Adapting Scheme-Like Macros to a C-Like Language

Kevin Atkinson, Matthew Flatt

University of Utah

ZL

- Adopts a Scheme-like approach to build C++ from a C like core
- Why C?
  - *The* system's programming language
  - Want to make life better in that world

Challenges

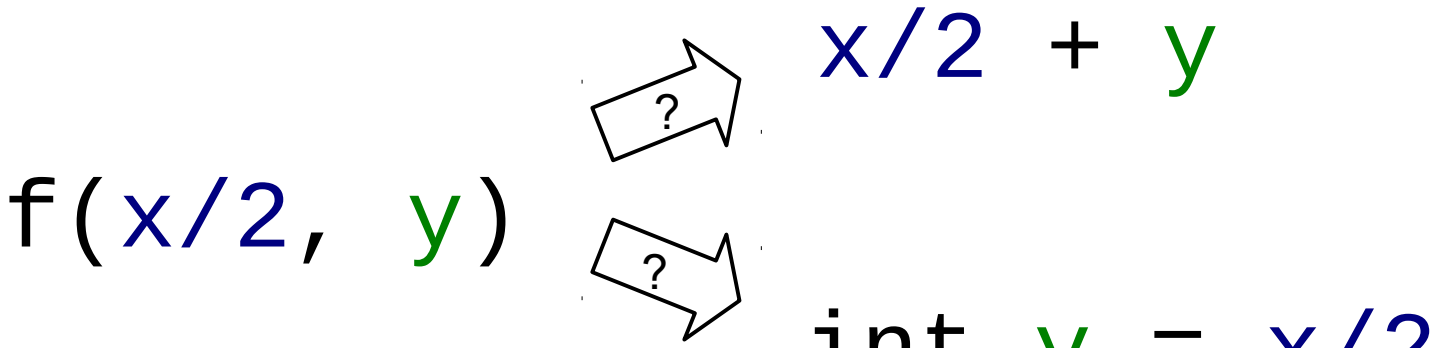
1. Parsing C Idiosyncratic Syntax While Also Allowing The Syntax to be Extensible
2. Finding Right Hygiene Model
3. Finding Right Reflective Operations

How to Parse This Expression?

```
f(x/2, y)
```

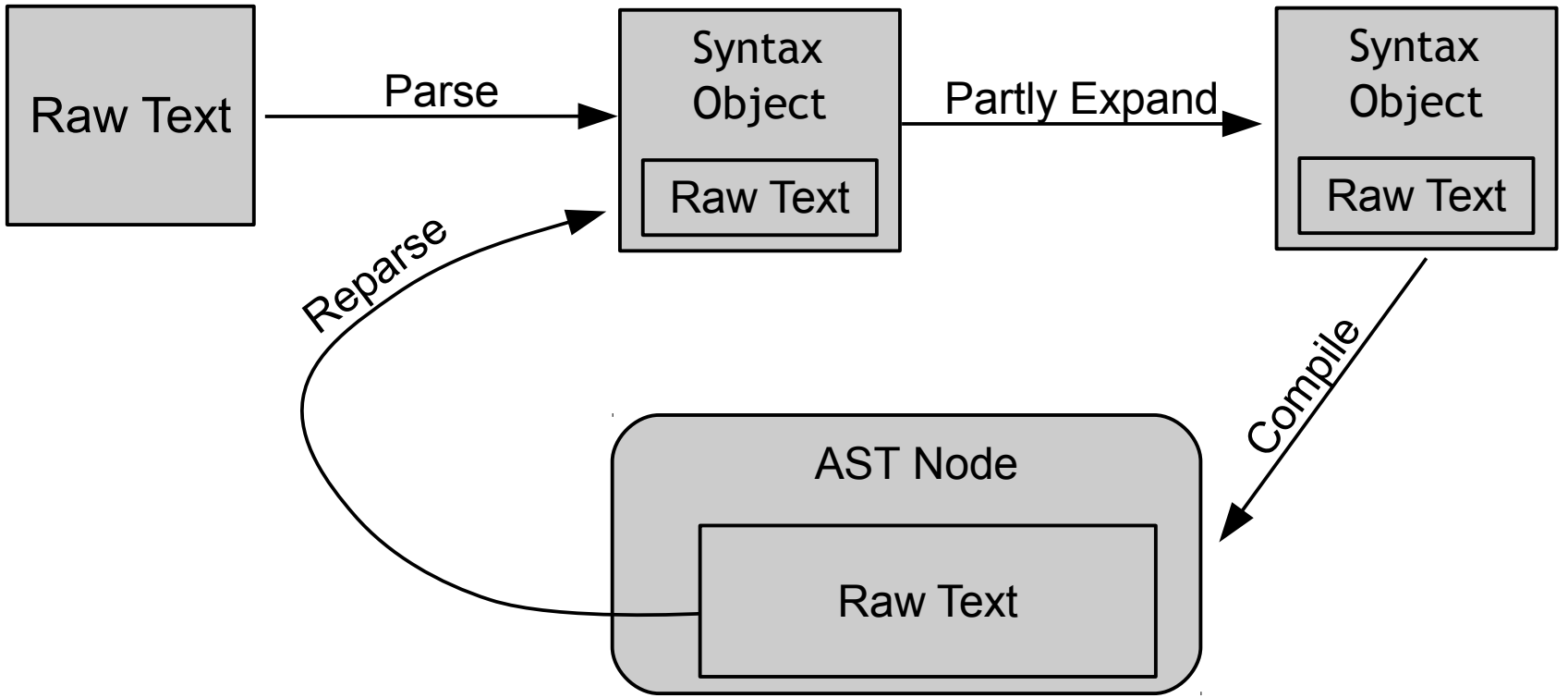
Function Call?

