Axioms as generic rewrite rules in C++ with concepts

Intro

- Optimizing compiler
- Transformations
  - i+0 → i
  - s + string("") → s
- Rewrite on built in types
- User-defined types

Generic Concepts

```
concept Monoid<typename Op, typename T> : SemiGroup <Op, T> {
  T identity_element (Op, T); // identity element
  axiom Identity (Op op, T x) {
    op (x, identity_element (op, x)) == x; // right identity law
    op (identity_element (op, x), x) == x; // left identity law
  }
}
concept_map Monoid<plus<int>, int>
  { int identity_element (plus<int> op, int x) { return 0; } };
concept_map Monoid<multiplies<int>, int>
  { int identity_element (multiplies<int> op, int x) { return 1; } };
concept_map Monoid<plus<string>, string>
  { string identity_element (plus<string> op, string x) { return std::string(""); } };
```

Rewrite rules

- Rewrite rules

$$x + \text{string}(\text{""}) \rightarrow x$$

- Conditional rewrite rules

$$x + y \mid \{ \text{def}(y) = \text{string}(\text{""}) \} \rightarrow x$$

```
string x ("text");
string z = x + string ("");
```

(a)

```
string x ("text");
string y ("");
string z = x + y;
```

(b)

```
string add (string a, string b)
  { return a + b; }

void main () {
  string y ("");
  string x ("text");
  add (x, y);
}
```

(c)

Generating rewrite rules

Instantiate axioms

```
axiom Identity (plus<string> op, string x) {
  op (x, Monoid<plus<string>, string>::identity_element (op, x)) == x;
  op (Monoid<plus<string>, string>::identity_element (op, x), x) == x;
}
```

Extract rewrite rules from axiom and divide a rule into two functions: LHS and RHS

```
string rule_string_identity_lhs (plus<string> op, string x) {
  return op (x, Monoid<plus<string>, string>::identity_element(op, x));
}

string rule_string_identity_rhs (plus<string> op, string x) { return x; }
```

Generating rewrite rules

```
string rule_string_identity_lhs (plus<string> op, string x) {
   return op (x, Monoid<plus<string>, string>::identity_element(op, x));
}
string rule_string_identity_rhs (plus<string> op, string x) { return x; }
```

Translate a rule function into an intermediate representation

```
string rule_string_identity_lhs (plus<string> op, string x) {
   t1 = Monoid<plus<string>, string>::identity_element(op, x);
   t2 = op(x, t1);
   return t2;
}
string rule_string_identity_rhs (plus<string> op, string x) { return x; }
```

Generating rewrite rules

```
string rule_string_identity_lhs (plus<string> op, string x) {
   t1 = Monoid<plus<string>, string>::identity_element(op, x);
   t2 = op(x, t1);
   return t2;
}
string rule_string_identity_rhs (plus<string> op, string x) { return x; }
```

Eliminate function abstractions

```
string rule_string_identity_lhs (plus<string> op, string x) {
   t1 = string("");
   t2 = operator+(x, t1);
   return t2;
}

string rule_string_identity_rhs (plus<string> op, string x) { return x; }
```

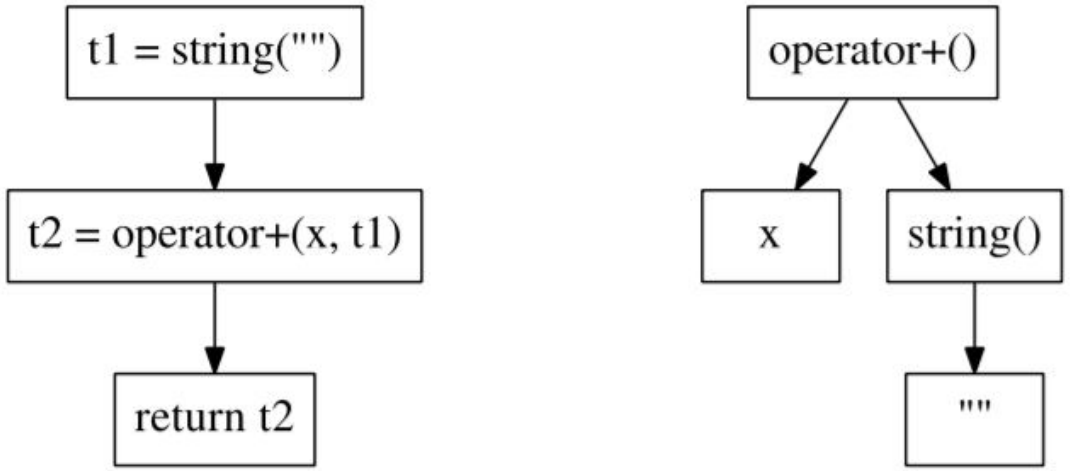
$x + y \mid \{ def(y) = string("") \} \rightarrow x$

Generating rewrite rules

```
string rule_string_identity_lhs (plus<string> op, string x) {
  t1 = string("");
  t2 = operator+(x, t1);
  return t2;
}

string rule_string_identity_rhs (plus<string> op, string x) { return x; }
```

