# mipl のプロジェクト管理について

理学部情報科学科　07251642　勝本真央

平成 23 年 2 月 14 日

**概 要**

mipl は、make を論理プログラミング言語 Prolog によって再実装するプロジェクトである。先行研究として「Prolog Shell」があり、Prolog で make の動作を実現することは理論上可能であるとされている。本プロジェクトはその実証実験を行うために、2005 年に立ち上げられた。mipl は make と同等の機能を持つことを目指しており、NetBSD のカーネルコンパイルが可能な程度まで実用的なシステムを実現することが最終的な目標である。

本研究は mipl プロジェクトを引き継いだものであり、mipl でこれまで実装されてきた機能をもとに、mipl の使い方についてのドキュメント整備や実行ファイルの整備、および成果物の公開に向けた準備などを行った。これらにより、下記に述べるような mipl プロジェクトの現時点での問題を克服し、活性化させることを目指した。

## 1　はじめに

make[1] はプログラムの構築に用いられるツールであり、ソフトウェア開発に広く利用されている。その有用性が広まり現在では様々な場面で利用されているが、make に多数の機能が追加されたことで makefile の肥大化・複雑化や非互換性の問題が起こっており、これが make を利用する際の弊害となっている。このような問題を解決するために立ち上げられたものが mipl プロジェクト [2, 3, 4, 5, 6] である。mipl は ‘make in Prolog’ の略であり、Prolog を用いて make を再実装する。

mipl プロジェクトではこれまで、さまざまな機能が実装されてきた。しかしその一方で、ドキュメントの整備やすぐに利用できる実行ファイルの整備、バージョン管理などの体制整備は十分に行われてこなかった。そのため、プロジェクトの引き継ぎの際には研究者がその都度開発内容や開発段階を調べ直さなければならず、効率の悪い状態が続いていた。そこで本研究では、現在の開発状況を正確に把握し、mipl の実行ファイルの整備や使い方についてのドキュメント整備、公開に向けた準備などを行った。そうして mipl を使用に耐えうるものとし、プロジェクトとして継続しやすい体制整備を目指した。

## 2　make とは

### 2.1　make の動作

make はコマンドジェネレータの一種であり、makefile と呼ばれるファイルに記述したルールやあらかじめ定義されている規則などを参照してシェルが実行するコマンド列を作成する。

ソフトウェアの開発においては、数多くのファイルが複雑に関連しあって存在している。ソースファイルの一部に修正を加えた場合、変更を反映したプログラムを作成するにはファイルをコンパイルし直さなければならない。この時、修正を加えていないファイルも含めて全てのファイルをコ

ンパイルし直すのは時間がかかり、無駄な作業でもある。また一方で変更の反映に必要なコマンドのみ手動で実行するのはミスも発生しやすく困難である。このような場面で用いられるのが make である。開発者は makefile にファイル同士の関係を記述しておけば、すべての更新作業は make が自動的に行うことになる。

make は makefile に記述されたファイル同士の依存関係をもとに、実行するコマンドを決定する。例えば、ソースファイルをコンパイルしてオブジェクトファイルを作り、それらのオブジェクトファイルをリンクしてプログラムを作るとする。この場合、それぞれのオブジェクトファイルがどのソースファイルから作成されるものであるか、またどのオブジェクトファイルをリンクしてプログラムを作成するかなどの関係を makefile に記述しておく。make はこれらの依存関係とともに、ファイルが更新された時間をファイルシステムから調べる。あるソースファイルの更新時間が、そのソースファイルから作成されたオブジェクトファイルの更新時間より新しければ、オブジェクトファイルを作成した後にソースファイルが変更されていることがわかる。このように全てのファイルについて変更状況を調べ、変更されたファイルに関係するコマンドのみをシェルに与えることで、ファイル更新時の構築作業を簡略化することができる。

make はもともとソフトウェア開発のために作られたものであるが、その有用性から現在ではソフトウェアのインストールやドキュメントの生成など様々な場面で利用されている。

### 2.2 make の問題点

make は広く利用されているツールであるが、いくつかの欠点がある。その一つが、makefile の肥大化や複雑化の問題である。makefile では関数定義を利用することができないため、類似した内容があっても何度も記述しなければならない。これにより makefile が肥大化し、可読性も低下してしまっている。また make 同士の互換性が無いという問題も生じている。make はもともと拡張に適した設計にはなっていなかったため、新たな機能が追加される度に make 自体が書き換えられていた。そのため、拡張された make はそれぞれ別のものになってしまい、ある種の規則に従って書かれた makefile は他種では利用できないといった問題が生じている。

