A General Path-Based Representation for Predicting Program Properties

Uri Alon, Meital Zilberstein, Omer Levy, Eran Yahav

University of Washington

SEVENTH FRAMEWORK PROGRAMME

Technion

Motivating Example #1

Prediction of Variable Names in Python

```python
def sh3(c):
    p = Popen(c, stdout=PIPE,
          stderr=PIPE, shell=True)
    o, e = p.communicate()
    r = p.returncode
    if r:
        raise CalledProcessError(r, c)
    else:
        return o.rstrip(), e.rstrip()
```

```python
def sh3(cmd):
    process = Popen(cmd, stdout=PIPE,
          stderr=PIPE, shell=True)
    out, err = process.communicate()
    retcode = process.returncode
    if retcode:
        raise CalledProcessError(retcode, cmd)
    else:
        return out.rstrip(), err.rstrip()
```

Motivating Example #2

Prediction of Method Names in JavaScript

```
function _______(object) {
    if (!object)
        return object;
    var clone = {};
    for (var key in object) {
        clone[key] = object[key];
    }
    return clone;
}
```

```
function cloneObject(object) {
    if (!object)
        return object;
    var clone = {};
    for (var key in object) {
        clone[key] = object[key];
    }
    return clone;
}
```

Motivating Example #3

Prediction of full types in Java

StackOverflow answer:

```
Configuration conf = HBaseConfiguration.create();
try {
    Connection connection = ConnectionFactory.createConnection(conf);
}
```

`com.mysql.jdbc.Connection` ?
`org.apache.http.Connection` ?

```
import org.apache.hadoop.hbase.client.Connection;
```

Previously – separate techniques for each problem / language

	Java	JavaScript	Python	C#	…
Variable name prediction	Bichsel et al. CCS'2016 (CRFs)	Raychev et al. POPL'2015 (CRFs)	Raychev et al. OOPSLA'2016 (Decision Trees)	..	..
Method name prediction	Allamanis et al. ICML'2016 (NNs)	Raychev et al. OOPSLA'2016 (Decision Trees)		..	..
Full types prediction				..	..
…	Raychev et al. PLDI'2014 (n-grams+RNNs)	Bielik et al. ICML'2016 (PHOG)	Raychev et al. OOPSLA'2016 (Decision Trees)	Allamanis et al. ICML'2015 (Generative)	..

Completely automatically!

How to represent a program element?

- Should work for many programming languages
- Should work for different tasks
- Useful in multiple learning algorithms

```
while (!done) {
    if (someCondition()) {
        done = true;
    }
}
```

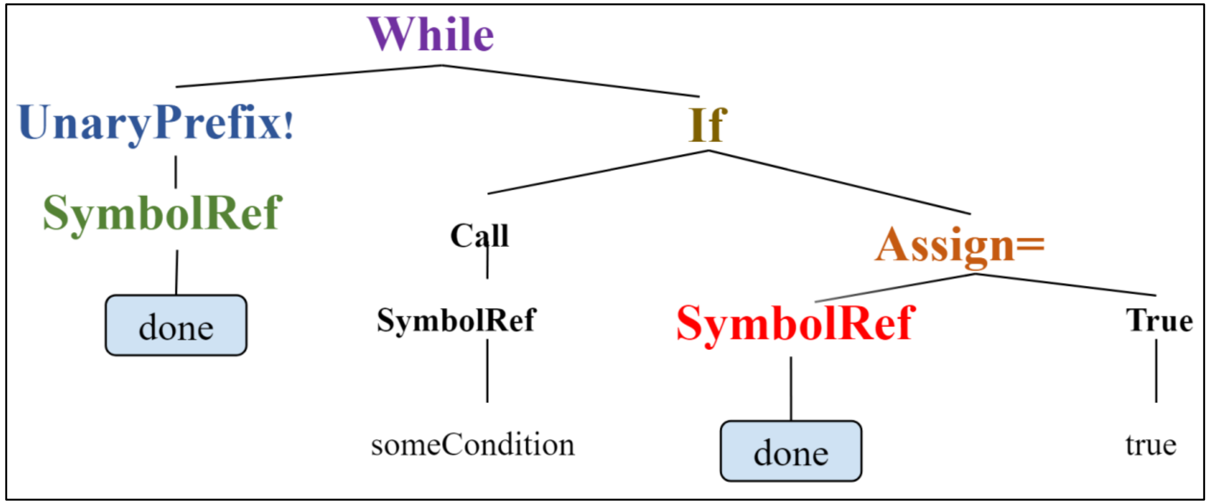
```
while (!count) {
    if (someCondition()) {
        count = true;
    }
}
```

- What are the properties that make “done” a “done”?

How to represent a program element?