Macro Invocation?



Parsing Overview

- ZL doesn't parse in a single linear pass
- Iterative-deepening approach



Parsing Details

```
"inline int f() {int x = 10; return x;}
 int main() {return f();}"
```

Parse

```
(@ (stmt inline int f ('()' "")
                      ('{}' "int x = 10; return x;")
   (stmt int main ('()' "") ('{}' "return f();")))
```

Expand & Compile

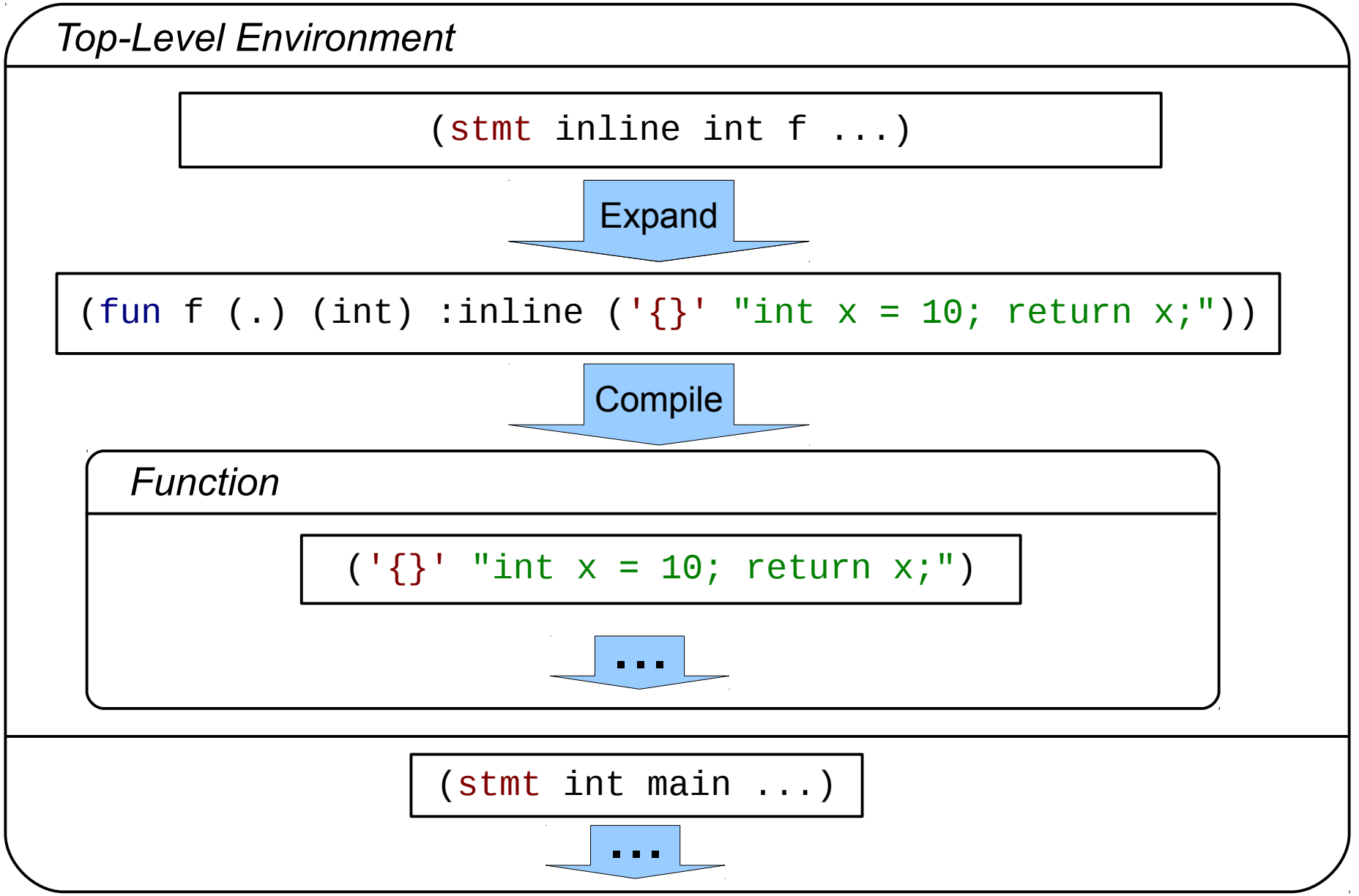
Top-Level Environment

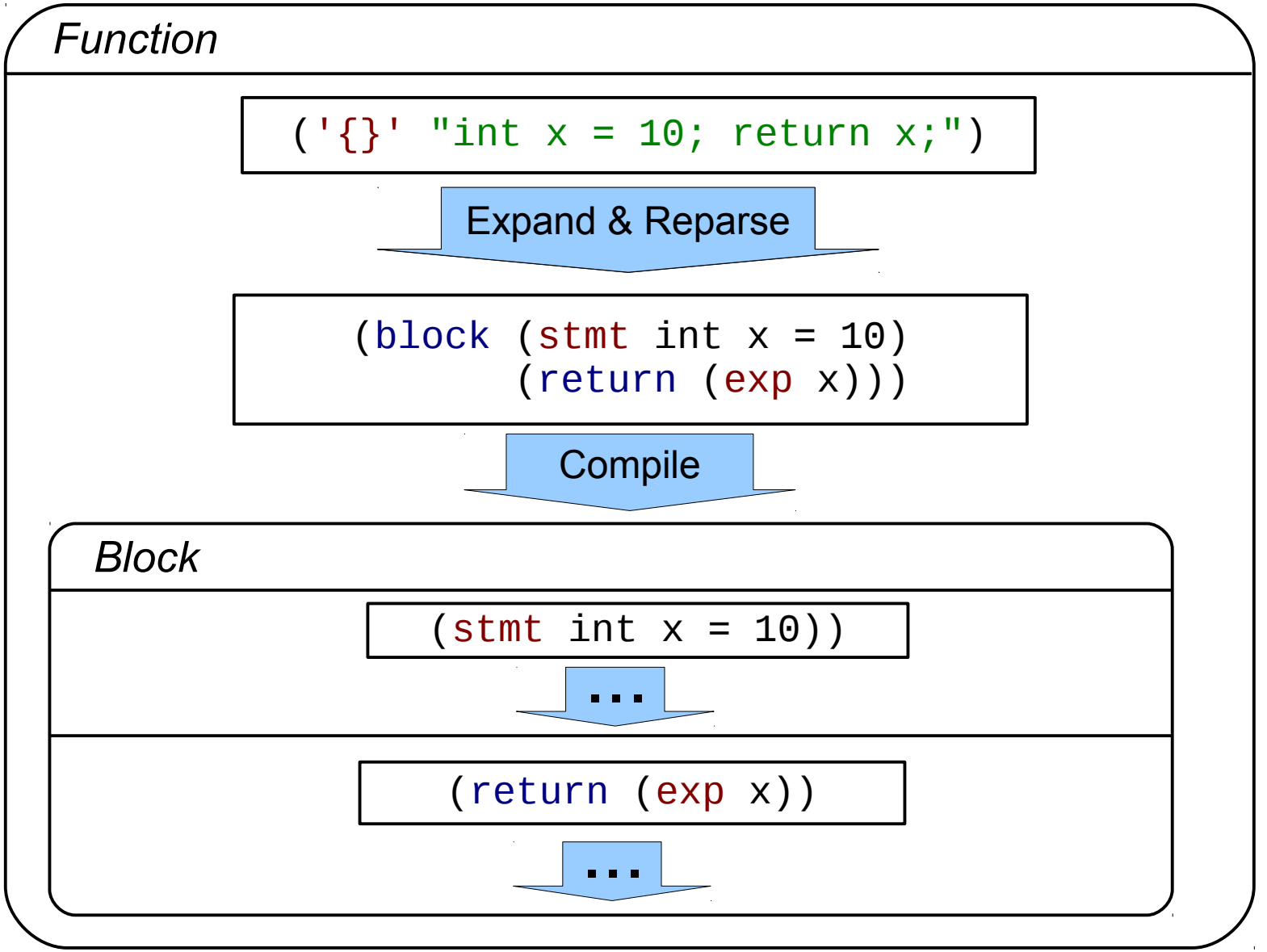
```
(stmt inline int f ...)
```

...

```
(stmt int main ...)
```

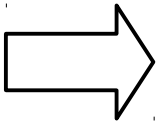
...


Top-Level Environment
(stmt inline int f ...)
Expand
(fun f (.) (int) :inline ('{}' "int x = 10; return x;"))
Compile
Function
('{}' "int x = 10; return x;")
...
(stmt int main ...)
...


Function
('{}' "int x = 10; return x;")
Expand & Reparse
(block (stmt int x = 10)
(return (exp x)))
Compile
Block
(stmt int x = 10))
...
(return (exp x))
...

Pattern Macros

macro or(*x*, *y*) { ({typeof(*x*) t = *x*; t ? t : *y*;}); }

or(0.0,t)  ({typeof(0.0) t_0 = 0.0; t_0 ? t_0 : t;});

