Conflict Resolution for Structured Merge via Version Space Algebra

Presented By

Sheikh Shadab Towqir

CS 6704

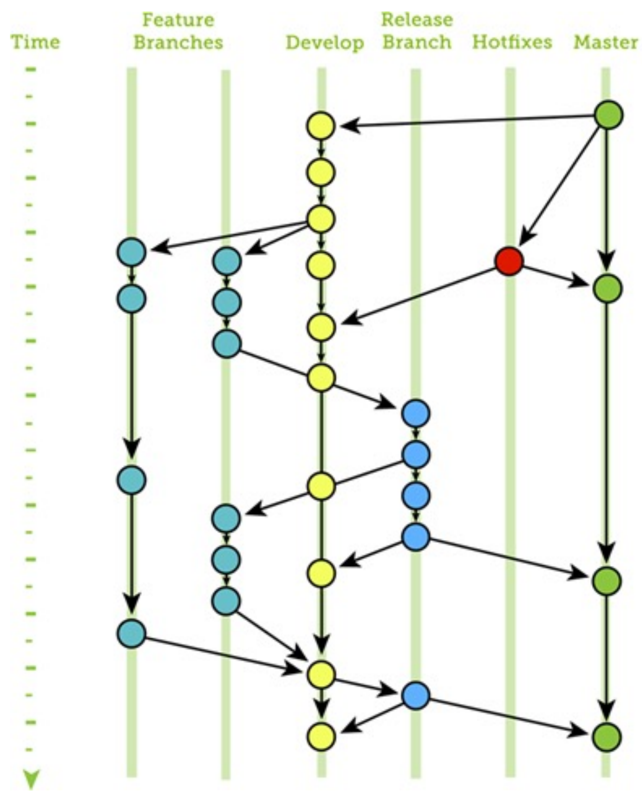
Problem Statement

- Resolving conflicts is the main challenge when merging branches of software.
- Existing merge tools usually rely on the developer to manually resolve these conflicts
- One main reason existing merge tools do not attempt to resolve conflicts because of safety.
- In the presence of conflicts, the resolution might be ambiguous, so guessing and applying a resolution is dangerous.

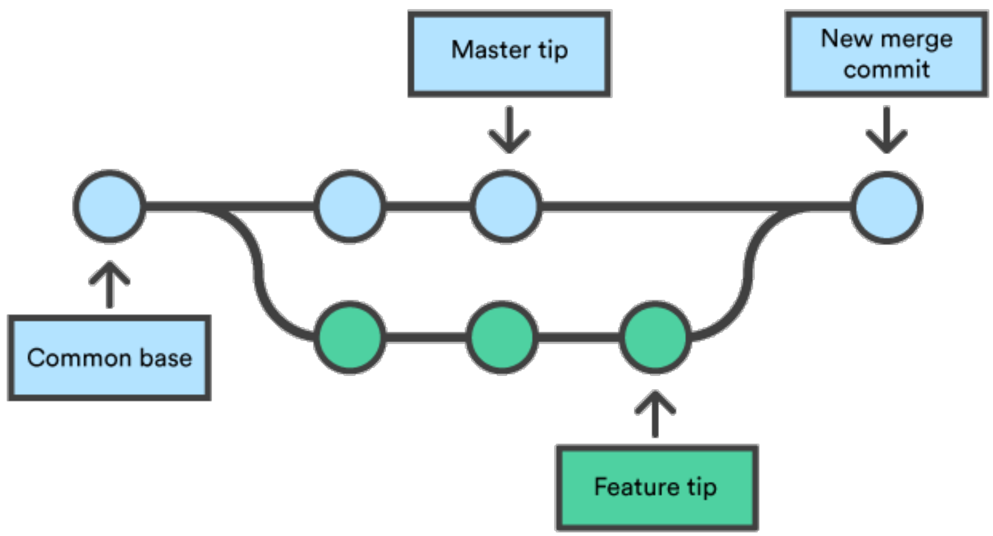
Proposed Tool

- AutoMerge
- Generate a large set of candidate programs to resolve the conflicting scenario.
- Use a simple mechanism to rank the resolutions.
- Present the top-ranked resolutions to the developer.

Software Merge



Software Merge



Software Merge

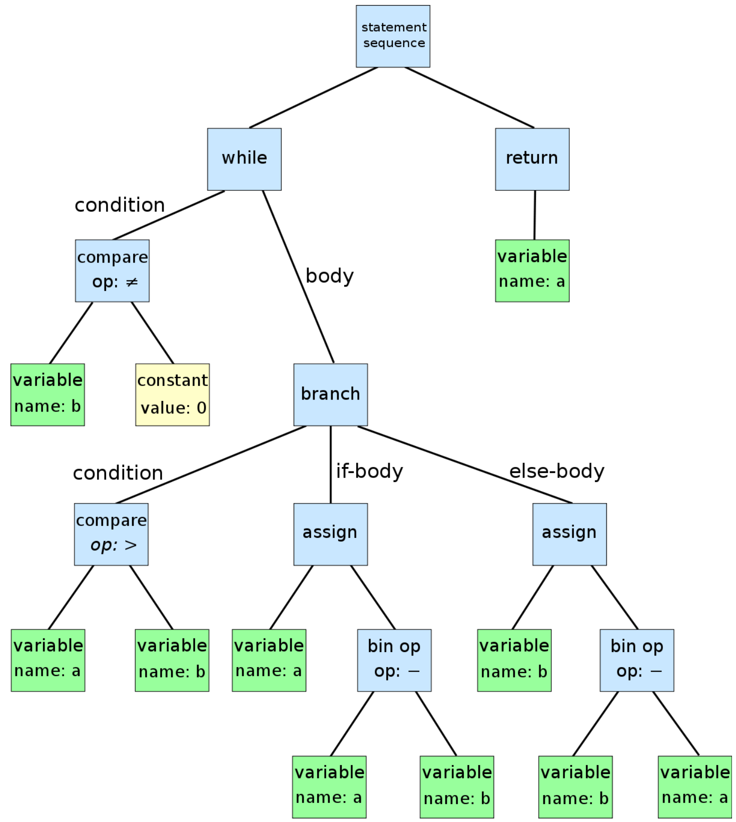
Table 1. Basic rules of three-way merge.

