Mastery: Shifted-Code-Aware Structured Merging

Fengmin Zhu, *Xingyu Xie*, Dongyu Feng, Na Meng, Fei He

School of Software, Tsinghua University

Key Laboratory for Information System Security, MoE, China

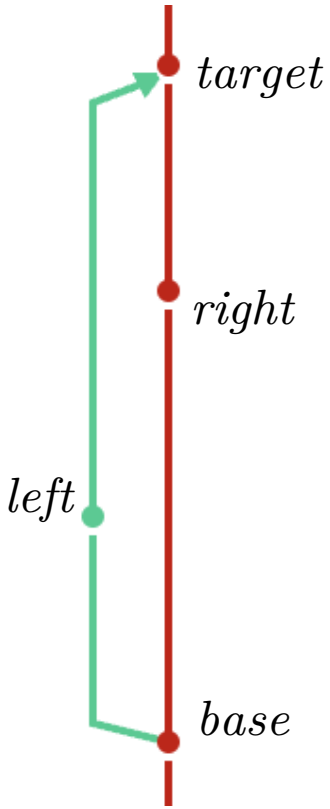
Beijing National Research Center for Information Science and Technology

Max Planck Institute for Software Systems

Virginia Tech

Oct. 28, 2022

Three-way merging

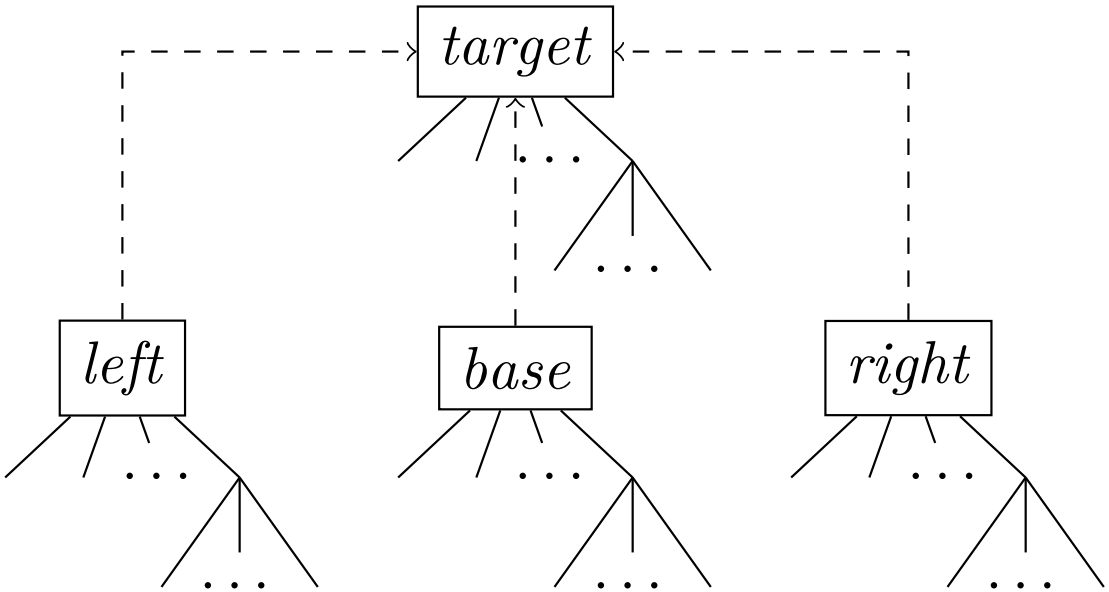


Three-way merging: integrate consistent changes introduced by the two branches.

Unstructured merging:

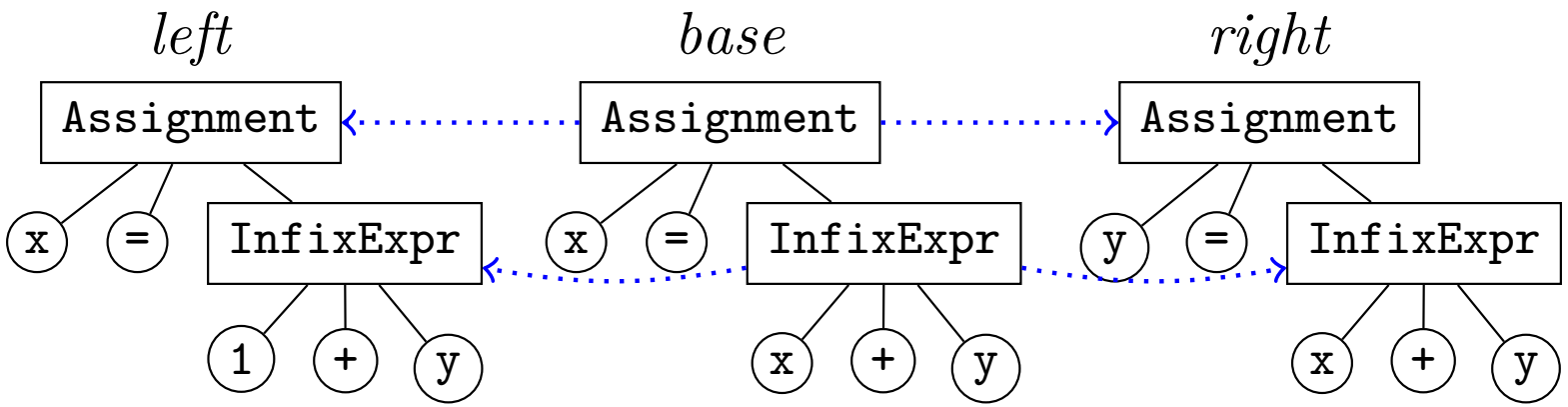
- broadly used in practice;
- recognizes source programs as lines;
- merge accuracy is unsatisfying.

