Conflict Resolution for Structured Merge via Version Space Algebra*

Fengmin Zhu and Fei He

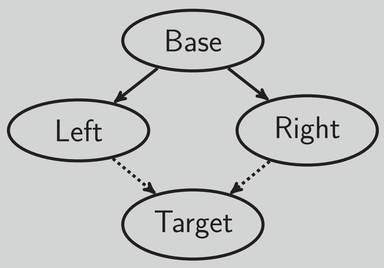
School of Software, Tsinghua University

Introduction

- Conflicts are widely exhibited in real-world software merge scenarios.
- Structured & unstructured approaches cannot resolve conflicts when concurrent changes contradict each other.
- We propose an interactive mechanism to provide the developer with candidate resolutions as recommendations.
- We present an algorithm to form the program space of resolutions and design a problem-specific ranking function for fast convergence.

Table: Three-way merge rules lead to conflicts.

	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



Objective: find recommended resolutions for conflicting scenarios.

Preliminary: VSA

Version space algebra (VSA) [1] succinctly represents a large set of programs [2].

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

Motivation

```
...                  ...                  ...                  ...
for (...) {          for (...) {          for (...) {          for (...) {
  try {                s3;                  try {                s3;
    s1;                if (...) {             s1;                if (...) {
  } catch {              try {                } catch {            try {
    // empty             s4;                  s2;                  s4;
  }                    } catch {            }                    } catch {
}                        // empty           }                      s2;
...                    }                   ...                    }
                     }                                          }
                   }                                          }
                   ...                                        ...
```

(a) *base* (b) *left* (c) *right* (d) *expected*

Figure: A conflicting merge scenario. Changes are highlighted.

"Combine" some of the changes and represent them as a VSA.

Methods

1. Conflict Detection
2. Program Space Representation with VSA
3. Resolution Ranking
4. User interaction

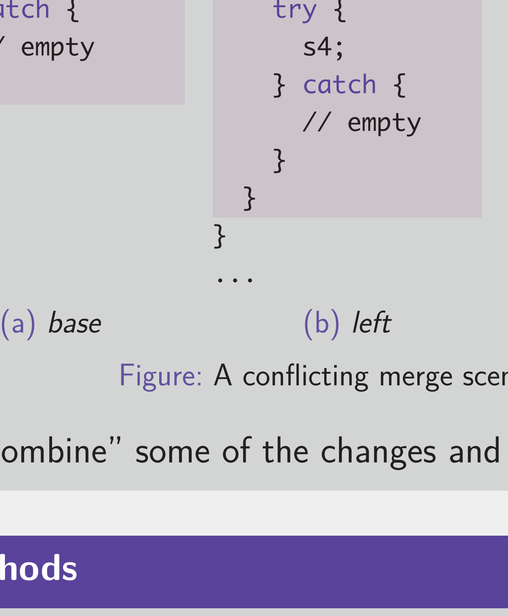


Figure: VSA representation for candidates.

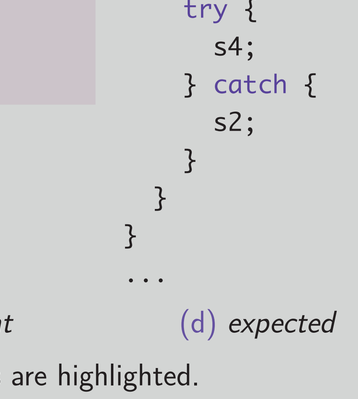


Figure: AST of the expected program. Node color represents from which version it is derived: blue for *left*, green for *right*, and purple for both.

System Architecture

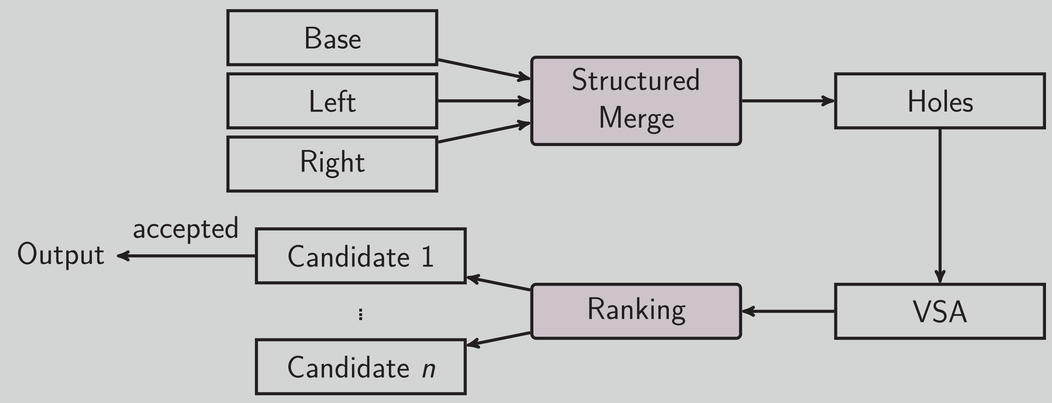


Figure: System architecture of `AutoMerge`, built on the top of `JDime` [3].

Evaluation Results

Table: Evaluation results on conflict resolution. Conf. files: number of conflicting files. k: interaction rounds. P.S.: size of program space per hole. Time: execution time of conflict resolution (excluding merge) per hole.

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

- High resolution rate by constructing expressive VSAs.
- Efficient implementation with restricted VSA construction.
- Few interaction rounds with the ranking mechanism.

Contributions

- We propose an interactive approach for resolving merge conflicts. To the best of our knowledge, this is the first attempt on conflict resolution of structured merge.
- We identify the expressiveness and efficiency of version space algebra, and elaborate an algorithm to automatically construct the VSA representation.
- We design a simple but effective mechanism for ranking resolutions.
- We prototype our approach as `AutoMerge`, and conduct experiments on real-world software projects. Results show great practicality of our approach.

References

[1] Tom M. Mitchell.
Generalization as search.
Artificial Intelligence, 18(2):203 – 226, 1982.

[2] Oleksandr Polozov and Sumit Gulwani.
FlashMeta: A framework for inductive program synthesis.
ACM SIGPLAN Notices, 50(10):107–126, 2015.

[3] Olaf Leßenich, Sven Apel, and Christian Lengauer.
Balancing precision and performance in structured merge.
Automated Software Engineering, 22(3):367–397, Sep 2015.

Contact Information

- Website: `https://thufv.github.io/automerge`
- Email: `zfm17@mails.tsinghua.edu.cn`, `hefei@tsinghua.edu.cn`

*This material is based upon work supported in part by the National Natural Science Foundation of China under Grant No. 61672310 and Grant No. 61527812, the National Science and Technology Major Project under Grant No. 2016ZX01038101.