NAME

Algorithm::Merge - Three-way merge and diff

SYNOPSIS

```
use Algorithm::Merge qw(merge diff3 traverse_sequences3);

@merged = merge(\@ancestor, \@a, \@b, {
CONFLICT => sub { }
});

@merged = merge(\@ancestor, \@a, \@b, {
CONFLICT => sub { }
}, $key_generation_function);

$merged = merge(\@ancestor, \@a, \@b, {
CONFLICT => sub { }
});

$merged = merge(\@ancestor, \@a, \@b, {
CONFLICT => sub { }
}, $key_generation_function);

@diff = diff3(\@ancestor, \@a, \@b);

@diff = diff3(\@ancestor, \@a, \@b, $key_generation_function);

$diff = diff3(\@ancestor, \@a, \@b);

$diff = diff3(\@ancestor, \@a, \@b, $key_generation_function);

@trav = traverse_sequences3(\@ancestor, \@a, \@b, {
# callbacks
});

@trav = traverse_sequences3(\@ancestor, \@a, \@b, {
# callbacks
}, $key_generation_function);

$trav = traverse_sequences3(\@ancestor, \@a, \@b, {
# callbacks
});

$trav = traverse_sequences3(\@ancestor, \@a, \@b, {
# callbacks
}, $key_generation_function);
```

USAGE

This module complements Algorithm::Diff by providing three-way merge and diff functions.

In this documentation, the first list to `diff3`, `merge`, and `traverse_sequences3` is called the 'original' list. The second list is the 'left' list. The third list is the 'right' list.

The optional key generation arguments are the same as in Algorithm::Diff. See Algorithm::Diff for more information.

diff3

Given references to three lists of items, `diff3` performs a three-way difference.

This function returns an array of operations describing how the left and right lists differ from the original list. In scalar context, this function returns a reference to such an array.

Perhaps an example would be useful.

Given the following three lists,

```
original: a b c e f h i k
left: a b d e f g i j k
right: a b c d e h i j k

merge: a b d e g i j k
```

we have the following result from diff3:

```
[ 'u', 'a', 'a', 'a' ],
[ 'u', 'b', 'b', 'b' ],
[ 'l', 'c', undef, 'c' ],
[ 'o', undef, 'd', 'd' ],
[ 'u', 'e', 'e', 'e' ],
[ 'r', 'f', 'f', undef ],
[ 'o', 'h', 'g', 'h' ],
[ 'u', 'i', 'i', 'i' ],
[ 'o', undef, 'j', 'j' ],
[ 'u', 'k', 'k', 'k' ]
```

The first element in each row is the array with the difference:

```
c - conflict (no two are the same)
l - left is different
o - original is different
r - right is different
u - unchanged
```

The next three elements are the lists from the original, left, and right arrays respectively that the row refers to (in the synopsis, these are `@ancestor`, `@a`, and `@b`, respectively).

merge

Given references to three lists of items, `merge` performs a three-way merge. The `merge` function uses the `diff3` function to do most of the work.

The only callback currently used is `CONFLICT` which should be a reference to a subroutine that accepts two array references. The first array reference is to a list of elements from the left list. The second array reference is to a list of elements from the right list. This callback should return a list of elements to place in the merged list in place of the conflict.

The default `CONFLICT` callback returns the following:

```
q{<!-- ------ START CONFLICT ------ -->},
(@left),
q{<!-- ---------------------------- -->},
(@right),
q{<!-- ------ END CONFLICT ------ -->},
```

traverse_sequences3

This is the workhorse function that goes through the three sequences and calls the callback functions.

The following callbacks are supported.

NO_CHANGE
: This is called if all three sequences have the same element at the current position. The arguments are the current positions within each sequence, the first argument being the current position within the first sequence.

A_DIFF
: This is called if the first sequence is different than the other two sequences at the current position. This callback will be called with one, two, or three arguments.

  If one argument, then only the element at the given position from the first sequence is not in either of the other two sequences.

  If two arguments, then there is no element in the first sequence that corresponds to the elements at the given positions in the second and third sequences.

  If three arguments, then the element at the given position in the first sequence is different than the corresponding element in the other two sequences, but the other two sequences have corresponding elements.

B_DIFF
: This is called if the second sequence is different than the other two sequences at the current position. This callback will be called with one, two, or three arguments.

  If one argument, then only the element at the given position from the second sequence is not in either of the other two sequences.

  If two arguments, then there is no element in the second sequence that corresponds to the elements at the given positions in the first and third sequences.

  If three arguments, then the element at the given position in the second sequence is different than the corresponding element in the other two sequences, but the other two sequences have corresponding elements.

C_DIFF
: This is called if the third sequence is different than the other two sequences at the current position. This callback will be called with one, two, or three arguments.

  If one argument, then only the element at the given position from the third sequence is not in either of the other two sequences.

  If two arguments, then there is no element in the third sequence that corresponds to the elements at the given positions in the first and second sequences.

  If three arguments, then the element at the given position in the third sequence is different than the corresponding element in the other two sequences, but the other two sequences have corresponding elements.

CONFLICT
: This is called if all three sequences have different elements at the current position. The three arguments are the current positions within each sequence.

BUGS

Most assuredly there are bugs. If a pattern similar to the above example does not work, send it to <jsmith@cpan.org> or report it on <http://rt.cpan.org/>, the CPAN bug tracker.

Algorithm::Diff's implementation of `traverse_sequences` may not be symmetric with respect to the input sequences if the second and third sequence are of different lengths. Because of this, `traverse_sequences3` will calculate the diffs of the second and third sequences as passed and swapped. If the differences are not the same, it will issue an 'Algorithm::Diff::diff is not symmetric for second and third sequences...' warning. It will try to handle this, but there may be some cases where it can't.

SEE ALSO

Algorithm::Diff.

AUTHOR

James G. Smith, <jsmith@cpan.org>

COPYRIGHT