On Finite Domains in First-Order Linear Temporal Logic

Julien Brunel   David Chemouil   **Denis Kuperberg**

ONERA/DTIM - IRIT

04/07/2015
LCC, Kyoto

Introduction

Alloy Language

- Specification language based on First-Order Logic
- Inspired by UML, user-friendly
- Arbitrary predicates $\rightarrow$ Expressivity

Alloy Analyzer

- Bounded verification $\rightarrow$ Decidability
- Use of SAT solvers $\rightarrow$ Efficiency, quick feedback
- 2015: unveiled a security breach in Android permission system

Example of Alloy Specification:

```
open util/ordering [Book] as BookOrder
sig Addr {}
sig Name {}
sig Book {
      names: set Name,
      addr: names→some Addr}
pred add [b1, b2: Book, n: Name, a: Addr] {
     b2.addr = b1.addr +n→a}
pred del [b1, b2: Book, n: Name, a: Addr] {
      b2.addr = b1.addr − n→a}
fact traces {
    all b: Book−BookOrder/last |
       let bnext = b.BookOrder/next |
           some n: Name, a: Addr |
               add [b, bnext, n, a] or del [b, bnext, n, a]}
```

One object book for each time instant. Tedious way of modeling time and reasoning about it.

Alloy Analyzer

Model finder

```
//Show a model where some name has two different addresses
run {some b: Book, n: Name, disj a1, a2: Addr
    | a1 in n.(b.addr) and a2 in n.(b.addr)}
```

Property checker

```
assert delUndoesAdd {
    all b1, b2, b3: Book, n: Name, a: Addr |
        no n.(b1.addr) and add [b1, b2, n, a] and del [b2, b3, n, a]
        implies b1.addr = b3.addr
}
check delUndoesAdd
```

Electrum : Alloy + new dedicated time operators like ′ (value at the next instant) and always:

```
sig Addr {}
sig Name {
  var addr : set Addr
}

pred add [n: Name, a: Addr] {
    addr' = addr +n→a}
pred del [n: Name, a: Addr] {
addr' = addr − n→a}

fact traces {
    always {
            some n: Name, a: Addr | add [n, a] or del [n, a]}
}
```

Infinite number of time instants, that can be referred to easily with a specialized syntax.

FO-LTL

Asbtraction: The logic FO-LTL.

LTL: Good properties of expressivity and complexity, widely used in verification to model infinite time traces.

The logic FO-LTL:

$$\varphi ::= (x_1 = x_2) \mid P_i(x_1, \ldots, x_n) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \mathrm{next}\varphi \mid \varphi \mathrm{until} \varphi.$$

We also define $\mathrm{eventually}\varphi = \mathit{true}\mathrm{until}\varphi$ and $\mathrm{always}\varphi = \neg\mathrm{eventually}(\neg\varphi)$.
We use FO-LTL as underlying logic of the new language Electrum.

- First-Order variables x_i: finite domain
- Implicit time: infinite domain $\mathbb{N}$

What is the theoretical cost of adding LTL ?

Complexity

BSAT Problem: Given φ and N, is there a model for φ of First-Order domain of size at most N ?

Parameters:

- Logic: FO versus FO-LTL
- Encoding of N: unary versus binary
- Rank of formulas (nested quantifiers): bounded ($\bot$) versus unbounded ($\top$).

Complexity

BSAT Problem: Given φ and N, is there a model for φ of First-Order domain of size at most N ?

Parameters:

- Logic: FO versus FO-LTL
- Encoding of N: unary versus binary
- Rank of formulas (nested quantifiers): bounded ($\perp$) versus unbounded ($\top$).

Theorem

	N unary	N binary
FO $\perp$	NP-complete	NEXPTIME-complete
FO $\top$	NEXPTIME-complete	NEXPTIME-complete
FO-LTL $\perp$	PSPACE-complete	EXPSPACE-complete
FO-LTL $\top$	EXPSPACE-complete	EXPSPACE-complete

Ideas of the proofs

Membership:

- Guess a structure and verify it,
- Re-encode the formula for bounded rank,
- Use PSPACE LTL Satisfiability.

Hardness

- Reduce from Turing machines or SAT for NP-hardness,
- Encode states and alphabet in the signature,
- Structure encodes space/time for FO and space for FO-LTL,
- $a(x, t)$ for "cell x at time t is labeled a",
- Use binary encoding for x and t for unbounded unary,
- formula in the wanted fragment encode run of the machine.

Finite Model Theory

Finite Model Property: If there is a model there is a finite one.
FO Fragments with FMP;

- $[\exists^*\forall^*, \mathit{all}]_=$ (Ramsey 1930)
- $[\exists^*\forall\exists^*, \mathit{all}]_=$ (Ackermann 1928)
- $[\exists^*, \mathit{all}, \mathit{all}]_=$ (Gurevich 1976)
- $[\exists^*\forall, \mathit{all}, (1)]_=$ (Grädel 1996)
- FO_2 (Mortimer 1975) : 2 variables.

Theorem

Adding $\mathrm{next}, \mathrm{eventually}$ *preserves FMP if the fragment imposes no constraint on the number and arity of predicates/functions.*

True for all above fragments except Grädel: only one function of arity one.

Axioms of infinity

In general, adding LTL allows to write axioms of infinity:

With one existential variable:

$$\mathrm{always}(\exists \mathrm{x}.\mathrm{P}(\mathrm{x}) \wedge \mathrm{next}(\mathrm{always}\neg \mathrm{P}(\mathrm{x})))).$$

Without nesting quantifiers in temporal operators:

$$\forall x \exists y. P(c) \wedge \mathrm{always}(\mathrm{P}(\mathrm{x}) \Rightarrow \mathrm{next}(\mathrm{P}(\mathrm{y}) \wedge \mathrm{always}\neg \mathrm{P}(\mathrm{x}))).$$

Without always:

$$\forall x \exists y. P(c) \wedge ((P(x) \wedge P(y))\mathrm{until}(\neg \mathrm{P}(\mathrm{x}) \wedge \mathrm{P}(\mathrm{y}))).$$

Conclusion

Theoretical study of FO-LTL versus FO

- Complexity
- Finite model property

On-going work with Univ. of Minho/IRIT

- Implementation of different verification procedures for Electrum:
  - Reduce to LTL satisfiability
  - Reduce to Alloy
- Use of efficient solvers
- Comparison with TLA and B