A Decision Procedure for (Co)datatypes in SMT Solvers

Andrew Reynolds

Jasmin Christian Blanchette

CADE August 4, 2015

Introductory Examples

Introductory Examples

datatype nat =

Z

| S(nat)

datatype list$_\tau$ =

Nil$_\tau$

| Cons$_\tau$(τ, list$_\tau$)

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

| $\text{LCons}_\tau(\ \tau, \text{llist}_\tau)$

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_τ =

  Nil_τ

| $\text{Cons}_\tau(\tau, \text{list}_\tau)$

Codatatypes need not be well-founded

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

Introductory Examples

$x \neq S(x)$

Introductory Examples

$x \not\approx S(x)$

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

Introductory Examples

Cyclic values exist

$x \neq S(x)$

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

Introductory Examples

Cyclic values exist

$x \not\approx S(x)$

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

$x \approx ES(x)$

$y \approx ES(y)$

Introductory Examples

$x \not\approx S(x)$

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

Cyclic values exist

$x \approx ES(x) \approx ES(ES(ES(\ldots)))$

$y \approx ES(y) \approx ES(ES(ES(\ldots)))$

Introductory Examples

$x \not\approx S(x)$

Cyclic values exist

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

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

Introductory Examples

$x \neq S(x)$

...but they are equal up to their expansion

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

Cyclic values exist

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

Introductory Examples

$x \neq S(x)$

…but they are equal up to their expansion

Cyclic values exist

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

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

μ-notation:

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

$x \approx \mu y.\ ES(\ y\ )$

Introductory Examples

Introductory Examples

xs ≈ LCons(0, LCons(1, LCons(2,…)))

Introductory Examples

xs ≈ LCons(0, LCons(1, LCons(2,…)))

Acyclic infinite values exist too, but they cannot be specified by finite q.f. formulas

Introductory Examples

datatype values =

all finite ground

constructor terms

(and only those)

xs ≈ LCons(0, LCons(1, LCons(2,…)))

Acyclic infinite values
exist too, but they cannot be
specified by finite q.f. formulas

Introductory Examples

datatype values =

all finite ground

constructor terms

(and only those)

xs ≈ LCons(0, LCons(1, LCons(2,…)))

Acyclic infinite values exist too, but they cannot be specified by finite q.f. formulas

codatatype values =

all finite **or infinite**

ground constructor terms

(and only those)

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

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

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

- Specification:

$$
\begin{aligned}
\textbf{(co)datatype } \delta_1 &= \mathsf{C}_{11}([\mathsf{s}_{11}^1{:}]\,\tau_{11}^1, \ldots, [\mathsf{s}_{11}^{n_{11}}{:}]\,\tau_{11}^{n_{11}}) \mid \cdots \mid \mathsf{C}_{1m_1}(\ldots) \\
&\;\vdots \\
\textbf{and } \delta_\ell &= \mathsf{C}_{\ell 1}(\ldots) \mid \cdots \mid \mathsf{C}_{\ell m_\ell}(\ldots)
\end{aligned}
$$

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

- Specification:

$$\begin{aligned}\textbf{(co)datatype}\ \delta_1 &= \mathsf{C}_{11}([\mathsf{s}_{11}^{1}\!:]\ \tau_{11}^{1},\ldots,[\mathsf{s}_{11}^{n_{11}}\!:]\ \tau_{11}^{n_{11}}) \mid \cdots \mid \mathsf{C}_{1m_1}(\ldots)\\ &\ \ \vdots\\ \textbf{and}\ \delta_\ell &= \mathsf{C}_{\ell 1}(\ldots) \mid \cdots \mid \mathsf{C}_{\ell m_\ell}(\ldots)\end{aligned}$$

- Properties:

$$\begin{aligned}\text{Distinctness:}\quad & \mathsf{C}_{ij}(\bar{x}) \not\approx \mathsf{C}_{ij'}(\bar{y}) \quad \text{if } j \neq j'\\ \text{Injectivity:}\quad & \mathsf{C}_{ij}(x_1,\ldots,x_{n_{ij}}) \approx \mathsf{C}_{ij}(y_1,\ldots,y_{n_{ij}}) \longrightarrow x_k \approx y_k\\ \text{Exhaustiveness:}\quad & \mathsf{d}_{i1}(x) \vee \cdots \vee \mathsf{d}_{im_i}(x)\\ \text{Selection:}\quad & \mathsf{s}_{ij}^{k}(\mathsf{C}_{ij}(x_1,\ldots,x_{n_{ij}})) \approx x_k\end{aligned}$$

