Vampire Usage and Demo

Krystof Hoder
Laura Kovacs
Andrei Voronkov

http://vprover.org/

Vampire modes

- 'Vampire' mode
  - uses a single specified strategy
- **CASC mode** (`--mode casc`)
  - selects best strategy based on problem characteristics
- LTB mode (`--mode casc_ltb`)
  - like CASC, allows solving multiple problems sharing large amounts of axioms
- Clausify (`--mode clausify`)
  - converts problem to CNF and outputs
- Axiom selection (`--mode axiom_selection`)
  - outputs axioms selected by Sine selection
- Grounding (`--mode grounding`)
  - performs grounding of EPR problems
- Consequence elimination (`--mode consequence_elimination`)
  - given set of claims, searches for relations between them

CASC Mode

- Usually the best for proving theorems
- First scan problem to determine characteristics
  - Unit, EPR, Horn, equality, large
- Then assign problem into one class
  - currently 43 classes
- Each class has a sequence of strategies that should solve problems in it
- Obtaining the strategies
  - run random strategies on a cluster of computers
  - take the best performing ones and try to further improve by doing slight changes
  - optimization techniques find the best sequence

LTB Mode

- Strategy selection like in CASC mode
- Input is a batch file according to CASC LTB specification
- First parse shared axioms
- Then add them into each of the problems
  - save on expensive parsing
- Supports multiprocessing
  - running multiple strategies in parallel

```
% SZS start BatchConfiguration
division.category LTB.SMO
output.required Assurance
output.desired Proof Answer
limit.time.problem.wc 60
% SZS end BatchConfiguration
% SZS start BatchIncludes
include('Axioms/CSR003+2.ax').
include('Axioms/CSR003+5.ax').
% SZS end BatchIncludes
% SZS start BatchProblems
/TPTP/Problems/CSR/CSR083+3.p /outputs/CSR083+3
/TPTP/Problems/CSR/CSR075+3.p /outputs/CSR075+3
/TPTP/Problems/CSR/CSR082+3.p /outputs/CSR082+3
/TPTP/Problems/CSR/CSR086+3.p /outputs/CSR086+3
/TPTP/Problems/CSR/CSR091+3.p /outputs/CSR091+3
/TPTP/Problems/CSR/CSR092+3.p /outputs/CSR092+3
% SZS end BatchProblems
```

Axiom Selection Mode

- Takes and outputs TPTP formulas/CNF
- Can be used as filter

  cat big_problem.tptp | vampire --mode axiom_selection | other_tool

- Performs Sine axiom selection
- Supports the Sine options (see CADE paper)

  `--sine_tolerance` (float >=1)

  `--sine_depth` (0,1,…)

Clausify Mode

- Converts TPTP formulas problem to CNF
  - supports typed formulas, arithmetic, answer literals
- Allows application of various Vampire preprocessing rules
  - axiom selection, transforming predicate definitions (inlining, merging, removing unused), naming, splitting,...

Grounding mode

- Converts EPR problem into propositional
- Input TPTP, output DIMACS
- Use splitting to reduce amount of variables in clauses (and therefore number of generated propositional clauses)

```
fof(a1,axiom, p(X,X)).
fof(a2,axiom, p(X,Y) => p(Y,X)).
fof(a3,axiom, p(a,b)).
fof(a3,axiom, ~p(b,c)).
```

```
p cnf 9  14
% 1: p(c,c)
% 2: p(b,b)
% 3: p(a,a)
% 4: p(c,b)
% 5: p(b,c)
% 6: p(c,a)
% 7: p(a,c)
% 8: p(b,a)
% 9: p(a,b)
% Grounding p(X0,X1) | ~p(X1,X0)
3 -3 0
9 -8 0
7 -6 0
8 -9 0
2 -2 0
5 -4 0
6 -7 0
4 -5 0
1 -1 0
% 9: p(a,b)
% Grounding p(a,b)
9 0
% 5: p(b,c)
% Grounding ~p(b,c)
-5 0
% 1: p(c,c)
% 2: p(b,b)
% 3: p(a,a)
% Grounding p(X0,X0)
3 0
2 0
1 0
0
```

