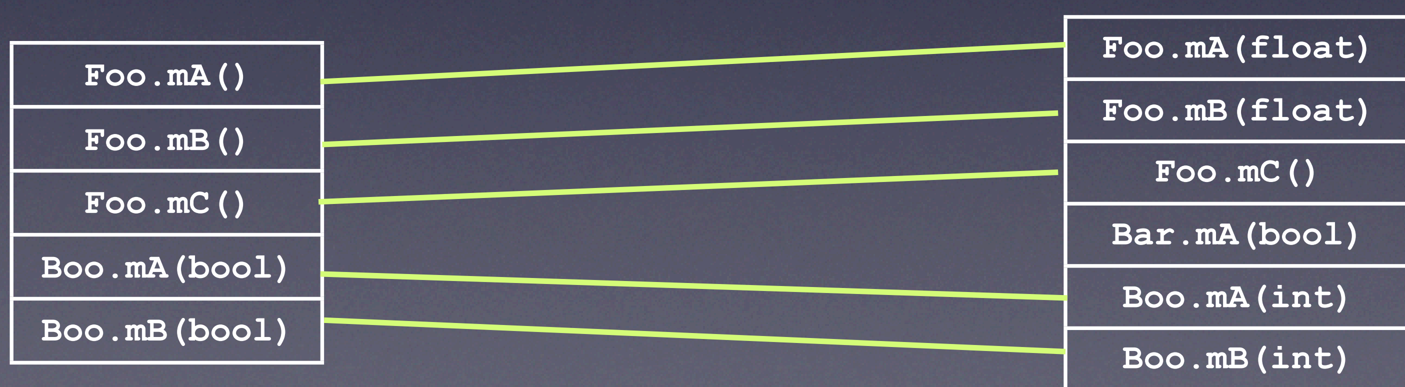
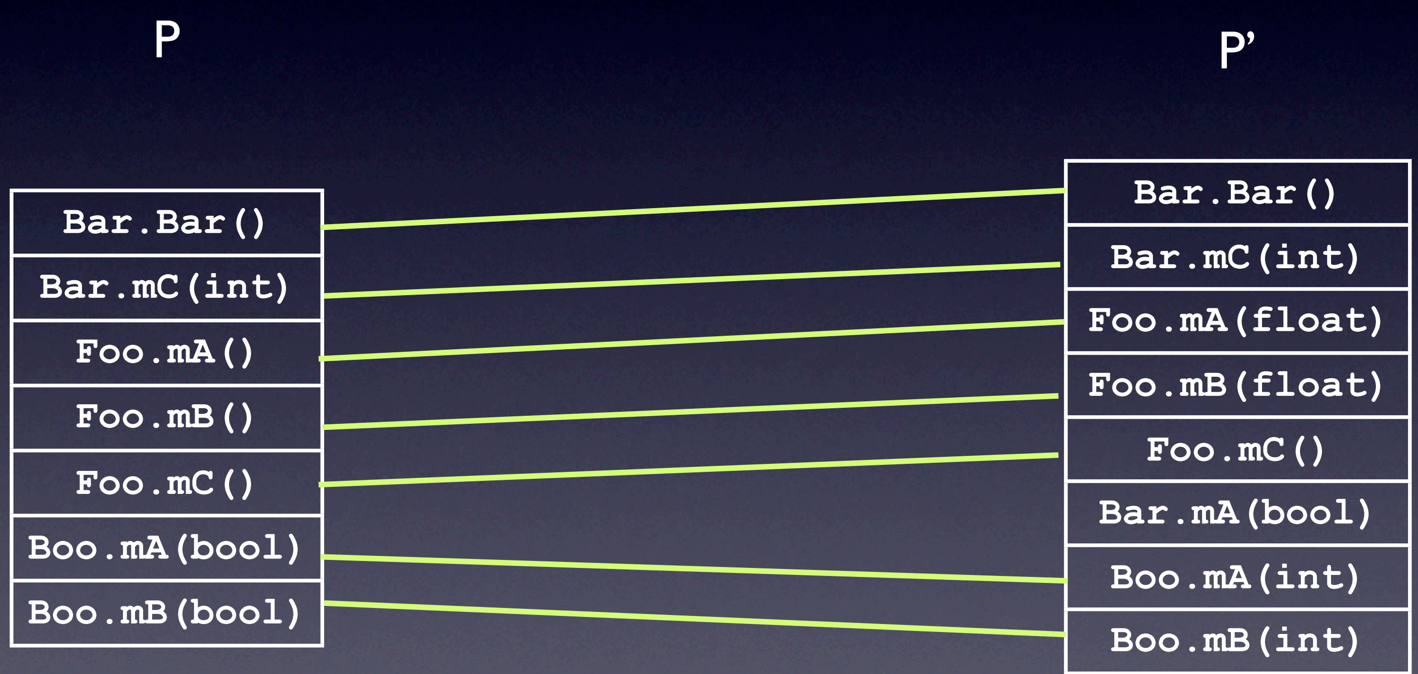
Automatic Inference of Structural Changes for Matching Across Program Versions

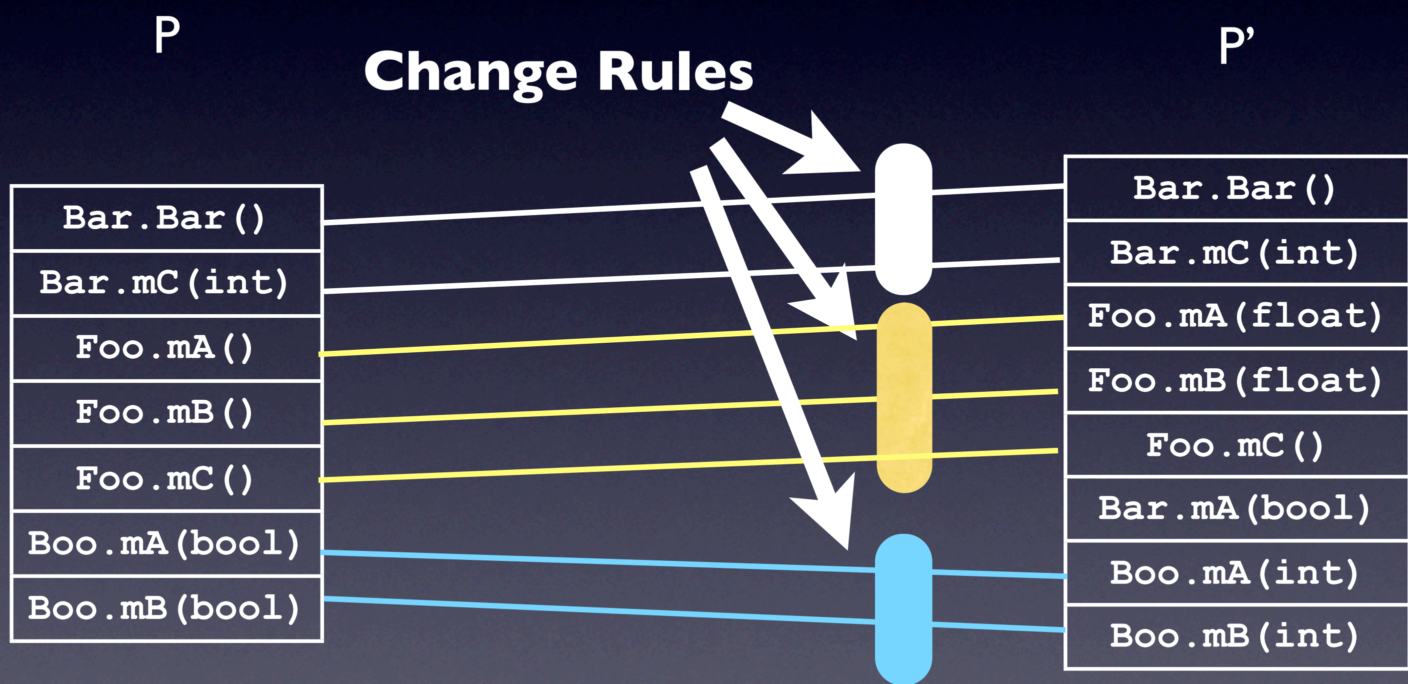
Miryung Kim, David Notkin, Dan Grossman
Computer Science & Engineering
University of Washington



