Alt-Ergo

An SMT Solver for Software Verification

Mohamed Iguernelala — OCamlPro SAS
About …
About the Speaker

Who am I?
- Mohamed Iguernelala
- Senior R&D engineer at OCamlPro SAS
- Research associate in the VALS team, LRI

My Research topics
- Satisfiability Modulo Theories (SMT)
- Combining rewriting and SMT techniques
- Designing and combining decision procedures for SMT

Formerly, I was a PhD student in the VALS team

Maintainer of the Alt-Ergo SMT solver at OCamlPro
Help companies to use OCaml in industrial projects ...

Developments on demand in OCaml

- OPAM package manager
- the Typerex toolbox (IDE, memory/GC profilers, libs, ...)

Code analysis and optimization

- analysis of OCaml code
- introduction of new optimization passes in OCaml compiler
- tools for numerical calculus (Scilab, Modelica)

Formal methods

- maintenance and improvement of Alt-Ergo
The Origins of Alt-Ergo ...
History of Alt-Ergo

Preliminary work by Sylvain Conchon in 2002 on decision procedures and their combination:

- Combining Shostak theories
- Generic framework for combining decision procedures à la Nelson-Oppen

SMT solvers at that time:

- Simplify (DEC/Compaq/HP)
- SVC, CVC (Stanford)
- ICS (SRI)
- UCLID (Berkeley)
History of Alt-Ergo : Motivations

Deductive program verification platforms at that time

- ESC/Java (DEC/Compaq/HP)
- Why/Caduceus/Krakatoa (LRI)

Main motivations:

- An automatic theorem prover for Why
- Polymorphic first-order logic with built-in theories (free equality, linear arithmetic) similar to Why’s syntax
- Certification (in Coq)
History of Alt-Ergo: First Release in 2006

70’s: Stanford Pascal Verifier (Nelson-Oppen combination)
1984: Shostak algorithm
1992: Simplify
1995: SVC
2001: ICS
2002: CVC, UCLID
2004: CVC Lite
2005: Barcelogic, MathSAT
2005: Yices
2006: CVC3, Alt-Ergo
2007: Z3, MathSAT4
2008: Boolector, OpenSMT, Beaver, Yices2
2009: STP, VeriT
2010: MathSAT5, SONOLAR
2011: STP2, SMTInterpol
2012: CVC4
Tools Using Alt-Ergo Today

- WhyML Programs
- C Code
- Transition Systems
- Ada Code
- Cryptographic Protocols
- B Method, ...

SPARK 2014, EasyCrypt and Atelier-B use it via Why3
Alt-Ergo: Contributors

Project leaders:

- Sylvain Conchon
- Evelyne Contejean

PhD students:

- Stéphane Lescuyer (reflexive Coq tactic)
- Mohamed Igernelala (AC, arithmetic, SAT)
- Claire Dross (quantifiers, user-defined axiomatic theories)

Post-docs, interns, engineers:

- François Bobot (arithmetic, case-split analysis)
- Denis Cousineau (prototype of a lightweight Coq traces)
- Johannes Kanig (Coq traces)
- Alain Mebsout (altgr-ergo, DO-178C)
- Cody Roux (prototype of a floating-point theory)
Alt-Ergo @ OCamlPro …
What Can OCamlPro Do For Alt-Ergo

at OCamlPro

- Better reactivity for bugs fixes
- Improving performances and expressiveness, even if there are no research challenges
- Commercial support and services on demand

in collaboration with the VALS team

- Exploring research/theoretical aspects such as designing new decision procedures and their combination
Alt-Ergo @ OCamlPro: Code Base

- Core of Alt-Ergo
  - CeCILL-C

- OCamlPro Plugins
  - OCamlPro Tools
    - Non Commercial
Alt-Ergo © OCamlPro : Devel & Release Processes

Yearly Delayed Public Releases

Development Process of Alt-Ergo © OCamlPro
<table>
<thead>
<tr>
<th>Month</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Septembre 2013</td>
<td>public release 0.95.2</td>
</tr>
<tr>
<td>Octobre 2013</td>
<td>private release 0.99</td>
</tr>
<tr>
<td>Juillet 2014</td>
<td>Try Alt-Ergo 0.99</td>
</tr>
<tr>
<td>Decembre 2014</td>
<td>public release 0.99.1</td>
</tr>
<tr>
<td>Decembre 2014</td>
<td>private release 1.00-beta</td>
</tr>
<tr>
<td>January 2015</td>
<td>private release 1.00</td>
</tr>
<tr>
<td>January 2015</td>
<td>Try Alt-Ergo 1.00</td>
</tr>
</tbody>
</table>
Under the Hood ...
(* this is a comment *)

type 'a list (* abstract parametric type *)

logic nil: 'a list (* function symbol of arity 0 *)
logic cons: 'a, 'a list -> 'a list
logic length: 'a list -> int

(* implicit quantification over a type variable 'a *)
axiom a1: length(nil) = 0

axiom a2:
  forall x:'a. forall l:'a list.
  length(cons(x,l)) = 1 + length(l)

goal g: forall x:'a. length(cons(x,nil)) = 1
Frontend

Typing : à la ML, with prenex polymorphism

Triggers inference :

- Compute triggers ("guards") that will be used to generate ground instances from universally quantified formulas

Example: 

```
axiom a2: 
    forall x:'a. forall l:'a list.
    length(cons(x,l)) = 1 + length(l)
```

- The type variable 'a is implicitly universally quantified
- Possible “good” triggers are `cons(x, l)` and `length(cons(x, l))`. They cover all variables and type variables

No additional pre-processing steps in Alt-Ergo’s frontend!
SAT Solver(s)

