Theorem Proving with Vampire for Rigorous Systems Engineering

Introducing Vampire

Giles Reger and Martin Suda

University of Manchester and TU Wein

Today

Session 1 - Getting started with Vampire

Session 2 - The theory bit

Lunch

Session 3 - The kinds of problems we can solve

Session 4 - What we need to know for loop invariant generation

Today

Session 1 - Getting started with Vampire

Session 2 - The theory bit

Lunch

Session 3 - The kinds of problems we can solve

Session 4 - What we need to know for loop invariant generation

Outline

Setting the Scene

Getting Started with Vampire

Automated Reasoning and Rigorous Systems Engineering

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

In rigorous systems engineering there are lots of places where we can apply this reasoning. For example,

- Proving partial correctness properties
- Generating loop invariants
- Synthesis
- Model checking
- Concolic testing
- Your idea?

Kinds of Reasoning

Given a statement S we can establish different conclusions about it

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

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

	Semantic view	Syntactic view
S is consistent	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

Models and Proofs

Models
A model is a structure that can be used to interpret the symbols in a logical statement making the statement semantically true.
S can have 0, 1, n, or ∞ models. A model may be infinite.
New expressions can be evaluated in a model.
If our statement is of the form $\neg S$ then we have a countermodel.

Proofs (in our context)
A proof is a sequence of derived statements that follow logically from the input, ending in a contradiction.
Steps may preserve validity, satisfiability, or models.
A proof may only use part of S and may introduce new symbols.

Kinds of Reasoners

	Input	Example(s)
SAT Solvers	Propositional formulae	MiniSat
Constraint Solvers	Conjunction of theory constraints	
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.

More about Logics

Propositional Logic

Propositions and boolean constructors e.g. $p \wedge q$, $\text{good} \rightarrow \neg\text{bad}$ Common normal forms (CNF, DNF). Models are assignments of true/false to propositions. SAT solvers. QBF.

First-Order or Predicate Logic

Adds predicates, functions, quantifiers, equality e.g. $\exists x : f(x) \neq x$, $\forall x : \text{man}(x) \rightarrow \text{human}(x)$. Skolemisation can remove $\exists$. Models interpret each predicate and function symbol.

Theories

Fix a class of interpretations for a subset of the signature e.g. $+$.

Higher-Order Logic

Allow quantification over functions. Think simply-typed λ-calculus.

Other Logics (live inside one of the above)

Modal logics (e.g. LTL). Description logics. Seperation Logic.

What is Vampire?

An automated theorem prover for first-order logic and theories

- It produces detailed proofs but also supports finite model finding
- It is very fast (44 trophies from CASC over the last 18 years)
- It competes with SMT solvers on their problems
- In normal operation it is saturation-based - it saturates a clausal form with respect to an inference system
- It is portfolio-based - it works best when you allow it to try lots of strategies
- It has unique proof search features such as LRS and AVATAR
- It supports lots of extra features helpful for rigorous systems engineering

CASC 2017 results

Higher-order Theorems	Satallax 3.2	Leo-III 1.1	Satallax 3.0	LEO-II 1.7.0	Zipperpin 1.1	Isabelle 2016
Solved/500	430/500	382/500	382/500	305/500	179/500	387/500
Solutions	430/500	382/500	375/500	301/500	179/500	0/500

Typed First-order Theorems +*-/	Vampire 4.1	Vampire 4.2	CVC4 ARI-1.5.2	Princess 170717	Zipperpin 1.1
Solved/250	194/250	191/250	188/250	130/250	39/250
Solutions	194/250	191/250	188/250	115/250	39/250

First-order Theorems	Vampire 4.2	Vampire 4.0	E 2.1	CVC4 NAR-1.5.2	iProver 2.6	Leo-III 1.1	lean-nan 1.0	Zipperpin 1.1	Prover9 1109a	iProverM 2.5-0.1	Scavenger EP-0.2
Solved/500	452/500	444/500	381/500	327/500	283/500	211/500	186/500	154/500	140/500	99/500	71/500
Solutions	452/500	440/500	381/500	327/500	279/500	211/500	186/500	154/500	138/500	99/500	71/500

First-order Non-theorems	Vampire SAT-4.1	Vampire SAT-4.2	iProver SAT-2.6	CVC4 SNA-1.5.2	E FNT-2.1	Scavenger EP-0.2
Solved/250	219/250	217/250	175/250	136/250	85/250	12/250
Solutions	217/250	204/250	175/250	136/250	85/250	12/250

