Smerge:
Smarter Merge
Conflict Resolutions

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

Motivation

- Version control systems (VCSs) use line-based analysis to detect merge conflicts
  - Unable to automatically resolve many merge conflicts, so developers have to resolve them manually
- Git (most popular VCS)
  - Capable of merging whitespace conflicts
  - Conservative to prevent undesired merges
  - Git mergetools
    - All-in-one merge solutions: GUI, code editor, automatic merge facility
- Manual merges = valuable developer time = more $$$ spent on projects
- **Goal:** Reduce the number of conflicts presented to the users
  - Automatically handle as many merge conflicts as possible

Merge Conflict Example

```
# Common Ancestor (Base):
for batch_index in range(0, nb_batch):
    batch_start = batch_index*batch_size
    batch_end = min(len(X), (batch_index+1)*batch_size)
    batch_ids = index_array[batch_start:batch_end]
```

Example from: Keras: Deep Learning for humans - https://github.com/keras-team/keras

Merge Conflict Example (cont.)

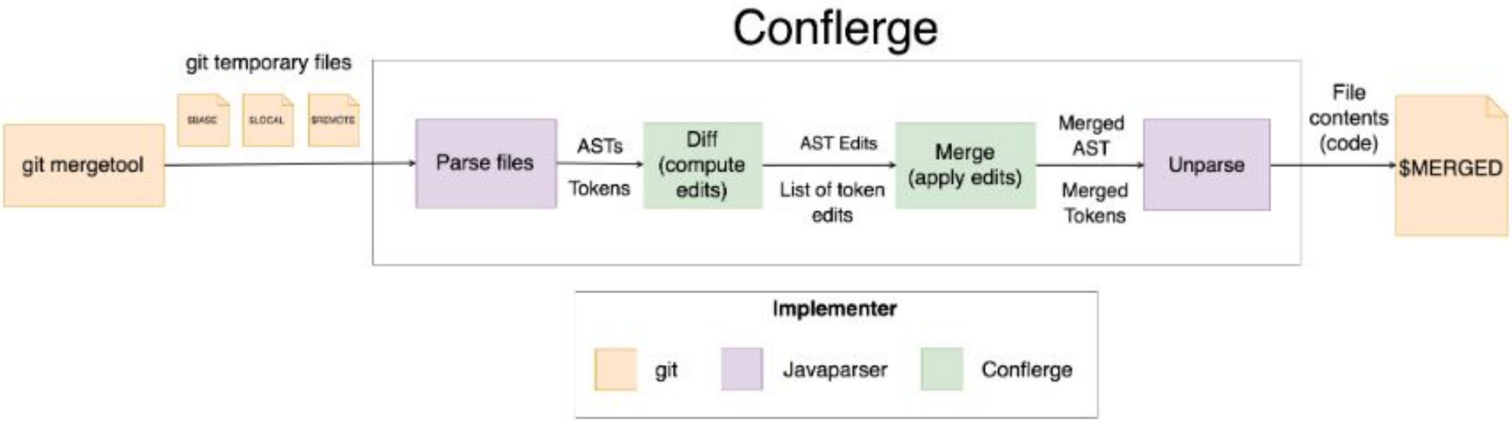
```python
# Theirs (Remote):
for batch_index, (batch_start, batch_end) in enumerate(batches):
   batch_start = batch_index*batch_size
   batch_end = min(len(X), (batch_index+1)*batch_size)
   batch_ids = index_array[batch_start:batch_end]
```

```python
# Yours (Local):
for batch_index in range(0, nb_batch):
   batch_start = batch_index*batch_size
   batch_end = min(len(X), (batch_index+1)*batch_size)
   if shuffle:
      batch_ids = index_array[batch_start:batch_end]
   else:
      batch_ids = slice(batch_start, batch_end)
```

Example from: Keras: Deep Learning for humans - https://github.com/keras-team/keras