+ induction resp. coinduction (and more)

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

- Specification:

$$\begin{aligned} \textbf{(co)datatype } \delta_1 &= \mathsf{C}_{11}([\mathsf{s}_{11}^1{:}]\,\tau_{11}^1, \ldots, [\mathsf{s}_{11}^{n_{11}}{:}]\,\tau_{11}^{n_{11}}) \mid \cdots \mid \mathsf{C}_{1m_1}(\ldots) \\ &\vdots \\ \textbf{and } \delta_\ell &= \mathsf{C}_{\ell 1}(\ldots) \mid \cdots \mid \mathsf{C}_{\ell m_\ell}(\ldots) \end{aligned}$$

- Properties:

$$\begin{array}{rl} \text{Distinctness:} & \mathsf{C}_{ij}(\bar{x}) \not\approx \mathsf{C}_{ij'}(\bar{y}) \quad \text{if } j \neq j' \\ \text{Injectivity:} & \mathsf{C}_{ij}(x_1, \ldots, x_{n_{ij}}) \approx \mathsf{C}_{ij}(y_1, \ldots, y_{n_{ij}}) \longrightarrow x_k \approx y_k \\ \text{Exhaustiveness:} & \mathsf{d}_{i1}(x) \vee \cdots \vee \mathsf{d}_{im_i}(x) \\ \text{Selection:} & \mathsf{s}_{ij}^k(\mathsf{C}_{ij}(x_1, \ldots, x_{n_{ij}})) \approx x_k \end{array}$$

+ induction resp. coinduction (and more)

acyclicity resp. uniqueness
are sufficient for q.f. formulas

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?

Counterexamples:

codatatype a = A(a)

codatatype
b = B(b,c,b,unit) and
c = C(a,b,c)

datatype unit = Unity

$\Rightarrow$"Corecursive singletons"

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

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

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

Calculus: Guarded Assignment Form

- Derivation rules of calculus written in guarded assignment form:

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

  - For example:

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

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

$$\frac{\mathtt{t}\approx\mathtt{s},\mathtt{t}\not\approx\mathtt{s}\in\mathtt{E}}{\perp}\ \text{conflict}$$

Calculus: Derivation Tree

$$\cfrac{\cfrac{\texttt{t}\approx\texttt{s},\texttt{s}\approx\texttt{r},\texttt{t}\not\approx\texttt{r}}{\texttt{t}\approx\texttt{s},\texttt{s}\approx\texttt{r},\texttt{t}\not\approx\texttt{r},\texttt{t}\approx\texttt{r}}\ \text{trans}}{\bot}\ \text{conflict}$$

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

Calculus: Overview

- Part 1:
  - Apply bidirectional closure (uniformly to both datatypes and codatatypes)
- Part 2:
  - Ensure all datatype values are acyclic
  - Ensure all codatatype values are unique
- Part 3:
  - Apply splitting/search, if necessary

Part 1: Bidirectional Closure

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

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

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

- $\mathbb{E}$ contains its upwards (congruence) and downwards (unification) closure
- Ensures: $\mathbb{E}$ induces a set of equivalence classes

Part 2: Acyclicity and Uniqueness

- To determine datatype values are acyclic, codatatype values are unique
  - Compute the class of values for each (co)datatype term
    - Assignment map $\mathcal{A}$ map from equivalence classes to μ-terms

  - For example:

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

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

  - Model construction will be based on $\mathcal{A}$

Part 2: Acyclicity and Uniqueness

```
E := {y≈C(x), x≈D(y)}
```

- Given input **E**

Part 2: Acyclicity and Uniqueness

```
E := {y≈C(x), x≈D(y), y≈y, x≈x, …}
```

- Given input `E`
- Compute bidirectional closure (as in step 1)

Part 2: Acyclicity and Uniqueness

```
E := {y≈C(x), x≈D(y), y≈y, x≈x, …}
```

$$\underbrace{\texttt{\{y,C(x)\},}}_{\texttt{[y]}}\ \underbrace{\texttt{\{x,D(y)\}}}_{\texttt{[x]}}$$

- Given input `E`
- Compute bidirectional closure, equivalence classes induced by `E`

Part 2: Acyclicity and Uniqueness

$\mathrm{E} := \{y \approx C(x),\ x \approx D(y),\ y \approx y,\ x \approx x,\ \ldots\}$

$\{y, C(x)\},\ \{x, D(y)\}$

$\mathcal{A} :=$

$[y] \rightarrow \tilde{y}$

$[x] \rightarrow \tilde{x}$

- Given input E
- Compute bidirectional closure, equivalence classes induced by E
- Compute assignment map $\mathcal{A}$
  - Initially unconstrained

