Datatypes with Shared Selectors

Andrew Reynolds\textsuperscript{1}, Arjun Viswanathan\textsuperscript{1}, Haniel Barbosa\textsuperscript{1}, Cesare Tinelli\textsuperscript{1} and Clark Barrett\textsuperscript{2}

\textsuperscript{1}University of Iowa, Iowa City, U.S.A.
\textsuperscript{2}Department of Computer Science, Stanford University

IJCAR 2018
Introductory example

\[ \text{Tree} = N_1(\text{Int}, \text{Tree}, \text{Tree}) | N_2(\text{Int}, \text{Int}, \text{Tree}, \text{Tree}) | L(\text{Bool}, \text{Int}) \]

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

\[
\begin{align*}
S^{N_1,1} : & \text{Tree} \rightarrow \text{Int} \\
S^{N_1,2} : & \text{Tree} \rightarrow \text{Tree} \\
S^{N_1,3} : & \text{Tree} \rightarrow \text{Tree}
\end{align*}
\]

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

\[
is_{N_1}, \ is_{N_2}, \ is_{L}
\]

▷ Given a term \( t \) of type \text{Tree} the following clause set states

\[
\{ \neg is_{N_1}(t) \lor S^{N_1,1}(t) \geq 0, \neg is_{L}(t) \lor S^{L,2}(t) \geq 0 \}
\]

▶ 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?

Tree = \( N_1(\text{Int}, \text{Tree}, \text{Tree}) \mid N_2(\text{Int}, \text{Int}, \text{Tree}, \text{Tree}) \mid L(\text{Bool}, \text{Int}) \)

- Consider a different kind of selector symbol
  \( S^{\text{Int}, 1} : \text{Tree} \rightarrow \text{Int} \)
  which maps each value of type \( \text{Tree} \) to its \textit{first} subfield of type \( \text{Int} \)

- Mapping is \textit{independent} of the term’s top constructor
Why share selectors?

\[
\text{Tree} = N_1(\text{Int}, \text{Tree}, \text{Tree}) \mid N_2(\text{Int}, \text{Int}, \text{Tree}, \text{Tree}) \mid L(\text{Bool}, \text{Int})
\]

- Consider a different kind of selector symbol
  \[
  S^{\text{Int}, 1} : \text{Tree} \rightarrow \text{Int}
  \]
  which maps each value of type \text{Tree} to its \text{first} subfield of type \text{Int}

- Mapping is \text{independent} of the term’s top constructor

- The previous clause set can be written using a single \text{shared} selector
  \[
  \{ \neg \text{isN}_1(t) \lor S^{\text{Int}, 1}(t) \geq 0, \neg \text{isL}(t) \lor S^{\text{Int}, 1}(t) \geq 0 \}
  \]

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

- The \text{Tree} datatype requires only five \text{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

```
datatype δ = C_1([S^{C_1,1}_δ : τ_1, \ldots, [S^{C_1,n_1}_δ : τ_{n_1}]) | \ldots | C_m(\ldots)
```

s.t. \( S^{C,k}_δ : δ \rightarrow τ_k \)

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

\[
\forall x_1, \ldots, x_n. S^{C,k}_δ(C(x_1, \ldots, x_n)) \approx x_k \quad (Standard \ selection)
\]
Theory of Datatypes with Shared Selectors ($\mathcal{D}$)

- Extend the signature with *shared selectors* $S^{\tau,k}_\delta$ for each datatype $\delta$ and type $\tau$ in $\mathcal{D}$ and each natural number $k$.

- $S^{\tau,k}_\delta$ when applied to a $\delta$-term $C(t_1,\ldots,t_n)$ returns the $k$-th argument of $C$ that has type $\tau$, if one exists.

- Formally represented with a partial function $\text{stoa}$, e.g. for

  \[
  \text{Tree} = N_1(\text{Int}, \text{Tree}, \text{Tree}) \mid N_2(\text{Int}, \text{Int}, \text{Tree}, \text{Tree}) \mid L(\text{Bool}, \text{Int})
  \]

  - $\text{stoa}(1, \text{Int}, N_1) = 1$, $\text{stoa}(2, \text{Tree}, N_1) = 3$
  - $\text{stoa}(2, \text{Int}, N_1)$, $\text{stoa}(1, \text{Bool}, N_2)$ are undefined.