	Type	Base B	Left L	Right R	Target T
1	Node	e	e	e'	e'
2	Node	e	e_L	e_R	conflict
3	List	$e \in B$	$e \in L$	$e \notin R$	$e \notin T$
4	List	$e \notin B$	$e \in L$	$e \notin R$	$e \in T$ or conflict

Abstract Syntax Tree

$$
\begin{array}{llll}
\text{AST } N & ::= & V & \text{(leaf)} \\
 & | & F(N_1, N_2, \ldots, N_k) & \text{(constructed)} \\
 & | & \texttt{List}(N_1, N_2, \ldots, N_k) & \text{(list)}
\end{array}
$$

Abstract Syntax Tree

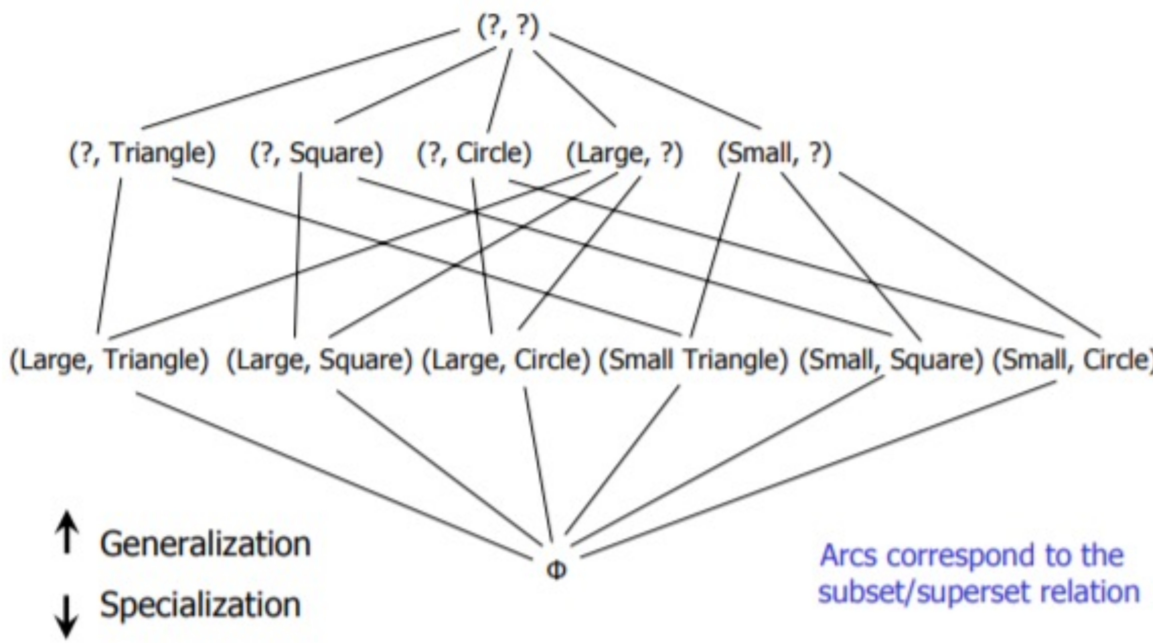


Version Space Learning

- Initially defined by [Mitchell, 1982] for concept learning.
- In simple terms, a set of hypotheses that are consistent with the training data refers to the version space.
- Contains a **most specific** and a **most general** hypothesis.

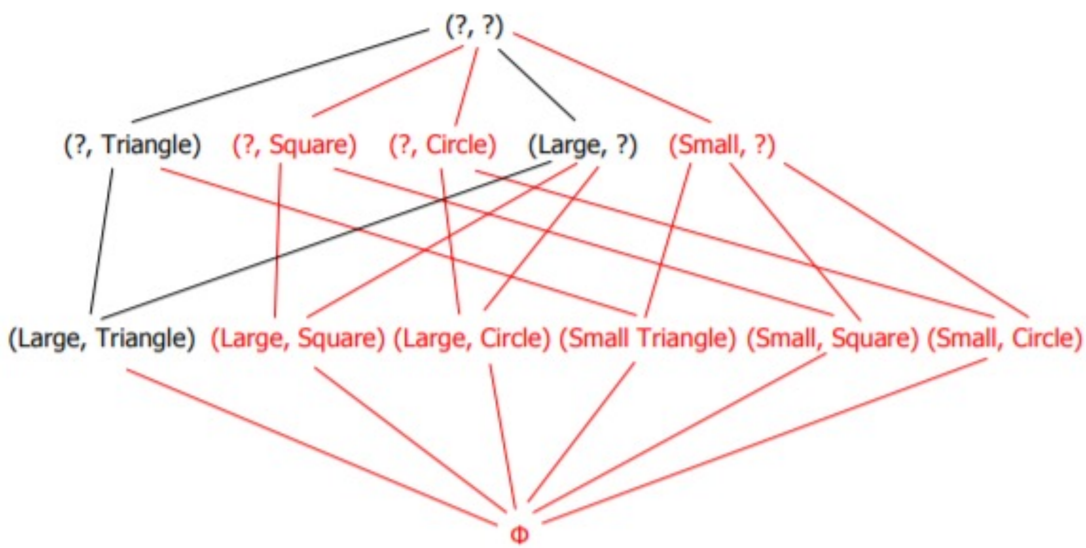
Version Space Learning

Attribute	Possible Values
Size	Large, Small
Shape	Triangle, Square, Circle



Version Space Learning

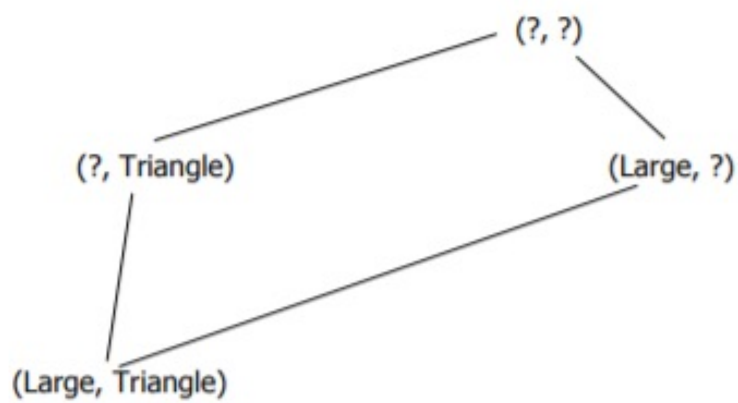
1st training example: (Large, Triangle) → +



