多言語に対応した衛生的マクロ機構導入方式

高桑 健太郎　渡部 卓雄

OMeta で記述された言語処理系に対する，対象言語に依存しない衛生的マクロ機構の導入方式を提案する．OMeta は強力なパターンマッチ機構を備えたオブジェクト指向プログラミング言語であり，構文解析器などを作成するのに適している．提案方式は，Racket のマクロ機構で用いられている Scope Set モデルに基づき，OMeta で記述されたプログラミング言語の構文解析器に衛生的マクロの定義構文および定義機構を導入するものである．本方式を Javascript のサブセットおよび MinCaml の構文解析器に適用し，その有効性を確認した．

In this paper, we propose an implementation method of hygienic macro-definition mechanisms into languages defined using OMeta, an OOP language with PEG-based general-purpose pattern matching. From the specification of a macro-definition mechanism and a parser written in OMeta, the proposed method generates an extended parser equipped with the macro-definition mechanism based on the Scoped Set model. We show the effectiveness of the method by applying it to two different languages: Javascript and MinCaml.

1 はじめに

プログラミング言語におけるマクロは，プログラムの断片を記録しておき，コンパイル時あるいは実行時にそのプログラムを展開する仕組みであり，プログラムの表現力を高めるものとして有用である．衛生的 (hygienic) なマクロとは，マクロ展開によって新たに導入される識別子と展開先のプログラム中の識別子が衝突して，意図とは異なる意味になることを防止する機構を備えたマクロである．これによってマクロの定義は容易になるが，現在のところ衛生的マクロを提供する言語は Scheme 等の一部に留まっている．

本研究では，OMeta[9] で書かれたプログラミング言語の構文解析器に対して，衛生的 (hygienic) なマクロ機構を導入する手法を提案する．OMeta は強力なパターンマッチ機構を備えたオブジェクト指向のプログラミング言語であり，構文解析器などを作成するのに適している．

現在までに我々は，本研究で提案する手法と同様の目的で，OMeta で書かれたプログラミング言語に衛生的なマクロ機構を提供する仕組みである OMeta-Macro[7] を提案している．しかし OMetaMacro では，マクロが衛生的であることを保証するために対象言語に一定の制約を課している．また，例題として採用した対象言語に依存した部分があり，他のプログラミング言語に適用することが難しい可能性が指摘されている．

これらを踏まえて，本研究では，できるだけ対象言語に依存しないようなマクロ機構導入方式として，O-Hygieia の設計および実装を行った．評価として，OMeta で書かれた 2 つのプログラミング言語の構文解析器に対して本手法を適用し，それぞれに対して複数のテストケースで記述実験を行った．その結果，複数のプログラミング言語に対し，一部の制約の中で衛生的なマクロを提供できていることを確認した．

---

*A Multi-Language Implementation Method for Hygienic Macro Definition Mechanisms

This is an unrefereed paper. Copyrights belong to the Authors.

Kentaro Takakuwa, Takuo Watanabe, 東京工業大学 情報理工学院 情報工学系, Dept. of Computer Science, Tokyo Institute of Technology.

2 背景

2.1 衛生的マクロ

衛生的マクロ（Hygienic Macro）は，マクロ展開によって導入される識別子と展開先のプログラムにある識別子が衝突して，意図とは異なる意味になることを防ぐ機構を持つマクロである．衛生的なマクロを利用することで，展開されたプログラムが予期せぬ振る舞いを起こす可能性に注意を向けなくてよいため，マクロの定義が容易になる．

C言語のソースコード中でマクロを定義する`#define`ディレクティブは，単純な置き換えによって実現される，衛生的ではないマクロの例である．衛生的でなくなる例として，2つの`int`型の変数の値を入れかえることを意図して定義されたマクロがプログラム1のように定義されたとする．そしてこのマクロをプログラム2のように利用した場合を考える．

プログラム2を読むと，3行目で変数`a`と`t`の値が交換され，`a=20`，`t=10`という出力がされると考えることができる．しかし実際はプログラム3のように展開され，マクロの外の変数`t`とマクロ展開によって導入された変数`t`が衝突することで，値の交換は失敗してしまう．結果としてプログラム2は`a=10`，`t=20`という出力をする．

