Makefileの自動編集支援に関する研究
– 書換えツール構築支援環境の提案 –

2007MI009 青木 賢太郎　　2007MI017 遠藤 潤　　2007MI159 中村 亮太
指導教員　野呂 昌満　　蜂巣 吉成

1　はじめに

ソフトウェア開発において，一連のコンパイル手順を記述することで，作業を自動化させるツールとしてmake がある．Makefile は，そのコンパイル手順を記述したファイルであり，依存関係の管理やコマンドを生成するための規則が記述されている．

Makefile は，アプリケーションの規模が大きくなるにつれ記述量が増える．また，改編が続くとさまざまなファイルが混在し，依存関係が増えるので複雑化しやすくなる．記述量の増加や複雑化は，Makefile の編集作業量の増加や，エディタによる直接的な編集で誤りの混入を許すことがある．よって Makefile の記述の整理や，新たな記述の追加作業を自動化することが望ましい．Makefile を記述する際，一つの方法として Autoconf[3] などの自動生成ツールを利用する．しかし，典型的なアプリケーションの構成にしか対応できず，特殊な構成をしたアプリケーションや OS などの Makefile は手作業で記述されており，そのような Makefile を編集する支援環境はない．

Makefile の編集を支援するためには，基本的には構文木を生成し，それらを操作すれば良いが，マクロの存在が問題となる．マクロとは文字列の置換を指す．エディタによる編集の際は，マクロを展開した結果を調べつつ，展開前の状態で編集する必要がある．また，あるマクロ定義がその参照箇所より後に出現する場合，マクロの展開は遅延される．ゆえにマクロの展開方法は複雑である．

本研究では Makefile の自動編集支援を目的としている．書換えツールを構築することで編集の自動化を図ることが可能である．しかし，Makefile の編集は make を利用するソフトウェアごとに編集方法が異なり，利用者が必要としている編集は多くあるので，Makefile の編集方法を全て列挙することは困難である．ゆえに本研究では Makefile の自動編集支援として，書換えツール構築支援環境の提案をおこなう．書換えツールの構築するための環境により，さまざまな編集支援が可能となる．そのために，Makefile の構成要素を編集の観点から整理し，それらの構成要素を編集するための基本操作の集合 (以下，基本操作群とする) を提案する．この基本操作の設計にあたっては，マクロの展開に関する問題を考慮し，展開の必要性の有無に柔軟に対応できるようにする．なお，Makefile の記述はソフトウェアの開発環境によって異なる場合があるので，本研究では，広く利用されている GNU Make[5] の記述に限定する．

2　書換えツール構築支援環境

図 1 は Makefile に対する編集をおこなう書換えツール構築環境の全体像である．書換えツール構築支援環境は，Makefile の属性付き字句系列を基に，基本操作群から構成される．

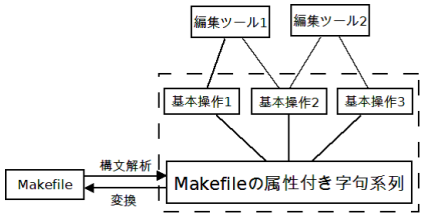

図 1　書換えツール構築支援環境

書換えツール構築環境では，まず Makefile を属性付き字句系列に変換する．変換した属性付き字句系列は基本操作群を用いて編集される．書換えツールは，その基本操作群を呼び出す形で実現し，字句系列の書換えをおこなう．最後に属性付き字句系列を Makefile に変換して書換え完了とする．