Remove all concepts that do not include (Large, Triangle)

I.e., remove all concept descriptions that (Large, Triangle) does not match

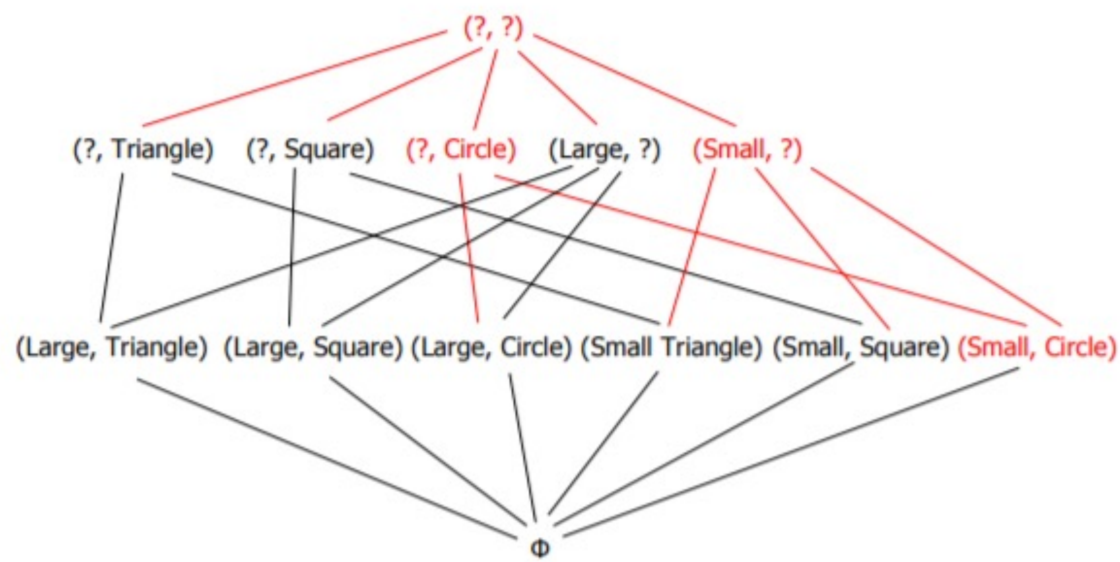
Version Space Learning



Updated version space after
(Large, Triangle) → +

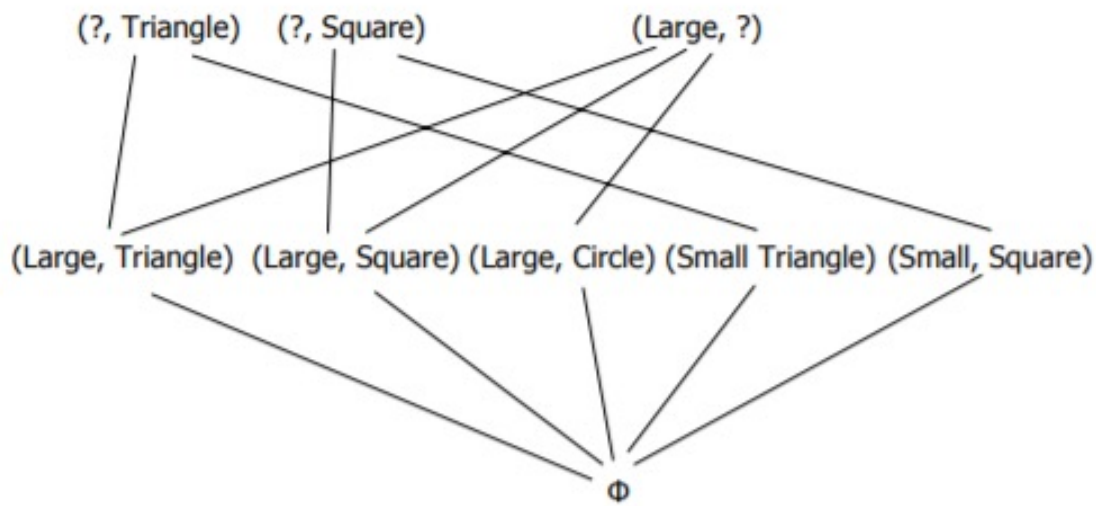
Version Space Learning

1st training example: (Small, Circle) → -



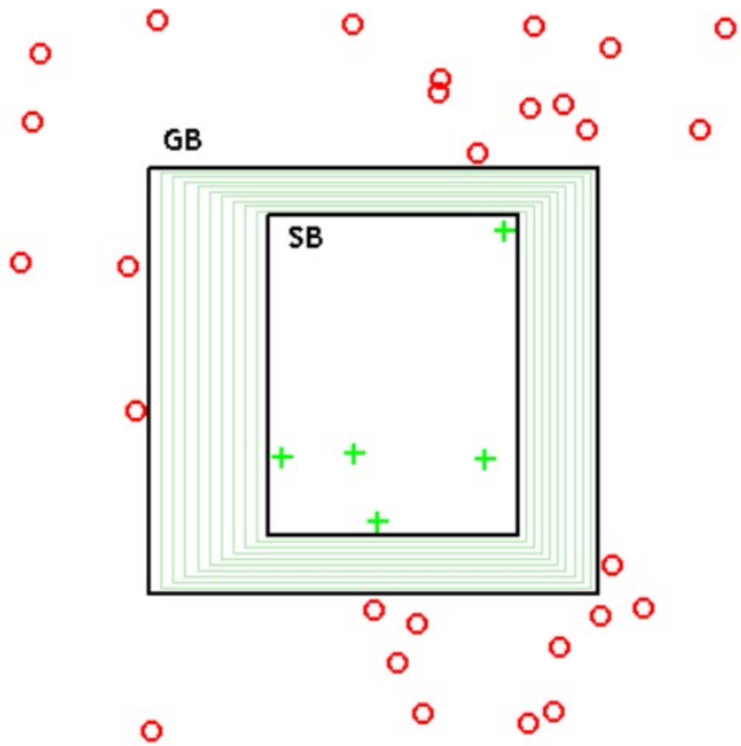
Remove all concept descriptions that (Small, Circle) matches

Version Space Learning



Updated version space

Version Space Learning



Version Space Algebra

- [Lau, 2000] extends the notion of version spaces beyond concept learning.
- It is proposed that carefully-tailored version spaces can be built for complex applications.
- Version space algebra (VSA) is defined: It uses a set of defined operations to compose together many simple version spaces to represent a complex composition.
- Allows arbitrary partial ordering of the hypotheses (not necessarily generality).
- Demonstrate effectiveness using SMARTedit, which is a repetitive text-editing tool.

Version Space Algebra

- From an intuitive aspect, a VSA can be viewed as a directed graph where each node represents a set of programs.

$$
\begin{aligned}
\text{AST } N \quad ::= \quad & V && \text{(leaf)} \\
\mid \quad & F(N_1, N_2, \ldots, N_k) && \text{(constructed)} \\
\mid \quad & \texttt{List}(N_1, N_2, \ldots, N_k) && \text{(list)}
\end{aligned}
$$

$$
\begin{aligned}
\text{VSA } \widetilde{N} \quad ::= \quad & \{P_1, P_2, \ldots, P_k\} && \text{(explicit)} \\
\mid \quad & \widetilde{N_1} \cup \widetilde{N_2} \cup \cdots \cup \widetilde{N_k} && \text{(union)} \\
\mid \quad & F_{\bowtie}(\widetilde{N_1}, \widetilde{N_2}, \ldots, \widetilde{N_k}) && \text{(join)} \\
\mid \quad & \texttt{List}_{\bowtie}(\widetilde{N}) && \text{(list join)}
\end{aligned}
$$

Motivating Example

```
...
for (...) {
  try {
    s1;
  } catch {
    // empty
  }
}
...
```

(a) *Base*

```
...
for (...) {
  s3;
  if (...) {
    try {
      s4;
    } catch {
      // empty
