社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ソースコードの木構造を考慮した差分計算を用いる版管理システムの提案

早瀬　康裕† 　松下　誠† 　井上　克郎†

† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻　〒 560–8531 大阪府豊中市待兼山町 1–3
E-mail: †{y-hayase,matusita,inoue}@ist.osaka-u.ac.jp

あらまし　オープンソースソフトウェア開発では，ソースコードなどファイルに対する変更履歴を記録する版管理システムが用いられる．この時，版管理システム上の一つのファイルを，同時に複数人が編集することによって，複数の変更をマージする作業が発生する，マージが必要になる．しかし，版管理システムが提供するマージ機能は，テキストの行を最小単位とした操作に基づいており，ソースコードに適用するものとしては適切ではないという問題があった．そこで本研究では，ソースコードの持つ木構造を考慮した差分計算をマージの際に用いることによって，複数の変更をより適切にマージする方法の提案を行う．また，本方法を実際の版管理システム上に構築する方法について考察を行う．

キーワード　木構造, 差分計算, 版管理システム

Revision Control System Using Delta Calculation Based on Tree Structure of Source Code

Yasuhiro HAYASE†, Makoto MATSUSHITA†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1–3, Machikaneyama-cho, Toyonaka-shi, Osaka 560–8531, Japan
E-mail: †{y-hayase,matusita,inoue}@ist.osaka-u.ac.jp

Abstract　In opensource software development, developers use a revision control system which records change histories over files such as source codes. In such case, two or more developers simultaneously edit one file on a revision control system, so merging of changes in source code are needed. However, the merge function provided by a revision control system is not appropriate to apply to a source code, because its operation unit is a line of a text. Then, we propose a method of merging more appropriately two or more changes using delta calculation which considers tree structure of source code, in this research. Moreover, we consider how to build this method on an actual revision control system.

Key words　Tree Structure, Delta Calculation, Revision Control System

1. はじめに

ソフトウェアの開発において，版管理システムは重要な役割を果たしている．版管理システムとは，ソースコードやドキュメント等のソフトウェアの開発履歴を保存するシステムである．

版管理システムの代表的な機能の一つに，独立して行われた複数の変更を統合するマージ機能がある．既存の版管理システムでは，マージ処理を行単位で行っていた．しかし行を単位としたマージでは，本来衝突すべきでない変更点が衝突したり，本来衝突とすべき変更を見逃すなどといった問題がある．このような問題のために，開発者が手作業でマージを行う必要があったり，問題のあるマージ結果を得たりしてしまっていた．

また，近年，インターネットの普及により，オープンソース [1] ソフトウェア開発が広まっている．オープンソースソフトウェア開発では，一つのファイルを，複数の開発者が同時に編集することが，頻繁に行われている．同時に編集されたファイルは，それぞれの変更をマージする必要がある．その結果，マージが頻繁に必要となり，マージの精度が低いことが，オープンソースソフトウェア開発で重要な問題となっている．

ソースコードのマージをより正確に行うため方法として，プログラミング言語の構文を理解したマージ処理が考えられる．しかし，プログラミング言語は数多く存在するため，それぞれ

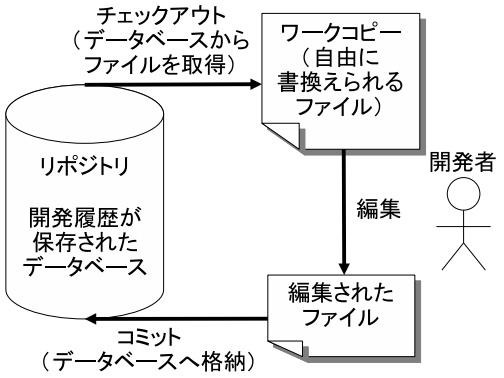

図 1　版管理システム

について，精度の高いマージシステムを作るのは困難である．そこで，ソースコードをツリー構造を持った中間言語に変換し，中間言語に対してツリーの差分計算やツリーのマージ処理を行うことで，言語依存部分とマージ処理の部分を分離する手法の提案を行う．また，本方法を実際の版管理システム上に構築する方法について考察を行う．