## 3 mipl について

### 3.1 mipl の実装

mipl では、Prolog を用いて make の動作を実装している。make と Prolog は再帰的後ろ向き検索を行うという点で類似しているため、make の動作を Prolog の文法に自然にあてはめることができる。東京大学萩谷氏の「Prolog Shell」[7] でも、make を Prolog で実装することは理論的には可能であるとされている。mipl プロジェクトでは、make でフラットな文字列によって表されていた規則を、Prolog の項やリストなどを用いて書き換えている。このように Prolog の文法を用いることで、データ構造を自然に表記することができる。

mipl は、make を利用する際に生じていたいくつかの問題にも対応している。mipl では関数定義を用いることができるので、makefile に記述する規則を簡略化することができ、makefile の訂正なども効率的に行えるようになる。また mipl では、拡張機能はモジュールとしているため、機能の着脱も自由に行うことができ、mipl を書き換えることなく新たな機能を導入することも可能である。このことにより、mipl の亜種の発生を防ぐことができる。

### 3.2 現行の mipl プロジェクトの問題点

mipl プロジェクトではこれまで、さまざまな機能が実装されてきた。しかしプロジェクトを引き継ぐ際、これまでの蓄積をまとめられておらず、各段階の mipl を回収することにまず時間を費した。その中でも結局一部のファイルは見つけることができなかった。また集めたファイルの中でも、それぞれがプロジェクトのどの段階で作られたものかは一見してわからず、論文と見比べながら整理していかなければならなかった。最新の mipl の中でも、実行に必要なファイルとそれ以外のファイルが同じディレクトリに置かれており区別がつかないという問題があった。またファイルの整理が一通りでき、いざ mipl を動かそうという段階においても問題は生じた。mipl を動かそうとしても、makefile の書き方や mipl の操作方法に関する記述がなく、論文を参考にして記述方法や操作方法を一から確認していかなければならなかった。また論文通りの記述ではうまくいかず、ファイルの中身を一つ一つみながら修正したり、新たな機能を追加したりしなければならない場面もあった。

このような状態が続くと今後も引き継ぎを効率よく行うことができず、開発が滞る原因にもなる。これはプロジェクトとして十分な体制とはいえない。このようなソフトウェア開発プロジェクトを活性化し、有用な成果を出すためには、プロジェクトの維持に関する議論を積むことには大きな意義がある。そこで本研究では、実行ファイルの整備やドキュメント整備などを行い、プロジェクトとして維持しやすい体制の整備を行った。

## 4 整備の詳細

mipl の現在の体制を見直し、以下のような環境整備を行った。

1. 現状把握
   現在の mipl の開発段階や開発内容を把握する。
2. ファイルの再配置
   それぞれのファイルの役割を確認し、適切な場所に配置しなおす。
3. 実行ファイルの整備
   mipl の実行に必要なファイルを使いやすい形に修正する。
4. 公開に向けた準備
   mipl の公開に向けて、バージョン管理システム Subversion[8] を導入する。
5. mipl の使い方についてのドキュメント作成
   makefile の書き方や mipl の起動方法、操作方法などに関するドキュメントを作成する。

以下、上記のうち特に 4 と 5 について詳しく述べる。

### 4.1 Subversion について

mipl の公開のため、集中型バージョン管理システムの一種である Subversion を導入した。Subversion は、歴史的に広く利用されてきたバージョン管理システム CVS に改良を加えたものであり、C 言語で実装されたフリーソフトウェアである。

Subversion では、リポジトリと呼ばれる場所にバックアップファイルを蓄積していく。開発者はリポジトリにアクセスすることで、バックアップファイルをコピーすることができる。コピーしたファイルを修正し、コミットと呼ばれる作業を行うことで、修正したファイルが次のバックアップとしてリポジトリに蓄積される。修正には簡単なメッセージを添えてることができ、誰がいつ、どのファイルをどのように修正したかの調査や、修正の取り消しなども可能である。

Subversion ではファイルの修正のみならず、新しくファイルを追加したり、これまでのファイルを削除したりすることも可能である。ファイルの修正、追加、削除、名称変更などの履歴は全て保存され、バックアップの蓄積を簡単に行うことができる。

