Comparing and Integrating CVC4 and Alt-Ergo

Hanwen Wu and Wenxin Feng

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

May 3, 2013

Abstract

This technical report summarizes the abilities of CVC4 and Alt-Ergo by testing them using SMT-LIB 2.0 benchmarks within the theories of boolean, free functions, integers, bit-vectors, 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[BCD$^+$11] and Alt-Ergo[BCC$^+$08]. Since they both can take SMT-LIB 2.0[BST10a] 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. It is developed as a standard logic to describe theories, input languages, and output languages. Currently it is supported by a variety of SMT solvers. They also hold competitions for SMT solvers every year, and have a collection of benchmarks which we will use for our tests.

2.1 An Introduction

Since we are going to use SMT-LIB 2.0 logic to describt the classification of formulas, it is necessary to understand the SMT-LIB 2.0 logic itself, which can be used as an input language for both solvers (Alt-Ergo, CVC4).

SMT-LIB 2.0 is basically a version of many-sorted first-order logic with equality[BST10a]. It provides us the ability to write formulas, define theories and logics, and interact with provers using scripts. Provers that support SMT-LIB 2.0 should implement required funcionalieis and use correct semantics.

2.1.1 Sets of Symbols

These are part of the sets defined by SMT-LIB 2.0[BST10b]. They are alphabets of the logic, namely, the sources of symbols. These symbols will be used in the following subsections.

- $\mathcal{S}$: Infinite set of sort symbols, containing `bool`.
- $\mathcal{U}$: Infinite set of sort parameters.
- $\mathcal{X}$: Infinite set of variables.
- $\mathcal{F}$: Infinite set of function symbols.
- $\mathcal{B}$: Boolean values {**true, false**}.
- $\vdots$

2.1.2 Sorts

SMT-LIB 2.0 is a sorted logic. Sorts over a set of sort symbols $\mathcal{S}$ are defined as Sort($\mathcal{S}$). Sorts are defined inductively as follows.

- $\sigma \in \mathcal{S}$ of arity 0 is a sort.
- $\sigma, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n$ is a sort if $\sigma \in \mathcal{S}$ is of arity n, and σ_1 to σ_n are sorts.

The second item uses sorts to form new sorts. A list of integers can be a good example.

2.1.3 Signature

Baiscly, a signature Σ defines sort symbols and arities, function symbols and ranks, some variables and their sorts.

- $\Sigma^{\mathcal{S}} \subset \mathcal{S}$: sort symbols, containing `bool`.

- $\Sigma^{\mathcal{F}} \subset \mathcal{F}$: function symbols, containing equality, conjunction, and negation.
- $\Sigma^{\mathcal{S}}$ to $\mathbb{N}$: a total mapping from sort symbol to its arity, including $\texttt{bool} \Rightarrow \texttt{0}$.
- $\Sigma^{\mathcal{F}}$ to Sort($\Sigma^{\mathcal{S}}$)+: a left total mapping from a function symbol to its rank, containing $= (\sigma, \sigma, \texttt{bool})$, $\neg(\texttt{bool}, \texttt{bool})$, $\wedge(\texttt{bool}, \texttt{bool}, \texttt{bool})$.
- $\mathcal{X}$ to Sort($\Sigma^{\mathcal{S}}$): a partial mapping from a variable to its sort.

In the logic definitions of SMT-LIB, we will see "expended sigmature" a lot. We will formalize the expension methods in later sections.

2.1.4 Formulas

In SMT-LIB 2.0, formulas are well sorted terms of sort `bool` over Σ. In actuall scripts, all the formulas are begin treated as closed formulas. This is possible since non-closed formulas can be quantified using extential quantifier, as far as its satisfiability is concerned.

2.1.5 Structure

A structure $\mathbf{A}$ in SMT-LIB 2.0 can be regarded as a model. It is defined as a tuple.

$$\mathbf{A} = \{A, \sigma^{\mathbf{A}}[1]\ , (f : \sigma)^{\mathbf{A}}, (f : \sigma_1, \sigma_2, \ldots, \sigma_n, \sigma)^{\mathbf{A}}\}$$

And the meaning of these four elements are the followings.

- A: the universe (of values) of $\mathbf{A}$, including $\texttt{bool}^{\mathbf{A}} = \{$**true, false**$\}$.
- $\sigma^{\mathbf{A}} \subset A$: gives the sort $\sigma \in$ Sort($\Sigma^{\mathcal{S}}$) a universe $\sigma^{\mathbf{A}} \subset A$. For example, $\texttt{bool}^{\mathbf{A}}$ is $\{$**true, false**$\} \subset A$. $\texttt{int}^{\mathbf{A}}$ could be all the integers $\mathbb{Z} \subset A$.
- $(f : \sigma)^{\mathbf{A}} \in \sigma^{\mathbf{A}}$: gives the constant symbol $f : \sigma$ a value in the universe of σ
- $(f : \sigma_1, \sigma_2, \ldots, \sigma_n, \sigma)^{\mathbf{A}}$: defines the function symbol as a relation from $(\sigma_1, \sigma_2, ..., \sigma_n)^{\mathbf{A}}$ to $\sigma^{\mathbf{A}}$. This must include the equality relations (or identity predicate over $\sigma^{\mathbf{A}}$, that is $= (\sigma, \sigma, \texttt{bool})$ as standard equality relations from $(\sigma^{\mathbf{A}}, \sigma^{\mathbf{A}})$ to $\{$**true, false**$\}$).