以下，2 節では，オープンソースソフトウェア開発と版管理システムのそれぞれについて説明し，既存の版管理システムの問題を示す．3 節では，問題の解決法として，ソースコードの構造を考慮したマージシステムを提案し，4 節ではそのシステムの実装について説明する．5 節では提案するシステムを用いた場合に解決する問題の例を示す．最後に 6 節で本研究のまとめと，今後の課題について述べる．

2. オープンソースソフトウェア開発と版管理システム

2.1 オープンソースソフトウェア開発

オープンソースソフトウェア開発は近年急速に普及し，多くの有用なソフトウェアが開発されている．オープンソースソフトウェア開発は，多人数が自由に参加することが出来るという特徴がある．

一般的なオープンソースソフトウェア開発では，以下の道具を用いて開発を行う．まず，プロジェクトの成果物などを公開するためのウェブページがある．開発の方向性や，ソフトウェアの問題点などの議論にはメーリングリストを用いている．そして，ソースコードやドキュメントの変更履歴を保存し，多人数でソースコードを編集するために，版管理システムを用いている．

2.2 版管理システム

CVS [2] に代表される版管理システムは，ソースコードやドキュメント等のファイルの，変更履歴を保存するシステムである．

図 1 に，版管理システムを用いた開発を簡略化した図を示す．版管理システムにはリポジトリと呼ばれるデータベースがあり，開発対象のファイルは全てリポジトリに保存される．開発者がファイルを編集する場合，リポジトリからファイルのコピーを取得する．この操作をチェックアウトと呼び，取得したファイルはワークコピーと呼ばれる．開発者はワークコピーを編集した後に，リポジトリにファイルを格納する．格納する操作をコミットと呼ぶ．新しいファイルが格納されても，リポジトリから古いファイルが失われることはなく，古いファイルをいつでも参照することが出来る．版管理システムを使ったソフトウェア開発は，この作業を繰り返す事で行われる．

また，版管理システムは，複数人によるソフトウェア開発をサポートしている．開発者がファイルをチェックアウトする際に，他の開発者がそのファイルをチェックアウトしていても良く，取得したワークコピーを自由に編集することが出来る．

このようにして，複数人によって同時並行的にファイルの編集が行われた場合，それぞれの開発者が自分の編集結果だけをコミットすると，最後にコミットしたファイル以外に行われた編集が失われてしまう．このため，複数人が行った変更を 1 つにまとめたファイルが欲しい．このファイルを得る処理を，マージと呼ぶ．マージは，版管理システムにより自動的に行われる場合と，人間の手による解決が必要になる場合とがある．人間の手による解決が必要になることを，変更点の衝突と呼ぶ．

2.3 問 題 点

既存の版管理システムでは，マージを行単位で行っており，マージの時に同じ行が編集されている場合にだけ，衝突として開発者に解決を求める．これにより，不要な衝突を引き起こしたり，不正なマージ結果を得たりするなどと言う問題があった．以下，それぞれについて説明する．

2.3.1 不要な衝突

編集されている行が同じである場合，本来はマージ可能な変更を，衝突としてしまう．

例えば，開発者 A と B が同じファイルをチェックアウトして，編集しているとする．そのファイルには以下のような行があった．

```
int refs;
```

開発者 A は以下のように，初期値を設定する変更を加え，コミットした．

```
int refs=0;
```

開発者 B は A の変更を知る前に，以下のように変数についてのコメントを追加し，コミットをしようとしたとする．

```
int refs; /* reference count */
```

開発者 A と B は同じ行を編集しており，版管理システムはこれを衝突と見做す．今の場合，衝突は，後からコミットをした開発者 B が，解決しなければならない．

もし，変更点を正しく把握できれば，システムは，初期値とコメントの両方を含んだ，以下のようなコードを得ることが出来るはずである．

```
int refs=0; /* reference count */
```

2.3.2 不正なマージ結果

変更された行が違う場合，本来衝突すべき変更点を見逃がす場合がある．

開発者 A と B が同じファイルをチェックアウトして編集しているとする．ファイルには，以下のように変数を宣言する行が含まれていた．

```
int num, sum, avg;
```

