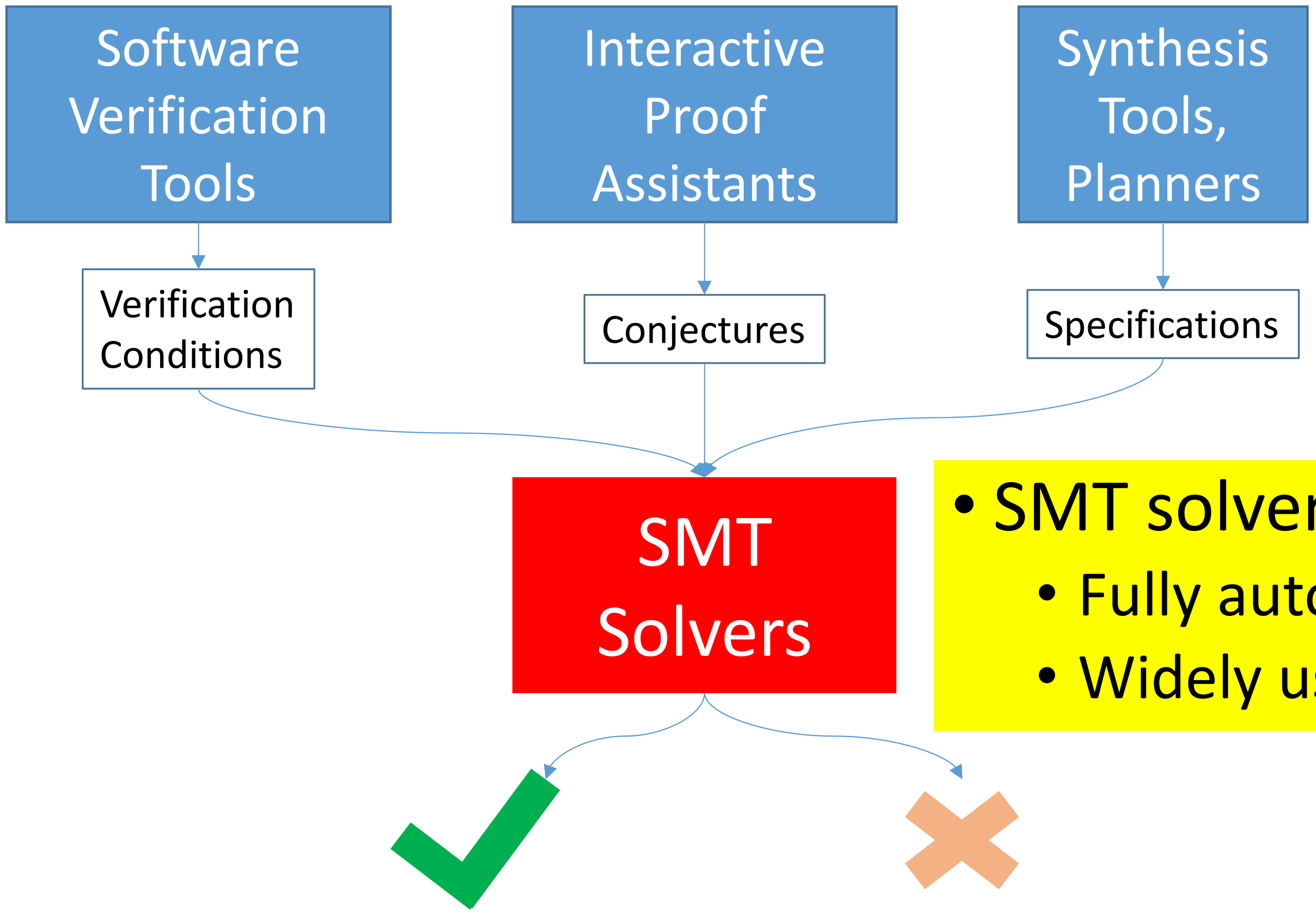
A Decision Procedure for (Co)datatypes in SMT Solvers