2.1.6 Valuation and Interpretation

Valuation v is a partial mapping from $\mathcal{X} \times$Sort($\Sigma^{\mathcal{S}}$) to $\sigma^{\mathbf{A}}$. That is to give variable x of sort σ a value in $\sigma^{\mathbf{A}}$.

Interpretation $\mathcal{I}$ is defined as $\mathcal{I} = (\mathbf{A}, v)$, that is the structure together with the valuation make the Σ-interpretation.

[1] $\sigma^{\mathbf{A}}$ is called the extension of σ in $\mathbf{A}$.

$\mathcal{I}$ will assign meanings to well-sorted terms by uniquely mapping them into the $\mathbf{A}$. And that is the semantic.

As long as we have semantics, we can talk about satisfiability. If φ is mapped to **true** by some $\mathcal{I}$, then it is satisfiable. If φ is not closed, we say $\mathcal{I} = (\mathbf{A}, v)$ makes true φ. If φ is closed, we say the structure $\mathbf{A}$ makes true φ. (Since it doesn't matter what valuation it is), and that means $\mathbf{A}$ is a model of φ.

2.1.7 Theories

Theory is a very important concept here. SMT stands for Satisfiablity Modulo Theory, that is to check the satisfiability of a given logical formula within some background theories. Traditionally, a theory is a set of enough axioms, with which we can induct the formula. But here a theory $\mathcal{T}$ consists of three parts.

- Signature: Σ
- Models: A set of Σ-structures, all of which are models of the theory.
- Axioms: This is actually part of the models, and is left for the people who implement solvers. Take integer theory as an example. Since we have the plus sign in our signature (we just denote it as `ADD`, so that we know it is only a symbol, not the actual operation), we will have an axiom like $\forall x : \texttt{int}.\forall y : \texttt{int}.\exists z : \texttt{int}.\texttt{ADD}(x, y, z) \leftrightarrow x + y = z$. Therefore, our model (or structure) must contain the correct relations to map `ADD` to the actual addition operation to satisfy this axiom. Also, some theories, like real numbers, include those axioms as plain text, like associativity, commutativity, etc.

The SMT-LIB 2.0 standard has defined six theories. They are Core (for propositional logic), Integer, Real, Real and Integer, Fixed Size Bit-Vector, and Arrays. Each of them defines corresponing signature, and models. The actually implmentations are left for the provers.

2.1.8 Logics

Logic in SMT-LIB is also very important. It is a sublogic of SMT-LIB logic with restrictions, and is based on some theories. Common restrictions are

- fixing a signature Σ and its theory $\mathcal{T}$
- restricting structures to the models of $\mathcal{T}$
- restricting input sentences as subset of Σ-sentences

The SMT-LIB standard has classify formulas into many well-defined logics, including QF-UF, QF-LIA, QF-NIA, QF-IDL, QF-LRA, QF-NRA, QF-RDL, QF-BV, QF-AX, etc. We will not discuss them all, but focusing on integers and fixed-size bit-vectors.

2.2 Theory

In the following, we are going to present some abstract 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.

2.2.1 Core Theory

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

Beyond propositional logic, there are two more features in the Core theory. The first is equality/distinction. These two function symbols are defined not only for `bool`, but also for all potential sorts in an expended signature. The second is **ite**, which is the **if** − **then** − **else** operator. It is also defined for other sorts.

sort	α	$::=$	$\texttt{bool}$
function	f	$::=$	$\textbf{true} : \texttt{bool} \mid \textbf{false} : \texttt{bool}$
		$\mid$	$(\textbf{not}\ \ \texttt{bool}) : \texttt{bool} \mid (\textbf{and}\ \ \texttt{bool bool}) : \texttt{bool}$
		$\mid$	$(\textbf{or}\ \ \texttt{bool bool}) : \texttt{bool}$
		$\mid$	$(\textbf{xor}\ \ \texttt{bool bool}) : \texttt{bool}$
		$\mid$	$(\Rightarrow\ \ \texttt{bool bool}) : \texttt{bool} \mid (=\ \ \alpha\ \alpha) : \texttt{bool}$
		$\mid$	$(\textbf{distinct}\ \ \alpha\ \alpha) : \texttt{bool} \mid (\textbf{ite}\ \ \texttt{bool}\ \alpha\ \alpha) : \alpha$
term	t	$::=$	$\textbf{true} \mid \textbf{false}$
		$\mid$	$(\textbf{not}\ t) \mid (\textbf{and}\ t\ t) \mid (\textbf{or}\ t\ t) \mid (\textbf{xor}\ t\ t)$
		$\mid$	$(\Rightarrow t\ t) \mid (= t\ t) \mid (\textbf{distinct}\ t\ t) \mid (\textbf{ite}\ t\ t\ t)$

Table 1: Core Theory

2.2.2 Integer Theory

Integer Theory defines the integer domain, and operations over integers. It is a superset of Core theory, thus includs all the sorts and function symbols defined in Core theory.

