The CVC4 SMT Solver

Martin Brain on behalf of Morgan Deters

February 2, 2015

Morgan Deters

http://cvc4.cs.nyu.edu/web/in-memoriam-morgan-deters/

The CVC4 Team



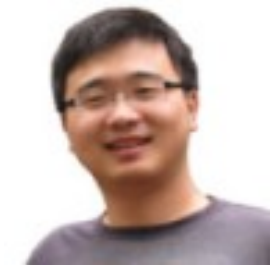
Clark Barrett (NYU)
Cesare Tinelli (U Iowa)
Morgan Deters (NYU)



Kshitij Bansal (NYU)
François Bobot (CEA)
Chris Conway (Google)



Liana Hadarean (NYU)
Dejan Jovanović (SRI)
Tim King (Verimag)



Tianyi Liang (U Iowa)
Andrew Reynolds (EPFL)
Nestan Tsiskaridze (U Iowa)

Does this hold? Does this not hold?

$$\begin{aligned} 0 &< x \\ 0 &< y \\ x + y &< x \end{aligned}$$

Does this hold? Does this not hold?

$$0 \clubsuit x$$

$$0 \clubsuit y$$

$$x \spadesuit y \clubsuit x$$

First Order Logic

Syntax

Fix a *signature* Σ
(i.e. $\Sigma = \{\clubsuit, \spadesuit\}$)

Semantics

An *interpretation* is
$M = (D, [\![.]\!] : \Sigma \to (2^{D^n}))$

Satisfiability

An interpretation M *satisfies* a formula ϕ:

$$M \models \phi$$

If ϕ evaluated over D (using $[\![.]\!]$) is true.

Does this hold? Does this not hold?

$$\begin{aligned} 0 &\clubsuit x \\ 0 &\clubsuit y \\ x \spadesuit y &\clubsuit x \end{aligned}$$

It depends on the interpretation (of $\clubsuit$ and $\spadesuit$)!

$$\begin{aligned} D &= \mathbb{Z} \\ [\![\clubsuit]\!] &= <_{\mathbb{Z}} \\ [\![\spadesuit]\!] &= +_{\mathbb{Z}} \end{aligned}$$

Does this hold? Does this not hold?

$$\begin{array}{rcl} 0 & \clubsuit & x \\ 0 & \clubsuit & y \\ x \spadesuit y & \clubsuit & x \end{array}$$

It depends on the interpretation (of $\clubsuit$ and $\spadesuit$)!

$$\begin{array}{rcl} D & = & \{00, 01, 10, 11\} \\ [\![\clubsuit]\!] & = & \texttt{bvult} \\ [\![\spadesuit]\!] & = & \texttt{bvplus} \end{array}$$

How Do We Fix The *Meaning* of Symbols?

Option 1 – Axiomatic

$$M \models Axioms \Rightarrow \phi$$

$$\begin{aligned} axioms &= \forall a, b, c \,.\, a \clubsuit b \wedge b \clubsuit c \Rightarrow a \clubsuit c \\ &= \forall a \,.\, \neg a \clubsuit a \\ &= \ldots \end{aligned}$$

Pros

+ Easy to implement
+ Flexible
+ Can add theorems

Cons

- All formulae quantified
- Axioms not always simple
- Hard to solve

How Do We Fix The *Meaning* of Symbols?

Option 2 – Algebraic

Fix signature Σ' and its interpretation $M' = (D, [\![.]\!] : \Sigma' \to (2^{D^n}))$.

$$D = \mathbb{Z} \quad [\![\clubsuit]\!] = <_{\mathbb{Z}} \quad [\![\spadesuit]\!] = +_{\mathbb{Z}}$$

Is there M extension of M' such that:

$$M \models \phi$$

Pros

+ Fast decision procedures
+ Counter-examples
+ Few quantifiers

Cons

- Theory has to be built into solver
- Implementation harder

How Do We Fix The *Meaning* of Symbols?