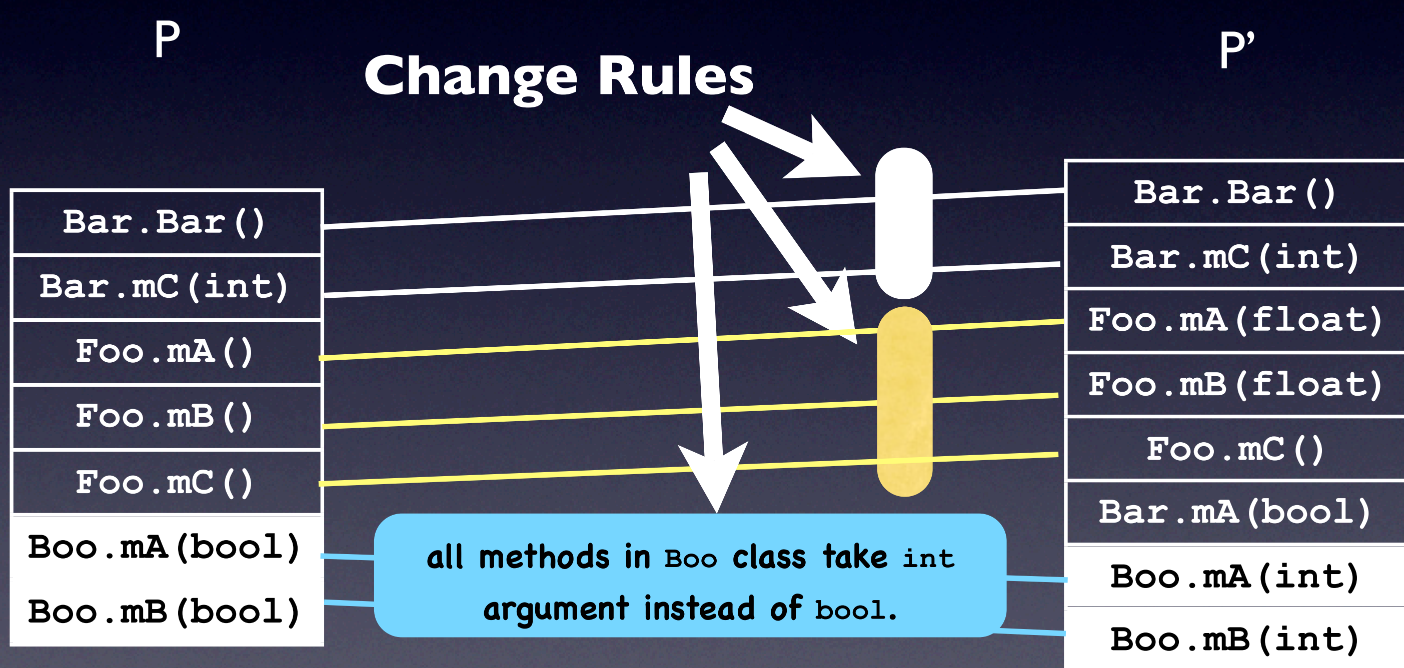
Code Matching Problem



Our Approach: Matching with Change Rules



Our Approach: Matching with Change Rules



Motivations for Matching Code

- A fundamental building block for mining software repositories
- Also a basis for classic software evolution research and tools
  - Software version merging
  - Regression testing
  - Profile propagation

Matching is Challenging.

- Matching is hard due to code addition & deletion, copy & paste, refactorings, etc.
- Delta between two versions can be very large.
- For many uses, matching results must be concise and comprehensible.

Outline

- **background**
- our rule-based matching approach
- inference algorithm
- evaluation
- potential applications of change rules

Matching Problem ≈ Change Identification Problem

The problem of identifying code matches

The problem of identifying changes

Existing Approaches

diff, Syntactic Diff (CDiff), Semantic Diff, JDiff, origin analysis, refactoring reconstruction tools, etc.

Individually compare code elements
at particular granularities
using similarity measures

Limitations of Existing Approaches

P

P’

Limitations of Existing Approaches

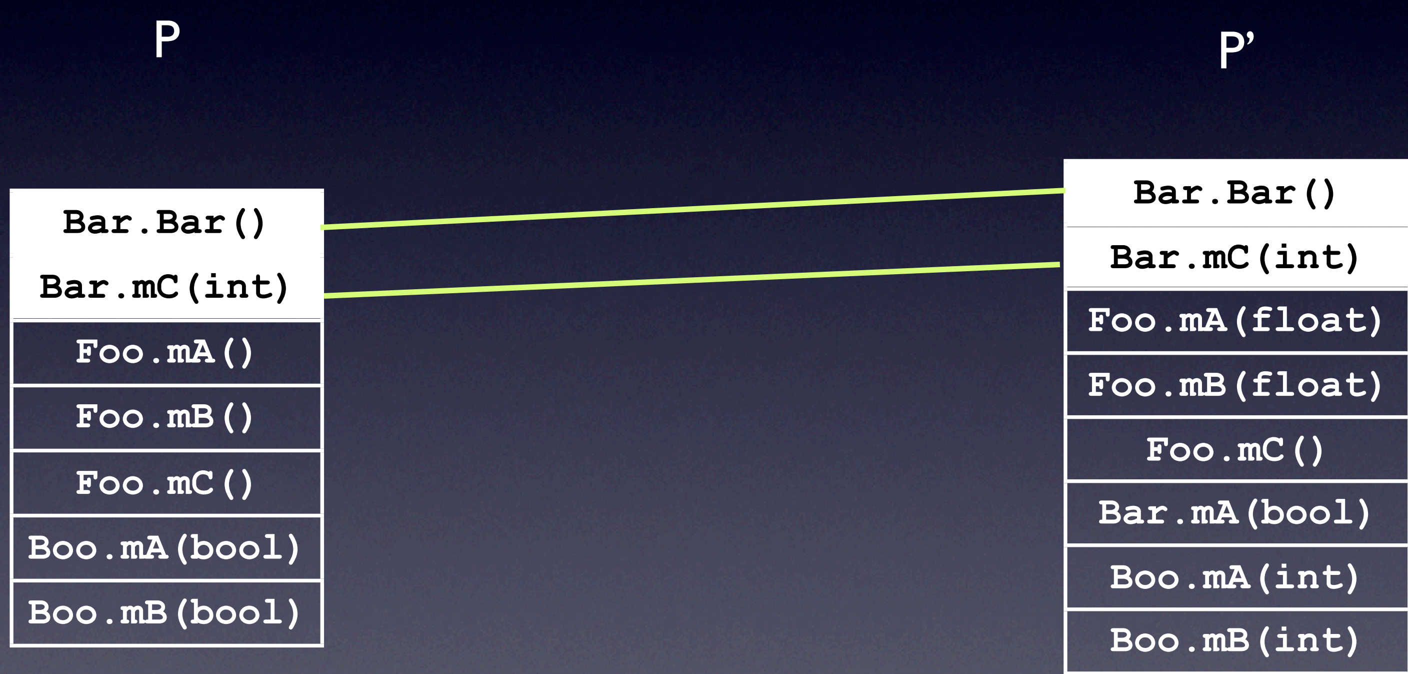
P

Bar.Bar()
Bar.mC(int)
Foo.mA()
Foo.mB()
Foo.mC()
Boo.mA(bool)
Boo.mB(bool)

P’

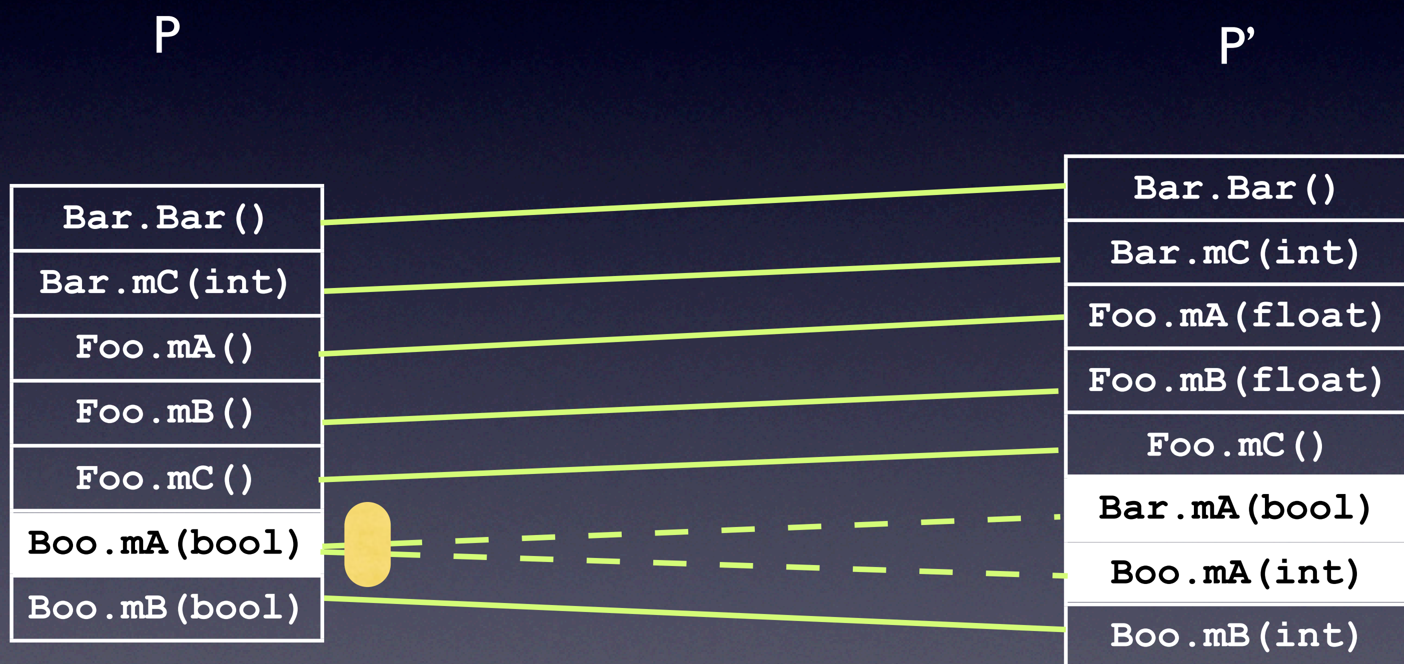
Bar.Bar()
Bar.mC(int)
Foo.mA(float)
Foo.mB(float)
Foo.mC()
Bar.mA(bool)
Boo.mA(int)
Boo.mB(int)

