Towards Structural Version Control

Yin Wang

You know, it's always safe to put "*Towards*" in the title when you haven't done much ;-)

Q: What's the best way to solve HARD problems?

A: Don't solve them. Make them DISAPPEAR.

This often just requires a slight change of DESIGN.

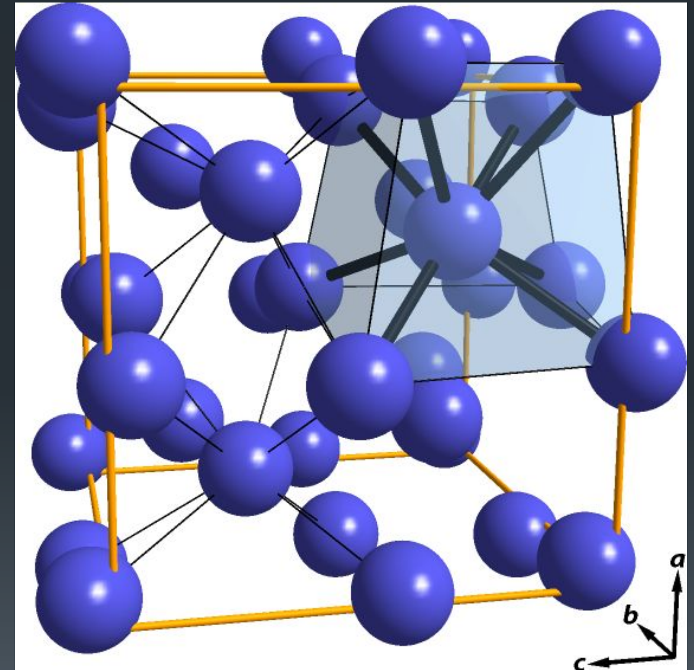
Introducing "Structural Programming"

Disambiguate:
Structural Programming
not Structured Programming

The idea has been decades old

Lambda calculus is even older

"What goes around comes around"

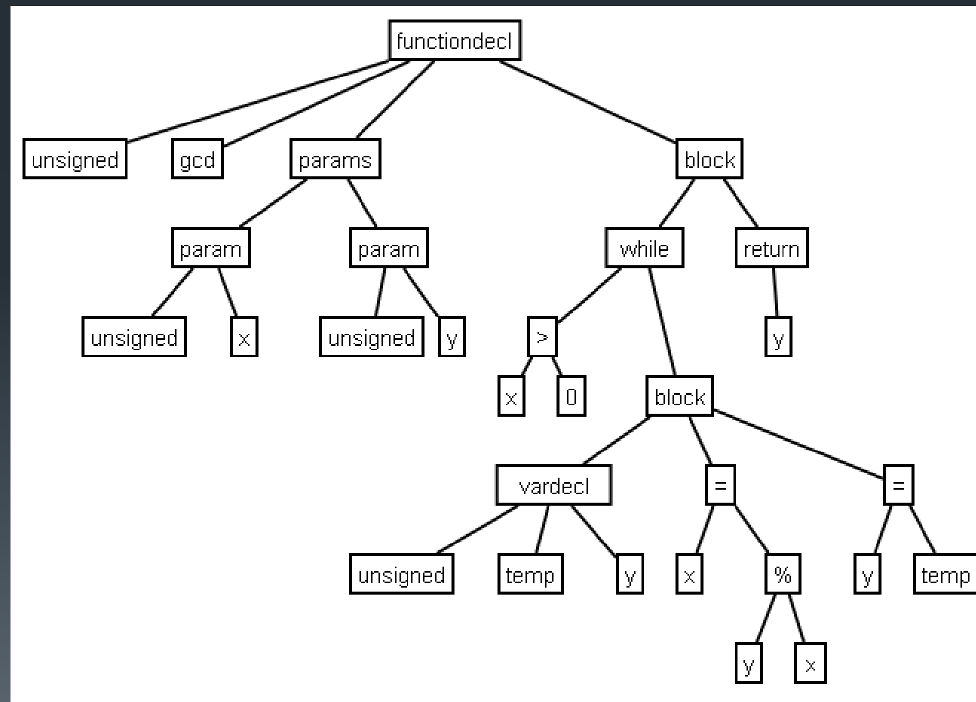


Outline

- Structural Editing (other people's work)
- Structural Comparison (my work)
- Structural Version Control (vaporware)

Programs are data structures

- Usually called “parse tree” or “AST” (abstract syntax tree)



