プログラム書換えツールから独立した スタイル維持方法に関する研究

2007MI054 本田 竜二　　2007MI072 石原 浩佑　　2007MI241 立道 昂太
指導教員　野呂 昌満　　蜂巣 吉成

1　はじめに

プログラマはプログラムを記述しているとき，一度ですべて書き切ることはなく，どのように仕様を満たすか試行錯誤しながら処理を書換えている．このような書換えは，頻出する処理の関数化・マクロ化など，対象のプログラムに特化したものである．関数化・マクロ化する箇所を関数・マクロに書換える作業は，対象の箇所を探して同じ記述に書換える単純な作業の繰り返しである．単純な作業を繰り返すことはプログラマにとって単調であり，人為的なミスが混入する危険もある．

人為的なミスの混入を防ぎ，さらに開発効率の向上も促せる手段として，単純な書換え作業を自動化するプログラム書換えツール（以下，書換えツール）を利用する方法がある．対象のプログラムに特化した汎用性の低い書換えも自動化することで開発効率の削減に効果があるが，そのためには手作業よりも低コストで書換えツールを作成しなければならない [2]．しかし，実際にはツールを作成し利用するよりも手作業で書換える方がコストが低い場合が多いので，結局手作業で書換えがおこなわれてしまう．書換えツールのコストが高くなる原因としてプログラミングスタイル（以下，スタイル）維持の処理が挙げられる．書換え前に存在したコメントやインデントなどのスタイルは，書換えツールを適用した後も維持すべきである．

本研究の目的は書換えツールを低コストで作成できる環境を構築することで，プログラム全体の開発効率向上を支援することである．コストの削減を目的とした記述量削減の方法として，対象から共通する処理を探し，それを独立した再利用できる要素とする．本研究では，再利用の方法として，スタイル維持の処理を独立してツール化する方法を提案する．スタイル維持の処理を再利用可能な形で独立してツール化すれば，書換えツールにスタイル維持の処理の記述がなくなり，書換えツール作成のコストを削減することができる．

スタイル維持を独立してツール化させることで生じる問題として，スタイル維持の精度が下がる問題がある．本研究の支援対象である書換えツールは低コストで作成することが前提である．よって，スタイル維持の精度よりも書換えツール作成のコストを下げることを優先するが，スタイル維持の精度の低下は抑えたい．本研究では，スタイル維持を独立してツール化させることで生じる精度の低下を抑える方法について研究し，実際にソースプログラムに対して適用したときにスタイル維持の精度の低下にどれほど影響があるかを確認する．

2　書換えの自動化の背景と問題

2.1　書換え支援環境

書換えツールの作成を支援する環境として TEBA[3] が提案されている．TEBA は字句に構文上の種別などを付け加える字句解析環境である．これで得られた字句の並びに対して操作することで書換えの自動化を可能にしている．

TEBA 以外にも自動化の支援環境として，抽象構文木の操作を用いて書換えを自動化する環境が提案されている．しかし，TEBA 以外の支援環境の多くは前処理前のプログラムを解析して操作できるものはほとんど存在しない．一方，TEBA は抽象構文木を用いないで，可能な限り構文の情報を付加した字句系列に対して処理をしているので，記述途中の構文が完全でないプログラムに対しても部分的に解析することができる．本研究ではこの TEBA を使って作成された書換えツールのコスト削減の支援をする．

2.2　書換えツール作成のコストとスタイル維持の精度

空白・改行・コメントなどのスタイルの要素は，用いることでプログラムが読みやすくなり，書換えの前後でも維持されるべきである．しかし，書換えツール作成のコストを下げるためにスタイル維持の処理を独立してツール化させることで，スタイル維持の精度が下がるという問題が生じる．書換えツールを利用した後のスタイル維持の精度を高めたい場合，書換える箇所のスタイルが入りうる部分の数だけ書換え後にその部分に存在したスタイルをどのように維持するのか考えなければならず，書換えツール作成のコストが増える．逆に，書換えツール作成のコストを下げるためにスタイル維持の処理を独立することで，書換えツールがおこなった書換えの内容が分からず，その書換え内容でスタイルを維持すべき位置が変わるので，書換え後の維持の精度が下がる可能性がある．

