Formal Method Mod. 1 (Automated Reasoning)

Laboratory 3

Giuseppe Spallitta
giuseppe.spallitta@unitn.it

Università degli studi di Trento

March 31, 2021

Outline

1. Automatic theorem provers for first-order logic
   Quick overview on E
2. Getting used with E
3. Simple real-life applications
4. Homework

E Theorem prover

- E is a theorem prover for full first-order logic with equality.
- It accepts a problem specification, typically consisting of a number of first-order clauses or formulas, and a conjecture.
- The system tries to find a formal proof for the conjecture, assuming the axioms.
- Accessible from the following link: `https://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html`

E input format: TPTP

- If we want to use E, we need to know the **input** format and the **output** provided by the tool.
- The input format accepted by the tool is called **TPTP format**.

General structure of TPTP file

A typical TPTP benchmark file is characterized by the following sections:

- The *Header* section
- The *Include* section
- The *Formulae* section

DFG file: header

- The header contains several comment lines (starting with %) describing the parsed problem.
- It is not mandatory, but it is strongly suggested to provide some minimal information:

```
% Name of the file
% Description of the problem
%
% Basic sintax information (n. of constants,
%    formulae and predicates involved)
```

DFG file: include

- Sometimes you have the opportunity to reuse axioms for multiple problems (i.e. properties of mathematical theories).
- Instead of copy-pasting the same amount of lines multiple time, you can create some additional file collecting some axioms and the call them using a single line:

```
include('path/to/file')
```

DFG file: formulae

- A variable number of lines containing formulae describing the problem.
- You can represent them in CNF format or (easier) in a FOL-like representation. You can mix them in the same file!

```
fof(<name>,<role>,<logic-formula>).
cnf(<name>,<role>,<logic-formula>).
```

Warning

The dot at the end of each formula is **mandatory**!

DFG formulae: names and roles

- Names identify the formula representing a property of our system, so be sure to give meaningful names.
- Roles determine the gives the user semantics of the formula.
  - Multiple `axiom` formulae can be implemented
  - A single `conjecture` or `negated_conjecture` can be provided and it will represent the core of the problem (the prover will try to find a proof to it).

DFG formulae: logic representation

Logic formulae can represented using a combination of Boolean operators and quantifiers:

- NEGATION is represented as ~
- OR is represented as |
- AND is represented as &
- NOR and NAND can be represented as ~| and ~& respectively
- IF is represented as => or <=, depending on the element that implies the other.
- IFF is represented as <=>
- XOR can be represented as <~>
- EQUALITY is represented as =
- DISEQUALITY is represented as !=

DFG formulae: logic representation (cont.d)

Logic formulae can represented using a combination of Boolean operators and quantifiers:

- The universal and existential quantifiers are represented respectively as `!` and `?`.
- The structure of a quantified formula is:

```
<Quantifier> [<Quantified variables>] : Formula
```

For instance, to represent the logical formula "There exists A so that it is equal to f(1)":

```
? [A] : (A = f(1))
```

DFG formulae: precedence

- Negation has higher precedence than quantification, which in turn has higher precedence than the binary connectives.
- No precedence is specified between the binary connectives; brackets are used to ensure the correct association.
- The binary connectives are left associative.

E output

Once you create your file using the TPTP format, you can feed it to the theorem prover using the command *./eprover –auto -s file.p*:

- If a proof is found, then the output string will contain the sentence *# Proof found!*
- In the case the prover cannot find a proof, we either can obtain: *# No proof found!* or a failure message: *# Failure: Resource limit exceeded*

If we want to see the output that generated the proof we must add the option *–proof-object*.

Outline

1. Automatic theorem provers for first-order logic
2. Getting used with E
3. Simple real-life applications
4. Homework

First encodings

UNIVERSITÀ DEGLI STUDI DI TRENTO

Exercise 3.1: socrates

Assume that out of the two sentences:

- Sokrates is a human.
- All humans are mortal.

you want to conclude that "Sokrates is mortal". Can you prove it using the E theorem prover?

Encoding step-by-step

To encode a problem and feed it to the prover, we must follow this procedure:

- Convert the problem from it's domain language to (first-order) logic.
- Create a file of the the axioms and conjecture using TPTP format.

First encodings: FOL formulation

Reading exercise 3.1, there are two axioms and one conjecture to encode:

1. (A) Human(sokrates)
2. (A) $\forall x.(Human(x) \Rightarrow Mortal(x))$
3. (C) Mortal(sokrates)

These sentences are easy to encode, so we can quickly write the final problem and feed it to the solver.

First encodings: results

- Now we can feed the encoding into E
  ⇒ The prover returns **Proof found!**, so a proof exists.
- We can output the entire proof using the command option *–proof-object* and see the intermediate steps.
  - The output syntax (*SZSOntology*) could seem complex to read, but we can easily grasp the main idea behind the proof.

Outline

1. Automatic theorem provers for first-order logic
2. Getting used with E
3. Simple real-life applications
4. Homework

Private investigations

UNIVERSITÀ DEGLI STUDI DI TRENTO

Exercise 3.2: who killed Agatha?

Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Aunt Agatha hates. No one hates everyone. Agatha is not the butler. Therefore : Agatha killed herself.