Data structures are usually *encoded* as text

```
function factorial(n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

keywords, delimeters

The encoding scheme is called syntax

Parsers

- A parser is a *decoder* from text to data structures
- Parsers are tricky to write and hard to debug

We need parsers because we *encode* programs into text!

Why text?

- Write programs that do one thing and do it well
- Write programs to work together
- Write programs to handle *text streams*, because that is *a universal interface*

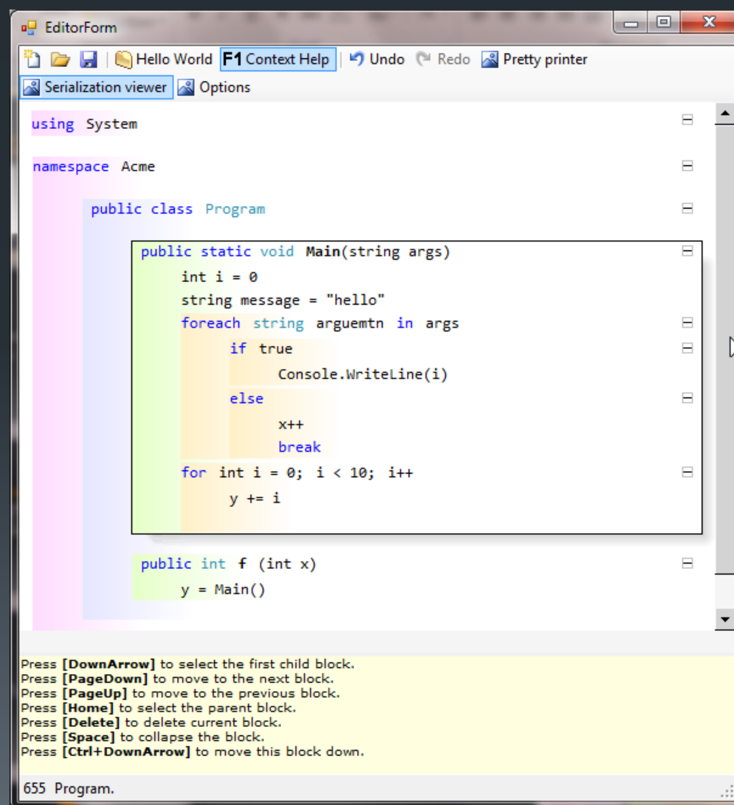
A universal interface =/= THE universal interface

Text is an inconvenient universal interface

- Data has different types: String, Int, records, functions, …
- Text is just one type: String
- Why should we encode all other types into strings?

Programming without syntax

(demo: Kirill Osenkov's editor prototype)



See also:

- MPS (JetBrains)
- Intentional Software
- Software Factories (Microsoft)
- paredit-mode (Emacs)

Potentials of Structural Editing

- *Easily extensible to ALL* programming languages
- Semantics-aware context help (limit number of choices)
- Unable to write ill-formed / ill-typed programs
- Efficient transformations and refactorizations
- Pictures, math formulas together with programs
- Incremental compilation at fine granularity
- Version control at fine granularity

New problems

- How do we display code in emails?
  - Need to standardize a data format for parse trees
  - Easy. We have been making standards all the time: ASCII, Unicode, JPEG …
- How do we do version control?
  - No more text means no more “lines”
  - … means most VC tools will stop working!

Outline

- Structural Editing (other people's work)
- Structural Comparison (my work)
- Structural Version Control (vaporware)

ydiff: Structural Diff

- **Language-aware**
- **Refactor-aware**
- **Format-insensitive**
- **Comprehensible output**
- **Open-source**

http://github.com/yinwang0/ydiff

[Demo]()

Ingredients

- Structural comparison algorithms
- Generalized parse tree format
- Home-made parser combinator library
- Experimental parsers for JavaScript, C++, Scheme, …

Parsec.ss: Parser Combinator Library in Scheme

- Modeled similar to Parsec.hs
- Macros make parsers look like BNF grammars (“DSL”)
- Left-recursion detection (direct / indirect)

```
(::= $function-definition 'function
     (@or (@... (@? $modifiers) $type
                (@= 'name $identifier ) $formal-parameter-list)

          (@... (@= 'name $identifier ) $formal-parameter-list))
     (@? $initializer)
     $function-body)
```

Left-recursion Detection

```
;;---------- indirect left-recursion -------------
;;
(::= $left1 'left1
  (@seq $left2 ($$ "ok")))

(::= $left2 'left2
  (@or (@seq $left1 ($$ "ok"))
       ($$ "ok")))

($eval $left1 (scan "ok ok"))
```

```
apply-check: left-recursion detected
parser: #<procedure:$left2>
start token: #(struct:Token 0 2 ok)
stack trace: #<procedure:...\ydiff\parsec.ss:364:4>
#<procedure:$left1>
#<procedure:...\ydiff\parsec.ss:364:4>
#<procedure:...\ydiff\parsec.ss:399:4>
#<procedure:$left2>

