An Implementation of a Hygienic Syntactic Macro System for JavaScript: a Preliminary Report

Hiroshi Arai and Ken Wakita

Tokyo Institute of Technology

Outline

- Overview of hygienic macro system
- Difficulties of implementation for hygienic macro system to other languages
- Extensible parser
- JavaScript ⇔ Scheme Translator
- Examples
- Conclusion and future work

Macro systems

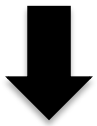
Macro systems are classified by the level of operations.

Lexical macro	*Syntactic macro*
CPP m4 TeX	Lisp, Scheme Camlp4 JavaScript (our work)

Lexical macro system: CPP

```
#define square(x) ((x) * (x))

2 * square(3 + 4);
```

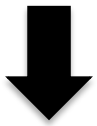


```
2 * ((3 + 4) * (3 + 4));
```

Lexical macro system: CPP

```
#define square(x) x * x
```

```
2 * square(3 + 4)
```



```
2 * 3 + 4 * 3 + 4
```

Syntactic macro system: Scheme macro

```
(define-syntax to_str
  (syntax-rules ()
    ((to_str sym)
     (symbol->string ‘sym))))

(to_str hello)
=> (symbol->string ‘hello)
=> ”hello”
```
eval

Definition for plural patterns

```
(define-syntax to_str
  (syntax-rules ()
    ((to_str) '())
    ((to_str s1 s2 ...)
     (cons (symbol->string ‘s1)
           (to_str s2 ...)))))

(to_str hello macro)
```

Definition

```
((to_str) '())
((to_str s1 s2 ...)
   (cons (symbol->string ‘s1)
         (to_str s2 ...)))
```

```
(to_str hello macro)
=>(cons (symbol->string ‘hello)
        (to_str macro))
```

Definition

```
((to_str) '())
((to_str s1 s2 ...)
   (cons (symbol->string ‘s1)
         (to_str s2 ...)))
```

```
(cons (symbol->string ‘hello)
      (to_str macro))

=>(cons (symbol->string ‘hello)
        (cons (symbol->string ‘macro)
              (to_str)))
```

Definition

```
((to_str) '())
((to_str s1 s2 ...)
   (cons (symbol->string ‘s1)
         (to_str s2 ...)))
```

```
(cons (symbol->string ‘hello)
         (cons (symbol->string ‘macro)
               (to_str)))
=>(cons (symbol->string ‘hello)
         (cons (symbol->string ‘macro)
               ‘()))
=>(“hello” “macro”)
```
eval

