Abstract Code Search

Isabel Garcia-Contreras    Jose F. Morales    Manuel V. Hermenegildo
New York, ICLP 2016, October 18, 2016

IMDEA Software Institute, and
Technical University of Madrid
**Motivation:** Find code to reuse.

**Problem:** The large amount of code available makes it impossible for a programmer to find useful fragments manually.

**Current code finders** are mainly based on:

- Documentation keyword indexing (Maarek et al. 1991).
- Signature matching (Rollins and Wing 1991 – λ Prolog).
- Combinations of both (Mitchell 2008 – Hooble/Haskell).
Motivation: Find code to reuse.

Problem: The large amount of code available makes it impossible for a programmer to find useful fragments manually.

Current code finders are mainly based on:
- Documentation keyword indexing (Maarek et al. 1991).
- Signature matching (Rollins and Wing 1991 – λ Prolog).
- Combinations of both (Mitchell 2008 – Hoole/Haskell).

Our approach:
- Find code based on its semantic characteristics.
  (that can be inferred automatically).
Assertions are a fundamental component of our approach. They are **linguistic constructions** for expressing abstractions of the meaning and behavior of programs.

<table>
<thead>
<tr>
<th>pred assertions (subset)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow specifying certain conditions on the state (current substitution or constraint store) that must hold at certain points of program execution.</td>
</tr>
</tbody>
</table>

\[-\text{pred } \text{Head} : \text{Pre} \Rightarrow \text{Post}.\]

- **Head**: normalized atom that denotes the predicate that the assertion applies to.
- **Pre** and **Post**: conjunctions of "prop" atoms.
Herein we will use *assertions* for:

- **Programming**: To specify calling modes for predicates.
- **[New!] Searching**: To express properties of the code to be found.
Our approach

Main idea of our method:

1. A **set of modules** is specified within which code is to be found.
Main idea of our method:

1. A **set of modules** is specified within which code is to be found.

```prolog
:- module(lengths, [check_length/2, gen_list/2], [assertions]).
:- pred check_length(L,N) : (list(L), int(N)).
check_length(L,N) :- length(L,N).

:- pred gen_list(L,N) : (var(L), var(N)) => (list(L), int(N)).
gen_list(L,N) :- length(L,N).

% length implementation ...
```
Our approach

Main idea of our method:

1. A **set of modules** is specified within which code is to be found.
2. A **static pre-analysis** is made to:
   - Infer semantic properties in one or more abstract domains
     (e.g.: shapes/types, variable sharing, inst. modes, polyhedra, ...).

Example:

```prolog
?- pred P(X,Y) : list(X) => sorted(Y).
```

Based on (Stulova et al. 2014).
Shapes and Modes/Sharing analysis:

1. `% :- pred check_length(L,N) : (list(L), int(N)).`
2. `:- true pred check_length(L,N) : (list(L), int(N))`  
`=> (list(L), int(N)).`
3. `:- true pred check_length(L,N) : (mshare([L],[L,N],[N]))`  
`=> (mshare([L]), ground([N])).`
4. `check_length(L,N) :- length(L,N).`
5. `%:- pred gen_list(L,N) : ((var(L), var(N))).`
6. `:- true pred gen_list(L,N) : ((term(L), term(N))`  
`=> ((list(L), int(N))).`
7. `:- true pred gen_list(L,N) : (mshare([L],[L,N],[N])), var(L), var(N))`  
`=> (mshare([L]), ground([N])).`
8. `gen_list(L,N) :- length(L,N).`
9. `% length implementation ...`
Main idea of our method:

1. A set of modules is specified within which code is to be found.
2. A static pre-analysis is made to:
   - Infer semantic properties in one or more abstract domains (e.g.: shapes/types, variable sharing, inst. modes, polyhedra, ...).
3. User specifies semantic properties in a query, using a new kind of assertions that we call query assertions:
   - Example: :- pred P(X,Y) : list(X) => sorted(Y).

Based on (Stulova et al. 2014).