開発者 A は変数 avg は不要であると考え，以下のように削除してコミットした．

```
int num, sum;
```

開発者 B は A がコミットしたこと知る前に，以下のように変数 avg を使う処理を追加した．

```
int num, sum, avg;
⋮
avg = num/sum;
⋮
```

B がコミットする前には， A と B の変更点をマージする必要がある．版管理システムは，マージ結果として，以下のような出力を行う．

```
int num, sum;
⋮
avg = num/sum;
⋮
```

A と B は同じ行を変更しておらず，マージの際に衝突は起こらない．しかし，このマージ結果は，宣言されていない変数 avg を利用する不正なソースコードとなってしまう．

3. ソースコードの構造を考慮したマージシステムの提案

2.3 節で述べた問題を解決するため，ソースコードの構造を考慮したマージアルシステムを提案する．マージシステムの行う処理の手順は，以下のとおりである．

（1） ソースコードを解析してツリーを作る

（2） ツリーの差分を計算する

（3） マージを行う

以下，それぞれについて説明する．

3.1 ツリーへの変換

まず，ソースコードを構文解析してツリー A を作成する．ツリーは順序木であり，子頂点の数に制限は無い．頂点には ID と，文字列の情報が記録されている．ツリーの形は，構文解析木に準じたものである．ツリーから元のソースコードに戻すことが出来るようにするために，ツリーには空白文字列やコメントを表す頂点を含める．ツリーのデータ構造は，プログラミング言語に依存しないので，マージシステムに汎用性を持たせることが出来る．

頂点に ID を付ける手順は，以下の通りである．版管理システムには，解析するソースコードの元となったバージョンのツリー B が，保存されている．B の頂点には既に ID が付けられている．まず， A と B を比較し，マッチングを計算を行う．マッチング計算とは， A, B から，それぞれの頂点集合の部分集合 A', B' と， A' から B' への全単射写像 f を見つけることである．写像 f では，頂点に格納されたデータと親子頂点や兄弟頂点から，似ていると判断された頂点同士が対応する． A の頂点のうち， A' に含まれた頂点には，対応する B' の頂点と同じ ID を付ける．A の頂点のうち，A' に含まれなかった頂点には，新しい ID を付ける．

マッチング計算には，3.2.2 節で説明する FMES アルゴリズム [3] を用いる．

次に，変数や関数を利用している頂点から，定義している頂点へのリンクを張る．

3.2 差分計算

次に，ツリーの差分を計算する．

ツリーの差分は，編集スクリプトを用いて表す．編集スクリプトは，編集操作の列であり，編集操作は以下の 4 種類である．

（1） 頂点の追加を表す insert 操作は，引数に，追加する頂点の ID，頂点が格納するデータ，親の ID，何番目の子供にするかを表す数値の 4 つを取る．

（2） 頂点の削除を表す delete 操作は，引数に，削除する頂点の ID を取る．

（3） 頂点に格納されたデータの更新を表す update 操作は，引数に，更新の対象となる頂点の ID と，その頂点に格納する新しいデータを取る．

（4） 部分木の移動を表す move 操作は，引数に，移動する部分木の根の ID，移動先の親 ID, 何番目の子供にするかを表す数値の 4 つを取る．

ツリーの編集は，対象となるツリーに上記の編集操作を適用することによって行う．編集スクリプトの先頭の操作から順に，ツリーに編集操作を適用することを，ツリーに編集スクリプトを適用すると呼ぶ．ツリー A に編集スクリプト S を適用するとツリー B が得られるとき，$A \xrightarrow{S} B$ と表すことにする．

任意のツリー A を B に変換する編集スクリプトは， insert 操作と delete 操作の組み合わせで表現可能である．例えば A の頂点を全て delete した後に，B の頂点を insert する編集スクリプトは，A を B に変換する．しかし，人間がソースコードを編集する時には， ソースコードの一部分を移動したり，文字列の一部を書換えるといった操作を行う．これらの操作を自然に表現するには，move 操作や update 操作を用いた方がよい．