Current Approaches: Conflerge

- Uses abstract syntax tree (AST) merging to automatically resolve conflicts
- Dependency on JavaParser means only compatible with Java
  - JavaParser, however, guarantees an AST represents valid Java code
- Discards custom whitespace placed by contributors

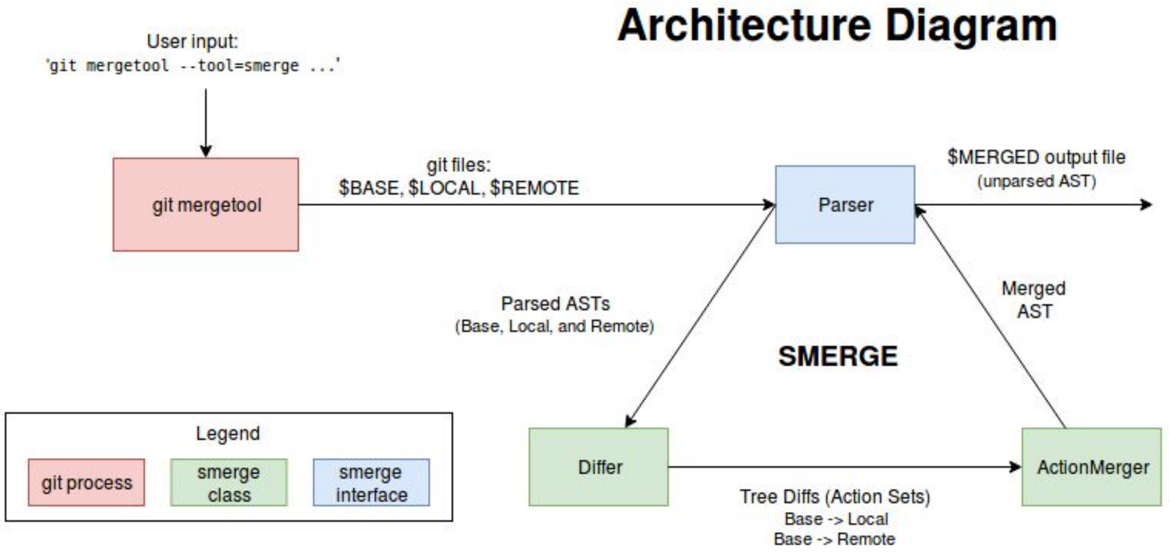


https://github.com/ishansaksena/Conflerge

Our Approach

- Centered on a language independent AST
- Our AST provides the following benefits:
    - Specialized for merging algorithms (don’t have to work around other existing AST formats)
        - Merging algorithms also become language independent
    - Retaining source code for unparsing the final merged tree
        - Allows for whitespace conservation
    - Control and flexibility

High Level Operations



Similar to Conflerge's, but completely different on the inside.

Parsing

- Must parse source code files into our generic AST
- Requires a separate Parser for each language
  - Each parser must implement our Parser interface
    - Requires methods for parsing and unparsing
  - We implemented our own simple Python Parser
- Base, local, and remote files are parsed line-by-line into our generic AST
- AST nodes store type, content, indentation, and an ID (for matching)
  - If a language requires more complex nodes, these nodes can be easily extended

Merging Process

- Input: BASE, LOCAL, REMOTE ASTs
- Compute tree diffs BASE→LOCAL and BASE→REMOTE
  - First, match the nodes between all three trees
  - Second, detect differences between BASE and LOCAL, and BASE and REMOTE
  - Tree diffs are represented as a set of actions (insert, delete, update a node)
- Merge and apply both diffs onto BASE
  - A fairly complicated process, especially where conflicts occur
- Output: a modified BASE AST, unparsed into merged source code

Conflict handling

- Conflerge’s solutions:
  - Imports
    - If a conflict involves import statements, keep all import statements
  - Comments:
    - If two users update a comment, keep the base version
- Otherwise:
  - Unresolvable conflict
    - Conflerge would fail
  - Smerge imitates git merge

```
<<<<<<<<<< REMOTE
__version__ = ‘1.0.3.dev’
==========
__version__ = ‘1.0.2.dev’
==========
__version__ = ‘1.1.dev’
>>>>>>>>>> LOCAL
```

Example from Flask - https://github.com/pallets/flask

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 Smerge's merging algorithm and record metric information automatically.
4. Compare the human resolution to the resolution presented by Smerge manually and categorize it into true and false positives and negatives.