Another example: or macro

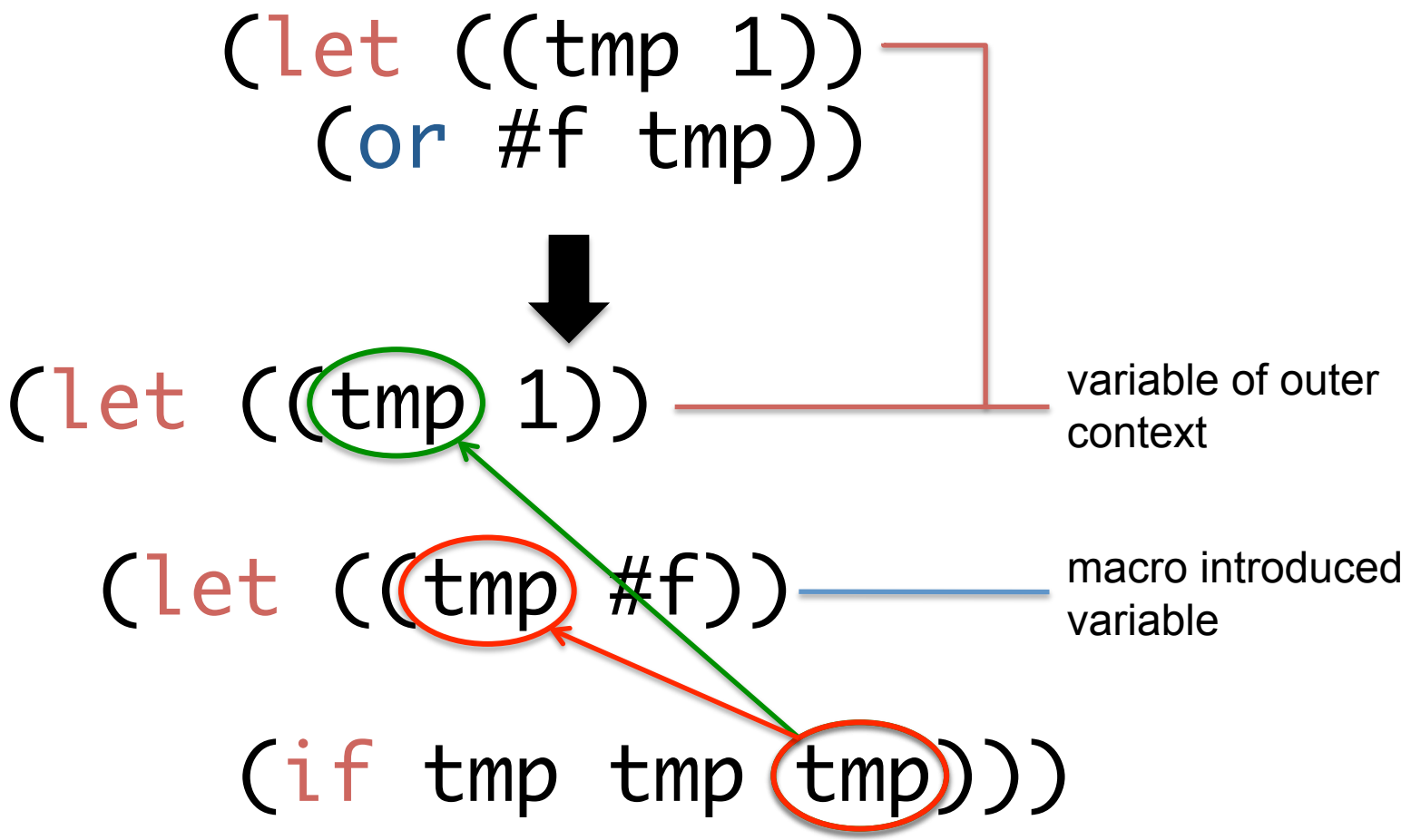
```
(define-syntax or
  (syntax-rules ()
    ((or) #f)
    ((or e) e)
    ((or e1 e2 ...)
     (let ((tmp e1))
       (if tmp tmp (or e2 ...))))))
```

Scheme macro system is Hygienic

Hygienic condition:
The property that avoids an accidental name collision between
the macro introduced variables and
free variables or
other macro introduced variables.

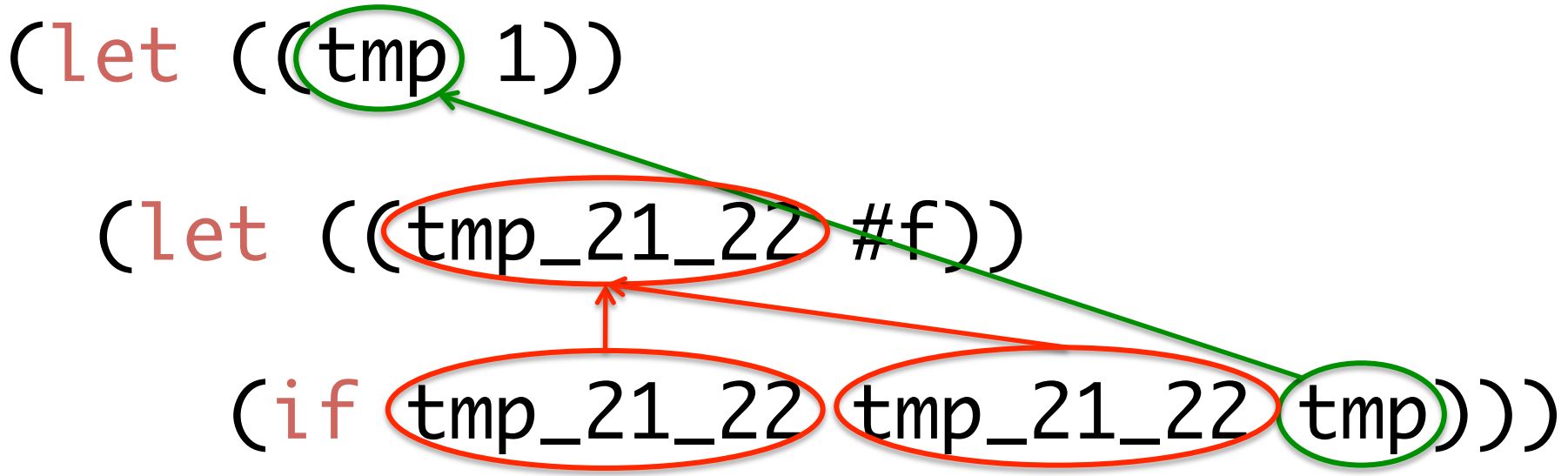
E. Kohlbecker, D. P. Friedman, M. Felleisen, and B. Duba. Hygienic macro expansion. In LFP '86