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

  $$\forall x_1, \ldots, x_n. \ S^{\tau,k}_\delta(C(x_1, \ldots, x_n)) \approx x_i, \ 	ext{where} \ i = \text{stoa}(k, \tau, C)$$
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 \( \mathcal{D} \)
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 $\text{SPLIT}$ rule, which unrolls terms by branching on different constructors

Instead of introducing standard selectors, the $\text{SPLIT}$ rule introduces shared selectors
Calculus for Theory of Datatypes with Shared Selectors $\mathcal{D}$

The **Split** rule:

$$S_\delta^{\tau, n}(t) \in \mathbf{T}(E) \text{ or } \delta \text{ is finite}$$

\[
\begin{align*}
E & := E, t \approx C_1(S_\delta^{\tau_1, 1, \text{atos}(\tau_1, 1, C_1, 1)}(t), \ldots, S_\delta^{\tau_1, n_1, \text{atos}(\tau_1, n_1, C_1, n_1)}(t)) \\
& \vdots \\
E & := E, t \approx C_m(S_\delta^{\tau_m, 1, \text{atos}(\tau_m, 1, C_m, 1)}(t), \ldots, S_\delta^{\tau_m, n_m, \text{atos}(\tau_m, n_m, C_m, n_m)}(t))
\end{align*}
\]

▷ Consider again the datatype

$$\text{Tree} = N_1(\text{Int, Tree, Tree}) \mid N_2(\text{Int, Int, Tree, Tree}) \mid L(\text{Bool, Int})$$

▷ For a term $S^{\text{Tree}, 1}(t)$, the split would introduce a branch with

\[
E := E, t \approx N_1(S^{\text{Int, atos}(\text{Int, N}_1, 1)}(t), S^{\text{Tree, atos(\text{Tree, N}_1, 2)}(t), S^{\text{Tree, atos(\text{Tree, N}_1, 3)}(t)) \\
\approx N_1(S^{\text{Int}, 1}(t), S^{\text{Tree}, 1}(t), S^{\text{Tree}, 2}(t))
\]

Datatypes with Shared Selectors
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

\[
\begin{align*}
A & \to x \mid y \mid 0 \mid 1 \mid A+A \mid A-A \mid \text{ite}(B, A, A) \\
B & \to A \geq A \mid A \approx A \mid \neg B
\end{align*}
\]
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

\[
A \rightarrow x \mid y \mid 0 \mid 1 \mid A + A \mid A - A \mid \text{ite}(B, A, A) \\
B \rightarrow A \geq A \mid A \approx A \mid \neg B
\]

A solution is e.g. $f = \lambda xy. 0$ or $f = \lambda xy. x + y$
Encode problem using a deep embedding into datatypes

\[ a = X | Y | \text{Zero} | \text{One} | \text{Plus}(a, a) | \text{Minus}(a, a) | \text{Ite}(b, a, a) \]

\[ b = \text{Geq}(a, a) | \text{Eq}(a, a) | \text{Neg}(b) \]

represent the grammar \( R \) and the specification becomes

\[ \forall xy. \text{eval}_a(d, x, y) \approx \text{eval}_a(d, y, x) \]

where \( d \) is a fresh constant of type \( a \).

eval maps datatype terms to their corresponding theory terms

- \( \text{eval}_a(\text{Plus}(X, X), 2, 3) \) is interpreted as \( (x + x)\{x \mapsto 2, y \mapsto 3\} = 4 \)
Enumerative SyGuS in SMT

- Encode problem using a deep embedding into datatypes

\[ a = X | Y | \text{Zero} | \text{One} | \text{Plus}(a, a) | \text{Minus}(a, a) | \text{Ite}(b, a, a) \]
\[ b = \text{Geq}(a, a) | \text{Eq}(a, a) | \text{Neg}(b) \]

represent the grammar \( R \) and the specification becomes

\[ \forall xy. \text{eval}_a(d, x, y) \approx \text{eval}_a(d, y, x) \]

where \( d \) is a fresh constant of type \( a \).