>
-1\**-  *scheme*    Bot (6      erior Scheme
```

problem token

trace

Generalized Parse Tree Format

```
Handle<String> Shell::ReadFile(const char* name) {
  int size = 0;
  char* chars = ReadChars(name, &size);
  if (chars == NULL) return Handle<String>();
}
```

```
(Expr 0 235 'function (list
  (Expr 0 14 'type (list
    (Expr 0 6 'identifier (list (Expr 0 6 'id (list (Token 0 6 "Handle")))))
      (Expr 6 14 'type-parameter
       (list (Expr 7 13 'type
         (list (Expr 7 13 'identifier
           (list (Expr 7 13 'id (list (Token 7 13 "String"))))))))))
     (Expr 15 30 'name
       (list
         (Expr 15 30 'identifier
           (list
             (Token 15 20 "Shell")
             (Token 20 22 "::")
             (Expr 22 30 'id (list (Token 22 30 "ReadFile")))))))
```

Parsers Built

- C++ (596 lines, incomplete, most of C++)
- JavaScript (464 lines, complete, may still contain bugs)
- (Scheme

```
(:: $open
     (@or (@~ "(") (@~ "[")))

(:: $close
     (@or (@~ ")") (@~ "]")))

(:: $non-parens
     (@and (@! $open) (@! $close)))

(::= $parens 'sexp
     (@seq $open (@* $sexp) $close))

(:: $sexp
    (@+ (@or $parens $non-parens)))

(:: $program $sexp)
```

)

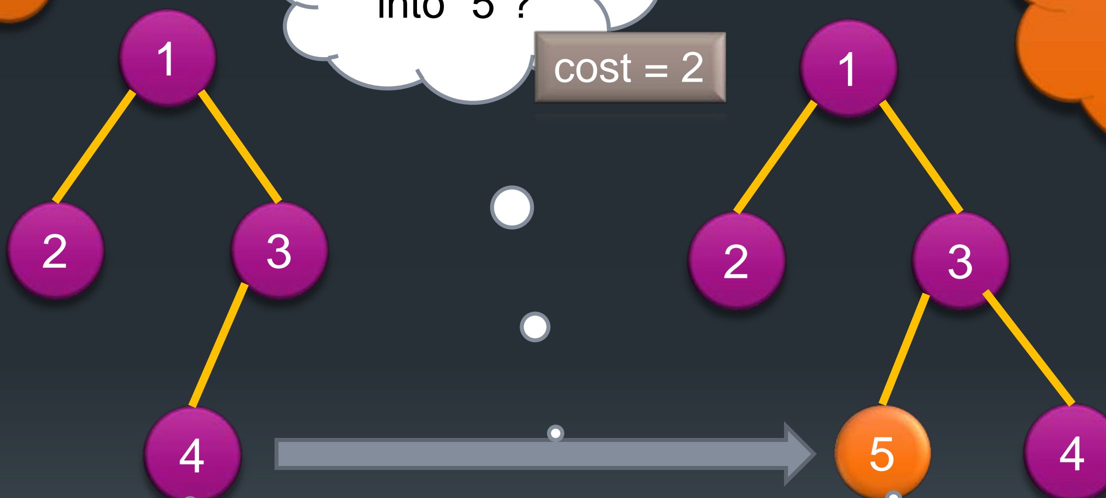
Key Algorithms

- Tree Editing Distance (TED)
- Move Detection
- Substructure Extraction

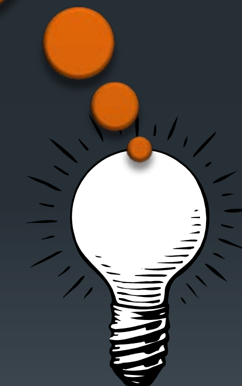
Tree Editing Distance

Node -> Node –> [Change]

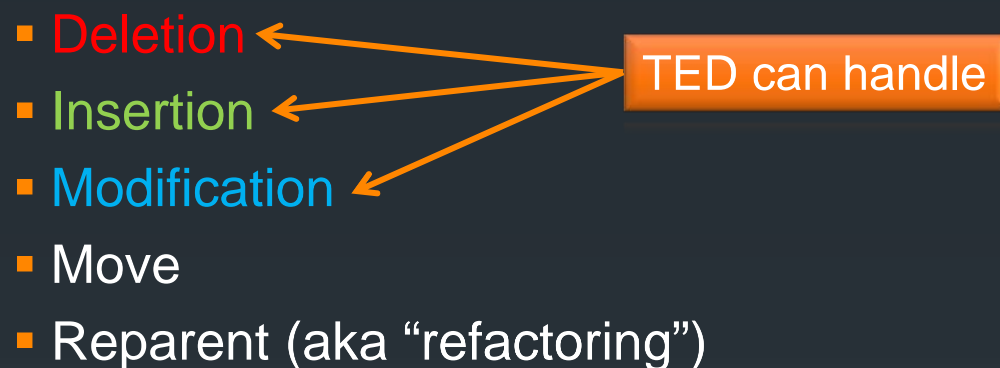
All three cases are equally possible



Minimize the number of changes that make the two trees equal

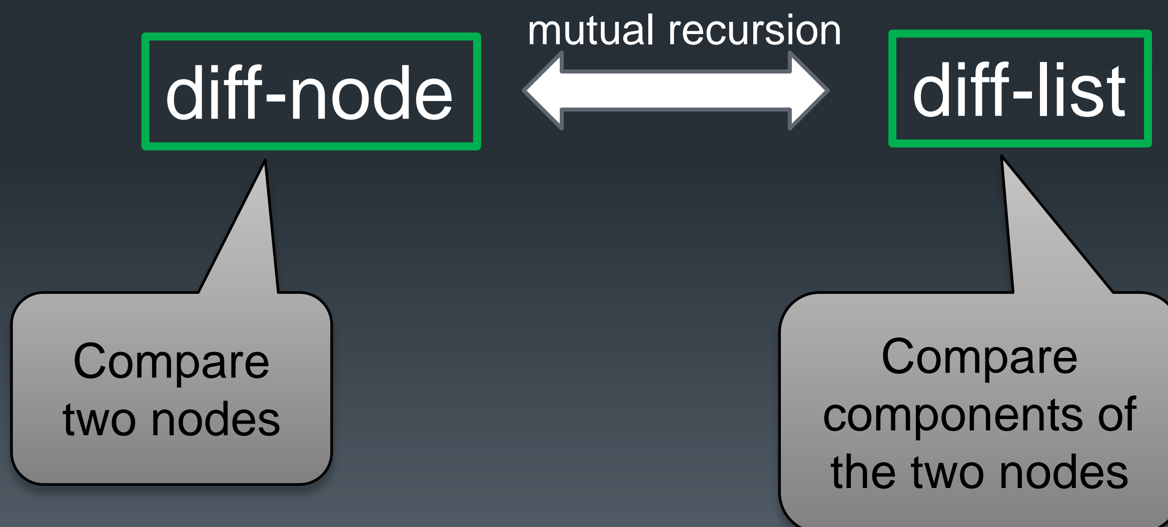


Types of Changes



Observation: allowing modification generates incomprehensible results

Tree Editing Distance with Recursion



diff-node :: Node -> Node -> [Change]

