 3 (June 16)*: Practice and Cookies

Outline

Automated Reasoning by First-Order Theorem Proving

In a vague sense, automated reasoning involves

1. Representing a problem as a mathematical/logical statement
2. Automatically checking this statement's consistency or truth

Automated Reasoning by First-Order Theorem Proving

In a vague sense, automated reasoning involves

1. Representing a problem as a mathematical/logical statement
2. Automatically checking this statement's consistency or truth

There are lots of places where we can apply automated reasoning. For example,

- Proving software correctness (partial/total correctness)
- Generating loop invariants
- Program synthesis
- Model checking
- Your idea?

Kinds of Automated Reasoning

Given a statement S we can establish different conclusions about it

- Consistency - there is a way of making S true
- Inconsistency - there is no way of making S true
- Validity - S is always true

Kinds of Automated Reasoning

Given a statement S we can establish different conclusions about it

- Consistency - there is a way of making S true
- Inconsistency - there is no way of making S true
- Validity - S is always true

We can look at these three notions from two different views.

	Semantic view	Syntactic view
S is consistent	Has a model	No proof of $\perp$ from S
S is inconsistent	No model	A proof of $\perp$ from S
S is valid	True in all models	A proof of $\perp$ from $\neg S$

Kinds of Automated Reasoning

Given a statement S we can establish different conclusions about it

- Consistency - there is a way of making S true
- Inconsistency - there is no way of making S true
- Validity - S is always true

We can look at these three notions from two different views.

	Semantic view	Syntactic view
S is consistent	Has a model	No proof of $\perp$ from S
S is inconsistent	No model	A proof of $\perp$ from S
S is valid	True in all models	A proof of $\perp$ from $\neg S$

Notes

1. Here we have focussed only on proofs of inconsistency.
2. Consistency is commonly referred to as satisfiability

Kinds of Automated Reasoners

	Input	Example(s)
SAT Solvers	Propositional formulae	MiniSat
SMT Solvers	(First-order) formulae + theories	Z3,CVC4
Theorem Provers	First-order formulae (+ theories)	Vampire,E
Proof Assistants (interactive)	High-order formulae	Isabelle,Coq

Kinds of Automated Reasoners

	Input	Example(s)
SAT Solvers	Propositional formulae	MiniSat
SMT Solvers	(First-order) formulae + theories	Z3,CVC4
Theorem Provers	First-order formulae (+ theories)	Vampire,E
Proof Assistants (interactive)	High-order formulae	Isabelle,Coq

Above the line focus on models and might be decidable. Below the line focus on proofs and are rarely decidable.

Kinds of Automated Reasoners

	Input	Example(s)
SAT Solvers	Propositional formulae	MiniSat
SMT Solvers	(First-order) formulae + theories	Z3,CVC4
Theorem Provers	First-order formulae (+ theories)	Vampire,E
Proof Assistants (interactive)	High-order formulae	Isabelle,Coq

Above the line focus on models and might be decidable. Below the line focus on proofs and are rarely decidable.

Outline

First-Order Logic: Exercises

Which of the following statements are true?

First-Order Logic: Exercises

Which of the following statements are true?

1. First-order logic is an extension of propositional logic;

First-Order Logic: Exercises

Which of the following statements are true?

1. First-order logic is an extension of propositional logic;
2. First-order logic is NP-complete.

First-Order Logic: Exercises

Which of the following statements are true?

1. First-order logic is an extension of propositional logic;
2. First-order logic is NP-complete.
3. In first-order logic you can use quantifiers over sets.

First-Order Logic: Exercises

Which of the following statements are true?

1. First-order logic is an extension of propositional logic;
2. First-order logic is NP-complete.
3. In first-order logic you can use quantifiers over sets.
4. First-order logic is decidable.

First-Order Logic: Exercises

Which of the following statements are true?

1. First-order logic is an extension of propositional logic;
2. First-order logic is NP-complete.
3. In first-order logic you can use quantifiers over sets.
4. First-order logic is decidable.
5. First-order logic is an extension of arithmetic;

First-Order Logic: Exercises

Which of the following statements are true?

1. First-order logic is an extension of propositional logic;
2. First-order logic is NP-complete.
3. In first-order logic you can use quantifiers over sets.
4. First-order logic is decidable.
5. First-order logic is an extension of arithmetic;
6. One can axiomatise integers in first-order logic;

First-Order Logic: Exercises

Which of the following statements are true?

1. First-order logic is an extension of propositional logic;
2. First-order logic is NP-complete.
3. In first-order logic you can use quantifiers over sets.
4. First-order logic is decidable.
5. First-order logic is an extension of arithmetic;
6. One can axiomatise integers in first-order logic;
7. Compactness is the following property: a set of formulas having arbitrarily large finite models has an infinite model;

First-Order Logic: Exercises

Which of the following statements are true?