Andrew Reynolds

Jasmin Christian Blanchette

IJCAI sister conference track, July 12, 2016

Satisfiability Modulo Theories (SMT) Solvers



- SMT solvers are:
  - Fully automated reasoners
  - Widely used in applications

Common Theories Supported by SMT Solvers

- SMT solvers support:
  - Arbitrary Boolean combinations of ground *theory* constraints
  - Examples of supported theories:
    - Uninterpreted functions: `f(a)=g(b,c)`
    - Linear real/integer arithmetic: `a≥b+2*c+3`
    - Arrays: `select(A,i)=select(store(A,i+1,3),i)`
    - BitVectors: `bvule(x,#xFF)`
    - Algebraic Datatypes: `x,y:List; tail(x)=cons(0,y)`
    - …

Common Theories Supported by SMT Solvers

- SMT solvers support:
  - Arbitrary Boolean combinations of ground *theory* constraints
  - Examples of supported theories:
    - Uninterpreted functions: `f(a)=g(b,c)`
    - Linear real/integer arithmetic: `a≥b+2*c+3`
    - Arrays: `select(A,i)=select(store(A,i+1,3),i)`
    - BitVectors: `bvule(x,#xFF)`
    - Algebraic Datatypes: `x,y:List; tail(x)=cons(0,y)`

…Recurrent Questions:

- What theories are **useful in applications**?
- What theories can we **handle efficiently**?

Common Theories Supported by SMT Solvers

- SMT solvers support:
  - Arbitrary Boolean combinations of ground *theory* constraints
  - Examples of supported theories:
    - Uninterpreted functions: `f(a)=g(b,c)`
    - Linear real/integer arithmetic: `a≥b+2*c+3`
    - Arrays: `select(A,i)=select(store(A,i+1,3),i)`
    - BitVectors: `bvule(x,#xFF)`
    - **(Co)Algebraic Datatypes**: `x,y:List|Stream; tail(x)=cons(0,y)`
      - Are of importance to the interactive proof assistant Isabelle
      - Can be decided by an efficient decision procedure

Introductory Examples

Introductory Examples

datatype nat =

  Z

| S(nat)

datatype $list_\tau$ =

  Nil_τ

| $Cons_\tau(\ \tau, list_\tau)$

Introductory Examples

datatype nat =
  Z
| S(nat)

datatype list$_\tau$ =
  Nil$_\tau$
| Cons$_\tau$(τ, list$_\tau$)

codatatype enat =
  EZ
| ES(enat)

codatatype llist$_\tau$ =
  LNil$_\tau$
| LCons$_\tau$(τ, llist$_\tau$)

Introductory Examples

datatype nat =
  Z
| S(nat)

datatype list_τ =
  Nil_τ
| $\text{Cons}_\tau(\tau, \text{list}_\tau)$

codatatype enat =
  EZ
| ES(enat)

codatatype llist_τ =
  LNil_τ
| $\text{LCons}_\tau(\tau, \text{llist}_\tau)$

codatatype stream_τ =
  $\text{SCons}_\tau(\tau, \text{stream}_\tau)$

Introductory Examples

datatype nat =
  Z
| S(nat)

datatype list$_\tau$ =
  Nil$_\tau$
| Cons$_\tau$(τ, list$_\tau$)

Codatatypes need not be well-founded

codatatype enat =
  EZ
| ES(enat)

codatatype llist$_\tau$ =
  LNil$_\tau$
| LCons$_\tau$(τ, llist$_\tau$)

codatatype stream$_\tau$ =
  SCons$_\tau$(τ, stream$_\tau$)

Introductory Examples

Introductory Examples

$x \neq S(x)$

Introductory Examples

$x \neq S(x)$

$\exists x.\ x = ES(x)$

Introductory Examples

Cyclic values exist

$x \neq S(x)$

$\exists x.\ x = ES(x)$

Introductory Examples

$x \neq S(x)$

$\exists x.\ x = ES(x)$

Cyclic values exist

$x = ES(x)$

$y = ES(y)$

Introductory Examples

Cyclic values exist

$x \neq S(x)$

$\exists x.\ x = ES(x)$

$x = ES(x) = ES(ES(ES(\ldots)))$

$y = ES(y) = ES(ES(ES(\ldots)))$

Introductory Examples

Cyclic values exist

$x \neq S(x)$

$\exists x.\ x = ES(x)$

$$\left.\begin{array}{l} x = ES(x) = ES(ES(ES(\ldots))) \\ y = ES(y) = ES(ES(ES(\ldots))) \end{array}\right\} =$$

