CMPT 379
Compilers

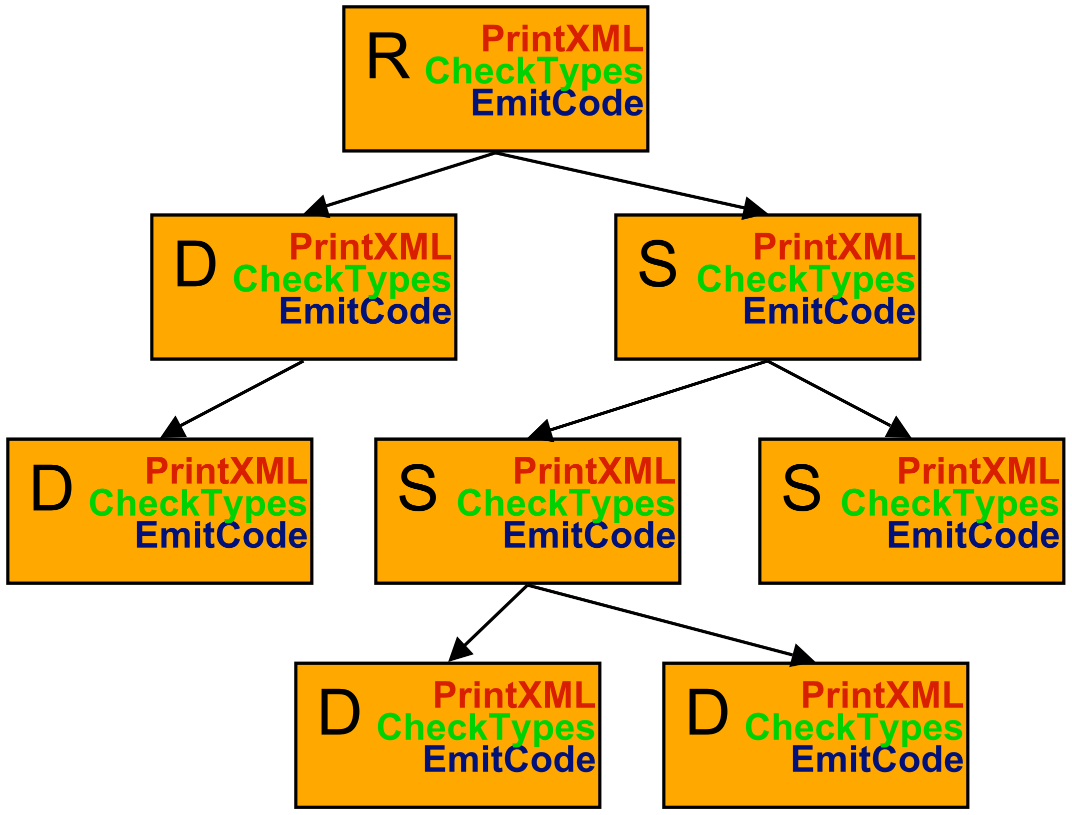
Anoop Sarkar

`http://www.cs.sfu.ca/~anoop`

Visitors

- A compiler pass is usually a traversal of the syntax tree
  - with actions performed on all or some nodes
- Visitors are a commonly used design pattern in compilers that provides an alternative to adding one method per pass
  - instead, have one visitor per pass
- Advantages + Disadvantages

