Comparing and Integrating CVC4 and Alt-Ergo

Hanwen Wu and Wenxin Feng

Department of Computer Science
Boston University
hwwu,wenxinf@bu.edu

May 1, 2013

Abstract

This technical report summarize the abilities of CVC4 and Alt-Ergo by testing them using SMT-LIB 2.0 benchmarks within the theories of boolean, free functions, integers, bitvectors, quantifiers and inductions. The results show that CVC4 is both more powerful and more efficient than Alt-Ergo. It is complete in ....

1 Overview

Our main goal is thoroughly characterizing and comparing the abilities of two different SMT solvers, CVC4[1] and Alt-Ergo[3]. Since they both can take SMT-LIB 2.0[2] as their input language, we will also summarize it.

In this report, we will first of all, give a short introduction on SMT-LIB logic. Second, formally classifying input formulas using SMT-LIB logic. Third, introducing and summarizing the two solvers. Forth, carefully characterizing and comparing the abilities of the two solvers by testing them within different classes of formulas. And finally integrating them using a lightweight frontend written in C programming language.

2 SMT-LIB 2.0 Logic

In this section, we will briefly introduce SMT-LIB 2.0 logic so that we can more easily describe the classification of input formulas using SMT-LIB format later.

2.1 Theory

In the following, we are going to present some abstract informal definition of different theories in SMT-LIB 2.0. Note that the Core Theory is included in all other theories by default.

In all the figures, function symbols will only be applied to well-sorted terms according to their own function ranks/signatures/definitions.

2.1.1 Core

Core Theory is all about boolean sort and boolean functions/constants. It is the very base for all other theories.

2.1.2 Integer Theory

Integer Theory defines the integer domain, and operations over integers.

2.1.3 Fixed-Size Bit Vectors Theory

This theory declaration defines a core theory for fixed-size bitvectors where the operations of concatenation and extraction of bitvectors as well as the usual logical and arithmetic operations are overloaded[2].

sort	α	::=	`bool`
function	f	::=	**true** : `bool` | **false** : `bool`
		|	(**not** `bool`) : `bool` | (**and** `bool bool`) : `bool`
		|	(**or** `bool bool`) : `bool`
		|	(**xor** `bool bool`) : `bool`
		|	($\Rightarrow$ `bool bool`) : `bool` | ($=$ α α) : `bool`
		|	(**distinct** α α) : `bool` | (**ite** `bool` α α) : α
term	t	::=	**true** | **false**
		|	(**not** t) | (**and** t t) | (**or** t t) | (**xor** t t)
		|	($\Rightarrow$ t t) | ($=$ t t) | (**distinct** t t) | (**ite** t t t)

Table 1: Core Theory

2.2 Logic

2.2.1 Quantifier-Free Uninterpreted Functions

Closed quantifier-free formulas built over an arbitrary expansion of the Core signature with free sort and function symbols [2]. Users can define there own sorts and function symbols, but all of them are abstract. Functions can contain variables, but they must be bounded by **let** binder, so that the formulas are closed.

2.2.2 Quantifier-Free Linear Integer Arithmetic

Closed quantifier-free formulas built over an arbitrary expansion of the Integer Theory with free *constant* symbols, but whose terms of sort `int` are all linear [2]. Note that user can only define constants, not arbitrary functions who take one or more arguments. User can't define sort either. Also, non-linear functions like **div**, **mod**, **abs** and non-linear $\times$ are not allowed.

3 CVC4

CVC4, the fifth generation of Cooperating Validity Checker from NYU and U Iowa, is a DPLL(*T*) solver with a SAT solver core and a delegation path to different decision procedure implementations, each in charge of solving formulas in some background theory.[1]. It has implmented decision procedures for the theory of uninterpreted/free functions, arithmetic, arrays, bitvectors and inductive datatypes. It uses a combination method based on Nelson-Oppen method to cooperate different theories. Also, it supports quantifiers through heuristic instantiaion[1] and has the ability to generate model.

4 Alt-Ergo

Alt-Ergo uses a Simplex-based extension of Fourier-Motzkin for solving linear integer arithmetic, and it is sound and complete when it is quantifier free[4].

