Smerge: Smarter Merge Conflict Resolutions

Alva Wei (alvawei)
Jediah Conachan (jediah6)
Kenji Nicholson (kenjilee)
Steven Miller (stevenm62)
Sungmin Rhee (srhee4)

Motivation

Problem: version control systems use line-based analysis to detect merge conflicts which results in many unnecessary manual merges

Manual merges = valuable developer time = more money spent on projects

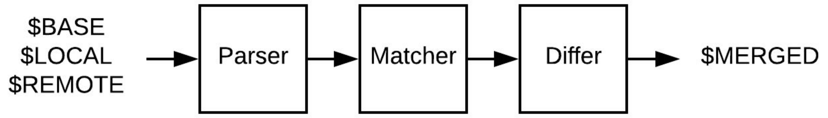
Current Approaches

- Conflerge
  - Uses Abstract Syntax Tree (AST) merging to automatically merge conflicts
  - Dependency on JavaParser means only compatible with Java
  - Discards custom whitespace placed by contributors
- Git
  - Capable of merging whitespace conflicts
  - Conservative to prevent undesired merges
  - Requires manual resolution on most merge conflicts

Goals

- Reduce the number of conflicts presented to the users
- Provide a merging module that can be used with several different language parsers
- Automatically handle as many trivial merge conflicts as possible
- Reduce frequency of undesired behavior

Approach and Implementation



- Parser takes three input files and generates an AST for each
    - Easier to parse into a generic AST that fits our needs with our own parser
- Matcher finds tree matchings from `$LOCAL` and `$REMOTE` ASTs to `$BASE` AST
- Differ produces action sets for both matchings and merges them into one AST

Parsing

- Files are parsed line-by-line into ASTs
- AST nodes store parent, children, type, label, indentation, and ID (for matching)
- Types:
    - Import
    - Method
    - For loop
    - Assignment
    - Comment
    - Whitespace

    etc.

Matching

- `$BASE` AST nodes are assigned unique IDs
- `$LOCAL` and `$REMOTE` ASTs are compared to `$BASE` AST
- Fuzzy label matching
  - Should `print(“hello”)` be considered the same as `print(“Hello”)`?
  - What about `x = 1` and `x = 2`?
- Parent and child matching (not yet implemented)

```
$BASE:
(1) import A
(2)
(3) x = 1
(4) y = “hello”
```

```
$LOCAL:
(1) import C
(2)
(3) x = 2
(5)
(4) y = “Hello”
```

```
$REMOTE:
(1) import B
(2)
(6) if True:
(3)     x = 1
(7) y = “hi world!”
```

Diffing

```
$BASE:
(1) import A
(2)
(3) x = 1
(4) y = “hello”
```

```
$LOCAL:
(1) import C
(2)
(3) x = 2
(5)
(4) y = “Hello”
```

```
$REMOTE:
(1) import B
(2)
(6) if True:
(3)   x = 1
(7) y = “hi world!”
```

- 4 kinds of node actions: insert, delete, move, update
  - Move can be considered as a delete followed by an insert
- Action set from `$BASE` to `$LOCAL`:
  - Update 1, `import A` to `import C`
  - Update 3, `x = 1` to `x = 2`
  - Update 4, `y = “hello”` to `y = “Hello”`
  - Insert 5 as the 4th child of `root`

Demo

Evaluation

- **Question**: How successful is Smerge at reducing conflicts?
- **Hypothesis**: Smerge will reduce the number of merge conflicts experienced by the programmer
- **Procedure**

1. Gather many GitHub repositories and their respective historical data
2. From the historical data, look for merge commits that have two parents
3. Use git’s standard merge tools on the commits to see how many conflicts arise as a baseline
4. Use Smerge’s merging algorithm and record metric information
5. Compare the human resolution to the resolution presented by Smerge automatically. Categorize merge result as correct, correct w/o comments, unresolved or incorrect.

Metrics

	Positives	**Negatives**
True	Tool detect merge conflict AND merge requires manual resolution	Tool does not detect merge conflicts AND merge can be done automatically
False	Tool detect merge conflict BUT merge can be done automatically	Tool does not detect merge conflicts BUT merge requires manual resolution

Metrics (cont.)

- **Conflicts:** The number of merge conflicts found in the repo by git merge
- **% Correct:** The conflicts that Smerge was able to resolve correctly.
- **% Correct w/o Comments/Whitespace:** The conflicts that were resolved correctly, ignoring comments and whitespace
- **% Unresolved:** The conflicts that Smerge aborted merging would result in possibly undesired behavior.
- **% Incorrect:** The conflicts that Smerge reported to have merged, but the solution it produced differed from the programmer's manual resolution.

Initial Results

Repository	**#Conflicts**	**%Correct**	**%CorrectCW**	**%Unresolved**	**%Incorrect**
TensorFlow Models	7	0	0	100	0
Keras	33	0	0	100	0
flask	30	0	0	100	0

- Untested repositories: snallygaster, django, face_recognition, ansible, XX-net, scikit-lean

Conclusions

- Data suggests our tool is currently unsuccessful
- Goals for coming weeks:
    - Analyze cases where tool fails to merge
    - Improve tool to show positive results
    - Ensure correctness of evaluation scripts

Risks and Challenges

- No false-negative conflicts
- Minimizing false-positives
- Validity Threats
  - Sample selection bias
    - We select a small # of repos
  - Undercoverage bias
    - Edge cases where tool is close to useless