Introductory Examples

$x \neq S(x)$

…but they are equal up to their expansion

$\exists x.\ x = ES(x)$

Cyclic values exist

$$\left.\begin{array}{l} x = ES(x) = ES(ES(ES(\ldots))) \\ y = ES(y) = ES(ES(ES(\ldots))) \end{array}\right\} =$$

Introductory Examples

$x \neq S(x)$

...but they are equal up to their expansion

$\exists x.\ x = ES(x)$

Cyclic values exist

$$\left.\begin{aligned} x &= ES(x) = ES(ES(ES(\ldots))) \\ y &= ES(y) = ES(ES(ES(\ldots))) \end{aligned}\right\} =$$

μ-notation:

$xs = LCons(\ 1, \mu\ ys.\ LCons(\ 0, LCons(\ 9, ys\ )\ )$

denotes the infinite sequence 1,0,9,0,9,0,9,...

Introductory Examples

Introductory Examples

xs = LCons(0, LCons(1, LCons(2,…)))

Acyclic infinite values exist

Introductory Examples

datatype values =

all finite ground

constructor terms

(and only those)

xs = LCons(0, LCons(1, LCons(2,…)))

Acyclic infinite values exist

Introductory Examples

xs = LCons(0, LCons(1, LCons(2,…)))

Acyclic infinite values exist

datatype values =

all finite ground

constructor terms

(and only those)

codatatype values =

all finite **or infinite**

ground constructor terms

(and only those)

A Degenerate Case

- Recursive datatypes are infinite
- Corecursive codatatypes admit infinite values
- Ergo: corecursive codatatypes are infinite?

A Degenerate Case

- Recursive datatypes are infinite
- Corecursive codatatypes admit infinite values
- Ergo: corecursive codatatypes are infinite?

Counterexample:

codatatype $\text{stream}_{\text{unit}}$ = SCons(unit, $\text{stream}_{\text{unit}}$)

datatype unit = Unity

$\Rightarrow$ "Corecursive singletons"

Our contributions

- Generalized [Barrett et al. 2007] (used in CVC3) to codatatypes
  - First decision procedure for codatatypes in SMT solvers
- Efficient implementation in CVC4
- Evaluation on Isabelle benchmarks

Calculus for Theory of (Co)datatypes $\mathcal{DC}$

- **Inputs**:
  - A finite set of literals $\mathtt{E}$
- **Outputs**:
  - Either "$\mathtt{E}$ is unsatisfiable" or "$\mathtt{E}$ is satisfiable"

- Can be described as a set of derivation rules which
  - Add additional literals to $\mathtt{E}$ until saturated or conflict

Calculus: Guarded Assignment Form

- Derivation rules of calculus written in guarded assignment form:

$$\frac{\text{… premises on } \mathtt{E}\text{…}}{\mathtt{E\ :=\ E'}}\ [\text{rule}]$$

  - For example:

$$\frac{\mathtt{t{=}s, s{=}r \in E}}{\mathtt{E\ :=\ E, t{=}s}}\ \text{trans}$$

- Derivation rules may have $\perp$ as conclusion:

$$\frac{\mathtt{t{=}s, t{\neq}s \in E}}{\perp}\ \text{conflict}$$

Calculus: Derivation Tree

$$\cfrac{\cfrac{\texttt{t=s, s=r, t}\neq\texttt{r}}{\texttt{t=s, s=r, t}\neq\texttt{r, t=r}}\ \text{trans}}{\bot}\ \text{conflict}$$

- A node is a set of literals
- Each node obtained as result of successfully applying rule to parent
- The derivation terminates when:
  - All leaves are $\bot$ … input is unsatisfiable
  - Some node is saturated … input is satisfiable