Effectively Propositional CNF	iProver 2.6	iProver 2.5	Vampire 4.2	E 2.1	Scavenger EP-0.1	Scavenger EP-0.2
Solved/200	174/200	171/200	168/200	53/200	5/200	4/200

SLedgeHammer Theorems	Vampire SLH-4.2	CVC4 SLH-1.5.2	ET 2.0	E SLH-2.1	Leo-III SLH-1.1	iProver SLH-2.6	Zipperpin SLH-1.1	iProverM 2.5-0.1
Solved/2000	1433/2000	1364/2000	1328/2000	1185/2000	652/2000	519/2000	472/2000	320/2000

Large Theory Batch Problems	Vampire LTB-4.0	Vampire LTB-4.2	MaLARe 0.6	iProver LTB-2.6	E LTB-2.1
Solved/1500	1156/1500	1144/1486	1131/1500	777/1499	683/1499
Solutions	1156/1500	1144/1486	1131/1500	777/1499	683/1499

The Vampire Team

University of Manchester, UK

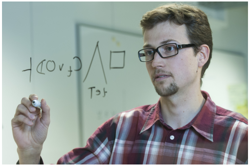
Andrei Voronkov

Giles Reger

- Ahmed Bhayat
- Michael Rawson

TU Wein, Austria

Laura Kovacs



Martin Suda

- Evgenii Kotelnikov
- Simon Robillard
- Bernhard Gleiss
- Andreas Humenberger

Issues to Consider when using Reasoners in Rigorous Systems Engineering

(that we won't necessarily be addressing)

Representing Programs

- How do we encode a program's behaviour in logic
- What abstraction do we want for different types of data
- How do we handle modularity, non-determinism etc

Representing Queries

- Is your query a traditional *does C follow from A*?
- or something else? e.g. which statements among $S_1, \ldots, S_n$ are redundant (implied by other statements)
- How do you turn what you want into a first-order query?

Handling Query Answers

- Is it just a yes/no answer you want?
- Or do you need a model or proof to extract further information from

Outline

Download and Install

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_z3_rel`
- For Windows users, the easiest route for this tutorial is a virtual machine and then use Linux

Trying Vampire

Check out

`https://github.com/vprover/ase17tutorial`

to get material that we're going to be using today.

In this session we will be using problems in intro

We encourage you to try running the things we show you

Hello World in TPTP (hello-world-1.p)

In this example:

- `logic(name,type,formula)` syntax
- Predicates (`hello`) and constants (`world`)
- Logical operators: `=>` but also `|`,`~`,`&`

We have three axioms giving some rules and a conjecture that we want to show follows from these axioms.

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

Hello World in TPTP (`hello-world-1.p`)

In this example:

- `logic(name,type,formula)` syntax
- Predicates (`hello`) and constants (`world`)
- Logical operators: `=>` but also `|`,`~`,`&`

We have three axioms giving some rules and a conjecture that we want to show follows from these axioms.

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

Hello World in TPTP (`hello-world-1.p`)

In this example:

- `logic(name,type,formula)` syntax
- Predicates (`hello`) and constants (`world`)
- Logical operators: `=>` but also `|`,`~`,`&`

We have three axioms giving some rules and a conjecture that we want to show follows from these axioms.

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

Hello World in TPTP (`hello-world-1.p`)

In this example:

- `logic(name,type,formula)` syntax
- Predicates (`hello`) and constants (`world`)
- Logical operators: `=>` but also `|`,`~`,`&`

We have three axioms giving some rules and a conjecture that we want to show follows from these axioms.

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

Hello World in TPTP (`hello-world-1.p`)

In this example:

- `logic(name,type,formula)` syntax
- Predicates (`hello`) and constants (`world`)
- Logical operators: `=>` but also `|`,`~`,`&`

We have three axioms giving some rules and a conjecture that we want to show follows from these axioms.

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

Hello World in TPTP (hello-world-1.p)

In this example:

- logic(name,type,formula) syntax
- Predicates (hello) and constants (world)
- Logical operators: => but also |,~,&

We have three axioms giving some rules and a conjecture that we want to show follows from these axioms.

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

Hello World in TPTP (`hello-world-1.p`)

In this example:

- `logic(name,type,formula)` syntax
- Predicates (`hello`) and constants (`world`)
- Logical operators: `=>` but also `|`,`~`,`&`

We have three axioms giving some rules and a conjecture that we want to show follows from these axioms.