本研究で作成を支援する書換えツールは，低コストで作成することが前提である．したがって，本研究ではスタイル維持の精度が多少低下しても，書換えツールのコストを下げることを優先する．しかし，スタイル維持の精度が低下することで，プログラムの可読性が下がってしまう可能性もあるので，スタイル維持の精度の低下は抑えたい．本研究では，スタイル維持の精度の低下を抑える方法としてスタイル維持方法について研究する．

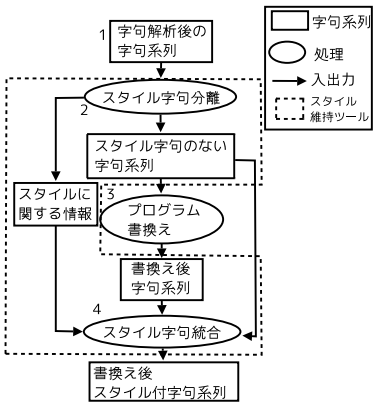

図 1　スタイル維持処理の流れ

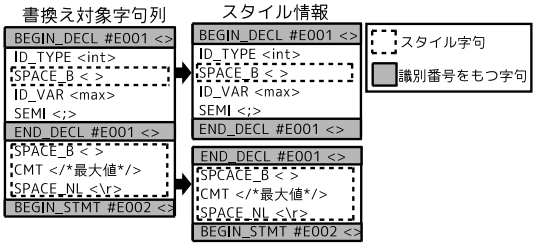

図 2　スタイル字句の分離とスタイル情報

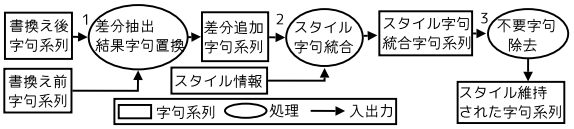

図 3　スタイル字句の統合処理の流れ

3　スタイル維持方法

3.1　スタイル維持方法の概要と問題点

書換え前のソースプログラムからスタイルに関する字句 (以下，スタイル字句) を取り除き，書換え後のソースプログラムにスタイル字句を統合することでスタイル維持をおこなう．スタイル維持方法およびスタイル維持ツールの全体像を図 1 に示し，各手順を次に示す．

1. 書換え対象のソースプログラムを字句解析し，得られた字句系列を入力として与える．
2. 字句系列からスタイル字句を分離する．
3. スタイル字句を除いた字句系列を書換えツールに渡し，書換えをおこなう．
4. 書換え前後の字句系列の差分を抽出し，差分の情報をもとに書換え前のスタイル字句を統合する．

コメントは必要性を機械的に判断できず，また一度失われると元のコメントを戻すことが不可能なのですべて維持する．空白・改行は機械的に書換え前のスタイルに近づけることは可能なので，可能な限り維持するという指針で維持ツールを設計する．

スタイル字句の統合は書換え前後の差分情報を付与した字句系列に対しておこなう．字句の書換え前後の対応関係をみつけるために識別番号を持った字句が必要になる．この識別番号を含めるか否かで書換え前後の差分を求めるときに結果が異なる．識別番号を含めた場合に書換え後の識別番号の割り当てにずれが生じると，書き変わっていない構文要素も差分に含んでしまい差分抽出箇所が増えてしまう．識別番号を無視すると本来と異なる構文要素間に対応関係があると判定され，書き変わっていない構文要素を含む可能性があり，書換え操作と一致しない差分を得る場合がある．書換えツールが識別番号を変更する可能性もあり，識別番号が書換え前後で保存されたか判定は困難である．よって，「識別番号を含めた差分」と「識別番号を除いた差分」を用いる 2 つの方法が考えられる．この 2 つの方法をどちらもおこない精度を確認する．各方法については次節以降で説明する．

スタイル維持の精度向上を目的とした字句の移動操作を提案する．手順 2 の前でおこなう操作を 3.2.3 節，手順 4 の後でおこなう操作を 3.2.4 節で説明する．

