構文マクロ定義を利用した動的拡張可能なエディタ

The Dynamically Extensible Editor Using Syntactic Macro Definition

甫水 佳奈子　　脇田 建 *

Summary.　本稿では，構文マクロ定義によって構文を拡張できるようなプログラミング言語のためのエディタを提案する．構文拡張機能として構文マクロシステムを提供する言語は，構文マクロ定義によって言語に新たな構文を追加することができる．構文マクロシステムは，言語の構造を変更することなく構文の拡張を可能にするもので，ドメイン特化型言語の作成に役立つ重要な機能である．ところで，プログラミング用のエディタの多くが，オートインデントやシンタックスハイライト等のプログラミング支援機能を提供しているが，これらの機能は構文の拡張には対応していない．予め対象とする言語の構文に従って定義されており，新たに追加された構文に対応させるためには定義の変更が必要となる．しかし，プログラマ自身が構文拡張のたびに定義を変更することは困難である．そこで，本研究では構文拡張のための構文マクロ定義をプログラミング支援機能に利用して，動的に拡張できるエディタを提案する．本エディタが提供するプログラミング支援機能はオートインデントとシンタックスハイライトである．インデント機能は，構文マクロ定義を構文の適切なレイアウトを与える手本として解釈することによって拡張する．ハイライト機能は，プログラマのインタラクティブな操作によって拡張する．本エディタのプロトタイプを構文マクロシステムを有する Scheme を対象言語として，JavaScript で実装した．

1　はじめに

プログラミング言語の中には，構文拡張機能として**構文マクロシステム**を提供する言語がある．このような言語は Scheme[2] 等の LISP 系言語がよく知られているが，Dylan[4] や構文マクロ付き JavaScript [3, 5] のようにオブジェクト指向言語のものもある．構文マクロシステムを有する言語は，**構文マクロ定義**によって言語に新たな構文を追加することができる．構文マクロシステムは，言語の構造を変更することなく構文の拡張を可能にするため，特定の領域の問題を解決するための**ドメイン特化型言語**（*DSL*）の作成に大いに役立つ重要な機能である．

ところで，プログラミング用のエディタの多くがオートインデントやシンタックスハイライト等のプログラミング支援機能を提供しているが，これらの機能は構文の拡張には対応していない．予め対象とする言語の構文に従って定義されており，プログラマによって新たに追加された構文に対しては適切に作動しない．構文の拡張に対応させるためには，ソースコードの書き換えやプラグインの作成による定義の変更が必要となる．しかし，プログラマ自身が構文拡張のたびに定義を変更することは困難である．そこで，本研究では構文拡張のための構文マクロ定義をプログラミング支援機能に利用して，機能を動的に拡張できるエディタを提案する．構文マクロ定義の利用により，機能の定義を変更することなく簡単な操作のみでの機能拡張を実現する．

2　構文マクロシステム

構文マクロシステムは抽象構文木の置き換えにより構文の抽象化を行う仕組みで，プログラムの簡略化や DSL の作成等に用いられる．このシステムは LISP 系言語だけでなくオブジェクト指向言語でも利用可能であることが [3, 5] によって示されている．

構文マクロシステムでは，構文マクロ定義によって言語に新たな構文を追加するが，追加された構文は**マクロ**と呼ぶ．構文マクロ定義は図 1 のように書き，**パターン**から**テンプレート**への変換規則を定義する．ここで，パターンはマクロの入力形式を表すものである．図 1 で定義している unless マクロは一つ以上の引数をとり，一つ目の引数が *test*，残りの引数が *expr* ... に束縛されてテンプレートに置き換えられる．マクロを使った式がパターンとマッチすると，その式は図 2 のように，対応するテンプレートに置き換えられて評価される．

```
(define-syntax unless
  (syntax-rules ()
    ((unless test expr ...)          ← パターン
     (if (not test) (begin expr ...)))))  ← テンプレート
```

図 1. Scheme での構文マクロ定義（unless マクロ）

このように，構文マクロ定義を使うと，言語の内部構造を変更することなく新たな構文を追加できる．

* Kanako Homizu and Ken Wakita, 東京工業大学大学院情報理工学研究科 数理・計算科学専攻

```
(unless                          (if
  (null? queue)                    (not (null? queue))
  (set! queue '())        ➔        (begin (set! queue '())
  (set! tail '()))    置き換え            (set! tail '())))
```