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

```
vampire -av off hello_world-1.p
```

```
10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

```
vampire -av off hello_world-1.p
```

```
10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).

vampire -av off hello_world-1.p

10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).

vampire -av off hello_world-1.p

10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

```
vampire -av off hello_world-1.p
```

```
10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).

vampire -av off hello_world-1.p

10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

```
vampire -av off hello_world-1.p
```

```
10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

```
vampire -av off hello_world-1.p
```

```
10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).

vampire -av off hello_world-1.p

10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

```
vampire -av off hello_world-1.p
```

```
10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World (a proof)

```
fof(a1,axiom,hello(world) => hello(usa)).
fof(a2,axiom,hello(usa) => hello(illinois)).
fof(a3,axiom,hello(illinois) => hello(ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

```
vampire -av off hello_world-1.p
```

```
10. ~hello(world) | hello(usa) [cnf transformation 6]
11. ~hello(usa) | hello(illinois) [cnf transformation 7]
12. ~hello(illinois) | hello(ase) [cnf transformation 8]
13. hello(world) [cnf transformation 9]
14. ~hello(ase) [cnf transformation 9]
15. hello(usa) [resolution 10,13]
16. hello(illinois) [resolution 15,11]
17. hello(ase) [resolution 16,12]
18. $false [subsumption resolution 17,14]
```

Hello World Again, more on TPTP (`hello-world-2.p`)

In this example

- Variables (upper case)
- Quantification (`!` for $\forall$, `?` for $\exists$)

We can rewrite the previous problem to axiomatise the idea that saying hello to something means saying hello to its parts.

```
fof(a1,axiom, ![X,Y] : (
  (has_part(X,Y) & hello(X)) => hello(Y)
)).
fof(f1,axiom, has_part(world,usa)).
fof(f1,axiom, has_part(usa,illinois)).
fof(f1,axiom, has_part(illinois,ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

Hello World Again, more on TPTP (`hello-world-2.p`)

In this example

- Variables (upper case)
- Quantification (`!` for $\forall$, `?` for $\exists$)

We can rewrite the previous problem to axiomatise the idea that saying hello to something means saying hello to its parts.

```
fof(a1,axiom, ![X,Y] : (
  (has_part(X,Y) & hello(X)) => hello(Y)
)).
fof(f1,axiom, has_part(world,usa)).
fof(f1,axiom, has_part(usa,illinois)).
fof(f1,axiom, has_part(illinois,ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

Hello World Again, more on TPTP (hello-world-2.p)

In this example

- Variables (upper case)
- Quantification (! for $\forall$, ? for $\exists$)

We can rewrite the previous problem to axiomatise the idea that saying hello to something means saying hello to its parts.

```
fof(a1,axiom, ![X,Y] : (
  (has_part(X,Y) & hello(X)) => hello(Y)
)).
fof(f1,axiom, has_part(world,usa)).
fof(f1,axiom, has_part(usa,illinois)).
fof(f1,axiom, has_part(illinois,ase)).
fof(con,conjecture,hello(world) => hello(ase)).
```

1+1=2 (one_plus_one.p)

In this example

- cnf form and negated_conjecture
- Equality

We define three constants and two functions. Notice that we're missing some key properties about numbers that are not necessary here (e.g. zero!=succ(X)).

```
cnf(one,axiom, one = succ(zero)).
cnf(two,axiom, two = succ(one)).
cnf(ident,axiom, plus(zero,X) = X).
cnf(plus,axiom, plus(succ(X),Y) = succ(plus(X,Y))).
cnf(oneplusone,negated_conjecture,plus(one,one)!=two).
```

1+1=2 (one_plus_one.p)

In this example

- cnf form and negated_conjecture
- Equality

We define three constants and two functions. Notice that we're missing some key properties about numbers that are not necessary here (e.g. `zero!=succ(X)`).

```
cnf(one,axiom, one = succ(zero)).
cnf(two,axiom, two = succ(one)).
cnf(ident,axiom, plus(zero,X) = X).
cnf(plus,axiom, plus(succ(X),Y) = succ(plus(X,Y))).
cnf(oneplusone,negated_conjecture,plus(one,one)!=two).
```

1+1=2 (one_plus_one.p)

In this example

- cnf form and negated_conjecture
- Equality

We define three constants and two functions. Notice that we're missing some key properties about numbers that are not necessary here (e.g. zero!=succ(X)).