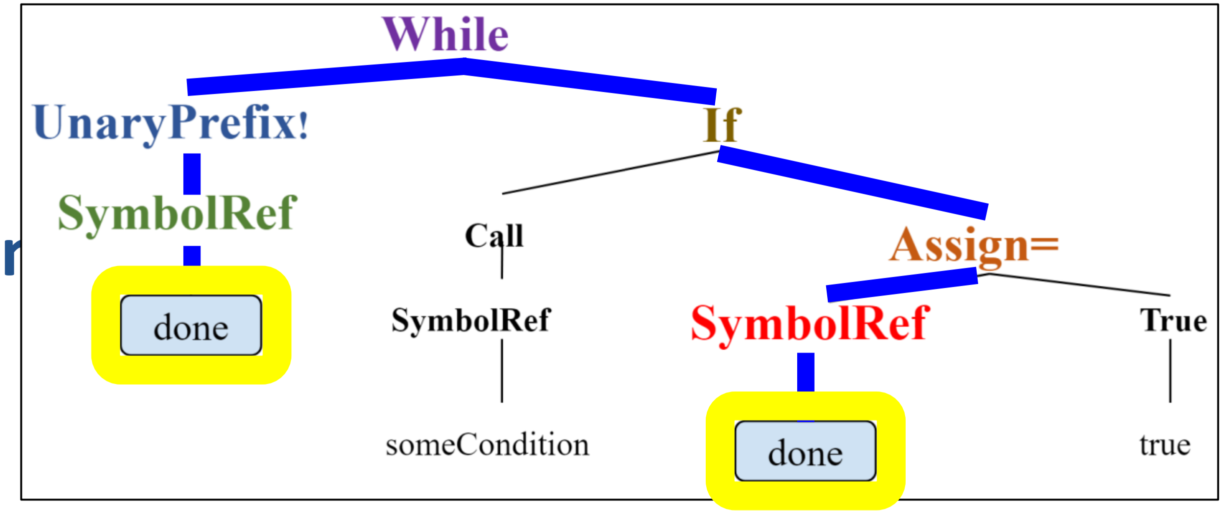
Key idea:

```
while (!done) {
    if (someCondition()) {
        done = true;
    }
}
```



- The semantic role of a program element is the set of all **structured contexts** in which it appears
- “done” is “done” because it appears in particular structured contexts

```
while (!done) {
    if (someCondition()) {
        done = true;
    }
}
```



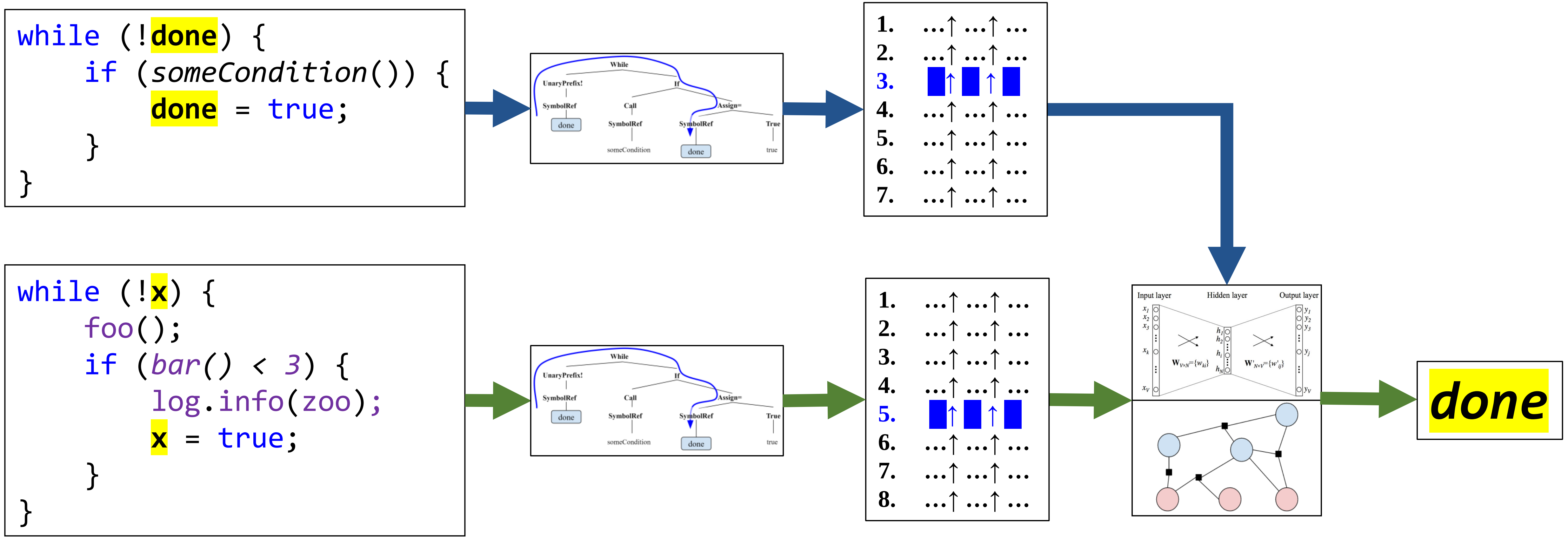
For example:

(SymbolRef ↑

done is represented as the set of all its paths

Example training & testing pipeline

Training



Testing

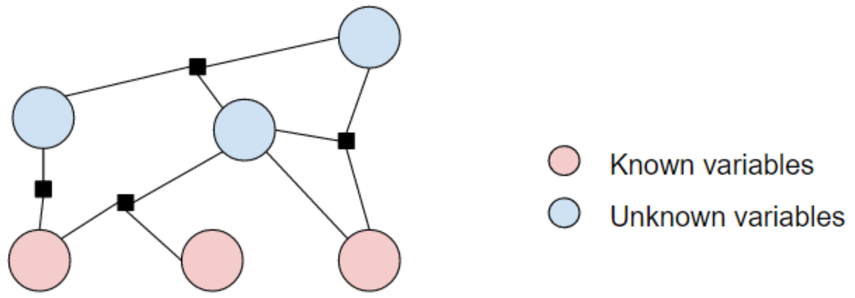
Advantages of AST-Paths representation

- ✓ Expressive enough to capture any property that is expressed syntactically.
- ✓ Independent of the programming language
- ✓ Automatically extractable – only requires a parser
- ✓ Not bound to the learning algorithm
- ✓ Works for different tasks

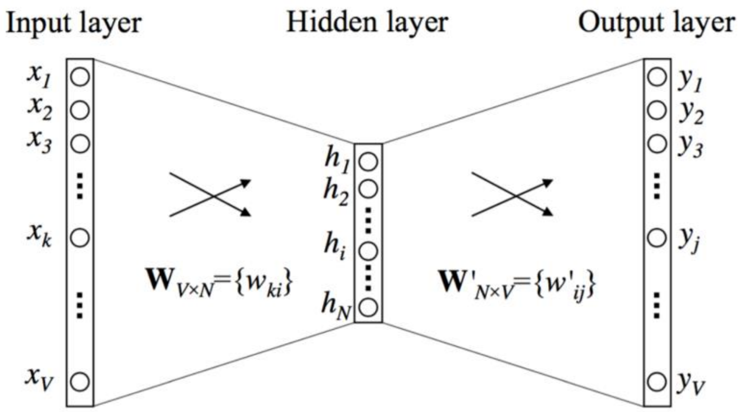
Predicting program properties with AST paths

- Off-the shelf algorithms
- Plug-in our representation

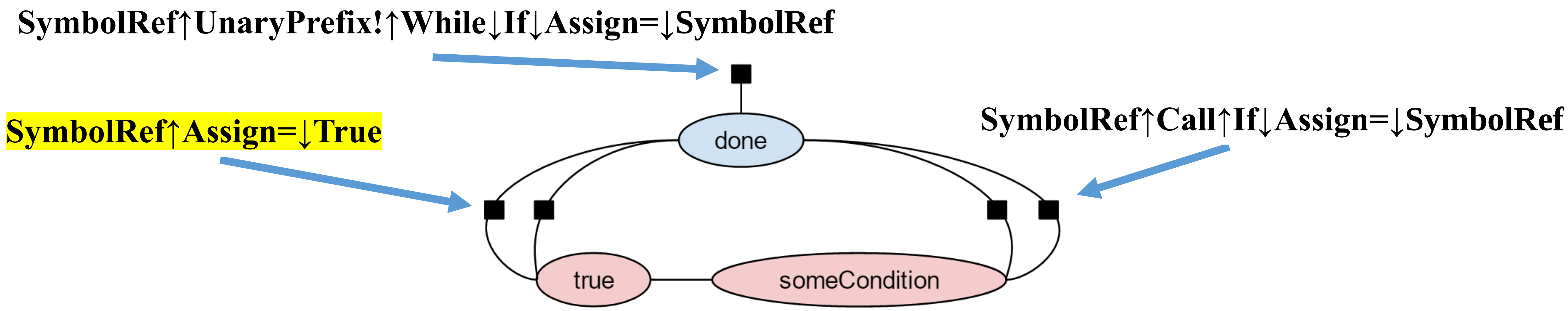
Conditional Random Fields (CRFs)



word2vec-based



Predicting properties with CRFs

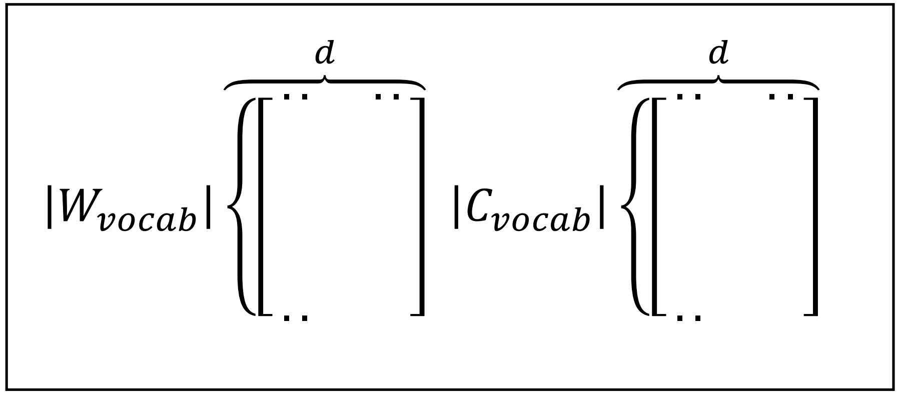


- Nodes: program elements
- Factors: learned scoring functions:
  - $(Values, Values, Paths) \rightarrow \mathbb{R}$
- The same as in (JSNice, Raychev et al., POPL'2015), but with our paths as factors

