EPP: Extensible Preprocessor Kit

Yuuji ICHISUGI

Electrotechnical Laboratory

1-1-4 Umezono, Tsukuba,
Ibaraki, Japan 305.

ichisugi@etl.go.jp

What is EPP?

- EPP is an application framework for preprocessors.
- EPP is a Java preprocessor which can be extened by EPP plugins.
- End user can use multiple EPP plugins simultaniously.
- EPP plugins can extend the syntax in a wide range.
- The architecture of EPP is applicable to other languages includeing C, C++ and extended C++.

What can EPP do for Parallel C++/Java?

- EPP supports prototype implementation of translator of parallel C++/Java.
- EPP can be used as a preprocessor of extended C++/Java. (It can be implemented by only defining some new grammer rules.)

Outline of this presentation

- Overview of EPP
- Examples of EPP Plugins
- Description language Ld-2 and system mix-ins
- The architecture of EPP
- Conclusion and Future work

EPP: Extensible preprocessor kit

- Recursive descent style parser. (Left recursion is allowed.)
- Grammer rules are written as modules.
- Macros.
- EPP loads specified plugins before starting preprocessing.
- Written in Common Lisp (but currently rewriting it to Java.)

EPP Plug-in

Original Source Code:

```
#epp load "swap"
public class test {
  public static void main(String[] argv){
    int a = 1, b = 2;
    swap(int, a, b);
  }
}
```

Translated Source Code:

```
public class test {
  public static void main(String argv[]){
    int a = 1, b = 2;
    System.out.println(((("a = ") + (a)) + (", b = ")) + (b));
    {int tmp; tmp = (a); a = (b); b = (tmp); };
    System.out.println(((("a = ") + (a)) + (", b = ")) + (b)); }}
```

The source code of the SWAP plugin

```
(defsystem <<swap>> ()
  (class <startup> ()
    (method :init-funcall-style-macro-table ()
      [:original]
      [:add-funcall-style-macro
       '|swap|
       #'(lambda (tree args)
           (check-args-length 3 args)
           (let ((type (first args))
                 (a (second args))
                 (b (third args)))
             `(block (decl (modifiers) ,type
                          (vardecls (id |tmp|)))
               (exp-statement (= (id |tmp|) ,a))
               (exp-statement (= ,a ,b))
               (exp-statement (= ,b (id |tmp|))))
             ))])))
```

Python style comparison operators

Original Source Code:

```
#epp load "pcomp"
public class test {
  public static void main(String[] argv){
    int a = 10;
    if (0 <= a < 100) System.out.println("Correct!");
  }
}
```

Translated Source Code:

```
public class test {
  public static void main(String argv[]){
    int a = 10; int T0;
    if((((0) <= (T0 = (a))) && ((T0) < (100))))
       System.out.println("Correct!");
  }
}
```

- The plugin can remove some original grammer rules.

Defmacro plugin

```
#epp load "defmacro"

defmacro max(a, b)  (a > b ? a : b)
defmacro swap(t, a, b) {
  t tmp; tmp = a; a = b; b = tmp;
}

public class test {
  public static void main(String argv[]){

    System.out.println(max(5, 10));

    int a = 1, b = 2;
    swap(int, a, b);
  }
}
```

Tiny Data parallel Java

```
#epp load "datap0"

public class datap0_pi {
  public static void main(String argv[]){
    int num = 1000;
    parallel.width = 1.0 / num;
    parallel p = new parallel(num);
    p.run();
    System.out.println(parallel.result
                         * parallel.width);
  }
}

class parallel {
  static double result = 0;
  static double width;
  double x;

  void run(){
    x = (VPID() + 0.5) * width;
    set_sum(result, 4.0 / (1.0 + x * x));
  }
}
```

Translated tiny data-parallel Java code

- Translated using same techniques as Data-parallel C [Hatcher and Quinn 1991].
- Virtual processors are emulated by some threads and loops.

```
void run(){
  for (vpi = (0); (vpi) < (vpn); vpi++) {
    (vp_x)[vpi] = (((((PE_i) * (vpn)) + (vpi)) + (0.5)) * (width));
  }
  sync();
  for (vpi = (0); (vpi) < (vpn); vpi++) {
    {
      double T0 = 0.0;
      for (vpi = (0); (vpi) < (vpn); vpi++)
        T0 += ((4.0) / ((1.0) + (((vp_x)[vpi])
                                    * ((vp_x)[vpi]))));
      result = (calc_sum(T0));
      break;
    }
  }
}
```

Plugins currently provided

- Swap macro (32 lines)
- Assert macro (45 lines)
- New operators for user defined class (114 lines)
- Pseudo call by reference (63 lines)
- Source code scrambring tool (159 lines)
- Python like comparison operators (100 lines)
- Defmacro (76 lines)
- Tiny data parallel Java (641 lines)

(Sizes includes comment lines.)