ここで，あるツリー A と B があるとき， A を B に変換するスクリプトは無数に存在する．例えば， $A \xrightarrow{S} B$ である任意の S の末尾に， A にも B にも含まれない頂点を追加する操作と，その頂点を削除する操作を加えた編集スクリプト S' も，$A \xrightarrow{S'} B$ を満たす．

ツリーの差分を表すスクリプトとしては，このような無駄な

編集操作が含まれているものは望ましくない．そこで，編集スクリプトにコストを定義し，コストが最も小さいスクリプトを，差分として採用することにする．編集スクリプトのコストは，含まれる編集操作のコストの総和であり，編集操作のコストは差分計算アルゴリズムによって定める．

ツリーの差分を計算するアルゴリズムとして， xmdiff [4] と FMES [3] の 2 つを採用する．以下，それぞれについて説明する．

3.2.1 xmdiff [4]

xmdiff は外部記憶を使って差分スクリプトを計算するアルゴリズムであり，入出力回数を減らすことに主眼を置いている．時間計算量は $O(n^2)$ である．

3.2 節で定義した 4 つの編集操作のうち， xmdiff では insert, delete, update の 3 つを用いる．編集操作のコストは 0 以上の値であり，編集の種類と編集の対象によって自由に決められる．

このアルゴリズムは Edit Graph という動的計画法の表を用いる．表の縦軸と横軸には，新旧それぞれのツリーを深さ優先探索したときに，到達する順に頂点を並べる．この表の水平の辺が delete 操作に，垂直の辺が insert 操作に，斜めの辺が update 操作に対応し，辺の重みが編集操作のコストである．原点から対角までの経路のうち，重みが最小となる経路が，コスト最小の編集スクリプトに対応する．

この Edit Graph を，メモリに収まるサイズに上手く分割することで，効率良く計算を行う．

3.2.2 FMES [3]

FMES は差分計算の近似アルゴリズムである．このアルゴリズムでは，コスト最小の解を得られるとは限らないが，$e \ll n$ であるときには良い近似解を得られる．比較するツリーの間の差の大きさを e とすると，時間計算量は $O(ne + e^2)$ と表される．

編集操作は， insert, delete, update, move の 4 つを用いる．insert, delete, move のコストは 1 であり， update のコストは，書換える前後の値によって 0 から 2 の値を取る．書換える前後の値が違うほど，update 操作のコストは大きな値を取るようにする．

FMES アルゴリズムは，前後 2 つの段階に分かれており，片方だけを用いることも出来る．

1 つ目の段階は，比較する二つのツリーの間で，頂点のマッチング計算を行う．葉と内部頂点を区別し，類似度が閾値以下の頂点同士はマッチしないという仮定を置くことで，計算量を削減している．

2 つ目の段階は，マッチングの終わった木の間で編集スクリプトを計算する．この段階では厳密解を求める．

3.3 マ ー ジ

ツリー A と B の差分を表す編集スクリプト S を，A 以外のツリー C に適用し，新たなツリーを得ることを，マージと呼ぶ．

編集操作は，頂点 ID を引数に取る．しかし，C に S を適用する際に，編集操作の引数となる ID を持った頂点が C にあるとは限らない．操作する ID を持った頂点が C に無い場合，操作を適用しないか，操作対象に類似した頂点を探して代用する．類似頂点の検索は，親子の頂点や兄弟の頂点に同一のものがあるか，格納されたデータが類似しているという条件で行う．

3.4 マージ結果の選択

同じ木の差分を計算しても，アルゴリズムによって異なった編集スクリプトを出力し得る．また，マージ処理の段階でも，操作の対象となる頂点が存在しない場合の処理には選択の余地がある．このように，編集スクリプトやマージ処理の選択肢が複数存在することによって，マージ結果もまた複数存在しうる．

マージ結果が複数得られた場合は，どれが正しい結果なのかは一概には決められない．そこで，ツリーにプログラミング言語の構文制約を反映した制約を設定する．この制約への違反の数で，マージ結果を整列し，ユーザに提示して選択を促す．

