Improving Coq Propositional Reasoning Using a Lazy CNF Conversion

Stéphane Lescuyer Sylvain Conchon

Université Paris-Sud / CNRS / INRIA Saclay – Île-de-France

FroCoS'09 – Trento – 18/09/2009

Outline

Alt-Ergo

Alt-Ergo : an SMT solver dedicated to program verification

http://alt-ergo.lri.fr

- Satisfiability Modulo Theories
  - $\Rightarrow$ linear arithmetic, pairs, AC symbols, bitvectors
- dedicated to program verification
  - $\Rightarrow$ proof obligations from program analysis
  - $\Rightarrow$ Why, Boogie/PL

The big picture

We want to verify Alt-Ergo in the Coq proof assistant

The goal is twofold :

1. validating the algorithms at work in Alt-Ergo
   SAT solver, congruence closure, combination with theories, ...
2. building a certified version of Alt-Ergo that could be used by Coq users directly as a tactic

The big picture

We want to verify Alt-Ergo in the Coq proof assistant

The goal is twofold :

1. validating the algorithms at work in Alt-Ergo
   SAT solver, congruence closure, combination with theories, ...
2. building a certified version of Alt-Ergo that could be used by Coq users directly as a tactic

Two main approaches :

- having the solver produce some certificate
- implement the solver in the proof assistant and use reflection

A taste of reflection

Coq is a programming language

- based on the Calculus of Inductive Constructions
- one can write ML-like programs
- efficient virtual machine for evaluation

A taste of reflection

Coq is a programming language

- based on the Calculus of Inductive Constructions
- one can write ML-like programs
- efficient virtual machine for evaluation

The conversion rule

- deduction modulo evaluation
- proving that two expressions are equal ? call the VM !

```
Theorem fib20 : fib 20 = 10946.
Proof.  vm_compute ; reflexivity. Qed.
```

Another taste of reflection

Given a decidable property P on objects of type t :

- write a program that decides P :

```
Definition P_dec (x : t) : bool := ...
```

- prove that it actually decides P :

```
Property P_1 : ∀x, P_dec x = true → P x.
Property P_2 : ∀x, P_dec x = false → ~(P x).
```

- to prove $P(a)$ for any concrete a of type t :

```
Corollary Pa : P a.
Proof. apply P_1 ; vm_compute ; reflexivity. Qed.
```

Another taste of reflection

Given a decidable property P on objects of type t :

- write a program that decides P :

```
Definition P_dec (x : t) : bool := ...
```

- prove that it actually decides P :

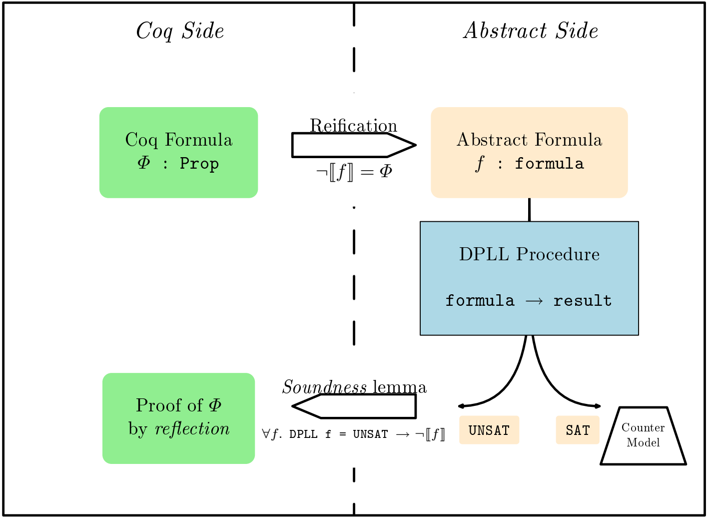
```
Property P_1 : ∀x, P_dec x = true → P x.
Property P_2 : ∀x, P_dec x = false → ~(P x).
```

- to prove $P(a)$ for any concrete a of type t :

```
Corollary Pa : P a.
Proof. apply P_1 ; vm_compute ; reflexivity. Qed.
```

We apply this method to a full SAT solver.