3.2　識別番号を含めた差分

3.2.1　スタイル字句の分離

書換え対象の字句系列からスタイル字句を取り除く．スタイル字句の統合時にスタイル字句を戻す箇所の指標が必要となるので，取り除いたスタイル字句に位置情報を持たせ保存する．これをスタイル情報と呼び，スタイル字句とその前の字句の並びで構成する．なお，前の字句だけでは同じ字句の並びがある場合に位置の特定が難しい．差分の求め方が異なるとスタイル字句の位置情報も 2 通り考えられる．書換え前後で識別番号の割り当てにずれがない場合，識別番号を用いた位置情報が正確となり，字句系列の中から統合する位置を一意に特定できる．そこで，スタイル情報には図 2 のようにスタイル字句の前の識別番号を持つ字句までの並びを位置情報として持たせる．

3.2.2　スタイル字句の統合

スタイル字句の統合の流れは図 3 の通りであり，次の手順でおこなう．

1. 書換え前後の字句系列から差分を抽出し，字句系列と差分の情報を合わせ 1 つの字句系列にする．
2. スタイル情報に基づきスタイル字句を手順 1 で得た字句系列に統合する．
3. 統合後に不要な字句を除去する．

分離したスタイル字句を書換え後の字句系列に統合するときに，スタイル情報と一致する箇所にスタイル字句を挿入する．しかし，書換え後の字句系列にスタイル字句を統合するのは難しい．これはスタイル情報のもつ位置情報は書換え前の字句系列だが，削除・変更の書換えにより書換え前の字句が失われてしまうことが理由として挙げられる．スタイル字句を挿入する位置を決めるにはどのような書換えがおこなわれたかの情報が必要になる．書換えツールが書換え後の字句系列以外の情報を出力する方法もあるが，書換えツールの処理の記述が増え

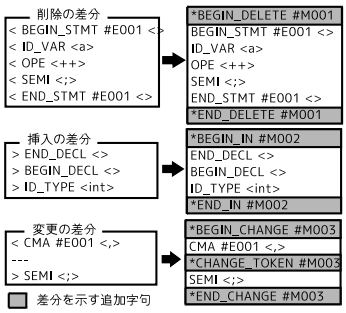

図 4　差分の字句への置換

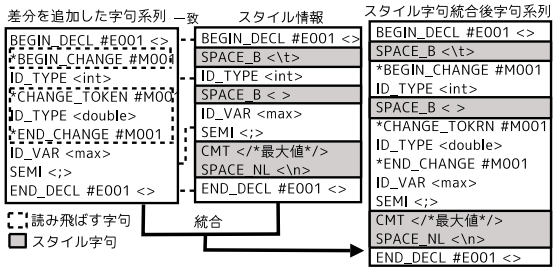

図 5　スタイル字句の統合

るという問題がある．そこで，書換え前後の字句系列の比較をおこない差分を抽出することで維持ツール側で書換えの情報を得る．

差分抽出では字句単位で削除・挿入・変更が得られる．差分の抽出結果を図 4 のように字句系列として表現し，書換え前後で変更のなかった共通する字句系列と合わせて 1 つの字句系列にする．書換え前の字句系列と差分による書換えの情報を合わせた字句系列が得られる．

スタイル字句の統合は差分による書換えの情報を持った字句系列におこない，図 5 のようにスタイル情報の位置情報の字句の並びと一致する所にスタイル字句を挿入する．統合対象の字句系列の先頭から字句がスタイル情報の字句と一致するか判定し，差分を字句に置換えたときに追加した字句 (以下，差分字句) と挿入された字句と変更のうち変更後の字句は書換え前に存在しなかった字句なので読み飛ばす．

次にスタイル字句統合後の不要な字句の除去をおこなう．ここでは，差分字句と削除範囲の字句，および，変更のうち変更前の範囲の字句を取り除く．設計指針に従い取り除く範囲にあるコメントの字句は残す．不要な字句が取り除かれた字句系列を字句結合させることで書換え後のソースプログラムを得る．

3.2.3　スタイル字句と構文要素の関連付け

統合時に不要な字句の除去で維持すべきスタイル字句も除去される問題がある．これは，スタイル字句は構文との関係が明確になっていないので，前後の構文の書換えの影響を受けるのが原因だと考えられる．例えば，1 つの構文が変更や削除された場合，その後ろの構文に関連のある空白が失われる可能性がある．そこで，スタイル字句がどの構文要素に関係しているか関連付け (以下，