```
cnf(one,axiom, one = succ(zero)).
cnf(two,axiom, two = succ(one)).
cnf(ident,axiom, plus(zero,X) = X).
cnf(plus,axiom, plus(succ(X),Y) = succ(plus(X,Y))).
cnf(oneplusone,negated_conjecture,plus(one,one)!=two).
```

1+1=2 (proof by rewriting)

- cnf form means no clausification
- Superposition
- (forward) Demodulation

```
1. one = succ(zero) [input]
2. succ(one) = two [input]
3. plus(zero,X0) = X0 [input]
4. succ(plus(X0,X1)) = plus(succ(X0),X1) [input]
5. two != plus(one,one) [input]
6. succ(plus(zero,X0)) = plus(one,X0) [superposition 4,1]
8. succ(X0) = plus(one,X0) [forward demodulation 6,3]
10. succ(one) != two [superposition 5,8]
11. $false [subsumption resolution 10,2]
```

1+1=2 (proof by rewriting)

- cnf form means no clausification
- Superposition
- (forward) Demodulation

```
1. one = succ(zero) [input]
2. succ(one) = two [input]
3. plus(zero,X0) = X0 [input]
4. succ(plus(X0,X1)) = plus(succ(X0),X1) [input]
5. two != plus(one,one) [input]
6. succ(plus(zero,X0)) = plus(one,X0) [superposition 4,1]
8. succ(X0) = plus(one,X0) [forward demodulation 6,3]
10. succ(one) != two [superposition 5,8]
11. $false [subsumption resolution 10,2]
```

1+1=2 (proof by rewriting)

- cnf form means no clausification
- Superposition
- (forward) Demodulation

```
1. one = succ(zero) [input]
2. succ(one) = two [input]
3. plus(zero,X0) = X0 [input]
4. succ(plus(X0,X1)) = plus(succ(X0),X1) [input]
5. two != plus(one,one) [input]
6. succ(plus(zero,X0)) = plus(one,X0) [superposition 4,1]
8. succ(X0) = plus(one,X0) [forward demodulation 6,3]
10. succ(one) != two [superposition 5,8]
11. $false [subsumption resolution 10,2]
```

1+1=2 (proof by rewriting)

- cnf form means no clausification
- Superposition
- (forward) Demodulation

```
1. one = succ(zero) [input]
2. succ(one) = two [input]
3. plus(zero,X0) = X0 [input]
4. succ(plus(X0,X1)) = plus(succ(X0),X1) [input]
5. two != plus(one,one) [input]
6. succ(plus(zero,X0)) = plus(one,X0) [superposition 4,1]
8. succ(X0) = plus(one,X0) [forward demodulation 6,3]
10. succ(one) != two [superposition 5,8]
11. $false [subsumption resolution 10,2]
```

1+1=2 (proof by rewriting)

- cnf form means no clausification
- Superposition
- (forward) Demodulation

```
1. one = succ(zero) [input]
2. succ(one) = two [input]
3. plus(zero,X0) = X0 [input]
4. succ(plus(X0,X1)) = plus(succ(X0),X1) [input]
5. two != plus(one,one) [input]
6. succ(plus(zero,X0)) = plus(one,X0) [superposition 4,1]
8. succ(X0) = plus(one,X0) [forward demodulation 6,3]
10. succ(one) != two [superposition 5,8]
11. $false [subsumption resolution 10,2]
```

1+1=2 (proof by rewriting)

- cnf form means no clausification
- Superposition
- (forward) Demodulation

```
1. one = succ(zero) [input]
2. succ(one) = two [input]
3. plus(zero,X0) = X0 [input]
4. succ(plus(X0,X1)) = plus(succ(X0),X1) [input]
5. two != plus(one,one) [input]
6. succ(plus(zero,X0)) = plus(one,X0) [superposition 4,1]
8. succ(X0) = plus(one,X0) [forward demodulation 6,3]
10. succ(one) != two [superposition 5,8]
11. $false [subsumption resolution 10,2]
```

Celsius to Fahrenheit (MSC023=2.p)

- tff form and type definition
- Typed quantification
- built-in theory symbols

```
tff(convertt,type,convert: ( $real * $real ) > $o ).

tff(convert,axiom,(
  ! [C: $real,F: $real] :
  ( $sum($product(1.8,C),32.0) = F => convert(C,F) )
)).