Predicting properties with word2vec

- Input: pairs of: $(word, context)$

- Model:
  - word vectors: W_{vocab}
  - context vectors: C_{vocab}



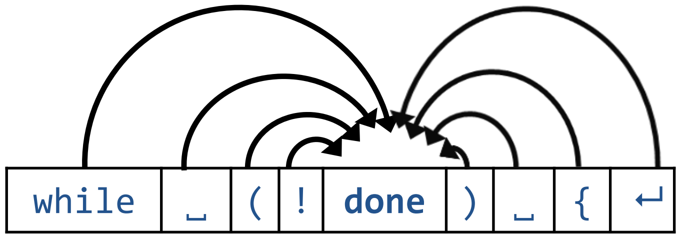
- Prediction:
  - $\text{predict}(\{\overrightarrow{c_1}, \dots, \overrightarrow{c_n}\}) = argmax_{w_i \in W_{vocab}} \left[\overrightarrow{w_i} \cdot \sum_j \overrightarrow{c_j}\right]$

Word2vec and different contexts

- Input: pairs of: $(word, context)$
- Train word2vec with 3 types of contexts:
  - Neighbor tokens
  - Surrounding AST-nodes
  - AST paths

Word2vec and different contexts

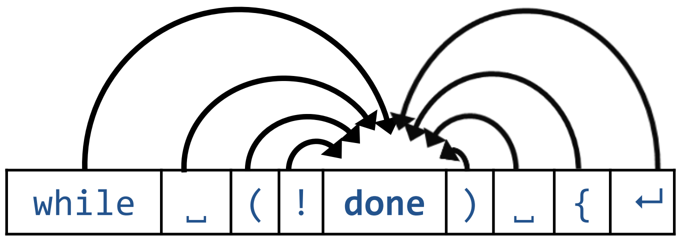
- Input: pairs of: $(word, context)$
- Train word2vec with 3 types of contexts:

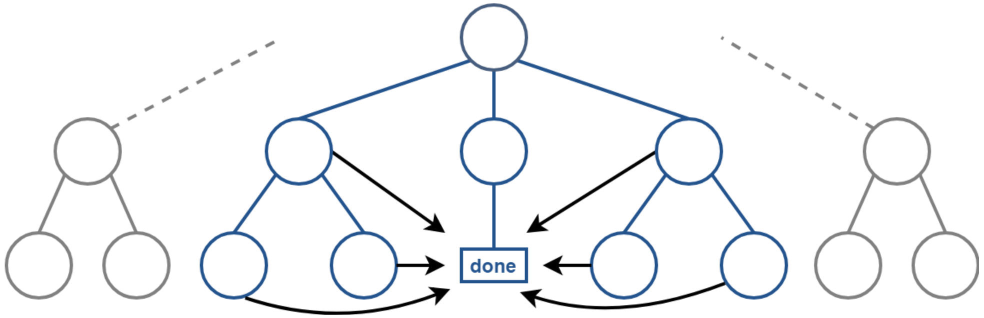


  - **Neighbor tokens**
  - Surrounding AST-nodes
  - AST paths

Word2vec and different contexts

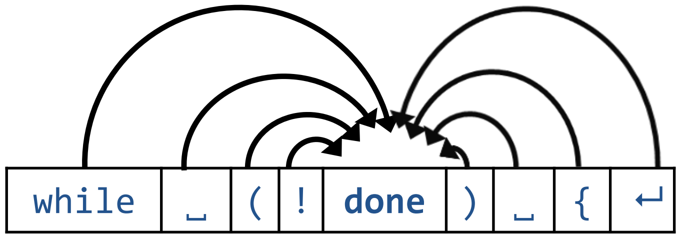
- Input: pairs of: $(word, context)$
- Train word2vec with 3 types of contexts:
  - Neighbor tokens
  - **Surrounding AST-nodes**
  - AST paths