Structured merging



Structured merging recognizes source code as AST (abstract syntax tree), which has higher merge accuracy than unstructured merging.

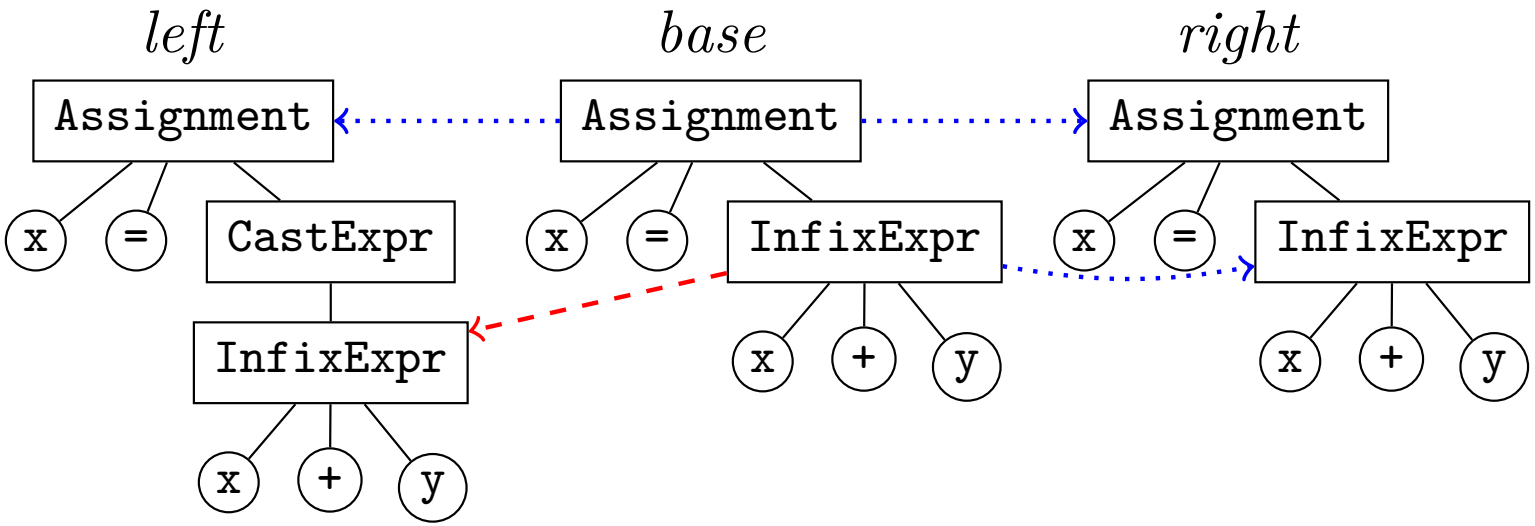
AST matching



AST matching: When manipulating *base* AST into some variant, the matched two nodes (e.g. `Assignments`) are recognized as the same one.

