Type-Directed Completion of Partial Expressions

Daniel Perelman†

Sumit Gulwani‡  Thomas Ball‡  Dan Grossman†

†University of Washington
‡Microsoft Research Redmond

June 12, 2012
I want to shrink an image...

```c
Document image = ...; Size newSize = ...;
image.Shrink(newSize)
```
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...; image.
```
I want to shrink an image...

```java
Document image = ...; Size newSize = ...;
image.
```
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...; image.
```
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...; image.
```
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
PaintDotNet.
```
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
PaintDotNet.
```
I want to shrink an image...

```
Document image = ...; Size newSize = ...;
PaintDotNet.
```
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
```

PaintDotNet.
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
```

![Image conversion options](image.png)
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
PaintDotNet.Document;
```
I want to shrink an image...

I want to shrink an image...
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
PaintDotNet.Data.
```

![TgaFileType](image)
I want to shrink an image...

```
Document image = ...; Size newSize = ...;
PaintDotNet.Actions.
```
I want to shrink an image...

Document image = ...; Size newSize = ...;
PaintDotNet.Actions.
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
PaintDotNet.Actions.
```

![Dropdown menu showing various actions including `ZoomOutAction` highlighted.](image)
I want to shrink an image...

Document image = ...; Size newSize = ...;
PaintDotNet.Actions.
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
```
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
PaintDotNet.Actions.
```
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
```
I want to shrink an image...

```csharp
Document image = ...; Size newSize = ...;
    .ResizeDocument(
        /* PaintDotNet.Document image */,
        /* System.Drawing.Size size */,
        /* PaintDotNet.AnchorEdge edge */,
        /* PointDotNet.ColorBgra bgColor */);
```
Programmer thought process

- I have a Document and a Size
- I want to shrink the Document
- There must be a method
Programmer thought process

- I have a Document and a Size
- I want to shrink the Document
- There must be a method

- Current code completion
  - Left-to-right
  - Complete, alphabetic list of just next token
  - Very limited filtering
Proposed workflow

```javascript
Document image = ...; Size newSize = ...;
var newImage = ?({image, newSize})
```
Proposed workflow

```csharp
Document image = ...; Size newSize = ...;

var newImage = ?({image, newSize})
```

```csharp
Partial Expression Completer

```
```
Programmer thought process

- I have a Document and a Size
- I want to shrink the Document
- There must be a method

- Query should contain what the programmer knows
  - Some values and types the expression should involve
  - Loose syntactic structure
- Query shouldn’t require what the programmer doesn’t know
  - Names
  - Argument order
  - Other arguments
- Show “best” results first
- Similar in spirit to Prospector [Mandelin et. al., PLDI’05]
Overview

- Expression of API queries as partial expressions
- Algorithm to generate results quickly in ranked order
- Experiment showing simple queries represent real code well
Unknown method queries

- Ex. ?:\{\text{image, size}\}

  - ⇒ PaintDotNet.Actions.CanvasSizeAction
    .ResizeDocument(img, size, ⬤, ⬤)
  - ⇒ PaintDotNet.Functional.Func.Bind(⬤, size, img)
  - ⇒ PaintDotNet.Pair.Create(size, img)
  - ⇒ PaintDotNet.Quadruple.Create(size, img, ⬤, ⬤)
  - ⇒ PaintDotNet.Triple.Create(size, img, ⬤)
  - ⇒ PaintDotNet.PropertySystem
    .StaticListChoiceProperty
    .CreateForEnum(img, size, ⬤)
  - ⇒ System.Drawing.Size.Equals(size, img)
  - ⇒ System.Object.ReferenceEquals(size, img)
Unknown lookup queries

- Ex. `float f = pointPair.*`
  - ⇒ `pointPair.P1.X`
  - ⇒ `pointPair.P1.Y`
  - ⇒ `pointPair.P2.X`
  - ⇒ `pointPair.P2.Y`
  - ⇒ `pointPair.Midpoint.X`
  - ⇒ `pointPair.Midpoint.Y`
  - ⇒ `pointPair.FirstValidValue().X`
  - ⇒ `pointPair.Length`
Unknown expression queries

- Ex. `XmlReader xr = ?`
  - ⇒ `System.Xml.XmlReader.Create(◊)`
  - ⇒ `new System.Xml.XmlNodeReader(◊)`
  - ⇒ `new System.Xml.XmlNodeReader(◊).ReadSubtree()`
  - ⇒ `new System.Xml.XmlValidatingReader(◊).Reader`
  - ⇒ `new System.Xml.XmlValidatingReader(◊).Reader.ReadSubtree()`
Partial expression language

(a) \[ e ::= \text{call} | \text{varName} | e \cdot \text{fieldName} | e := e | e < e \]
\[ \text{call} ::= \text{methodName}(e_1, \ldots, e_n) \]

(b) \[ \tilde{e} ::= \tilde{a} | ? | \diamond \]
\[ \tilde{a} ::= e | \tilde{a} \cdot \ast | \tilde{\text{call}} | \tilde{e} := \tilde{e} | \tilde{e} < \tilde{e} \]
\[ \tilde{\text{call}} ::= ?(\{\tilde{e}_1, \ldots, \tilde{e}_n\}) | \text{methodName}(\tilde{e}_1, \ldots, \tilde{e}_n) \]
Partial expression language

(a) \[ e ::= \text{call} | \text{varName} | e.\text{fieldName} | e:=e | e<e \]

\[ \text{call} ::= \text{methodName}(e_1, \ldots, e_n) \]

(b) \[ \tilde{e} ::= \tilde{a} | \tilde{?} | \diamond \]

\[ \tilde{a} ::= e | \tilde{a}.\ast | \text{call} | \tilde{e}:=\tilde{e} | \tilde{e}<\tilde{e} \]

\[ \text{call} ::= \tilde{?}(\{\tilde{e}_1, \ldots, \tilde{e}_n\}) | \text{methodName}(\tilde{e}_1, \ldots, \tilde{e}_n) \]

▶ Ex. \[ \tilde{?}(\{\text{strBuilder.\ast}, e.\ast\}) \]

\[ \Rightarrow \tilde{?}(\{\text{strBuilder, e.StackTrace}\}) \]

\[ \Rightarrow \text{strBuilder.Append(e.StackTrace)} \]
Algorithm

Problem: given query, generate completions
Method index by parameter type

ArrayList
2299 methods
BinarySearch
Reverse
...