衛生的マクロはこのような変数の衝突を防ぐ仕組みを持つマクロである．

2.2 OMeta

OMeta[9]は，PEG[5] (Parsing Expression Grammar) をベースとしたオブジェクト指向のプログラミング言語である．OMetaはPEGと同様に宣言的に文法の記述ができるだけでなく，文字列以外のオブジェクトに対するパターンマッチ，規則のパラメータ化，左再帰への対応など，PEGにはない機能も備えている．これらの点からOMetaは字句解析器や構文解析器の作成に適していると言える．

OMetaは様々なプログラミング言語で実装が行われているが[8]，本研究で紹介するマクロ導入機構は，そのうちのJavaScriptによる実装であるOMetaJSで書かれた構文解析器を対象とする．OMetaJSは，

```
#define SWAP(x,y) \
    { int t = x; x = y; y = t; }
```

プログラム 1　C 言語での SWAP マクロの定義

```
int main(void) {
    int a = 10, t = 20;
    SWAP(a, t)
    printf("a=%d,␣t=%d\n", a, t);
    return 0;
}
```

プログラム 2　SWAP マクロの利用例

```
int main(void) {
    int a = 10, t = 20;
    { int t = a; a = t; t = t; }
    printf("a=%d,␣t=%d\n", a, t);
    return 0;
}
```

プログラム 3　プログラム 2 のマクロ展開後のプログラム

OMetaのプログラムをJavaScriptのオブジェクトからなる中間表現に変換し，それをさらにJavaScriptのプログラムに変換する形で利用できる．

2.3 OMetaMacro

OMetaMacro[7]は，OMetaで書かれた字句解析器と構文解析器を持つプログラミング言語の処理系に対して，衛生的なマクロ機構を導入するための仕組みである．対象の言語の処理系にOMetaMacroを適用することで，対象の言語のソースコード中でOMetaMacroが導入したマクロを利用することが可能になる．OMetaMacroが適用された構文解析器がマクロの定義を解析すると構文解析器が動的に拡張され，マクロ定義にもとづく新たな構文を解析することができるようになる．OMetaMacroは論文[7]において，検証用に作成されたプログラミング言語の字句解析器・構文解析器に対して，いくつかのユースケースで衛生的なマクロ機構を提供できていることが示されている．

3 提案手法

本節では，本研究で作成したマクロ定義機構導入

ツール O-Hygieia の概要と仕様について紹介する．

3.1 O-Hygieia

O-Hygieia は，OMetaJS で書かれた構文解析器に対して以下のような機能を提供する衛生的マクロ機構導入ツールである．

- 対象の構文解析器に対してマクロ定義構文を提供する．このマクロ定義構文を言語の中で記述することで，構文解析器を動的に拡張することができる．
- 構文解析器が生成する抽象構文木にスコープに関する情報を持たせることで，その情報にもとづいた衛生的なマクロ展開機構を提供する．

O-Hygieia が開発された背景として，先立って開発された衛生的マクロ機構導入ツールである OMeta-Macro がある．OMetaMacro も同様の機能を提供していたが，衛生的なマクロ展開を行うモジュール内に検証用の言語の仕様に依存して設計された部分があるために，他のプログラミング言語に対して適用することが難しいことが指摘されていた．これを受けて O-Hygieia は，衛生的なマクロ展開のために必要なスコープの情報をツール内部には持たせず，抽象構文木の追加情報として要求する設計となっている．

また，OMetaMacro の衛生的なマクロ展開器は Scheme の初期の衛生的マクロ機構が採用している Syntactic Closure [2] をモデルとして設計されている．一方 O-Hygieia はよりシンプルとされる Scope Set モデル [4] を参考にして作成されたマクロ展開器を持つ．

3.2 マクロの仕様

マクロを定義する文法はプログラム 4 の形式で定義されている．*pattern* と *template* はどちらもバッククオートで囲まれており，*pattern* にはマクロ呼出しの構文を記述し，*template* にはそれをどのように展開するかを記述する．*rule* には，定義したマクロを構文解析器のどの規則に追加するかを記述する．例えばここにプログラムの文に相当する規則を記述した時は，定義されたマクロは文として呼び出すことができる．

```
defsyntax ‘pattern‘ rule => ‘template‘
```