Consequence Elimination Mode

- Given a set of claims (possibly with underlying theory), attempts to discover which claims follow from others

```
fof(c1, claim, a=>b).
fof(c2, claim, b=>c).
fof(c3, claim, a=>c).
```

```
# vampire --mode consequence_elimination
Pure cf clause: c2 | c1
Pure cf clause: ~c1 | c3 | ~c2
Consequence found: c3
```

clauses stating relations between claims:

c2 | c1

- both c1 and c2 cannot be false

¬c1 | c3 | ¬c2

- can be written as c3 :- c1, c2

c3 is a consequence of other claims

API

- Vampire has an API for building, manipulating, preprocessing and clausifying formulas

```
FormulaBuilder api;

Var xv = api.var("Var");
Term x = api.varTerm(xv);
Predicate p=api.predicate("p",1);
Predicate q=api.predicate("q",0);

Formula fpx=api.formula(p,x);
Formula fq=api.formula(q);
Formula fQpx=api.formula(FormulaBuilder::FORALL, xv, fpx);
Formula fQpxOq=api.formula(FormulaBuilder::OR, fQpx, fq);

AnnotatedFormula af=api.annotatedFormula(fQpxOq,FormulaBuilder::CONJECTURE, ”conj1");
Problem prb;
prb.addFormula(af);
prb.output(cout);

Problem cprb=prb.clausify(0,false,Problem::INL_OFF,false);
cprb.output(cout);
```

fof(conj1,conjecture,
((![Var] : (p(Var))) | q)).

cnf(conj1_2,negated_conjecture,
~p(sK0_Var)).

cnf(conj1_1,negated_conjecture,
~q).

Solution Output

- Proof
  - may use TPTP format
- Interpolant (see Session 3)
- Answer
  - for existentially quantified conjectures
- Model
  - currently only for certain strategies on EPR problems

Proofs

2_01_proof_ex.tptp:
cnf(commutativity, axiom, **f(X,Y)=f(Y,X)**).
cnf(identity, axiom, **f(i,X)=X**).
fof(c, conjecture, **(! [X]: f(j,X)=X) => j=i**).

22. **$false** (2:0) [subsumption resolution 16,7]
7. **i != j** (0:3) [cnf transformation 5]
5. **! [X0] : f(j,X0) = X0 & i != j**[ennf transformation 4]
4. **~(! [X0] : f(j,X0) = X0 => i = j)**[negated conjecture 3]
3. **! [X0] : f(j,X0) = X0 => i = j**[input]
16. **i = j** (2:3) [superposition 8,2]
2. **f(i,X0) = X0** (0:5) [input]
8. **f(X0,j) = X0** (1:5) [superposition 1,6]
6. **f(j,X0) = X0** (0:5) [cnf transformation 5]
1. **f(X0,X1) = f(X1,X0)** (0:7) [input]

```
fof(f22,plain,(
  $false),
  inference(subsumption_resolution,[],[f16,f7])).
fof(f7,plain,(
  i != j),
  inference(cnf_transformation,[],[f5])).
fof(f5,plain,(
  ! [X0] : f(j,X0) = X0 & i != j),
  inference(ennf_transformation,[],[f4])).
fof(f4,negated_conjecture,(
  ~(! [X0] : f(j,X0) = X0 => i = j)),
  file('PROBLEM3.p',unknown)).
fof(f3,axiom,(
  ! [X0] : f(j,X0) = X0 => i = j),
  file('PROBLEM3.p',unknown)).
fof(f16,plain,(
  i = j),
  inference(superposition,[],[f8,f2])).
fof(f2,axiom,(
  ( ! [X0] : (f(i,X0) = X0) )),
  file('PROBLEM3.p',unknown)).
fof(f8,plain,(
  ( ! [X0] : (f(X0,j) = X0) )),
  inference(superposition,[],[f1,f6])).
fof(f6,plain,(
  ( ! [X0] : (f(j,X0) = X0) )),
  inference(cnf_transformation,[],[f5])).
fof(f1,axiom,(
  ( ! [X0,X1] : (f(X0,X1) = f(X1,X0)) )),
  file('PROBLEM3.p',unknown)).
```

Proofs

Vampire native format:

11. ~female(X0) | ~from_venus(X0) | truthteller(X0) (0:6) [input]

48_2. $false | (~$bdd4 & ($bdd3 & $bddnode1)) (2:0) [merge 48_3,107_1]

BDD definition: $bddnode1 = ($bdd2 ? $bdd1 : ~$bdd1)

TPTP proof format:

```
fof(f11,axiom,(
 ( ! [X0] : (~female(X0) | ~from_venus(X0) | truthteller(X0)) )),
 file('Problems/PUZ/PUZ007-1.p',unknown)).
fof(f48_2,plain,(
 $false | ( ( $bdd4 => $false) & ( ~$bdd4 => ( ( $bdd3 => ( ( $bdd2 => $bdd1) & ( ~$bdd2 => ~$bdd1 ) )) & ( ~$bdd3 => $false ) ) ) )),
 inference(merge,[],[f48_3,f107_1])).
```