Metrics for Automation

- **Conflicts:** The number of merge conflicts found in the repo by git merge. A conflict is counted as a conflicting portion.
- **Modified:** The conflicts that Smerge modified because it deemed the conflicts automatically resolvable.
- **Unresolved:** The conflicts that Smerge aborted because it deemed merging would result in possibly undesired behavior.

Automatic Results

Repository	#Conflicts	#Modified	#Unresolved	% Modified	%Unresolved
pipenv	234	215	19	91	8
TensorFlow Models	28	23	5	82	17
Keras	1052	871	181	82	17
flask	422	379	43	89	10
XX-net	14	12	2	85	14
ansible	933	814	119	87	12
scikit-learn	1761	1383	378	78	21
TOTAL:	**4444**	**3697**	**747**	**83**	**16**

Metrics for Manual Categorization

	Positives (“Modified”)	**Negatives (“Unresolved”)**
True	Tool does not detect merge conflicts AND performs the merge correctly	Tool detect merge conflicts AND merge requires manual resolution
False	Tool does not detect merge conflicts BUT performs the merge incorrectly	Tool detect merge conflicts BUT merge can be done automatically

Criteria

- True Positives:
  - If developer added new code, ignore it
  - Mismatch in whitespace
  - Choosing wrong variable between two equally likely choices
- False Positives:
  - Tool loses code in some cases
  - Code gets put in incorrect order
- True Negatives:
  - Differences in println statements, comments, string values
- False Negatives:
  - Tool sometimes doesn’t merge when one commit is blank

Manual Categorization Results

Repository	**# Conflicts**	**# T-Pos**	**# F-Pos**	**# T-Neg**	**# F-Neg**
pipenv	234	93	122	8	11
TensorFlow Models	28	12	11	5	0
XX-net	13	3	8	2	0
flask	422	172	207	24	19
TOTAL:	**697**	**280**	**348**	**39**	**30**

- Only manually checked 697 conflicts due to time constraints
- 83% were modified by Smerge, 45% of those were correctly resolved

Conclusions

- High amount of false-positives is concerning
    - Caused by tool not recognizing all possible merge cases
- Validity threats
    - Sample selection bias
        - We select a small # of repos
    - Undercoverage bias
        - Edge cases where tool is close to useless

Future Work & Goals

- **Conservativity:** Reduce the amount of cases where tool exhibits undesirable behavior (false-positives)
- **Parsing:** Adding more language parsers to support more languages
  - There is also room to improve our Python Parser (such as parsing method parameters into separate nodes)
- **Git Merge Integration:** Rework tool as git merge strategy (our tool is essentially an alternative non-line-based merging strategy)
- **Git Mergetool Feature:** Smerge may work well as a feature to an existing git mergetool (imagine a “Smart Merge” option in kdiff3)

Related Work

- **Flexible Tree Matching**, http://theory.stanford.edu/~tim/papers/ijcai11.pdf
- **Abstract Syntax Tree Matching**, http://shodhganga.inflibnet.ac.in/bitstream/10603/36935/12/12_chapter%204.pdf
- **Designing a Tree Diff Algorithm Using Dynamic Programming and A***, http://thume.ca/2017/06/17/tree-diffing/#the-algorithm
- **Fine-grained and Accurate Source Code Differencing**, https://hal.archives-ouvertes.fr/hal-01054552/document

Questions?