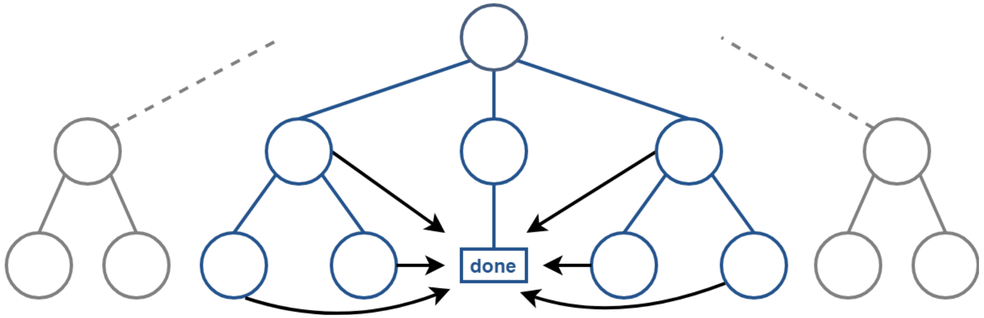


Word2vec and different contexts

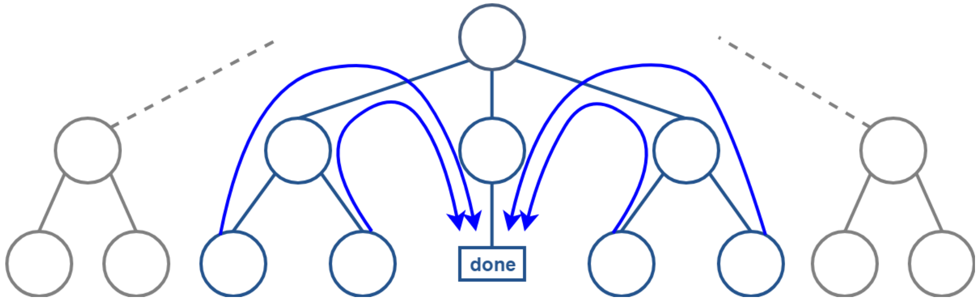
- Input: pairs of: $(word, context)$
- Train word2vec with 3 types of contexts:



  - Neighbor tokens



  - Surrounding AST-nodes

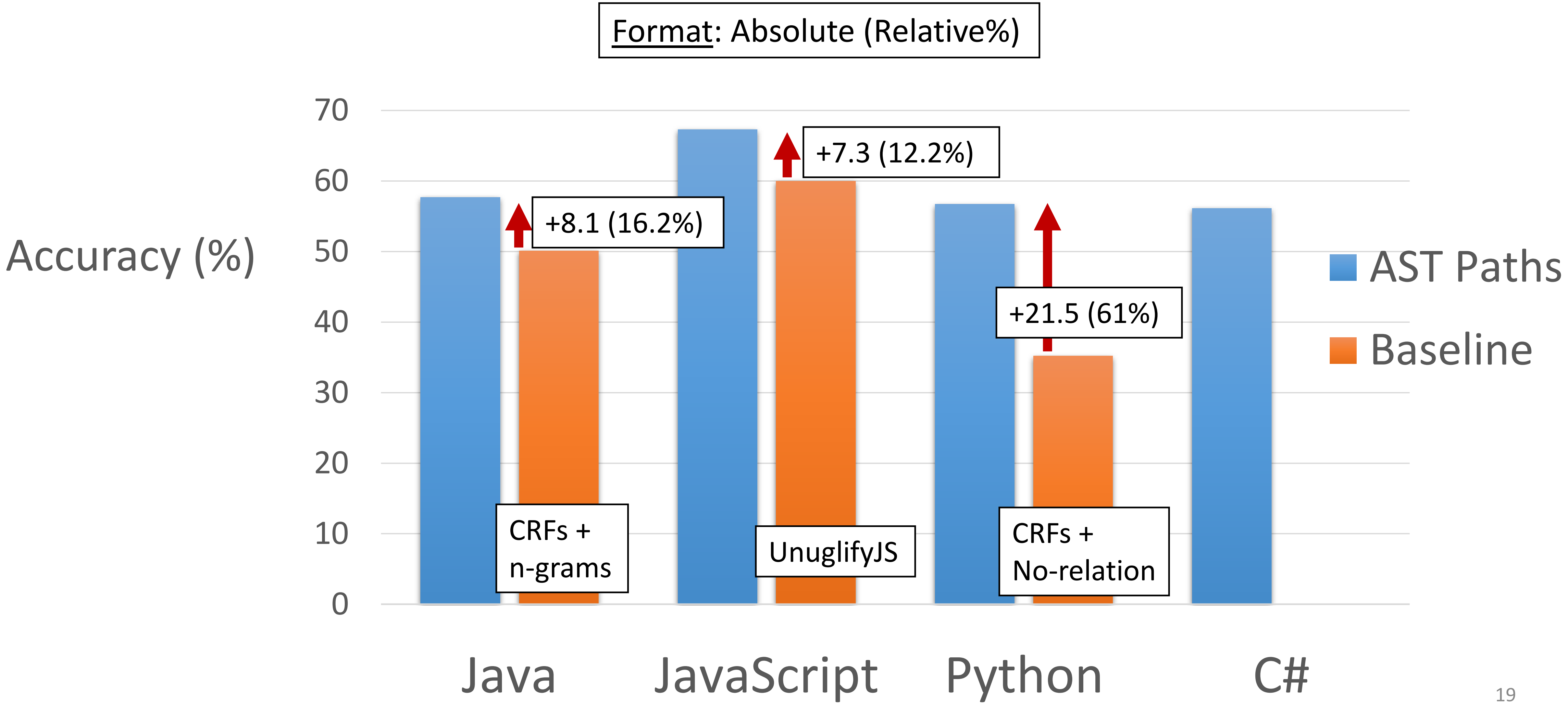


  - **AST paths**

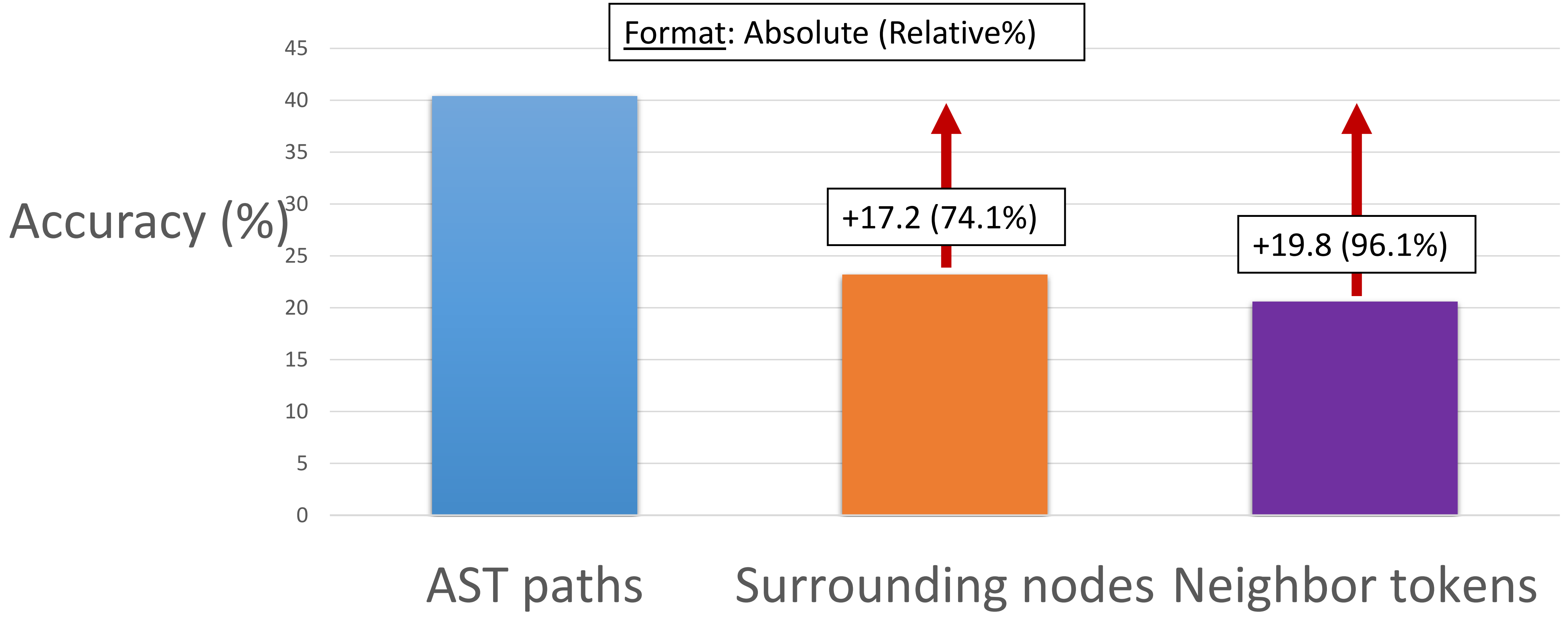
Evaluation

- 4 programming languages
  - Java, JavaScript, Python, C#
- 3 tasks
  - predicting method names, variable names, full types ("...hbase.client.Connection")
- 2 learning algorithms
  - CRFs, word2vec-based