```
(cond
 [(hash-get *diff-hash* node1 node2)
  => (lambda (cached)
       (values (car cached) (cdr cached)))]
 [(and (Char? node1) (Char? node2))
  (diff-string (char->string (Char-text node1))
               (char->string (Char-text node2))
               node1 node2)]
 [(and (Str? node1) (Str? node2))
  (diff-string (Str-text node1) (Str-text node2) n
 [(and (Comment? node1) (Comment? node2))
  (diff-string (Comment-text node1) (Comment-text node                1 node2)]
 [(and (Token? node1) (Token? node2))
  (diff-string (Token-text node1) (Token-text node     node1 node2)]
 [(and (Expr? node1) (Expr? node2)
       (eq? (get-type node1) (get-type node2)))
  (letv ([(m c) (diff-list (Expr-elts node1) (Expr-elts node2) move?)])
    (try-extract m c))]
 [(and (pair? node1) (not (pair? node2)))
  (diff-list node1 (list node2) move?)]
 [(and (not (pair? node1)) (pair? node2))
  (diff-list (list node1) node2 move?
 [(and (pair? node1) (pair? node2))
  (diff-list node1 node2 move?)]
 [else
  (letv ([(m c) (total node1
    (try-extract m c)))))
```

dispatch on node types

memoization

base cases

only compare nodes of the same type

compare subnodes

substructure extraction from the changes

diff-list :: [No le] -> [No ange]

```
(define guess
  (lambda (ls1  ls2)
    (letv ([(m0 c0)  (diff-node (car ls1)  (car  2)  move?)]
           [(m1 c1)  (diff-list1 table (cdr ls   (cdr ls2)  move?)
           [cost1 (+ c0 c1)])
      (cond
       [(or (same-def? (car ls1) (car ls2))
            (and (not (different-def? (car ls1) (car ls2)))
                 (similar? (car ls1) (car ls2) c0)))
        (memo (append m0 m1) cost1)]
       [else
        (letv ([(m2 c2) (diff-list1 table (cdr ls1) ls2  move?)]
               [(m3 c3) (diff-list1 table ls1 (cdr ls2) move?)]
               [cost2 (+ c2 (node-size (car ls1)))]
               [cost3 (+ c3 (node-size (car ls2)))])
          (cond
           [(<= cost2 cost3)
            (memo (append (del (car ls1)) m2) cost2)]
           [else
            (memo (append (ins (car ls2)) m3) cost3)]))]))))
```

compare head nodes

shortcut: same definition or unchanged

Otherwise, two choices:
delete head1
or
insert head2

pick the branch with lower cost

Move Detection

- Some moved node can be detected by simple pairwise comparison between DELETED and INSERTED change sets.