Option 2 – Algebraic

Fix signature Σ' and its interpretation $M' = (D, [\![.]\!] : \Sigma' \rightarrow (2^{D^n}))$.

$$D = \mathbb{Z} \quad [\![\clubsuit]\!] = <_{\mathbb{Z}} \quad [\![\spadesuit]\!] = +_{\mathbb{Z}}$$

Is there M extension of M' such that:

$$M \models \phi$$

Satisfiability Modulo Theories (SMT)

Pros

+ Fast decision procedures
+ Counter-examples
+ Few quantifiers

Cons

- Theory has to be built into solver
- Implementation harder

The SMT-LIB Initiative

`http://smt-lib.org`

- International initiative
- Rigorously standardise descriptions of theories for SMT
  Arithmetic ($\mathbb{Z}$ and $\mathbb{R}$), arrays, bit-vectors, floating-point
  (in preparation strings, data-types, sets ...)
- Promote common syntax for SMT interactions
- Benchmarks
- Annual competition

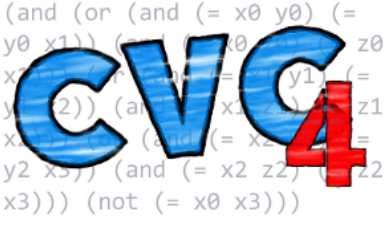
SMT Solvers

Platform			Features						Notes
Name	OS	License	SMT-LIB	CVC	DIMACS	Built-in theories	API	SMT-COMP [2]	
ABsolver	Linux	CPL	v1.2	No	Yes	linear arithmetic, non-linear arithmetic	C++	no	DPLL-based
Alt-Ergo	Linux, Mac OS, Windows	CeCILL-C (roughly equivalent to LGPL)	partial v1.2 and v2.0	No	No	empty theory, linear integer and rational arithmetic, non-linear arithmetic, polymorphic arrays, enumerated datatypes, AC symbols, bitvectors, record datatypes, quantifiers	OCaml	2008	Polymorphic first-order input langu SAT-solver based, combines Shos Nelson-Oppen like approaches for modulo theories
Barcelogic	Linux	Proprietary	v1.2			empty theory, difference logic	C++	2009	DPLL-based, congruence closure
Beaver	Linux, Windows	BSD	v1.2	No	No	bitvectors	OCaml	2009	SAT-solver based
Boolector	Linux	GPLv3	v1.2	No	No	bitvectors, arrays	C	2009	SAT-solver based
CVC3	Linux	BSD	v1.2	Yes		empty theory, linear arithmetic, arrays, tuples, types, records, bitvectors, quantifiers	C/C++	2010	proof output to HOL
CVC4	Linux, Mac OS, Windows	BSD	Yes	Yes		rational and integer linear arithmetic, arrays, tuples, records, inductive data types, bit-vectors, strings, and equality over uninterpreted function symbols	C++	2010	version 1.4 released July 2014
Decision Procedure Toolkit (DPT)	Linux	Apache	No				OCaml	no	DPLL-based
iSAT	Linux	Proprietary	No			non-linear arithmetic		no	DPLL-based
MathSAT	Linux	Proprietary	Yes		Yes	empty theory, linear arithmetic, bitvectors, arrays	C/C++, Python, Java	2010	DPLL-based
MiniSmt	Linux	LGPL	partial v2.0			non-linear arithmetic		2010	SAT-solver based, Yices-based
OpenSMT	Linux, Mac OS, Windows	GPLv3	partial v2.0		Yes	empty theory, differences, linear arithmetic, bitvectors	C++	2011	lazy SMT Solver
SatEEn	?	Proprietary	v1.2			linear arithmetic, difference logic	none	2009	
SMTInterpol	Linux, Mac OS, Windows	LGPLv3	v2.0			uninterpreted functions, linear real arithmetic, and linear integer arithmetic	Java	2012	Focuses on generating high quality interpolants.
SMCHR	Linux, Mac OS, Windows	GPLv3	No	No	No	linear arithmetic, nonlinear arithmetic, heaps	C	no	Can implement new theories using Handling Rules.
SMT-RAT	Linux, Mac OS	GPLv3	v2.0	No	No	linear arithmetic, nonlinear arithmetic	C++	no	Toolbox offering theory solver mod development of SMT solvers for n arithmetic (NRA). Example embedd OpenSMT available.
SONOLAR	Linux, Windows	Proprietary	partial v2.0			bitvectors	C	2010	SAT-solver based
Spear	Linux, Mac OS,	Proprietary	v1.2			bitvectors		2008	

