A FOOLish Encoding of the Next State Relations of Imperative Programs

Evgenii Kotelnikov, Laura Kovács and Andrei Voronkov

9th International Joint Conference on Automated Reasoning (IJCAR)
15 July 2018

Motivation

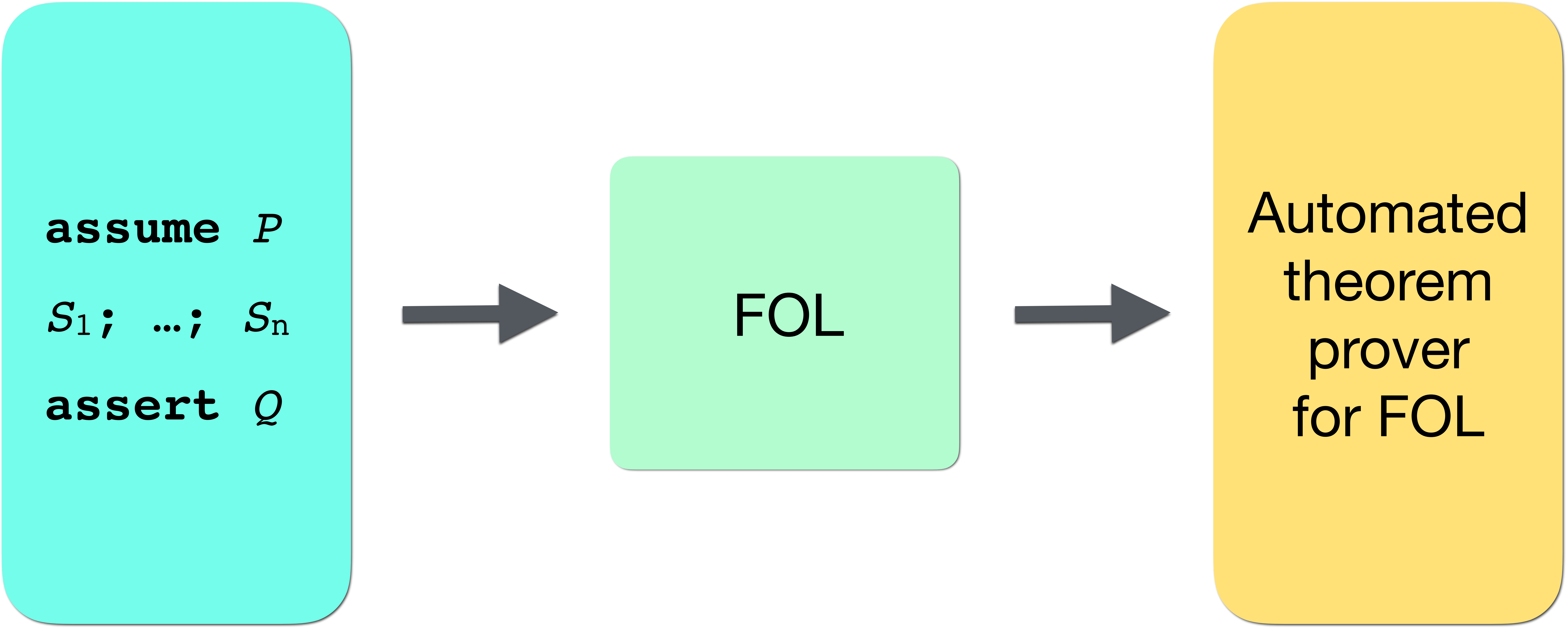
assume P

S_1; …; S_n

assert Q

Automated theorem prover for FOL

Motivation



Motivation

assume P

S_1; …; S_n

assert Q

FOL

Automated theorem prover for FOL

Motivation

assume *P*

S_1; …; S_n

assert *Q*

FOL

FOL

Automated theorem prover for FOL

Motivation

What can a first-order prover do with extensions of FOL?