```
@@ -1,11 +1,11 @@
-(define-syntax run*
-  (syntax-rules ()
-    ((_ (x) g ...) (run #f (x) g ...))))
-
 (define-syntax rhs
   (syntax-rules ()
     ((_ x) (cdr x))))

+(define-syntax run*
+  (syntax-rules ()
+    ((_ (x) g ...) (run #f (x) g ...))))
+
 (define-syntax lhs
   (syntax-rules ()
     ((_ x) (car x))))
```

normal diff (Git)

```
(define-syntax run*                                   (define-syntax rhs
  (syntax-rules ()                                      (syntax-rules ()
    ((_ (x) g ...) (run #f (x) g ...))))                  ((_ x) (cdr x))))

(define-syntax rhs                                    (define-syntax run*
  (syntax-rules ()                                      (syntax-rules ()
    ((_ x) (cdr x))))                                     ((_ (x) g ...) (run #f (x) g ...))))

(define-syntax lhs                                    (define-syntax lhs
  (syntax-rules ()                                      (syntax-rules ()
    ((_ x) (car x))))                                     ((_ x) (car x))))
```

ydiff

Substructure Extraction

frame: keep as a new change for further extractions

```
117
118
119 # append was moved into appendAll as an inner function
120 # with some modifications.
121 def append(ls1, ls2):
122     if (ls1 == nil):
123         return ls2
124     else:
125         return append(ls1.rest, Cons(ls1.first, ls2))
126
```