Part 2: Acyclicity and Uniqueness

$$\mathtt{E} := \{\mathtt{y} \approx \mathtt{C(x)},\ \mathtt{x} \approx \mathtt{D(y)},\ \mathtt{y} \approx \mathtt{y},\ \mathtt{x} \approx \mathtt{x},\ \ldots\}$$

$$\{\mathtt{y}, \mathbf{C(x)}\},\ \{\mathtt{x}, \mathtt{D(y)}\}$$

$$\mathcal{A} :=$$
$$[\mathtt{y}] \rightarrow \mu\tilde{\mathtt{y}}.\mathtt{C}(\tilde{\mathtt{x}})$$
$$[\mathtt{x}] \rightarrow \tilde{\mathtt{x}}$$

$$\{\tilde{\mathtt{y}} \rightarrow \mu\tilde{\mathtt{y}}.\mathbf{C}(\tilde{\mathbf{x}})\}$$

- Given input E
- Compute bidirectional closure, equivalence classes induced by E
- Compute assignment map $\mathcal{A}$
  - Initially unconstrained, updated based on constructor terms

Part 2: Acyclicity and Uniqueness

```
E := {y≈C(x), x≈D(y), y≈y, x≈x, …}
```

```
{y,C(x)}, {x,D(y)}
```

```
𝒜 :=
[y] → μỹ.C( μx̃.D(ỹ))
[x] → μx̃.D(ỹ)
```

```
{ x̃ → μx̃.D(ỹ) }
```

- Given input E
- Compute bidirectional closure, equivalence classes induced by E
- Compute assignment map $\mathcal{A}$
  - Initially unconstrained, updated based on constructor terms

Part 2: Acyclicity and Uniqueness

`E := {y≈C(x), x≈D(y), y≈y, x≈x, …}`

`{y,`**`C(x)`**`}, {x,D(y)}`

$\mathcal{A}$ `:=`

`[y]` $\rightarrow$ $\mu\tilde{y}.C(\mu\tilde{x}.D(\tilde{y}))$

`[x]` $\rightarrow$ $\mu\tilde{x}.D(\mu\tilde{y}.C(\tilde{x}))$

$\{\tilde{y} \rightarrow \mu\tilde{y}.\mathbf{C(\tilde{x})}\}$

- Given input `E`
- Compute bidirectional closure, equivalence classes induced by `E`
- Compute assignment map $\mathcal{A}$
  - Initially unconstrained, updated based on constructor terms

Part 2: Acyclicity and Uniqueness

$$\mathtt{E} := \{y \approx C(x),\ x \approx D(y),\ y \approx y,\ x \approx x,\ \ldots\}$$

$$\{y, C(x)\},\ \{x, D(y)\}$$

$$\mathcal{A} :=$$

$$[y] \rightarrow \mu\tilde{y}.C(\mu\tilde{x}.D(\tilde{y}))$$

$$[x] \rightarrow \mu\tilde{x}.D(\mu\tilde{y}.C(\tilde{x}))$$

$$\{\tilde{y} \rightarrow \mu\tilde{y}.C(\tilde{x})\}$$

- Given input $\mathtt{E}$
- Compute bidirectional closure, equivalence classes induced by $\mathtt{E}$
- Compute assignment map $\mathcal{A}$
  - Initially unconstrained, updated based on constructor terms, until no free variables

Part 2: Acyclicity and Uniqueness

$$E \; := \; \{y{\approx}C(x),\; x{\approx}D(y),\; y{\approx}y,\; x{\approx}x,\; \ldots\}$$

$$\{y, C(x)\},\; \{x, D(y)\}$$

$$\mathcal{A} \; :=$$
$$[y] \;\rightarrow\; \mu\tilde{y}.C(\,\mu\tilde{x}.D(\tilde{y}))$$
$$[x] \;\rightarrow\; \mu\tilde{x}.D(\,\mu\tilde{y}.C(\tilde{x}))$$

- This indicates:
  - The value of y in all models is of the form $C(D(C(D(C(\ldots)\ldots))))$
  - The value of x in all models is of the form $D(C(D(C(D(\ldots)\ldots))))$

Part 2: Acyclicity and Uniqueness

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

- Rule Acyclic:
  - Checks whether a $\mathcal{A}$[`t`] contains a bound variable for some datatype term `t`
  - For example: `E={x≈C(x)}`
    - $\mathcal{A}[\mathtt{x}] = \mu\tilde{\mathtt{x}}.\mathtt{C}(\tilde{\mathtt{x}})$, thus $\mathtt{E} \models_{\mathcal{DC}} \bot$
- 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 $\mathtt{E} \models_{\mathcal{DC}} \mathtt{x} \approx \mathtt{y}$

