多言語に対応した衛生的マクロ定義機構導入方式

高桑 健太郎・渡部 卓雄 (東京工業大学)

概要

本研究は、プログラミング言語OMetaで記述された言語処理系に対する、対象言語の仕様に依存しない衛生的マクロ機構の導入方式を提案する。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。

Scope Setモデル[Flatt 2016]

Scope Setモデルは、2015年にプログラミング言語Racketへ導入された、衛生的マクロ機構のモデルである。Scope Setモデルでは、変数束縛およびマクロ展開によってスコープが生じて、プログラム中の変数は、その変数に範囲がおよぶスコープの集合を持つ。そのスコープの集合から、変数が参照できる束縛を判別する。

下の図では、Racketのプログラム中の変数にスコープの集合を持たせながら、マクロを展開している様子を示している。let式、let-syntax式およびlambda式による変数束縛と、マクロmの展開によってスコープが作成され、それぞれのスコープがおよぶ変数に与えられる。

```
(let ([x 1]) # before expansion
  (let-syntax ([m (syntax-rules () [(m) #'x])])
    (lambda (x) (m))
```

Racketのマクロ展開の例

```
(let ([x{a_let} 1]) # after expansion
  (let-syntax ([m{a_let, b_ls}  (syntax-rules () [(m) #'x{a_let}])])
  (lambda (x{a_let, b_ls, c_lam}) x{a_let, b_ls, d_intro})
```

OMeta[Warth 2007]

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る。PEG(Parsing Expression Grammar)をベースとした言語であり、さらにPEGがサポートしていない左再帰やパラメータ付き規則などもサポートしている。そのため、字句解析器や構文解析器などの設計に適している。

```
ometa Calculator {
  expr = expr:l '+' term:r -> (l + r)
       | expr:l '-' term:r -> (l - r),
  term = term:l '*' fact:r -> (l * r)
       | term:l '/' fact:r -> (l / r),
  fact = <digit+>:d      -> parseInt(d)
       | '(' expr:e ')' -> e
}
```

整数の四則演算を行うOMeta/JSのソースコード

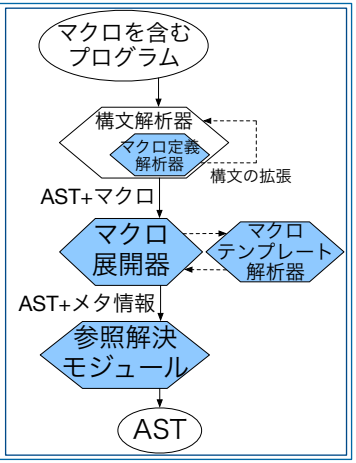
O-Hygieia

本研究で提案する手法の実装であるO-Hygieiaは、OMeta/JSで書かれた構文解析器を対象とし、次のようなマクロ定義構文を提供する。

```
defsyntax `«マクロ構文のパターン»` «マクロ構文の規則» => `«マクロのテンプレート»`
```

O-Hygieiaは以下のモジュールから構成されている:

- マクロ定義解析器：対象の構文解析器に構文規則として挿入する。マクロ定義の解析結果から新たな構文規則を生成し、それを構文解析器に動的に追加する
- マクロテンプレート解析器：マクロのテンプレートを解析する
- マクロ展開器：マクロを展開し、衛生的な変数書き換えのための情報を与える
- 参照解決モジュール：抽象構文木に与えられたメタ情報を元に、適切に変数名を書き変える



O-Hygieiaの設計

構文解析器への記述

O-Hygieiaを対象の構文解析器に適用する前に、その言語におけるスコープのふるまいを構文解析器に記述する。この記述は、Scope Setモデルに基づいた衛生的なマクロ展開を行うために、O-Hygieia内のマクロ展開器で利用される。

現在サポートしている、変数のふるまいに関する記述項目は以下のとおりである。

1. 参照を表すノード(e.g. 変数)の記述
2. 変数束縛を作るノードについての記述
3. 変数束縛が有効である場所の記述
4. 衛生的なマクロ展開の際の変数名書き換えの対象から除外する記述

右の図は、実験のために用意したMinCamlの構文解析器に、let式におけるスコープのふるまいを記述したものである。この図で行われている追加の記述は、

```
let <id> = <val_exp> in <body>
```

における変数**<id>**で束縛が作成され、それが**<body>**の内部で有効であることを表す。

```
ometa MinCamlParser {
  …
  Ident = Type('IDENT'):id -> {
    type: 'Var', name: id, _x_ref: 'name' // 1
  },
  Exp = Token('LET') Ident:id Token('EQUAL')
        Exp:val_exp Token('IN') Exp:body -> {
    // let <id> = <val_exp> in <body>
    type: 'Let', id: id, val_exp: val_exp, body: body,
    _x_bound_variables: ['id'], // 2
    _x_scope_specs: [
      { type: 'inner', from: '', to: ['body'] }
    ] // 3
  }
  …
```

MinCamlの構文解析器(一部抜粋)

適用例

O-Hygieiaの検証として、OMetaで書かれたJavaScriptサブセットおよびMinCamlの構文解析器を用意し、その言語におけるスコープのふるまいを記述し、O-Hygieiaを適用した。

例1-a：JavaScriptサブセットへの適用例

```
defsyntax `%[a Variable] <-> %[b Variable];` Stmt
=> `{ let t = %[a]; %[a] = %[b]; %{b] = tmp; }
let t = 10, u = 20;
t <-> u; // swap values of t and u
print(t, u);
```

例1-b：図1-aのマクロ展開後に相当するプログラム

```
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);
```

例2-a：MinCamlへの適用例

```
defsyntax `p! %[exp SimpleExp]` Exp => `print_int %[exp]`
let print_int = 42 in p! print_int
```

例2-b：図2-aのマクロ展開後相当のプログラム

```
let print_int_0 = 42 in print_int print_int_0
```

今後の課題

- ブロック内で定義したマクロがブロック外でも利用できてしまう不具合の修正
- 本方式が、今回適用した以外のプログラミング言語にも適用できるかの調査
- より高度なマクロ定義への対応
  - 可変長のパラメータ、意図的に衛生的でなくする仕組みなど
- 本方式が衛生的マクロを正しく提供できているかの形式的な検証

関連研究

- EX-JS[甫水 2013]：Schemeのマクロ展開器を利用したJavaScriptのための衛生的マクロ機構
- Sweet.js[Disney 2014]：JavaScriptのための衛生的マクロ機構
- OMetaMacro[星野 2016]：O-Hygieiaのきっかけとなった、OMeta/JSで書かれた字句解析器・構文解析器のための衛生的マクロ機構