1. First-order logic is an extension of propositional logic;
2. First-order logic is NP-complete.
3. In first-order logic you can use quantifiers over sets.
4. First-order logic is decidable.
5. First-order logic is an extension of arithmetic;
6. One can axiomatise integers in first-order logic;
7. Compactness is the following property: a set of formulas having arbitrarily large finite models has an infinite model;
8. Having proofs is good.

First-Order Logic: Exercises

Which of the following statements are true?

1. First-order logic is an extension of propositional logic;
2. First-order logic is NP-complete.
3. In first-order logic you can use quantifiers over sets.
4. First-order logic is decidable.
5. First-order logic is an extension of arithmetic;
6. One can axiomatise integers in first-order logic;
7. Compactness is the following property: a set of formulas having arbitrarily large finite models has an infinite model;
8. Having proofs is good.
9. Vampire is a first-order theorem prover.

General

VAMPIRE: an automated first-order theorem prover

General

VAMPIRE: an automated first-order theorem prover

Go to

`https://vprover.github.io/download.html`

and pick the route most suitable to you.

Notes:

- For Linux users, a binary is probably the easiest route
- For Mac users, you need to build from source
  - run `make vampire_rel`
- For Windows users, the easiest route for this tutorial is a virtual machine and then use Linux

First-Order Theorem Proving. Example

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

First-Order Theorem Proving. Example

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."

First-Order Theorem Proving. Example

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

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

In the TPTP Syntax

The TPTP library (Thousands of Problems for Theorem Provers), `http://www.tptp.org` contains a large collection of first-order problems.

For representing these problems it uses the TPTP syntax, which is understood by all modern theorem provers, including Vampire.

In the TPTP Syntax

In the TPTP syntax this group theory problem can be written down as follows:

```
%---- 1 * x = x
fof(left_identity,axiom,
    ! [X] : mult(e,X) = X).
%---- i(x) * x = 1
fof(left_inverse,axiom,
    ! [X] : mult(inverse(X),X) = e).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,
    ! [X,Y,Z] : mult(mult(X,Y),Z) = mult(X,mult(Y,Z))).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
    ! [X] : mult(X,X) = e).
%---- prove x * y = y * x
fof(commutativity,conjecture,
    ! [X] : mult(X,Y) = mult(Y,X)).
```

Running Vampire of a TPTP file

is easy: simply use

```
vampire <filename>
```

Running Vampire of a TPTP file

is easy: simply use

```
vampire <filename>
```

One can also run Vampire with various options, some of them will be explained later. For example, save the group theory problem in a file `group.tptp` and try

```
vampire --thanks <your name> group.tptp
```

First-Order Logic (FOL) and TPTP

- Language: variables, function and predicate (relation) symbols. A constant symbol is a special case of a function symbol.

First-Order Logic (FOL) and TPTP

- Language: variables, function and predicate (relation) symbols. A constant symbol is a special case of a function symbol.
  In TPTP: Variable names start with upper-case letters.

First-Order Logic (FOL) and TPTP

- Language: variables, function and predicate (relation) symbols. A constant symbol is a special case of a function symbol.
  In TPTP: Variable names start with upper-case letters.
- Terms: variables, constants, and expressions $f(t_1, \ldots, t_n)$, where f is a function symbol of arity n and $t_1, \ldots, t_n$ are terms.

First-Order Logic (FOL) and TPTP

- Language: variables, function and predicate (relation) symbols. A constant symbol is a special case of a function symbol.
  In TPTP: Variable names start with upper-case letters.
- Terms: variables, constants, and expressions $f(t_1, \ldots, t_n)$, where f is a function symbol of arity n and $t_1, \ldots, t_n$ are terms. Terms denote domain elements.

First-Order Logic (FOL) and TPTP

- Language: variables, function and predicate (relation) symbols. A constant symbol is a special case of a function symbol.
  In TPTP: Variable names start with upper-case letters.
- Terms: variables, constants, and expressions $f(t_1, \ldots, t_n)$, where f is a function symbol of arity n and $t_1, \ldots, t_n$ are terms. Terms denote domain elements.
- Atomic formula: expression $p(t_1, \ldots, t_n)$, where p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms.

First-Order Logic (FOL) and TPTP

- Language: variables, function and predicate (relation) symbols. A constant symbol is a special case of a function symbol.
  In TPTP: Variable names start with upper-case letters.
- Terms: variables, constants, and expressions $f(t_1, \ldots, t_n)$, where f is a function symbol of arity n and $t_1, \ldots, t_n$ are terms. Terms denote domain elements.
- Atomic formula: expression $p(t_1, \ldots, t_n)$, where p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms. Formulas denote properties of domain elements.
- All symbols are uninterpreted, apart from equality $=$.