```
169 # append was moved into appendAl[…]an inner function
170 # with some modifications. append[…] is considered to
171 # be a wrapping function for appen[…].
172 def appendAll(*lists):
173
174
175
176
177
178     return foldl(append1, nil, slist(lists))
```

Outline

- Structural Editing (other people's work)
- Structural Comparison (my work)
- Structural Version Control (vaporware)

Merging in text-based version control

1 apples
2 bananas
3 cookies
4 rice

1 apples
2 bananas
3 **beer**
4 cookies
5 rice

Insert “beer” on line 3

1 apples
2 bananas
3 cookies
4 **pasta**
5 rice

Insert “pasta” on line 4

1 apples
2 bananas
3 **beer**
4 cookies
5 **pasta**
6 rice

Insert “beer” on line 3

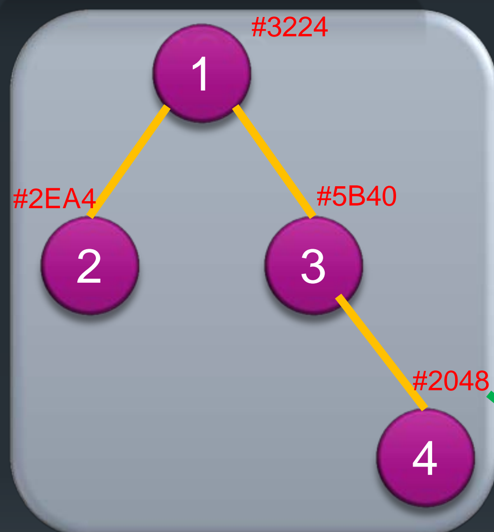
Insert “pasta” on line 5

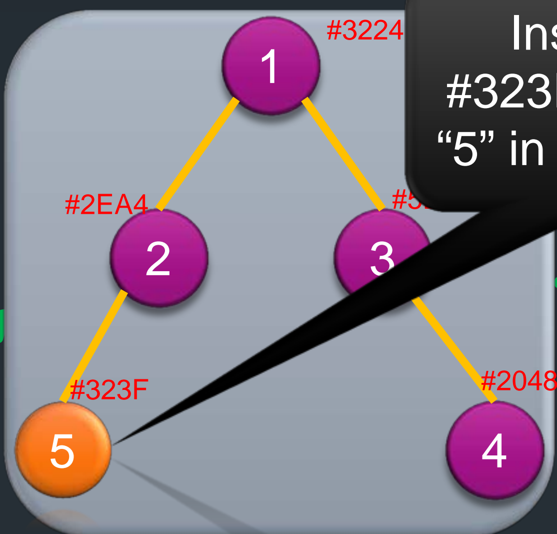
- Modifying a line of text changes the *line number* of consequent lines
- Patch that says “insert pasta to line 4” must relocate to line 5
- → Patch Theory (Darcs)

Prediction 1: merging will no longer be a problem in Structural Version Control

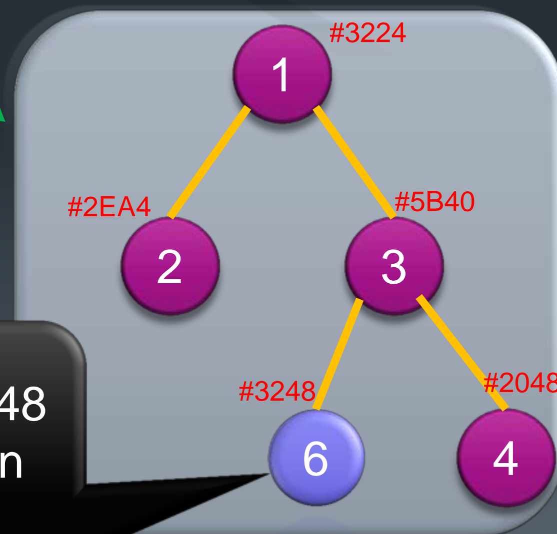
Modifying Different Nodes

Each node has a GUID

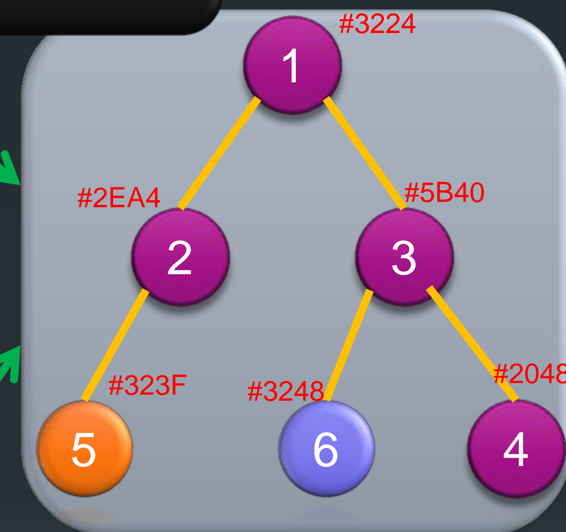




Insert node #323F containing "5" in node #2EA4



Insert node #3248 containing "6" in node #5B40