2.1　属性付き字句系列

Makefile の属性付き字句系列は，種別情報を付加した字句系列である．字句系列に基づく書換え環境を用いることは，まずパターンマッチングの利用による書換えが容易になるので解析器として扱いやすい．よって Makefile から属性付き字句系列へ変換する解析器には TEBA[10] を用いる．TEBA は字句系列に分解する際に，構文に関する情報である属性を付加する．また，構文としての情報を保持するために，構文の始点と終点に仮想的な字句 (以下，仮想字句とする) も一緒に出力する．これにより，字句系列をパターン変換する際，構文情報を利用できるので，構文木の変形に近い操作ができる．付加する属性には本来の構文解析で破棄される文字列も字句として扱い，スタイル情報を維持している．属性の例として，サフィックスを表す字句の属性を “ SFX ”，マクロ定義をあらわす仮想字句の属性を “ BEGIN_MACRO ”, “ END_MACRO ” として解析される．

2.2　基本操作群

基本操作とは，Makefile の編集手順に共通して現れる操作であり，これらを組み合わせて書換えツールを実現する．これら Makefile の属性付き字句系列や基本操作群から必要なものを選択して利用することで，書換えツールの構築をおこないやすくなる．また基本操作群は Makefile に用いられる編集方法を分析することで，書換えに必要とされる操作を基本操作として抽出する．この

分析を 3 節でおこない，また抽出した基本操作の分析を 4 節でおこなう．

3　編集方法の分析

後述する書換えツール構築支援環境の基盤となる基本操作の分析のために，Makefile の編集方法を分析し，その編集手順を考察する．編集方法の分析には，Makefile の編集方法が記載された文献 [5, 7] とさまざまなオープンソースの編集履歴の 2 つを用いた．

3.1　文献に基づく事例分析

文法的な視点から推測される基本的な操作を見つけるために，文法や用法が記述されている文献 [5, 7] を調査し，Makefile の典型的な編集としてマクロ化，サフィックスルール化，ダミーターゲットの適用の編集を選別し，手順を分析した．また，一例としてマクロ化の編集方法の詳細を 3.1.1 節に，編集手順を 3.1.2 節に示す．

3.1.1　マクロ化の詳細

マクロ定義の記述を変更させることで，適用させるマクロの参照箇所の内容を変更できる．Makefile 内に何度も現れるファイル名やオプションなどを，マクロ参照として反復して使うことで定義内容に対応したマクロ参照の内容の変更がおこないやすくなる．

3.1.2　マクロ化の手順

手順は以下のようになる．

1. 重複している箇所が記述されている個所を探す．
2. マクロに置換える文字列に対するマクロ定義の記述を追加する．
3. 手順 2 で定義したマクロの定義内容と一致する文字列を探す．
4. 適用箇所とマクロ参照の記述を置換える．

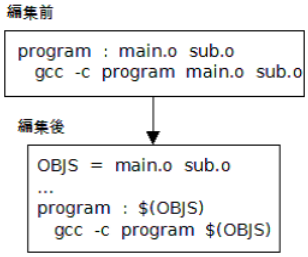

図 2　マクロ化の例

図 2 はマクロ化をおこなった事例である．図 2 の編集前の記述では，“ main.o sub.o ”が重複している．よってマクロ定義内容を “ main.o sub.o ” とする “ OBJS ” のマクロ定義を追加する．Makefile の記述を探していき，“ main.o sub.o ” と一致する文字列を “ $(OBJS) ” に書換える．

3.2　オープンソースに基づく事例分析

対象としたオープンソースは，Linux[6]，Git[2]，Wine[9] であり，これらの Git のリポジトリから Makefile の編集履歴を抽出した．編集履歴から分析した操作として，マクロ定義の改行操作とマクロ定義の分割の編集を選別し，手順を分析した．また，一例としてマクロ定義の改行操作の編集方法の詳細を 3.2.1 節，編集手順を 3.2.2 節に示す．

3.2.1　マクロ定義の改行操作の詳細

1 つのマクロ定義に多くのファイル名やオプションやマクロ参照が書かれていることでファイルの追加・削除の手間が多くなる．そういった場合，マクロ定義を改行，もしくはファイル名ごとに分割し，分割したものをそれぞれマクロ定義することで記述の追加や削除，またコメントアウトがおこないやすくなる．コメントアウトは，マクロ定義行において記述の追加・削除は繰り返されやすいので，記述を一時的に保存することに適している．