Background

- Many kind of language extentions are proposed.
  However, the end-user must select only one implementation.
- Many kind of extended C++/Java are implemented as preprocessors.
- Low extensibility of existent languages (compared to Lisp).

Related work

Sage++[Bodin 1994], MPC++[Ishikawa 1994], OpenC++[Chiba 1995]

Advantages:

1. Provides tree manipuration primitives and libraries.
2. Extensions can be provided as a separated modules.
3. Syntax can be extended.

However,

1. Extensibility of syntax is resticted.
2. Metalevel architectures are based on C++ inheritance model. (Reusability is worse than CLOS, for example.)

Structure of EPP

```
+------------------------------+
|   (Application programs)     |
+------------------------------+
|         EPP Plugins          |
+------------------------------+
|     Java grammar rules       | (1957 lines)
+------------------------------+
|         EPP core             | ( 597 lines)
+------------------------------+
|     Ld-2 (An OO language)    | ( 764 lines)
+------------------------------+
|        Common Lisp           |
+------------------------------+
```

Description Language Ld-2

- Ld-2 is a simple Object-Oriented language implemented in Common Lisp.
- Ld-2 has a unique inheritance mechanism, **system mixin**.

System mixins will enhance flexibility and reusability of codes.

A Monolithic method definition

A method containing nested **if** statements.

```
(class <foo> ()
  (defmethod :m (data)
    (if (eq data 'B)
        (do-B)
        (if (eq data 'A)
            (do-A)
            (do-default-behavior)))))
```

To add a new if-then-else clause, the programmer must

- edit the source code or,
- define new class which inherit the original class.

System Mixin

```
(defsystem <<skeleton>> ()
  (class <foo> ()
    (defmethod :m (data)
      (do-default-behavior))))

(defsystem <<A>> ()
  (class <foo> (data)
    (method :m (data)
      (if (eq data 'A)
          (do-A)
          [:original data]))))

(defsystem <<B>> ()
  (class <foo> (data)
    (method :m (data)
      (if (eq data 'B)
          (do-B)
          [:original data]))))

(setup (list <<B>> <<A>> <<skeleton>>))
```

System Mixin (cont.)

System mixins are

- simillar to mixins used in Flavors or CLOS, but the unit of inheritance is not a class but the whole system.
- modules (units of resuse), that is completely orthogonal to classes.
- patch files (with less probability of conflicts.)
- simillar to conditional compilation: #ifdef (with smaller grain and high possibility of combinations).

Three passes of EPP

1. The parsing pass.
2. The macro expansion pass.
3. The code-emitting pass.

The plugin programmer can add:

- new grammer rules.
  (Statements, prefix/postfix operators, left/right associative operators.)
- new macro nodes which are expanded into plain Java nodes.
- new passes before or after each pass.

A definition of a non-terimnal.

```
(defsystem <<exp>> ()
  (class <startup> ()

    (defmethod :exp ()
      (let ((tree [:exp-top])
            new-tree)
        (loop ; Infinite loop.
         (setq new-tree [:exp-left tree])
         (if (eq new-tree 'no-left)
             (return)) ; Exit from the loop.
         (setq tree new-tree))
        ;; Return value of this method.
        tree))

    (defmethod :exp-top ()
      [:exp-right [:exp-1]])

    (defmethod :exp-right (tree) tree)

    (defmethod :exp-left (tree) 'no-left)

    (defmethod :exp-1 () [:term])
   ))
```

Unary Operators

```
;; a prefix operator
(defsystem <<unary-plus>> ()
  (class <startup> ()
    (method :exp-top ()
      (if (eq [:lookahead] '+)
          (progn
             [:match-any]
             (list 'unary+ [:exp]))
          [:original]))))

;; a postfix operator
(defsystem <<post++>> ()
  (class <parser> ()
    (method :exp-left (tree)
      (if (eq [:lookahead] '++)
          (progn
            [:match-any]
            (list 'post++ tree))
          [:original tree]))))
```