図 2. unless マクロを使った式とその置き換え後の式

既存のエディタのプログラミング支援機能は構文の追加に応じた機能拡張を提供していない．予め定義された構文についてのみ作動し，プログラマが追加したマクロに対しては適切に作動しない．例えば，Emacs[1] の Scheme 編集用モードでは，マクロはプログラマ定義の関数として扱われてしまう．マクロに対応させるためには，プログラマ自身が機能の定義を変更しなければならない．

そこで，本研究が提案するエディタは，構文マクロ定義をプログラミング支援機能の拡張に利用することにより，この問題を解決する．構文マクロ定義のパターンがマクロの入力形式を表していることを利用して，構文の追加に応じた動的な機能拡張を実現する．

3 提案エディタ

本研究が提案するエディタはプログラミング支援機能として，オートインデントとシンタックスハイライトを提供する．そして，これらの機能をプログラミングしながら動的に拡張することができる．インデント機能は，プログラマが本エディタに入力した構文マクロ定義によって拡張でき，ハイライト機能はプログラマのインタラクティブな操作によって拡張できる．

3.1 構文マクロ定義の書き方

構文マクロ定義を使ってインデント機能を拡張するために，構文マクロ定義のパターンはレイアウト付きで記述する．レイアウトとは，インデント後のトークンの位置関係を表すものである．図 3 のように，プログラマはレイアウトを表すためにパターンに適宜，空白や改行を挿入して記述する．このレイアウト付きのパターンが，マクロを使った式に対してインデントを行うときの手本となる．よって，本エディタはインデント機能について一種の例示プログラミングのような形式をとる．なお，パターン以外の部分については書き方に制約はない．

```
(define-syntax unless
  (syntax-rules ()
    ((unless
       test
       expr ...)
     (if (not test) (begin expr ...)))))
```

図 3. レイアウト付き構文マクロ定義

3.2 エディタの概要

本エディタは，編集エリア，マクロ登録ボタン，カラーパネルからなる．

プログラマは，編集エリア内でソースコードの編集を行うことができる．編集エリア内のソースコードには，本エディタによって自動的に適切なインデントやハイライトが施される．

マクロ登録ボタンは，プログラマが定義したマクロを本エディタに認識させるためのボタンである．マクロ登録ボタンが押されると，本エディタは編集エリア内のソースコードを解析して，構文マクロ定義からインデント機能のために必要な情報を読み取り，データ構造を生成する．インデント機能はここで生成したデータ構造を参照して実行するため，プログラマはエディタの定義を変更することなく，ボタン一つでインデント機能の拡張が可能となる．

ハイライト機能についてはプログラマがマクロの色情報を付加する必要があり，これをカラーパネルを用いて行う．プログラマはマクロ登録ボタンによって登録したマクロの名前をソースコード上で選択し，付けたい色をカラーパネルから選択する．この操作により，マクロ名に色が設定され，ソースコード上のマクロ名は対応する色でハイライトされる．このように，ハイライト機能も簡単に拡張することができる．

4 実装

本エディタのプロトタイプを構文マクロシステムを有する Scheme を対象言語として，JavaScript で実装した．実装したエディタは Web ブラウザ上で動く．図 4 に実装したエディタを使ってプログラミングした例を示す．