4. システムの実装

本節では，システムをどのように実装するかについて説明する．

システムの実装は，既存の版管理システムである subversion [5] を拡張して行う．

4.1 システムの概要と既存のシステムとの比較

図 2 が，既存の subversion システムの概略図であり，矢印はデータの流れを表している． subversion は，リポジトリを管理するサーバと，差分計算やマージ処理を行うクライアントに分かれている．開発者は，テキストファイルを subversion クライアントに渡し，subversion クライアントは，開発者から渡されたファイルと，リポジトリに保存されたファイルを使って，テキストのマージを行ったり，差分を計算したりする．

図 3 が，拡張を施した subversion システムの概略図を示す．開発者が扱うファイルが，ソースコードである場合にだけ，特別な処理をする．システムの拡張は，クライアントだけに行い，開発者の操作と，サーバ側の処理には変更を加えない．サーバのリポジトリには，ソースコードを変換したツリーのデータが格納される．ツリーの表現には XML を用いる．

開発者が扱うファイルは全てソースコードであり， subversion クライアントがマージや差分計算を行う時には XML で行う．

4.2 チェックアウトとコミットの実装

チェックアウトとコミットを行う時の，データの流れを表したのが，図 4 である．実線がチェックアウトの時，破線がコミットの時のデータの流れを表している．チェックアウトの時には，subversion クライアントが XML からソースコードへの変換を行い，開発者に渡す．逆に，リポジトリにコミットする前には，入力されたソースコードから XML への変換を行ない，チェックアウトした時の XML と頂点の対応を計算する．

4.3 マージの実装

マージを行う時のデータの流れを表したのが，図 5 である．

まず，コミットの時と同様に，開発者の編集したソースコードを XML に変換した後に，チェックアウトした時の XML ファイルと比較することで，頂点 ID 付きの XML ファイルを作る．