[1] See `http://cvc4.cs.nyu.edu/wiki/About_CVC4`

$$
\begin{array}{rlll}
\text{sort} & \alpha & ::= & \texttt{bool} \mid \texttt{int} \\
\\
\text{function} & f & ::= & \ldots \\
 & & \mid & \mathbb{Z} : \texttt{int} \\
 & & \mid & (- \ \texttt{int}) : \texttt{int} \mid (- \ \texttt{int} \ \texttt{int}) : \texttt{int} \\
 & & \mid & (+ \ \texttt{int} \ \texttt{int}) : \texttt{int} \mid (\times \ \texttt{int} \ \texttt{int}) : \texttt{int} \\
 & & \mid & (\textbf{div} \ \texttt{int} \ \texttt{int}) : \texttt{int} \mid (\textbf{mod} \ \texttt{int} \ \texttt{int}) : \texttt{int} \\
 & & \mid & (\textbf{abs} \ \texttt{int}) : \texttt{int} \\
 & & \mid & (\leqslant \ \texttt{int} \ \texttt{int}) : \texttt{bool} \mid (< \ \texttt{int} \ \texttt{int}) : \texttt{bool} \\
 & & \mid & (\geqslant \ \texttt{int} \ \texttt{int}) : \texttt{bool} \mid (> \ \texttt{int} \ \texttt{int}) : \texttt{bool} \\
 & & \mid & ((\_ \ \textbf{divisible} \ n) \ \texttt{int}) : \texttt{bool} \qquad (n \text{ is a positive integer}) \\
\\
\text{term} & t & ::= & \ldots \\
 & & \mid & \ldots \quad -1, 0, 1 \quad \ldots \\
 & & \mid & (-\, t) \mid (-\, t\ t) \mid (+\, t\ t) \mid (\times\, t\ t) \\
 & & \mid & (\textbf{div}\ t\ t) \mid (\textbf{mod}\ t\ t) \mid (\textbf{abs}\ t) \\
 & & \mid & (\leqslant t\ t) \mid (< t\ t) \mid (\geqslant t\ t) \mid (> t\ t) \\
 & & \mid & ((\_ \ \textbf{divisible} \ n)\ t)
\end{array}
$$

Table 2: Integer Theory

References

[1] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovi, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Proceedings of the 23rd international conference on Computer aided verification*, CAV'11, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] Clark Barrett, Aaron Stump, and Cesare Tinelli. The satisfiability modulo theories library (SMT-LIB). *www.smtlib.org*, 2010.

[3] François Bobot, Sylvain Conchon, Évelyne Contejean, Mohamed Iguernelala, Stéphane Lescuyer, and Alain Mebsout. The Alt-Ergo automated theorem prover, 2008. `http://alt-ergo.lri.fr/`.

[4] François Bobot, Sylvain Conchon, Evelyne Contejean, Mohamed Iguernelala, Assia Mahboubi, Alain Mebsout, and Guillaume Melquiond. A Simplex-Based Extension of Fourier-Motzkin for Solving Linear Integer Arithmetic. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *6th International Joint Conference on Automated Reasoning*, volume 7364 of *Lecture Notes in Computer Science*, pages 67–81, Manchester, Royaume-Uni, 2012. Springer.

sort	α	::=	`bool`
		|	(_ `BitVec` m) (m is a positive integer, we use `bv` for short)
function	f	::=	...
		|	`#b`X : `bv` (all binary constants)
		|	`#x`X : `bv` (all hexadeximal constants)
		|	(**concat** `bv bv`) : `bv`
		|	((_ **extract** i j) `bv`) : `bv` (i, j specify the range)
		|	(**bvnot** `bv`) : `bv` | (**bvneg** `bv`) : `bv`
		|	(**bvand** `bv bv`) : `bv` | (**bvor** `bv bv`) : `bv`
		|	(**bvadd** `bv bv`) : `bv` | (**bvmul** `bv bv`) : `bv`
		|	(**bvudiv** `bv bv`) : `bv` | (**bvurem** `bv bv`) : `bv`
		|	(**bvshl** `bv bv`) : `bv` | (**bvlshr** `bv bv`) : `bv`
		|	(**bvult** `bv bv`) : `bool`
term	t	::=	...
		|	`#b`X (all binary constants)
		|	`#x`X (all hexadeximal constants)
		|	(**concat** t t) | ((_ **extract** i j) t)
		|	(**bvnot** t) | (**bvneg** t) | (**bvand** t t) | (**bvor** t t)
		|	(**bvadd** t t) | (**bvmul** t t) | (**bvudiv** t t) | (**bvurem** t t)
		|	(**bvshl** t t) | (**bvlshr** t t) | (**bvult** t t)

Table 3: Fixed-Size Bitvectors Theory

sort	α	::=	$\ldots \mid \alpha' \, (\alpha^*)$ (user defined, abstract)
function	f	::=	$\ldots \mid (f' \, \alpha^*) : \alpha$ (user defined, abstract)
term	t	::=	...
		|	(**let** (bindings$^+$) t)
		|	$(f \, t^*)$

Table 4: QF-UF Logic

$$
\begin{array}{rcll}
\text{sort} & \alpha & ::= & \texttt{bool} \mid \texttt{int} \\
\\
\text{function} & f & ::= & \ldots \mid f' : \alpha \qquad \text{(user defined constant)} \\
\\
\text{term} & t & ::= & \ldots \\
 & & \mid & \ldots \quad -1, 0, 1 \quad \ldots \\
 & & \mid & (-\ t) \mid (-\ t\ t) \mid (+\ t\ t) \\
 & & \mid & (\times\ c\ t) \mid (\times\ t\ c) \qquad (c \text{ is an integer literal}) \\
 & & \mid & (\leqslant t\ t) \mid (< t\ t) \mid (\geqslant t\ t) \mid (> t\ t) \\
 & & \mid & (\ (\_\ \ \textbf{divisible}\ \ n\ )\ t\ ) \\
 & & \mid & (\ \textbf{let}\ (\ \text{bindings}^{+}\ )\ t\ )
\end{array}
$$

Table 5: QF-LIA Logic