- Translate to plain FOL that suits it best
- Try multiple translations in a portfolio mode
- Implement heuristics, special inference rules and preprocessing for the extensions

Our previous work

FOOL [Kotelnikov, Kovács and Voronkov; CICM 2015]

A modification of FOL with

- First class boolean sort $(\forall x : bool)(x \vee F(x)) \quad P(\varphi)$
- Conditional expressions $Q(\texttt{if}\ \varphi\ \texttt{then}\ s\ \texttt{else}\ t)$
- Local definitions $\texttt{let}\ a = f(b)\ \texttt{in}\ R(a, a)$

TFX [Sutcliffe and Kotelnikov; PAAR 2018]

Clausification for FOOL [Kotelnikov, Kovács, Suda and Voronkov; GCAI 2016]

Clausification

Large CNF vs large signature

Clausification

Large CNF vs large signature

$$(A_1 \land B_1) \lor \ldots \lor (A_n \land B_n)$$

Clausification

Large CNF vs large signature

$$(R_1 \vee \ldots \vee R_n) \wedge (R_1 \Leftrightarrow (A_1 \wedge B_1)) \wedge \ldots \wedge (R_n \Leftrightarrow (A_1 \wedge B_n))$$

This work

Efficient encoding of sequences of variable assignments and conditional statements in FOOL

```
x := 2
x := x - 1
if (x > 0) {
  y := x + 1
} else {
  y := x - 1
}
y := y + x
```

This work

Efficient encoding of sequences of variable assignments and conditional statements in FOOL

```
x₁ := 2
x₂ := x₁ - 1
if (x₂ > 0) {
  y₁ := x₂ + 1
} else {
  y₂ := x₂ - 1
}
y₃ := y? + x₂
```

FOOL

Conditional expressions

$$max(x, y) = \texttt{if } x \geq y \texttt{ then } x \texttt{ else } y$$

$$\texttt{if } max(x, y) = x \texttt{ then } x \geq y \texttt{ else } y \geq x$$

Definitions of tuples

$$(1, 2) : (\mathbb{Z}, \mathbb{Z})$$

$$\texttt{let } (x, y) = (y, x) \texttt{ in } P(x, y)$$

$$\texttt{let } (x, y) = \texttt{if } x \geq y \texttt{ then } (x, y) \texttt{ else } (y, x) \texttt{ in } P(x, y)$$

A programming language

$$\tau ::= \text{bool} \mid \text{int} \mid \text{array}(\tau_1, \tau_2)$$

$$\begin{aligned} e ::=\ & n \\ \mid\ & \text{true} \mid \text{false} \\ \mid\ & x \mid x[e] \\ \mid\ & e_1 = e_2 \\ \mid\ & e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2 \mid e_1 < e_2 \\ \mid\ & \neg e \mid e_1 \vee e_2 \mid e_1 \wedge e_2 \end{aligned}$$

$$\begin{aligned} s ::=\ & \texttt{skip} \\ \mid\ & x_1, \ldots, x_n := e_1, \ldots, e_n \\ \mid\ & x[e_1] := e_2 \\ \mid\ & \texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2 \\ \mid\ & s_1\ ;\ s_2 \end{aligned}$$

Next state relation

$$\{\varphi\}\ s\ \{\psi\} \qquad\qquad \varphi \Rightarrow \mathcal{N}(s)(\psi)$$

$$\begin{aligned}
\mathcal{N}(\texttt{skip})(t) &= t \\
\mathcal{N}(s_1\ ;\ s_2)(t) &= (\mathcal{N}(s_1) \circ \mathcal{N}(s_2))(t) \\
\mathcal{N}(x_1, \ldots, x_n := e_1, \ldots, e_n)(t) &= \texttt{let}\ (x_1, \ldots, x_n) = (\mathcal{T}(e_1), \ldots, \mathcal{T}(e_2))\ \texttt{in}\ t \\
\mathcal{N}(x[e_1] := e_2)(t) &= \texttt{let}\ x = \mathit{store}(x, \mathcal{T}(e_1), \mathcal{T}(e_2))\ \texttt{in}\ t
\end{aligned}$$

Next state relation

$$\{\varphi\}\ s\ \{\psi\} \qquad\qquad \varphi \Rightarrow \mathcal{N}(s)(\psi)$$

$$\begin{aligned}\mathcal{N}(\texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2)(t) \ &= \ \texttt{let}\ (x_1,\ldots,x_n) = \texttt{if}\ \mathcal{T}(e)\ \texttt{then}\ \mathcal{N}(s_1)((x_1,\ldots,x_n)) \\ &\qquad\qquad\qquad\qquad\qquad\quad\ \texttt{else}\ \mathcal{N}(s_2)((x_1,\ldots,x_n)) \\ &\quad\ \ \texttt{in}\ t,\end{aligned}$$