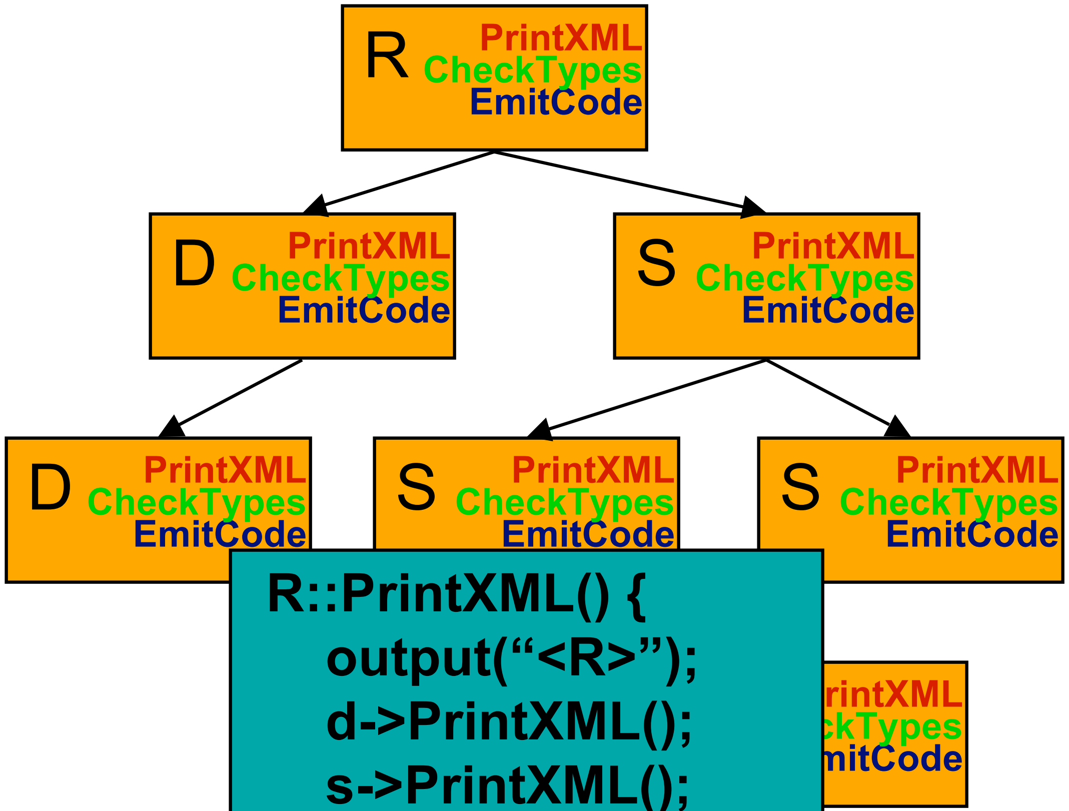
Tree traversal w/o visitors



R PrintXML CheckTypes EmitCode
D PrintXML CheckTypes EmitCode
S PrintXML CheckTypes EmitCode
D PrintXML CheckTypes EmitCode
S PrintXML CheckTypes EmitCode
S PrintXML CheckTypes EmitCode
D PrintXML CheckTypes EmitCode
D PrintXML CheckTypes EmitCode

```
class R {
  virtual PrintXML();
  virtual CheckTypes();
  virtual EmitCode();
}
class S {
  virtual PrintXML();
  virtual CheckTypes();
  virtual EmitCode();
}
class D {
  virtual PrintXML();
  virtual CheckTypes();
  virtual EmitCode();
}
```

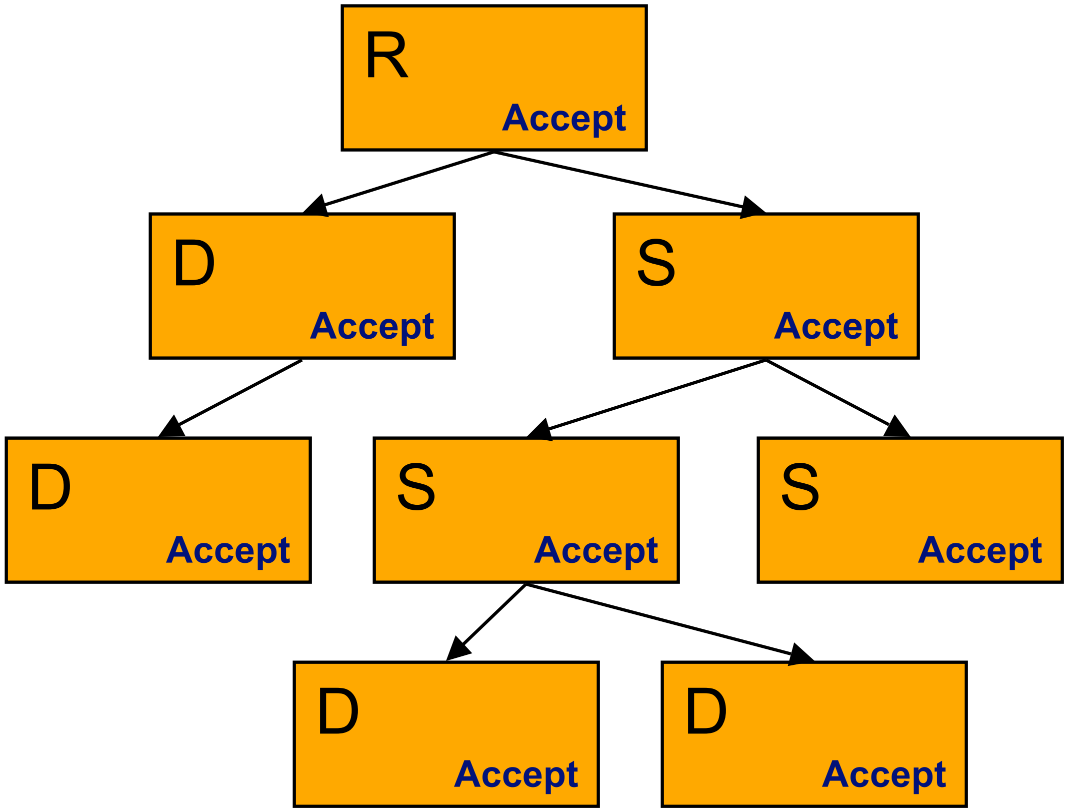
Tree traversal w/o visitors



```
R::PrintXML() {
    output("<R>");
    d->PrintXML();
    s->PrintXML();
    output("</R>");
}
```