3.2.2　マクロ定義の改行操作の手順

手順は以下のようになる．

1. 改行したいマクロ定義行を探す．
2. どのように改行するかを判断する．
   (a) ファイル名，オプション，マクロ参照ごとに分割して定義する場合，マクロ定義を追加する．
   (b) 改行する場合，ファイルやオプションやマクロ参照ごとにバックスラッシュを加え改行する．

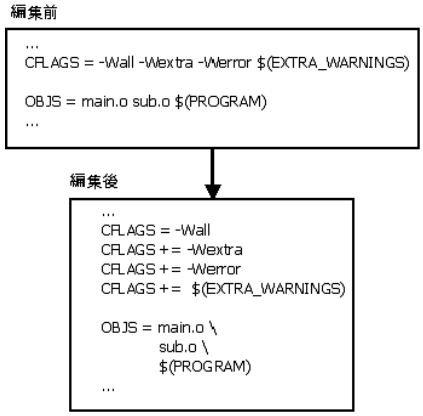

図 3　マクロ定義の改行操作の例

図 3 はマクロ定義の改行操作をおこなった事例である．図中の編集前の “ CFLAGS ”, “ OBJS ” のマクロ定義行を編集の対象とする．各マクロ定義内容をスペースで区切り，改行，もしくは分割をおこなう．

4　基本操作の抽出

4.1　基本操作の抽出方法

Makefile の編集手順に共通して現れる操作を，3 節で挙げた Makefile の編集方法の手順から基本操作を抽出する．また編集事例の手順だけでなく，Makefile を構成している字句，構文からも考えられる基本操作を抽出する．これにより基本操作を洗練することができる．

4.2　基本操作の抽出と整理

4.2.1　編集方法から抽出

基本操作を分析するにあたり，3.1 節，3.2 節で事例分析から，編集方法の手順の中で共通する操作や手順を明確化させた操作を抽出する．

4.2.2 字句系列や Makefile の構成から抽出

字句系列や Makefile の構文内容から各構文の構成をまとめ，その構成内容の一例を図 4 に示す．ここで Makefile の構成のクラスを構成要素の種別とする．ただし，マクロ参照は Makefile のどの記述にも相当することができるのでマクロ参照を展開した構成とする．Makefile の構成を示す各クラス図の構成要素から字句に着目し，字句を含むことで意味を持つ操作を基本操作として抽出した．

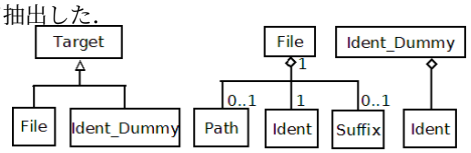

図 4　Makefile の構成のクラス図の例

4.2.3 Makefile の記述の構文から抽出

Makefile の記述の構文に着目したときに，構文を意識した構成要素に対する操作を基本操作として抽出した．

4.2.4 基本操作の整理

4.2.1 節，4.2.2 節，4.2.3 節から抽出した基本操作をまとめ，整理すると以下の通りになる．

- マクロ参照の展開
- マクロ定義の抽出
- 内部マクロの適用
- 記述名の変更
- 記述内容の書換え
- コマンド行の抽出
- コマンド行のオプションとその他の判別
- 依存関係行の抽出
- 依存関係行の追加
- 記述のユニーク化
- 記述のソート
- ターゲットの抽出
- ターゲットを生成していない依存関係行の抽出
- パスの抽出
- ワイルドカードの判別
- ターゲットの種類の判別
- サフィックスルール化
- サフィックス毎によるファイルの判別