Construct the rule patterns and put them in the rule repository



Applying rewrite rules

- For a) and b) match rule pattern with AST of each statement
- For c) first inline then match
- Abstraction index

$$\phi(f) = \max\big(\phi(g_1), \ldots, \phi(g_n)\big) + 1,$$

$$\phi(\text{main}) = \max(\phi(\text{string}()), \phi(\text{add})) + 1 = 3.$$

$$\phi(f) < \phi(r)$$

```
string x ("text");
string z = x + string ("");
```

(a)

```
string x ("text");
string y ("");
string z = x + y;
```

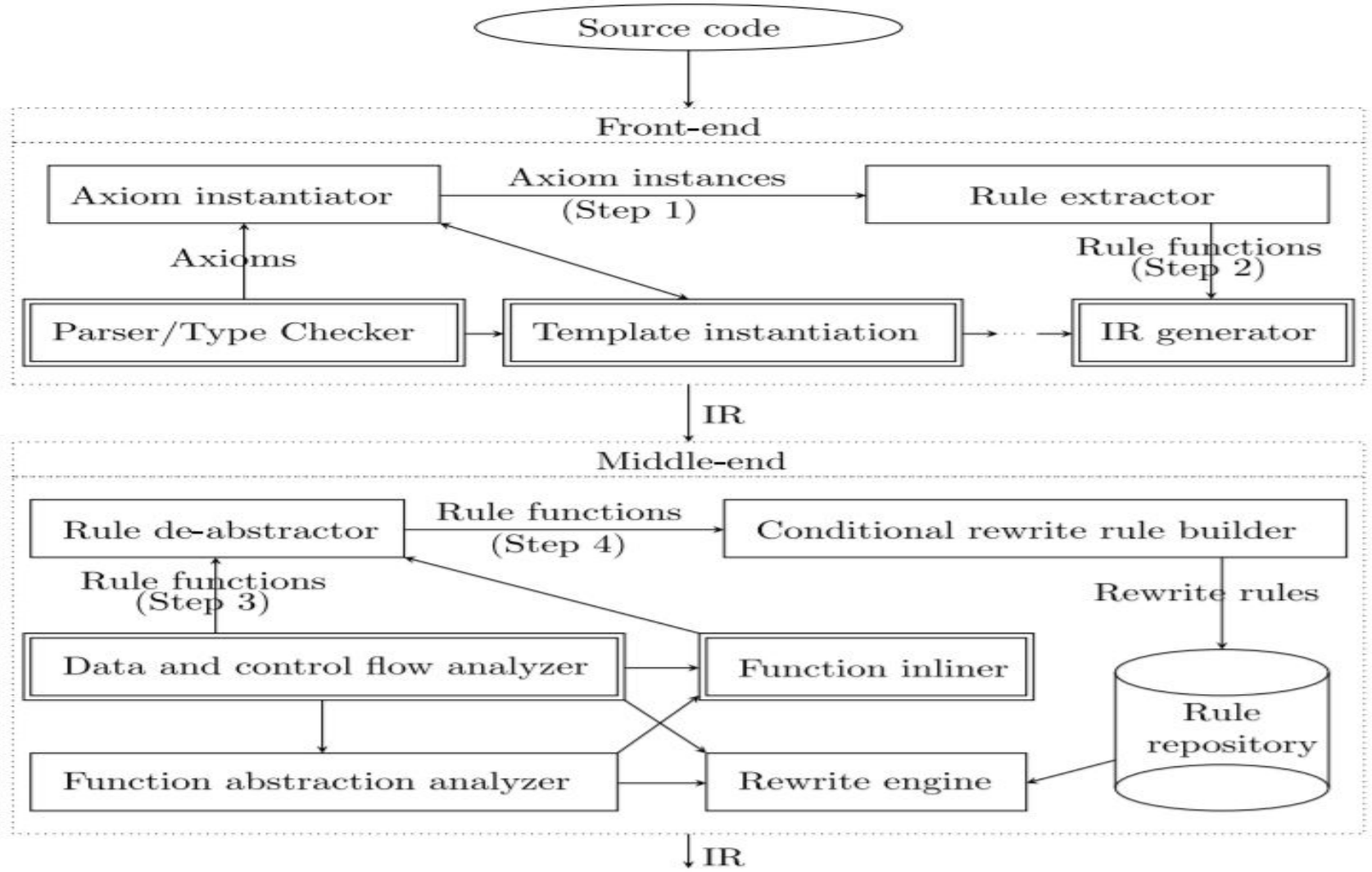
(b)

```
string add (string a, string b)
  { return a + b; }

void main () {
  string y ("");
  string x ("text");
  add (x, y);
}
```

(c)

Processing pipeline



Evaluation

- Abstraction penalty: ratio of the execution time of an abstracted implementation over a direct implementation
- Wraps a varying number of double values into user-defined classes
- A is with optimizations, B without

Test	D1(A)	D1(B)	D2(A)	D2(B)	D4(A)	D4(B)
add zero	1.00	1.25	0.99	1.65	1.01	1.70
zero minus	1.00	1.32	0.99	1.89	1.01	1.79
times one	1.00	1.00	0.99	1.00	1.00	0.99
mixed algebra	1.03	1.63	0.99	2.36	1.00	2.42

- Compilation time increased by a factor of 1.0035