Conflict Resolution for Structured Merge via Version Space Algebra

Fengmin Zhu[1,2,3] Fei He[1,2,3]

[1]School of Software, Tsinghua University, Beijing, China
[2]Key Laboratory for Information System Security, MoE
[3]Beijing National Research Center for Information Science and Technology

November 8, 2018

Questions

- What is structured merge?
- Why do conflicts present?
- How do we resolve them with version space algebra?

Overview

Contents

Three-way Merge Scenario

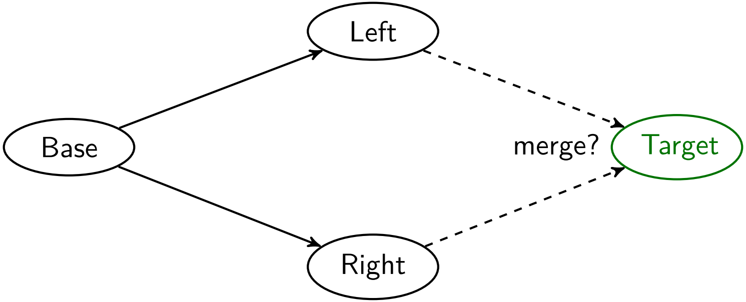


Figure: A three-way merge scenario (Base, Left, Right).

Unstructured & Structured Merge

- Unstructured merge: programs are regarded as lines of plain text (as in tool `diff`).
- Structured merge (Buffenbarger 1995; Westfechtel 1991): programs are regarded as abstract syntax trees (ASTs).
- Structured merge is more precise than unstructured merge.
- Both approaches follow the three-way merge rules.

Three-way Merge Rules

Table: Three-way merge rules lead to conflicts. (Westfechtel 1991)

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

Unstructured & Structured Merge

- Unstructured merge: programs are regarded as lines of plain text (as in tool `diff`).
- Structured merge (Buffenbarger 1995; Westfechtel 1991): programs are regarded as abstract syntax trees (ASTs).
- Structured merge is more precise than unstructured merge.
- Both approaches follow the three-way merge rules. Therefore,

Unstructured & Structured Merge

- Unstructured merge: programs are regarded as lines of plain text (as in tool `diff`).
- Structured merge (Buffenbarger 1995; Westfechtel 1991): programs are regarded as abstract syntax trees (ASTs).
- Structured merge is more precise than unstructured merge.
- Both approaches follow the three-way merge rules. Therefore, there exist conflicts which they CANNOT resolve.

Contents

A Conflicting Scenario

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

base

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

left

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

right

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

expected

Figure: A conflicting merge scenario. Changes are highlighted.

A Conflicting Scenario (AST)

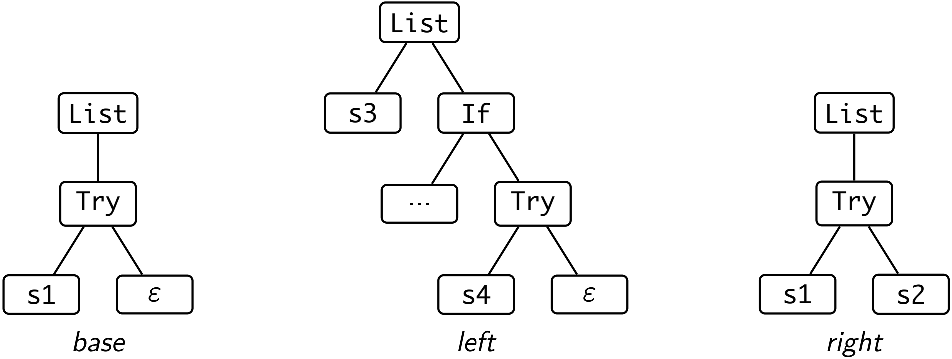


Figure: AST representation of the conflicting section.

Problem

Q: Is it possible to resolve the conflict?

Problem

Q: Is it possible to resolve the conflict?