ただし，抽出したこれらの基本操作のうち，マクロ参照の有無により操作に影響がでることで，利用する際にマクロ参照の展開をしなければならない基本操作がある．マクロ参照の展開の基本操作を除く 17 の操作のうち，5 の操作が事前にマクロ参照の展開をおこなう必要がある．

5 設計と実装

5.1 設計

5.1.1 基本操作設計の問題点

基本操作を設計するにあたりマクロ参照の展開に関する問題がある．編集をする際には記述の比較や編集箇所を探すことが必須になるが，その際マクロ参照の有無でその結果に影響がでる．例えば，記述の比較や編集箇所を探す際，参照するマクロに探したい字句の種別が含まれるならマクロ展開する必要がある．これは，探す範囲に参照するマクロが定義している内容が含まれないのでマクロ参照では操作に影響が出る．マクロ参照の展開をおこなう例としてマクロ化があげられる．しかし，構文要素ごとで比較や編集箇所を探すときはマクロ展開をしてはいけない場合がある．

また，マクロ参照の展開は遅延するものがある．これは，Makefile の全体で定義されているマクロを把握し，マクロの最終的な値が確定してから展開するというものである．遅延するかどうかにより，マクロ参照の展開の結果が異なってくる．

5.1.2 基本操作の設計方法

マクロ参照の展開が必要かどうかの考慮をする設計方法として，マクロ参照の展開が必要な基本操作にマクロ参照の展開の基本操作を組み込むという方法で設計をおこなう．これによりマクロ参照の展開が必要な基本操作のみ展開するので，余計なマクロ展開操作を防ぐことが出来る．

マクロ参照の展開は遅延するものがあるという問題に対する設計方法として，マクロ参照の展開の基本操作に Makefile 内に存在するマクロ定義の内容をあらかじめ全て抽出するという基本操作を組み込む設計をおこなう．ただし，遅延するものはマクロ定義の際の演算子が“＝”の場合なので，演算子によって抽出をするかどうかの場合わけを必要とする．

5.2 実装

実装する基本操作と書換えツールは字句ごとに分割された Makefile を対象に編集をおこなう．

5.2.1 解析による字句系列

TEBA は C 言語に限定された開発環境なので，本研究では Makefile に適用できるように 8 つの解析ルール TEBA をカスタマイズをした．字句解析により，Makefile の記述は属性が付加された字句が列挙された記述に変換される．ただし，TEBA により字句に付加された属性と，図 4 で示されている Makefile の構文情報とは異なっている場合がある．これは，図 4 の File と Ident_Dummy のように構文に関する情報だけではどちらか判別ができないからである．よって判別のできない字句はすべて Ident という字句で統一している．また，編集する上で字句を区別しなければならない場合，基本操作を利用することで字句の判別をおこなう．

5.2.2 基本操作

4.2 節で抽出した基本操作の実装をおこなった．提供する基本操作はいくつかのメソッドで構成され，状況に応じて必要なメソッドを利用する．マクロ参照の展開が必要である基本操作を用いた書換えツールと展開が必要である基本操作を用いない書換えツールの比較をおこなうために，抽出した 16 の基本操作のうち 6 の基本操作の実装をおこなった．

6 評価・考察

6.1 評価方法

本研究で提案した基本操作群による編集支援がなされているかをコスト削減の点から評価する．同じ編集操作をおこなう書換えツールを，基本操作を利用するものと基本操作を利用しないものの二種類実装する．二つのツールのソースの記述量を比較し，削減されている記述量を確認する．

6.2 実装した基本操作の評価

実装した書換えツールの例として，ファイル群のマクロ化の適用の詳細を示す．このツールで利用している基本操作としてマクロ展開，ユニーク化の適用，記述の書換えがある．このツールにおいて，各書換えツールを構成する基本操作が動作することによって Makefile の書換えをおこなうことができた．よって基本操作を用いて書換えツールの実装をおこなうことができていると考えられる．