次に，リポジトリに格納されている最新の XML ファイルと，

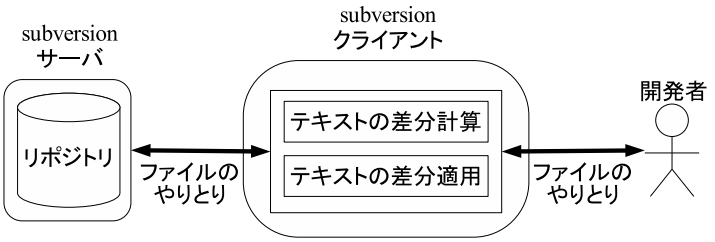

図 2　subversion の概要

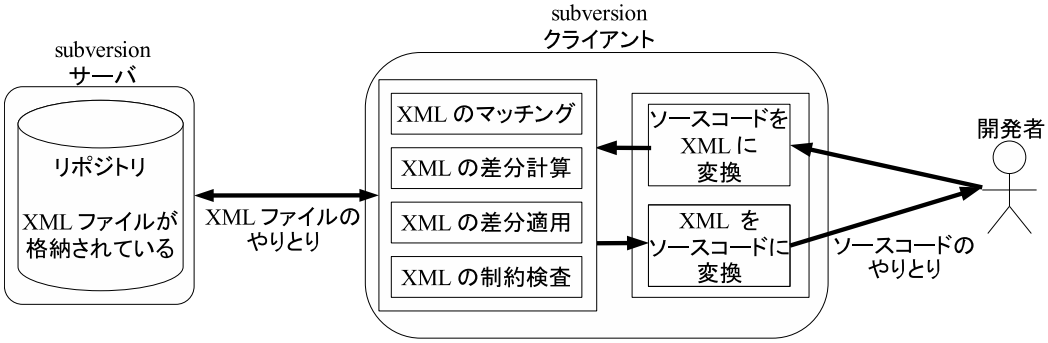

図 3　ソースコードの階層構造をサポートした subversion の概略

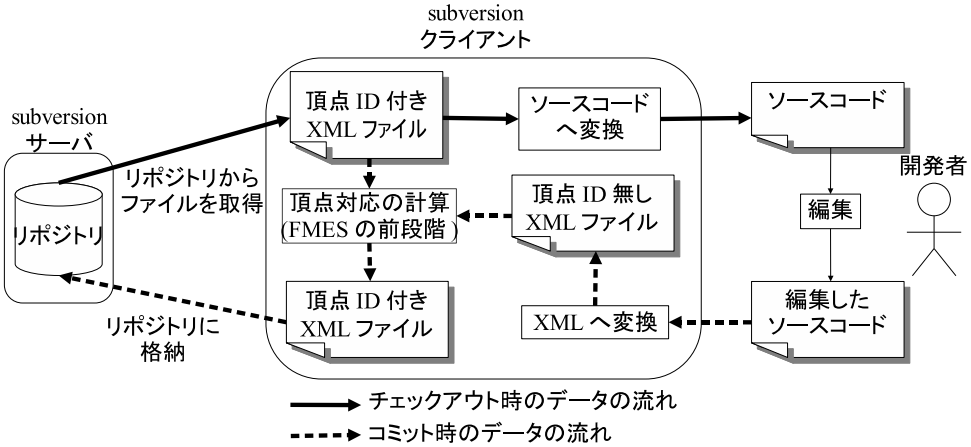

図 4　チェックアウトとコミット

チェックアウトした時の XML ファイルとの間で差分計算を行う．差分は編集スクリプトであり，複数の差分が発見されることがある．

この差分を，ソースコードから作った XML ファイルに適用する．マージ結果は XML ファイルとなり，複数の XML ファイルが得られることがある．

マージ結果が複数あった場合は，スキーマ定義にどれだけ従っているかでスコアリングし，スコアの良いものから整列して，ユーザに提示し，選択させる．

5. 具 体 例

2.3 節で述べた問題が，本システムによって，具体的にどのような形で解決するかを例として示す．

2.3.1 節で示したように，行単位のマージでは，同じ行が編集されているときには，自動的なマージ処理を行うことは出来なかった．

提案する手法では，図 6 のように，元のソースコードの行と，開発者 A の編集結果，開発者 B の編集結果をツリーにする．開発者 A と B が行った変更は，水平線と斜線で表示した頂点の追加である．開発者 A と B の変更をマージすると，マージ結果として，図の下に表示されているツリーが得られ，マージに成功する．マージ結果のツリーをソースコードに戻した結果は，図の一番下に表示されている．

このように，同じ行に対する変更であっても，ツリー上で操作した場所が異なれば，マージすることが出来る．

また， 2.3.2 節で示したように，行単位のマージでは，変数の定義と利用を考慮しておらず，宣言していない変数を使うマージ結果を生む場合があった．

提案する手法では，図 7 のように，元のソースコードと，開発者 A の編集結果，開発者 B の編集結果をツリーにする．B の編集結果では，B の追加した，変数 `avg` を利用している頂点から，変数を定義している頂点へのリンクが張られている．