Binary Operators

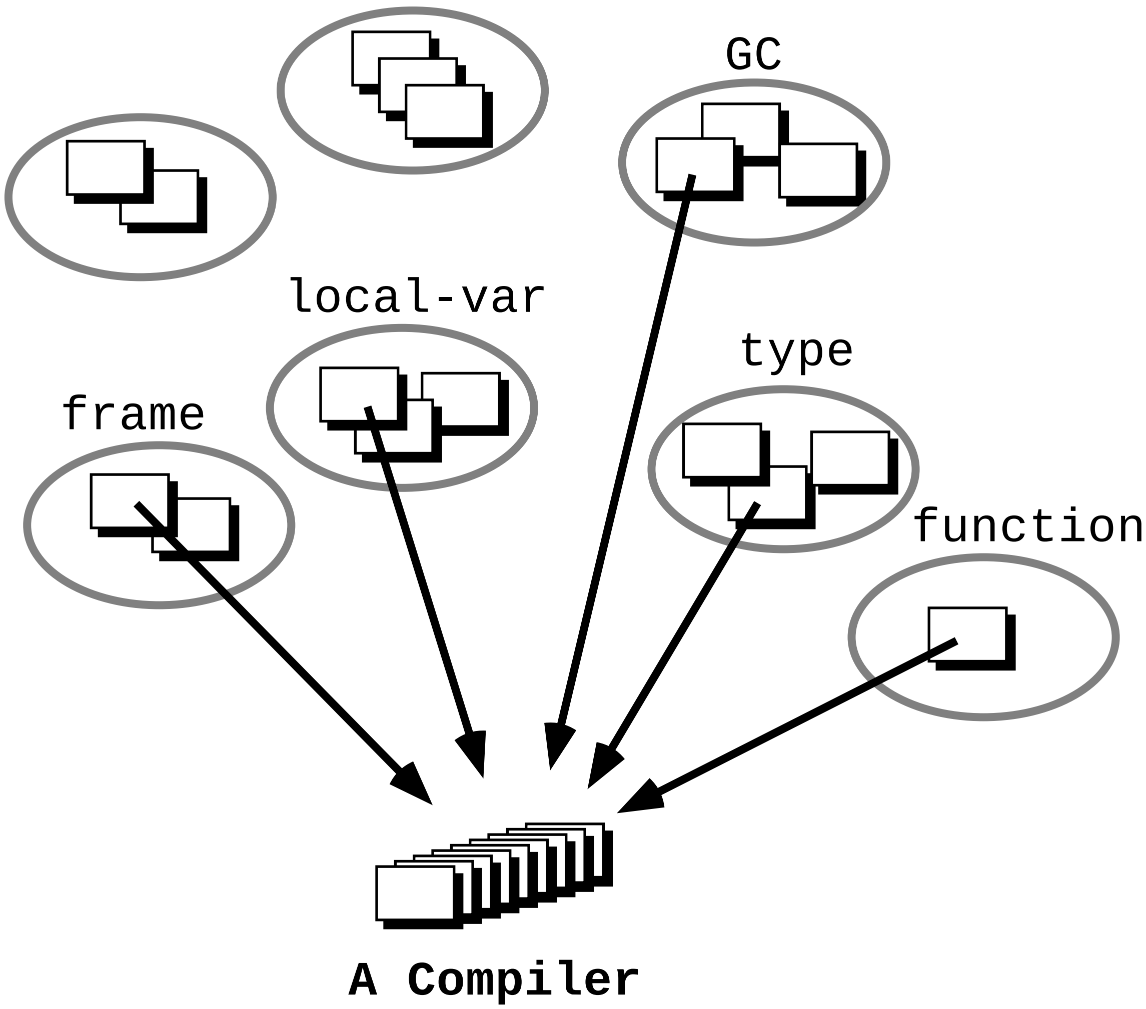
```
;; a left associative binary operator
(defsystem <<plus>> ()
  (class <startup> ()
    (method :exp-left (tree)
      (if (eq [:lookahead] '+)
          (progn
            [:match-any]
            (list '+ tree [:exp-1]))
          [:original tree]))))

;; a right associative binary operator
(defsystem <<assign>> ()
  (class <startup> ()
    (method :exp-right (tree)
      (if (eq [:lookahead] '=)
          (progn
            [:match-any]
            (list '= tree [:exp]))
          [:original tree]))))
```

Advantage of this parser wrinting style

The mechanism of system mixin maximizes both reusability of grammar rules and degree of freedom of implementation.

- Reusability
  Each grammar rule can be written as a system mixin that has small dependency to the other grammar rules.
- Degree of freedom of implementation
  Each grammar rule is written as a program, not a restricted declarative notation.

Extensible Compiler Kit



Conclusion

- EPP is an application framework for preprocessors.
- EPP is a Java preprocessor that can be extended by EPP plugins.
- End user can use multiple EPP plugins simultaniously.
- EPP plugins can extend the syntax.
- EPP consists of resusable modules.

Future work

1. EPP on Java (extended by EPP).
2. Type check pass that deals with type imformation and scope rules.
3. Extensible lexical analizer.
4. C/C++ as input/output language of EPP.