大人数で開発を行う際にはさらに有用である。複数人が同じファイルを同時に修正した場合でも、変更箇所が重ならなければそれらの修正は全て反映される。変更箇所が重なった時にはその箇所を表示し、どう対処するか開発者の指示を待つことになる。

mipl にも実際にこの Subversion を導入し、mipl 本体やドキュメント、サンプルファイルなどを順に蓄積していった。今後は Subversion のリポジトリからバックアップファイルを取得し、これらのファイルを修正したり、新たなファイルを追加したりすることにより、開発を進めていくことになる。Subversion の使い方についてはドキュメントにも追加したので、4.2.2 節で詳しく述べる。

## 4.2 ドキュメント

本研究では、mipl の使い方についてのドキュメントを作成した。ドキュメントの内容は、おおむね以下の通りである。

### 4.2.1 ファイルの構成

mipl の配布物に含まれるファイルや、それらのファイルをどのようなディレクトリに配置すればよいかなどを述べる。

mipl の使用において、まず以下のものを作業ディレクトリに準備しておく必要がある。それぞれのファイルの役割は次のとおりである。

- mipl
  SWI-Prolog を起動し、mipl.pl を読み込むシェルスクリプト。今回の整備で追加した。
- mipl.pl
  makefile.pl と prog ディレクトリ下のファイルを読み込み、make を実行する。
  マクロを定義した場合はそれをマクロリストに加える。今回の整備で追加した。
- prog
  - bash.pl
    Prolog のコマンド構文をシェルのコマンド行に変換する。
  - cmd_make.pl
    実行すべきコマンド列を作成する。
  - command.pl
    受け取ったコマンド列を実行する。

- dcg_suffix.pl
  与えられた文字コードリストがサフィックスを持つファイル名か、
  サフィックスを持たないファイル名かを認識する。
- exe_command.pl
  コマンドを実際に実行する。
- macro.pl
  マクロ展開を行う。
- macro_op.pl
  マクロ操作に関連するもののまとめ
- macrolist.pl
  デフォルトのマクロリスト
- make.pl
  make を行う。
- newer.pl
  更新時間の比較を行う。
- rule.pl
  サフィックスルールやデフォルトルール
- suffix.pl
  サフィックス操作を行う。

このうち一部のファイルは、そのままでは mipl が正しく動作しなかったため修正を加えた。

#### 4.2.2 管理の方法

Subversion によってバージョン管理を行う方法を述べる。
システムの導入および管理は以下のような手順で行われる。

(1) インストール
Subversion をインストールする。

(2) リポジトリ作成
Subversion による管理では、「リポジトリ」と「作業コピー」という二つのディレクトリを用意しておく必要がある。リポジトリはバックアップをためていく場所で、作業コピーはファイル修正などの作業を実際に進めていく領域のことである。ここでまずリポジトリを作成する。

(3) 初期インポート
作成したリポジトリに最初のバックアップをとる。
以上でリポジトリの準備は完了である。次からは作業コピーでの操作になる。

(4) 最初のチェックアウト
リポジトリのバックアップファイルを作業コピーに反映する。

(5) 作業コピーの修正
作業コピーでファイルを修正する。

(6) コミット
作業コピーからリポジトリにアクセスし、次のバックアップをとる。このことをコミットという。

(7) 以降は作業コピーの修正とコミットを繰り返していく。

Subversion は修正の他にも、ファイルの追加や管理しているファイルの削除、ファイル名の変更や修正の取り消しなどさまざまな機能が用意されている。

### 4.2.3 mipl の操作方法

Prolog を立ち上げて mipl 本体を読み込ませ、メイクファイルを与えてプログラムの構築を行うための操作方法を記述する。

(1) メイクファイル (makefile.pl) を書く
メイクファイルは mipl, mipl.pl, prog と同じディレクトリにおく。

(2) $ ./mipl
上記のコマンドをシェルで実行すると、Prolog の起動と mipl.pl の読み込みが行われる。
このとき mipl.pl 内では makefile.pl と prog ディレクトリ下のファイルが読み込まれている。

(3) ?- make. または ?- make(Target).
で make を実行する。

(4) ?- halt.
で Prolog を終了する。

### 4.2.4 メイクファイルの書き方

メイクファイルにどのようなことを書けば何ができるか、および文法などを述べる。内容は、ルールの書き方、マクロの定義方法などである。make の makefile にあたるものが、mipl の makefile.pl である。

makefile では次のような文法が用いられていた。

```
Target : Component
	Command
```

これを makefile.pl で表すと次のようになる。

```
rule(Target, Component,Command, MacroList).
```

