Implementing real-time collaboration in TouchDevelop using AST merges

Jonathan Protzenko Sebastian Burckhardt
Michał Moskal
Microsoft Research, USA
{sburckha,micmo,protz}@microsoft.com

Jedidiah McClurg
University of Colorado Boulder, USA
jedidiah.mcclurg@colorado.edu

Abstract

Collaborating on a piece of code is notoriously difficult when the number of people involved goes above 1. In particular, every computer programmer dreads the "merge conflict", a brutal, unforgiving experience, where they must reconcile their changes with someone else's.

If offline collaboration is already so painful, real-time collaboration seems even less of an option. It turns out, though, that by reasoning on changes at the level of the program AST, rather than the program text, we can devise a new conflict-free merge algorithm. The algorithm is particularly well-suited to real-time collaboration: we implemented it in the TouchDevelop web programming environment and dub the algorithm **diffTree**.

Categories and Subject Descriptors D.2.6 [*Programming Environments*]: Integrated environments

Keywords Merge, diff, collaborative editing

1. A new merge for new times

TouchDevelop [1] is a web-based programming environment; it has been used over the past few years as an experimental testbed. Indeed, the language is small and self-contained, and the user base is significant enough that we can conduct experiments.

The TouchDevelop editor is syntax directed; this means that statements, such as conditionals and loops, are always well-formed; the user can edit the condition or the body of an `if` but cannot delete the `if` token by itself. Expressions, on the other hand, are free-form and are seen as series of tokens. The editor is thus *AST-aware*, in the sense that moves, deletions and renamings operate on AST nodes rather than mere text sections.

In this context, many changes are semantic rather than syntactic; renaming a variable merely modifies its declaration node; moving an `if` statement re-attaches a node to another part of the syntax tree. Furthermore, some changes that would be conflicts using a text merge are no longer conflicts in this new setting: renaming a variable and moving its uses are two well-separated operations (on the definition node and on the use node, respectively); similarly, moving a `while` loop and reordering statements in its body are two non-conflicting operations (one is about moving a node, while another one is about reordering the children of said node).

```
function incr(x: number) {
  var y := x + 1;
  return y
}
```

Figure 1. Sample TouchDevelop program

Our merge algorithm is designed to be conflict-free; the user is never presented with the fearful three-way merge-resolution window. This does mean we make arbitrary decisions; the main use-case for our algorithm, though, is real-time collaboration, where visual clues and inter-personal coordination, along with the non-conflicting nature of most refactoring operations, limit the opportunities for arbitrary decisions.

2. Overview of the AST merge algorithm

The algorithm that we implemented for TouchDevelop operates on a tree representation of the program. A sample program is shown in Figure 1; its corresponding tree representation is shown in Figure 2.

We tag each AST node with a unique identifier (m, n, ...): the editor must preserve these identifiers when moving nodes around. A reference to a variable (such as x) is translated to a reference to the corresponding definition node.

Some nodes may be reordered (*e.g.* two parameters of a function); some other nodes must not be mixed (*e.g.* a function parameter and a statement in the function's body). To that effect, we introduce placeholder nodes (dotted) whose main purpose is to separate unrelated nodes. Finally, we track the *order* of the nodes ($x + 1$ is not the same as $1 + x$) by remembering the sibling of each node (thin dotted arrows).

The algorithm takes three trees as an input: the original tree T_O, "my" tree T_A, and "their" tree T_B. From a high-level perspective, the algorithm works as follows.

- We first decide which nodes to keep (Figure 2); seeing our trees as a set of nodes, we take nodes added by A and B, as well as the original nodes that *both* A and B agreed to keep.
- Then, we "re-wire" the nodes by attaching them to their parents; we follow the "parenting" choices from A; for nodes that are still not reattached after that, we follow the choices from B. This operation may create cycles; we provide a cycle-breaking procedure.
- Once each node is attached to the right parent, we decide on an order for the siblings; that is, we give a total order for the