A problem of non-hygienic expansion



Hygienic expansion

```
(let ((tmp 1))
  (or #f tmp))
```

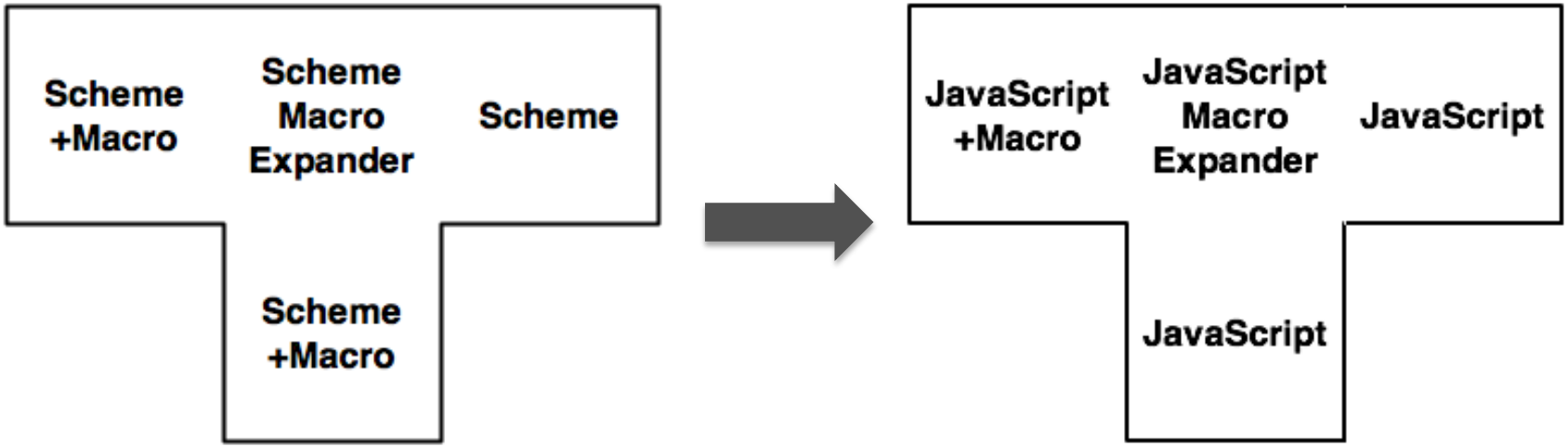


Outline

- Overview of hygienic macro system
- Difficulties of implementation for hygienic macro system to other languages
- Extensible parser
- JavaScript ⇔ Scheme Translator
- Examples
- Conclusion and future work

Simple porting method

Obvious strategy for porting a system is translation of existing code into the others.