Limitations of Existing Approaches



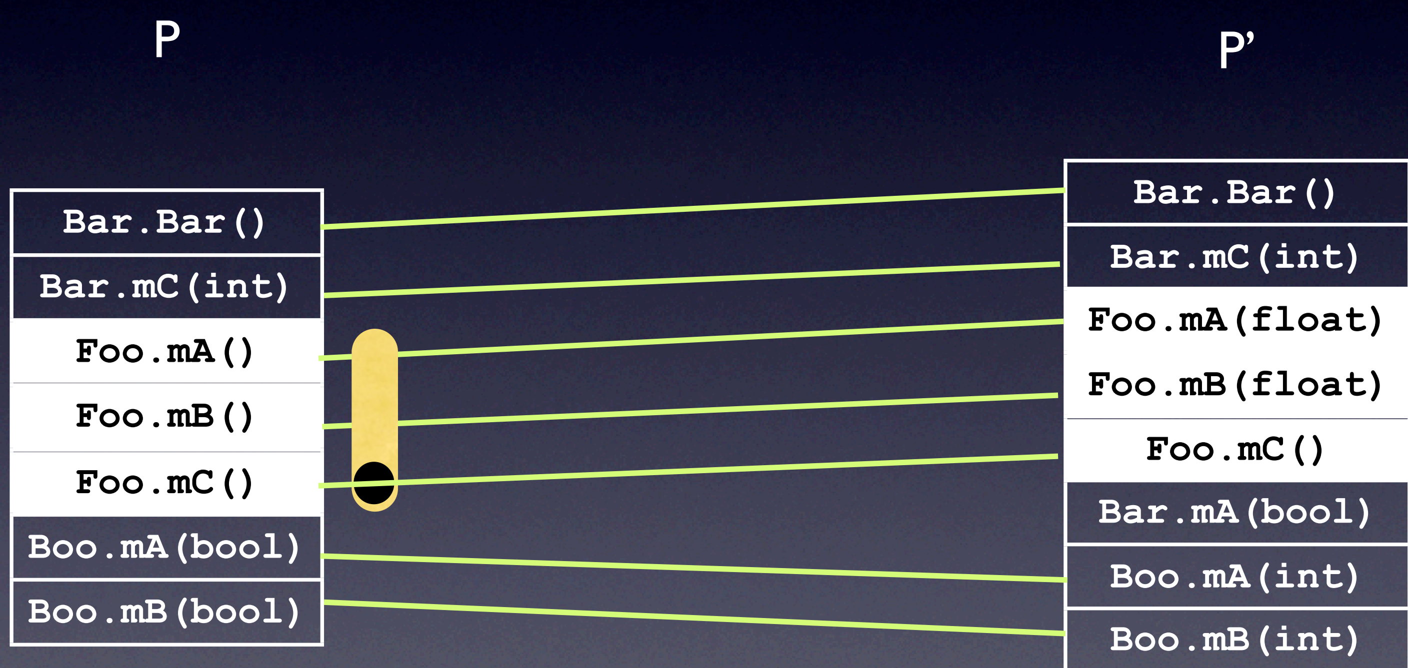
Limitations of Existing Approaches

Limitations of Existing Approaches



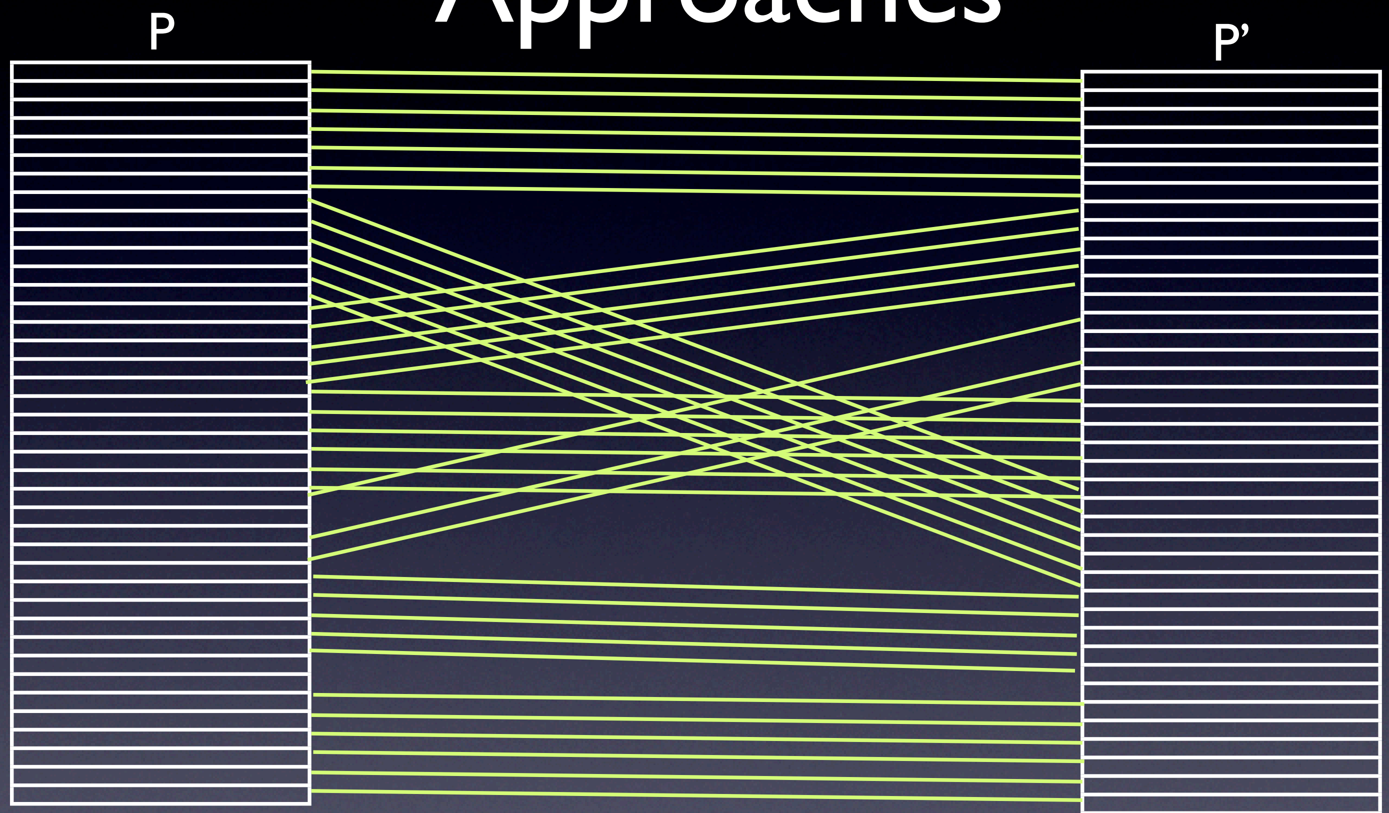
Cannot disambiguate among many potential matches

Limitations of Existing Approaches



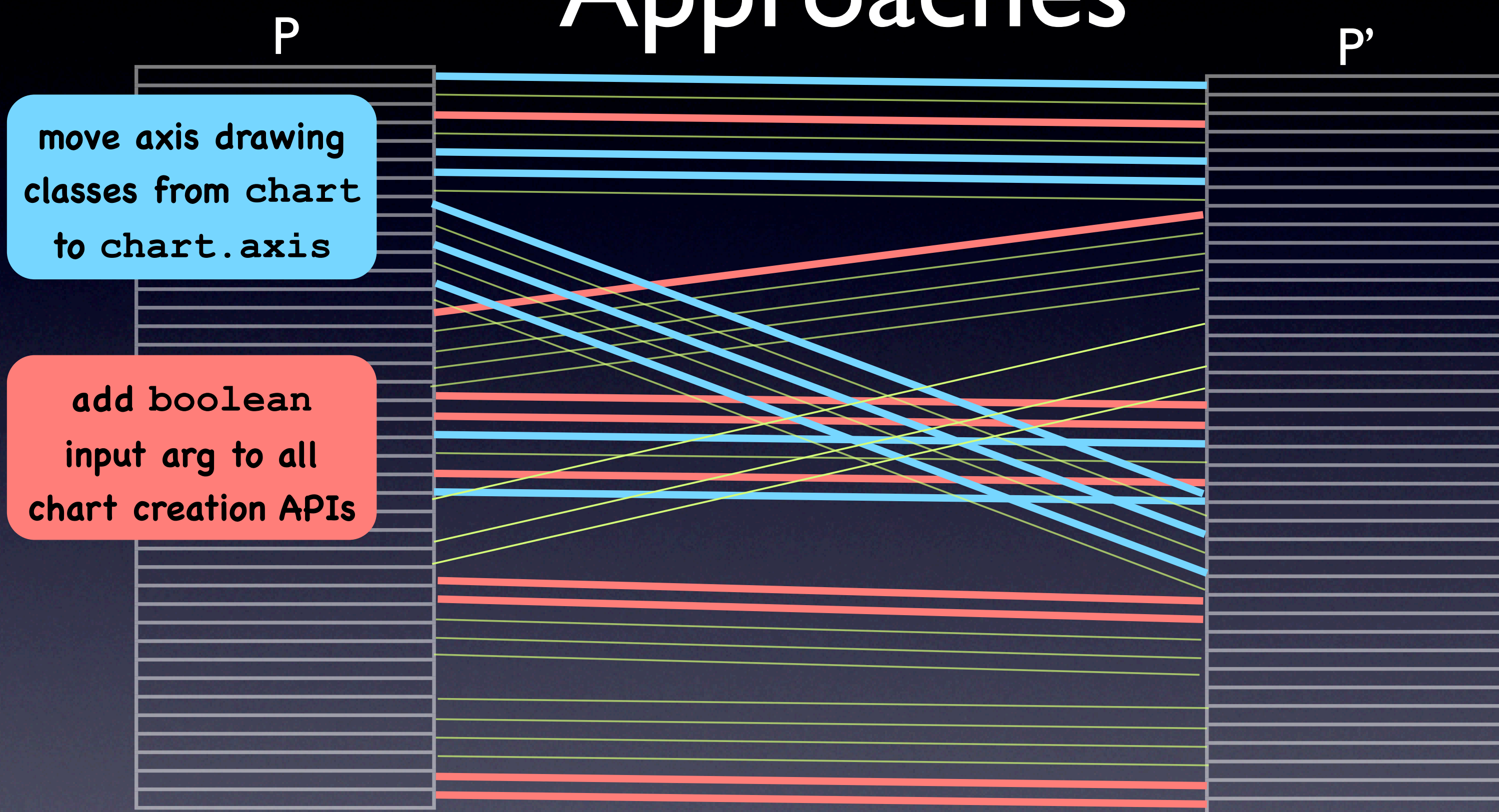
Difficult to spot inconsistent and incomplete changes