プログラム 4　マクロ定義の文法

マクロには引数をもたせることが可能であり，*pattern* 中に `%[`*param*　*rule*`]` の形式で記述した箇所が引数となる．*param* には引数名を入れ，*rule* には引数に与えられるプログラム片を解析する規則を記述する．*template* 中で引数を利用する場合は，`%[`*param*`]` の形式で記述する．

マクロ宣言以降でマクロを呼び出すには，プログラム片で引数を埋めた *pattern* を記述すればよい．引数に書かれたプログラム片が指定した規則によって解釈できなかった場合や，マクロが呼ばれた箇所を解釈する規則が定義中で指定した規則と一致しなかったときは，構文解析に失敗する．構文解析の結果として得られるマクロを含んだ抽象構文木に対して，Scope Set モデルをベースとした衛生的なマクロ展開が行われる．

3.3 マクロ機構導入のための構文解析器の拡張

O-Hygieia で衛生的なマクロ展開を行うためには，構文解析器の返す抽象構文木に追加の情報を持たせる必要がある．利用者はこの追加情報で変数のスコープのふるまいを定義することができる．現時点で扱うことができる情報は以下のとおりである．

1. 変数束縛が生じる文法要素の情報
2. 変数束縛のスコープに関する情報
3. 変数参照を表す文法要素の情報
4. 変数名の変更をさせたくない文法要素の情報

4 実装

O-Hygieia がどのように設計されているかを説明する．O-Hygieia をモジュールごとに分割すると次のようになる．

- マクロ定義解析器: プログラム中に現れるマクロ定義の構文を解析し，構文解析器を拡張する
- マクロテンプレート解析器: マクロ定義のテンプレートの部分の解析を行う
- マクロ展開モジュール: マクロを展開して，Scope

Set モデルにもとづいてスコープの情報を変数に与える

- 参照解決モジュール: 変数に与えられたスコープの集合をもとに，変数名を適切に変更する

マクロ定義解析器はプログラム中に現れるマクロ定義の構文を解析し，得られた結果にもとづいて対象の構文解析器を動的に拡張するモジュールである．マクロ定義に対する構文解析器は OMetaJS で記述されている．

マクロテンプレート解析器はマクロ定義内のテンプレートを解析する．解析はマクロ定義において指定された構文をベースにしながら，テンプレート中の引数も解釈できるような形で行われる．

マクロ展開モジュールは，マクロを含んだ抽象構文木のマクロを展開しながらスコープの情報を抽象構文木に追加する．追加されるスコープの種類としては以下の 4 種類がある．

- 束縛によるスコープ: 元のプログラミング言語の変数宣言あるいは変数束縛に伴って作成されるスコープ
- マクロ定義によるスコープ: 本マクロ導入機構が導入するマクロの宣言に対して作成されるスコープ
- マクロのテンプレートに対するスコープ: マクロ展開時にマクロのテンプレートから導入された変数に与えるスコープ
- マクロ呼び出しの引数に対するスコープ: マクロ展開時にマクロの引数から導入された変数に与えるスコープ

このモジュールにより，変数の文法要素はその文法要素に範囲が及ぶスコープの集合を持つことになる．

参照解決モジュールは，変数の文法要素がスコープの集合を持つ抽象構文木に対して適切な変数名の変更を行う．まず，抽象構文木中の変数束縛の文法要素とそれに対応するスコープの集合を集める．次に，抽象構文木中の変数参照の文法要素とそれに対応するスコープの集合を集める．そして，変数束縛とスコープの集合のペアをキーとし，新しい変数名と関連付けたテーブルを作成する．基本的に新しい変数名はキーの変数名の末尾に通し番号を加えたものとしている．それが自由変数と重複する場合は，通し番号の値を増やして重複しないような対応を行っている．最後に，抽象構文木全体の変数を以下のように変更する．

