2K-4

構文マクロ定義からのプログラム整形器の生成*

甫水佳奈子†
東京工業大学 理学部
情報科学科

脇田建‡
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

1 はじめに

インデント（字下げ）によるソースコードの整形はプログラムの構造を表すために有用である．さらに，適切な量のインデントはプログラムの理解を高める [2]．そのため，多くのエディタがプログラミング言語の構造・構文に則ったプログラム整形システムを提供している．しかし，言語の中には Scheme のように，構文マクロ定義による構文拡張機能 [1] を備えた言語も存在するが，これらの整形システムには構文の拡張に対応したものは存在しない．そこで，本研究では構文拡張に対応するため，マクロ定義から構文のレイアウトを表す**インデント規則**を自動生成し，その規則に基づいてプログラムの整形を行うシステムを提案する．この考えを Hygienic 構文マクロシステムを有する Scheme を対象として Emacs の上で実装した．

2 Scheme の構文マクロシステム

本研究では Scheme の構文マクロシステムの内，パターンマッチによるマクロシステムを採用する．マクロの定義は次の形式で記述する．

```
(define-syntax ⟨macro-name⟩
  (syntax-rules (⟨literal⟩ ...)
    (⟨pattern⟩ ⟨template⟩) ...))
```

⟨*macro-name*⟩ で始まるマクロの使用が ⟨*pattern*⟩ とマッチすると，その使用は ⟨*template*⟩ に置き換えられる．Hygienic 性とは自動的に変数衝突を避ける仕組みのことである．

マクロ定義の例を以下に示す．`arith-if` は `test` の値が負数であれば `neg-from`，0 であれば `zero-form`，正数であれば `pos-form` を評価するマクロである．

```
(define-sytnax arith-if
  (syntax-rules ()
    ((arith-if test
         neg-form zero-form pos-form)
     (let ((var test))
       (cond ((< var 0) neg-form)
             ((= var 0) zero-form)
             ((> var 0) pos-form))))))
```

本研究では，マクロ定義を本来の構文拡張の目的を超えて，マクロの適切なインデントを与えるものとして解釈することで，マクロ使用を ⟨*pattern*⟩ に記述されたレイアウトに従って整形することを目的とする．

3 プログラム整形器の生成

プログラム整形器はマクロ定義から生成するインデント規則と**インデント幅算出システム**からなる．

インデント規則はマクロの形式とレイアウトを保持するための情報であり，マクロ定義を解析しながら生成する．本稿では簡略化のため，⟨*pattern*⟩ は以下の構文規則に従うものとする．

⟨*pattern*⟩ `::=` (⟨*macro-name*⟩ ⟨*arg*⟩`*`)
⟨*arg*⟩ `::=` ⟨*identifier*⟩ | ⟨*atom*⟩ | (⟨*arg*⟩`*`)

マクロ定義の ⟨*pattern*⟩ にあたる S 式を P とする．P の ⟨*arg*⟩ にあたる各 S 式を $e_{11}, \ldots, e_{1n_1}, \ldots, e_{l_P1}, \ldots, e_{l_Pn_{l_P}}$（$l_P$ は P の行数，n_i は i 行目に存在する ⟨*arg*⟩ の個数）とすると，P のインデント規則 $R(P)$ は以下で定義できる．

$$R(P) = ((I(e_{11}, P) \,.\, T(e_{11})) \ldots)$$
$$I(e_{ij}, P) = c(e_{i1}) - c(P)$$
$$T(e_{ij}) = \begin{cases} \texttt{symbol} & e_{ij} \in \langle identifier \rangle \\ \texttt{"}e_{ij}\texttt{"} & e_{ij} \in \langle atom \rangle \\ R(e_{ij}) & \text{otherwise} \end{cases}$$

ここで，$I(e, P)$ は e の**最内包含 S 式 P** の開き括弧（**最内包含括弧**）からの**インデント幅**，$T(e)$ は引数 e の変換，$c(e)$ は行頭から e の第一文字までの文字数を表す．この定義に従って，インデント規則を生成する．

---

* Generation of program shaper from syntax macro definition
† Kanako Homizu, Dept. of Information Science, Tokyo Institute of Technology
‡ Ken Wakita, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology

図 1 にマクロ定義 arith-if のインデント規則導出の例を示す．

```
(define-syntax arith-if
  (syntax-rules ()
    ((arith-if test
               neg-form zero-form pos-form) ...)))
```

c(P) = e_{11}, c(e_{11}) = e_{21}, c(e_{21}) = e_{22}, e_{23}

$$
\begin{aligned}
R(P) &= ((I(e_{11},P)\,.\,T(e_{11}))(I(e_{21},P)\,.\,T(e_{21})) \\
&\quad\ (I(e_{22},P)\,.\,T(e_{22}))(I(e_{23},P)\,.\,T(e_{23}))) \\
&= ((c(e_{11})-c(P)\,.\,T(e_{11}))(c(e_{21})-c(P)\,.\,T(e_{21})) \\
&\quad\ (c(e_{21})-c(P)\,.\,T(e_{22}))(c(e_{21})-c(P)\,.\,T(e_{23}))) \\
&= ((15-5\,.\,T(\texttt{test}))(9-5\,.\,T(\texttt{neg-form})) \\
&\quad\ (9-5\,.\,T(\texttt{zero-form}))(9-5\,.\,T(\texttt{pos-form}))) \\
&= ((10\,.\,\texttt{symbol})(4\,.\,\texttt{symbol})(4\,.\,\texttt{symbol})(4\,.\,\texttt{symbol}))
\end{aligned}
$$

図 1: arith-if のインデント規則の導出

インデント幅算出システムは，マクロ使用とインデント規則の各要素の $T(e)$ との照合を行い，インデント幅を算出する．$T(e)$ が symbol ならば任意の S 式，"e"ならば e と等価な S 式を受理し，リストならば (を受理してリスト内の $T(e)$ との照合を行う．マクロ使用内でインデントが要求された位置に達したときに，着目している要素のインデント幅が算出する値である．規則とマクロ使用の照合がパターンマッチと同等の働きをすることにより，複数のパターンを持つマクロへの対応も可能となる．

4 評価

本システムは Emacs の Scheme モードのインデントシステムを拡張して実装した．本研究で実装したプログラムがインデント幅を計算し，その結果を基にインデント処理自体は Emacs が担っている．Emacs の Scheme モードの機能の多くが C で実装されている一方，本システムは Emacs Lisp で実装したために，応答性の劣化が懸念されるが，通常の Scheme プログラムの編集に本システムを用いた限りでは問題はなかった．さらに，本システムの応答性を精密に計測する実験を行った．

まず，インデント規則の生成時間を計測した．引数の種類と数が異なるマクロ定義に対してインデント規則の生成を 1000 回行った．図 2 が示すように，インデント規則の生成時間はマクロ定義の引数部に出現する部分 S 式の数（すべての $\langle arg \rangle$ の数）に対して線形である．

さらに，インデント幅の算出時間を計測した．先の実験で定義したマクロを使用してインデント幅の算出を 1000 回行った．インデント幅の算出にはインデント規則とマクロ使用との照合を行うため，図 3 が示すように，規則生成と同じく，マクロ定義の引数部に出現する部分 S 式の数に対して線形の時間が必要である．

結果から，インデント規則の生成，インデント幅算出のどちらについてもプログラミングの障害とはならないことが確かめられた．

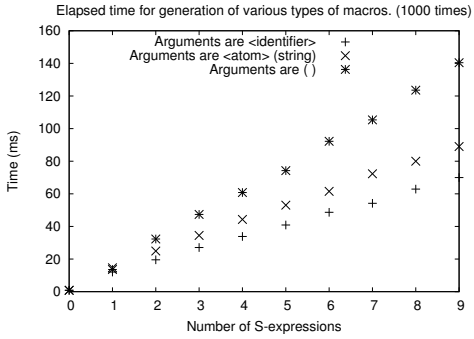

図 2: マクロ定義からのインデント規則生成時間

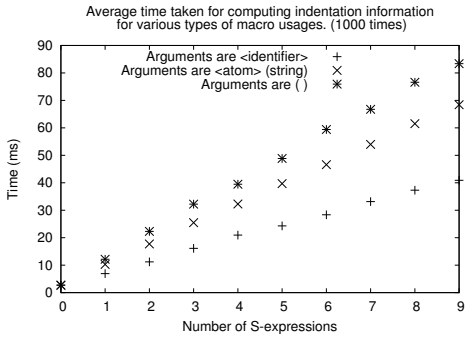

図 3: マクロ使用のインデント幅算出時間

5 まとめ

構文マクロ定義からマクロのレイアウトを表すインデント規則を生成し，その規則に基づいてプログラムの整形を行うシステムを提案した．このシステムにより，構文の拡張に対応できるようになった．

本研究では対象のマクロシステムとして Scheme のパターンマッチマクロを採用したが，Scheme には他にもマクロシステムが存在し，それらに対応することが今後の課題として挙げられる．また，本システムを他の言語やエディタ上で実現することも今後の課題である．

参考文献

[1] N. I. Adams, IV, D. H. Bartley, et al. Revised^5 report on the algorithmic language Scheme. *SIGPLAN Not.*, Vol. 33, pp. 26–76, September 1998.

[2] Richard J. Miara, Joyce A. Musselman, et al. Program indentation and comprehensibility. *Commun. ACM*, Vol. 26, No. 11, pp. 861–867, 1983.