Predicting variable names with CRFs



Word2vec with different context types

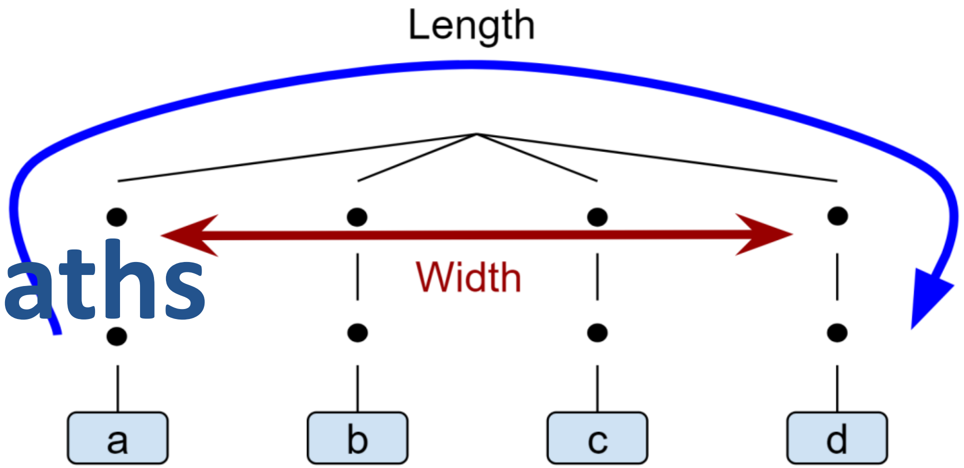


Task: Variable names, word2vec, JavaScript

Reducing the number of paths

- Limiting path-length and path-width
  - Path vocabulary size (JavaScript):

$$\mathit{length}: 7 \rightarrow 6: \quad 13M \rightarrow 11M$$
$$\mathit{width}: 3 \rightarrow 2: \quad 13M \rightarrow 12M$$



- Path abstraction
  - Path vocabulary size (Java):