    }
  }
}
...
```

(b) *Left*

```
...
for (...) {
  try {
    s1;
  } catch {
    s2;
  }
}
...
```

(c) *Right*

```
...
for (...) {
  s3;
  if (...) {
    try {
      s4;
    } catch {
      s2;
    }
  }
}
...
```

(d) *Merged*

Fig. 1. The *base*, *left*, *right* and *merged* versions of the motivating example. Changes are highlighted.

Merged code is a combination of the left and right branches

General Algorithm

- (Line 4) Conflict detection. A structured merger is applied on the merge scenario (B, L, R) to generate a target program T_H with a set of holes H.
- (Line 7) Program space representation. For each hole h ∈ H, we construct a VSA S_h, which represents all possible resolutions that can instantiate the hole h.
- (Line 8) Resolution ranking. We rank the candidate resolutions in Sh with a ranking function.
- (Line 9) Instantiate the hole h with the accepted resolution P.

Algorithm 1 Merge

1: **procedure** MERGE(B, L, R)
2: **Input:** Base version B, left version L and right version R
3: **Output:** Merged result T
4: perform a structured merge on (B, L, R) and generate T_H;
5: $T \leftarrow T_H$;
6: **for all** hole $h \in H$ **do**
7: $\widetilde{S_h} \leftarrow$ CONSTRUCTVSA(h);
8: rank $\widetilde{S_h}$;
9: $T \leftarrow T[h \mapsto P]$ where $P \in \widetilde{S_h}$ is the accepted resolution;
10: **return** T;

AST to VSA Conversion

AST N	::=	V	(leaf)
	|	$F(N_1, N_2, \ldots, N_k)$	(constructed)
	|	$\texttt{List}(N_1, N_2, \ldots, N_k)$	(list)

$$\frac{}{\alpha(V) = \{V\}} \text{ A-EXP}$$

$$\frac{\widetilde{N_1} = \alpha(N_1), \widetilde{N_2} = \alpha(N_2), \ldots, \widetilde{N_k} = \alpha(N_k)}{\alpha(F(N_1, N_2, \ldots, N_k)) = F_{\bowtie}(\widetilde{N_1}, \widetilde{N_2}, \ldots, \widetilde{N_k})} \text{ A-JOIN}$$

$$\frac{\widetilde{N} = \alpha(N_1) \cup \alpha(N_2) \cup \cdots \cup \alpha(N_k)}{\alpha(\texttt{List}(N_1, N_2, \ldots, N_k)) = \texttt{List}_{\bowtie}(\widetilde{N})} \text{ A-LISTJOIN}$$

Fig. 2. Conversion rules for AST to VSA, where α is the conversion operation.