Difficulty for porting

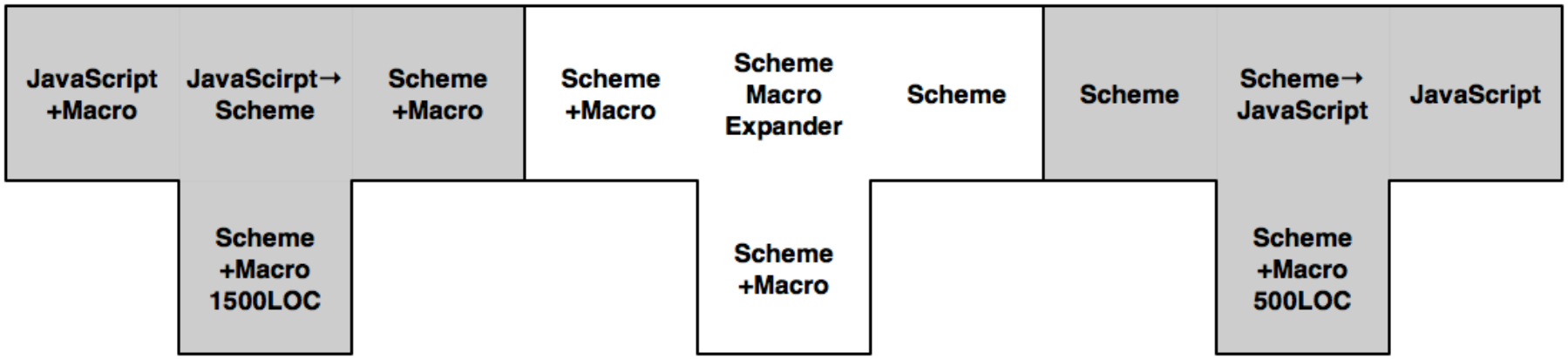
A reference implementation of Scheme's macro system is expanded to *30,000 lines* of code like this.

```
(lambda (#{g0 |UwoF2=FWF67?Hzg=|}
  ((lambda (#{g3 |JRDLKQ9XIcBUwDHQ|})
     (if (= #{g3 |JRDLKQ9XIcBUwDHQ|} '1)
         (apply
           (lambda (#{x |&E!%8VqL2Ux5zIh7|})
             (#{make-parameter |tJ?rb56xFSKFjB67|}
               #{x |&E!%8VqL2Ux5zIh7|}
               (lambda (#{x |kq/GEZu/KkI1gPuc|})
                 #{x |kq/GEZu/KkI1gPuc|}))))
```

Ghuloum and R. K. Dybvig. R6RS libraries and syntax-case system.
http://ikarus-scheme.org/r6rs-libraries/

Implementation overview

Our method uses Scheme macro system to expand JavaScript macro.
(Each T-diagram represents a translator)



Outline

- Overview of hygienic macro system
- Difficulties of implementation for hygienic macro system to other languages
- Extensible parser
- JavaScript ⇔ Scheme Translator
- Examples
- Conclusion and future work

Expanded form of Scheme macro is always S-expression

We can use same parser for the user-defined macro.

```
(to_str hello)
(to_str hello world of macros)
(let ((tmp 1))
  (or #f tmp))
```

Difficulty for parsing

The parser can not analyze the syntax of user defined macro usage.

```
var a = Let x = 3
        in x * x;
```

JavaScript does not have this syntax.
How to parse macro usages?

Top Down Operator precedence Parser

This parser allows us to modularize a parser into a set of independent parsing modules.

```
Let id = expr
  in body
```

```
(define parse_Let
  (lambda ()
    (advance “Let”)
    (parse_identifier)
    (parse_=)
    (parse_expression)
    (advance “in”)
    (parse_expression)))
```

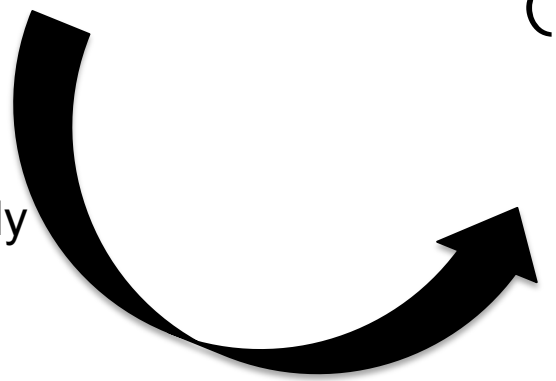
This is the essence. Actually, it is needed to store results of each parse.