1. 変数名を v，スコープの集合を S とする．
2. テーブルから，変数名が v でかつ対応するスコープ集合が S の部分集合であるようなキーの中で，最も要素数が多い集合を持つキーを集める．
3. 該当するキーがただ 1 つ存在するならば，そのキーに対応する値に変更する．該当するキーが存在しなければ変数名の変更は行わない．
4. 該当するキーが複数存在する時は，参照が曖昧であるとし，エラーとする．この振る舞いは，Racket のマクロ機構と同様の振る舞いである．

5 実験

実験として，JavaScript サブセットと MinCaml の構文解析器を OMeta で設計し，それぞれに対して O-Hygieia を適用した．ここでは，JavaScript サブセットの構文解析器に対して適用した例を紹介する．

プログラム 5 は，引数の 2 つの変数の値を交換する SWAP マクロを定義し，それを利用したプログラムである．プログラム 5 では，マクロの内部と外部で同じ名前の変数 t が利用されている．素朴なマクロでは，展開後に変数の衝突が発生するようなケースである．

プログラム 5 に対して，O-Hygieia を適用した構文解析器が返す結果はプログラム 6 のようになった．マクロ内部と外部の t には異なる変数名がつけられており，結果として値の交換は成功している．

```
defsyntax ‘SWAP %[a Variable] %[b Variable];‘ Stmt
=> ‘{ let t = %[a]; %[a] = %[b]; %[b] = t; }‘
let t = 10, u = 20;
SWAP t u;
print(t, u);
```

プログラム 5　SWAP マクロの実装

```
let t_0 = 10, u_2 = 20;
{ let t_1 = t_0; t_0 = u_2; u_2 = t_1; }
print(t_0, u_2);
```

プログラム 6　プログラム 5 に対する出力結果

6 今後の課題

実験の結果，ブロック内でのマクロ定義で追加された構文解析の規則が，ブロックの外でも有効になっていることを発見した．このため，元の言語の構文がマクロ定義に上書きされて利用できなくなる現象が起こりうる．この対策として，構文解析の過程において，ブロックの外に出たことを検知し，ブロック内で定義された規則を削除する必要がある．

提案手法によって導入されるマクロには，マクロ定義構文に関するいくつかの制約が存在する．まず，マクロに関する制御構文が提供されていないので，繰り返しを伴うマクロ展開はできない．また，Scheme や Racket の衛生的マクロは，マクロのテンプレート中の変数を意図的に衛生的でなくするような操作が可能であるが，O-Hygieia は現在サポートしていない．こうした操作が可能になると，より高度な処理を行うマクロが書けるようになる．

本研究のマクロ導入機構を適用するためには，対象のプログラミング言語の仕様に対して一定の制約があることを節にて説明した．しかし，本機構を適用することが可能であるための十分な条件については考察が及んでいない．

また，本研究の機構が導入する衛生的マクロは，Racket で導入されている Scope Set モデルを参考にして実装されているが，この実装の正当性は，実験を通して確認された段階に留まっている．Adams は，衛生的マクロの明確なモデルを考案し，それが衛生的であることの形式的な議論を行った [1]．このような形式的な証明を与えることができれば，マクロ導入機構としての強い有効性を示すことができると考えている．

7 関連研究

7.1 OMetaMacro

本研究は，OMetaMacro が開発されたことを受けて行われた．その結果として，OMetaMacro で課題となっていた点をいくつか解決することができた．

まず，OMetaMacro は，対象のプログラミング言語の仕様に依存している部分があることが指摘されていた．これを踏まえて，本研究ではそのようなことができるだけ起こらないような設計および実装を行った．その結果，実験として 2 つの言語の構文解析器に対してマクロ導入機構を適用し，その有効性を確認した．

また，OMetaMacro で導入されるマクロが展開できるプログラムは，プログラムの文のみに限定されていた．本研究のマクロ機構では，対象の構文解析器で記述されている規則で解析できるものであれば，それをマクロの展開結果とすることができる．実験では，文に展開されるマクロだけでなく，式に展開されるマクロを定義し，期待通りに動作することを確認した．

7.2 Sweet.js

Sweet.js [3] は，JavaScript のための衛生的マクロ機構である．Scheme が備える `syntax-case` や `syntax-rules` のような，パターンマッチによる柔軟なマクロ定義が可能なマクロ機構を提供している．また，衛生的マクロのモデルとして，Racket に導入されている Scope Set モデルが同じく採用されている．