VSA Construction

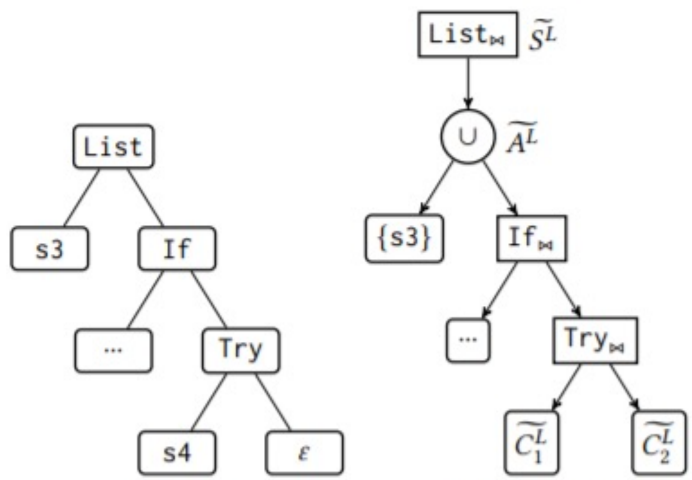
Algorithm 2 VSA Construction

procedure ConstructVSA(Hole(b, l, r))
  Visit$(l, 1, S)$;
  Visit$(r, 1, S)$;
  return $\widetilde{S}$;
procedure Visit(t, d, N)
  if $d \geq D$ **then** $\widetilde{N} \leftarrow \widetilde{N} \cup \{t\}$;
  else
    match t
      case V **then** $\widetilde{N} \leftarrow \widetilde{N} \cup \{V\}$; ▹ t is a leaf
      case $F(N_1, N_2, \ldots, N_k)$ **then** ▹ t is a constructed node
        for $i = 1$ to k **do**
          $V_i \leftarrow f(F, i, N)$; ▹ mapper f returns an identifier
          Visit$(N_i, d+1, V_i)$;
        $\widetilde{N} \leftarrow \widetilde{N} \cup F_{\bowtie}(\widetilde{V_1}, \widetilde{V_2}, \ldots, \widetilde{V_k})$;
      case List$(N_1, N_2, \ldots, N_k)$ **then** ▹ t is an (ordered or unordered) list
        for $i = 1$ to k **do** Visit(N_i, d, V_N);
        $\widetilde{N} \leftarrow \widetilde{N} \cup \text{List}_{\bowtie}(\widetilde{V_N})$;

AST to VSA- Left Branch

```
...
for (...) {
  s3;
  if (...) {
    try {
      s4;
    } catch {
      // empty
    }
  }
}
...
```

(b) *Left*



(a) AST l (b) VSA $\alpha(l)$

$$\widetilde{S^L} = \mathtt{List}_{\bowtie}(\widetilde{A_L})$$

$$\widetilde{A^L} = \{\mathtt{s3}\} \cup \mathtt{If}_{\bowtie}(\_, \widetilde{B})$$

$$\widetilde{B} = \mathtt{Try}_{\bowtie}(\widetilde{C_1^L}, \widetilde{C_2^L})$$

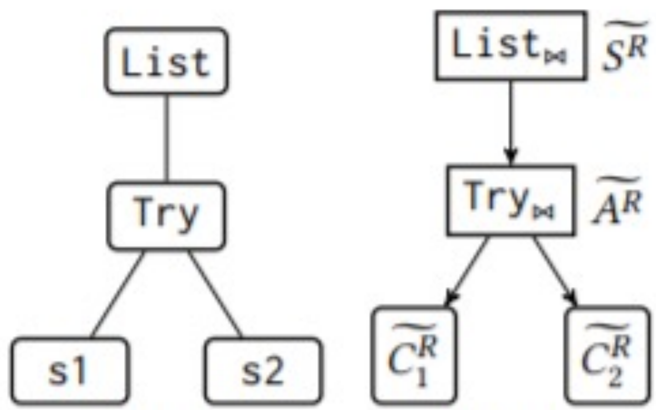
$$\widetilde{C_1^L} = \{\mathtt{s4}\}$$

$$\widetilde{C_2^L} = \{\varepsilon\}$$

AST to VSA- Right Branch

```
...
for (...) {
  try {
    s1;
  } catch {
    s2;
  }
}
...
```

(c) *Right*



(c) AST r (d) VSA $\alpha(r)$

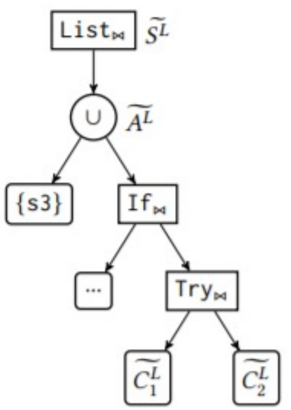
$\widetilde{S^R} = \texttt{List}_{\bowtie}(\widetilde{A_R})$

$\widetilde{A^R} = \texttt{Try}_{\bowtie}(\widetilde{C_1^R}, \widetilde{C_2^R})$

$\widetilde{C_1^R} = \{\texttt{s1}\}$

$\widetilde{C_2^R} = \{\texttt{s2}\}$

AST to VSA- Merged



(b) VSA $\alpha(l)$

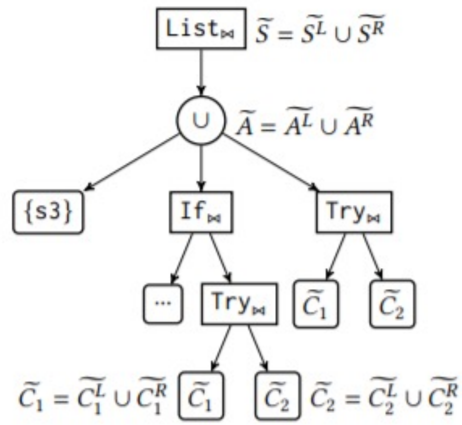
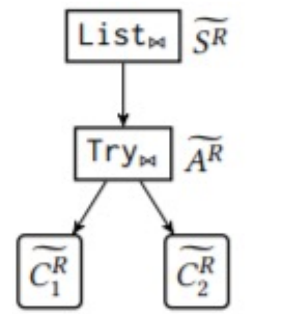


Fig. 4. Merged VSA of $\widetilde{S^L}$ and $\widetilde{S^R}$ in Figure 3.