General overview of the tactic



A Modular DPLL procedure

$$\textsc{Unit}\ \frac{\Gamma, l \vdash \Delta}{\Gamma \vdash \Delta, l} \qquad \textsc{Red}\ \frac{\Gamma, l \vdash \Delta, C}{\Gamma, l \vdash \Delta, \bar{l} \vee C} \qquad \textsc{Elim}\ \frac{\Gamma, l \vdash \Delta}{\Gamma, l \vdash \Delta, l \vee C}$$

$$\textsc{Conflict}\ \frac{}{\Gamma \vdash \Delta, \emptyset} \qquad \textsc{Split}\ \frac{\Gamma, l \vdash \Delta \qquad \Gamma, \bar{l} \vdash \Delta}{\Gamma \vdash \Delta}$$

A Modular DPLL procedure

$$\textsc{Unit}\ \frac{\Gamma, l \vdash \Delta}{\Gamma \vdash \Delta, l} \qquad \textsc{Red}\ \frac{\Gamma, l \vdash \Delta, C}{\Gamma, l \vdash \Delta, \bar{l} \vee C} \qquad \textsc{Elim}\ \frac{\Gamma, l \vdash \Delta}{\Gamma, l \vdash \Delta, l \vee C}$$

$$\textsc{Conflict}\ \frac{}{\Gamma \vdash \Delta, \emptyset} \qquad \textsc{Split}\ \frac{\Gamma, l \vdash \Delta \qquad \Gamma, \bar{l} \vdash \Delta}{\Gamma \vdash \Delta}$$

```
Module Type LITERAL.
  Parameter t : Set.
  Parameter mk_not : t → t.
  Axiom mk_not_invol : ∀l, mk_not (mk_not l) = l.
  ...
End LITERAL.
Module DPLL (L : LITERAL) ....
```

CNF conversion

A formula needs to be converted into CNF for DPLL

1. De Morgan rules
   $A \vee (B \wedge C) \quad \longrightarrow \quad (A \vee B) \wedge (A \vee C)$
2. Introducing Tseitin variables
   $A \vee (B \wedge C) \quad \longrightarrow \quad (A \vee X) \wedge (\bar{X} \vee B) \wedge (\bar{X} \vee C) \wedge (X \vee \bar{B} \vee \bar{C})$
3. Plaisted/Greenbaum
   $A \vee (B \wedge C) \quad \longrightarrow \quad (A \vee X) \wedge (\bar{X} \vee B) \wedge (\bar{X} \vee C)$
4. many more variants...

The need for another conversion

Tseitin-style conversions raise issues in SMT solvers

The need for another conversion

Tseitin-style conversions raise issues in SMT solvers

- breaks the logical structure of the original formula

The need for another conversion

Tseitin-style conversions raise issues in SMT solvers

- breaks the logical structure of the original formula
- the Tseitin variables must be given a valuation

The need for another conversion

Tseitin-style conversions raise issues in SMT solvers

- breaks the logical structure of the original formula
- the Tseitin variables must be given a valuation
  - $A \models A \vee (B \wedge C)$ should be trivially satisfiable

The need for another conversion

Tseitin-style conversions raise issues in SMT solvers

- breaks the logical structure of the original formula
- the Tseitin variables must be given a valuation
  - $A \models A \vee (B \wedge C)$ should be trivially satisfiable
  - because of the conversion, the problem actually becomes $A \models (A \vee X) \wedge (\bar{X} \vee B) \wedge (\bar{X} \vee C) \wedge (X \vee \bar{B} \vee \bar{C})$

The need for another conversion

Tseitin-style conversions raise issues in SMT solvers

- breaks the logical structure of the original formula
- the Tseitin variables must be given a valuation
  - $A \models A \vee (B \wedge C)$ should be trivially satisfiable
  - because of the conversion, the problem actually becomes $A \models (\bar{X} \vee B) \wedge (\bar{X} \vee C) \wedge (X \vee \bar{B} \vee \bar{C})$

