Automated Deduction

Laura Kovács

Outline

First-Order Theorem Proving

We will use the VAMPIRE theorem prover throughout the lecture.

Go to

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

and pick the route most suitable to you.

Note:

- only running Vampire in the browser at `https://tptp.org/cgi-bin/SystemOnTPTP`

First-Order Theorem Proving

We will use the VAMPIRE theorem prover throughout the lecture.

Go to

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

and pick the route most suitable to you.

Note:

- only running Vampire in the browser at https://tptp.org/cgi-bin/SystemOnTPTP

First-Order Theorem Proving. An Example

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

First-Order Theorem Proving. An 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. An 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

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 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)).
```

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;
- Input formula roles (very important);

```
%---- 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;
- Input formula roles (very important);
- 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) ).
```

First-Order Logic and TPTP

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

First-Order Logic 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 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 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.
- 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 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.
- 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.
- All symbols are uninterpreted, apart from equality $=$.

First-Order Logic 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.
- 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.
- All symbols are uninterpreted, apart from equality $=$.

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`

Running Vampire on a TPTP file

is easy: simply use

```
vampire <filename>
```

Running Vampire on 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 ADuct25 group.tptp
```

Proof by Vampire (Slightly Modified)

```
Refutation found.
251. $false [trivial inequality removal 250]
250. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,159]
159. mult(X0,X1) = mult(X1,X0) [superposition 23,87]
87. mult(X1,mult(X0,X1)) = X0 [forward demodulation 79,25]
79. mult(X1,mult(X0,X1)) = mult(X0,e) [superposition 23,20]
25. mult(X0,e) = X0 [superposition 23,13]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 13,12]
15. mult(X0,mult(X0,X1))=mult(e,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]
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 axiom]
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(conjecture)]
4. ![X0]: e = mult(X0,X0)[input(assumption)]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input(axiom)]
1. ![X0]: mult(e,X0) = X0 [input(axiom)]
```

Proof by Vampire (Slightly Modified)

```
Refutation found.
251. $false [trivial inequality removal 250]
250. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,159]
159. mult(X0,X1) = mult(X1,X0) [superposition 23,87]
87. mult(X1,mult(X0,X1)) = X0 [forward demodulation 79,25]
79. mult(X1,mult(X0,X1)) = mult(X0,e) [superposition 23,20]
25. mult(X0,e) = X0 [superposition 23,13]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 13,12]
15. mult(X0,mult(X0,X1))=mult(e,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]
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 axiom]
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(conjecture)]
4. ![X0]: e = mult(X0,X0)[input(assumption)]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input(axiom)]
1. ![X0]: mult(e,X0) = X0 [input(axiom)]
```

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

Proof by Vampire (Slightly Modified)

```
Refutation found.
251. $false [trivial inequality removal 250]
250. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,159]
159. mult(X0,X1) = mult(X1,X0) [superposition 23,87]
87. mult(X1,mult(X0,X1)) = X0 [forward demodulation 79,25]
79. mult(X1,mult(X0,X1)) = mult(X0,e) [superposition 23,20]
25. mult(X0,e) = X0 [superposition 23,13]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 13,12]
15. mult(X0,mult(X0,X1))=mult(e,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]
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 axiom]
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(conjecture)]
4. ![X0]: e = mult(X0,X0)[input(assumption)]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input(axiom)]
1. ![X0]: mult(e,X0) = X0 [input(axiom)]
```

- 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.
251. $false [trivial inequality removal 250]
250. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,159]
159. mult(X0,X1) = mult(X1,X0) [superposition 23,87]
87. mult(X1,mult(X0,X1)) = X0 [forward demodulation 79,25]
79. mult(X1,mult(X0,X1)) = mult(X0,e) [superposition 23,20]
25. mult(X0,e) = X0 [superposition 23,13]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 13,12]
15. mult(X0,mult(X0,X1))=mult(e,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]
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 axiom]
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(conjecture)]
4. ![X0]: e = mult(X0,X0)[input(assumption)]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input(axiom)]
1. ![X0]: mult(e,X0) = X0 [input(axiom)]
```

- 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.
251. $false [trivial inequality removal 250]
250. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,159]
159. mult(X0,X1) = mult(X1,X0) [superposition 23,87]
87. mult(X1,mult(X0,X1)) = X0 [forward demodulation 79,25]
79. mult(X1,mult(X0,X1)) = mult(X0,e) [superposition 23,20]
25. mult(X0,e) = X0 [superposition 23,13]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 13,12]
15. mult(X0,mult(X0,X1))=mult(e,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]
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 axiom]
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(conjecture)]
4. ![X0]: e = mult(X0,X0)[input(assumption)]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input(axiom)]
1. ![X0]: mult(e,X0) = X0 [input(axiom)]
```