Part 3: Splitting

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

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

- Split on the type of constructor for terms `t` such that either:
  - There exists a selector applied to `t`, or
  - `t` is of finite type
- 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 $\mathtt{E}$ is $\mathcal{DC}$-unsatisfiable
  - **Model-sound**
    - If a saturated node exists, then indeed $\mathtt{E}$ is $\mathcal{DC}$-satisfiable
      - Proof is constructive
- Thus, is a decision procedure for $\mathcal{DC}$

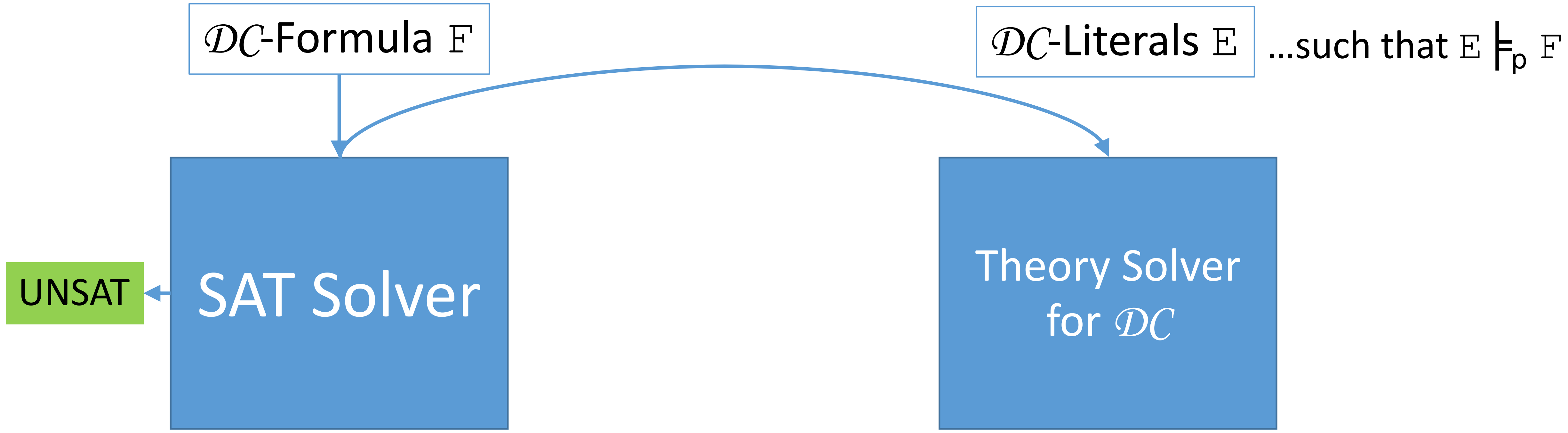
Implementation in SMT Solver

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

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