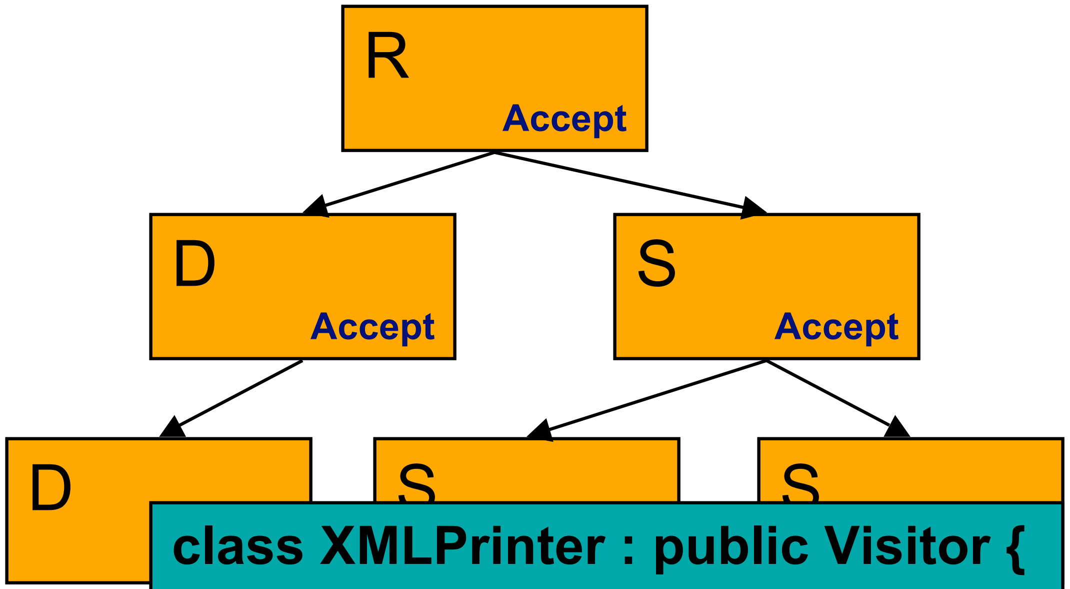
```
class R {
   virtual PrintXML();
   virtual CheckTypes();
   virtual EmitCode();
}
class S {
   virtual PrintXML();
   virtual CheckTypes();
   virtual EmitCode();
}
class D {
   virtual PrintXML();
   virtual CheckTypes();
   virtual EmitCode();
}
```

Tree traversal with visitors



```
class R {
   virtual Accept(Visitor *);
}
class S {
   virtual Accept(Visitor *);
}
class D {
   virtual Accept(Visitor *);
}
class Visitor {
   virtual Visit(R *) = 0;
   virtual Visit(S *) = 0;
   virtual Visit(D *) = 0;
}
```

Tree traversal with visitors



```
class XMLPrinter : public Visitor {
    virtual Visit(R *r) {
        output("<R>");
        r->d->Accept(this);
        r->s->Accept(this);
        output("</R>");
    }
}
```

```
class R {
    virtual Accept(Visitor *);
}
class S {
    virtual Accept(Visitor *);
}
class D {
    virtual Accept(Visitor *);
}
class Visitor {
    virtual Visit(R *) = 0;
    virtual Visit(S *) = 0;
    virtual Visit(D *) = 0;
}
```

Virtual methods vs. Visitors

- Virtual methods
  - R::PrintXML()
  - R::CheckTypes()
  - R::EmitCode()
  - S::PrintXML()
  - S::CheckTypes()
  - S::EmitCode()
  - D::PrintXML()
  - D::CheckTypes()
  - D::EmitCode()

- Visitors
  - XMLPrinter::Visit(R*)
  - XMLPrinter::Visit(S*)
  - XMLPrinter::Visit(D*)
  - CheckTypes::Visit(R*)
  - CheckTypes::Visit(S*)
  - CheckTypes::Visit(D*)
  - EmitCode::Visit(R*)
  - EmitCode ::Visit(S*)
  - EmitCode ::Visit(D*)

Visitor Pattern

- All Nodes must accept visitors. Why?

```
struct NonTerminal : Symbol {
    virtual void Accept(ASTVisitor *v) {
        v->Visit(this);
    }};
```

```
SomeASTVisitor v;
Symbol *nt;
nt = getSymbol();
nt->Accept(&v);
```

Why so?

```
SomeASTVisitor v;
Symbol *nt;
nt = getSymbol();
v.Visit(nt);
```

And not like so?