Datatypes with Shared Selectors

Andrew Reynolds[1], Arjun Viswanathan[1], Haniel Barbosa[1], Cesare Tinelli[1] and Clark Barrett[2]

[1]University of Iowa, Iowa City, U.S.A.

[2]Department of Computer Science, Stanford University

IJCAR 2018

2018–07–15, Oxford, UK

Introductory example

$$\mathbf{Tree} = \mathrm{N}_1(\mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{N}_2(\mathbf{Int}, \mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{L}(\mathbf{Bool}, \mathbf{Int})$$

- ▷ Subfields are accessed with *selectors*, which are associated with *each* constructor, e.g.

$$\mathrm{S}^{\mathrm{N}_1,1} : \mathbf{Tree} \to \mathbf{Int}$$
$$\mathrm{S}^{\mathrm{N}_1,2} : \mathbf{Tree} \to \mathbf{Tree}$$
$$\mathrm{S}^{\mathrm{N}_1,3} : \mathbf{Tree} \to \mathbf{Tree}$$

- ▷ Each constructor is associated with a *tester* predicate, i.e.

$$\mathrm{isN}_1,\ \mathrm{isN}_2,\ \mathrm{isL}$$

- ▷ Given a term t of type $\mathbf{Tree}$ the following clause set states

$$\{\, \neg\mathrm{isN}_1(t) \vee \mathrm{S}^{\mathrm{N}_1,1}(t) \geq 0,\ \neg\mathrm{isL}(t) \vee \mathrm{S}^{\mathrm{L},2}(t) \geq 0 \,\}$$

  - ▶ when t has top symbol N_1, its first subfield is non-negative
  - ▶ when t has top symbol L, its second subfield is non-negative

Why share selectors?

$\mathbf{Tree} = \mathrm{N}_1(\mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{N}_2(\mathbf{Int}, \mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{L}(\mathbf{Bool}, \mathbf{Int})$

- ▷ Consider a different kind of selector symbol

$$\mathrm{S}^{\mathbf{Int},1} : \mathbf{Tree} \rightarrow \mathbf{Int}$$

  which maps each value of type $\mathbf{Tree}$ to its *first* subfield of type $\mathbf{Int}$

- ▷ Mapping is *independent* of the term's top constructor

Why share selectors?

$\mathbf{Tree} = \mathrm{N}_1(\mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{N}_2(\mathbf{Int}, \mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{L}(\mathbf{Bool}, \mathbf{Int})$

- ▷ Consider a different kind of selector symbol

$$\mathrm{S}^{\mathbf{Int},1} : \mathbf{Tree} \to \mathbf{Int}$$

  which maps each value of type $\mathbf{Tree}$ to its *first* subfield of type $\mathbf{Int}$

- ▷ Mapping is *independent* of the term's top constructor

- ▷ The previous clause set can be written using a single *shared* selector

$$\{\, \neg\mathrm{isN}_1(t) \vee \mathrm{S}^{\mathbf{Int},1}(t) \geq 0,\ \neg\mathrm{isL}(t) \vee \mathrm{S}^{\mathbf{Int},1}(t) \geq 0 \,\}$$

- ▷ Note that the arithmetic literal is now the same in both clauses

- ▷ The $\mathbf{Tree}$ datatype requires only *five* shared selectors instead of *nine* standard selectors

Outline

- ▷ Theory of Datatypes with Shared Selectors
- ▷ Application: Syntax-Guided Synthesis (SyGuS)
  - ▶ Overview of the SyGuS problem
  - ▶ Using Shared Selectors for Syntax-Guided Synthesis
- ▷ Evaluation
  - ▶ SyGuS
  - ▶ SMT-LIB

Theory of Datatypes with Shared Selectors

Theory of Datatypes

- Specification

$$\textbf{datatype}\ \delta = \mathrm{C}_1([\mathrm{S}_{\boldsymbol{\delta}}^{\mathrm{C}_1,1}] : \tau_1,\ \ldots,\ [\mathrm{S}_{\boldsymbol{\delta}}^{\mathrm{C}_1,n_1}] : \tau_{n_1}) \mid \ldots \mid \mathrm{C}_\mathrm{m}(\ldots)$$

s.t. $\mathrm{S}_{\boldsymbol{\delta}}^{\mathrm{C},k} : \delta \to \tau_k$

- Besides basic properties of *Distinctness*, *Injectivity*, *Exhaustiveness*, and *Acyclicity*, datatypes also respect

$$\forall x_1, \ldots, x_n.\, \mathrm{S}_{\boldsymbol{\delta}}^{\mathrm{C},k}(\mathrm{C}(x_1,\ \ldots,\ x_n)) \approx x_k \qquad (\textit{Standard selection})$$