- \( \text{eval} \) maps datatype terms to their corresponding theory terms
  - \( \text{eval}_a(\text{Plus}(X, X), 2, 3) \) 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 \( \text{Zero} \) or \( \text{Plus}(X, Y) \), 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

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

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.

$$\neg\text{isPlus}(S^\text{Int,1}(d)) \lor \neg\text{isX}(S^\text{Int,1}(S^\text{Int,1}(d))) \lor \neg\text{isZero}(S^\text{Int,2}(S^\text{Int,1}(d)))$$
	n rules out terms that have $x + 0$ as their first child of type $a$, such as

$$\begin{align*}
(x + 0) + y & \equiv x + y \\
\text{ite}(x \geq y, x + 0, y) & \equiv \text{ite}(x \geq y, x, y) \\
(x + 0) - 1 & \equiv x - 1
\end{align*}$$
Pruning the search space: symmetry breaking

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

\[
\neg \text{isPlus}(S^{\text{Int},1}(d)) \lor \neg \text{isX}(S^{\text{Int},1}(S^{\text{Int},1}(d))) \lor \neg \text{isZero}(S^{\text{Int},2}(S^{\text{Int},1}(d)))
\]

rules out terms that have \( x + 0 \) as their first child of type \( a \), such as

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

\( \triangleright \) Sharing selectors allows the same blocking clause to be reused for the different constructors

\( \triangleright \) standard selectors would require three different clauses in this case

\[
\neg \text{isPlus}(S^{\text{Plus},1}(d)) \lor \neg \text{isX}(S^{\text{Plus},1}(S^{\text{Plus},1}(d))) \lor \neg \text{isZero}(S^{\text{Plus},2}(S^{\text{Plus},1}(d))) \\
\neg \text{isPlus}(S^{\text{Ite},2}(d)) \lor \neg \text{isX}(S^{\text{Ite},2}(S^{\text{Plus},1}(d))) \lor \neg \text{isZero}(S^{\text{Ite},2}(S^{\text{Plus},2}(d))) \\
\neg \text{isPlus}(S^{\text{Minus},1}(d)) \lor \neg \text{isX}(S^{\text{Minus},1}(S^{\text{Plus},1}(d))) \lor \neg \text{isZero}(S^{\text{Minus},1}(S^{\text{Plus},2}(d)))
\]
Evaluation
Impact on SyGuS-COMP 2017 benchmarks

Datatypes with Shared Selectors

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

- Over 80% reduction in average number of selectors for PBE_BV
- PBE_Strings, General also show significant improvements
Comparison with other SygGuS solvers

<table>
<thead>
<tr>
<th>Family</th>
<th>#</th>
<th>EUSOLVER</th>
<th>CVC4-si-sh</th>
<th>CVC4-si-std</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>535</td>
<td>404</td>
<td>391</td>
<td>334</td>
</tr>
<tr>
<td>CLIA</td>
<td>73</td>
<td>71</td>
<td>73</td>
<td>73</td>
</tr>
<tr>
<td>Invariant</td>
<td>67</td>
<td>42</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>PBE_BV</td>
<td>750</td>
<td>739</td>
<td>665</td>
<td>253</td>
</tr>
<tr>
<td>PBE_Strings</td>
<td>108</td>
<td>68</td>
<td>93</td>
<td>64</td>
</tr>
</tbody>
</table>

- Comparison also includes CVC4’s single-invocation approach (impacts General and CLIA)
- CVC4 is only competitive on General, PBE\_Strings 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

<table>
<thead>
<tr>
<th>Family</th>
<th>#</th>
<th>Solved (both)</th>
<th>Time</th>
<th>Decs</th>
<th>Terms</th>
<th>Sels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sh</td>
<td>std</td>
<td>sh</td>
<td>std</td>
<td>sh</td>
<td>std</td>
</tr>
<tr>
<td>Leon</td>
<td>410</td>
<td>179 175 (175)</td>
<td>0.96</td>
<td>0.75</td>
<td>9.9k</td>
<td>9.9k</td>
</tr>
<tr>
<td>Sledgehammer</td>
<td>321</td>
<td>113 112 (112)</td>
<td>0.47</td>
<td>0.47</td>
<td>6.9k</td>
<td>6.9k</td>
</tr>
<tr>
<td>Nunchaku</td>
<td>158</td>
<td>67 67 (67)</td>
<td>0.49</td>
<td>0.44</td>
<td>7.1k</td>
<td>6.6k</td>
</tr>
</tbody>
</table>

- 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

—we introduced a correct decision procedure for the new theory

—we 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

—we 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