Shifted code

```
x = (int)(x + y) // left        x = x + y // base
```



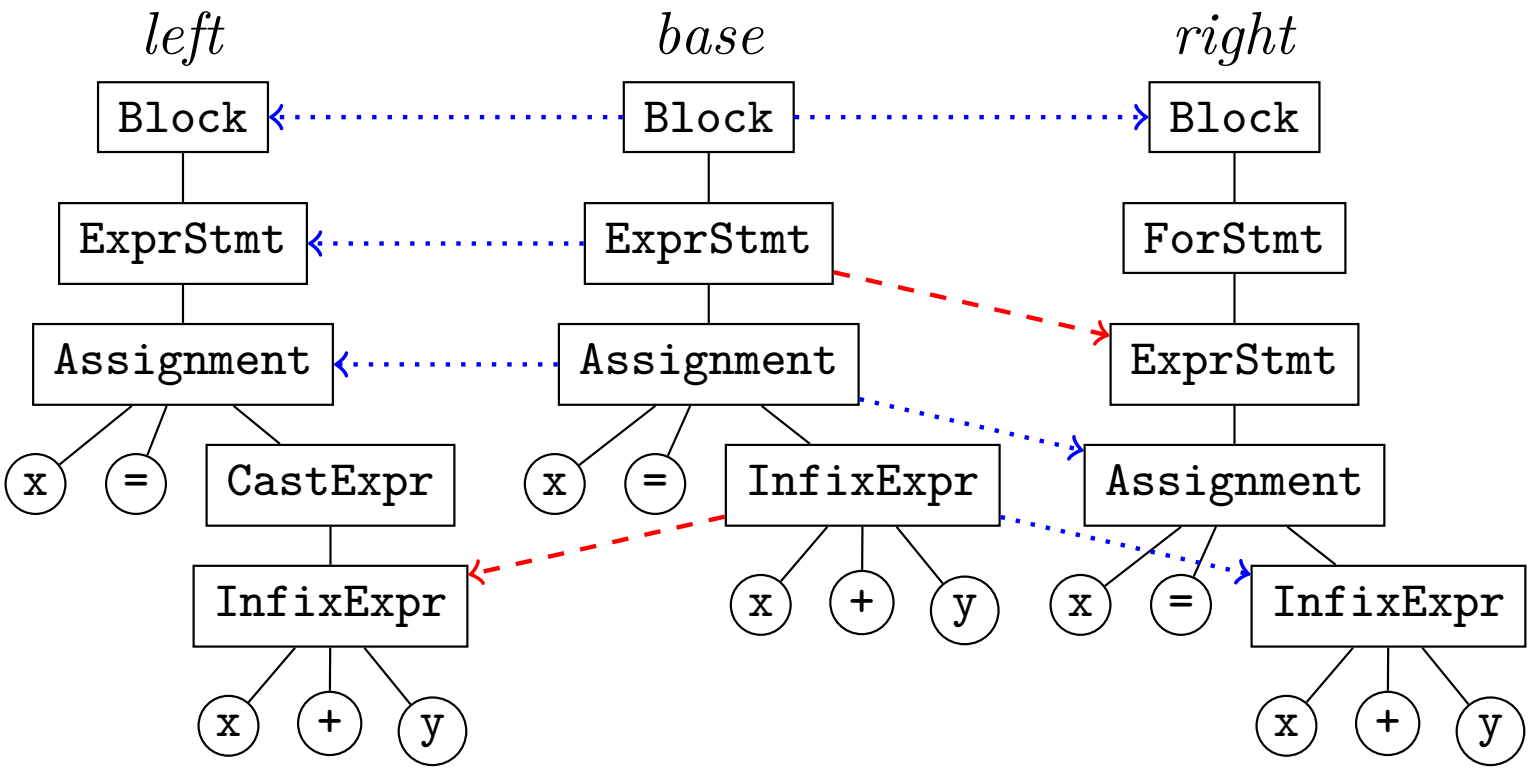
Definition: Given two mappings (u', v') and (u, v), if u is a child of u', whereas v is not a child but a later descendant of v', then the code fragment corresponding to v is called a **shifted code**.

Challenge: Existing structured merging algorithms visits nodes in a top-down way, so they **cannot handle shifted code**.