Note that the Integer theory itself doesn't have any restriction on linear or non-linear operations. They should instead be defined in logics based on Integer theory. Also, the division, modulo operations here are defined for integers which actually involve flooring and ceiling.

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

Table 2: Integer Theory

2.2.3 Fixed-Size Bit-Vectors Theory

This theory defines a series of sorts for different size of bit-vectors. Concatenation and extraction of bit-vectors, and the usual logical and arithmetic operations are also defined. The universe of bit-vectors theory is those numeral constants in bit-vector format. They are defined using a SMT-LIB syntax of the form `#b`X and `#x`X for binary and hexadeximal constants.

In the table, we use `bv` for (_ `BitVec` m), and omitting the size of the bit-vectors, only for layout reasons.

2.3 Logic

Logic is the main tool we use for classifying formulas and testing solver abilities. In the followings, we will formalize the definition of various logics of SMT-LIB.

2.3.1 Quantifier-Free Uninterpreted Functions

Closed quantifier-free formulas built over an arbitrary expansion of the Core signature with free sort and function symbols [BST10a]. Users can define there own sorts and function

sort	α	::=	`bool`
		|	(_ `BitVec` m) (m is a positive integer, we use `bv` for short)
function	f	::=	...
		|	`#bX` : `bv` (all binary constants)
		|	`#xX` : `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	::=	...
		|	`#bX` (all binary constants)
		|	`#xX` (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 Bit-Vectors Theory

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.

2.3.2 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 [BST10a]. 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 × are not allowed.

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

Table 4: QF-UF Logic

sort	α	::=	`bool` $\mid$ `int`	
function	f	::=	$\ldots \mid f' : \alpha$	(user defined constant)
term	t	::=	$\ldots$	
		$\mid$	$\ldots \quad -1, 0, 1 \quad \ldots$	
		$\mid$	$(- \ t) \mid (- \ t \ t) \mid (+ \ t \ t)$	
		$\mid$	$(\times \ c \ t) \mid (\times \ t \ c)$	(c is an integer literal)
		$\mid$	$(\leqslant \ t \ t) \mid (< \ t \ t) \mid (\geqslant \ t \ t) \mid (> \ t \ t)$	
		$\mid$	((_ **divisible** n) t)	
		$\mid$	(**let** (bindings$^+$) t)	

Table 5: QF-LIA Logic

3 Comparing CVC4 and Alt-Ergo

In this section, we will give a short summary of both solvers first. Their overall architectures, built-in theories, combination methods, and unique features will be briefly discussed. Then we will summarize our test methods and results to show their abilities within different sublogics using SMT-LIB 2.0 benchmarks.

3.1 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[BCD$^+$11]. It works for first-order logics. It has implmented decision procedures for the theory of uninterpreted/free functions, arithmetic(integer, real, linear, non-linear), arrays, bit-vectors and datatypes. It uses a combination method based on Nelson-Oppen to cooperate different theories. Also, it supports quantifiers through heuristic instantiaion[2] and has the ability to generate model. By our tests of k-induction over linear integer arithmetic, it supports induction very well.

For both satisfiable/unsatisfiable formulas, CVC4 will come up with the correct answer

3.2 Alt-Ergo

Alt-Ergo is dedicated to program verification. It works in first-order logic. It uses a CC(X)[3], a variant of Shostak algorithm, to combine free theory with equality and an arbitrary solvable built-in theory X[Con]. Alt-Ergo has implemented decision procedures for the theory of uninterpreted/free functions, arithmetic(integer, real, linear, non-linear), arrays, bit-vectors, datatypes, etc. It also has direct support for polymorphism in its native input language. Associative and commutative symbols are being handled specially using its AC(X) theory to boost the performance. It has some support for universal and extential quantifiers through instantiation. It has the ability to generate proof. Also, by our test of k-induction, it can prove them quickly.

For unsatisfiable formulas, Alt-Ergo will eventually answer `unsat` correctly. But for satisfiable formulas, it never answers `sat`, but `unknown`.

Since integer theory are intensively used in program verification,

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[BCC$^+$12].

3.3

References

[BCC$^+$08] 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/`.

[BCC$^+$12] 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.

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

[3] CC(X): Congruence closure modulo X

[BCD$^+$11] 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.

[BST10a] Clark Barrett, Aaron Stump, and Cesare Tinelli. The satisfiability modulo theories library (SMT-LIB). *www.smtlib.org*, 2010.

[BST10b] Clark Barrett, Aaron Stump, and Cesare Tinelli. The smt-lib standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*, volume 13, 2010.

[Con] Sylvain Conchon. *SMT Techniques and their Applications: from Alt-Ergo to Cubicle*. PhD thesis.