Extensible parser

The parser has to be extended when the macro definition.

```
define_expression Let() {
  Let id = expr
    in body
```

Create dynamically like this.



```
(add-new-parser “Let”
  (lambda ()
    (advance “Let”)
    (parse_identifier)
    (parse_=)
    (parse_expression)
    (advance “in”)
    (parse_expression)))
```

Arguments information

When the macro definition time, the programmer has to specify the type of arguments because the parser needs the information.

```
define_expression macro() {
    identifier: id;
    expression: expr;
    statement: stm;
    {
        macro(id, expr, stm) =>

        /*expanded form */
    }
};
```

Outline

- Overview of hygienic macro system
- Difficulties of implementation for hygienic macro system to other languages
- Extensible parser
- JavaScript ⇔ Scheme Translator
- Examples
- Conclusion and future work

Simple translation (Bad example)

- Translate from JavaScript into Scheme semantically similar elements.

```
if (exp) {
    do_something;
} else {
    do_otherthing;
}
```

```
(if exp
    do_something
    do_otherthing)
```

- But how do we treat JS peculiar syntax?
- And there are often subtle differences between two languages.

Our translation rules

Correct treatment of variables and their bindings is important.

$$T[c] = c_S \quad T[v] = v$$

$$T[\textbf{var } v \ = \ e] = (\textbf{define } v \ T[e])$$

$$T[\textbf{function } (v_1, v_2, \ldots) \ \{E\}] = (\textbf{lambda } (v_1 \ v_2 \ \ldots) \ T[E])$$

$$T[\ominus e] = (\textbf{JS op1 } \text{"}\ominus\text{"} \ T[e])$$

$$T[e_1 \oplus e_2] = (\textbf{JS op2 } \text{"}\oplus\text{"} \ T[e_1] \ T[e_2])$$

$$T[\textbf{define\_syntax } \text{macro } () \ \ldots \ \{ \textit{pattern} \Rightarrow \textit{expansion}; \ \ldots \}] =$$
$$(\textbf{define-syntax } \text{macro } (\textbf{syntax-rules } () \ (\textit{pattern } \textit{expansion}) \ \ldots))$$

Outline

- Overview of hygienic macro system
- Difficulties of implementation for hygienic macro system to other languages
- Extensible parser
- JavaScript ⇔ Scheme Translator
- Examples
- Conclusion and future work

Example1: Let macro

```
define_expression Let() {
  identifier: id;
  expression: expr, body;
  {
      Let((id, expr), body) =>
      ((function (id) {
                  return body;
              })(expr));
  }
};
```

Example2: Or macro

```
define_expression Or() {
  identifier: tmp;
  expression: exp1, exp2;
  {
                    Or() => false;
                Or(exp1) => exp1;
   Or(exp1, exp2, ...) =>
        Let((tmp, exp1),
            tmp ? tmp : Or (exr2, ...));
   }
};

var tmp = 10;
Or(false, tmp);
```

Expanded form: Or macro

```
{
  var _L12 = function (tmp_26_28_){
    return ((tmp_26_28_) ?
            (tmp_26_28_) :
            (tmp));
  };
  var tmp = 10;
  _L12(false);
}
```

Create a new object by macro

```
define_expression new() {
  identifier: proto, specs;
  {
      new(proto, specs) =>
      ((function () {
          var new_obj = { __proto__: proto };
          var id;
          for (id in specs) new_obj[id] = specs[id];
          return new_obj;
      })());
   }
};
var rectangle = { height:10, width:20 };
var red_rectangle = new(rectangle, { color:"red" });
red_rectangle.height; // =>10
red_rectangle.color;  // =>"red"
```

Limitations

- Macros that are expanded to statement or operator is now implementing.
  - We can write only expression macro now.
- Dynamically extensible parser is imperfect.
  - The syntax of macro usages is restricted to function-call-like form.

Outline

- Overview of hygienic macro system
- Difficulties of implementation for hygienic macro system to other languages
- Extensible parser
- JavaScript ⇔ Scheme Translator
- Examples
- Conclusion and future work

Conclusion

- We show how to implement Scheme-like macro system to other languages.
  - Usage of Scheme macro expander to our macro system.
  - Making the modularized parser by top down operator precedence parser.
  - Introduction of translation rules to avoid semantic differences between two languages.

Future work

- Making dynamic extensible parser work.
- To implement statement and operator macro.
- To implement this system to other languages (such as C, Java, Ruby, etc.)