CVC4 : The Cooperating Validity Checker, Version 4

- Full-fledged open-source SMT solver
- Jointly developed at NYU and U. Iowa
- Competitive with other top solvers
- Theories include linear arithmetic, arrays, bit-vectors, algebraic data-types, uninterpreted functions, quantifiers
- Support for CVC, SMT-LIB 1.2/2.0/2.5, TPTP (FOF) input formats

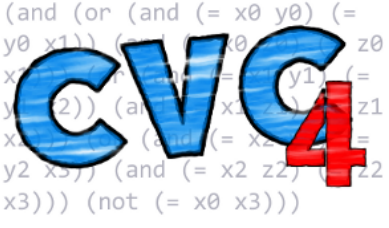
http://cvc4.cs.nyu.edu/



History of CVC

SVC	1996, own SAT solver
CVC	Chaff, optimized internal design
CVC Lite	2003, rewrite to make more flexible supported quantifiers
CVC3	major overhaul better DP implementations
CVC4	first stable release 2012 complete redesign of internal architecture significant performance improvements

http://cvc4.cs.nyu.edu/



Using CVC4

Stand-alone tool

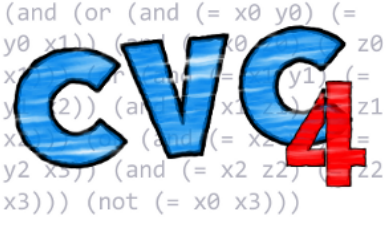
- SMT-LIB 1.2, 2.0, 2.5 (plus Z3's extensions)
- Native CVC language
- TPTP format

Library

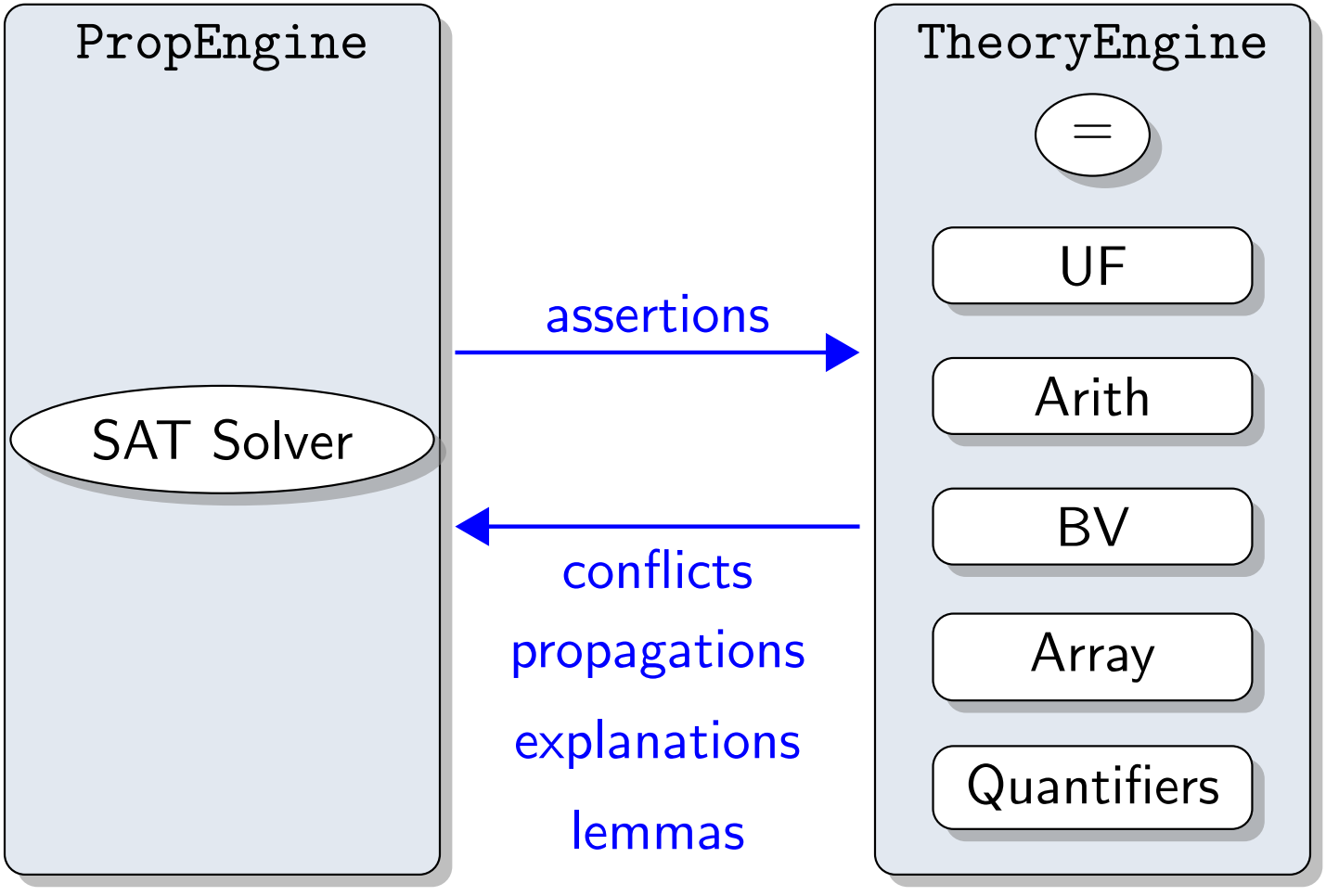
Bindings for:

- C++
- Java
- OCaml

http://cvc4.cs.nyu.edu/



High-Level Architecture



Quantifiers

Compactness

ϕ is unsatisfiable

$\Leftrightarrow$

There is a finite set S of instantiation of the quantifiers that is also unsatisfiable

How do we find S? How do we know when we have found it?

E-Matching

E-Matching

Use unification to identify which terms could be relevant

Optimisations

- Minimise the number of new terms created
- Equality aware (one unification per congruence class)
- Prefer instantiations that create unit clauses
- Theory propagation aware

$$\psi(x, y, z) \wedge \forall a, b \,.\, \phi(a, b) \quad \rightsquigarrow \quad \psi(x, y, z) \wedge \phi(x, y) \wedge \phi(x, 4)$$

Model Based Quantifier Instantiation

MBQI

1. Generate model of the ground part of ϕ
2. Search the space of quantification
3. Use counter-examples to pick instantiations

$$\begin{aligned} \psi(x,y,z) \wedge \forall a,b \,.\, \phi(a,b) &\rightsquigarrow \psi(x,y,z) \\ &\rightsquigarrow x = 1, y = 4, z = 23 \wedge \neg\phi(a,b) \\ &\rightsquigarrow \psi(x,y,z) \wedge \phi(17,42) \end{aligned}$$

Finite Model Finder

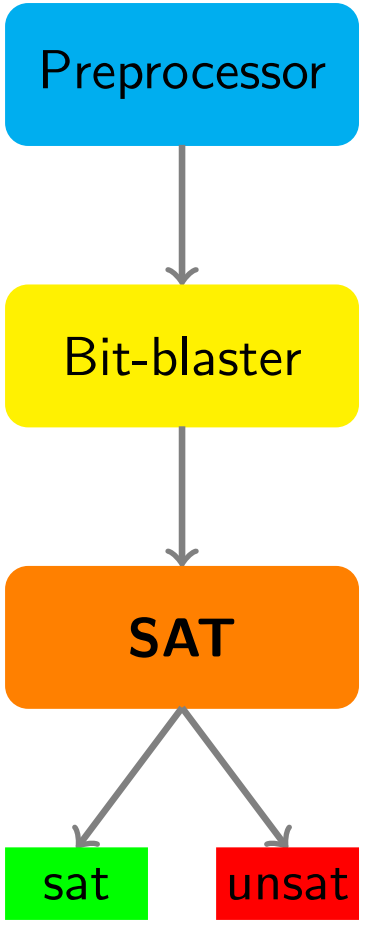
FMF

1. Generate a model of the ground part of ϕ
2. Minimise the number of equality classes of values
3. Complete instantiation for these values only

