プログラム断片に対する解析器における補正用書き換え規則の系統的管理方法に関する研究

2009SE062 細井紀子　2009SE307 山田梨紗
指導教員：吉田敦

1　はじめに

プログラムの開発や保守は，プログラムの書き換えの連続であり，書き換え作業をできる限り自動化することで，プログラム編集の作業量と手作業による誤り混入，書き換え対象箇所の見落しを減らせる．書き換え作業の自動化には，プログラム解析技術が必須だが，解析対象が構文的に完全とは限らず，構文規則を満たさない断片の解析技術が不可欠である．ここでの断片とは，部品化されたプログラム記述や，前処理を前提としたC言語といった，本来の構文規則を満たさないプログラム記述を指す．

断片を対象とする解析器は提案されているが，対象を広げた代わりに解析精度を犠牲にしている．この欠点を補う手段として補正用書き換え規則を用いる解析器が存在する [1][3]．補正規則とは，構文規則の持つ制約から構文を満たすように，抽象構文木を書き換える規則であり，この補正により解析精度を向上させる．より精度を高めるためには，対象プログラムのコーディングスタイルを前提として，補正規則を最適化したり，特殊な記述に特化した規則を追加するといった，対象に特化した補正規則の修正が必要となる．しかし，編集によって，規則の適用順序や適用の有無が変わり，解析精度の低下や解析の誤りが生じるにも関わらず，それらを防ぐような系統的な管理方法は提示されていない．

規則の編集で生じる問題として，新たに追加した規則による適用順序の変化や，無限ループの発生がある．これらの問題の回避には，補正規則間の関係の把握が必要である．規則間の関係を解析するには，すべての組み合わせを調べる必要があり，その数は膨大になる．しかし，単純な作業であることから，これを自動化できる．

本研究では補正規則を系統的に保守できるよう，規則の編集で生じる問題から，補正規則を保守するときに行う補正規則の編集と解析結果の確認，規則間の関係の確認を支援する方法を提案する．

補正規則の保守の支援方法を，規則間の関係，構文的要素，規則の必要性の観点から整理して検討する．また，規則の編集で生じる問題の症状から対応する原因を整理し，問題の解決にあたりどの支援方法を用いればよいか指針を示す．規則の編集で生じる問題の自動的な解消は難しいので，本研究ではその解消は開発者に委ねるとし，問題の発見や分析を支援する．TEBA[3] を対象に提案した支援方法を適用し，支援方法の妥当性を検証する．他の解析器においても，規則間の関係を観点に管理するという考え方は，規則による書き換えで，規則を定義する順序によって解析結果が異なる場合と無限ループが生じる場合に対して適用できる．

2　補正用書き換え規則

2.1　補正用書き換え規則について

本研究では，ソースコード書き換え環境である TEBA の補正規則 47 個を対象とし，整理をする．対象の補正規則では，識別子の種別を詳細化や，不正確な解析結果の修正などの補正を行う．

図 1 の補正規則は，“=>” の前の書き換え条件と直後の書き換え結果で構成される．書き換え条件の各字句は “$変数名:種別” または “$変数名:’字句’” と表現し，それぞれ種別または字句に適合する字句を表す．一方，書き換え結果には，置換後の字句の並びを変数名を用いて記述する．変数は，書き換え条件でその変数に適合した属性付き字句を表現する．ただし，“$変数名:種別” と記述した場合は，変数に適合した字句の種別を変更したものを表す．図 1 は実際に TEBA で使用されている補正規則の例である．構造体のタグにあたる字句の種別を IDENT から ID_TAG へ書き換える．

```
# 構造体のタグ　IDENT->ID_TAG
{
 $struct:'struct' $sp:SP $tag:IDENT
} => {
 $struct          $sp    $tag:ID_TAG
}
```