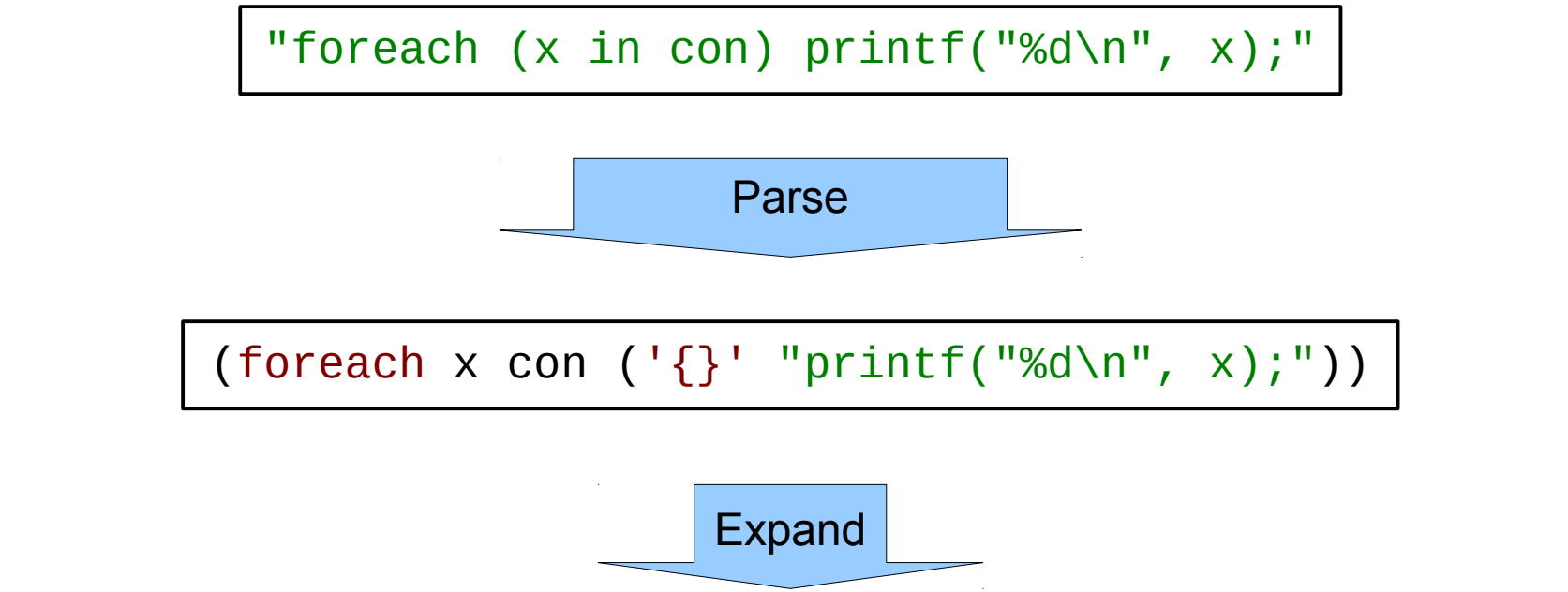
Syntax Macros

Add to PEG Grammer:

```
<foreach> "foreach" "(" {ID} "in" {EXP} ")" {STMT}
```

In Source Code:

smacro foreach (id, container, body) {...}



Procedural Macros

```
Syntax * or(Syntax * syn, Environ *) {
  Match * m = match(NULL, syntax (_, x, y), syn);
  return replace(syntax
                   {({typeof(x) t = x; t ? t : y;});},
                 m, new_mark());
}

make_macro or;
```

Syntax Forms:	syntax	Callbacks:	match
	make_macro		replace
	new_mark		error

Procedural Macro Example

```
float pi = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {
  Mark * mark = new_mark();
  Match * m = match(NULL, syntax (_, R), syn);
  UnmarkedSyntax * r = syntax { ({float r = R; pi*r*r;}); };
  return replace (r, m, mark);
}
make_macro area_circle;

int main() {
  float pi = 3.14;
  float r = 10;
  ... area_circle(r) ...
}
```

Expansion of `area_circle(r)`

```
float pi = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {
  Mark * mark = new_mark();
                ('0, pi => ... )
  Match * m = match(NULL, syntax (_, R), syn);
  UnmarkedSyntax * r = syntax { ({float r = R; pi*r*r;}); };
  return replace (r, m, mark);
}
make_macro area_circle;

int main() {
  ...
  ... area_circle(r) ...
}
```

Expansion of `area_circle(r)`

```
float pi = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {
  Mark * mark = ('0, pi => ... )
  Match * m = match(NULL, syntax (_, R), syn);
                [R => r]
  UnmarkedSyntax * r = syntax { ({float r = R; pi*r*r;}); };
  return replace (r, m, mark);
}
make_macro area_circle;

int main() {
  ...
  ... area_circle(r) ...
}
```

Expansion of `area_circle(r)`

```
float pi = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {
  Mark * mark = ('0, pi => ... )
  Match * m = [R => r]
  UnmarkedSyntax * r = syntax { ({float r = R; pi*r*r;}); };
  return replace (r, m, mark);
}
make_macro area_circle;

int main() {
  ... area_circle(r) ...
}
```

Expansion of `area_circle(r)`

```
float pi = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {
  Mark * mark = ('0, pi => ... )
  Match * m = [R => r]
  UnmarkedSyntax * r = syntax { ({float r = R; pi*r*r;}); };
  return replace (r, m, mark);
                 (syntax { ({float r = R; pi*r*r;}); },
                  [R => r],
                  ('0, pi => ... ));
}
make_macro area_circle;

int main() {
  ... area_circle(r) ...
}
```

Expansion of `area_circle(r)`

```
float pi = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {
  ...
  return replace (syntax { ({float r = R; pi*r*r;}); },
                  [R => r],
                  ('0, pi => ... ));
}
make_macro area_circle;

int main() {
  ...
  ... area_circle(r) ...
  ... ({ float r'0 = r; pi'0 * r'0 * r'0; }) ...
}
```

