Theorem Proving with Vampire for Rigorous Systems Engineering

Capturing programs

Giles Reger and Martin Suda

University of Manchester and TU Wein

This Session

In this session we

- Look at the FOOL extension to TPTP
- Look at the issue of capturing next state relationships of programs
- Discuss some additional issues

Outline

FOOL

Next State Relationship

Further Issues

FOOL

Stands for FOL with `$o`, which is the TPTP boolean sort.

FOOL extends the logic to allow

- Formulas in term position
- Terms in formula position
- Representation of a `$ite` term standing for *if-then-else*
- Representation of `$let` terms allow binding of terms within another expression

Note that SMTLIB already contains these things as they do not differentiate terms and formulas.

FOOL Examples

```
![X:$int,Y:$int] : max(X,Y) = $ite($greater(X,Y),X,Y)
```

```
$let(one := succ(zero),
  $let(two = suc(one),
    two = plus(one,one)))
```

Outline

Encoding (Loop-Free) Programs

Want to encode a program so that we can reason about the possible values of variables.

Issue: `x := x+1` is not logical

A common approach is Static Single Assignment form (SSA). Every variable is assigned to exactly once.

However, we want to avoid SSA for reasons.

Solution: nested `let-in` statements with a theory of tuples

General Idea

A program is a sequence of statements (in general).

Abstract it as `letting` the changes of the current statement apply to the rest of the program.

Issue: `let` talks about the value of a single thing, programs tend to have multiple variables. Therefore, we need to be able to talk about set of states - theory of tuples.

Polymorphic theory of tuples

A union of theories of tuples parametrised by $n > 1$ and sorts $\sigma_1, \ldots, \sigma_n$ of their elements.

- $\mathsf{Sort}(\sigma_1, \ldots, \sigma_n)$
- Constructor function $t : \sigma_1 \times \ldots \times \sigma_n \to (\sigma_1, \ldots, \sigma_n)$
- Projection functions $\pi_i : (\sigma_1, \ldots, \sigma_n) \to \sigma_1$
- Axioms
  1. $(\forall s_1 : \sigma_1) \ldots (\forall s_n : \sigma_n)$
     $(\pi_1(t(s_1, \ldots, s_n)) = s_1 \wedge \ldots \pi_n(t(s_1, \ldots, s_n)) = s_n)$
  2. $(\forall t_1 : (\sigma_1, \ldots, \sigma_n))(\forall t_2 : (\sigma_1, \ldots, \sigma_n))$
     $(t_1 = t_2 \Rightarrow \pi_1(t_1) = \pi_1(t_2) \wedge \ldots \wedge \pi_n(t_1) = \pi_n(t_2))$

Allows tuple let-expressions of the form `let` $(x_1, \ldots, x_n)$ `= binding in body` where `binding` is of sort $(\sigma_1, \ldots, \sigma_n)$ and `body` can use constants $x_1 : \sigma_1, \ldots, x_n : \sigma_n$.

Tools

We have two tools that translate from some language to TPTP:

- `kyckling` targets C-like loopless imperative programs
- `voogie` targets

Both tools were developed by Evgenii Kotelnikov at Chalmers University of Technology

kyckling

`https://github.com/aztek/kyckling`

Targets programs containing statements of the form

- `x := e`
- `if e then` P_1 `else` P_2
- $P_1\,;P_2$

Using the translation

$[x_i\ \mathtt{:=\ e}] \Rightarrow \mathtt{let}\ (\ldots, x_i, \ldots)\ \mathtt{=}\ (\ldots, \mathtt{e}, \ldots)\ \mathtt{in}\ (x_1, \ldots, x_n)$
$[\mathtt{if\ e\ then}\ P_1\ \mathtt{else}\ P_2] \Rightarrow \mathtt{let}\ (x_1, \ldots, x_n)\ \mathtt{=\ if\ e\ then}$
$[P_1]\ \mathtt{else}\ [P_2]\ \mathtt{in}\ (x_1, \ldots, x_n)$
$[P_1\,;P_2] \Rightarrow \mathtt{let}\ D\ \mathtt{in}\ [P_2]$ for $[P_1] = \mathtt{let}\ D\ \mathtt{in}\ (\ldots)$

A Simple Program

```
int[] a; int x = 0, y = 0;
if (a[0] > 0) x++; else y++;
if (a[1] > 0) x++; else y++;
assert x + y == 2;
```

```
./kyckling count_two2.ky
./vampire --newcnf on prob
```

```
thf(asserts, conjecture,
    $let(x := 0, $let(y := 0,
    $let([x, y] := $ite($greater($select(a, 0), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
    $let([x, y] := $ite($greater($select(a, 1), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
         $sum(x, y) = 2))))).
```