Theory of Datatypes with Shared Selectors ($\mathcal{D}$)

- Extend the signature with *shared selectors* $\mathrm{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k}$ for each datatype δ and type τ in $\mathcal{D}$ and each natural number k
- $\mathrm{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k}$ when applied to a δ-term $\mathrm{C}(t_1, \ldots, t_n)$ returns the k-th argument of C that has type τ, if one exists
- Formally represented with a partial function stoa, e.g. for

$$\mathbf{Tree} = \mathrm{N}_1(\mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{N}_2(\mathbf{Int}, \mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{L}(\mathbf{Bool}, \mathbf{Int})$$

  - $\mathrm{stoa}(1, \mathbf{Int}, \mathrm{N}_1) = 1$, $\mathrm{stoa}(2, \mathbf{Tree}, \mathrm{N}_1) = 3$
  - $\mathrm{stoa}(2, \mathbf{Int}, \mathrm{N}_1)$, $\mathrm{stoa}(1, \mathbf{Bool}, \mathrm{N}_2)$ are undefined.

- Datatypes in $\mathcal{D}$ also respect the property

$$\forall x_1, \ldots, x_n.\, \mathrm{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k}(\mathrm{C}(x_1, \ldots, x_n)) \approx x_i, \text{ where } i = \mathrm{stoa}(k, \boldsymbol{\tau}, \mathrm{C})$$

From standard selectors to shared selectors

- We reduce arbitrary constraints to constraints with only shared selectors
- Thus our calculus only needs to account for shared selectors
- We prove that the resulting reduction is equisatisfiable to the original constraints
- Reduction can be applied as a preprocessing step in an implementation of $\mathcal{D}$

Calculus for Theory of Datatypes with Shared Selectors $\mathcal{D}$

- Similar to previous calculi from [Barrett et al. 2007, Reynolds and Blanchette 2015]
- Tableau-like calculus to decide the $\mathcal{D}$-satisfiability of a set of quantifier-free constraints E
- Our main modification is in the $\textsc{Split}$ rule, which unrolls terms by branching on different constructors
- Instead of introducing standard selectors, the $\textsc{Split}$ rule introduces shared selectors

Calculus for Theory of Datatypes with Shared Selectors $\mathcal{D}$

The Split rule:

$$\frac{\mathrm{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},n}(t) \in \mathbf{T}(E) \ \text{ or } \delta \text{ is finite}}{\begin{array}{lll} E & := & E,\, t \approx \mathrm{C}_1(\mathrm{S}_{\boldsymbol{\delta}}^{\tau_{1,1},\mathrm{atos}(\boldsymbol{\tau_{1,1}},\,\mathrm{C}_1,\,1)}(t),\, \ldots,\, \mathrm{S}_{\boldsymbol{\delta}}^{\tau_{1,\mathrm{n}_1},\mathrm{atos}(\boldsymbol{\tau_{1,n_1}},\,\mathrm{C}_1,\,n_1)}(t)) \\ & \vdots & \\ E & := & E,\, t \approx \mathrm{C}_\mathrm{m}(\mathrm{S}_{\boldsymbol{\delta}}^{\tau_{\mathrm{m},1}1,\mathrm{atos}(\boldsymbol{\tau_{m,1}},\,\mathrm{C}_\mathrm{m},\,1)}(t),\, \ldots,\, \mathrm{S}_{\boldsymbol{\delta}}^{\tau_{\mathrm{m},\mathrm{n_m}},\mathrm{atos}(\boldsymbol{\tau_{m,n_m}},\,\mathrm{C}_\mathrm{m},\,n_m)}(t)) \end{array}}$$

- Consider again the datatype
  $\mathbf{Tree} = \mathrm{N}_1(\mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{N}_2(\mathbf{Int}, \mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathrm{L}(\mathbf{Bool}, \mathbf{Int})$
- For a term $\mathrm{S}^{\mathbf{Tree},1}(t)$, the split would introduce a branch with

$$\begin{aligned} E := E, \quad t &\approx \mathrm{N}_1(\mathrm{S}^{\mathbf{Int},\mathrm{atos}(\mathbf{Int},\,\mathrm{N}_1,\,1)}(t),\, \mathrm{S}^{\mathbf{Tree},\mathrm{atos}(\mathbf{Tree},\,\mathrm{N}_1,\,2)}(t),\, \mathrm{S}^{\mathbf{Tree},\mathrm{atos}(\mathbf{Tree},\,\mathrm{N}_1,\,3)}(t)) \\ &\approx \mathrm{N}_1(\mathrm{S}^{\mathbf{Int},1}(t),\, \mathrm{S}^{\mathbf{Tree},1}(t),\, \mathrm{S}^{\mathbf{Tree},2}(t)) \end{aligned}$$

