構文解析器の動的拡張による衛生的マクロ機構の実現

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、特定の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[Flatt 2016]モデルに基づき、OMeta[Warth 2007]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]

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る。
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。

```
(let ([x 1]) ; before expansion
  (let-syntax ([m (syntax-rules () [(m) #'x])])
    ((lambda (x) (m)) 42)))
```

```
(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, d_intro}) 42)))
```

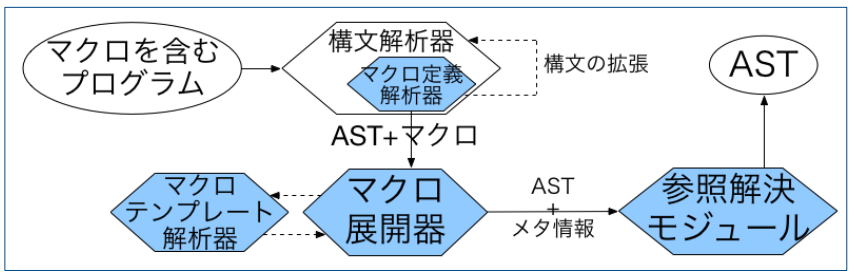
Scope Setモデルに基づいたRacketでの衛生的マクロ展開の例

O-Hygieia

OMeta/JSで書かれた構文解析器を対象に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、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、MinCamlにおける `let in` 式についてのスコープのふるまいを記述したものである。

MinCamlの構文解析器(一部抜粋)

```
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 -> {
    type: 'Let', id: id, val_exp: val_exp, body: body,
    _x_bound_vars: ['id'], // 2
    _x_scope_specs: [
      { type: 'local', from: '', to: ['body'] }
    ] // 3
  }
  ...
```

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


ケーススタディ

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、書く言語におけるスコープのふるまいを記述したものにO-Hygieiaを適用した。

適用例1-a：JavaScriptサブセット

```
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);
```

適用例1-b：図1-aのマクロ展開後

```
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);
```

適用例2-a：JavaScriptサブセット

```
let count = 0;
defsyntax `COUNTER` Expr19 => `(count++)`
{
    let count = 10000;
    print(COUNTER);
}
```

適用例2-b：図2-aのマクロ展開後

```
let count_0 = 0;

{
    let count_1 = 10000;
    print(count_0);
}
```

適用例3-a：JavaScriptサブセット

```
defsyntax `%[obj Expr18] &. %[attr Variable];` Expr18
=> `(%[obj] === undefined) ? undefined : %[obj].%[attr])`

let o = { x : { y : 20 } };

o &. x &. y;
```

適用例3-b：図3-aのマクロ展開後

```
let o_0 = { x : { y : 20 } };

((o_0 === undefined ? undefined : o_0.x) === undefined
    ? undefined
    : (o_0 === undefined ? undefined : o_0.x).y);
//  「フィールド名は変更しない」という指定を構文解析器に記述している
```

適用例4-a：MinCamlへの適用例

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

適用例4-b：図2-aのマクロ展開後

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

今後の展望

- より高度・実用的なマクロ定義への対応に向けて、OMetaから生成された構文解析器に対して動的な変更を加える設計では困難になってきた
  - OMetaを含む従来のPEGでは動的な変更は想定されていないため
- 動的な文法拡張に対応している構文解析のモデル・仕組みを導入し、対応するO-Hygieiaの部分を置き換える

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
- Sugar*[Erdweg 2013]：様々な言語に対する衣構文をIDE上で実現するためのフレームワーク
- APEG[d.S. Reis 2014]：構文拡張に対応したPEGのモデル