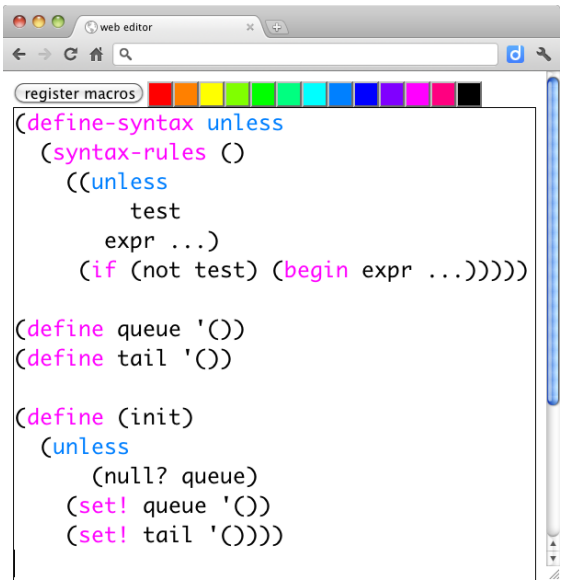


図 4. 実装したエディタを使ったプログラミング例

5 まとめ

構文マクロ定義を利用して，プログラミング支援機能を動的に拡張できるエディタを提案した．このエディタを使えば，構文拡張のたびにプログラマが各機能の定義を変更することなく，機能の拡張が可能となる．

本研究の今後の課題としては，構文マクロシステム付きの JavaScript 等，他の言語に対応することが挙げられる．また，本エディタが提供しているプログラミング支援機能はオートインデントとシンタックスハイライトのみであるが，コード補完や構文エラーチェック等の機能を充実させることも今後の課題である．

謝辞

本研究を進めるにあたり，様々な意見を下さった脇田研究室の皆様に感謝いたします．本研究は文部科学省科学研究費補助金基盤研究 (C) 課題番号 23500034 の援助を受けています．

参考文献

[1] GNU Emacs. http://www.gnu.org/software/emacs/.

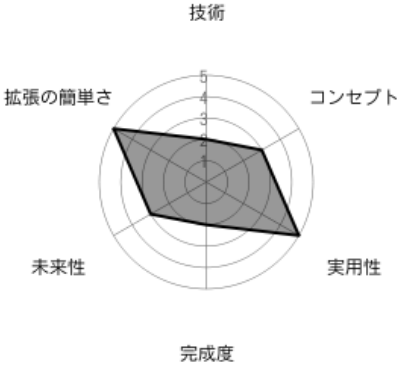
[2] N. I. Adams, IV, D. H. Bartley, G. Brooks, R. K. Dybvig, D. P. Friedman, R. Halstead, C. Hanson, C. T. Haynes, E. Kohlbecker, D. Oxley, K. M. Pitman, G. J. Rozas, G. L. Steele, Jr., G. J. Sussman, M. Wand, and H. Abelson. Revised^5 report on the algorithmic language Scheme. *SIGPLAN Not.*, 33:26–76, September 1998.

[3] H. Arai and K. Wakita. An implementation of a hygienic syntactic macro system for JavaScript: a preliminary report. In *Workshop on Self-Sustaining Systems*, S3 '10, pp. 30–40, New York, NY, USA, 2010. ACM.

[4] A. Shalit. *The Dylan Reference Manual: The Definitive Guide to the New Object-Oriented Dynamic Language*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1996.

[5] 荒井浩．「プログラム≠データ」な言語に対する Hygienic マクロシステム．修士論文, 東京工業大学大学院, 2011.

アピールチャート



未来ビジョン

本研究が提案するエディタは，構文マクロ定義によって構文を拡張できるようなプログラミング言語，すなわち構文マクロシステムを有する言語を対象としている．そのため，LISP 系のごく一部の言語のみを対象としているようにとらえられがちである．しかし，本文中でも述べたように，構文マクロシステムは LISP 系言語だけでなくオブジェクト指向言語等の他言語でも利用可能であることが示されている．よって，どのような言語でも構文マクロシステムを利用することができ，そのような言語はすべて本エディタの対象言語となる．

構文マクロシステムの主な利用は，ドメイン特化型言語（DSL）の作成である．構文マクロシステムを有する言語を様々な DSL を生み出す核言語ととらえると，本エディタは核言語から派生する DSL のためのエディタを自動で生成する核エディタと見なすことができる．

現在の実装では，核エディタとして対応できている言語は Scheme のみであるが，対応できる言語が増えればそれだけ自動生成できるエディタの幅も広がる．そして将来的に，すべてのプログラミング言語がある核言語から派生した DSL として表せたとすると，核エディタがすべてのプログラミング言語に対してその言語用のエディタを自動生成することが可能となる．本研究では，そのような核エディタの実現を目指している．