The need for another conversion

Tseitin-style conversions raise issues in SMT solvers

- breaks the logical structure of the original formula
- the Tseitin variables must be given a valuation
  - $A \models A \vee (B \wedge C)$ should be trivially satisfiable
  - because of the conversion, the problem actually becomes $A \models (\bar{X} \vee B) \wedge (\bar{X} \vee C) \wedge (X \vee \bar{B} \vee \bar{C})$
  - both choices for X must be tried until all the definitional clauses are eliminated

The need for another conversion

Tseitin-style conversions raise issues in SMT solvers

- breaks the logical structure of the original formula
- the Tseitin variables must be given a valuation
  - $A \models A \vee (B \wedge C)$ should be trivially satisfiable
  - because of the conversion, the problem actually becomes $A \models (\bar{X} \vee B) \wedge (\bar{X} \vee C) \wedge (X \vee \bar{B} \vee \bar{C})$
  - both choices for X must be tried until all the definitional clauses are eliminated
  - additional work and additional terms!

The need for another conversion

Tseitin-style conversions raise issues in SMT solvers

- breaks the logical structure of the original formula
- the Tseitin variables must be given a valuation
  - $A \models A \vee (B \wedge C)$ should be trivially satisfiable
  - because of the conversion, the problem actually becomes $A \models (\bar{X} \vee B) \wedge (\bar{X} \vee C) \wedge (X \vee \bar{B} \vee \bar{C})$
  - both choices for X must be tried until all the definitional clauses are eliminated
  - additional work and additional terms!

We don't want to make the formula look harder than it really is.

Lazy CNF conversion

Solution used in Simplify :

- separate definitional clauses from other clauses
- only add them when needed
- “relevancy propagation” in Z3

Lazy CNF conversion

Solution used in Simplify :

- separate definitional clauses from other clauses
- only add them when needed
- "relevancy propagation" in Z3

Detlefs, Nelson, et al. (2005)

...introducing lazy CNF into Simplify avoided such a host of performance problems that [..] it converted a prover that didn't work in one that did.

Lazy CNF conversion

Solution used in Simplify :

- separate definitional clauses from other clauses
- only add them when needed
- "relevancy propagation" in Z3

Detlefs, Nelson, et al. (2005)

...introducing lazy CNF into Simplify avoided such a host of performance problems that [..] it converted a prover that didn't work in one that did.

Our idea : Tseitin variables should be the formulas they represent !

Expandable literals

Expandable literals are literals that can represent any formulas.

- such a literal can be a regular literal l;
- or a proxy for a non-atomic formula F : $\boxed{F}$

Expandable literals

Expandable literals are literals that can represent any formulas.

- such a literal can be a regular literal l ;
- or a proxy for a non-atomic formula F : $\boxed{F}$

Expansion of a proxy literal returns a CNF of literals

For instance, $\boxed{A \vee (D \wedge C)}$ can return

- {{A, C}, {D, C}} $\rightarrow$ full CNF
- {{A, $\boxed{D \wedge C}$ }} $\rightarrow$ one layer only

Expandable literals

Expandable literals are literals that can represent any formulas.

- such a literal can be a regular literal l ;
- or a proxy for a non-atomic formula F : $\boxed{F}$

Expansion of a proxy literal returns a CNF of literals

For instance, $\boxed{A \vee (D \wedge C)}$ can return

- {{A, C}, {D, C}} $\rightarrow$ full CNF
- {{A, $\boxed{D \wedge C}$ }} $\rightarrow$ one layer only

$\Rightarrow$ on-the-fly incremental CNF conversion

Changing DPLL

$$\textsc{Unit}\ \frac{\Gamma, l \vdash \Delta}{\Gamma \vdash \Delta, l} \qquad \textsc{Red}\ \frac{\Gamma, l \vdash \Delta, C}{\Gamma, l \vdash \Delta, \bar{l} \vee C}$$

$$\textsc{Elim}\ \frac{\Gamma, l \vdash \Delta}{\Gamma, l \vdash \Delta, l \vee C} \qquad \textsc{Conflict}\ \frac{}{\Gamma \vdash \Delta, \emptyset}$$

$$\textsc{Split}\ \frac{\Gamma, l \vdash \Delta \qquad \Gamma, \bar{l} \vdash \Delta}{\Gamma \vdash \Delta}$$

$$\Phi \text{ unsatisfiable} \quad \Leftrightarrow \quad \emptyset \vdash \Delta_\Phi$$