Calculus is a decision procedure for $\mathcal{D}$

Calculus is

- ▷ Terminating
  - ▶ All derivation trees are finite
- ▷ Refutation sound
  - ▶ If a closed derivation tree exists, then indeed E is $\mathcal{D}$-unsatisfiable
- ▷ Solution sound
  - ▶ If a saturated node exists, then indeed E is $\mathcal{D}$-satisfiable
  - ▶ Proof is constructive

Thus the calculus is a decision procedure for $\mathcal{D}$

Application: Syntax-Guided Synthesis (SyGuS)

Problem statement

- Synthesizing a function that satisfies a given specification, while considering explicit syntactic restrictions on the solution space
  - specification is given by a (second-order) T-formula of the form $\exists f. \forall \bar{x}. \varphi[f, \bar{x}]$
  - syntactic restrictions on the solutions for f given by a grammar R
- A solution for f is a lambda term $\lambda \bar{y}. e$ of the same type as f s.t. $\forall \bar{x}. \varphi[\lambda \bar{y}. e, \bar{x}]$ is valid in T and e is in the language generated by R

Problem statement

- ▷ Synthesizing a function that satisfies a given specification, while considering explicit syntactic restrictions on the solution space
  - ▶ specification is given by a (second-order) T-formula of the form $\exists f. \forall \bar{x}. \varphi[f, \bar{x}]$
  - ▶ syntactic restrictions on the solutions for f given by a grammar R
- ▷ A solution for f is a lambda term $\lambda \bar{y}. e$ of the same type as f s.t. $\forall \bar{x}. \varphi[\lambda \bar{y}. e, \bar{x}]$ is valid in T and e is in the language generated by R

To synthesize e.g. a commutative binary function f over integers, i.e. solve

$$\exists f \, \forall xy. \, f(x, y) \approx f(y, x)$$

such that the solution space of f is defined by the grammar

$A \to x \mid y \mid 0 \mid 1 \mid A{+}A \mid A{-}A \mid \mathrm{ite}(B,\, A,\, A)$ $\qquad B \to A \geq A \mid A \approx A \mid \neg B$

Problem statement

- ▷ Synthesizing a function that satisfies a given specification, while considering explicit syntactic restrictions on the solution space
  - ▶ specification is given by a (second-order) T-formula of the form $\exists f.\, \forall \bar{x}.\, \varphi[f, \bar{x}]$
  - ▶ syntactic restrictions on the solutions for f given by a grammar R
- ▷ A solution for f is a lambda term $\lambda \bar{y}.\, e$ of the same type as f s.t. $\forall \bar{x}.\, \varphi[\lambda \bar{y}.\, e, \bar{x}]$ is valid in T and e is in the language generated by R

To synthesize e.g. a commutative binary function f over integers, i.e. solve

$$\exists f\, \forall xy.\, f(x, y) \approx f(y, x)$$

such that the solution space of f is defined by the grammar

$$A \to x \mid y \mid 0 \mid 1 \mid A{+}A \mid A{-}A \mid \mathrm{ite}(B,\, A,\, A) \qquad\qquad B \to A \geq A \mid A \approx A \mid \neg B$$

A solution is e.g. $f = \lambda xy.\, 0$ or $f = \lambda xy.\, x + y$

Enumerative SyGuS in SMT

[Reynolds et al. FMSD'17]

- Encode problem using a deep embedding into datatypes

$$\mathbf{a} = \mathrm{X} \mid \mathrm{Y} \mid \mathrm{Zero} \mid \mathrm{One} \mid \mathrm{Plus}(\mathbf{a}, \mathbf{a}) \mid \mathrm{Minus}(\mathbf{a}, \mathbf{a}) \mid \mathrm{Ite}(\mathbf{b},\ \mathbf{a},\ \mathbf{a})$$
$$\mathbf{b} = \mathrm{Geq}(\mathbf{a},\ \mathbf{a}) \mid \mathrm{Eq}(\mathbf{a},\ \mathbf{a}) \mid \mathrm{Neg}(\mathbf{b})$$

  represent the grammar R and the specification becomes

$$\forall xy.\, \mathrm{eval}_{\mathbf{a}}(d, x, y) \approx \mathrm{eval}_{\mathbf{a}}(d, y, x)$$

  where d is a fresh constant of type $\mathbf{a}$.