where $\mathrm{updates}(s_1) \cup \mathrm{updates}(s_2) = \{x_1,\ldots,x_n\}$

Example

```
if (x > y) {
  t := y
  y := x
  x := t
}
d := y - x
assert d >= 0;
```

$$
\begin{array}{l}
\texttt{let}\ (x, y, t) = \ \texttt{if}\ x > y\ \texttt{then} \\
\qquad\qquad \texttt{let}\ t = y\ \texttt{in} \\
\qquad\qquad \texttt{let}\ y = x\ \texttt{in} \\
\qquad\qquad \texttt{let}\ x = t\ \texttt{in}\ (x, y, t) \\
\qquad \texttt{else} \\
\qquad\qquad (x, y, t) \\
\quad \texttt{in let}\ d = x - y \\
\qquad \texttt{in}\ d \geq 0
\end{array}
$$

Example

```
if (x > y) {
  t := y
  y := x
  x := t
}
d := y - x
assert d >= 0;
```

$$
\begin{array}{l}
\texttt{let}\ (x, y, t) = \texttt{if}\ x > y\ \texttt{then} \\
\qquad \texttt{let}\ t = y\ \texttt{in} \\
\qquad \texttt{let}\ y = x\ \texttt{in} \\
\qquad \texttt{let}\ x = t\ \texttt{in}\ (x, y, t) \\
\quad \texttt{else} \\
\qquad (x, y, t) \\
\texttt{in}\ \texttt{let}\ d = x - y \\
\quad \texttt{in}\ d \geq 0
\end{array}
$$

Example

```
if (x > y) {
   t := y
   y := x
   x := t
}
d := y - x
assert d >= 0;
```

$$
\begin{array}{l}
\texttt{let}\ (x, y, t) = \begin{array}[t]{l}
\texttt{if}\ x > y\ \texttt{then} \\
\quad \texttt{let}\ t = y\ \texttt{in} \\
\quad \texttt{let}\ y = x\ \texttt{in} \\
\quad \texttt{let}\ x = t\ \texttt{in}\ (x, y, t) \\
\texttt{else} \\
\quad (x, y, t)
\end{array} \\
\quad \texttt{in let}\ d = x - y \\
\qquad \texttt{in}\ d \geq 0
\end{array}
$$

Example in TFX

```
$let([x: $int, y: $int, t: $int],
     [x, y, t] := $ite($greater(x, y),
                       $let(t: $int, t := y,
                       $let(y: $int, y := x,
                       $let(x: $int, x := t,
                            [x, y, t]))),
                       [x, y, t]),
     $let(d: $int,
          d := $difference(x, y),
          $greatereq(d, 0)))
```

Experimental evaluation

- Take a collection of 50 simple imperative programs
- Implement them in the Boogie language
- Generate verification conditions of these programs using
  - Boogie itself
  - Our implementation of the encoding Voogie (github.com/aztek/voogie)
  - Program verifier BLT
- Compare the performance of Vampire on the results of the translations

Benchmarks

- Write 10 programs with loops — textbook algorithms and solutions to verification competitions
- Unroll each loop 1, 2, 3, 4, 5 times

Loop unrolling

```
assume P;
while e invariant I {
  s;
}
```

Loop unrolling

if $\neg e$ { *bad* := true }; *s*;

Experimental results

	BLT (19)					Boogie (25)					Voogie (36)				
Benchmark	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
binary-search	0,821	163,790				0,884	2,420	3,364	10,709	27,648	1,979	25,135	6,560		163,803
bubble-sort	3,511										0,394	53,192	2,073		
dutch-flag	4,049					24,789					11,384				
insertion-sort	1,780					122,354					18,262	38,169	3,369	21,698	11,639
matrix-transpose	0,465	12,437				1,311		1,078			0,266	8,362			
maxarray	0,174	1,567	47,724			0,205	0,587	1,197	1,702	1,692	0,170	0,587	0,489	2,635	6,325
maximum	0,069	0,140	0,724	12,234		0,066	0,078	0,082	0,095	0,129	0,062	0,065	0,070	0,087	0,102
one-duplicate	0,307	10,039									0,125	2,402	2,231	93,746	145,243
select-k	3,142					96,993					0,216	0,612	203,655		
two-way-sort	0,319	24,622				0,191	0,205	0,647	1,384	1,344	0,464	5,360			