図 1　補正規則の一例

TEBA では，補正規則を図 2 のようにファイル毎に定義順に適用する．図 2 の F1，F2 は規則が定義されたファイル，R1〜6 は規則を表す．書き換え条件の適合箇所が複数存在するときは，出現順にすべて書き換える．規則が使用された場合，再度，ファイルの先頭に定義された規則から順に適用される．最終的に，どの規則でも書き換わらなければ終了する．例えば，図 2 の F1 の R2 が使用された場合，F1 の先頭 R1 から順に適用を繰り返す．また，F1 のいずれの規則でも書き換えが起こらない場合，F2 の規則を適用する．

2.2　保守作業における問題

規則の保守作業は，追加・修正・削除の 3 つの基本作業の組み合わせで構成される．規則を追加・修正するときの問題として，規則間の関係を考慮せずに作業を行うと，解析が停止しない問題と，適合する箇所がないにも関わらず、適合の判定を繰り返し、時間を浪費する問題がある．また，他にどのような意図の規則が存在するか把握せずに規則を削除すると，それまで適用されていた規則が適用されなくなり，解析精度が低くなる問題がある．

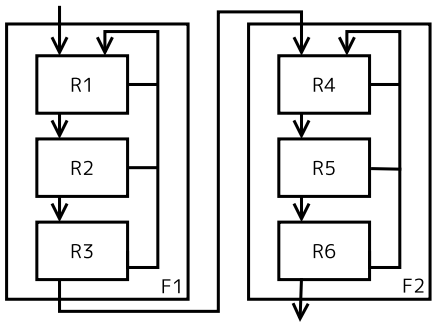

図 2 適用方法

2.3 補正用書き換え規則の問題

2.3.1 解析が停止しない問題

規則が相互に逆の変換になる場合，無限ループが発生し，解析が停止しない．相互に逆の変換となる規則とは，図 3 の (r1) の ID_VAR から ID_MEMBER にする規則と (r2) の ID_MEMBER を ID_VAR にする規則である．書き換え規則 A と B に依存関係があるとは，規則 A を適用すると規則 B による書き換えが起こる状態にあることを言う．さらに，複数の規則間の依存関係において，閉路が存在する場合，解析が停止しない可能性がある．

整理をする際に，規則間の依存関係を把握することで相互に逆の変換になる規則の組を発見でき，無限ループの原因となる規則の発見にも繋がる．

```
# (r1) 構造体のメンバ変数 ID_VAR->ID_MEMBER
{
 $b#1:BEGIN_STRUCT $s:'struct' $t:ANYEXPR
 $cl#2:CUR_L $a1:ANYDECL $v:ID_VAR $a2:ANYDECL $cr#2:CUR_R
 $e#1:END_STRUCT
} => {
 $b $s $t $cl $a1 $v:ID_MEMBER $a2 $cr $e
}

# (r2) 配列の添字 ID_MEMBER->ID_VAR
{
 $pl#1:ARR_L $s1:SP $v:ID_MEMBER $s2:SP $ar#1:ARR_R
} => {
 $pl $sp1 $var:ID_VAR $sp2 $ar
}
```

図 3 相互に逆の関係の規則

2.3.2 定義順の問題

ふたつの規則の書き換え条件に共通の種別の並びが存在するとき，それらの規則の間には選択関係があると言う．選択関係にある規則は，定義順において先に存在する方の規則が適用されることで，もう一方の規則は適用されず，期待する解析結果が得られない．また，適合する箇所がない規則は字句系列の全体に対して適合を試みるので，解析時間を長くする要因となる．よって，選択関係にあるふたつの規則については，定義順が適切かどうか検証する必要がある．