tff(fahrenheit_451_to_celsius,conjecture,(
    ? [C: $real] : convert(C,451.0) )).
```

Celsius to Fahrenheit (`MSC023=2.p`)

- tff form and type definition
- Typed quantification
- built-in theory symbols

```
tff(convertt,type,convert: ( $real * $real ) > $o ).

tff(convert,axiom,(
  ! [C: $real,F: $real] :
  ( $sum($product(1.8,C),32.0) = F => convert(C,F) )
)).

tff(fahrenheit_451_to_celsius,conjecture,(
    ? [C: $real] : convert(C,451.0) )).
```

Celsius to Fahrenheit (MSC023=2.p)

- tff form and type definition
- Typed quantification
- built-in theory symbols

```
tff(convertt,type,convert: ( $real * $real ) > $o ).

tff(convert,axiom,(
  ! [C: $real,F: $real] :
  ( $sum($product(1.8,C),32.0) = F => convert(C,F) )
)).

tff(fahrenheit_451_to_celsius,conjecture,(
    ? [C: $real] : convert(C,451.0) )).
```

Celsius to Fahrenheit (MSC023=2.p)

- tff form and type definition
- Typed quantification
- built-in theory symbols

```
tff(convertt,type,convert: ( $real * $real ) > $o ).

tff(convert,axiom,(
  ! [C: $real,F: $real] :
  ( $sum($product(1.8,C),32.0) = F => convert(C,F) )
)).

tff(fahrenheit_451_to_celsius,conjecture,(
    ? [C: $real] : convert(C,451.0) )).
```

Celsius to Fahrenheit (proof)

- Equality resolution
- Constrained resolution
- Evaluation
- We get a solution (see Question Answering later)

```
./vampire MSC023=2.p -uwa all
```

```
23. convert(X0,X1) | $sum($product(1.8,X0),32.0) != X1
24. ~convert(X0,451.0)
26. convert(X0,$sum($product(1.8,X0),32.0)) [eq res 23]
27. convert(X0,$sum(32.0,$product(1.8,X0))) [demod 26,4]
64. $sum(32.0,$product(1.8,X0)) != 451.0 [con res 27,24]
76. $quotient($sum(451.0,-32.0),1.8) != X0 [evaluation 64]
77. $quotient(419.0,1.8) != X0 [evaluation 76]
78. 232.778 != X0 [evaluation 77]
86. $false [equality resolution 78]
```

Celsius to Fahrenheit (proof)

- Equality resolution
- Constrained resolution
- Evaluation
- We get a solution (see Question Answering later)

```
./vampire MSC023=2.p -uwa all
```

```
23. convert(X0,X1) | $sum($product(1.8,X0),32.0) != X1
24. ~convert(X0,451.0)
26. convert(X0,$sum($product(1.8,X0),32.0)) [eq res 23]
27. convert(X0,$sum(32.0,$product(1.8,X0))) [demod 26,4]
64. $sum(32.0,$product(1.8,X0)) != 451.0 [con res 27,24]
76. $quotient($sum(451.0,-32.0),1.8) != X0 [evaluation 64]
77. $quotient(419.0,1.8) != X0 [evaluation 76]
78. 232.778 != X0 [evaluation 77]
86. $false [equality resolution 78]
```

Celsius to Fahrenheit (proof)

- Equality resolution
- Constrained resolution
- Evaluation
- We get a solution (see Question Answering later)

```
./vampire MSC023=2.p -uwa all
```

```
23. convert(X0,X1) | $sum($product(1.8,X0),32.0) != X1
24. ~convert(X0,451.0)
26. convert(X0,$sum($product(1.8,X0),32.0)) [eq res 23]
27. convert(X0,$sum(32.0,$product(1.8,X0))) [demod 26,4]
64. $sum(32.0,$product(1.8,X0)) != 451.0 [con res 27,24]
76. $quotient($sum(451.0,-32.0),1.8) != X0 [evaluation 64]
77. $quotient(419.0,1.8) != X0 [evaluation 76]
78. 232.778 != X0 [evaluation 77]
86. $false [equality resolution 78]
```

Celsius to Fahrenheit (proof)

- Equality resolution
- Constrained resolution
- Evaluation
- We get a solution (see Question Answering later)