Private investigations: variables

First we must define constants, predicates and functions that efficiently describe the problem:

- **Constants**: agatha, butler, charles
- **Predicates/Functions**: lives(X), killed(X,Y), hates(X,Y), richer(X,Y)

Private investigations: FOL formulae (1)

UNIVERSITÀ DEGLI STUDI DI TRENTO

- Someone who lives in Dreadbury Mansion killed Aunt Agatha:
  $\exists x.(lives(x) \wedge killed(x, agatha))$
- Agatha, the butler, and Charles live in Dreadbury Mansion...
  $lives(agatha)$
  $lives(butler)$
  $lives(charles)$
- ... and are the only people who live therein.
  $\forall x.(lives(x) \rightarrow (x = agatha \vee x = butler \vee x = charles))$

Private investigations: FOL formulae (2)

- A killer always hates his victim...
  $\forall xy.(killed(x,y) \rightarrow hates(x, y))$
- and is never richer than his victim.
  $\forall xy.(killed(x,y) \rightarrow \neg\, richer(x, y))$
- Charles hates no one that Aunt Agatha hate.
  $\forall x.(hates(agatha, x) \rightarrow \neg\, hates(charles, x)))$

Private investigations: FOL formulae (3)

- Agatha hates everyone except the butler.
  $\forall x.(x \mathrel{!=} butler \rightarrow (hates(agatha, x))$
- The butler hates everyone not richer than Aunt Agatha $\forall x.(\neg richer(x, agatha) \rightarrow (hates(butler, x))$
- The butler hates everyone Aunt Agatha hates.
  $\forall x.(hates(agatha, x) \rightarrow hates(butler, x)))$

Private investigations: FOL formulae (4)

- No one hates everyone.
  $\forall x exists y.(\neg\ hates(x,y))$
- Agatha is not the butler.
  agatha != butler
- CONJECTURE: Agatha killed herself. *killed(agatha, butler)*

Private investigations: results

- Now we can feed the encoding into E
  $\Rightarrow$ The prover returns **Proof found!**, so a proof exists.

Task manager

Exercise 3.3: task manager

Your PC needs to complete 5 different tasks (A,B,C,D and E) to correctly save a file. There are some constraints about the order execution of the tasks:

- We can execute A after D is completed.
- We can execute B after C and E are completed.
- We can execute E after B or D are completed.
- We can execute C after A is completed.

Which is the task that will execute for last?

Task manager: variables

First we must define constants, predicates and functions that efficiently describe the problem:

- **Constants**: A, B, C, D, E
- **Predicates/Functions**: before(x,y)

Task manager: FOL formulae (1)

UNIVERSITÀ DEGLI STUDI DI TRENTO

For each clue we can build a FOL formula representing the priority:

- We can execute A after D is completed.
  before(D,A)
- We can execute B after C and E are completed.
  before(C,B) $\wedge$ *before(E,B)*
- We can execute E after B or D are completed.
  before(B,E) $\vee$ *before(D,E)*
- We can execute C after A is completed.
  before(A,C)

Task manager: FOL formulae (2)

UNIVERSITÀ DEGLI STUDI DI TRENTO

If we run the current encoding, we won't be able to find a proof. The main reason is the absence of some hidden conditions (again...)

- We must ensure that no event can happen before itself: $\forall x.(\neg \mathit{before}(x,x))$
- We must ensure that other events does not interfere with our problem (we can assume each external event happens before the 5 ones discussed in the exercise's corpus): $\forall xy.(x \mathrel{!=} \{a,b,c,d,e\} \Rightarrow \mathit{before}(x,y))$

Task manager: FOL formulae (3)

If we run the current encoding, we won't be able to find a proof. The main reason is the absence of some hidden conditions (again...)

- We must ensure the transitivity of the function *before*: $\forall xyz.((before(x,y) \wedge before(y,z)) \Rightarrow before(x,z))$
- CONJECTURE: there is an event that is preceded by everyone (other than itself): $\exists x \forall y.(x!{=}y \Rightarrow before(y,x))$

Task manager: results

- Now we can feed the encoding into E
  ⇒ The prover returns **Proof found!**, so a proof exists. Which is the actual event that executes for last?
- If the conjecture is expressed an an existential formula (such as the case we are analyzing) we can employ the *answer* option.
- The conjecture will be assigned a new role, *question*, and the output will return the set of constants that can satisfy.

Outline

Homework

Homework 3.1: love me, love me not...

Anyone whom Mary loves is a football star.
Any student who does not pass does not play.
John is a student.
Any student who does not study does not pass.
Anyone who does not play is not a football star.
Can we conclude that "If John does not study, then Mary does not love John"?

Homework

UNIVERSITÀ DEGLI STUDI DI TRENTO

Homework 3.2: who hated Caesar?

- Marcus was a man.
- Marcus was a Roman.
- All men are people.
- Caesar was a ruler.
- All Romans were either loyal to Caesar or hated him (or both).
- Everyone is loyal to someone.
- People only try to assassinate rulers they are not loyal to.
- Marcus tried to assassinate Caesar.

Use E to find who hated Caesar, if someone who hated Caesar exists.