First-Order Logic and TPTP

FOL	TPTP
$\bot$, $\top$	`$false`, `$true`
$\neg a$	`~a`
$a_1 \wedge \ldots \wedge a_n$	`a1 & ... & an`
$a_1 \vee \ldots \vee a_n$	`a1 | ... | an`
$a_1 \rightarrow a_2$	`a1 => a2`
$(\forall x_1) \ldots (\forall x_n) a$	`! [X1,...,Xn] : a`
$(\exists x_1) \ldots (\exists x_n) a$	`? [X1,...,Xn] : a`

More on the TPTP Syntax

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
       mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

More on the TPTP Syntax

- Comments

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
       mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

More on the TPTP Syntax

- Comments
- Input formula names

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
       mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

More on the TPTP Syntax

- Comments
- Input formula names and roles

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
       mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

More on the TPTP Syntax

- Comments
- Input formula names and roles
- Equality

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
       mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

Proof by Vampire (Slightly Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1)!=mult(X1,X0)<=>mult(sK0,sK1)!=mult(sK1,sK0) [choice]
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(X0,mult(X1,X2)) = mult(mult(X0,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.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1)!=mult(X1,X0)<=>mult(sK0,sK1)!=mult(sK1,sK0) [choice]
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(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;

Proof by Vampire (Slightly Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1)!=mult(X1,X0)<=>mult(sK0,sK1)!=mult(sK1,sK0) [choice]
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(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus

Proof by Vampire (Slightly Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1)!=mult(X1,X0)<=>mult(sK0,sK1)!=mult(sK1,sK0) [choice]
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(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus

Proof by Vampire (Slightly Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1)!=mult(X1,X0)<=>mult(sK0,sK1)!=mult(sK1,sK0) [choice]
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(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus

Proof by Vampire (Slightly Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1)!=mult(X1,X0)<=>mult(sK0,sK1)!=mult(sK1,sK0) [choice]
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(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus

Proof by Vampire (Slightly Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1)!=mult(X1,X0)<=>mult(sK0,sK1)!=mult(sK1,sK0) [choice]
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(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus
- Proof by refutation, generating/ simplifying inferences, unused formulas

Proof by Vampire (Slightly Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1)!=mult(X1,X0)<=>mult(sK0,sK1)!=mult(sK1,sK0) [choice]
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(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus
- Proof by refutation, generating/ simplifying inferences, unused formulas

Proof by Vampire (Slightly Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1)!=mult(X1,X0)<=>mult(sK0,sK1)!=mult(sK1,sK0) [choice]
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(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus
- Proof by refutation, generating/ simplifying inferences, unused formulas

Vampire - CASC 2020 competition results

Higher-order Theorems	Zipperpin 2.0	*Satallax* *3.4*	Satallax 3.5	Vampire 4.5	Leo−III 1.5	CVC4 1.8
Solved/500	424/500	323/500	319/500	299/500	287/500	194/500
Solutions	424 84%	323 64%	319 63%	299 59%	287 57%	194 38%

Typed First-order Theorems +*-/	Vampire 4.5	*Vampire* *4.4*	CVC4 1.8
Solved/250	191/250	190/250	187/250
Solutions	191 76%	190 76%	187 74%

First-order Theorems	Vampire 4.5	*Vampire* *4.4*	Enigma 0.5.1	E 2.5	CSE_E 1.2	iProver 3.3
Solved/500	429/500	416/500	401/500	351/500	316/500	312/500
Solutions	429 85%	416 83%	401 80%	351 70%	316 63%	312 62%

First-order Non-theorems	Vampire SAT−4.5	*Vampire* *SAT−4.4*	iProver SAT−3.3	CVC4 SAT−1.8	E FNT−2.5	PyRes 1.3
Solved/250	238/250	226/250	182/250	98/250	63/250	13/250
Solutions	238 95%	226 90%	182 72%	98 39%	63 25%	13 5%

Vampire

- Completely automatic: once you started a proof attempt, it can only be interrupted by terminating the process.

Vampire

- Completely automatic: once you started a proof attempt, it can only be interrupted by terminating the process.

- Champion of the CASC world-cup in first-order theorem proving: won CASC $>$ 50 times.

What an Automatic Theorem Prover is Expected to Do

Input:

- a set of axioms (first order formulas) or clauses;
- a conjecture (first-order formula or set of clauses).

Output:

- proof (hopefully).

Proof by Refutation

Given a problem with axioms and assumptions $F_1, \ldots, F_n$ and conjecture G,

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

Proof by Refutation

Given a problem with axioms and 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.

Proof by Refutation

Given a problem with axioms and 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.

In this formulation the negation of the conjecture $\neg G$ is treated like any other formula.

In fact, Vampire (and other provers) internally treat conjectures differently, to make proof search more goal-oriented.

General Scheme (simplified)

- Read a problem;
- Determine proof-search options to be used for this problem;
- Preprocess the problem;
- Convert it into CNF;
- Run a saturation algorithm on it, try to derive *false*.
- If *false* is derived, report the result, maybe including a refutation.

General Scheme (simplified)

- Read a problem;
- Determine proof-search options to be used for this problem;
- Preprocess the problem;
- Convert it into CNF;
- Run a saturation algorithm on it, try to derive *false*.
- If *false* is derived, report the result, maybe including a refutation.

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