- eval maps datatype terms to their corresponding theory terms
  - $\mathrm{eval}_{\mathbf{a}}(\mathrm{Plus}(\mathrm{X}, \mathrm{X}), 2, 3)$ is interpreted as $(x + x)\{x \mapsto 2, y \mapsto 3\} = 4$

Enumerative SyGuS in SMT [Reynolds et al. FMSD'17]

- Encode problem using a deep embedding into datatypes

$$\mathbf{a} = \mathrm{X} \mid \mathrm{Y} \mid \mathrm{Zero} \mid \mathrm{One} \mid \mathrm{Plus}(\mathbf{a}, \mathbf{a}) \mid \mathrm{Minus}(\mathbf{a}, \mathbf{a}) \mid \mathrm{Ite}(\mathbf{b},\ \mathbf{a},\ \mathbf{a})$$
$$\mathbf{b} = \mathrm{Geq}(\mathbf{a},\ \mathbf{a}) \mid \mathrm{Eq}(\mathbf{a},\ \mathbf{a}) \mid \mathrm{Neg}(\mathbf{b})$$

  represent the grammar R and the specification becomes

$$\forall xy.\, \mathrm{eval}_{\mathbf{a}}(d, x, y) \approx \mathrm{eval}_{\mathbf{a}}(d, y, x)$$

  where d is a fresh constant of type $\mathbf{a}$.

- eval maps datatype terms to their corresponding theory terms
  - $\mathrm{eval}_{\mathbf{a}}(\mathrm{Plus}(\mathrm{X}, \mathrm{X}), 2, 3)$ is interpreted as $(x + x)\{x \mapsto 2, y \mapsto 3\} = 4$

- Solutions are models in which d is interpreted is interpreted e.g. as Zero or $\mathrm{Plus}(\mathrm{X}, \mathrm{Y})$, corresponding to $f = \lambda xy.\, 0$ and $f = \lambda xy.\, x + y$

Pruning the search space: symmetry breaking

- Given the explosive nature of enumeration, reducing the number of candidate terms is key
- Only consider terms whose theory interpretation is unique up to theory-specific simplification!
  - Since x and $x + 0$ are equivalent, ignore one of them

Pruning the search space: symmetry breaking

- ▷ Given the explosive nature of enumeration, reducing the number of candidate terms is key
- ▷ Only consider terms whose theory interpretation is unique up to theory-specific simplification!
  - ▶ Since x and $x + 0$ are equivalent, ignore one of them
- ▷ Symmetry breaking clauses

$$\neg\text{isPlus}(z) \vee \neg\text{isX}(\text{S}^{\text{Int},1}(z)) \vee \neg\text{isZero}(\text{S}^{\text{Int},2}(z))$$

which can be read as "do not consider solutions s.t. z is $x + 0$"

Pruning the search space: symmetry breaking

By instantiating z with selector chains we can rule out *entire families* of redundant candidates, e.g.

$$\neg \text{isPlus}(\text{S}^{\text{Int},1}(d)) \vee \neg \text{isX}(\text{S}^{\text{Int},1}(\text{S}^{\text{Int},1}(d))) \vee \neg \text{isZero}(\text{S}^{\text{Int},2}(\text{S}^{\text{Int},1}(d)))$$

rules out terms that have $x + 0$ as their first child of type $\mathbf{a}$, such as

$$(x + 0) + y \equiv x + y$$
$$\text{ite}(x \geq y,\, x + 0,\, y) \equiv \text{ite}(x \geq y,\, x,\, y)$$
$$(x + 0) - 1 \equiv x - 1$$

Pruning the search space: symmetry breaking

By instantiating z with selector chains we can rule out *entire families* of redundant candidates, e.g.

$$\neg \text{isPlus}(\text{S}^{\text{Int},1}(d)) \vee \neg \text{isX}(\text{S}^{\text{Int},1}(\text{S}^{\text{Int},1}(d))) \vee \neg \text{isZero}(\text{S}^{\text{Int},2}(\text{S}^{\text{Int},1}(d)))$$

rules out terms that have $x + 0$ as their first child of type $\mathbf{a}$, such as