図 4 に選択関係となる規則を示す．(r3) と (r4) の書き換え条件の種別を比較すると IDENT SP IDENT が重なり合うので，(r3) と (r4) には選択関係がある．また，(r4) が (r3) を包含している．包含する側はより特殊な文脈を扱う規則なので，先に適用する必要があり，(r3) より前に (r4) を定義しなければ，正しい結果を得られない．

```
# (r3) 宣言の変数の推測 IDENT->ID_VAR
{
 $t:IDENT $s:SP $v:IDENT
} => {
 $t $s $v:ID_VAR
}

# (r4) 関数定義の型と関数名の推測
{
 $b:BEGIN_FUNC $t:IDENT $s1:SP $f:IDENT $s2:SP $a:PAR_L
} => {
 $b $t:ID_TYPE $s1 $f:ID_FUNC $s2 $arg
}
```

図 4 選択関係となる規則

2.3.3 解析精度の問題

規則は開発者の知見に基づき作成されているので構文要素に対する網羅性が確保されている保証がなく，解析精度が低い構文要素が生じる可能性がある．解析精度が低くなる原因として規則の不足，定義ミス，定義する順序の誤りが考えられる．

解析器の抽象構文木の要素の種類と規則間の関係を整理することで，不足がないか調査したり，新たな規則を追加するときに他の要素についての必要性を検討する材料になる．

3 系統的管理方法の支援の提案

3.1 原因と対処法

規則の問題の原因と対処するときの着眼点 P1〜P3 を表 1 に示し，以降の章で各着眼点の対処法を示す．P1 を規則間の関係，P2 を構文要素，P3 を不必要な規則とする．

表 1 原因と着眼点

原因	P1	P2	P3
相互に逆の変換がある	○		○
不必要な規則がある			○
必要な規則がない		○	
定義ミス		○	○
定義する順序の誤り	○		

3.2 規則間の関係

第 2.3.1 章の解析が停止しない問題と第 2.3.2 章の定義順の問題を発見，解決するためには規則間に存在する依存関係と選択関係を把握する必要がある．規則間の関係を把握する方法には，書き換えの履歴から関係を求める動的解析と，規則間の共通性から関係を求める静的解析の 2 種類がある．

動的解析で求まる規則間の関係は，解析対象のプログラムに依存する．一方，静的解析ではすべての規則間の関係を発見できるので，規則を効率良く保守するには，動的解析よりも静的解析の方が適している．

3.2.1 動的解析

規則間の関係を調べたい規則に対し，その規則を削除する前後の解析過程を比較する．使用されなくなった規則が存在する場合は，削除した規則と依存関係があると判定し，同じ箇所に対して書き換えを行った規則とは選択関係があると判定する．規則によっては依存関係と選択関係のどちらの関係も持つ規則が存在する．TEBA Ver.0.6.3 のすべての規則について調査した結果を表 2 に示す．

表 2 規則間に関係がある数

	数（個）
依存関係を持つ規則	14
選択関係を持つ規則	18

3.2.2 静的解析

依存関係と選択関係は，補正規則の書き換え条件と書き換え結果を構成する字句列の一致の仕方で判定する．字句列が部分的に一致するときの組み合わせを図 5 に示す．

書き換え箇所を含まない箇所が一致した場合は，規則間いずれの関係も存在しない．規則 A に規則 B が依存する場合，規則 A の書き換え結果と規則 B の書き換え条件の一致箇所には規則 A の書換え箇所が含まれる．規則 A 規則 B が選択関係にある場合，規則 A と B の書き換え条件の一致箇所に，どちらかの書き換え箇所が含まれる．