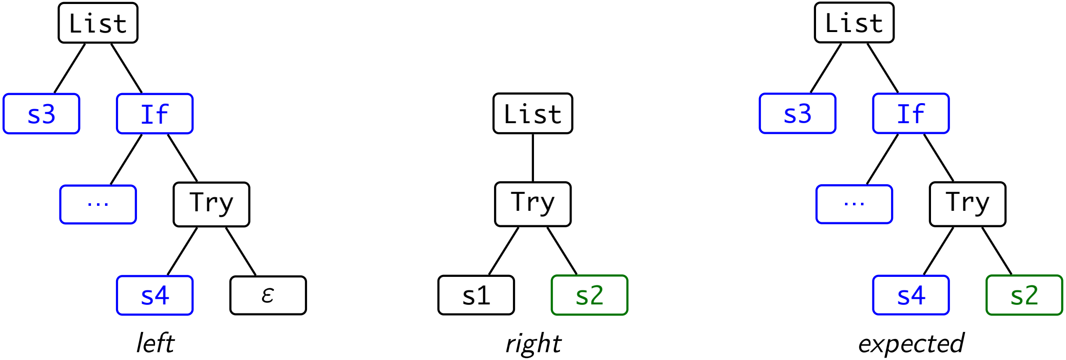


Figure: Connections between *expected* and (*left*, *right*).

Key Observations

- A resolution can be considered as a “merge” of the changes.
- Each change could be kept or not.
- There are possibly many such resolutions.
- Which ones are more likely to be accepted by the user?

Key Observations

- A resolution can be considered as a "merge" of the changes.
- Each change could be kept or not.
- There are possibly many such resolutions.
- Which ones are more likely to be accepted by the user?

- Q: Why structured merge didn't try to resolve conflicts?

Key Observations

- A resolution can be considered as a "merge" of the changes.
- Each change could be kept or not.
- There are possibly many such resolutions.
- Which ones are more likely to be accepted by the user?

- Q: Why structured merge didn't try to resolve conflicts?
- A: *Safety* matters.

Contents

Top-level Steps

1. Conflict detection by performing a structured merge with JDime (Leßenich, Apel, and Lengauer 2015)
2. Program space (possible resolutions) representation with *version space algebra*
3. Resolution ranking

Version Space Algebra (VSA)

- Defined by Mitchell 1982.
- Expanded upon *program synthesis* by Gulwani 2011; Polozov and Gulwani 2015.
- Succint representation by memory-sharing mechanism.

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

Version Space Algebra (Cont.)

- Each VSA node represents a set of concrete programs:

$$\llbracket \{P_1, P_2, \ldots, P_k\} \rrbracket = \{P_1, P_2, \ldots, P_k\}$$
$$\llbracket \widetilde{N_1} \cup \widetilde{N_2} \cup \cdots \cup \widetilde{N_k} \rrbracket = \llbracket \widetilde{N_1} \rrbracket \cup \llbracket \widetilde{N_2} \rrbracket \cup \cdots \cup \llbracket \widetilde{N_k} \rrbracket$$
$$\llbracket F_{\bowtie}(\widetilde{N_1}, \widetilde{N_2}, \ldots, \widetilde{N_k}) \rrbracket = \{F(P_1, P_2, \ldots, P_k) \mid P_1 \in \llbracket \widetilde{N_1} \rrbracket, P_2 \in \llbracket \widetilde{N_2} \rrbracket, \ldots, P_k \in \llbracket \widetilde{N_k} \rrbracket\}$$

List Join

- List structures are commonly seen, e.g. a for-loop body consists of a sequence of statements.
- Elements are chosen from a program space, e.g. possible statements $\{\mathtt{s3}, \mathtt{If}, \mathtt{Try}\}$.

$$\begin{array}{lcll} \text{VSA } \widetilde{N} & ::= & \cdots & \\ & \mid & \mathtt{List}_{\bowtie}(\widetilde{N}) & \text{(list join)} \end{array}$$

$$[\![\mathtt{List}_{\bowtie}(\widetilde{N})]\!] = \{\mathtt{List}(N_1, N_2, \ldots, N_k) \mid k \geq 0, N_1, N_2, \ldots, N_k \in [\![\widetilde{N}]\!] \text{ are distinct}\}$$