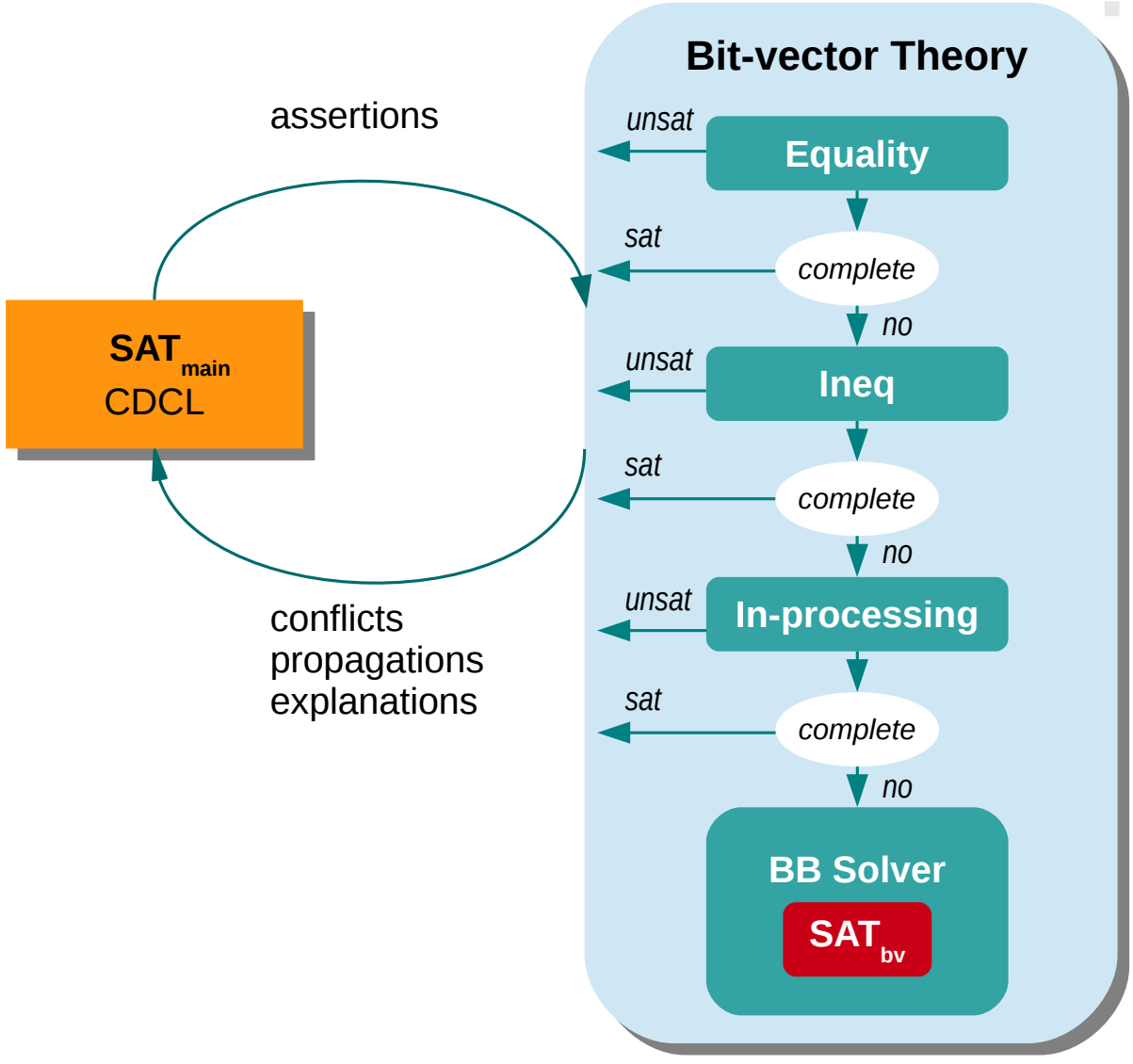
Also...

- Conflict-based instantiation
- Induction
- Local theory extensions
- Rewrite rules