4. Look within the modules for predicates that meet those properties (by comparing to inferred information).
Inferring semantic properties

Abstract interpretation:
- To simulate the execution using an abstract domain $D_{\alpha}$.
- It guarantees:
  - Analysis termination, provided that $D_{\alpha}$ meets some conditions.
  - Results are safe approximations of the concrete semantics.

### PLAI algorithm (in Ciao)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$: Program</td>
<td>$\text{Analysis}(P, L:\lambda, D_{\alpha}) = { \langle L_1, \lambda_1^c, \lambda_1^s \rangle, \ldots, \langle L_n, \lambda_n^c, \lambda_n^s \rangle }$, where:</td>
</tr>
<tr>
<td>$D_{\alpha}^i$: Abstract Domain(s)</td>
<td>- $L_i$ is an atom</td>
</tr>
<tr>
<td>$Q_{\alpha} = L:\lambda$: Initial call pattern</td>
<td>- $\lambda_i^c$ are abstract call state in $D_{\alpha}$</td>
</tr>
<tr>
<td></td>
<td>- $\lambda_i^s$ are abstract success state in $D_{\alpha}$</td>
</tr>
</tbody>
</table>
Predicate query

?- findp({ As }, M:Pred/A, Residue, Status).

<table>
<thead>
<tr>
<th>Input</th>
<th>As: Set of query assertions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>M:Pred/A: Module, predicate descriptor and arity.</td>
</tr>
<tr>
<td></td>
<td>Residue: Information of condition matching.</td>
</tr>
<tr>
<td>Input or Output</td>
<td>Status: Result of the proof for the whole set of conditions.</td>
</tr>
<tr>
<td></td>
<td>• checked if all conditions are proved to be checked.</td>
</tr>
<tr>
<td></td>
<td>• false if any condition is false.</td>
</tr>
<tr>
<td></td>
<td>• check if neither checked nor false can be proved.</td>
</tr>
</tbody>
</table>

?- findp({ :- pred P(L, S) => (list(L), num(S)). }, M:P/A, Res, St).

P/A = check_length/2    St = checked
## Normalizing the query

### Assertion Conditions

Given a predicate represented by a normalized atom \( \text{Head} \), and a corresponding set of assertions \( \mathcal{A} = \{A_1 \ldots A_n\} \), with \( A_i = \text{\texttt{:- pred Head : \texttt{Pre}_i \Rightarrow \texttt{Post}_i}} \). The set of assertion conditions for \( \text{Head} \) determined by \( \mathcal{A} \) is \( \{C_0, C_1, \ldots, C_n\} \), with:

\[
C_i = \begin{cases} 
\text{calls(Head, } \bigvee_{j=1}^{n} \text{Pre}_j) & i = 0 \\
\text{success(Head, Pre}_i, Post}_i) & i = 1..n 
\end{cases}
\]

1. \( \text{:- pred my_length(L,N) : ((list(L), \texttt{var(N)}) \Rightarrow ((list(L), \texttt{int(N))}).} \)
2. \( \text{:- pred my_length(L,N) : ((list(L), \texttt{int(N)}) \Rightarrow ((list(L), \texttt{int(N))}.} \)
3. \( \text{my_length(L,N) :- length(L,N).} \)

Assertion conditions from \texttt{my_length/2}:

\[
C_i = \begin{cases} 
\text{calls(} \text{length(L,N)}, (\text{list(L)} \land \text{var(N)}) \lor (\text{list(L)} \land \text{int(N)})), & (\text{list(L), int(N)))}, \\
\text{success(} \text{length(L,N)}, (\text{list(L)} \land \text{var(N)}), & (\text{list(L), int(N))}, \\
\text{success(} \text{length(L,N)}, (\text{list(L)} \land \text{int(N)}), & (\text{list(L), int(N))} 
\end{cases}
\]
Matching success conditions

Match \( C = \text{success}(X(V_1, \ldots, V_n), \text{Pre}, \text{Post}) \) against \( \text{analysis}(P, Q_\alpha) \)

### Checked matches

If \( \text{Pre} \) holds at the time of calling the matching predicate and the execution succeeds then the \( \text{Post} \) conditions hold.