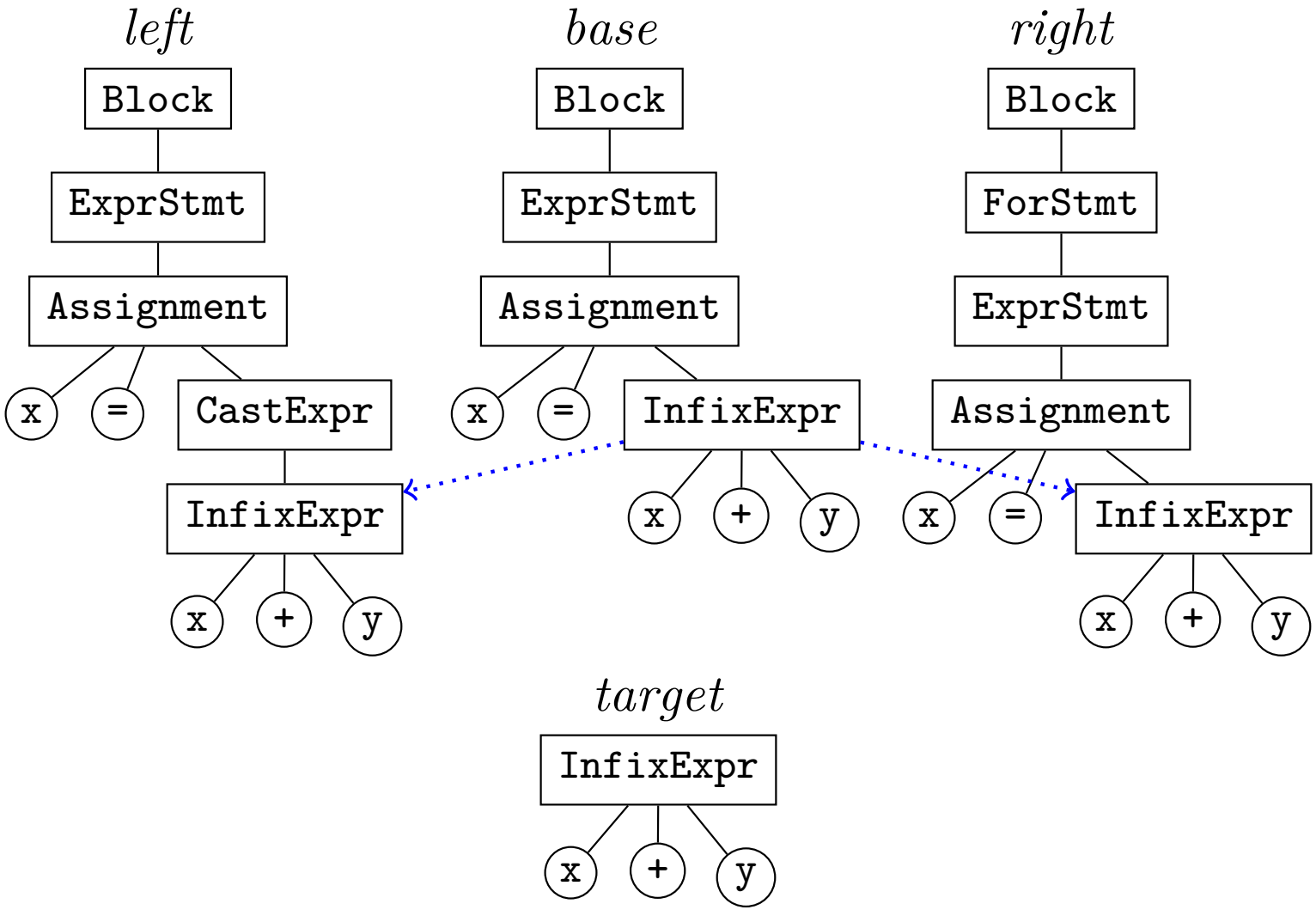
Our idea

Problem: how to merge shifted code?
Method: introduce a **bottom-up** merging algorithm.