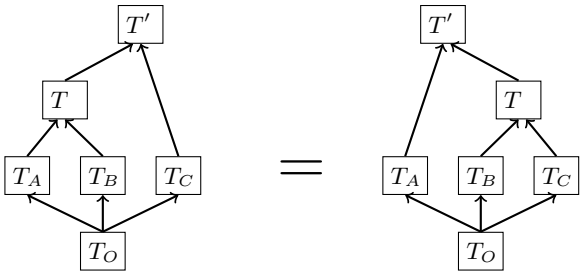

Figure 5. Associative property

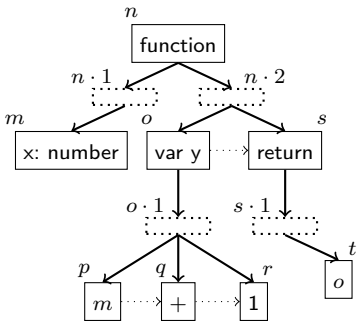

Figure 2. Sample tree

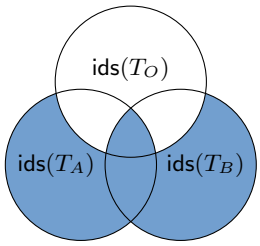

Figure 3. Determining the nodes of the merged tree

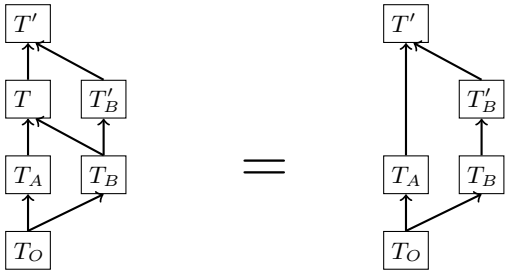

Figure 4. Cumulative property

transversal dotted arrows. We use a diff3-like algorithm on the sequence of siblings.

- Finally, we merge *properties* of the node, such as the actual name (rather than the internal id) of a variable binding.

The merge algorithm we describe enjoys the following properties.

- If both sides leave a piece of code untouched; it remains untouched in the output tree.
- If one of the two sides changes a piece of code, the output tree contains the change.
- If both sides agree on a change, then the output tree contains an identical change.
- If both sides argue about a change, then A wins.

Furthermore, in the case that both sides operate on distinct parts of the AST (we say the edits are *well-separated*), then the merge enjoys further properties. The first one is the cumulative property (Figure 4); in version-control lingo, if B is doing a feature branch, merging it into the master branch A once or twice has the same net effect. The second one is associativity (Figure 5).

2.1 An example

Consider the sample tree from Figure 2. The following changes are well-separated and enjoy the properties above:

- changing `x+1` for `1+x` (operates on the subtree of o) and renaming `x` into `z` (operates on the definition node m);
- moving `x+1` into the `return` statement to obtain `return x+1` (this is a change in the parent-of relation), and deleting the definition of `y` (this deletes the node o).

2.2 Handling conflicts

When a true conflict happens, an arbitrary decision is made: we favor A, that is, "our" changes. The rationale is that when one performs a change, discarding the change would be confusing

Our user interface provides a variety of mitigation mechanisms to avoid the true situation where people are "fighting" over a piece of code. We show placeholders in the code that indicate the location of each user; we offer a chat area so that people can coordinate; we offer a way to disable synchronization temporarily to polish changes locally before pushing them to other users.

3. Generalizing this approach to other languages

Our method is capable of dealing with well-formed AST fragments as well as unstructured series of tokens. TouchDevelop uses the former to represent statements, and the latter for expressions. This is not a definitive choice: an editor for a general-purpose language may choose to represent syntactically correct fragments of a program as an AST, and in-progress, syntactically invalid fragments as a mere series of tokens.

Indeed, we claim that this approach is generalizable to a general-purpose language, as long as the editor can parse the program being edited and transparently tag AST nodes with an identifier. Naturally, this requires non-trivial support from the editor: renaming a variable should operate on the definition node; copy/pasting should preserve identifiers whenever possible; pretty-printing should make sure that AST changes received from other participants are beautifully integrated in the existing editing buffer while not disrupting the rest of the programming experience.

References

[1] T. Ball, S. Burckhardt, J. de Halleux, M. Moskal, J. Protzenko, and N. Tillmann. Beyond Open Source: The TouchDevelop cloud-based integrated development and runtime environment. In *Proceedings of Second International Conference on Mobile Software Engineering and Systems*, MOBILESoft 2015, 2015. To appear.