\( C \) is abstractly ‘checked’ for predicate \( p \in P \) w.r.t. \( Q_\alpha \) in \( D_\alpha \) iff

\[
\exists L = p(V_1', \ldots, V_n') \text{ s.t. } \forall \langle L, \lambda^c, \lambda^s \rangle \in \text{analysis}(P, Q_\alpha) \text{ s.t. } \exists \sigma \in \text{ren}, \ L = p(V_1', \ldots, V_n') = X(V_1, \ldots, V_n)\sigma, \lambda^c \sqsubseteq \lambda^+_{TS(Pre \ \sigma, P)} \implies \lambda^s \sqsubseteq \lambda^-_{TS(Post \ \sigma, P)}
\]

### False matches

If \( \text{Pre} \) holds at the time of calling the matching predicate and the execution succeeds then its conditions and \( \text{Post} \) are disjoint.

\( C \) is abstractly false for \( p \in P \) w.r.t. \( Q_\alpha \) in \( D_\alpha \) iff \( \exists L = p(V_1', \ldots, V_n') \text{ s.t. } \forall \langle L, \lambda^c, \lambda^s \rangle \in \text{analysis}(P, Q_\alpha) \text{ s.t. } \exists \sigma \in \text{ren}, \ L = p(V_1', \ldots, V_n') = X(V_1, \ldots, V_n)\sigma, \lambda^c \sqsubseteq \lambda^-_{TS(Pre \ \sigma, P)} \land (\lambda^s \sqcap \lambda^+_{TS(Post \ \sigma, P)} = \bot) \)
Perfect b1.
mixed b0.

Perfect b0.
mixed b1.
mixed z.

Reduced b1.

Hard X :- functor(b1(_), X, _).

Out b z.

:- regtype b/1.

b(b0).
b(b1).
Matching \textit{success} conditions - Example

\begin{verbatim}
:- true pred perfect(A) => b(A).
perfect(b1).
perfect(b0).

:- true pred mixed(X) => top(X).
mixed(b0).
mixed(b1).
mixed(z).

:- true pred reduced(A) => t1(A).
reduced(b1).

:- true pred outb(A) => t2(A).
outb(z).

:- regtype b/1.
b(b0).
b(b1).

:- regtype t1/1.
t1(b1).

:- regtype t2/1.
t2(z).
\end{verbatim}

\[
D = \{b0, b1, z\} \quad D_\alpha = \{\top, b, t1, t2, \bot\}
\]

\[
\begin{tikzpicture}
  \node (top) at (0,0) {$\top$};
  \node (b) at (-1,-1) {$b$};
  \node (t1) at (-2,-2) {$t1$};
  \node (t2) at (0,-2) {$t2$};
  \node (bot) at (0,-3) {$\bot$};
  \draw (top) -- (b);
  \draw (b) -- (t1);
  \draw (t1) -- (bot);
  \draw (top) -- (t2);
\end{tikzpicture}
\]
Matching success conditions - Example

```prolog
:- true pred perfect(A) => b(A).
perfect(b1).
perfect(b0).

:- true pred reduced(A) => t1(A).
reduced(b1).

:- true pred outb(A) => t2(A).
outb(z).

:- regtype b/1.
b(b0).
b(b1).

:- regtype t1/1.
t1(b1).

:- regtype t2/1.
t2(z).

?- findp({ :- pred P(V) => b(V). }, M:P/A, Res, St).
P/A = perfect/1    St = checked
P/A = reduced/1    St = checked
P/A = outb/1       St = false
P/A = mixed/1      St = check
P/A = hard/1       St = check

D = {b0, b1, z}
Dα = {⊤, b, t1, t2, ⊥}
```

Abstract Code Search
Matching calls conditions - Example

:- pred check_length(L,N) : ((list(L), int(N))).
check_length(L,N) :- length(L,N).

:- pred gen_list(L,N) : ((var(L), var(N))).
gen_list(L,N) :- length(L,N).