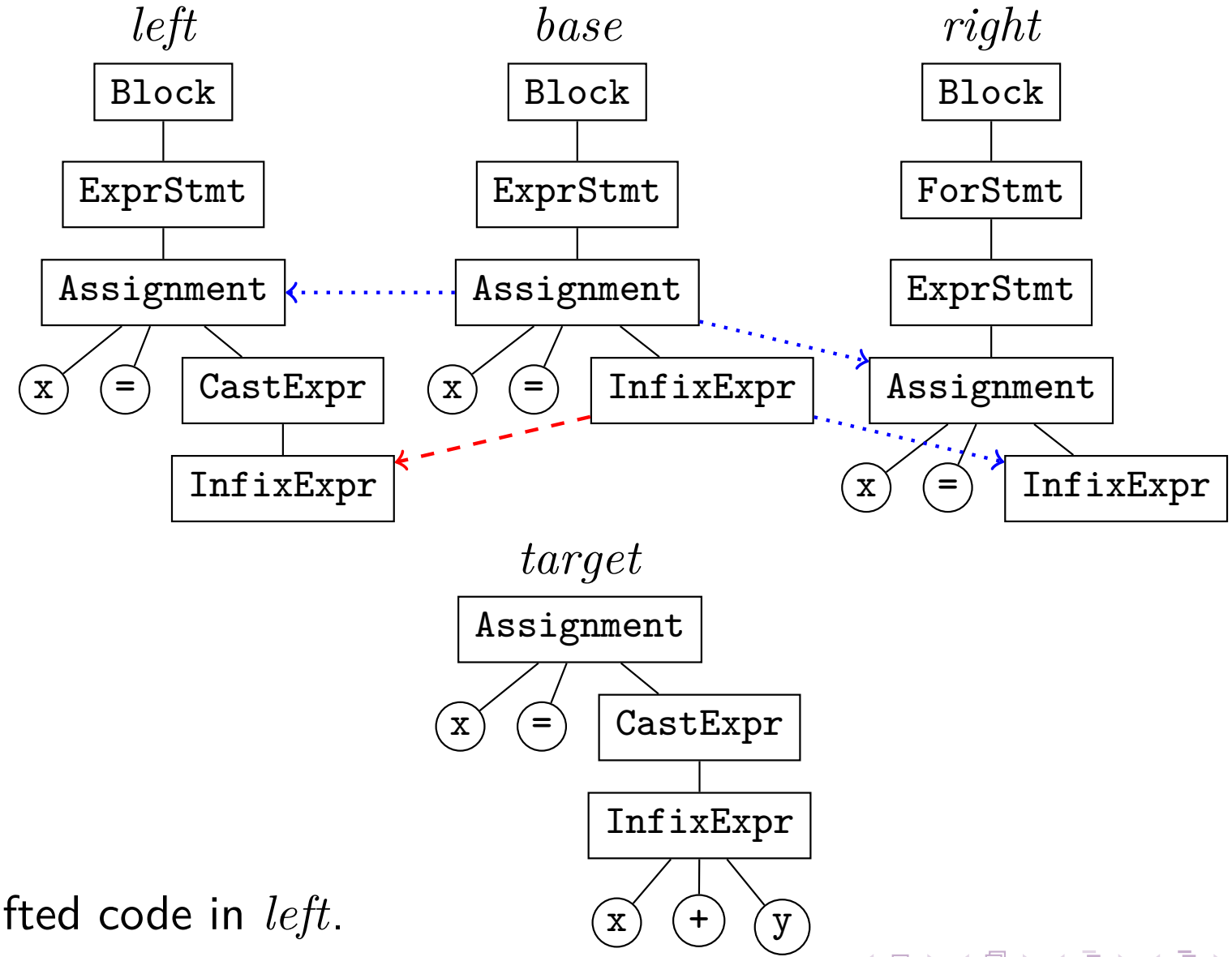
A motivating example



Four merge scenarios of inner nodes.



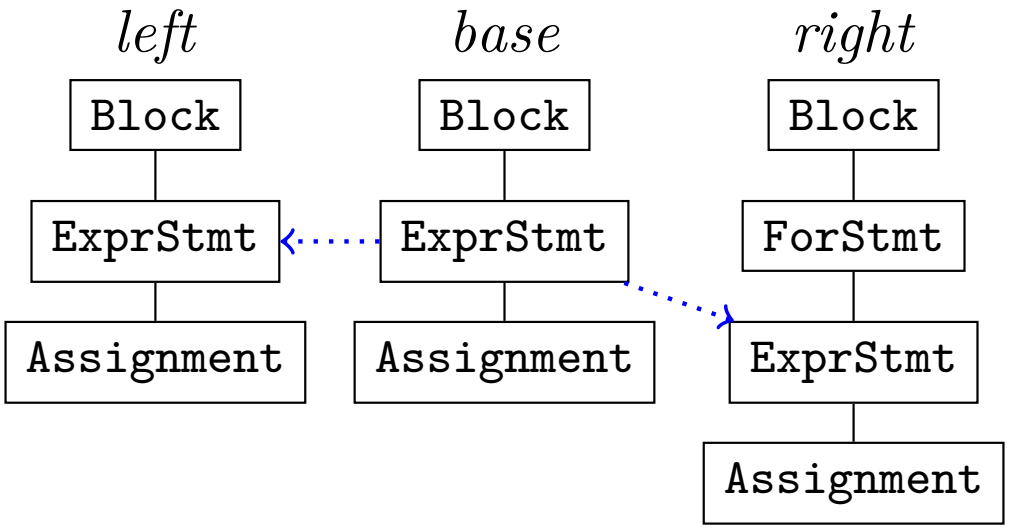
Three subtrees of `InfixExpr` are equal, so their merging is trivial.

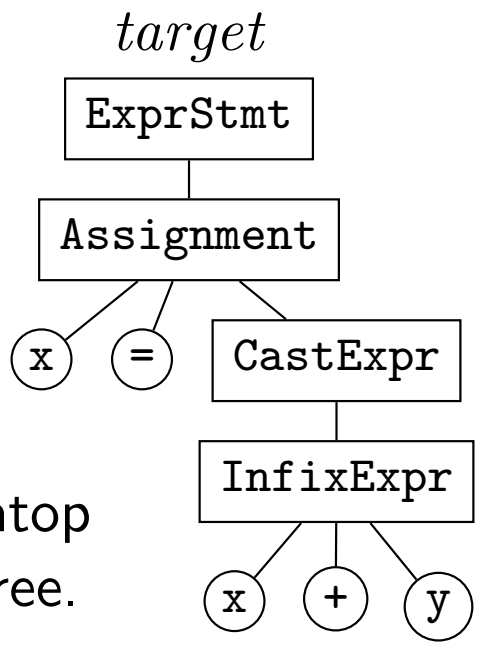


Shifted code in *left*.