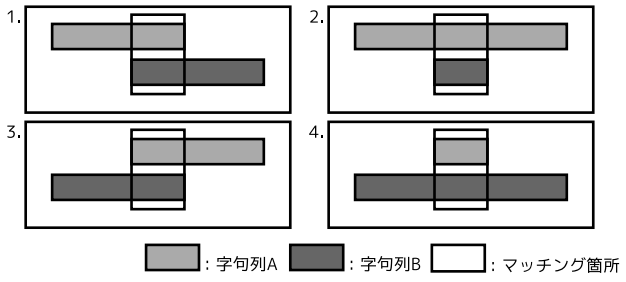

図 5 字句列が部分的に一致するときの組み合わせ

3.3 構文要素

第 2.3.3 章の解析精度の問題を解決することを目的に，構文要素と規則の間の関係を整理し，保守者に提供する．保守者は，補正規則に記述されている種別と字句に着目し，規則が属する構文要素を把握する．構文要素とは，TEBA が扱う抽象構文木のモデルに基づいて決まり，宣言文，型，変数，構造体，関数，制御文，式，マクロの 8 個である．種別と構文要素の関係の一部を表 3 に示す．対象となる種別は，全構文要素に当てはまる SP など空白類を除いた 35 個である．表 3 の丸印は種別がその構文要素の実体であることを意味する．三角形の印はその種別の字句が構文要素に対応する構文要素の構成要素である可能性を示す．構成要素となるか否かの基準も別に定める

例として，図 6 の規則を表 3 に従い，構文要素に分類する．まず，書き換え後の種別が ID_MEMBER であるので，

表 3 分類表一部

種別	宣言文	構造体	式	マクロ
ID_TAG	△	○		
ID_MEMBER	△	○	△	
ID_MACRO				○

表 3 より構造体に分類する．次に書き換え条件の種別に記号 “->” が記述されていることから式にも分類できる．

```
# 構造体，共用体のメンバ参照
{
 $r:'->' $s:SP $m:IDENT
} => {
 $r      $s     $m:ID_MEMBER
}
```

図 6 構造体の参照

このような分類により，各構文要素に対応する規則を把握できるので，構文要素の網羅性の観点から不足する規則を発見できる．例えば，構造体のタグに関する規則から共用体のタグに関する規則の不足を発見できる．

規則を作成するときは，表 3 から使用できる種別を確認できる．例えば，構造体に関する規則を作成したいときは，ID_TAG と ID_MEMBER は使用できるが ID_MACRO は使用できない．

3.4 不必要な規則の発見

使用されない規則を排除することを目的に全規則に対応するテストケースを作成し，使用されない規則を発見する．使用されない原因として，規則を定義する順番，定義ミス，前段階の解析での問題が挙げられる．規則間の関係や書き換え履歴から使用されない原因を解消する．

3.5 原因別の対処法

補正規則について問題の原因と症状を表 4 に示す．各問題の原因に対する対処方法を以下に示す．

規則が相互に逆の変換がある

まず，どの種別間で逆の変換となるのか確認する．次に，具体的にどの規則が無限ループの原因となるか依存関係を調査し，発見する．最後に，原因となる規則が不必要な規則か判断する．不必要と判断した場合は規則を削除して手続きを終了する．必要と判断した場合は，原因となるふたつの規則を別のファイルに記述する．

不必要な規則がある

まず，全規則に対応するテストケースを構文解析する．テストケースは作成した規則の目的に沿って規則の作成時にあらかじめ用意しておく．次に，書き換えの履歴から使用されない規則を発見する．使用されない原因を調査し，修正または削除する．

必要な規則がない

前提として，不必要な規則がない状態にあるものとする．まず，作成する規則で使用できる種別を構文要素の

表 4 問題の原因と症状

問題	症状	原因
解析が停止しない	無限ループ	規則が相互に逆の変換がある
無駄な解析がされる	解析時間が長い	不必要な規則がある
解析精度が低くなる	正しい結果が出力されない	必要な規則がない
		規則の定義ミス
		規則を定義する順序の誤り

表 3 から求め，規則を作成する．次に，追加する規則と既存の規則に選択関係が生じないように，規則を定義する．

規則の定義ミス

全規則に対応するテストケースの書き換え履歴から，定義ミスによって使用されない規則と誤った解析をする規則を発見する．次に，発見した規則の誤り箇所を修正する．

規則を定義する順序の誤り

まず，書き換え履歴から誤った解析結果となる箇所に対して適用される規則を発見する．次に，発見した規則と選択関係となる規則を発見し，特殊な文脈を扱う規則が先に適用されるよう規則を配置する．