VSA Construction: Conversion Rules

$$\alpha :: \text{AST} \to \text{VSA}$$

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

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

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

VSA Construction: Conversion

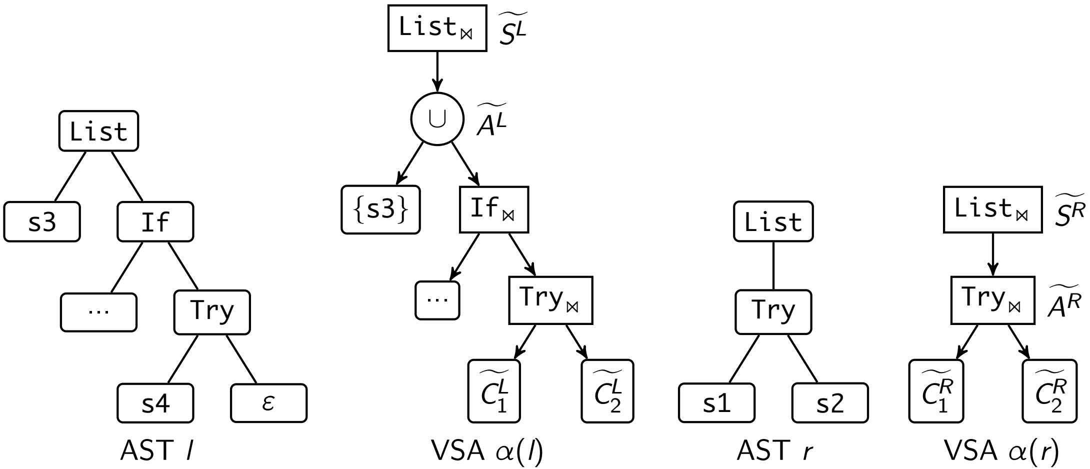


Figure: AST to VSA conversion.

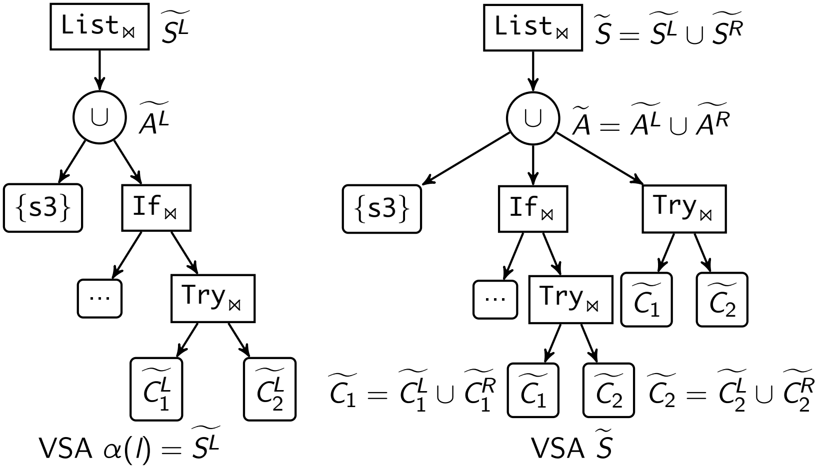


Figure: VSA "merge".

Algorithm

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);
        $\widetilde{N} \leftarrow \widetilde{N} \cup \mathtt{List}_{\bowtie}(\widetilde{V_N})$;