Limitations of Existing Approaches



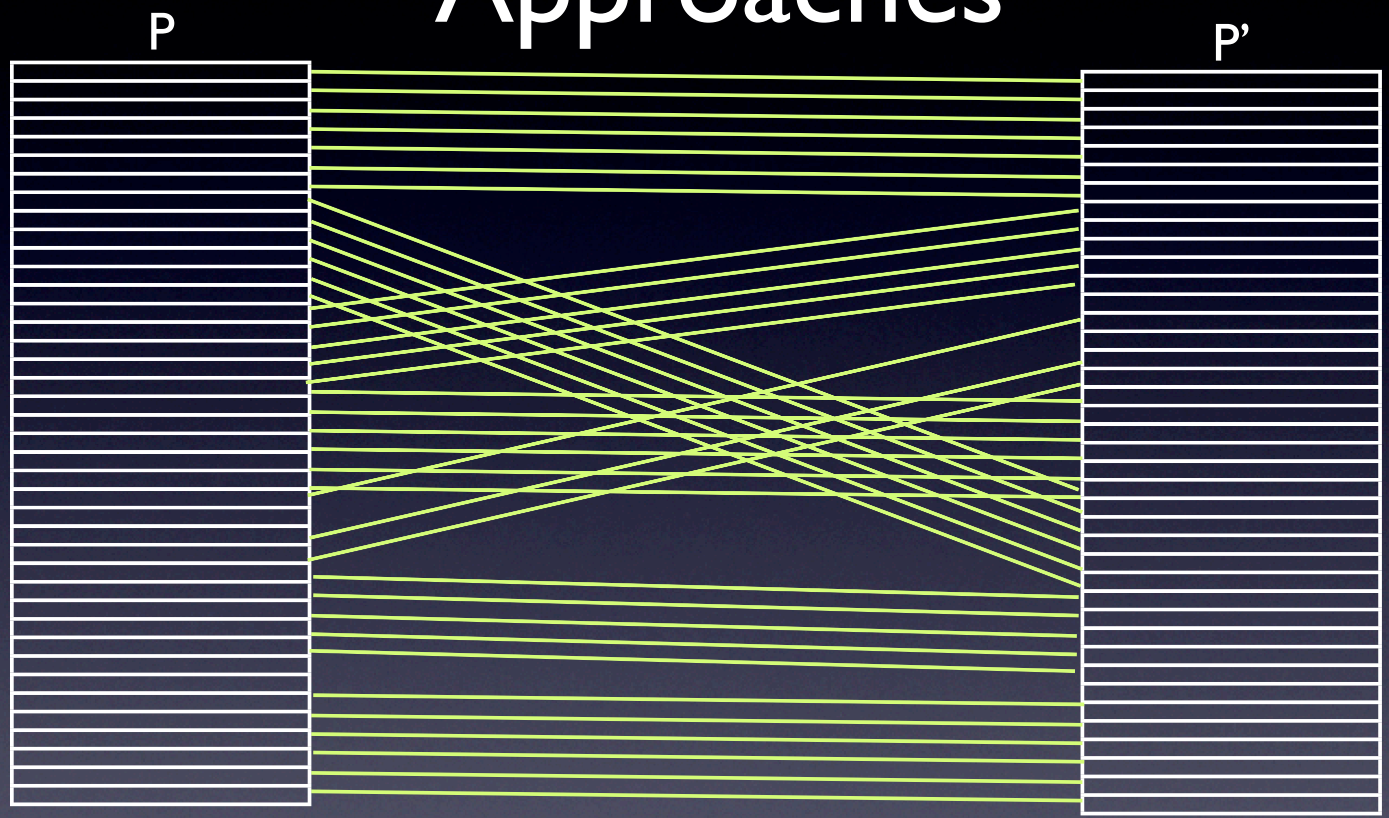
Output is an unstructured, usually lengthy list of matches

Limitations of Existing Approaches



Output is an unstructured, usually lengthy list of matches

Limitations of Existing Approaches



Output is an unstructured, usually lengthy list of matches

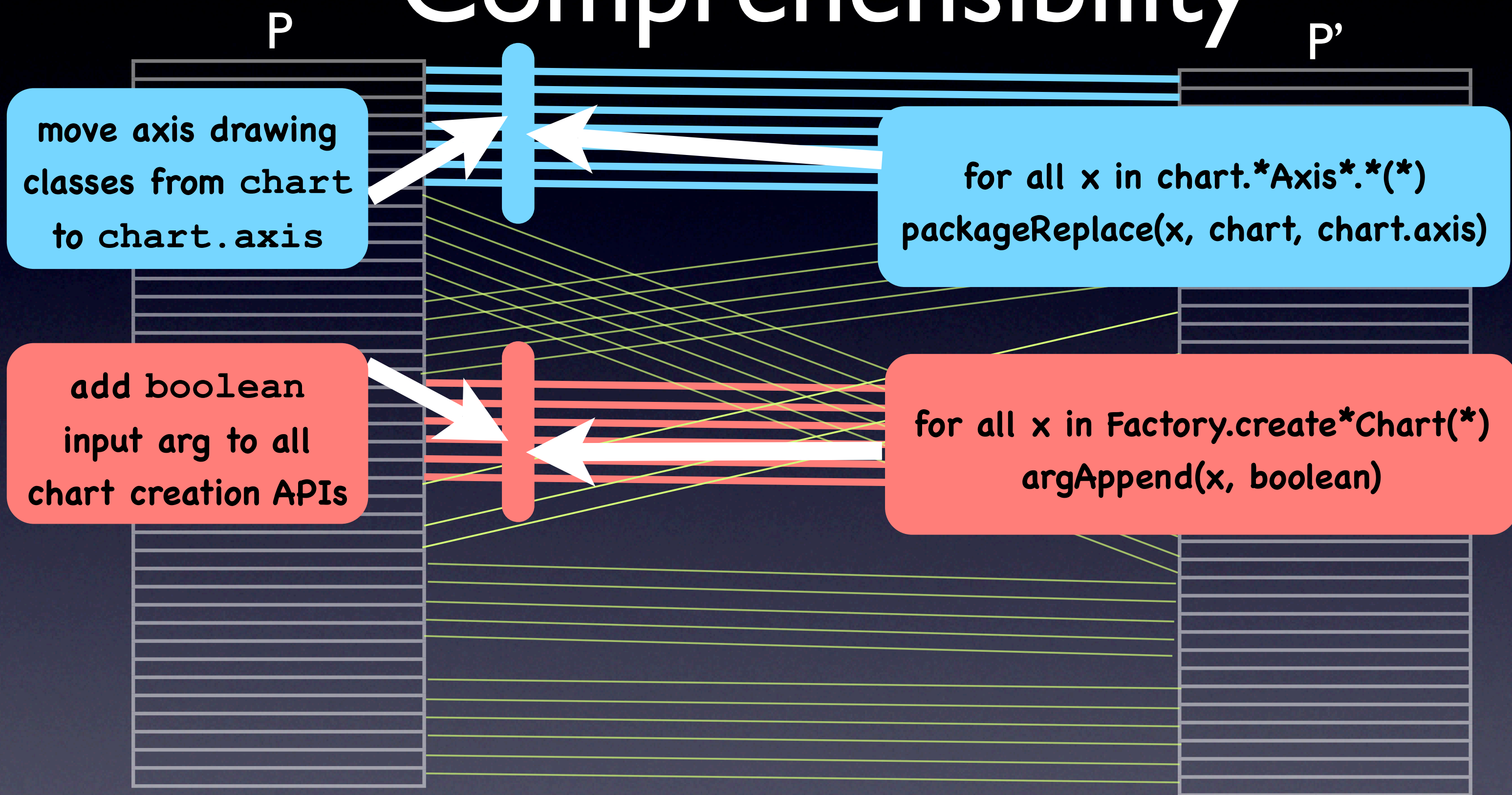
Outline

- ✓ background
- **our rule-based matching approach**
- inference algorithm
- evaluation
- potential applications of change rules

