Alt-Ergo 2.5

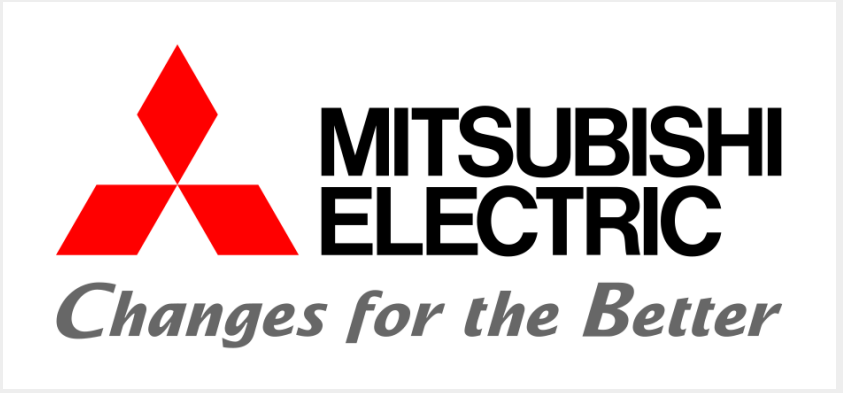
vers un solveur modèle

21/11/2022

Quick presentation

The Alt-Ergo club

Alt-Ergo is an automatic prover of mathematical formulas used by



The Team

- Hichem Rami Ait El Hara – Junior R&D Engineer and PhD student in SMT solvers under the supervision of François Bobot
- Guillaume Bury – R&D Engineer holding a PhD in SMT solvers
- Steven De Oliveira – R&D Engineer holding a PhD in Formal Verification
- Pierre Villemot – R&D Engineer holding a PhD in Mathematics and Full-time maintainer of Alt-Ergo

Features in the last release (Alt-Ergo 2.4.2)

- Partial support of the input language *SMT-LIB 2*
- Supported theories:
  - **U**ninterpreted **F**unction (**UF**)
  - **L**inear **I**nteger **A**rithmetic (**LIA**)
  - **L**inear **R**ational **A**rithmetic (**LRA**)
  - Floating-point arithmetic (only for the native input format)
  - ADT
  - Bitvectors and Arrays (initial support)
- First-order polymorphism
- Lablgtk3 and Dune 3 updates
- Bug fixes

Upcoming Features on Next

- (Merged - MERCE project) Models (counterexamples) for Arithmetics and Enums
- (This week) Finalize Dolmen integration
  - Complete support of the input language *SMT-LIB 2*
  - Better syntax error handling
  - More input languages supported (as tptp)
- Support for *SMT-LIB 2* floating-point arithmetic (**FP**)
- Better CLI (simpler options) (user feedback)
- Integrate the model feature in Why3

Models generation

Use cases

```
$ alt-ergo grothendieck.ae --model
```

grothendieck.ae

```
1  logic a,b,n:int
2
3  axiom init: n=57
4  axiom range: 1<=a<=n and 1<=b<=n
5
6  goal is_prime:
7    a*b = n -> a=1 or b=1
```

output model

```
1  (model
2    (define-fun a () int 3)
3    (define-fun b () int 19)
4    (define-fun n () int 57))
5
6
7
```

```
$ alt-ergo commutative.ae --model
```

commutative.ae

```
1  logic f: int,int->int
2
3  axiom a: forall x:int. forall y:int.
4    forall z:int. f(x,f(y,z))=f(f(x,y),z)
5
6  goal g:
7    forall x:int. forall y:int. f(x,y)=f(y,x)
```

output model

```
1  (model
2    (define-fun x () int (- 1))
3    (define-fun y () int 0)
4    (define-fun f ((u int) (v int)) int
5      (ite (and (= u y) (= v x)) 2 0)))
6
7
```

Use cases

```
$ alt-ergo sudoku.ae --model
```

sudoku.ae

```
type one_four = One | Two | Three | Four

logic grid: int, int -> one_four

predicate by_row =
    forall i:int. 0<=i<4 ->
    distinct(grid(i,0),grid(i,1),
      grid(i,2),grid(i,3))

predicate by_column =
    forall j:int. 0<=j<4 ->
    distinct(grid(0,j),grid(1,j),
      grid(2,j),grid(3,j))

predicate by_square =
    forall i,j:int. 0<=i<2 -> 0<=j<2 ->
    distinct(grid(2*i,2*j),grid(2*i,2*j+1),
      grid(2*i+1,2*j),grid(2*i+1,2*j+1))

axiom init:
    grid(0,0)=Three
    and grid(0,1)=Four
    and grid(0,2)=One
    and grid(1,1)=Two
    and grid(2,2)=Two
```

output model

```
(model
  (define-fun by_row () bool true)
  (define-fun by_column () bool true)
  (define-fun by_square () bool true)

  (define-fun grid ((arg_0 int) (arg_1 int)) <su
    (ite
      (or
        (and (= arg_0 3) (= arg_1 0))
        (and (= arg_0 2) (= arg_1 2))
        (and (= arg_0 1) (= arg_1 1))
        (and (= arg_0 0) (= arg_1 3)))
      Two
    (ite
      (or
        (and (= arg_0 3) (= arg_1 3))
        (and (= arg_0 2) (= arg_1 1))
        (and (= arg_0 1) (= arg_1 2))
        (and (= arg_0 0) (= arg_1 0)))
      Three
    (ite
      (or
        (and (= arg_0 3) (= arg_1 1))
        (and (= arg_0 2) (= arg_1 0))
        (and (= arg_0 1) (= arg_1 3))
```

What will happen next!

- Checking models with Dolmen (for unquantified theories)
- (OptiAE project - MERCE project) Optimization of values in generated models
- (Décysif project) Generate models for other theories:
  - (WIP) ADT
  - Floating-point arithmetic
  - Bitvectors
  - Arrays
  - (Recursive) Records
- New (documented) API
- Improve ground reasoning on Bitvectors and ADT

Thanks

Blog post about Models in Alt-Ergo