また，基本操作を使用せずに実装した書換えツールと基本操作を利用して実装した書換えツールの記述量を比較する．マクロ化の適用のツールの場合，基本操作を利用しないで実装するとソースコードはおよそ 250 行となる．しかし基本操作を利用することで編集ツールの実装のソースコードがおよそ 160 行ほどに短縮することができた．よって，全体的に 30% ほど記述量を短縮することができた．実装した書換えツールでマクロ展開を必要とする基本操作がユニーク化，記述の書換えの二つであったので，記述量の短縮に影響が少なかった．ただし，基本操作を使用しない書換えツールも構文解析にカスタマイズした TEBA を用いている．カスタマイズした TEBA の行数は全体で 300 行であり，この環境がなければ書換えツールの実装に同程度の記述が必要になる．よって，書換えツールの記述量を大幅に削減することが出来，本研究で提案する環境による編集支援が成されている．

6.3 考察

基本操作の設計方法，基本操作の網羅性の観点から環境に対する考察をおこなった．

6.3.1 基本操作の設計

マクロ参照の展開操作の設計では，マクロ展開が遅延するという問題から前提としてマクロ定義の内容を抽出する操作が必要になる．その際，条件文がある場合は条件に一致するかどうかは make の実行時にしかわからないのでマクロ定義の内容の抽出は無視しなければならない．また，条件文を利用してマクロ定義の内容が変化することによって編集結果が異なってくる可能性があるので，条件文内で定義しているマクロを含んだ行は編集操作の対象とすることが出来ない．条件文を含めた記述は Makefile のソースではよく見られる記述なので，条件文を考慮した設計方法が必要になる．

6.3.2 基本操作の網羅性

網羅性は提供する基本操作がどのような書換えツールに対しても支援ができるかを考察する．編集に共通する操作に，編集する箇所を探索し抽出する操作と抽出した記述に対して記述の追加や書換える操作という部分がある．今回 4 節で洗練した基本操作は，編集手順の共通した操作から抽出した基本操作であり，編集がおこなわれる可能性がある記述に対する操作を網羅している．これは，Makefile を対象としているので，eclipse[4] のようなビルドツールの機能を検証することで基本操作として利用できる操作もある可能性がある．また，make と同様の機能を持つ Ant[1]，Maven[8] というツールがあり，Ant や Maven は Makefile にあたるファイルである build ファイルを持つ．この build ファイルに本研究であげた基本操作群を適用することが出来るかを検証し，基本操作が妥当である検討が必要になる．

7 おわりに

本研究では，Makefile の自動編集支援によるさまざまな編集ツールに対応させるために，汎用性を考えた基本操作とそれらを構築するための環境を提案した．また，基本操作とそれらから構成された書換えツールの実装をおこなった．実装した基本操作の評価をおこない，書換えをおこなうことができた．また，各書換えツールでマクロ展開が必要かどうかの検証をおこない，基本操作を利用することで，書換えツールの実装がおこないやすくなっていることを確認した．実装した書換えツールにおいてのマクロ展開の有無を区別することができた．今後の課題としては，条件文を考慮した基本操作の設計方法の検討が挙げられる．また，Ant や Maven と本研究で挙げた基本操作群との関連性を考察する事が挙げられる．

参考文献

[1] Ant, http://ant.apache.org/

[2] J.C.Hamano, *Git - Fast Version Control System*, http://git-scm.com/

[3] G.V.Vanghan, and B. Elliston, and T. Tromey, and I. L. Taylor, *GNU Autoconf/Automake/Libtool*, Ohmsha (2001).

[4] eclipse, http://www.eclipse.org/

[5] R.Mecklenburg, *GNU Make*, O'REILLY (2005).

[6] linux, http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=summary

[7] A.Oram, and S.Talbott, *make*, O'REILLY (1997).

[8] Maven, http://maven.apache.org/

[9] Wine, http://repo.or.cz/w/wine/hacks.git

[10] 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), pp.119–126, Aug 2010.