A Simple Program

```
int[] a; int x = 0, y = 0;
if (a[0] > 0) x++; else y++;
if (a[1] > 0) x++; else y++;
assert x + y == 2;
```

```
./kyckling count_two2.ky
./vampire --newcnf on prob
```

```
thf(asserts, conjecture,
    $let(x := 0, $let(y := 0,
    $let([x, y] := $ite($greater($select(a, 0), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
    $let([x, y] := $ite($greater($select(a, 1), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
         $sum(x, y) = 2))))).
```

A Simple Program

```
int[] a; int x = 0, y = 0;
if (a[0] > 0) x++; else y++;
if (a[1] > 0) x++; else y++;
assert x + y == 2;
```

```
./kyckling count_two2.ky
./vampire --newcnf on prob
```

```
thf(asserts, conjecture,
    $let(x := 0, $let(y := 0,
    $let([x, y] := $ite($greater($select(a, 0), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
    $let([x, y] := $ite($greater($select(a, 1), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
         $sum(x, y) = 2))))).
```

A Simple Program

```
int[] a; int x = 0, y = 0;
if (a[0] > 0) x++; else y++;
if (a[1] > 0) x++; else y++;
assert x + y == 2;
```

```
./kyckling count_two2.ky
./vampire --newcnf on prob
```

```
thf(asserts, conjecture,
    $let(x := 0, $let(y := 0,
    $let([x, y] := $ite($greater($select(a, 0), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
    $let([x, y] := $ite($greater($select(a, 1), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
        $sum(x, y) = 2))))).
```

A Simple Program

```
int[] a; int x = 0, y = 0;
if (a[0] > 0) x++; else y++;
if (a[1] > 0) x++; else y++;
assert x + y == 2;
```

```
./kyckling count_two2.ky
./vampire --newcnf on prob
```

```
thf(asserts, conjecture,
    $let(x := 0, $let(y := 0,
    $let([x, y] := $ite($greater($select(a, 0), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
    $let([x, y] := $ite($greater($select(a, 1), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
        $sum(x, y) = 2))))).
```

A Simple Program

```
int[] a; int x = 0, y = 0;
if (a[0] > 0) x++; else y++;
if (a[1] > 0) x++; else y++;
assert x + y == 2;
```

```
./kyckling count_two2.ky
./vampire --newcnf on prob
```

```
thf(asserts, conjecture,
    $let(x := 0, $let(y := 0,
    $let([x, y] := $ite($greater($select(a, 0), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
    $let([x, y] := $ite($greater($select(a, 1), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
        $sum(x, y) = 2))))).
```

A Simple Program

```
int[] a; int x = 0, y = 0;
if (a[0] > 0) x++; else y++;
if (a[1] > 0) x++; else y++;
assert x + y == 2;
```

```
./kyckling count_two2.ky
./vampire --newcnf on prob
```

```
thf(asserts, conjecture,
    $let(x := 0, $let(y := 0,
    $let([x, y] := $ite($greater($select(a, 0), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
    $let([x, y] := $ite($greater($select(a, 1), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
        $sum(x, y) = 2))))).
```

A Simple Program

```
int[] a; int x = 0, y = 0;
if (a[0] > 0) x++; else y++;
if (a[1] > 0) x++; else y++;
assert x + y == 2;
```

```
./kyckling count_two2.ky
./vampire --newcnf on prob
```

```
thf(asserts, conjecture,
    $let(x := 0, $let(y := 0,
    $let([x, y] := $ite($greater($select(a, 0), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
    $let([x, y] := $ite($greater($select(a, 1), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
        $sum(x, y) = 2))))).
```

A Simple Program

```
int[] a; int x = 0, y = 0;
if (a[0] > 0) x++; else y++;
if (a[1] > 0) x++; else y++;
assert x + y == 2;
```

```
./kyckling count_two2.ky
./vampire --newcnf on prob
```

```
thf(asserts, conjecture,
    $let(x := 0, $let(y := 0,
    $let([x, y] := $ite($greater($select(a, 0), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
    $let([x, y] := $ite($greater($select(a, 1), 0),
                        $let(x := $sum(x, 1),
                             [x, y]),
                        $let(y := $sum(y, 1),
                             [x, y])),
        $sum(x, y) = 2))))).
```

Task

Play with the included programs i.e. look at their translations and run Vampire on them.

Try and write your own simple program with assertions (at the end only) and see if Vampire can prove the assertions from the program.

voogie

`https://github.com/aztek/voogie`

Does a very similar thing for the much more powerful Boogie language

I introduce the previous one first as it is more simple but `voogie` is more powerful

Example

Run some examples in the command line (programs and output too big to sensibly put on slides).

Outline

Further Issues

Loops

- Need to be abstracted by invariant - Vampire can help here

Modularity (calls to external functions)

- Introduce uninterpreted function and axiomitise it with that function's pre and post conditions