Our Rule-based Matching Approach

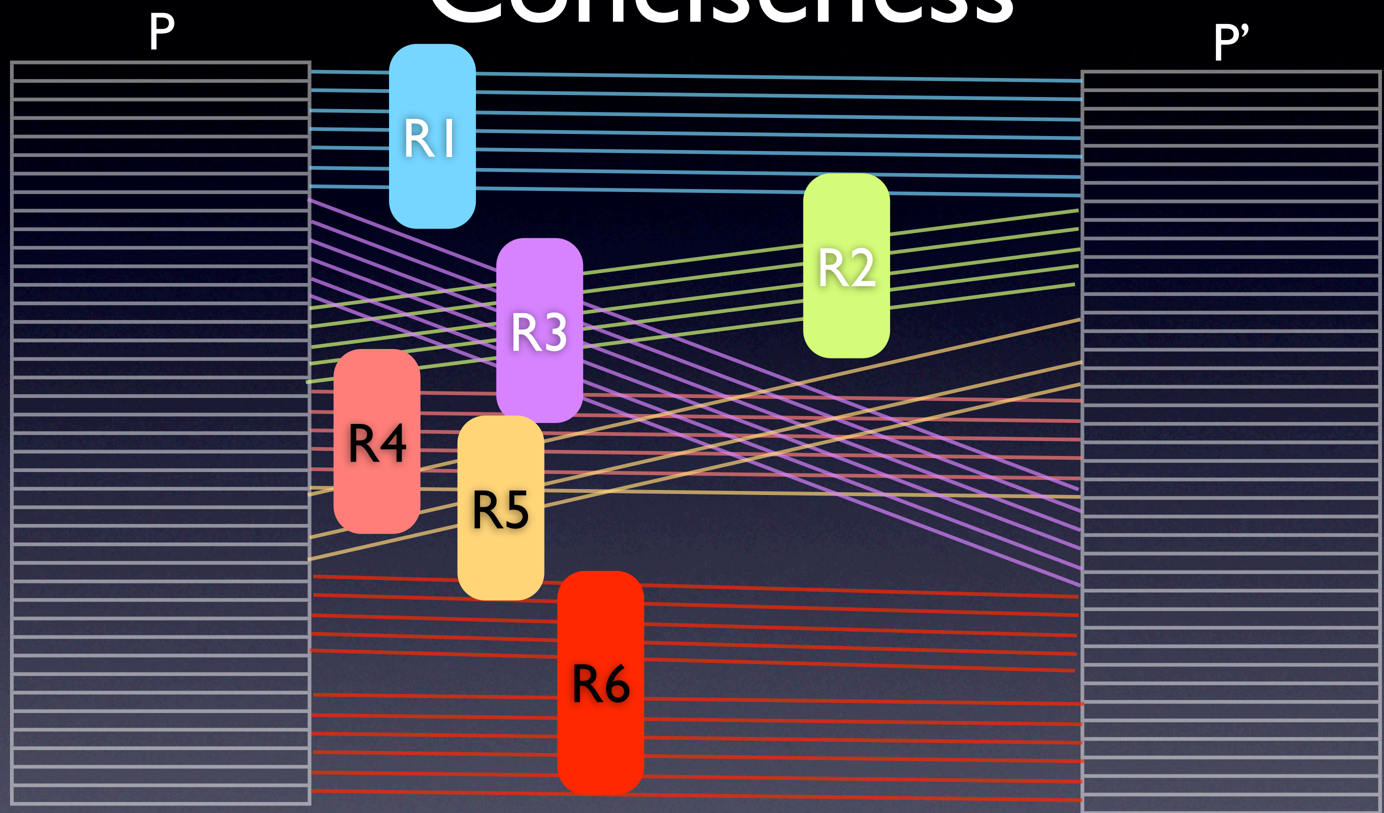
- Our ***change rule*** can concisely describe a set of related refactorings and API changes at or above the method header level.
- Our tool ***automatically infers*** a set of ***likely change rules*** between two versions of a program.

Our Contribution 1. Comprehensibility



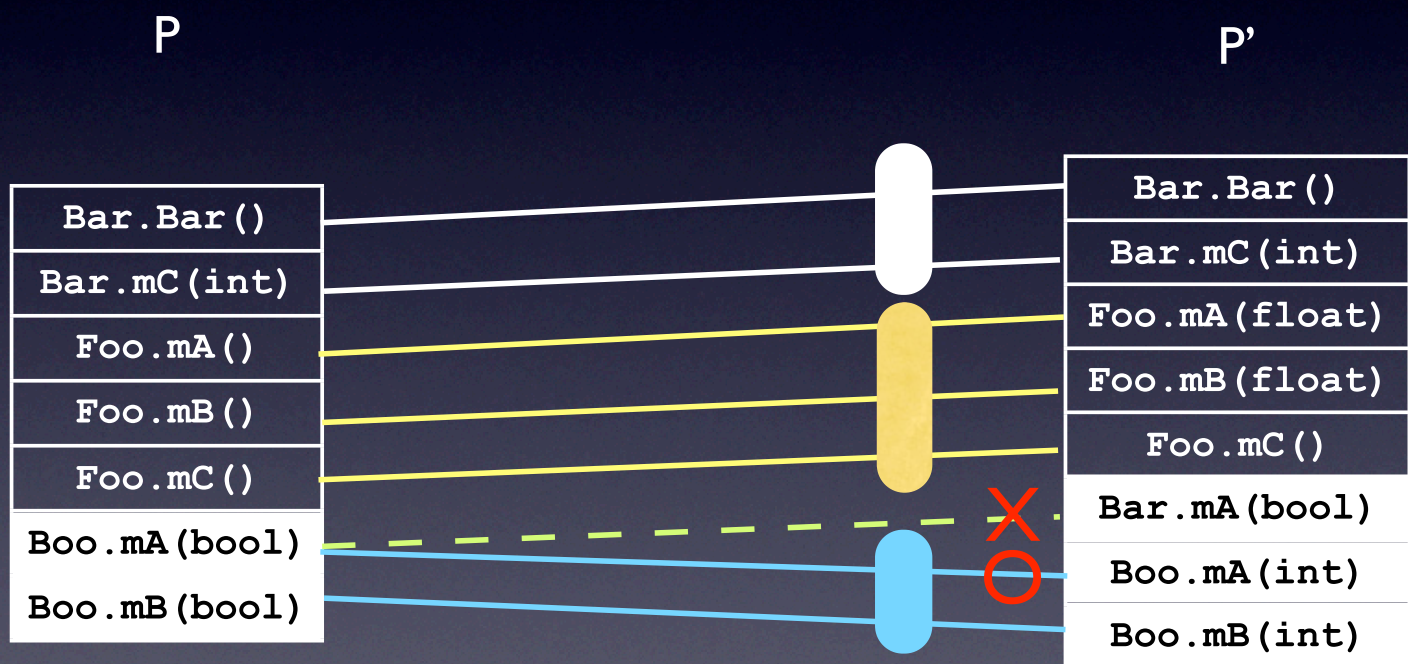
Represent a high-level change pattern using a change rule
➡ **Easy to understand** change intent

Our Contribution 2. Conciseness



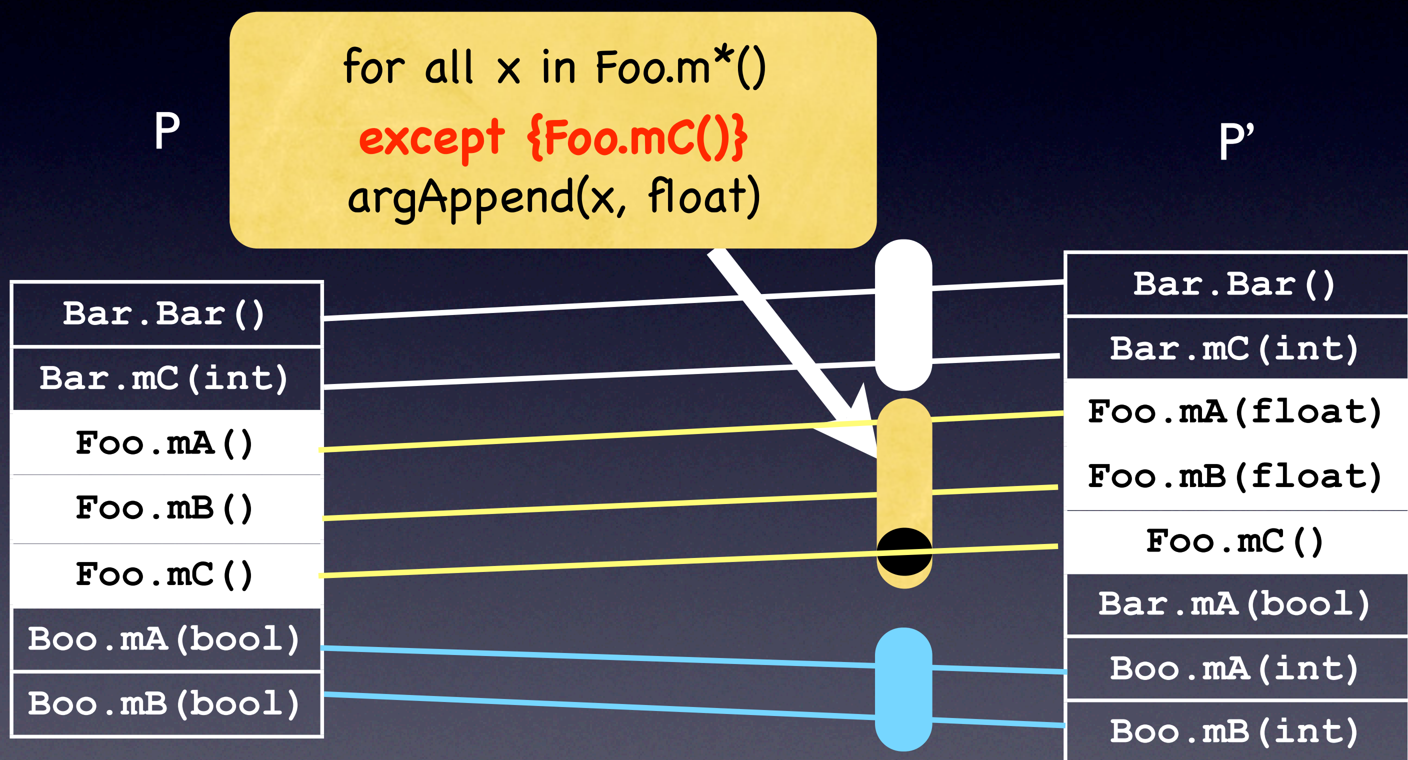
Concisely represent large deltas using a small number of change rules