Part 1: Bidirectional Closure

$$\frac{t \in \mathcal{T}(E)}{E := E, t \approx t}\ \textsf{Refl} \qquad \frac{t \approx u \in E}{E := E, u \approx t}\ \textsf{Sym} \qquad \frac{s \approx t,\ t \approx u \in E}{E := E, s \approx u}\ \textsf{Trans}$$

$$\frac{\bar{t} \approx \bar{u} \in E \quad \mathsf{f}(\bar{t}), \mathsf{f}(\bar{u}) \in \mathcal{T}(E)}{E := E, \mathsf{f}(\bar{t}) \approx \mathsf{f}(\bar{u})}\ \textsf{Cong} \qquad \frac{t \approx u,\ t \not\approx u \in E}{\bot}\ \textsf{Conflict}$$

$$\frac{\mathsf{C}(\bar{t}) \approx \mathsf{C}(\bar{u}) \in E}{E := E, \bar{t} \approx \bar{u}}\ \textsf{Inject} \qquad \frac{\mathsf{C}(\bar{t}) \approx \mathsf{D}(\bar{u}) \in E \quad \mathsf{C} \neq \mathsf{D}}{\bot}\ \textsf{Clash}$$

- $\mathbb{E}$ contains its upwards (congruence) and downwards (unification) closure

Part 2: Acyclicity and Uniqueness

- To determine datatype values are acyclic, codatatype values are unique:
  - Compute the class of values $\mathcal{A}[\texttt{t}]$ for each (co)datatype term $\texttt{t}$

  - For example:

    $\mathcal{A}[\texttt{t}] := \mu\texttt{x}.\texttt{C}(\texttt{x})$: the value of $\texttt{t}$ is $\texttt{C}(\texttt{C}(\texttt{C}(\ldots)))$

    $\mathcal{A}[\texttt{t}] := \mu\texttt{x}.\texttt{C}(\texttt{y})$: the top symbol of $\texttt{t}$ is $\texttt{C}$

Part 2: Acyclicity and Uniqueness

$$\frac{\delta \in \mathcal{Y}_{\mathrm{dt}} \quad \mathcal{A}[t^{\delta}] = \mu x.\, u \quad x \in \mathrm{FV}(u)}{\perp}\ \text{Acyclic} \qquad \frac{\delta \in \mathcal{Y}_{\mathrm{codt}} \quad \mathcal{A}[t^{\delta}] =_{\alpha} \mathcal{A}[u^{\delta}]}{E := E, t \approx u}\ \text{Unique}$$

- Rule Acyclic:
  - Checks whether $\mathcal{A}$[t] contains a bound variable for some datatype term t
  - For example: E={x=S(x)}
    - $\mathcal{A}[\mathtt{x}] = \mu\tilde{\mathtt{x}}.\mathtt{S}(\tilde{\mathtt{x}})$, thus $\mathtt{E} \models_{\mathcal{DC}} \perp$
- Rule Unique:
  - Checks whether $\mathcal{A}$[t], $\mathcal{A}$[u] are α-equivalent for some codatatype terms t, u
  - For example: E={x=C(x), y=C(y)}
    - $\mathcal{A}[\mathtt{x}] = \mu\tilde{\mathtt{x}}.\mathtt{C}(\tilde{\mathtt{x}}) =_{\alpha} \mu\tilde{\mathtt{y}}.\mathtt{C}(\tilde{\mathtt{y}}) = \mathcal{A}[\mathtt{y}]$, thus $\mathtt{E} \models_{\mathcal{DC}} \mathtt{x}=\mathtt{y}$

Part 3: Splitting

$$\frac{t^{\delta} \in \mathcal{T}(E) \quad \mathcal{F}^{\delta}_{\text{ctr}} = \{\mathsf{C}_1, \ldots, \mathsf{C}_m\} \\ \big(\mathsf{s}(t) \in \mathcal{T}(E) \text{ and } \mathsf{s} \in \mathcal{F}^{\delta}_{\text{sel}}\big) \text{ or } \big(\delta \in \mathcal{Y}_{\text{dt}} \text{ and } \delta \text{ is finite}\big)}{E := E, t \approx \mathsf{C}_1\big(\mathsf{s}^1_1(t), \ldots, \mathsf{s}^{n_1}_1(t)\big) \quad \cdots \quad E := E, t \approx \mathsf{C}_m\big(\mathsf{s}^1_m(t), \ldots, \mathsf{s}^{n_m}_m(t)\big)} \text{ Split}$$

$$\frac{t^{\delta}, u^{\delta} \in \mathcal{T}(E) \quad \delta \in \mathcal{Y}_{\text{codt}} \quad \delta \text{ is a singleton}}{E := E, t \approx u} \text{ Single}$$