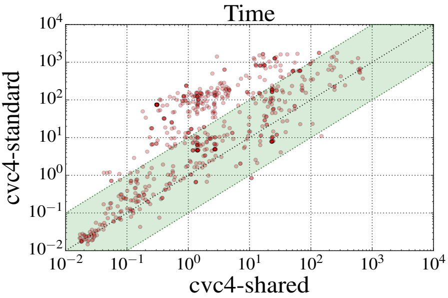
$$(x + 0) + y \equiv x + y$$
$$\text{ite}(x \geq y,\, x + 0,\, y) \equiv \text{ite}(x \geq y,\, x,\, y)$$
$$(x + 0) - 1 \equiv x - 1$$

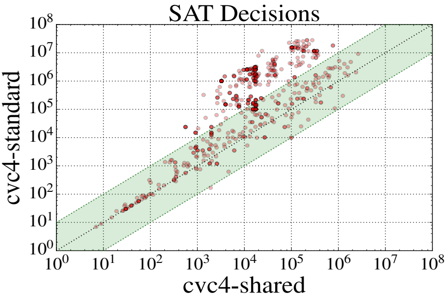
- ▷ Sharing selectors allows the same blocking clause to be reused for the different constructors
- ▷ standard selectors would require three different clauses in this case

$$\neg \text{isPlus}(\text{S}^{\text{Plus},1}(d)) \vee \neg \text{isX}(\text{S}^{\text{Plus},1}(\text{S}^{\text{Plus},1}(d))) \vee \neg \text{isZero}(\text{S}^{\text{Plus},1}(\text{S}^{\text{Plus},2}(d)))$$
$$\neg \text{isPlus}(\text{S}^{\text{Ite},2}(d)) \vee \neg \text{isX}(\text{S}^{\text{Ite},2}(\text{S}^{\text{Plus},1}(d))) \vee \neg \text{isZero}(\text{S}^{\text{Ite},2}(\text{S}^{\text{Plus},2}(d)))$$
$$\neg \text{isPlus}(\text{S}^{\text{Minus},1}(d)) \vee \neg \text{isX}(\text{S}^{\text{Minus},1}(\text{S}^{\text{Plus},1}(d))) \vee \neg \text{isZero}(\text{S}^{\text{Minus},1}(\text{S}^{\text{Plus},2}(d)))$$

Evaluation

Impact on SyGuS-COMP 2017 benchmarks





Family	#	Solved sh	Solved std	Solved (both)	Terms sh	Terms std	Sels sh	Sels std
General	535	319	235	(232)	189k	284k	5.8	16.8
CLIA	73	18	17	(17)	25k	60k	9.6	22.2
Invariant	67	46	46	(46)	37k	61k	5.7	13.1
PBE_BV	750	665	253	(253)	14k	202k	3.0	16.0
PBE_Strings	108	93	64	(64)	14k	41k	8.6	18.7

- Over 80% reduction in average number of selectors for PBE_BV
- PBE_Strings, General also show significant improvements

Comparison with other SygGuS solvers

Family	#	EUSOLVER	CVC4-si-sh	CVC4-si-std
General	535	**404**	391	334
CLIA	73	71	**73**	73
Invariant	67	42	**46**	46
PBE_BV	750	**739**	665	253
PBE_Strings	108	68	**93**	64

- ▷ Comparison also includes CVC4's single-invocation approach (impacts General and CLIA)
- ▷ CVC4 is only competitive on General, PBE_Strinsg and, specially, in PBE_BV due to shared selectors
- ▷ Further improvements with other techniques in the past months now have CVC4 leading EUSolver in all families in SyGuS-COMP 2018

Evaluation on SMT-LIB benchmarks

Family	#	Solved			Time		Decs		Terms		Sels	
		sh	std	(both)	sh	std	sh	std	sh	std	sh	std
Leon	410	179	175	(175)	0.96	0.75	9.9k	9.9k	718	929	8.67	23.10
Sledgehammer	321	113	112	(112)	0.47	0.47	6.9k	6.9k	185	185	10.50	12.76
Nunchaku	158	67	67	(67)	0.49	0.44	7.1k	6.6k	1373	1297	6.22	7.22

- Leon benchmarks show the most impact of sharing selectors
  - Reduction of over 60% in the average number of selectors
  - 4 more problems solved

- Overall SMT-LIB benchmarks are not significantly impacted

Conclusions

- We have presented an extension to theory of algebraic datatypes that adds shared selectors
- Introduced a correct decision procedure for the new theory
- Shared selectors can lead to significant gains in SyGuS solving
  - A main reason for CVC4 becoming the best known solver is certain classes of SyGuS problems
- Possible future work is to generalize our approach for selector *chains*
  - Compressing chain of applications to a single one
  - Requires more sophisticated criteria for transformation
  - We expect that such an extension can lead to performance improvements as well