Our Contribution 3.
High Recall



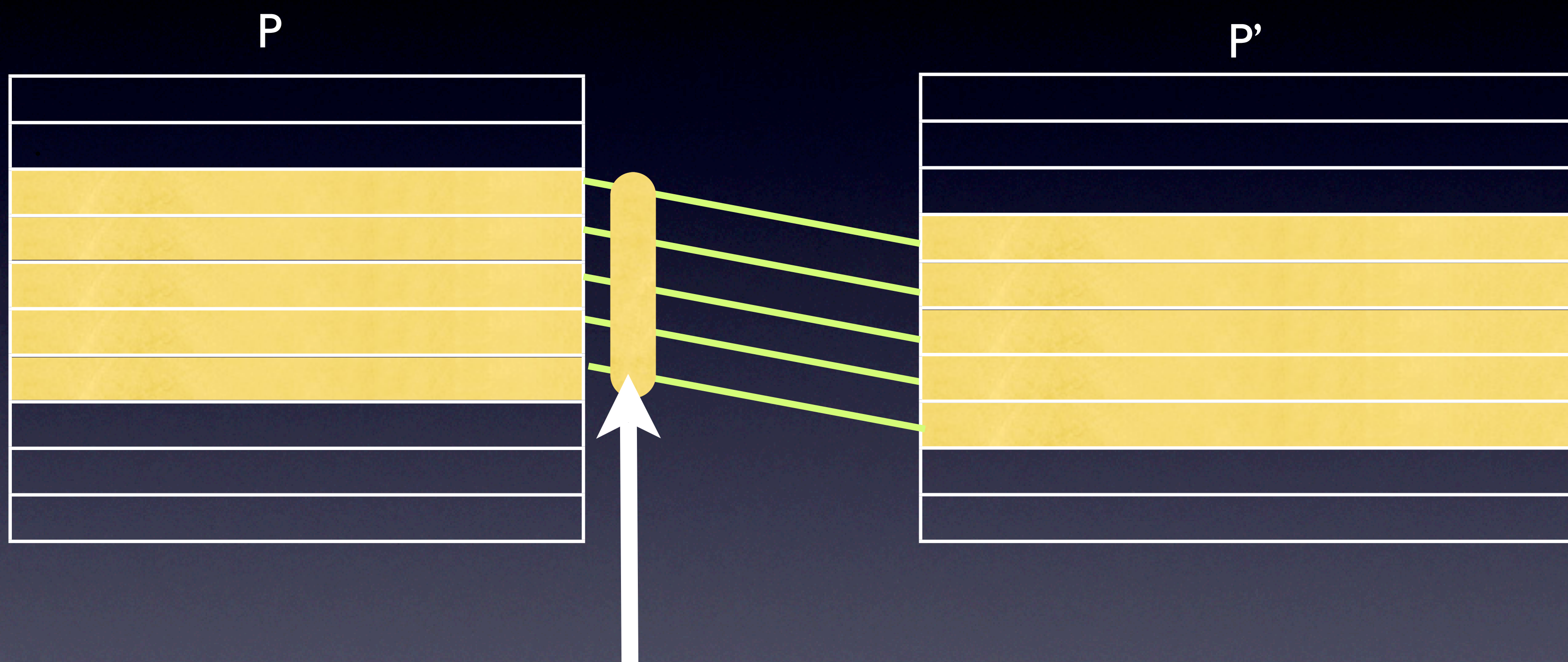
Find matches evidenced by a more general change pattern
➡ **Improving recall**

Our Contribution 4. Explicit Exceptions



Our rule **encodes exceptions explicitly**
➡ **Easy to notice** inconsistent and incomplete changes

Change Rule




```
for all x:method in scope
transformation(x)
```

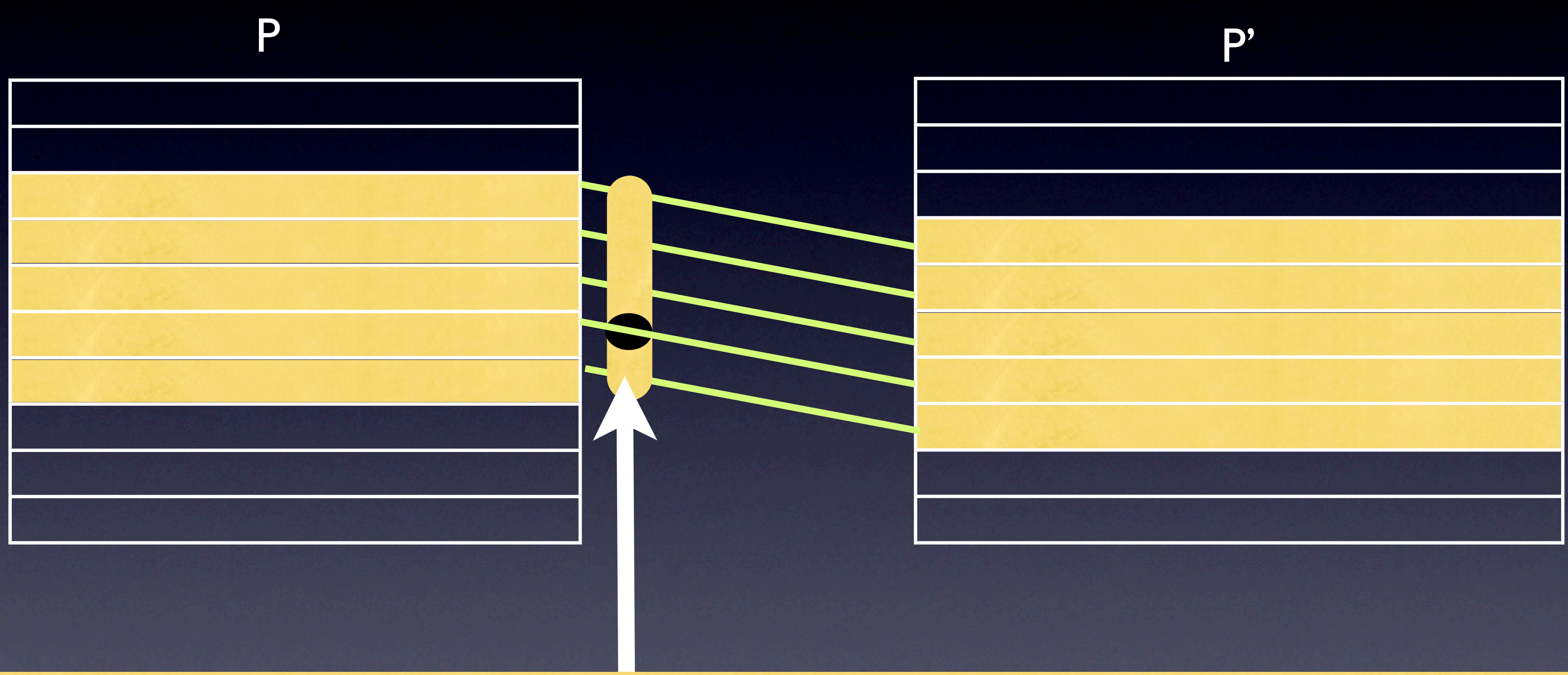
Scope

- We use a regular expression to denote a set of methods
  - e.g. *chart.Factory.create*Chart(*)*

Transformations At or Above the Level of Method Header

- 9 types of transformations representing:
  - replace the name of package, class, and method
  - replace the return type
  - modify the input signature, etc.

Change Rule with Exceptions




```
for all x:method in (scope - exceptions)
        transformation(x)
```

Example Change Rule

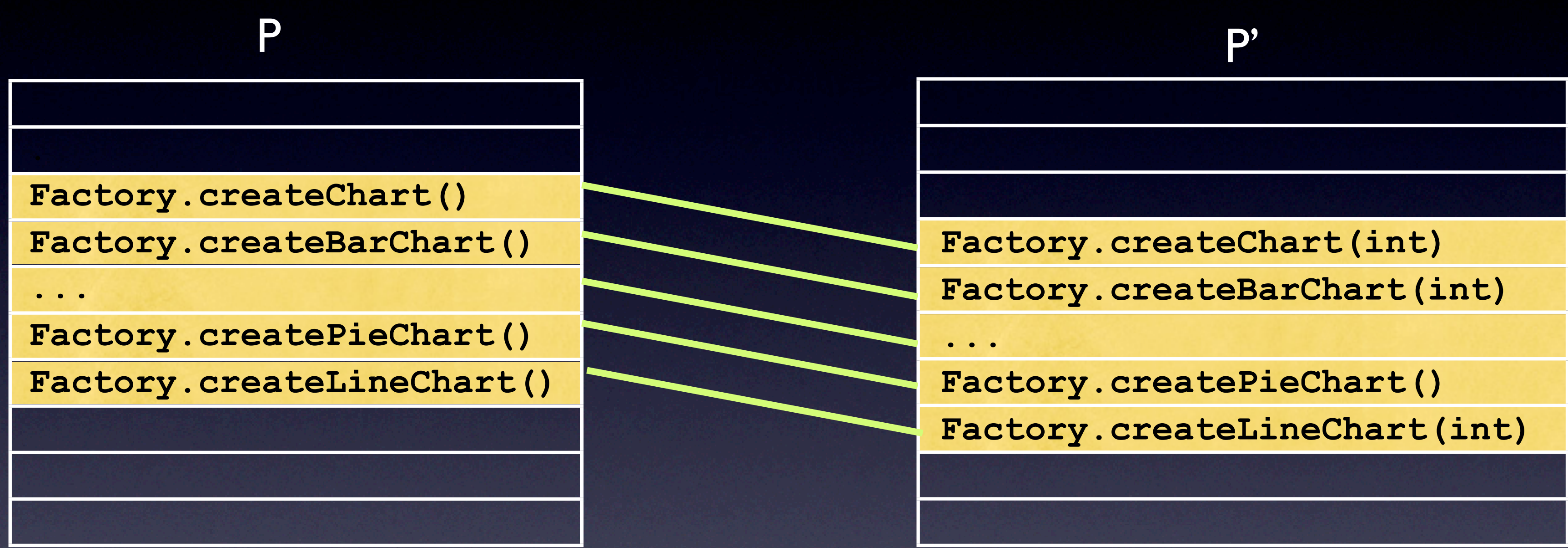
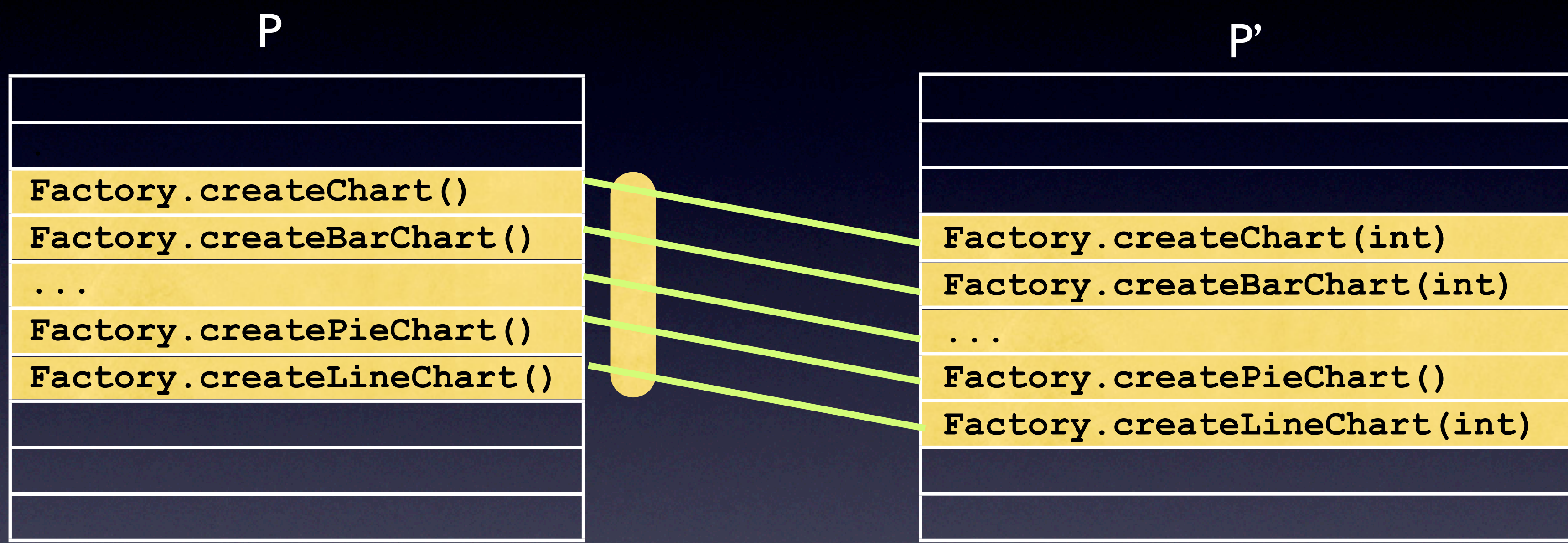