- Split on the type of constructor for terms `t`
- Add equalities between all pairs of corecursive singleton terms `t`, `u`

Calculus is a Decision Procedure for $\mathcal{DC}$

- Calculus is:
  - **Terminating**
    - All derivation trees are finite
  - **Refutation-sound**
    - If a closed derivation tree exists, then indeed $\mathbb{E}$ is unsatisfiable
  - **Model-sound**
    - If a saturated node exists, then indeed $\mathbb{E}$ is satisfiable
      - Proof is constructive
- Thus, is a decision procedure

Implementation in SMT Solver

- Calculus is implemented as a DPLL(T) theory solver in CVC4

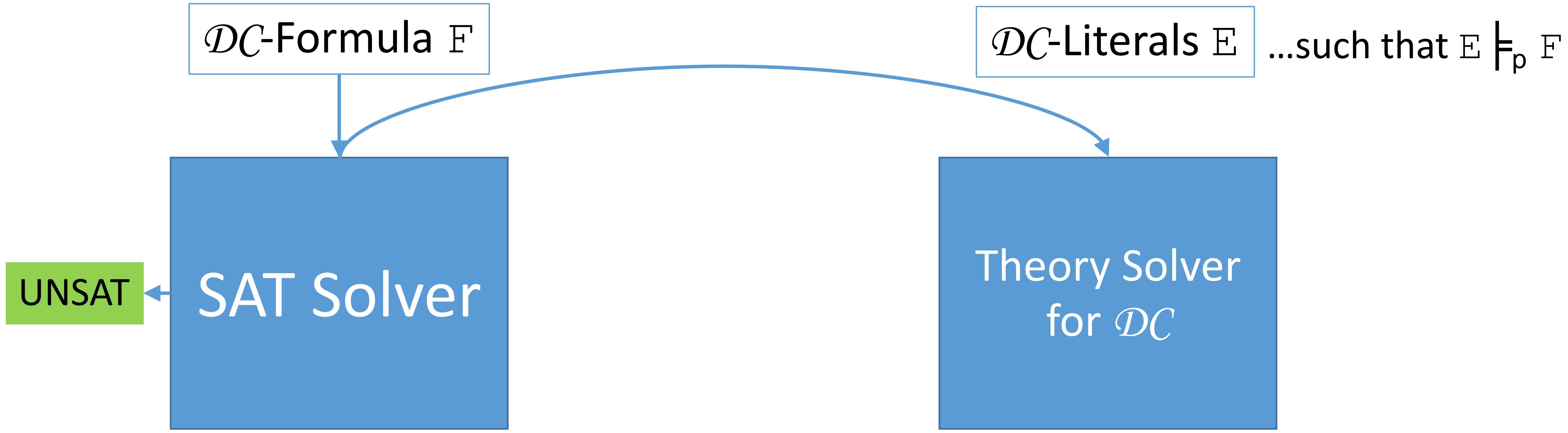
Implementation in SMT Solver

- Calculus is implemented as a DPLL(T) theory solver in CVC4

- SAT solver reasons about $\mathcal{DC}$-formulas at propositional level

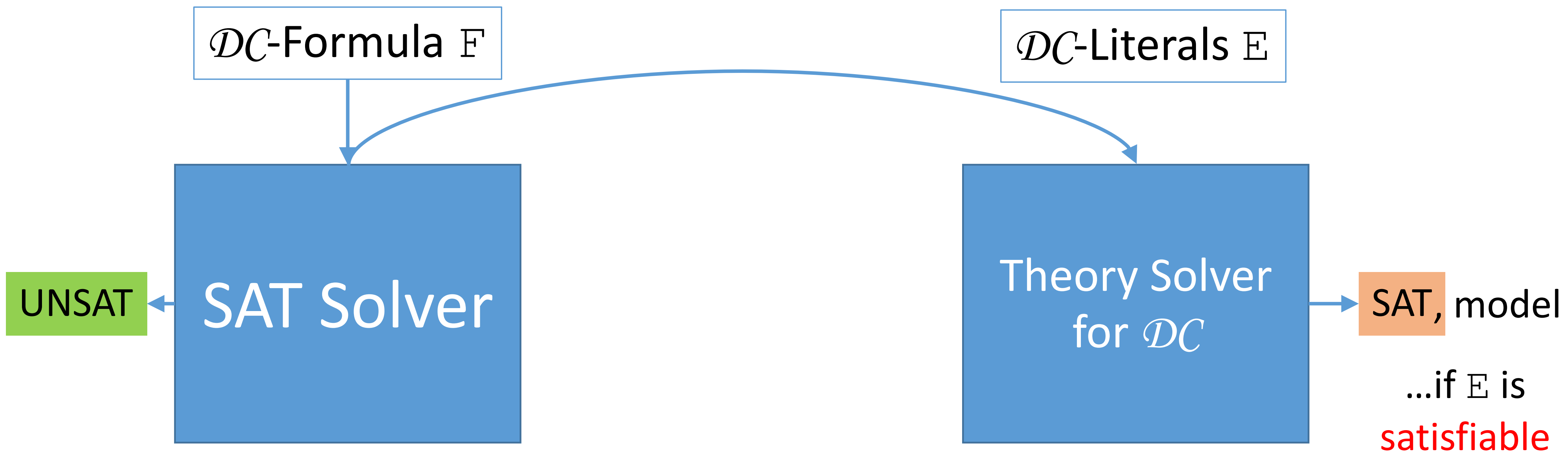
Implementation in SMT Solver

- Calculus is implemented as a DPLL(T) theory solver in CVC4



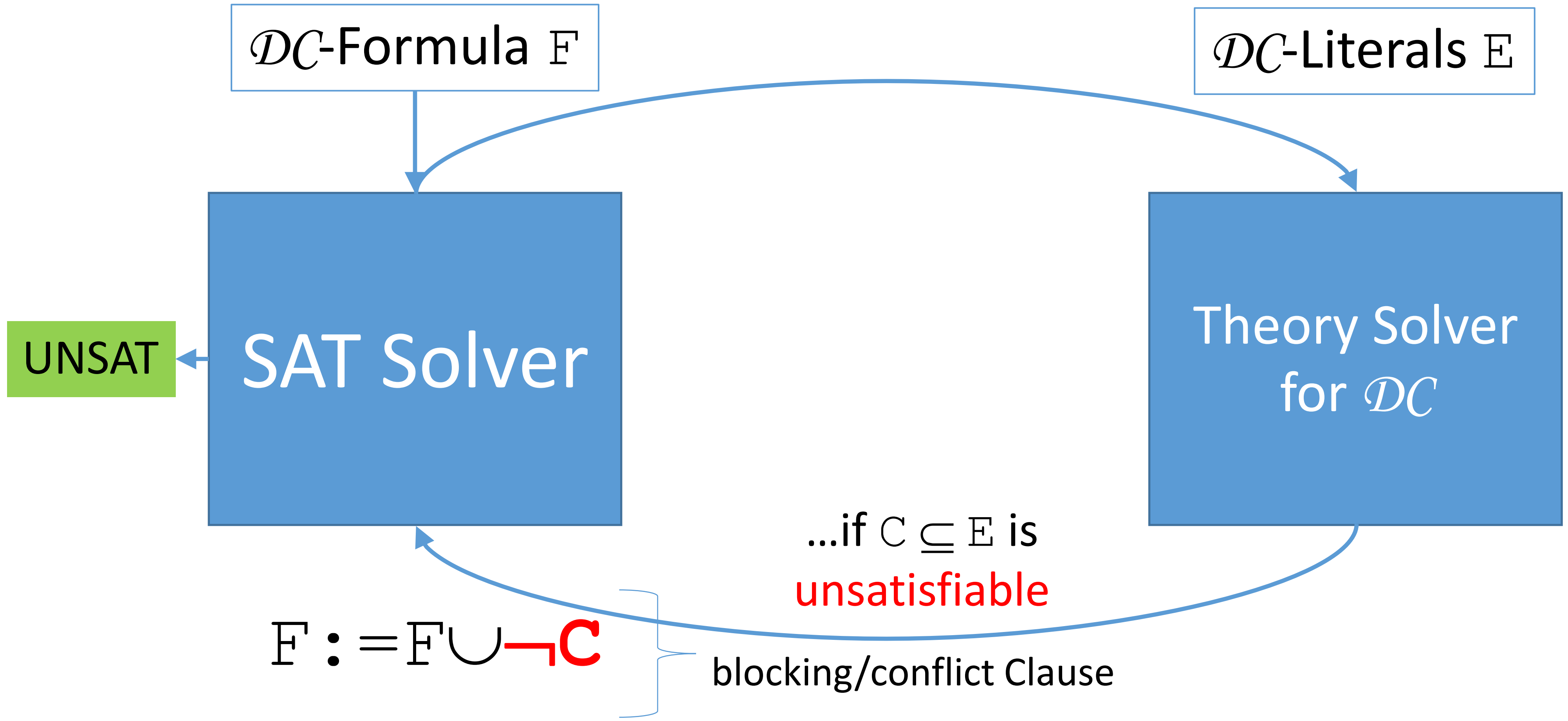
Implementation in SMT Solver

- Calculus is implemented as a DPLL(T) theory solver in CVC4