(d) VSA $\alpha(r)$

$$\widetilde{S^L} = \texttt{List}_{\bowtie}(\widetilde{A_L})$$
$$\widetilde{A^L} = \{\texttt{s3}\} \cup \texttt{If}_{\bowtie}(\_, \widetilde{B})$$
$$\widetilde{B} = \texttt{Try}_{\bowtie}(\widetilde{C_1^L}, \widetilde{C_2^L})$$
$$\widetilde{C_1^L} = \{\texttt{s4}\}$$
$$\widetilde{C_2^L} = \{\varepsilon\}$$

$$\widetilde{S} = \widetilde{S^L} \cup \widetilde{S^R} = \texttt{List}_{\bowtie}(\widetilde{A})$$
$$\widetilde{A} = \widetilde{A^L} \cup \widetilde{A^R} = \{\texttt{s3}\} \cup \texttt{If}_{\bowtie}(\_, \widetilde{B}) \cup \texttt{Try}_{\bowtie}(\widetilde{C_1}, \widetilde{C_2})$$
$$\widetilde{B} = \texttt{Try}_{\bowtie}(\widetilde{C_1}, \widetilde{C_2})$$
$$\widetilde{C_1} = \widetilde{C_1^L} \cup \widetilde{C_1^R} = \{\texttt{s4}\} \cup \{\texttt{s1}\}$$
$$\widetilde{C_2} = \widetilde{C_2^L} \cup \widetilde{C_2^R} = \{\varepsilon\} \cup \{\texttt{s2}\}$$

$$\widetilde{S^R} = \texttt{List}_{\bowtie}(\widetilde{A_R})$$
$$\widetilde{A^R} = \texttt{Try}_{\bowtie}(\widetilde{C_1^R}, \widetilde{C_2^R})$$
$$\widetilde{C_1^R} = \{\texttt{s1}\}$$
$$\widetilde{C_2^R} = \{\texttt{s2}\}$$

AST to VSA- Merged

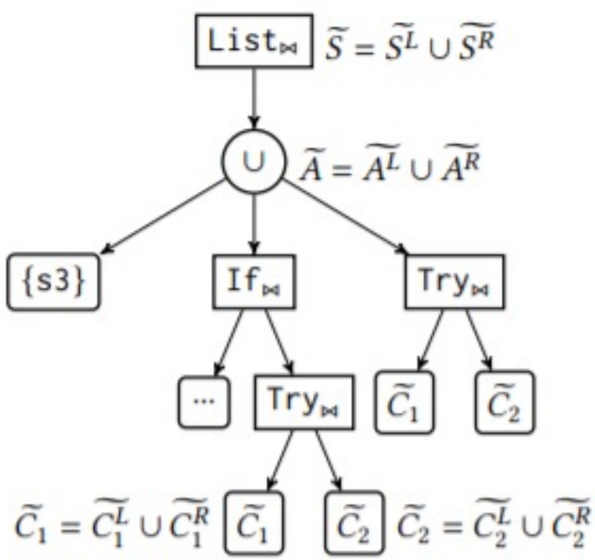


Fig. 4. Merged VSA of $\widetilde{S^L}$ and $\widetilde{S^R}$ in Figure 3.

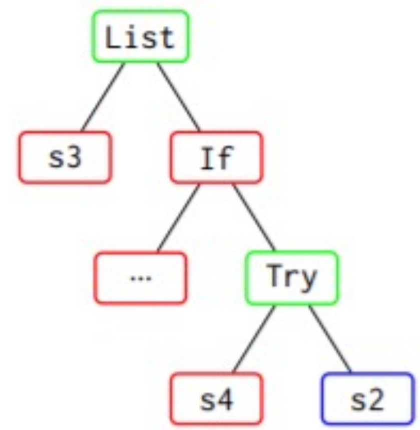


Fig. 5. AST of the program in Figure 1d. Node color represents from which version it is derived: red for *left*, blue for *right*, and green for both.

Mappers

- Direct Mapper
- Block Mapper
- Expression Mapper
- Takes three arguments:
  - the type of the constructor (F)
  - the index of the argument (i)
  - the context (N) in which the mapper is invoked

Direct Mapper

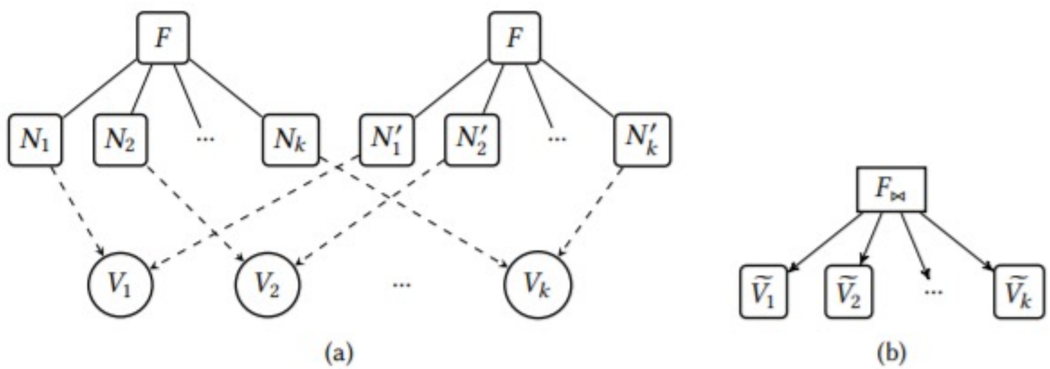
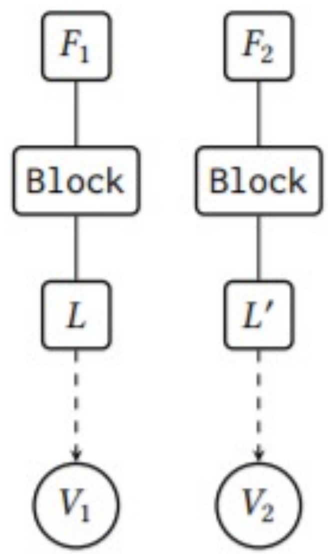