Changing DPLL

$$\textsc{Unit}\ \frac{\Gamma, l \vdash \Delta, \texttt{expand}(l)}{\Gamma \vdash \Delta, l} \qquad \textsc{Red}\ \frac{\Gamma, l \vdash \Delta, C}{\Gamma, l \vdash \Delta, \bar{l} \vee C}$$

$$\textsc{Elim}\ \frac{\Gamma, l \vdash \Delta}{\Gamma, l \vdash \Delta, l \vee C} \qquad \textsc{Conflict}\ \frac{}{\Gamma \vdash \Delta, \emptyset}$$

$$\textsc{Split}\ \frac{\Gamma, l \vdash \Delta, \texttt{expand}(l) \qquad \Gamma, \bar{l} \vdash \Delta, \texttt{expand}(\bar{l})}{\Gamma \vdash \Delta}$$

$$\Phi \text{ unsatisfiable} \quad \Leftrightarrow \quad \emptyset \vdash \Phi$$

Changing DPLL

$$\textsc{Unit}\ \frac{\Gamma, l \vdash \Delta, \textcolor{red}{\mathtt{expand}(l)}}{\Gamma \vdash \Delta, l} \qquad \textsc{Red}\ \frac{\Gamma, l \vdash \Delta, C}{\Gamma, l \vdash \Delta, \bar{l} \vee C}$$

$$\textsc{Elim}\ \frac{\Gamma, l \vdash \Delta}{\Gamma, l \vdash \Delta, l \vee C} \qquad \textsc{Conflict}\ \frac{}{\Gamma \vdash \Delta, \emptyset}$$

$$\textsc{Split}\ \frac{\Gamma, l \vdash \Delta, \textcolor{red}{\mathtt{expand}(l)} \qquad \Gamma, \bar{l} \vdash \Delta, \textcolor{red}{\mathtt{expand}(\bar{l})}}{\Gamma \vdash \Delta}$$

$$\Phi \text{ unsatisfiable} \quad \Leftrightarrow \quad \emptyset \vdash \boxed{\Phi}$$

- adding proxies to the partial model is not mandatory
- helps taking advantage of sharing : $\boxed{\phi} \vee \boxed{\neg\phi}$

Defining expandable literals in Coq

```
Inductive t : Set :=
| Proxy (pos neg : list (list t))
| L (a : atom) (b : bool).
```

Defining expandable literals in Coq

```
Inductive t : Set :=
| Proxy (pos neg : list (list t))
| L (a : atom) (b : bool).
```

Negation can be computed in constant time.

```
Definition mk_not (l : t) : t :=
  match l with
  | Proxy pos neg ⇒ Proxy neg pos
  | L a b ⇒ L a (negb b)
  end.
```

Defining expandable literals in Coq

```
Inductive t : Set :=
| Proxy (pos neg : list (list t))
| L (a : atom) (b : bool).
```

Negation can be computed in constant time.

```
Definition mk_not (l : t) : t :=
  match l with
  | Proxy pos neg ⇒ Proxy neg pos
  | L a b ⇒ L a (negb b)
  end.
```

⇒ To convince Coq, one requires invariants on the structure...