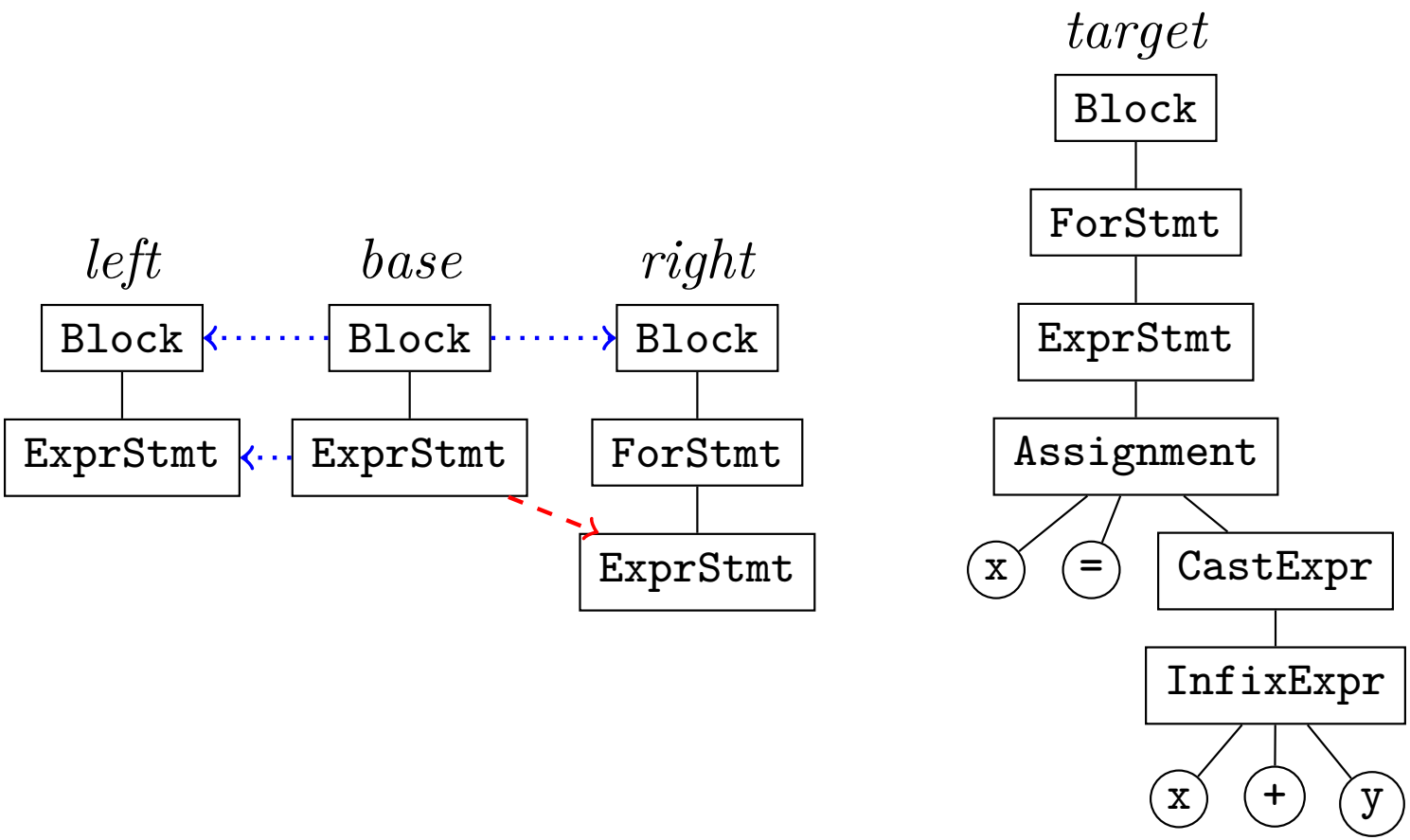
Relevant

If there exsists a descendant w of v such that u matches w, we say that u is **relevant** to v.





Just put an ExprStmt atop the computed merged tree.



Shifted code in *right*: copy the subtree of `ForStmt`, and replace its child subtree (`ExprStmt`) with computed merged result.