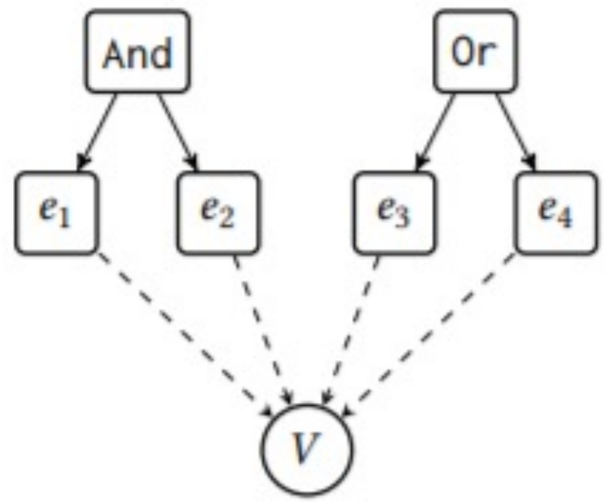
Fig. 6. Direct mapper. (a) For two constructed nodes $F(N_1, N_2, \ldots, N_k)$, $F(N'_1, N'_2, \ldots, N'_k)$ with a common constructor F, the direct mapper maps N_i and N'_i to the same identifier V_i, for $i = 1, 2, \ldots, k$, as shown by the dashed arrows. (b) The join node composed of VSAs $\widetilde{V_1}, \widetilde{V_2}, \cdots, \widetilde{V_k}$.

Block Mapper



(a) Block mapper

Expression Mapper



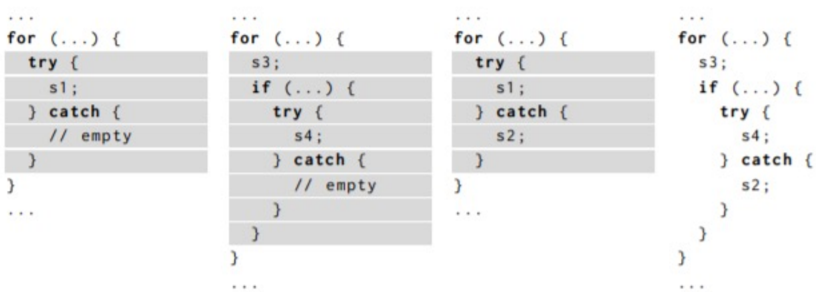
(b) Expression mapper

Resolution Ranking

- "Prior to" relation is defined on VSA identifiers (Example: $V_1 < V_2$).
- Given two identifiers V_1, V_2, we define the partial relation motivated by the basic rules of three-way merge.
- Given two identifiers V_1, V_2, we define $V_1 < V_2$ if:

$$(S_{V_1} = \{\mathsf{L}\} \land \mathsf{B} \in S_{V_2} \land \mathsf{R} \in S_{V_2}) \lor (S_{V_1} = \{\mathsf{R}\} \land \mathsf{B} \in S_{V_2} \land \mathsf{L} \in S_{V_2}) \lor (S_{V_1} = \{\mathsf{L},\mathsf{R}\} \land \mathsf{B} \in S_{V_2}).$$

Resolution Ranking



(a) *Base* (b) *Left* (c) *Right* (d) *Merged*

Fig. 1. The *base*, *left*, *right* and *merged* versions of the motivating example. Changes are highlighted.

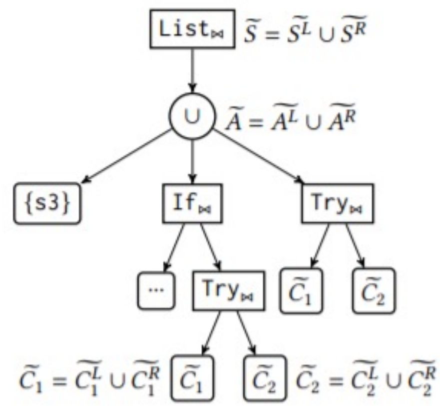


Fig. 4. Merged VSA of $\widetilde{S^L}$ and $\widetilde{S^R}$ in Figure 3.

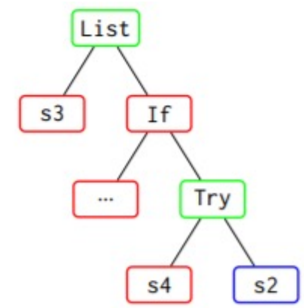


Fig. 5. AST of the program in Figure 1d. Node color represents from which version it is derived: red for *left*, blue for *right*, and green for both.

Evaluation: Research Questions

- **RQ1:** : How effective and efficient is AutoMerge at resolving conflicts?
- **RQ2:** How do various mappers affect the effectiveness and efficiency of our approach?

Evaluation: Data Set

- 10 open-source projects with high stars from GitHub are selected and their commit histories are analyzed for merge commits.
- The merged versions committed by the developers are considered the ground truth.
- Focuses on merge commits that cannot be fully merged by JDime (improved version).
- 95 conflicting merge commits are retrieved.
- By default, the direct and block mappers are used.

Evaluation: Data Set

Table 2. Summary of extracted merge scenarios. Conf. commits: number of conflicting merge commits.

Project	Conf. commits	Description
auto	1	A collection of source code generators for Java.
drjava	2	A lightweight programming environment for Java.
error-prone	6	Catch common Java mistakes as compile-time errors.
fastjson	6	A fast JSON parser/generator for Java.
freecol	4	A turn-based strategy game.
itextpdf	47	Core Java Library + PDF/A, xtra and XML Worker.
jsoup	2	Java HTML Parser, with best of DOM, CSS, and jquery.
junit4	21	A programmer-oriented testing framework for Java.
RxJava	1	Reactive Extensions for the JVM.
vert.x	5	A tool-kit for building reactive applications on the JVM.