Hygiene System

```
float pi = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {
  Mark * mark = new_mark();
  Match * m = match(NULL, syntax (_, R), syn);
  UnmarkedSyntax * r = syntax { ({float r = R; pi*r*r;}); };
  return replace (r, m, mark);
}
make_macro area_circle;

int main() {
  float pi = 3.14;
  float r = 10;
  ... area_circle(r) ...
}
```

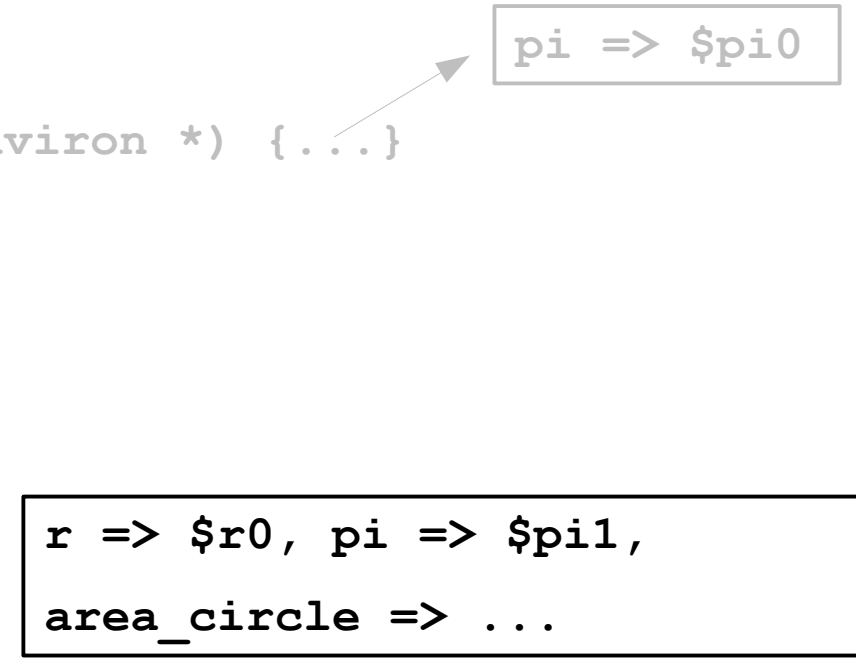
```
({float r = r; pi * r * r;})
```

```
float pi $pi0 = 3.14159;
```

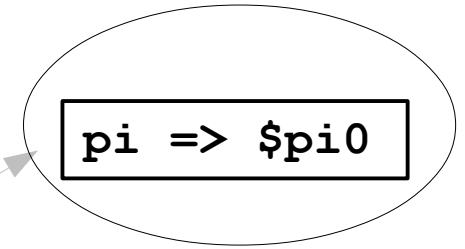
pi => $pi0

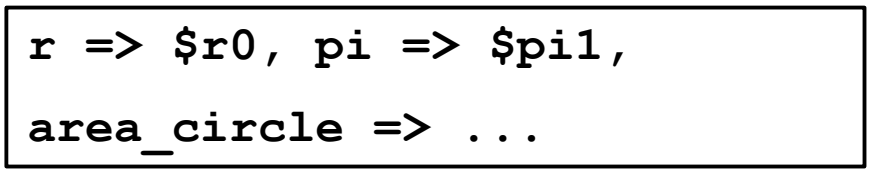
```
float pi $pi0 = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {
  Mark * mark = new_mark();
  ...
}
make_macro area_circle;
```

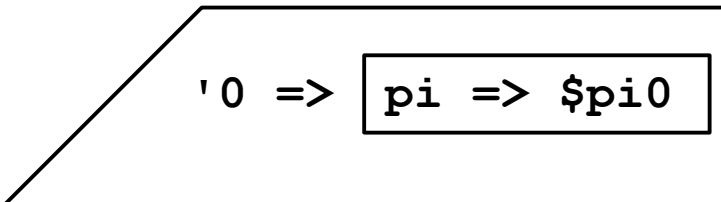
```
float pi $pi0 = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {...}

int main() {
  float pi $pi1 = 3.14;
  float r $r0 = 10;

  ... area_circle(r) ...

}
```



```
float pi $pi0 = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {...}

int main() {
  float pi $pi1 = 3.14;
  float r $r0 = 10;

  ... area_circle(r) ...
  ... ({ float r'0 = r; pi'0 * r'0 * r'0; }) ...
}
```







```
float pi $pi0 = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {...}

int main() {
  float pi $pi1 = 3.14;
  float r $r0 = 10;

  ... area_circle(r) ...
  ... ({ float r'0 $r1 = r; pi'0 * r'0 * r'0; }) ...
}
```

pi => $pi0

r'0 => $r1, r => $r0, pi => $pi1,
area_circle => ...