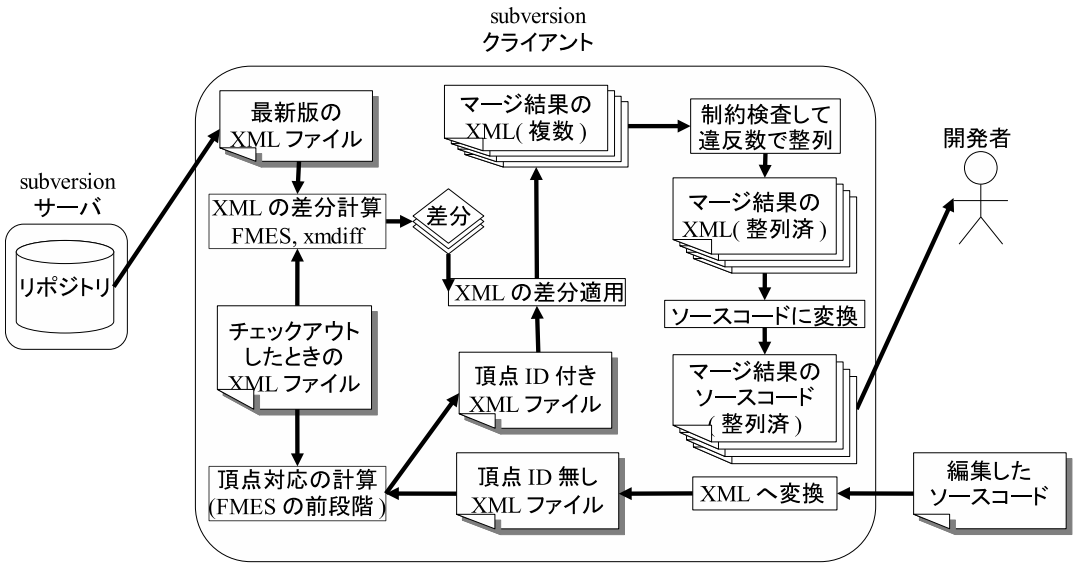

図 5　階層構造に対応したマージ

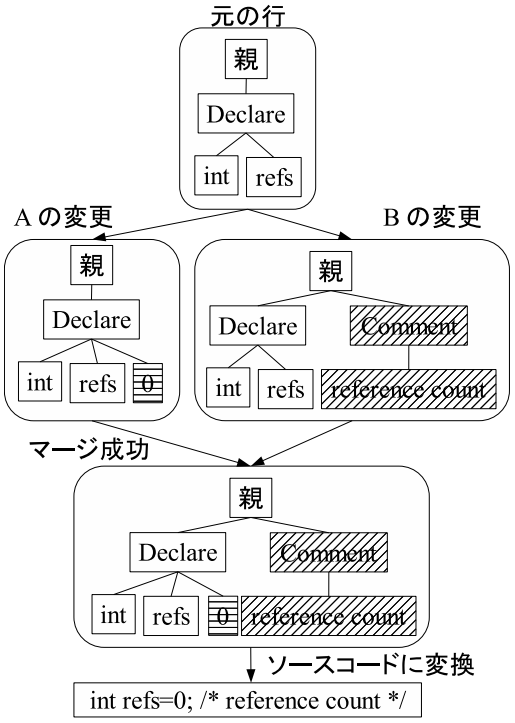

図 6　同じ行のマージに成功する場合

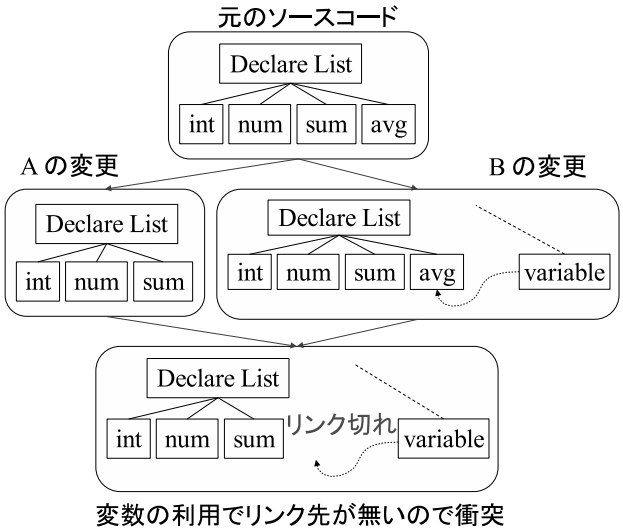

図 7　未定義変数を発見して衝突する場合

これらの変更点をまとめると，図の下のツリーになる．このツリーでは，変数へのリンクが切れており，不正なソースコードであることが分かる．

このように，提案手法では，異なる行に対する変更であっても，変数の定義と使用の関係が崩れる場合を発見することが出来る．

6. ま と め

本研究では，版管理システムのマージ機能の問題を示し，問題への対処として，ソースコードのツリー構造に従ったマージ処理を提案した．また，そのシステムの設計を示した．このシステムにより，マージを行う際の衝突を減らし，マージによって起こる問題を減らすことが出来る．

今後の課題は，システムを実装すること，実装したシステムの評価を行うことである．ソースコード以外の文書形式への対応も重要な課題である．

文　献

[1] Open Source Initiative. The open source definition. http://www.opensource.org/docs/definition.php.

[2] Concurrent version system. http://www.cvshome.org/.

[3] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 493–504, 1996.

[4] Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the Twenty-fifth International Conference on Very Large Data Bases*, pp. 90–101, Edinburgh, Scotland, U.K., 1999.

[5] subversion. http://subversion.tigris.org/.