```
./vampire MSC023=2.p -uwa all
```

```
23. convert(X0,X1) | $sum($product(1.8,X0),32.0) != X1
24. ~convert(X0,451.0)
26. convert(X0,$sum($product(1.8,X0),32.0)) [eq res 23]
27. convert(X0,$sum(32.0,$product(1.8,X0))) [demod 26,4]
64. $sum(32.0,$product(1.8,X0)) != 451.0 [con res 27,24]
76. $quotient($sum(451.0,-32.0),1.8) != X0 [evaluation 64]
77. $quotient(419.0,1.8) != X0 [evaluation 76]
78. 232.778 != X0 [evaluation 77]
86. $false [equality resolution 78]
```

Celsius to Fahrenheit (proof)

- Equality resolution
- Constrained resolution
- Evaluation
- We get a solution (see Question Answering later)

```
./vampire MSC023=2.p -uwa all
```

```
23. convert(X0,X1) | $sum($product(1.8,X0),32.0) != X1
24. ~convert(X0,451.0)
26. convert(X0,$sum($product(1.8,X0),32.0)) [eq res 23]
27. convert(X0,$sum(32.0,$product(1.8,X0))) [demod 26,4]
64. $sum(32.0,$product(1.8,X0)) != 451.0 [con res 27,24]
76. $quotient($sum(451.0,-32.0),1.8) != X0 [evaluation 64]
77. $quotient(419.0,1.8) != X0 [evaluation 76]
78. 232.778 != X0 [evaluation 77]
86. $false [equality resolution 78]
```

Celsius to Fahrenheit (proof)

- Equality resolution
- Constrained resolution
- Evaluation
- We get a solution (see Question Answering later)

```
./vampire MSC023=2.p -uwa all
```

```
23. convert(X0,X1) | $sum($product(1.8,X0),32.0) != X1
24. ~convert(X0,451.0)
26. convert(X0,$sum($product(1.8,X0),32.0)) [eq res 23]
27. convert(X0,$sum(32.0,$product(1.8,X0))) [demod 26,4]
64. $sum(32.0,$product(1.8,X0)) != 451.0 [con res 27,24]
76. $quotient($sum(451.0,-32.0),1.8) != X0 [evaluation 64]
77. $quotient(419.0,1.8) != X0 [evaluation 76]
78. 232.778 != X0 [evaluation 77]
86. $false [equality resolution 78]
```

Celsius to Fahrenheit (proof)

- Equality resolution
- Constrained resolution
- Evaluation
- We get a solution (see Question Answering later)

```
./vampire MSC023=2.p -uwa all
```

```
23. convert(X0,X1) | $sum($product(1.8,X0),32.0) != X1
24. ~convert(X0,451.0)
26. convert(X0,$sum($product(1.8,X0),32.0)) [eq res 23]
27. convert(X0,$sum(32.0,$product(1.8,X0))) [demod 26,4]
64. $sum(32.0,$product(1.8,X0)) != 451.0 [con res 27,24]
76. $quotient($sum(451.0,-32.0),1.8) != X0 [evaluation 64]
77. $quotient(419.0,1.8) != X0 [evaluation 76]
78. 232.778 != X0 [evaluation 77]
86. $false [equality resolution 78]
```

The same in SMTLIB (MSC023=2.smt2)

- SMTLIB is a Lisp-like syntax
- We only support declare/assert commands
- Although support for incrementality is being added

```
(declare-fun convert (Real Real) Bool)
(assert (forall ((c Real) (f Real))
        (=> (= (+ (* 1.8 c) 32.0) f) (convert c f))))
(assert (not (exists ((c Real)) (convert c 451.0))))
```

```
./vampire -uwa all --input_syntax smtlib2 MSC023=2.smt2
```

An Aside: TPTP and SMT Infrastructure

Let's look at:

- `http://www.cs.miami.edu/~tptp/`
- `http://www.cs.miami.edu/~tptp/casc`
- `http://smtlib.cs.uiowa.edu/`
- `http://smtcomp.sourceforge.net/`

These pages are good starting points for finding out about languages and solvers for first-order problems.

Task

Use the TPTP or SMTLIB2 language to write some simple statements and use Vampire to try and prove them.

For example,

- Properties of sets (see `intersect_in_union.p`)
- A puzzle (see `PUZ001+1.p`)
- Some standard results from group theory (see `group.p`)

you can also try modifying the problems you've been given.

You might observe some strange things - hopefully those strange things will have been explained by Lunch.