:- pred get_length(L,N) : var(N).
get_length(L,N) :- length(L,N).

?- findp({ :- pred P(L, Size) : var(L), var(Size)). }, M:P/A, Res, St).
    P/A = check_length/2      St = false
    P/A = gen_list/2          St = checked
    P/A = get_length/2        St = check
Combining abstract domains

?- findp({ :- pred P(L, Size) : list(L), num(Size)). }, M:P/A, Res, St).

<table>
<thead>
<tr>
<th>PredName/A</th>
<th>regtypes proof</th>
<th>shfr proof</th>
<th>combined proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_length/2</td>
<td>checked</td>
<td>check</td>
<td>checked</td>
</tr>
<tr>
<td>gen_list/2</td>
<td>check</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Conclusions

Finding code by its semantic characteristics:

- Ensures that the found code behaves correctly.
- Reasons with relations between properties (implication, abstraction).
- Is independent from the documentation.
- Implemented in Ciao, in combination with other types of search (which it complements).

Future work:

- Use more domains.
- Extend to other programming languages and combinations.
- Other uses: finding duplicated code.
Thanks!
Matching \textit{calls} \textit{conditions}

\[
C = \text{calls}(X(V_1, \ldots, V_n), Pre)
\]

\textbf{Checked matches}

The admissible calls of the matching predicate are within the set of \textit{Pre} conditions.

\[C\] is abstractly ‘checked’ for a \(p \in P\) w.r.t. \(Q_\alpha\) in \(D_\alpha\) iff

\[
\forall \langle L, \lambda^c, \lambda^s \rangle \in \text{analysis}(P, D_\alpha, Q_\alpha) \text{ s.t. } \exists \sigma \in \text{ren}, \ L = p(V_1, \ldots, V_n) = X(V_1, \ldots, V_n)\sigma, \lambda^c \sqsubseteq \lambda^-_{TS(Pre, \sigma, P)}.
\]

\textbf{False matches}

The admissible calls of the matching predicate and the set \textit{Pre} conditions are disjoint.

\[C\] is abstractly ‘false’ for a \(p \in P\) w.r.t. \(Q_\alpha\) in \(D_\alpha\) iff

\[
\forall \langle L, \lambda^c, \lambda^s \rangle \in \text{analysis}(P, D_\alpha, Q_\alpha) \text{ s.t. } \exists \sigma \in \text{ren}, \ L = p(V_1, \ldots, V_n) = X(V_1, \ldots, V_n)\sigma, \lambda^c \sqcap \lambda^+_{TS(Pre, \sigma, P)} = \perp.
\]
### Performance

Search times in $\mu$s.

<table>
<thead>
<tr>
<th>Ar\Cnds</th>
<th>1</th>
<th>1 (AVG)</th>
<th>2</th>
<th>2 (AVG)</th>
<th>3</th>
<th>3 (AVG)</th>
<th>4</th>
<th>4 (AVG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (85 pr)</td>
<td>19,064</td>
<td>224</td>
<td>53,530</td>
<td>630</td>
<td>180,246</td>
<td>2,121</td>
<td>298,292</td>
<td>3,509</td>
</tr>
<tr>
<td>2 (74 pr)</td>
<td>110,092</td>
<td>1,488</td>
<td>207,871</td>
<td>2,809</td>
<td>221,061</td>
<td>2,987</td>
<td>477,440</td>
<td>6,452</td>
</tr>
<tr>
<td>3 (47 pr)</td>
<td>294,962</td>
<td>6,276</td>
<td>3,757,208</td>
<td>79,941</td>
<td>3,806,917</td>
<td>80,998</td>
<td>6,127,015</td>
<td>130,362</td>
</tr>
<tr>
<td>4 (12 pr)</td>
<td>5,116</td>
<td>426</td>
<td>12,939</td>
<td>1,078</td>
<td>22,508</td>
<td>1,876</td>
<td>30,300</td>
<td>2,525</td>
</tr>
</tbody>
</table>
The theme of the presentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

The source of this theme is located at:

github.com/matze/mtheme