Implementation in SMT Solver

- Calculus is implemented as a DPLL(T) theory solver in CVC4



Evaluation

- Evaluated SMT solvers
  - **CVC4** : support for (co)datatypes from this talk
  - **Z3** : supports datatypes only
- …on Isabelle benchmarks from three libraries:
  - Isabelle Distribution (**Distro**)
  - Archive of Formal Proofs (**AFP**)
  - Two unpublished theories involving Bird and Stern-Brocot trees (**G&L**)

$\Rightarrow$Benchmarks involve quantified formulas + (co)datatypes

Evaluation

- Encodings:
  - With native (co)datatypes symbols e.g. `tail`, `cons`, `nil`:

$$\exists ab.\,\mathtt{tail}(a)=\mathtt{cons}(0,b)$$

  - Without, axiomatization of uninterpreted symbols, e.g. f_{tail}, f_{cons}, f_{nil}:

$$\forall xyzw.\, f_{cons}(x,y)\neq f_{nil}$$
$$\forall xyzw.\, f_{cons}(x,y)=f_{cons}(z,w)\Rightarrow(x=z\wedge y=w)$$
$$\forall x.\, x=f_{cons}(f_{head}(x),f_{tail}(x))\vee x=f_{nil}$$
$$\forall x.\, f_{tail}(f_{cons}(x,y))=y$$
$$\exists ab.\, f_{tail}(a)=f_{cons}(0,b)$$

Evaluation : Results

		Distro		AFP		G&L		Overall	
		CVC4	Z3	CVC4	Z3	CVC4	Z3	CVC4	Z3