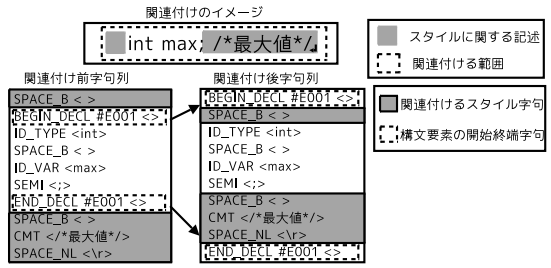

図 6　スタイル字句と構文要素の関連付け

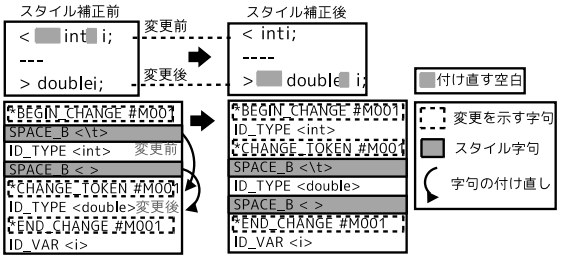

図 7　スタイル字句の補正

関連付け) をおこなう．関連付けをおこなうことでスタイル維持の精度を向上できる．ただし，コメントがどの構文要素に関連付くかはコメントの意味にもよるので，必ずしも精度を向上させるとはかぎらない．関連付けの方法は，図 6 のように，構文要素の開始と終端の仮想字句の位置を関係のあるスタイル字句まで移動させる．関連付けはスタイル字句の分離前におこなうことで，スタイル情報に関連のある構文要素の字句のみが含まれる．その後，スタイル字句の分離や書換えをおこなう．構文要素に関連付けるスタイル字句は，構文と同じ行に存在する空白・改行・コメントとその構文の直前の構文までのものとする．

3.2.4　スタイル字句の補正

前節の維持すべきスタイル字句が失われる原因として，差分で変更と判定された箇所にスタイル字句を統合し除去されることが挙げられる．特に識別番号を含めた差分を用いる場合は，字句の識別番号の変更があるとその字句の前後のスタイル字句が維持できない．そこで，スタイル字句の挿入位置の補正をおこなう．スタイル字句の補正では，図 7 のように変更の差分の変更前の範囲に統合したスタイル字句を変更後の字句系列に付け直す．変更前の字句の後ろにあるスタイル字句に位置の補正をおこない，変更後の字句の後ろに位置を移動させる．ただし変更の差分が必ずしも書換えの操作と一致しないので，正しい位置にスタイル字句が移動するとはかぎらない．

3.3　識別番号を除いた差分

識別番号を除いた差分を用いる方法は 3.2 節とほぼ同じであるが，分離と統合の一部が異なる．スタイル字句の分離でのスタイル情報は図 8 のように対象の字句系列から識別番号を除去したものとなる．統合の手順は図 3

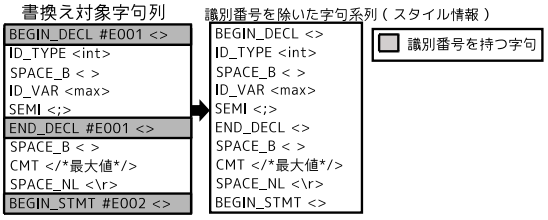

図 8　識別番号を除いたスタイル情報

と同じであるが，差分抽出は書換え前後の字句系列から識別番号を除去してからおこなう．

4　評価・考察

4.1　評価方法

3 節で提案したスタイル維持ツールの実装をおこなった．また実装したツールの実用性とスタイルの維持の精度を確認するために GNU coreutils[1] Ver.5.0 と Ver.8.5 の 2 つの版の差分を書換えツールによる書換えと見なして適用実験をおこなった．

4.2　評価基準

コメントと空白・改行では異なる評価基準を用いる．コメントを維持できたかの判定基準は次の 3 つである．

- 書換え前のコメントをすべて残せているか．
- 前後の字句に書換えがおこなわれた箇所で，コメントの位置関係が保てたか．
- 前後の字句に書換えがおこなわれていない箇所でコメントの位置関係を保てたか．