4 検証

4.1 静的解析ツール検証と考察

静的解析ツールを作成し，TEBA Ver.0.6.3 の全補正規則 47 個の間の依存関係と選択関係を求めた．それらの関係にある規則の組が適用されるプログラムが構文的に正しいかの確認を行った．関係を定義するときに，書換え箇所を含む方がより精度は高まると考えられるが，精度を下げる可能性もある．この検証を，一致箇所に書き換え箇所を必ず含む場合と，含まないことがある場合で行い，それぞれの再現率と適合率を求め，書き換え箇所を含む必要があるか検証した．再現率を正解の規則の組に対する，ツールが判定した規則の組の割合とし，適合率をツールが判定をしたすべての規則の組に対する，正解と判定した規則の組の割合とする．静的解析に対する正解として，動的解析の結果を利用する．

いずれの場合も再現率は 1.00 だった．また，適合率は依存関係は 0.06 と 0.47，選択関係は 0.16 と 0.21 になり，書き換え箇所を含む方が高かった．よって，書き換え箇所による制限は妥当である．判定にはパターンマッチングを用いていることから，関係があると判定した規則の組が適用されるプログラムが，構文的に成り立つかまでは判定できない．よって，最終的な判断は保守者が行う必要がある．

4.2 原因別の対処法の検証と考察

すべての規則を修正したものとみなし，問題の発見と対処を行い，系統的に作業ができるか検証した．規則の修正にともない追加や削除が行われるので，それらの基本操作についても検証した．また，不必要な規則の削除の前後での解析結果の差分を求め，差分が出力された場合は解析精度が向上したか確認した．

この検証でで表 4 の 5 つの原因すべてを発見できた．さらに，第 3.5 章の対処法を使用して，不必要な規則の削除や定義ミスの修正ができた．これにより，系統的な対処が可能であることが確認できた．また，不必要な規則の削除の前後での解析結果を比較すると差が生じた．差分に含まれる箇所は種別がすべて正しく書き換わっており，解析精度の向上も確認できた．

すべての問題に対して系統的に対処できたが，規則の定義順に関わる課題も明らかになった．図 4 のふたつの規則には選択関係があり，手順に従い，(r4) を削除した．しかし，(r4) が (r3) より先に定義される場合，どちらの規則も削除できない．結果的に，正しい結果を得られるので (r4) を削除すると判定できたが，最適な定義順を決定する方法が必要である．

5 おわりに

本研究では，補正規則を系統的に保守できるよう，規則の編集で生じる問題を明らかにし，保守作業の支援方法を提案した．保守の基本作業である追加・修正・削除から解析が停止しない問題と定義順の問題，解析精度の問題を分析した．これらの問題に対して，規則間の関係と構文要素，不必要な規則に着目した支援方法を提案し，問題の手順を示した．提案方法を用いて，問題の解決と解析精度の向上ができた．

今後の課題として，選択関係がある規則の定義順の決定方法の検討が挙げられる．選択関係がある規則は，合流性 [2] の有無を確認し，合流性がある場合は一方を削除し，ない場合は規則の定義順を決定する必要がある．これにより，さらなる解析精度の向上や，保守者の労力軽減が実現できる

参考文献

[1] Y. Padioleau, “Parsing C/C++ Code without Preprocessing,” Proceedings of the 18th International Conference on Compiler Construction, pp.109-125, 2009.

[2] 外山　芳人，“項書き換えシステム入門，” 電子情報通信学会技術研究報告 SS, ソフトウェアサイエンス, vol.98，no.229，pp.31-38，1998.

[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ソースコード書き換え支援環境”，情報処理学会論文誌，vol.53，no.7，pp.1832-1849，2012.