Weaker	No (co)datatypes	221	209	775	777	52	51	1048	1037
	Datatypes without Acyclic	227	–	780	–	52	–	1059	–
	Full datatypes	227	213	786	791	52	51	1065	1055
	Codatatypes without Unique	222	–	804	–	56	–	1082	–
	Full codatatypes	223	–	804	–	**59**	–	1086	–
Stronger	Full (co)datatypes	**229**	–	**815**	–	**59**	–	**1103**	–

- Stronger decision procedures subsume weaker ones
  - Rules for acyclicity, uniqueness contribute to precision of solvers
- Dedicated support for codatatypes in CVC4 improves state of the art
  - CVC4 with full (co)datatypes solves **1103**
    - CVC4 and Z3 with only datatypes solve 1065 and 1055 respectively

Summary

- Decision procedure for theory of (co)datatypes
  - Can be implemented in SMT solvers
- Evaluation on Isabelle benchmarks
  - Beneficial to use stronger decision procedures
- Future work:
  - Reconstruction of (co)datatype proofs in Isabelle
  - Apply to higher-order model finding

    (Our original motivation)

Thanks!

- Paper:
  - 25th Conference on Automated Deduction (CADE-25)
- Implementation:
  - CVC4, available at http://cvc4.cs.nyu.edu/downloads/