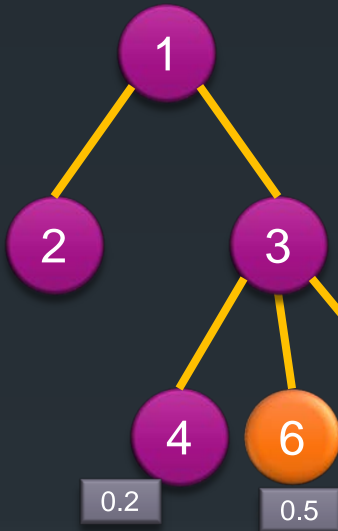


Insert node #323F containing "5" in node #2EA4
Insert node #3248 containing "6" in node #5B40

Modifying The Same Node

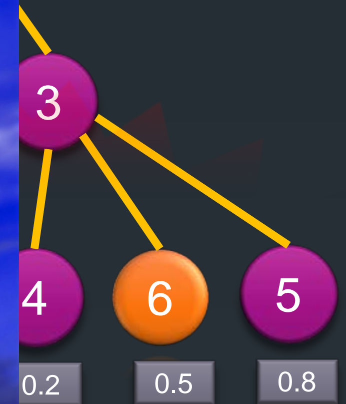
Because the real line can be infinitely divided, we can always sort the numbers into relative positions!

100% conflict-free merging!!



Insert node #2048 containing "6" in node #5B40, at position 0.5

containing "7" in node #5B40, at position 0.1

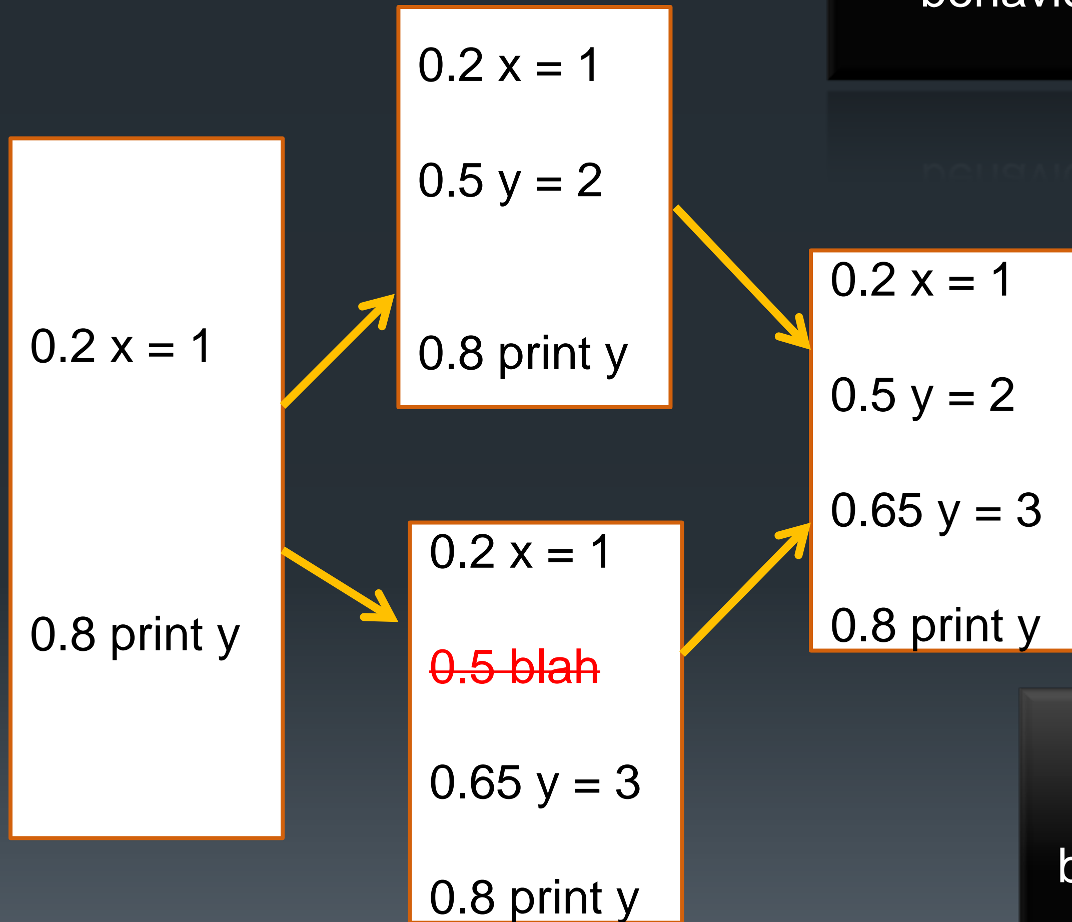


Insert node #2048 containing "6" in node #5B40, at position 0.5

Insert node #2056 containing "7" in node #5B40, at position 0.1

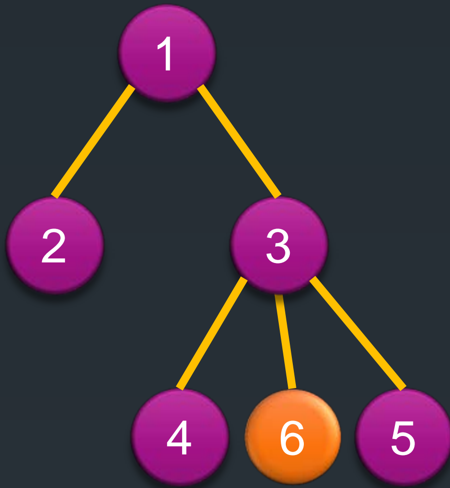
What's wrong?

All line-based VC tools have this behavior. Try it!

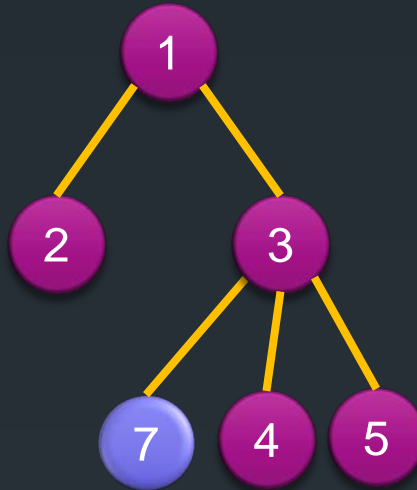


Merge succeed, but bugs introduced!

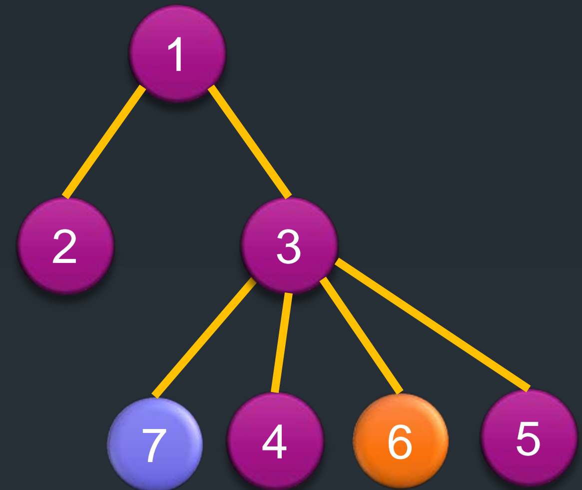
Modifying The Same Node (a more sensible way)



Insert node #2048 containing “6” in node #5B40, between #31FE and #3208



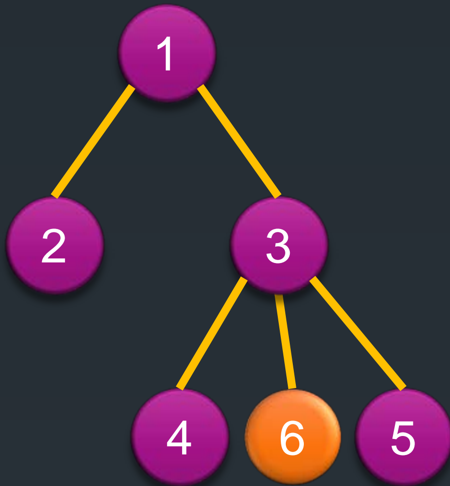
Insert node #2056 containing “7” in node #5B40, before #31FE

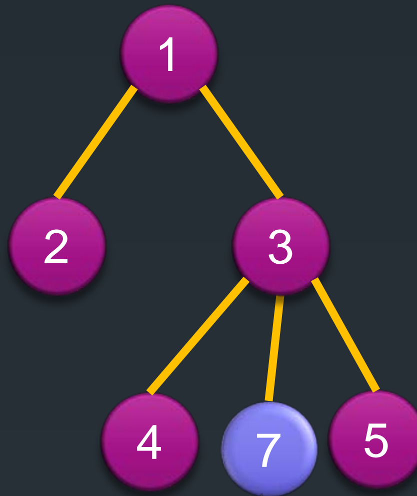


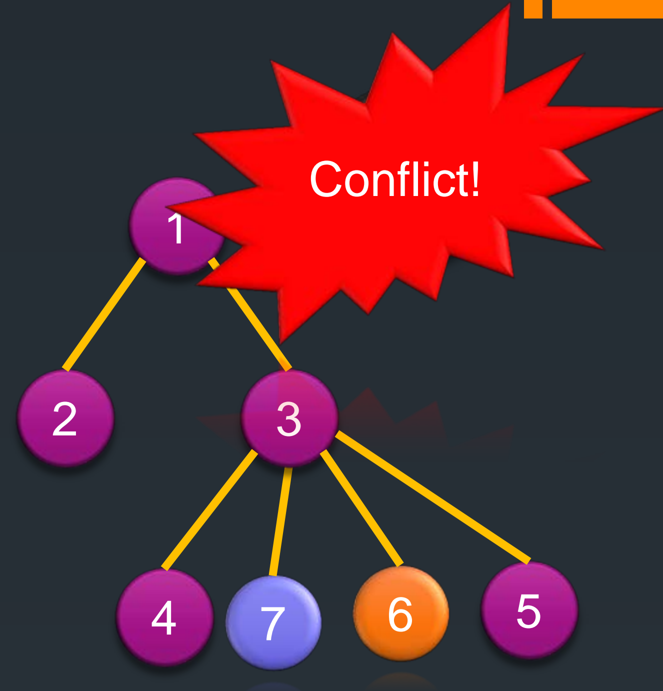
Insert node #2048 containing “6” in node #5B40, between #31FE and #3208

Insert node #2056 containing “7” in node #5B40, before #31FE

Modifying The Same Node (again)







Insert node #2048 containing “6” in node #5B40, between #31FE and #3208

Insert node #2056 containing “7” in node #5B40, between #31FE and #3208

Insert node #2048 containing “6” in node #5B40, between #31FE and #3208

Insert node #2056 containing “7” in node #5B40, between #31FE and #3208

Some observations into text-base VC tools

- *Grounds* are where programs sit on.
- Merging is hard because simultaneous edits change the grounds in different ways, but text-based VC tools don't have a *handle* on them.
- This is why Darcs uses Patch Theory, which gives us limited power for reasoning about the grounds.
- Git uses hash values to locate the grounds, but has larger granularity. Also, hash values have dependency on the contents.
- Once we have true handles on the grounds, the problem disappears.

What’s next?

- Other scenarios
- HOW MUCH and WHAT context to include in the patches?
- A *descriptive language* for patches, and a constraint solver for merging them?
- A database-like transaction system for parse tree structures?
- Let the structural editor construct the change sets?
- Generalize structural programming to natural languages?

Discussions