空白・改行を維持できたかどうか判断する基準として，最小編集距離を用いる．最小編集距離は一方から他方を得るためにおこなう最小の操作回数のことである．スタイル維持ツール使用後のファイルに改行・空白を削除・追加して，スタイル維持ツール使用前ファイルとコメント以外の内容が同じになる操作回数を数える．この操作回数を書換え箇所の行数で割った値が少ないほど維持ができているといえる．

4.3　評価結果

25 個のファイルを対象として 2 つの方法でスタイル維持をおこなった結果，どちらもすべてのコメントを残すことができた．コメントの位置関係では識別番号を除いた差分を用いた場合，175 個のコメントから 172 個のコメントを保つことができ，識別番号を含めた差分を用いた場合，161 個のコメントを保つことができた．識別番号を含めた差分を用いたときに維持できなかった 14 個のコメントのうち，書換えがおこなわれていない箇所で 4 個のコメントの維持ができなかった．空白・改行は，書換え箇所 1 行に対して識別番号を除いた差分では平均 2 回の操作が必要であり，識別番号を含めた差分では平均 3 回の操作が必要であった．この評価方法では，維持できたコメントの数や書換えがおこなわれていない箇所のコメントを維持できたことから，識別番号を除いた差分を用いる方がスタイル維持の精度は高かったといえる．

4.4　考察

評価結果から，書換え箇所以外のスタイルはすべて維持でき，コメントに関してはどちらの方法でも 9 割以上のコメントの位置関係を保つことができたので，このスタイル維持ツールは実用性があることが言える．識別番号を除いた方法の精度が高くなる理由はオープンソースの 2 つの版の識別番号が対応していないからである．よって識別番号の有無で差分の抽出箇所が変わり，スタイルの維持の精度に影響する．これは字句の削除や追加により書換えがおこなわれた箇所以降の字句系列は識別番号が変更されているからである．このとき識別番号を含めた差分を用いると，差分抽出箇所が増えスタイル字句統合後に維持できる空白・改行が少なくなる．オープンソースの評価方法のように書換え後に識別番号の割り当てがずれる場合，識別番号を除いた差分を用いる方法が適している．しかし，書換え箇所に同じ種別をもつ字句の並びが連続する場合，識別番号を除いた差分を用いた方法でもコメントの位置関係が維持できないことがあった．この場合，識別番号を保存できる書換えツールで，識別番号を含めた差分を用いる方法を使えば似た字句系列が存在しても識別番号の違いにより正確に位置関係を維持できる可能性がある．そこで目視で識別番号の対応関係の確認をおこない手動で識別番号を修正して実験をおこなった結果，識別番号を除いた差分を用いる方法で維持できなかったコメントの位置関係を維持できた．この結果から TEBA の環境下で実装された識別番号を保存できる書換えツールであれば，識別番号を含む差分を用いた方がスタイル維持の精度が高くなる．

5　おわりに

本研究ではスタイル維持処理を書換えツールから分離し，独立したスタイル維持方法として提案した．維持方法ではコメントを残すことを重視して維持をおこなったが，空白・改行の維持は書換えツール適用後に記述が増えた場合に対応できない．空白・改行については書換え前のスタイルの傾向をみて，書換え後に書換え前のスタイルを推測して適用した方が維持できると考えられる．今後の課題は空白・改行の維持の検討があげられる．

参考文献

[1] Free Software Foundation, Inc., “*Coreutils - GNU core utilities*,” http://www.gnu.org/software/coreutils/，Oct 2010.

[2] 吉田敦，“軽量下流 CASE ツール構築のためのソースプログラム表現形式の提案，” 情報処理学会論文誌，Vol.46，No.9，pp.2326-2336，2005.

[3] 吉田敦, 蜂巣吉成, 沢田篤史, 張漢明, 野呂昌満, “属性付き字句系列に基づくプログラム書換え支援環境の試作，” ソフトウェアエンジニアリング最前線 (ソフトウェア・エンジニアリング・シンポジウム 2010 予稿集)，Aug 2010，pp.119–126.