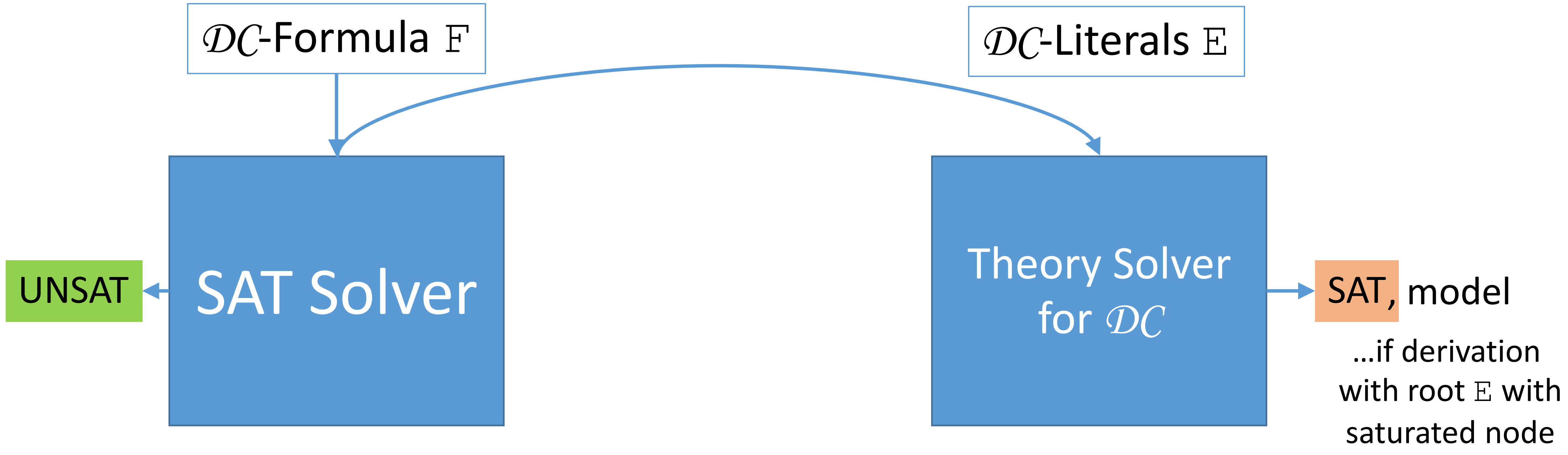
Implementation in SMT Solver

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



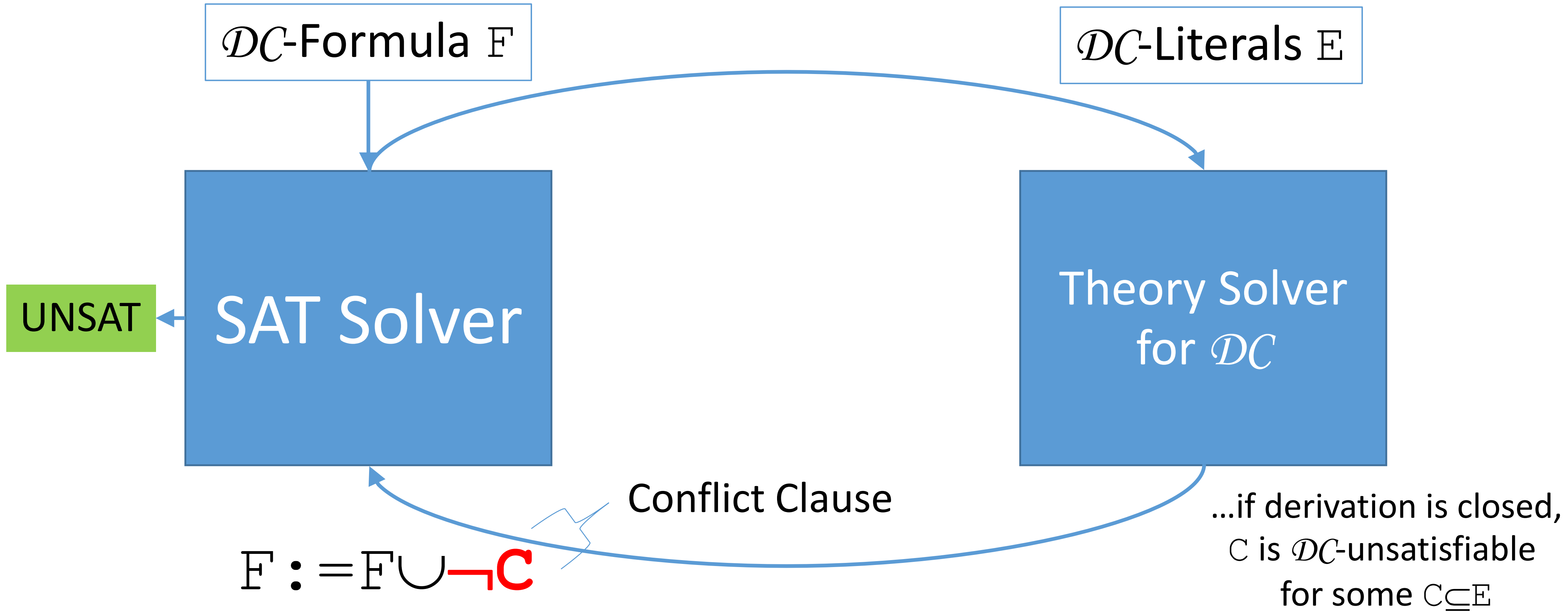
Implementation in SMT Solver

- 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 : Results

Weaker

Stronger

	Distro		AFP		G&L		Overall	
	CVC4	Z3	CVC4	Z3	CVC4	Z3	CVC4	Z3
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	–
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
  - Proved correct
  - Can be implemented in SMT solvers
- Evaluation on Isabelle benchmarks
  - Beneficial to use stronger decision procedures

Future Work

- Reconstruction proofs in Isabelle
- Apply to higher-order model finding
  (Our original motivation)