Add invariants with dependent types

We want the `pos` and `neg` to really be the negation of one another.

$$\mathcal{N}((\bigvee_{i=1}^{n} x_i) \wedge C) = \bigwedge_{i=1}^{n} \bigwedge_{D \in \mathcal{N}(C)} (\bar{x}_i \vee D)$$

Add invariants with dependent types

We want the pos and neg to really be the negation of one another.

$$\mathcal{N}((\bigvee_{i=1}^{n} x_i) \land C) = \bigwedge_{i=1}^{n} \bigwedge_{D \in \mathcal{N}(C)} (\bar{x}_i \lor D)$$

$\Rightarrow$ `Proxy pos neg` is well-formed if :

$$\mathcal{N}(\mathtt{neg}) = \mathtt{pos}, \quad \mathcal{N}(\mathtt{pos}) = \mathtt{neg}, \quad \forall l \in \mathtt{pos}, l \text{ is well-formed}$$

```
Definition t : Type := {l | wf_lit l}.
```

Add invariants with dependent types

We want the `pos` and `neg` to really be the negation of one another.

$$\mathcal{N}((\bigvee_{i=1}^{n} x_i) \wedge C) = \bigwedge_{i=1}^{n} \bigwedge_{D \in \mathcal{N}(C)} (\bar{x}_i \vee D)$$

$\Rightarrow$ `Proxy pos neg` is well-formed if :

$$\mathcal{N}(\texttt{neg}) = \texttt{pos}, \quad \mathcal{N}(\texttt{pos}) = \texttt{neg}, \quad \forall l \in \texttt{pos}, l \text{ is well-formed}$$

```
Definition t : Type := {l | wf_lit l}.
Property wf_mk_not : ∀l, wf_lit l → wf_lit (mk_not l).
Proof. ..... Qed.
Definition mk_not (l : t) : t := ... (* uses wf_mk_not *)
```

Translation for logical connectives

Proxy	*pos*	*neg*
$X \equiv F \vee G$	$\{F \vee G\}$	$\{\bar{F}\}\{\bar{G}\}$
$X \equiv F \wedge G$	$\{F\}\{G\}$	$\{\bar{F} \vee \bar{G}\}$
$X \equiv (F \rightarrow G)$	$\{\bar{F} \vee G\}$	$\{F\}\{\bar{G}\}$
$X \equiv (F_1 \vee \ldots \vee F_n)$	$\{F_1 \vee \ldots \vee F_n\}$	$\{\bar{F}_1\} \ldots \{\bar{F}_n\}$
$X \equiv (F_1 \wedge \ldots \wedge F_n)$	$\{F_1\} \ldots \{F_n\}$	$\{\bar{F}_1 \vee \ldots \vee \bar{F}_n\}$

- $F \leftrightarrow G$ is treated as a conjunction or a disjunction

Benchmarks

	tauto	CNF_C	CNF_A	Tseitin	Tseitin2	Lazy	LazyN
hole3	–	0.72	0.06	0.24	0.21	0.06	0.05
hole4	–	3.1	0.23	3.5	6.8	0.32	0.21
hole5	–	10	2.7	80	–	1.9	1.8
deb5	83	–	0.04	0.15	0.10	0.09	0.03
deb10	–	–	0.10	0.68	0.43	0.66	0.09
deb20	–	–	0.35	4.5	2.5	7.5	0.35
equiv2	0.03	–	0.06	1.5	1.0	0.02	0.02
equiv5	61	–	–	–	–	0.44	0.42
franzen10	0.25	16	0.05	0.05	0.03	0.02	0.02
franzen50	–	–	0.40	1.4	0.80	0.34	0.35
schwicht20	0.48	–	0.12	0.43	0.23	0.10	0.10
schwicht50	8.8	–	0.60	4.3	2.2	0.57	0.7
partage	–	–	–	13	19	0.04	0.06
partage2	–	–	–	–	–	0.12	0.11

Results

Our contribution

- a tactic for propositional fragment of Coq
- outperforms existing tactic by orders of magnitude
- validates the lazy CNF conversion of our SMT solver
- improves on standard CNF conversion techniques
- intuitionnistic tactic using classical techniques
- solves the issue of "predicate definitions"
  $p(x_1, \ldots, x_n) \equiv \phi(x_1, \ldots, x_n)$
  $\Rightarrow$ should p be unfolded or not ?

For the daring souls

The whole development is documented and browsable at :

`http://www.lri.fr/~lescuyer/unsat`

Follow the checkmarks ✓ !

Thank You