Mark Becomes Part of The Name

'0 => pi => $pi0

```
float pi $pi0 = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {...}

int main() {
  float pi $pi1 = 3.14;
  float r $r0 = 10;

  ... area_circle(r) ...
  ... ({ float r'0 $r1 = r $r0; pi'0 * r'0 * r'0; }) ...
}
```

pi => $pi0

r'0 => $r1, r => $r0, pi => $pi1,
area_circle => ...

'0 => pi => $pi0

```
float pi $pi0 = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {...}

int main() {
  float pi $pi1 = 3.14;
  float r $r0 = 10;

  ... area_circle(r) ...
  ... ({ float r'0 $r1 = r $r0; pi'0 * r'0 * r'0; }) ...
}
```

pi => $pi0

r'0 => $r1, r => $r0, pi => $pi1,
area_circle => ...

Look Inside the Mark

'0 => pi => $pi0

```
float pi $pi0 = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {...}

int main() {
  float pi $pi1 = 3.14;
  float r $r0 = 10;

  ... area_circle(r) ...
  ... ({ float r'0 $r1 = r $r0; pi'0 $pi0 * r'0 * r'0; }) ...
}
```

pi => $pi0

r'0 => $r1, r => $r0, pi => $pi1,
area_circle => ...

Strip Mark

pi

'0 => pi => $pi0

```
float pi $pi0 = 3.14159;

Syntax * area_circle(Syntax * syn, Environ *) {...}

int main() {
  float pi $pi1 = 3.14;
  float r $r0 = 10;

  ... area_circle(r) ...
  ... ({ float r'0 $r1 = r $r0; pi'0 $pi0 * r'0 $r1 * r'0 $r1; })...
}
```

pi => $pi0

r'0 => $r1, r => $r0, pi => $pi1,
area_circle => ...

'0 => pi => $pi0

```
float $pi0 = 3.14159;
...

int main() {
  float $pi1 = 3.14;
  float $r0 = 10;
  ... ({ float $r1 = $r0; $pi0 * $r1 * $r1; }) ...
}
```

Everything Resolves Correctly

Bending Hygiene:
Replace Context

- *datum->syntax-object* =>
  - Contex * get_context(Syntax *)
  - Syntax * replace_context
    (UnmarkedSyntax *, Context *)

Bending Hygiene:
Fluid Binding

- *define-syntax-parameter* => fluid_binding
- *syntax-parameterize* => fluid

```
fluid_binding this;
macro m() {f(this);}
int main() {X * fluid this = ...; return m();}
```

Other API Functions

```
Syntax * foreach (Syntax * syn, Environ * env) {
  Syntax * con = ...;
  if (!symbol_exists(syntax begin, con, mark, env) ||
      ...
    return error(con,
                 "Container lacks proper method.");
  ...
}
make_syntax_macro foreach;

int main() {
  ...
  foreach(x in container) {printf("%d\n", x);});
  ...
}
```

- Additional API Functions and Examples in Paper

Results

- Used ZL to Mitigate Problems of:
  - Adding and Removing C++ fields and methods
  - Incomparable ABI's Due to Compiler Changes

    (See GPCE'10 Paper)
- Compile Time Only 2-3 slower than G++
- No Impact on Run-time Performance

Conclusion

- Presented Macro System that:
  - Handle C's rich syntax
  - Preserves Lexical Scope
  - Offers power of Scheme's syntax-case
- Parts of ZL Also Presented in GPCE'10:

  ABI Compatibility Through a Customizable Language
- Additional Parts Presented in my Dissertation
- Implementation Available:

  http://www.cs.utah.edu/~kevina/zl