A “default” DPLL solver, coded in a functional style:

- Efficient in the context of deductive program verification
- No “clauses learning” when backjumping
- BCP modulo theories
- Lazy CNF conversion
- Generated instances in a branch of the SAT are ignored when looking for a model in another branch

A CDCL solver inspired by minisat, coded in an imperative style:

- Efficient when a complex propositional reasoning is needed
- Clauses learning when backjumping
- All the instances generated during the matching process are kept in the SAT’s environment
Matching

Generates ground “consequences” from universally quantified formulas

Works by **Matching** techniques:

- if a universally quantified formula $\forall x. F(x)$ is “guarded“ by a trigger $g(h(x))$
- if some ground term $g(h(a))$ is present in decision procedures’ environment
- then, add the instance $F(a)$ to the SAT solver’s context
The Matching process is done:

1. modulo the equalities that are true when matching:
   - the trigger $g(h(x))$ “matches” the term $g(c)$ modulo the equality $c = h(b)$.
   - the resulting substitution is $\{x \mapsto b\}$

2. modulo the theory of records:
   - the trigger $\{field_1 = x ; field_2 = y\}$ “matches” any ground terms $r$ in decision procedures’ context of the same type.
   - the resulting substitution is $\{x \mapsto r.field_1 ; y \mapsto r.field_2\}$
Decision Procedures: Supported Theories

- The free theory of equality with uninterpreted symbols
- Linear arithmetic over integers and rationals
- Non-linear arithmetic
- Polymorphic functional arrays
- Enumerated datatypes
- Record datatypes
- Associative and commutative (AC) symbols
- Fixed-size bit-vectors with concatenation and extraction
Convex equational theories

- Produces a union-find data structure modulo theories

Diagram:

- LIA (=)
- LRA (=)
- ENUM (=)
- Recs
- Bitv (=)

Combination of Shostak theories

Ground Completion modulo AC and Shostak theories

Congruence Closure algorithm
Non convex or non equational theories

This part has access to the union-find data structure constructed so far

Case-split analysis over finite domains is needed for completeness
The case-split module maintains two environments:
- real theory env
- real theory env + guesses (for values over finite domains)

All components are implemented in a pure functional style, which facilitates backtracking.
Decision Procedures: Theoretical Foundations

- CC(X): Semantical Combination of Congruence Closure with Solvable Theories.
- AC(X): Canonized Rewriting and Ground AC Completion Modulo Shostak Theories
- A Simplex-based extension of Fourier-Motzkin for solving linear integer arithmetic
- A Collaborative Framework for Non-Linear Integer Arithmetic Reasoning in Alt-Ergo
- Combining Shostak Theories (Work In Progress)

More on alt-ergo.lri.fr
Current Funding / R&D Projects …
Extend Atelier-B with state-of-the-art automatic solvers

- We did a lot of optimizations in different parts of Alt-Ergo to make it scale on B proof obligations

- We developed profiling tools to monitor the solver and quickly detect the parts that need improvements

- We identified some theoretical challenges on which we are currently working
Explore new combination schemes and tackle hard theories

- Integrate floating-point arithmetic in Alt-Ergo
- Extend the theory of bit-vectors (bitwise operators, conversion from/to bounded integers, ...)
- Generate models when formulas are satisfiable
Performances of Alt-Ergo 1.00 ...
Quick Comparison Between Different Versions

<table>
<thead>
<tr>
<th></th>
<th>public release 0.95.2</th>
<th>public release 0.99.1</th>
<th>private release 1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proved valid</td>
<td>15980</td>
<td>16334</td>
<td>17638</td>
</tr>
<tr>
<td>Proved valid (%)</td>
<td>84,01 %</td>
<td>85,77 %</td>
<td>92,62 %</td>
</tr>
<tr>
<td>Required time (seconds)</td>
<td>10831</td>
<td>10504</td>
<td>9767</td>
</tr>
<tr>
<td>Average speed (formulas / second)</td>
<td>1,47</td>
<td>1,55</td>
<td>1,81</td>
</tr>
</tbody>
</table>

- timeout = 60 seconds, benchmark of 19044 formulas
- some formulas are known to be invalid
- some formulas need inductive reasoning
### Zoom on BWare Benchmarks (1/2)

<table>
<thead>
<tr>
<th></th>
<th>before the project started (v. 0.95.1)</th>
<th>private release 1.00</th>
<th>1.00 + dedicated options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proved valid</td>
<td>5696</td>
<td>10287</td>
<td>10422</td>
</tr>
<tr>
<td>Proved valid (%)</td>
<td>53.73 %</td>
<td>97.03 %</td>
<td>98.30 %</td>
</tr>
<tr>
<td>Required time (seconds)</td>
<td>4081</td>
<td>5578</td>
<td>3045</td>
</tr>
<tr>
<td>Average speed (formulas / second)</td>
<td>1.40</td>
<td>1.84</td>
<td>3.43</td>
</tr>
</tbody>
</table>

- timeout = 60 seconds, benchmark of 10602 formulas
- success rate of Atelier-B’s automatic prover = 86 %
- remaining formulas (14%) proved interactively
- dedicated options: no E-Matching, one trigger per axiom (default value is 2)
Zoom on BWare Benchmarks (2/2)
Main Changes in Version 1.00 (w.r.t. 0.99.x)

- 20 bugs fixes
- Two new OCamlPro plugins (profiling and linear integer arithmetic inequalities)
- A lot of improvements in different data structures
- Improvements/rewriting/simplification of many components and algorithms
- Enhancement of SAT and instantiation heuristics
- ...

More on http://www.ocamlpro.com/blog/index.html