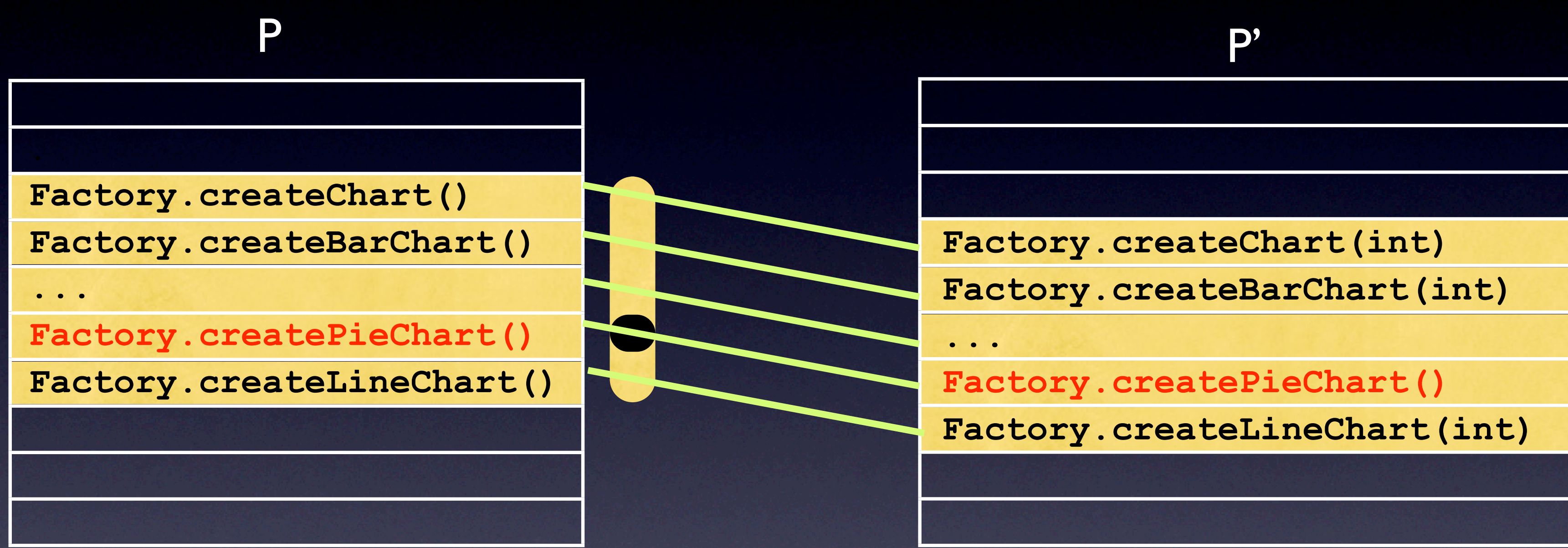
Chart creation APIs were changed to take an additional `int` parameter.

Example Change Rule



For all x in Factory.create*Chart(*)
argAppend(x, [int])

Example Change Rule



For all x in Factory.create*Chart(*)
except {Factory.createPieChart()}
argAppend(x, [int])
14 matches and 1 exception

Outline

- ✓ background
- ✓ our rule-based matching approach
- **inference algorithm**
- evaluation
- potential applications of change rules

Inference Algorithm Overview

Input: two versions of a program

Output: a set of likely change rules

1. Generate seed matches
2. Generate candidate rules by generalizing seed matches
3. Evaluate and select candidate rules (greedy algorithm)

Step 1: Generate Seed Matches

- Seed matches provide **hints** about likely changes.
- We generate seeds based on textual similarity between two method headers.
- Seed matches need not be all correct matches.

textual similarity: 0.75

```
Foo.getBar(int)
```

```
Foo.getBar(bool)
```

Step 2: Generate Candidate Rules for each seed *[x, y]*

- Compare **x** and **y** and reverse engineer a set of **transformations, T**.
- Based on **x**, guess a set of **scopes, S**.
- Generate **candidate rules** for each pair in **S × PowerSet(T)**.

```
Given a seed match,
[Foo.getBar(int), Boo.getBar(bool)]

Transformations = {
replaceArg(x, int, bool)
replaceClass(x, Foo, Boo)}

Scopes = {*.*(*), Foo.*(*), ...,
 *.get*(*), *.*Bar(*), ... ,
 Foo.get*(int),... }

Candidate Rules = {
 for all x in *.*(*)
  replaceArg(x, int, bool),
 for all x in Foo.*(*)
  replaceClass(x, Foo, Boo), ...,
 for all x in *.*(*)
  replaceArg(x, int, bool) AND
  replaceClass(x, Foo, Boo)
```

Step 3: Evaluate and Select Rules

- Greedily select **a small subset of candidate rules** that explain **a large number of matches**.
- In each iteration
  - evaluate all candidate rules
  - select a ***valid*** rule with the most number of matches
  - exclude the matched methods from the set of remaining unmatched methods
- Repeat until no rule can find any additional matches.

Finding Exceptions

a rule is ***valid*** if # exceptions < $\epsilon \times |\text{scope}|$

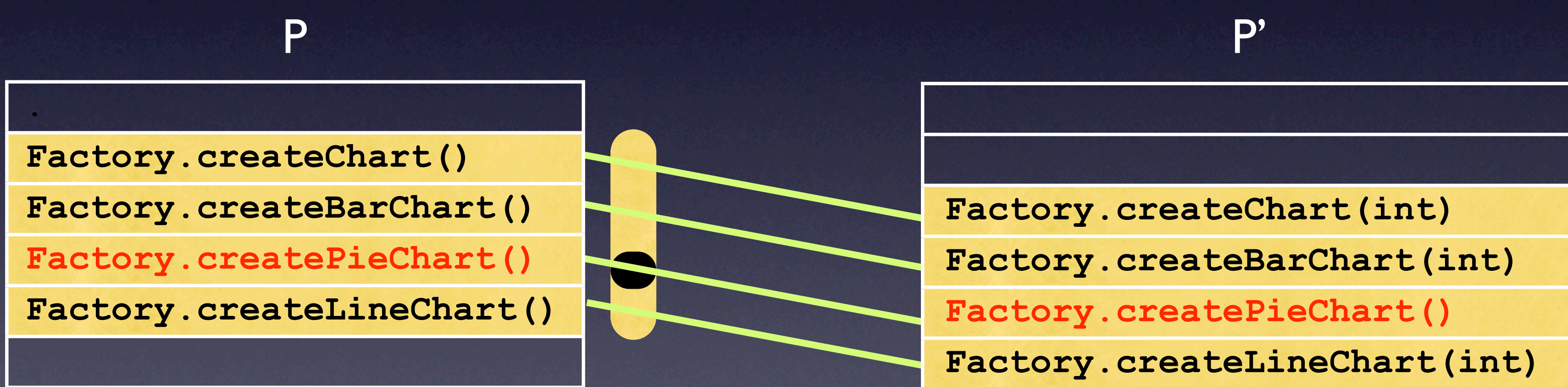
P
`Factory.createChart()`
`Factory.createBarChart()`
`Factory.createPieChart()`
`Factory.createLineChart()`

P'
`Factory.createChart(int)`
`Factory.createBarChart(int)`
`Factory.createPieChart()`
`Factory.createLineChart(int)`

```
For all x in Factory.create*Chart(*)
argAppend(x, [int])
```

Finding Exceptions

a rule is ***valid*** if # exceptions < $\epsilon \times |scope|$




```
For all x in Factory.create*Chart(*)
except {Factory.createPieChart}
argAppend(x, [int])
3 matches 1 exceptions
```

Optimizations

- We ***create*** and ***evaluate*** rules ***on demand***.