ICloneable
2211 methods
Clone

IList
2257 methods
Add
Remove
...

Object
2210 methods
Equals
GetHashCode
Registry.SetValue
Array.IndexOf
IList.Add
Console.WriteLine
...

ArrayList
BinarySearch
Reverse
...
Infinite results

- **Problem: too many results**
  - inefficient to generate thousands of results to show only 20 to the programmer
  - programmer does not want to look at every result
  - result set is often infinite

- **Ex.** `var res = foo.*;`
  - ⇒ foo
  - ⇒ foo.GetType()
  - ⇒ foo.GetType().GetType()
  - ⇒ foo.GetType().GetType().GetType()
  - ⇒ foo.GetType().GetType().GetType().GetType()
  - ⇒ ...

- **Solution: generate in ranked order**
Algorithm

- Simple structurally recursive algorithm
- Group by type to minimize redundant work
- Generate results in ranking order
  - Allows determination of top $n$ without computing all results
Heuristics: Type distance

![Diagram showing the relationship between Object, IDrawingElement, Shape, and Rectangle.]

- Object
- IDrawingElement
- Shape
- Rectangle
Heuristics: Type distance

Object \rightarrow \text{Shape} \rightarrow \text{Rectangle}

\text{Object} \rightarrow \text{IDrawingElement} \rightarrow \text{Shape} \rightarrow \text{Rectangle}

2 \rightarrow 2

1 \rightarrow 0
Heuristics: Length

- Number of field/property lookups or method calls added
Heuristics: Length

- Number of field/property lookups or method calls added
- \(?({\text{strBuilder}.*, e.*})\)

Good (1): \(\Rightarrow\) `strBuilder.Append(e.StackTrace)`

Bad (3): \(\Rightarrow\) `strBuilder.Clear().Append(e.Data.Count)`
Heuristics: Inferred abstract types

Example usages elsewhere in codebase:
string f = Path.GetTempFileName(); ...;
File.Delete(f);

File.Delete(Path.Combine(dir, filename));
if(File.Exists(Path.Combine(otherDir, file))) {...}

Query:
string p = Path.GetTempFileName();
?(p)
⇒ GetCursor(p)
⇒ File.Delete(p)
⇒ File.Exists(p)
Ranking function

- Linear combination of these and other heuristics
- Sensitivity analysis showed these are most important and coefficients do not matter much
Outline

Motivation

Approach
  Language
  Algorithm
  Ranking

Experiment
  Results

Related work

Conclusion
Experiment

- Automated test of expressiveness of partial expressions
- Generated queries for each call and looked at rank of actual call in query results

- Advantage: able to do many queries
- Disadvantage: many of the method calls are not ones a programmer would need API discovery for
Experiment

- Used Microsoft CCI to disassemble mature C# projects
- Converted every call with at least 3 arguments (including receiver) to a query with 1 or 2 arguments (including receiver)
  - For `ResizeDocument(document, size, anchorEdge, background)` 16 queries would be generated:
    \[ \Rightarrow \text{?(document)} \Rightarrow \text{?(size)} \Rightarrow \text{?(anchorEdge)} \Rightarrow \text{?(background)} \Rightarrow \text{?(document, size)} \Rightarrow \text{?(document, background)} \Rightarrow \ldots \]
- Report rank for best-performing query for each call
Projects used

- Paint.NET image editor
- Windows Installer XML library
- Gnome Do program launcher
- Banshee music player
- .NET core libraries
- Family.Show (WPF example application)
- LiveGeometry geometry visualizer

- Scale: .NET contains 280,000 methods in 30,000 types
- Analyzed 21,176 method calls in these applications
CDF of rank for best method query

Proportion of analyzed calls
Rank of correct answer is < x
?({foo, bar})
baz.
Partial expressions
Code completion
CDF of rank for best method query (correct is static)

Proportion of analyzed static calls

Rank of correct answer is < x

?({foo, bar})

NS.Baz.

Partial expressions

Code completion
CDF of rank for best method query

Rank of correct answer is < x

Using two arguments
Using one argument
Code completion

Proportion of analyzed calls
Other experiments

- Time: unknown method queries take under 0.1 second
- Ran similar experiments on other partial expression templates
- Similar results: one argument or one lookup could be predicted within the top 10 about 80% of the time
Related work

- Lots of other work on API discovery discussed in paper
Related work

- Lots of other work on API discovery discussed in paper

- Prospector (for Java) [Mandelin et. al., PLDI’05]
  - Input is target type
    - Similar to `XmlParser xr = ? query`
  - Uses mined expressions which convert from one type to another
  - Output is chain of mined expressions starting with some local

- Advantage: able to synthesize larger expressions
- Disadvantage: queries only specify a single input type and a single output type
Contributions

- Expressed API searches in terms of partial expressions
- Leveraged rich type structure to reduce information needed for queries
- Automated experiments across large codebases show small partial expressions often match real method calls

- Created Visual Studio plugin
  - https://pec.codeplex.com/