LaTeX output:

[11, input]

$$\neg \mathit{female}() \vee \neg \mathit{from\_venus}() \vee \mathit{truthteller}()$$

$[48_3, 107_1 \to 48_2, \text{merge}]$

$$\frac{\square \vee n_1 \qquad \square \vee (\neg b_4 \vee b_1)}{\square \vee (\neg b_4 \wedge (b_3 \wedge n_0))}$$

$$n_0 \leftrightarrow (b_2 ? b_1 : \neg b_1)$$
$$n_1 \leftrightarrow (b_4 ? (\neg b_3 \wedge (\neg b_2 \wedge \neg b_1)) : (b_3 \wedge n_0))$$

Question Answering

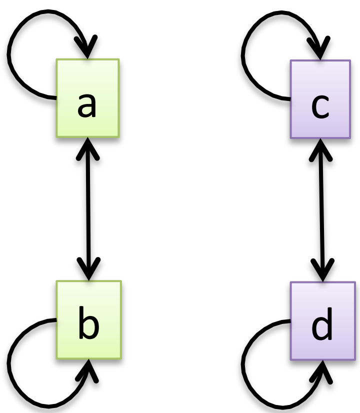
```
2_02_answer_ex.tptp:
fof(a1,axiom,son("jimmy","jane")).
fof(a2,axiom,son("johny","jane")).
fof(a3,axiom, (son(X,Z) & son(Y,Z) & X!=Y) => brother(X,Y)).
fof(q,question, ?[X] : brother("jimmy", X)).
```

```
# vampire PROBLEM.p -question_answering answer_literal
% SZS answers Tuple [["johny"]|_] for PROBLEM2
23. $false (0:0) [unit resulting resolution 22,21]
21. ~sP0_ans("johny") (1:2) [resolution 20,15]
15. ~brother("jimmy",X0) | ~sP0_ans(X0) (0:5) [cnf transformation 10]
10. ! [X0] : (~sP0_ans(X0) | ~brother("jimmy",X0))[ennf transformation 6]
6. ~? [X0] : (sP0_ans(X0) & brother("jimmy",X0))[answer literal 5]
5. ~? [X0] : brother("jimmy",X0)[negated conjecture 4]
4. ? [X0] : brother("jimmy",X0)[input]
20. brother("jimmy","johny") (0:3) [distinct equality removal 19]
19. "jimmy" = "johny" | brother("jimmy","johny") (2:6) [resolution 16,13]
…
```

Model Output

2_03_model_ex.tptp:

```
fof(a1,axiom, p(X,X)).
fof(a2,axiom, p(X,Y) => p(Y,X)).
fof(a3,axiom, (p(X,Y) & p(Y,Z)) => p(X,Z)).
fof(a4,axiom, p(a,b)).
fof(a5,axiom, p(c,d)).
fof(a6,axiom, ~p(b,c)).
```




```
# vampire PROBLEM.p -sa inst_gen -updr off
Refutation not found!
fof(model1,interpretation_domain,
   ! [X] : ( X = "d" | X = "c" | X = "b" | X = "a" ) ).
fof(model2,interpretation_terms,
   ( b = "b" & d = "d" & a = "a" & c = "c") ).
fof(model3,interpretation_atoms,
   ( p("c","d") &
    p("d","c") &
    ~p("b","d") &
    ~p("d","b") &
    ~p("b","c") &
    ~p("c","b") &
    p("d","d") &
    p("c","c") &
    p("a","b") &
    ~p("a","c") &
    ~p("c","a") &
    p("b","b") &
    p("a","a") &
    ~p("a","d") &
    ~p("d","a") &
    p("b","a") ) ).
```

Input Language

- Sorts

```
tff(list_type,type,(
   list: $tType )).
tff(nil_type,type,(
   nil: list )).
tff(cons_type,type,(
   cons: ( $int * list ) > list )).
```

- If-then-else (both for terms and formulas)

```
tff(c1,axiom,  $itef(p & q, ~p|~q, p & q) )
        sP0 <=> (p & q)
        ((p & q) & ~sP0) | ((~p | ~q) & sP0)

tff(c2,axiom, $itet(p,a,b) != a  &  p  ).
        $itef(p,sG0(X0,X1) = X0,sG0(X0,X1) = X1)
        sG0(a,b) != a  &  p
```

Input Language

- Let…in
  - inside terms or formulas
  - assigning to functions or predicates

tff(c1,axiom, **$lettt(f(X),g(X),f(a))** != g(a)).

g(a) != g(a)

tff(c2,axiom, **$letff(p(X), q(X)|r(X), p(c))** & ~q(c) & ~r(c)).