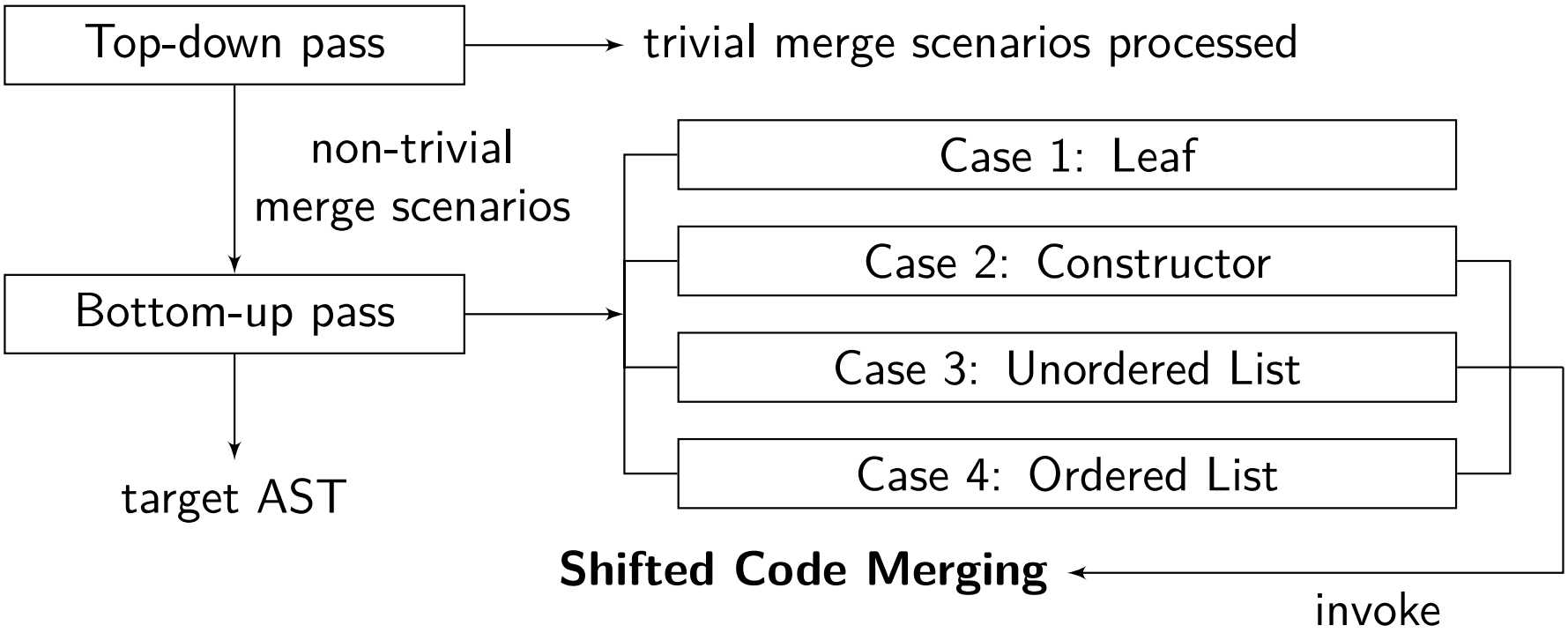
Pseudocode

Algorithm: Shifted Code Merging

1 **Function** `IssueShifted`(*b: Node, l: Node, r: Node*)**:**
2 **let** l', r' be nodes s.t. $(b, l') \in \mathcal{M}_L, (b, r') \in \mathcal{M}_R$;
3 **if** $l' = l \wedge r' = r$ **then return** $\mathcal{R}(b)$;
4 **if** $l' \neq l \wedge r' = r$ **then return** $l[\mathcal{R}(b)/l']$;
5 **if** $r' \neq r \wedge l' = l$ **then return** $r[\mathcal{R}(b)/r']$;
6 **if** $l[\mathcal{R}(b)/l'] = r[\mathcal{R}(b)/r']$ **then return** $l[\mathcal{R}(b)/l']$;
7 **return** Conflict(l, r);

b is required to be relevant to l and r.

Framework



The technical details could be found in our paper.