具体例を示す。次は makefile の例である。

```
hahaha: hahaha.o
	cc hahaha.o -o hahaha
hahaha.o: hahaha.c
	cc hahaha.c -c
```

makefile.pl で同様のものを書くと次のようになる。

```
target(hahaha).
rule(hahaha, ['hahaha.o'], [[cc, 'hahaha.o', '-o', hahaha]], _).
rule('hahaha.o', ['hahaha.c'], [[cc, 'hahaha.c', '-c']], _).
rule('hahaha.c', [], [], _).
```

'hahaha.o' の'.' や'-o' の'-' など特殊文字として扱われる可能性のあるものは、シングルクォーテーションで囲ってエスケープし、Prolog のアトムとして与えている。 また、コンポーネントとコマンドはリストとして与えている。 マクロリストは prog ディレクトリ下の macrolist.pl から与えられるので、ここでは変数としている。

従来の make では、ターゲットを指定しないとき一番上のターゲットが作成される。mipl で同様のことをするには、makefile.pl に

```
target(Target).
```

を記述しておくことが必要である。定義できるのはひとつだけだが、一番上のターゲットでなくてもよい。これにより引数なしの

```
?- make.
```

で `target(Target).` で指定されたターゲットが作成される。それ以外のターゲットは

```
?- make(Target).
```

で作成できる。

なお、従来の make ではソースファイルの情報

```
hahaha.c:
```

を書く必要はないが、現在の mipl では

```
rule('hahaha.c', [], [], _).
```

を書かなければならない。

mipl では他にマクロ定義や関数定義などの機能が実装されている。これらを用いることで、より多様なメイクファイルを作ることができる。

# 5 まとめと今後の課題

mipl の実行ファイルの整備や使い方についてのドキュメント整備などにより、mipl を使用に耐えうるものとすることができた。またバージョン管理システムの導入により、開発を継続しやすい体制が整った。

卒業研究のような研究体制においては特に、時間の制約やマンパワーの限界などのためにドキュメント部分が十分な完成度に至らない場合が多いと考えられる。しかし本来、ドキュメントの整備やバージョン管理などの体制整備は研究を継続していくための大変重要な部分であり、こういった体制が整わないままでは研究が維持しやすい状態とは言いがたい。このような理由で研究成果の蓄積や継続が行われない損失は大きく、改善が必要であると考えられる。ソフトウェア開発プロジェクトを活性化するためには、プロジェクト整備的な意味を持った取り組みは大変重要であり、今後積極的に行われていくべきである。

また今回、現状の開発段階を調べ直したことで、mipl には実装されるべき機能がまだ多数あることがわかった。これらの機能を実装し、NetBSD のカーネルコンパイルが可能な程度まで実用的なシステムを実現することが、mipl プロジェクトの今後の課題である。

## 謝辞

本研究を進めるにあたり、様々なご指導を頂きました新出尚之先生に深謝いたします。また、make の学習の際にお世話になった鴨浩靖先生に感謝いたします。

## 参考文献

[1] Andrew Oram, Steve Talbott 共著, 矢吹道郎監訳, 菊地彰訳: make 改訂版, O'REILLY, 1997

[2] 笹山琴由: Prolog による make の実装〜mipl について〜, 奈良女子大学理学部情報科学科 2005 年度卒業論文, 2006

[3] 野口真理子: Prolog による make の実装と従来の make の欠点の解消について, 奈良女子大学理学部情報科学科 2006 年度卒業論文, 2007

[4] 藤本尚子: Prolog による make の実装− mipl の新機能について, 奈良女子大学理学部情報科学科 2006 年度卒業論文, 2007

[5] 内田有紀: mipl でのシェル制御構造の実現について, 奈良女子大学理学部情報科学科 2007 年度卒業論文, 2008

[6] 山西康世: 依存関係構築ツール mkdep とカーネルコンパイルディレクトリ作成ツール config の汎用化〜mipl 関連研究〜, 奈良女子大学理学部情報科学科 2008 年度卒業論文, 2009

[7] Masami Hagiya: Prolog Shell–Prolog with Modality, IPSJ SIGNotes, SYMbol manipulation Abstract, 026-002, 1983

[8] 上平哲著: 入門 Subversion Windows/Linux 対応, 秀和システム, 2006