(**q(c) | r(c)**) & ~q(c) & ~r(c)

tff(c3,axiom, **$lettf(f(X), g(X), p(f(X)))** & ~p(g(X)))

! [X1] : (**p(g(X1)**) & ~p(g(X1)))

tff(c4,axiom, **$letft(p(X),q,$itet(p(a),a,b)**) != $itet(q,a,b)).

$itef(q,sG0(X0,X1) = X0,sG0(X0,X1) = X1)

$itef(q,sG1(X0,X1) = X0,sG1(X0,X1) = X1)

sG0(a,b) != sG1(a,b)

Arithmetic

- TFA arithmetic syntax specified in the TPTP standard
  - integers, rationals, reals
- Currently we
  - add axioms for the interpreted symbols present in the problem
  - evaluate interpreted expressions with numeric arguments
    - e.g. 10<5+3 --> 10<8 --> ⊥

2_04_arith_ex.tptp:

```
tff(f_type,type,(
   f: $int > $int )).

tff(integers,axiom,
  ?[Y:$int] : ![X:$int] : ( f(X)=$sum(X,Y) ) ).

tff(integers,conjecture,
  ![X:$int,Y:$int] : ( $less(f(X),f(Y)) <=> $less(X,Y) ) ).
```

2_05_arith_answer.tptp:

```
tff(integers,question,
  ?[X:$int] : ( $product(X,X)=$sum(X,X) & X!=0 )).

% SZS status Theorem for alt_2_05_arith_answer_ex
% SZS answers Tuple [[2]|_] for alt_2_05_arith_answer_ex
% SZS output start Proof for alt_2_05_arith_answer_ex
450. $false (0:0) [unit resulting resolution 449,448]
448. ~sP0_ans(2) (0:2) [distinct equality removal 447]
447. 0 = -1 | ~sP0_ans(2) (8:5) [trivial inequality removal 446]
446. 4 != 4 | 0 = -1 | ~sP0_ans(2) (8:8) [evaluation 445]
445. $product(2,2) != $uminus(-4) | 0 = -1 | ~sP0_ans(2) (8:11)
  [evaluation 444]
444. $product($uminus(-2),$uminus(-2)) != $uminus($sum(-2,-2))
  | $sum(1,-2) = 0 | ~sP0_ans($uminus(-2)) (8:18) [evaluation 443]
…
```

Preprocessing

- Eliminate if-then-else and let…in terms and formulas
- Sine selection
- Predicate definitions and EPR
  - Skolemization of definitions such as "p(X) <=> F[X]" introduces non-constant functions
  - if all occurrences of p(X) are ground, this is not necessary
  - blind inlining may be infeasible (exponential blow-up)
  - Vampire has several rules to deal with this situation
- Removal of trivial predicates
  - E.g. "p(X) | ~p(b)" "p(a)"
- Equivalent predicate discovery, naming, splitting, detecting Horn structure,…

Strategies

- Saturation (Discount, Otter, LRS)
  - splitting (backtracking, without backtracking)
  - BDDs (to represent propositional predicates)
  - global subsumption resolution
  - unit-resulting resolution
- Tabulation
- Instantiation
  - InstGen calculus
- Instantiation and saturation can run in parallel
  - saturation clauses are used in the InstGen literal selection
  - global subsumption resolution indexes are shared

New symbol introduction

- Some Vampire rules may introduce new symbols
  - in certain applications (interpolation) this is not desirable
  - some such rules cannot be disabled (skolemization), other can
- BDDs (introducing prop. predicates for BDD variables)

  `--forced_options propositional_to_bdd=off`
- Splitting (introducing prop. predicates for decision points)

  `--forced_options splitting=off`
- Other rules

  equality_proxy, general_splitting, inequality_splitting
- Naming introduces new predicates to avoid exponential blow-up during clausification
  - setting naming to larger values will lead to less introduced names, 0 disables it

    `naming=32000`
    `naming=0`
    `naming=8` (default)
- To disable all of the above

```
--forced_options
propositional_to_bdd=off:splitting=off:equality_proxy=off:
general_splitting=off:inequality_splitting=0:naming=0
```

Overview

Usage
Single strategy
CASC mode
LTB
Clausifier
Axiom selection
Consequence elimination
Grounding
API

Supported input
Sorts
Arithmetic
If-then-else
Let...in

Solution output
Proof
Interpolant
Answer
Model

Solving strategies
Saturation
Tabulation
InstGen

Preprocessing
Sine selection
EPR restoring
Trivial predicate removal
Inlining definitions
…

http://vprover.org/