Constructed VSA

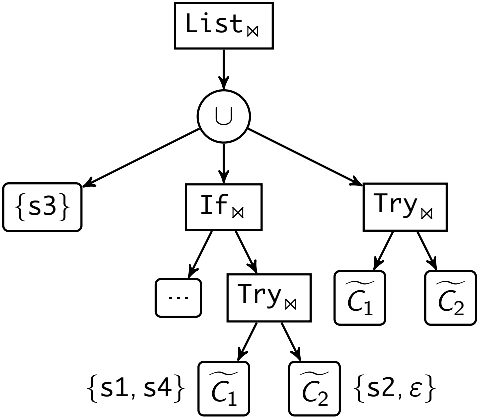
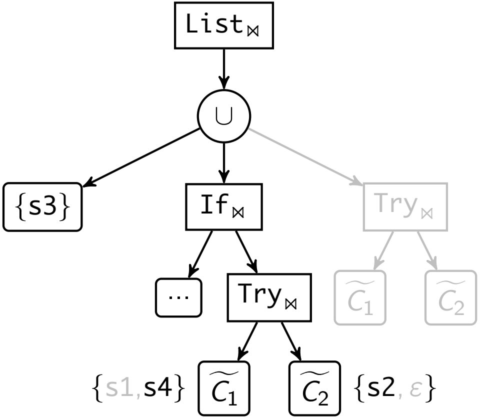


Figure: Constructed VSA

includes

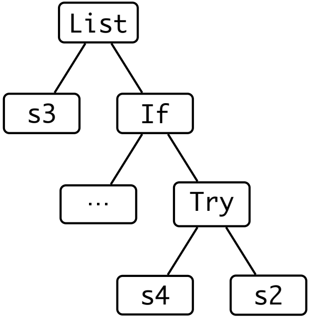
- *left*
- *right*
- a lot more other programs in between

Constructed VSA is Expressive



constructed VSA

includes



expected

Ranking Rules

	Type	Base B	Left L	Right R	Target T	Explanation
3	List	$e \in B$	$e \in L$	$e \notin R$	$e \notin T$	deletion
4	List	$e \notin B$	$e \in L$	$e \notin R$	$e \in T$ [1]	insertion

- Motivated by three-way merge rules:
  - If one node appears in base and left/right version, then it is likely not to appear in the merged version.
  - If one node appears only in left/right version, then it is likely to appear in the merged version.
- "Prior to" partial order relation on VSAs.

[1] Assume the list is unordered.

Contents

Benchmark Suite

Table: 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 Results

Table: Evaluation results. 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	[illegible]	[illegible]	[illegible]	7.24	63.22
Overall	138	244	232 (95.1%)	18	1.79	48.88	127.10

High resolution rate: 95.1%

Evaluation Results

Table: Evaluation results. 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	[illegible]
vert.x	11	41	41 (100%)	4	[illegible]	[illegible]	[illegible]
Overall	138	244	232 (95.1%)	18	1.79	48.88	127.10

Few interaction rounds: 1.79

Evaluation Results

Table: Evaluation results. 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%	[illegible]	[illegible]	[illegible]	[illegible]
Overall	138	244	232 (95.1%)	18	1.79	48.88	127.10

Efficient implementation: 127.10ms

Summary

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

References I

- Jim Buffenbarger. "Syntactic software merging". In: *Software Configuration Management.* Ed. by Jacky Estublier. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 153–172.
- Sumit Gulwani. "Automating String Processing in Spreadsheets Using Input-output Examples". In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '11. Austin, Texas, USA: ACM, 2011, pp. 317–330.
- Olaf Leßenich, Sven Apel, and Christian Lengauer. "Balancing precision and performance in structured merge". In: *Automated Software Engineering* 22.3 (2015), pp. 367–397.
- Tom M. Mitchell. "Generalization as search". In: *Artificial Intelligence* 18.2 (1982), pp. 203 –226.
- Oleksandr Polozov and Sumit Gulwani. "FlashMeta: A framework for inductive program synthesis". In: *ACM SIGPLAN Notices* 50.10 (2015), pp. 107–126.

References II

Bernhard Westfechtel. "Structure-oriented Merging of Revisions of Software Documents". In: *Proceedings of the 3rd International Workshop on Software Configuration Management*. SCM '91. Trondheim, Norway: ACM, 1991, pp. 68–79.

Further Information

Please visit our site

https://thufv.github.io/automerge

Thanks for your listening!