  1. Candidate rules have subsumption structure. e.g. *.*.*(*Axis) $\subset$ *.*.*(*)

  2. The nature of greedy algorithm

- Running time: a few seconds (usual check-ins), average 7 minutes (releases)

Outline

- ✓ background
- ✓ our rule-based matching approach
- ✓ inference algorithm
- **evaluation**
- potential applications of change rules

Quantitative Evaluation

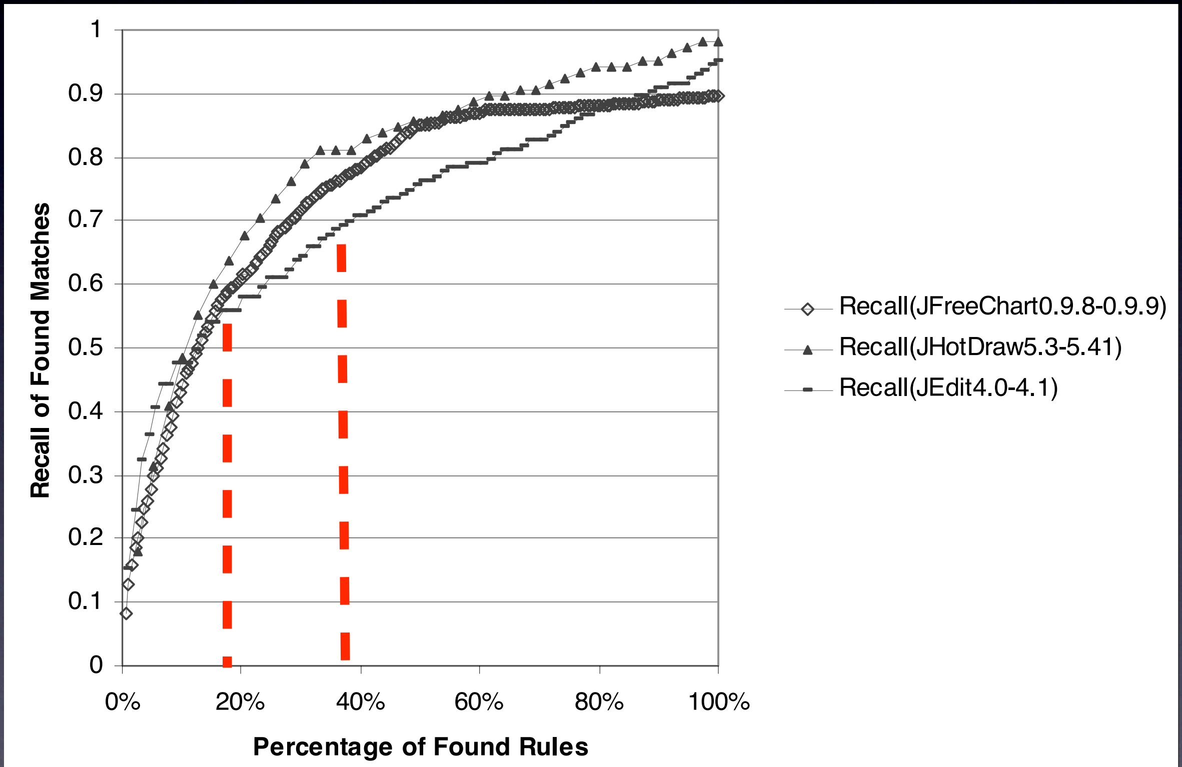
- Precision
- Recall
- Conciseness = |Matches| / |Rules| *(M/R Ratio)*

- *We created evaluation data sets by manually inspecting our results combined with the results from other tools.*

Rule-based Matching Results for Three Release Archives

	JFreeChart	jHotDraw	jEdit
	(17 release pairs)	(4 release pairs)	(4 release pairs)
Precision Median (Min ~ Max)	94% (78~100%)	99% (82~100%)	93% (87~95%)
Recall Median (Min ~ Max)	93% (70~100%)	99% (92~100%)	98% (95~100%)
M/R ratio Median (Min ~ Max)	3.50 (1.20~135.23)	2.54 (1.00~244.26)	1.73 (1.23~2.39)

Rule-based Matching Results for Three Release Archives



Top 20% of the rules find over 55% of the matches.
Top 40% of the rules find over 70% of the matches.

Comparison with Three Existing Tools

- UMLDiff [Xing and Stroulia 05]
- Refactoring Reconstruction [Weißgerber and Diehl 06]
- Automatic Renaming Identification [S. Kim, Pan, and Whitehead 05]

Comparison: Recall & Precision

	programs	Other's Recall	Our Recall	Other's Prec.	Our Prec.
[XS05]	jfreechart 18 releases	92%	**98%**	99%	**97%**
[WD06]	jEdit 2715 check-ins	72%	**96%**	93%	**98%**
	Tomcat 5096 check-ins	82%	**89%**	89%	**93%**
[KPW05]	jEdit 1189 check-ins	70%	**96%**	98%	**96%**
	ArgoUML 4683 check-ins	82%	**95%**	98%	**94%**

Comparison: Recall & Precision

	programs	Other's Recall	Our Recall	Other's Prec.	Our Prec.
[XS05]	jfreechart 18 releases	92%	**98%**	99%	**97%**
[WD06]	2 [illegible]	[illegible]	[illegible]	[illegible]	**98%**
	5 [illegible]	[illegible]	[illegible]	[illegible]	**93%**
[KPW05]	1 [illegible]	[illegible]	[illegible]	[illegible]	**96%**
	ArgoUML 4683 check-ins	82%	**95%**	98%	**94%**

6-26% higher recall with roughly the same precision

Comparison: Conciseness

	programs	Other's Results	Our Results	Our Improvement
[XS05]	jfreechart 18 releases	4004 refactorings	939 rules	77% decrease in size
[WD06]	jEdit 2715 check-ins	1218 refactorings	906 rules	26% decrease in size
	Tomcat 5096 check-ins	2700 refactorings	1033 rules	62% decrease in size
[KPW05]	jEdit 1189 check-ins	1430 matches	1119 rules	22% decrease in size
	ArgoUML 4683 check-ins	3819 matches	2127 rules	44% decrease in size

Comparison: Conciseness

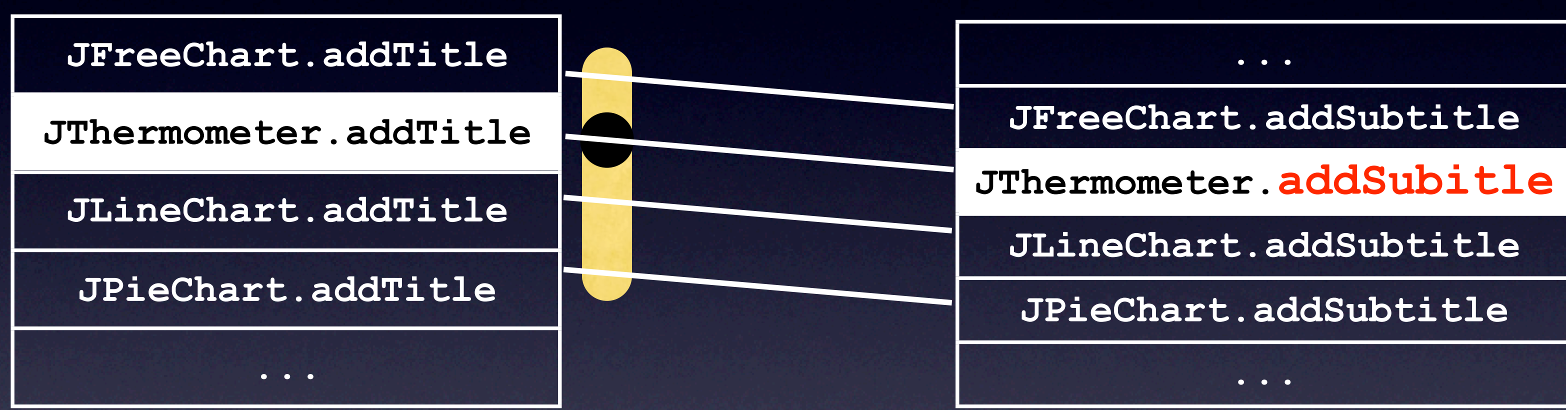
	programs	Other's Results	Our Results	Our Improvement
[XS05]	jfreechart 18 releases	4004 refactorings	939 rules	77% decrease in size
[WD06]	2			% decrease in size
	5			% decrease in size
[KPW05]	I			% decrease in size
	ArgoUML 4683 check-ins	3819 matches	2127 rules	44% decrease in size

22-77% reduction in the size of matching results

Outline

- ✓ background
- ✓ our rule-based matching approach
- ✓ inference algorithm
- ✓ evaluation
- **potential applications of change rules**
  - bug finding, documentation assistant, API catch up, API evolution analysis, etc.

Potential App: Bug Finding Tool




```
for all x in J*.addTitle(Title)
except {JThermometer.addTitle(Title)}
  procedureReplace(x, addTitle, addSubtitle)
```

Dynamic dispatching of `JFreeChart.addSubtitle` **does not work properly.**

Conclusions

- ***Matching*** is a basis for a variety of software engineering research & tools.
- Our approach is the first to ***automatically infer*** structural changes and ***concisely represent*** them as a set of change rules.
- Our tool find matches with **high precision and recall**.

Acknowledgment

David Notkin
University of Washington

Dan Grossman
University of Washington

Zhenchang Xing
Eleni Stroulia
University of Alberta

Peter Weißgerber
Stephan Diehl
University of Trier

Sunghun Kim,
Jim Whitehead
University of California,
Santa Cruz