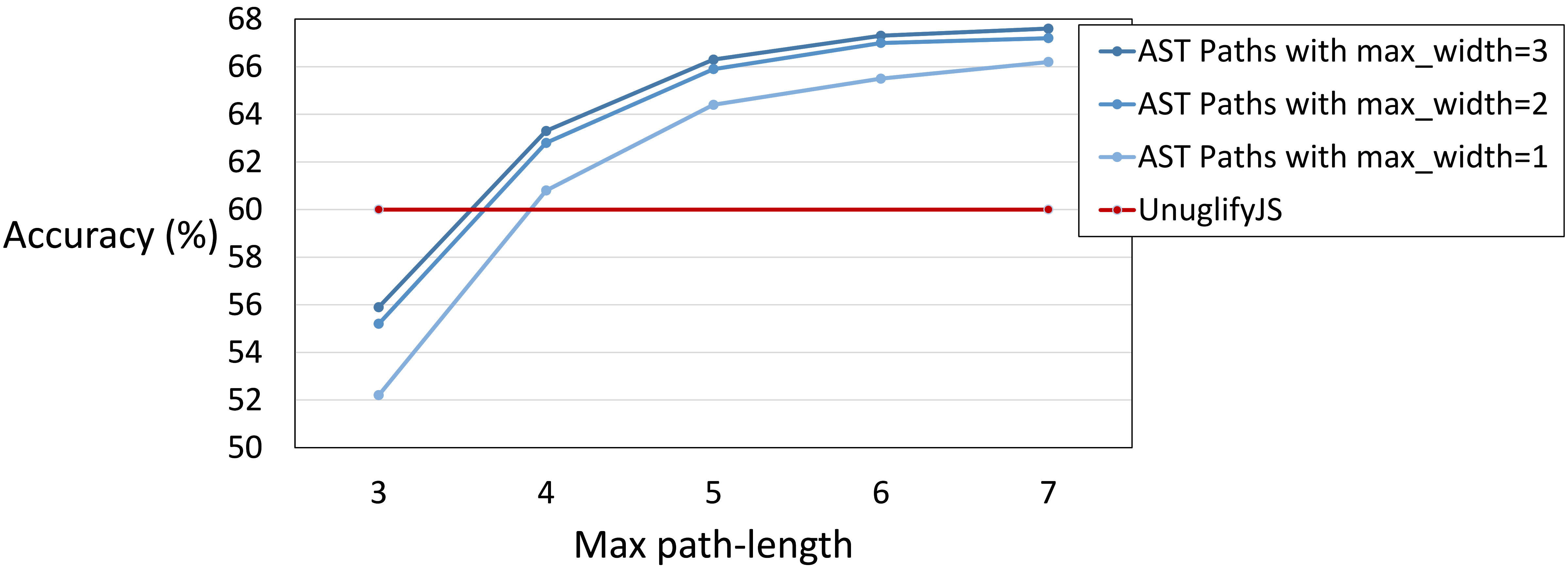
$$\sim 10^7 \rightarrow \sim 10^2$$

SymbolRef ↑ UnaryPrefix! ↑ While ↓ If ↓ Assign= ↓ SymbolRef

…↑ While ↓…

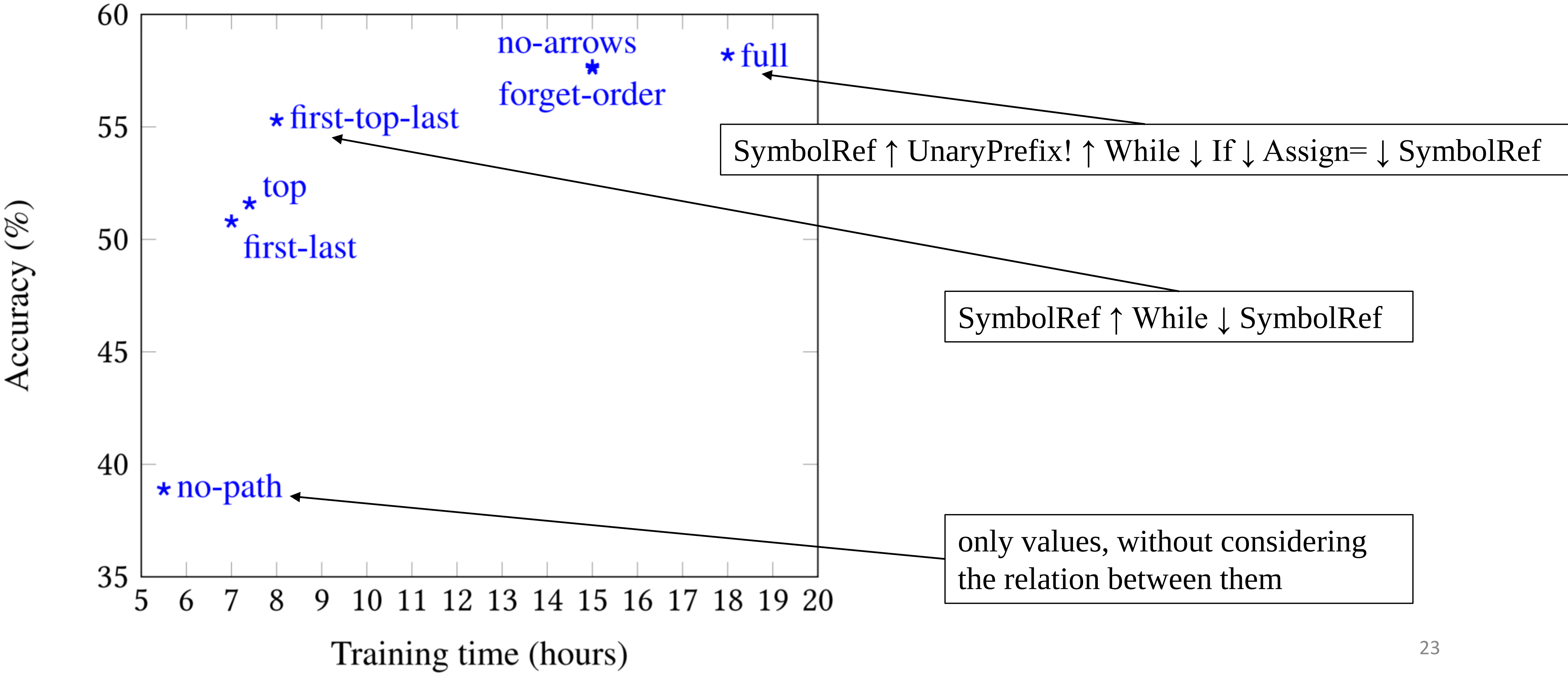
Effect of limiting path length and width

Task: Variable names, CRFs, JavaScript



AST Path Abstractions

Task: Variable names, CRFs, Java



Example (JavaScript)

```
function countSomething(x, t) {
    var c = 0;
    for (var i = 0, l = x.length; i < l ; i++) {
        if (x[i] === t) {
            c++;
        }
    }
    return c;
}
```

Example (JavaScript)

```
function countSomething(array, target) {
    var count = 0;
    for (var i = 0, l = array.length; i < l ; i++) {
        if (array[i] === target) {
            count++
        }
    }
    return count;
}
```

Example (Java)

```
public String sendGetRequest(String l) {
    HttpClient c =  HttpClientBuilder.create().build();
    HttpGet r = new HttpGet(l);
    String u = USER_AGENT;
    r.addHeader("User-Agent", u);
    HttpResponse s = c.execute(r);
    HttpEntity t = s.getEntity();
    String g = EntityUtils.toString(t, "UTF-8");
    return g;
}
```

Example (Java)

```java
public String sendGetRequest(String url) {
    HttpClient client =  HttpClientBuilder.create().build();
    HttpGet request = new HttpGet(url);
    String user = USER_AGENT;
    request.addHeader("User-Agent", user);
    HttpResponse response = client.execute(request);
    HttpEntity entity = response.getEntity();
    String result = EntityUtils.toString(entity, "UTF-8");
    return result;
}
```

Semantic Similarity Between Names

CRFs

```
var d = false;
while (!d) {
    doSomething();
    if (someCondition()) {
        d = true;
    }
}
```

	Candidate
1.	done
2.	ended
3.	complete
4.	found
5.	finished
6.	stop
7.	end
8.	success

More Semantic Similarities

Similarities
req ~ request
count ~ counter ~ total
element ~ elem ~ el
array ~ arr ~ ary ~ list
res ~ result ~ ret
i ~ j ~ index

Summary: a trade-off between learning effort and generalizability

- Surface text – too noisy
- Complex analyses are great, but specific to language and task
- AST paths – sweet spot of simplicity, expressivity and generalizability
- “Structural n-grams”
- A strong baseline for any machine learning for code task

Questions?