- 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.
251. $false [trivial inequality removal 250]
250. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,159]
159. mult(X0,X1) = mult(X1,X0) [superposition 23,87]
87. mult(X1,mult(X0,X1)) = X0 [forward demodulation 79,25]
79. mult(X1,mult(X0,X1)) = mult(X0,e) [superposition 23,20]
25. mult(X0,e) = X0 [superposition 23,13]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 13,12]
15. mult(X0,mult(X0,X1))=mult(e,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]
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 axiom]
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(conjecture)]
4. ![X0]: e = mult(X0,X0)[input(assumption)]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input(axiom)]
1. ![X0]: mult(e,X0) = X0 [input(axiom)]
```

- 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.
251. $false [trivial inequality removal 250]
250. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,159]
159. mult(X0,X1) = mult(X1,X0) [superposition 23,87]
87. mult(X1,mult(X0,X1)) = X0 [forward demodulation 79,25]
79. mult(X1,mult(X0,X1)) = mult(X0,e) [superposition 23,20]
25. mult(X0,e) = X0 [superposition 23,13]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 13,12]
15. mult(X0,mult(X0,X1))=mult(e,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]
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 axiom]
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(conjecture)]
4. ![X0]: e = mult(X0,X0)[input(assumption)]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input(axiom)]
1. ![X0]: mult(e,X0) = X0 [input(axiom)]
```

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

Proof by Vampire (Slightly Modified)

```
Refutation found.
251. $false [trivial inequality removal 250]
250. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,159]
159. mult(X0,X1) = mult(X1,X0) [superposition 23,87]
87. mult(X1,mult(X0,X1)) = X0 [forward demodulation 79,25]
79. mult(X1,mult(X0,X1)) = mult(X0,e) [superposition 23,20]
25. mult(X0,e) = X0 [superposition 23,13]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 13,12]
15. mult(X0,mult(X0,X1))=mult(e,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]
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 axiom]
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(conjecture)]
4. ![X0]: e = mult(X0,X0)[input(assumption)]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input(axiom)]
1. ![X0]: mult(e,X0) = X0 [input(axiom)]
```

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

Proof by Vampire (Slightly Modified)

```
Refutation found.
251. $false [trivial inequality removal 250]
250. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,159]
159. mult(X0,X1) = mult(X1,X0) [superposition 23,87]
87. mult(X1,mult(X0,X1)) = X0 [forward demodulation 79,25]
79. mult(X1,mult(X0,X1)) = mult(X0,e) [superposition 23,20]
25. mult(X0,e) = X0 [superposition 23,13]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 13,12]
15. mult(X0,mult(X0,X1))=mult(e,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]
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 axiom]
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(conjecture)]
4. ![X0]: e = mult(X0,X0)[input(assumption)]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input(axiom)]
1. ![X0]: mult(e,X0) = X0 [input(axiom)]
```

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

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 $>$ 70 times.

Vampire - The Team at CASC 2025

Colour key: 1st place 2nd place 3rd place Demonstration *Previous winner*

Higher-order Theorems	*Vampire* 4.9	Vampire 5.0	Zipperpin 2.1.9999	E 3.3.0	cvc5 1.3.0	Leo-III 1.7.19
Solved/500	455/500	449/500	415/500	395/500	242/500	242/500
Solutions/500	0/500	449/500	0/500	395/500	0/500	242/500

Typed First-order Theorems +*-/	Vampire 5.0	*Vampire* 4.9	iProver 3.9.3	cvc5 1.3.0
Solved/150	131/150	123/150	74/150	62/150
Solutions/150	131/150	0/150	74/150	0/150

Typed First-order Non-theorems	Vampire 5.0	iProver 3.9.3	*iProver* 3.9	cvc5 1.3.0
Solved/150	110/150	86/150	84/150	29/150
Solutions/150	110/150	86/150	84/150	29/150

First-order Theorems	*Vampire* 4.9	Vampire 5.0	CSI_Enig 1.0.6	iProver 3.9.3	E 3.3.0	Drodi 4.1.0	CSE_E 1.7	Zipperpin 2.1.9999	cvc5 1.3.0	Prover9 1109a	Connect 0.6.1
Solved/500	466/500	455/500	402/500	367/500	364/500	325/500	295/500	267/500	262/500	119/500	102/5
Solutions/500	0/500	455/500	0/500	367/500	364/500	325/500	293/500	0/500	0/500	0/500	0/5