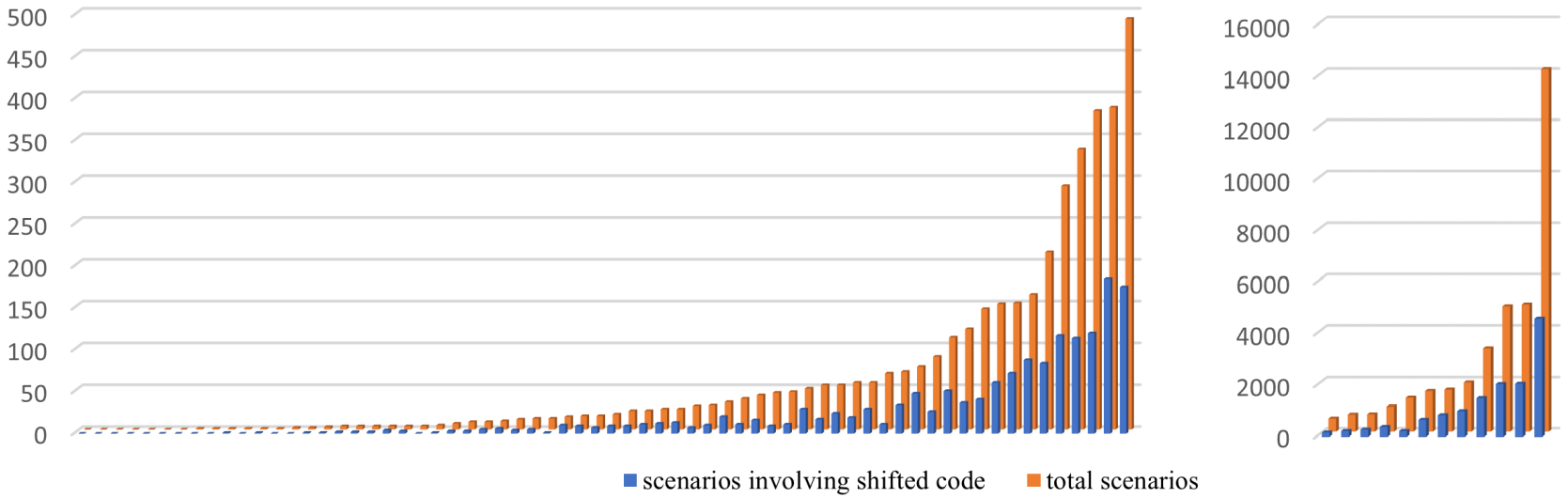
Dataset

We seek the top-100 most popular open-source Java projects on GitHub[1]; exclude any non-software-project (e.g., tutorials).
On the remaining 78 projects, we extract **40,533** merge scenarios via an analysis of their commit histories.

[1] According to the following list, until July 12, 2021:
`github.com/EvanLi/Github-Ranking/blob/master/Top100/Java.md`

Frequency of shifted code

38.54% among 40,533 merge scenarios

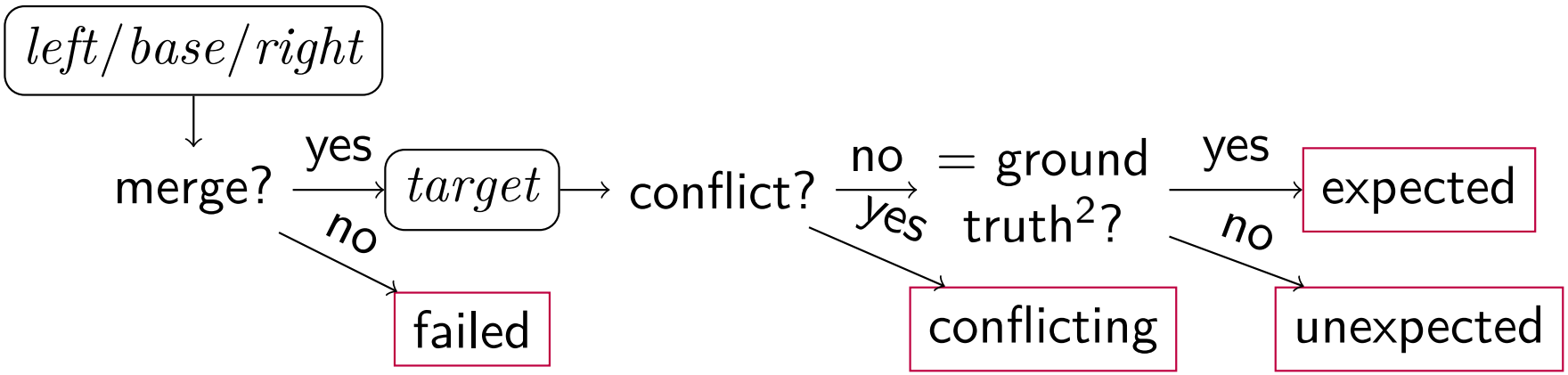


Baseline tools

We compare Mastery with four state-of-the-art merge tools:

- JDime: structured;
- jFSTMerge: semistructured, tree-based;
- IntelliMerge: semistructured, graph-based, refactoring-aware;
- GitMerge: unstructured, the default merging tool in Git.

Taxonomy of results



[2]We use code merged by developer as ground truth.

Distribution of results

Tool	Expected (Accuracy)	Unexpected	Conflicting	Failed
Mastery	**82.26%**	8.64%	9.09%	0.00%
JDime	80.89%	6.03%	11.37%	1.70%
jFSTMerge	74.17%	9.47%	16.35%	0.01%
IntelliMerge	24.11%	60.58%	**8.49%**	6.81%
GitMerge	75.60%	**1.95%**	22.45%	**0.00%**

Shifted-aware merging is effective

- Among the 1,398 scenarios where Mastery's results are expected whereas JDime's are not, 48.78% (+10.25%) involves shifted code.
- Among the 1,650 scenarios where JDime's results are conflicting whereas Mastery's are not, 51.82% (+13.28%) involves shifted code.

Runtime performance

Tool	Mastery	JDime	jFSTMerge	IntelliMerge
Sort	structured		semi-structured	
Avg Time	10.33 s	24.06 s	13.21 s	4.34 s

Mastery is 2.4x as fast as JDime, and 1.3x as fast as jFSTMerge.

Recap

Problem: how to merge scenarios containing shifted code in structured merging?
Method: merging in both top-down and **bottom-up** way.
Evaluation: improve overall efficiency and effectiveness.

Tool: `https://github.com/thufv/mastery`

Thanks!