Eager Bit-Vector Solver



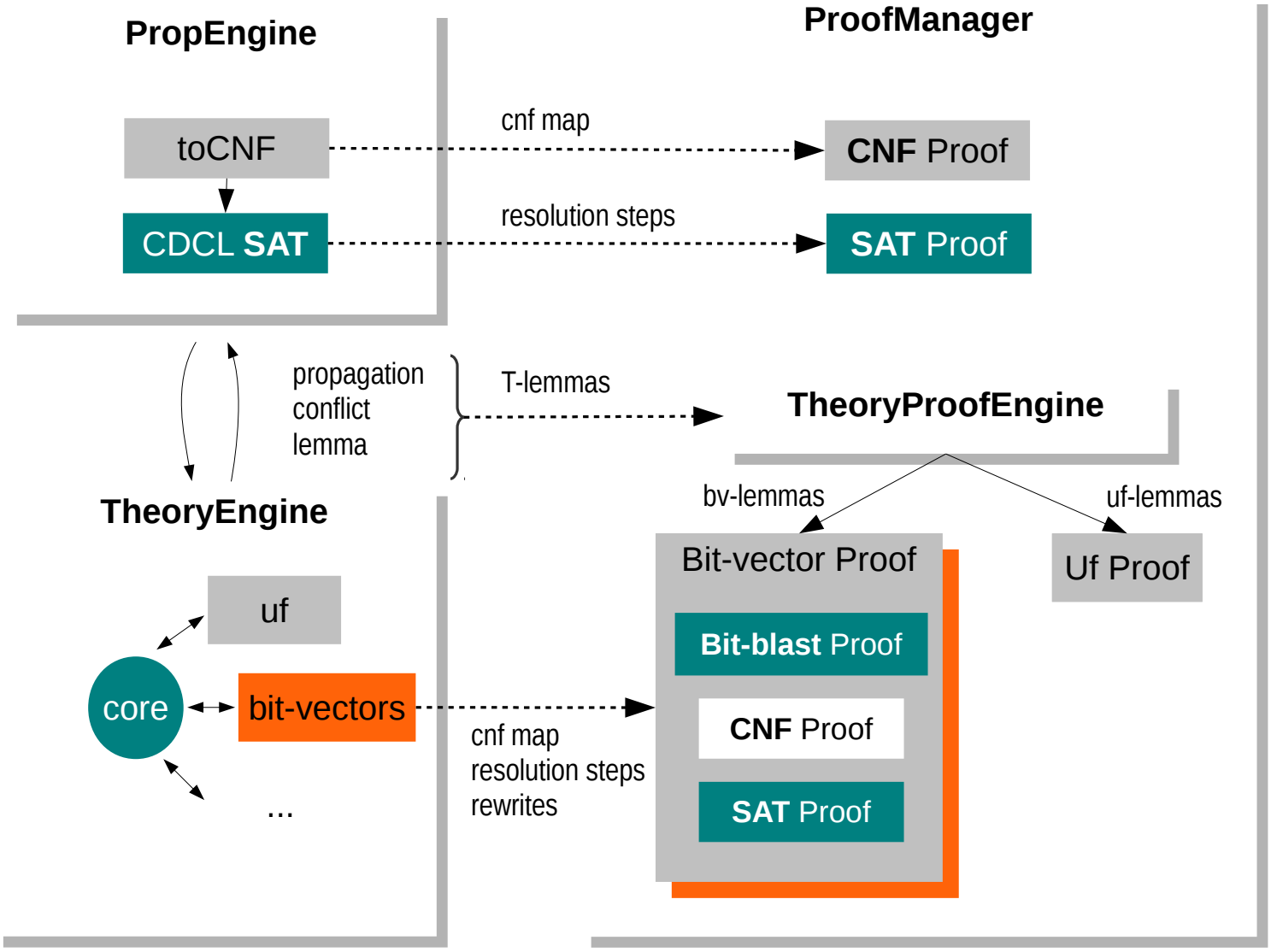
Lazy Bit-Vector Solver



Proof Generation

- Solver is beyond current functional verification.
- Instead, generate proof for unsatisfiable formulae
- Current proof uses LFSC meta-proof framework

Proof Module in CVC4



Getting CVC4

- Binaries for x86/Linux, x86/Windows, x64/Linux
  `http://cvc4.cs.nyu.edu/download/`
- Packaged in Debian and Fedora
- MacPorts package
- Build from source
  `https://github.com/CVC4/CVC4`
- Try it online!
  `http://cvc4.cs.nyu.edu/tryit/`

The Future

- New Theories:
  strings, finite sets, floating-point
- Unsatisfiable cores
- Proofs
- Quantifier elimination
- Optimisation problems

Conclusions

SMT Fix interpretation, not axiomatisation

CVC4 State-of-the-art, open source SMT solver

You? Always looking for new users and new collaborators!

Conclusions

SMT Fix interpretation, not axiomatisation

CVC4 State-of-the-art, open source SMT solver

You? Always looking for new users and new collaborators!

Thank you for your time and attention.