Effectively Propositional CNF	Vampire 5.0	iProver 3.9.3	CSI_Enig 1.0.6	E 3.3.0	Drodi-EP 4.1.0	SPASS-SC 0.1
Solved/200	186/200	168/200	98/200	88/200	80/200	64/200

Unit Equality CNF	Vampire 5.0	*Vampire* 4.9	Twee 2.6.0	CSI_Enig 1.0.6	E 3.3.0	iProver 3.9.3	CSE_E 1.7	Drodi 4.1.0	Toma 0.7
Solved/300	263/300	258/300	258/300	238/300	222/300	205/300	167/300	121/300	114/300
Solutions/300	263/300	0/300	258/300	0/300	222/300	205/300	167/300	121/300	114/300

SLedgeHammer Theorems	*Vampire* 4.9	Vampire 5.0	E 3.3.0	cvc5 1.3.0	Leo-III 1.7.19
Solved/1000	441/1000	434/1000	429/1000	294/1000	38/1000
Solutions/1000	441/1000	434/1000	429/1000	0/1000	36/1000

I Challenge yoU	*Vampire* 4.9	Vampire 5.0	CSI_Enig 1.0.6	E 3.3.0	iProver 3.9.3	Drodi 4.1.0	CSE_E 1.7	ConnectP 0.6.1
Solved/101	70/101	69/101	46/101	42/101	34/101	21/101	18/101	1/101
Solutions/101	0/101	69/101	0/101	42/101	34/101	21/101	18/101	0/101

Vampire - The Team at CAV 2025

Announcing Distinguished Papers

Paolo Baldan, Sebastian Gurke, Barbara König, Tommaso Padoan, Florian Wittbold: ***"Approximating Fixpoints of Approximated Functions"***

Franck van Breugel, Syyeda Zainab Fatmi, Stefan Kiefer, David Parker: ***"Robust Probabilistic Bisimilarity for Labelled Markov Chains"***

Yuheng Su, Qiusong Yang, Yiwei Ci, Tianjun Bu, Ziyu Huang: ***"The rIC3 Hardware Model Checker"***

Nils Froleyks, Emily Yu, Mathias Preiner, Armin Biere, Keijo Heljanko: ***"Introducing Certificates to the Hardware Model Checking Competition"***

Filip Bartek, Ahmed Bhayat, Robin Coutelier, Marton Hajdu, Matthias Hetzenberger, Petra Hozzova, Laura Kovacs, Jakob Rath, Michael Rawson, Giles Reger, Martin Suda, Johannes Schoisswohl, Andrei Voronkov: ***"The Vampire Diary"***

Siddhartha Prasad, Ben Greenman, Tim Nelson, Shriram Krishnamurthi: ***"A Misconception-Driven Adaptive Tutor for Linear Temporal Logic"***

Check for updates

The VAMPIRE Diary

Filip Bártek[1], Ahmed Bhayat[4], Robin Coutelier[3], Márton Hajdu[3], Matthias Hetzenberger[3], Petra Hozzová[1], Laura Kovács[3(✉)], Jakob Rath[3], Michael Rawson[5(✉)], Giles Reger[4], Martin Suda[1(✉)], Johannes Schoisswohl[3], and Andrei Voronkov[2,4(✉)]

[1] Czech Technical University in Prague, Prague, Czech Republic
martin.suda@cvut.cz
[2] EasyChair, Manchester, UK
andrei@voronkov.com
[3] TU Wien, Vienna, Austria
laura.kovacs@tuwien.ac.at
[4] University of Manchester, Manchester, UK
[5] University of Southampton, Southampton, UK
michael@rawsons.uk

CAV Artifact Evaluation ★ Available

CAV Artifact Evaluation ★★★ Reusable

Abstract. During the past decade of continuous development, the theorem prover VAMPIRE has become an automated solver for the combined theories of commonly-used data structures. VAMPIRE now supports arithmetic, induction, and higher-order logic. These advances have been made to meet the demands of software verification, enabling VAMPIRE to effectively complement SAT/SMT solvers and aid proof assistants. We explain how best to use VAMPIRE in practice and review the main changes VAMPIRE has undergone since its last tool presentation, focusing on the engineering principles and design choices we made during this process.