JavaScript 等のプログラミング言語に衛生的なマクロ機構を導入するにあたり，字句解析器に求められる出力が，マクロ機構による新しい構文の導入によって曖昧になることが課題となっていた．Sweet.js では，字句解析から得られるトークン列を S 式のようなトークン木に変換する Reader を字句解析器と構文解析器の間に置くことで，従来必要とされてきた構文解析器から字句解析器へのフィードバックを不要とした．一方で，O-Hygieia はこうした課題に対して，字句解析と構文解析を 1 つにまとめることを要求することで対応している．

7.3 Scheme のマクロ展開器を用いた衛生的マクロ機構

甫水らは，Scheme が備える衛生的マクロ機構を用いた，JavaScript 向けの衛生的マクロ機構を開発した [6]．この機構では，まず解析表現文法を用いて，マクロによる動的な構文の追加に対応した，拡張可能な構文解析器を実現した．そして衛生的なマクロ展開については，マクロを含む JavaScript のコードを，そ

れと等価な S 式に変換し，Scheme のマクロ展開器で展開を行い，展開後の S 式を JavaScript のコードに逆変換するという方法で実現している．

本研究では，OMeta で書かれた，一定の条件を満たす言語の構文解析器に対して，動的な構文拡張を行っている．また，甫水らの機構では，Syntactic Closure [2] を採用した Scheme のマクロ展開器を利用しているのに対し，本研究では，Racket の衛生的マクロ機構のモデルである Scope Set モデルを採用したマクロ展開器を作成している．

8 まとめ

本研究では，OMeta で書かれた構文解析器からなる言語処理系に対して衛生的なマクロを導入するための，特定のプログラミング言語の仕様にできるだけ依存しないような機構の設計および実装を行った．これに対する検証として，OMeta で書かれた異なるパラダイムの 2 つの言語の構文解析器を用意し，それぞれに対して本マクロ導入機構を適用し，いくつかのユースケースで評価を行った．結果として，いくつかの制約の中で，対象の言語に衛生的なマクロ機構を導入できていることを確認し，本マクロ導入機構の有効性を示した．

謝辞

本研究は JSPS 科研費 26330079 の助成を受けている．

参 考 文 献

[1] Adams, M. D.: Towards the Essence of Hygiene, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, New York, NY, USA, ACM, 2015, pp. 457–469.

[2] Bawden, A. and Rees, J.: Syntactic Closures, *Proceedings of the 1988 ACM Conference on LISP and Functional Programming*, LFP '88, New York, NY, USA, ACM, 1988, pp. 86–95.

[3] Disney, T., Faubion, N., Herman, D., and Flanagan, C.: Sweeten Your JavaScript: Hygienic Macros for ES5, *Proceedings of the 10th ACM Symposium on Dynamic Languages*, DLS '14, New York, NY, USA, ACM, 2014, pp. 35–44.

[4] Flatt, M.: Binding as Sets of Scopes, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, New York, NY, USA, ACM, 2016, pp. 705–717.

[5] Ford, B.: Parsing Expression Grammars: A Recognition-based Syntactic Foundation, *SIGPLAN Not.*, Vol. 39, No. 1(2004), pp. 111–122.

[6] 甫水佳奈子, 脇田建, 佐々木晃: 解析表現文法と Scheme マクロ展開器を用いた JavaScript 向け Hygienic 構文マクロシステムの実装, 情報処理学会論文誌プログラミング（*PRO*）, Vol. 6, No. 2(2013), pp. 85–101.

[7] 星野友宏, 高桑健太郎, 渡部卓雄: OMeta のための衛生的マクロ定義機構導入方式, 日本ソフトウェア科学会第 *33* 回大会, (2016).

[8] Warth, A.: OMeta: An Object-Oriented Language for Pattern Matching, `http://www.tinlizzie.org/ometa/`.

[9] Warth, A. and Piumarta, I.: OMeta: An Object-oriented Language for Pattern Matching, *Proceedings of the 2007 Symposium on Dynamic Languages*, DLS '07, New York, NY, USA, ACM, 2007, pp. 11–19.