Evaluation: Result

Project	Conf. files	Holes	Resolved holes	Max. k	Avg. k	P.S.	Time (ms)
auto	4	11	10 (90.9%)	2	1.18	191.1	94.72
drjava	2	2	2 (100%)	2	1.50	515	297.50
error-prone	8	13	8 (61.5%)	13	4.62	6.31	146.46
fastjson	8	19	19 (100%)	18	2.37	8.37	119.16
freecol	22	57	57 (100%)	2	1.81	23.9	87.91
itextpdf	47	47	47 (100%)	1	1.00	6	231.94
jsoup	2	2	2 (100%)	1	1.00	6	116
junit4	33	51	45 (88.2%)	13	1.78	133	126.73
RxJava	1	1	1 (100%)	2	2.00	6	1
vert.x	11	41	41 (100%)	4	1.78	7.24	63.22
Overall	138	244	232 (95.1%)	18	1.79	48.88	127.10

Evaluation: Depth Analysis

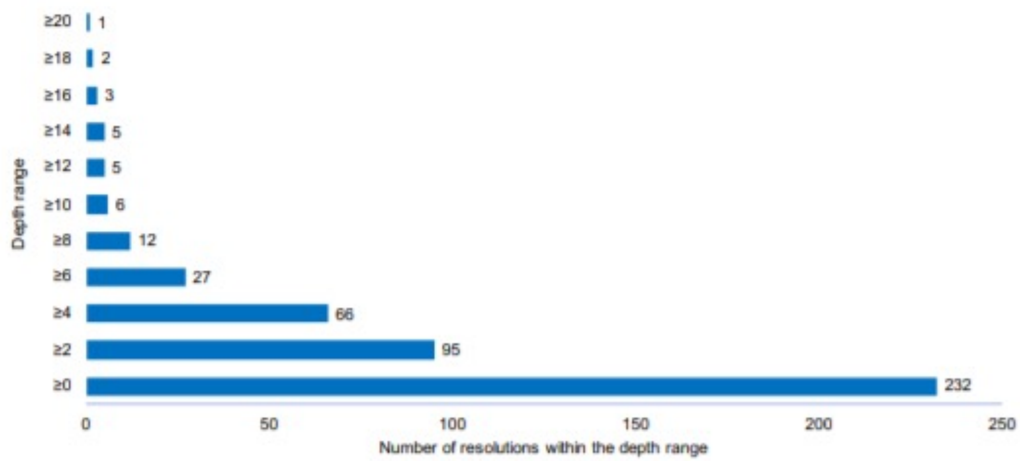


Fig. 8. Complexity of resolutions measured by the depths of the merged AST. The stripe annotated with $\geq d$ shows the number of resolutions with a depth greater or equal to d.

Evaluation: Failed Cases

- Failed to find expected resolution in the top 50 candidates for 12 holes.

- Assumption Violation (8 cases)
  - Assumption that expected resolution is a combination of the left and right branches.

- Insufficient Expressiveness (2 cases)
  - Assumption that the constructed VSA requires the root of the AST must have the identical kind with either the left or the right version.

- Huge Program Space (2 cases)
  - The ranking function is unable to rank the expected program within top 60 for these 2 cases.

Evaluation: Renaming Resolution

```
/* base */
if (...) {
  deserizer = parser.getConfig().getDeserializer(userType);
} else {
  ...
}

/* left */
deserizer = parser.getConfig().getDeserializer(userType);

/* right */
if (...) {
  deserializer = parser.getConfig().getDeserializer(userType);
} else {
  ...
}

/* expected */
deserializer = parser.getConfig().getDeserializer(userType);
```

Evaluation: Mapper Effectiveness

Fig. 9. Evaluation results on different configurations. A configuration is determined by enabling (on) / disabling (off) M1 and M2. Red stripe indicates the number of holes that perform worse than the baseline configuration c_1. No cases perform better, and the others keep unchanged.

Related Work

- Software Merge
- VSA-based Program Synthesis
- Program Transformation

Related Work

- Software Merge:
  - Unstructured Merge
  - Semistructured Merge
  - Structured Merge
  - Conflict Detection
  - Refactoring Aware Merge

Related Work

- VSA-based Program Synthesis:
  - Program synthesis aims to find executable programs that accomplish a wide range of categories of user intents.
  - Programming by Example (PBE) is a leading inductive synthesis technique which generates programs from input-output examples.

Related Work

- Program Transformation:
  - In general, program transformation is the process of formally changing a program to a different program with the same semantics as the original program.
  - Frameworks have been developed using PBE methodology to learn program transformation from input-output examples.
  - Graph-based technique has been developed that guides developers in adapting API usages [Nguyen, 2010].

Conclusion

- This paper proposes a VSA-based conflict resolution approach.
- Experiments are conducted on 95 merge commits from 10 open source projects on GitHub.
- AutoMerge detects 244 conflicts spread over 138 files, and successfully resolves as high as 95.1% of the conflicts.
- The ranking technique needs to try 1.79 candidates in average until the expected resolution is found.

Discussion

- Are the use of mappers justified?
- Is the complexity of the algorithm reasonable?
- Can it successfully resolve rename conflicts? What about higher order conflicts?
- Thoughts on the ranking mechanism?
- Any weaknesses in the experiment design?

References

1. Zhu, F., & He, F. (2018). Conflict resolution for structured merge via version space algebra. *Proceedings of the ACM on Programming Languages*, *2*(OOPSLA), 1-25.
2. Mitchell, T. M. (1982). Generalization as search. *Artificial intelligence*, *18*(2), 203-226.
3. Lau, T. A., Domingos, P. M., & Weld, D. S. (2000, June). Version Space Algebra and its Application to Programming by Demonstration. In *ICML* (pp. 527-534).
4. Nguyen, H. A., Nguyen, T. T., Wilson Jr, G., Nguyen, A. T., Kim, M., & Nguyen, T. N. (2010). A graph-based approach to API usage adaptation. *ACM Sigplan Notices*, *45*(10), 302-321.

Thank you