Ada Semantic Interface Specification

Peter C. Chapin
Vermont Technical College

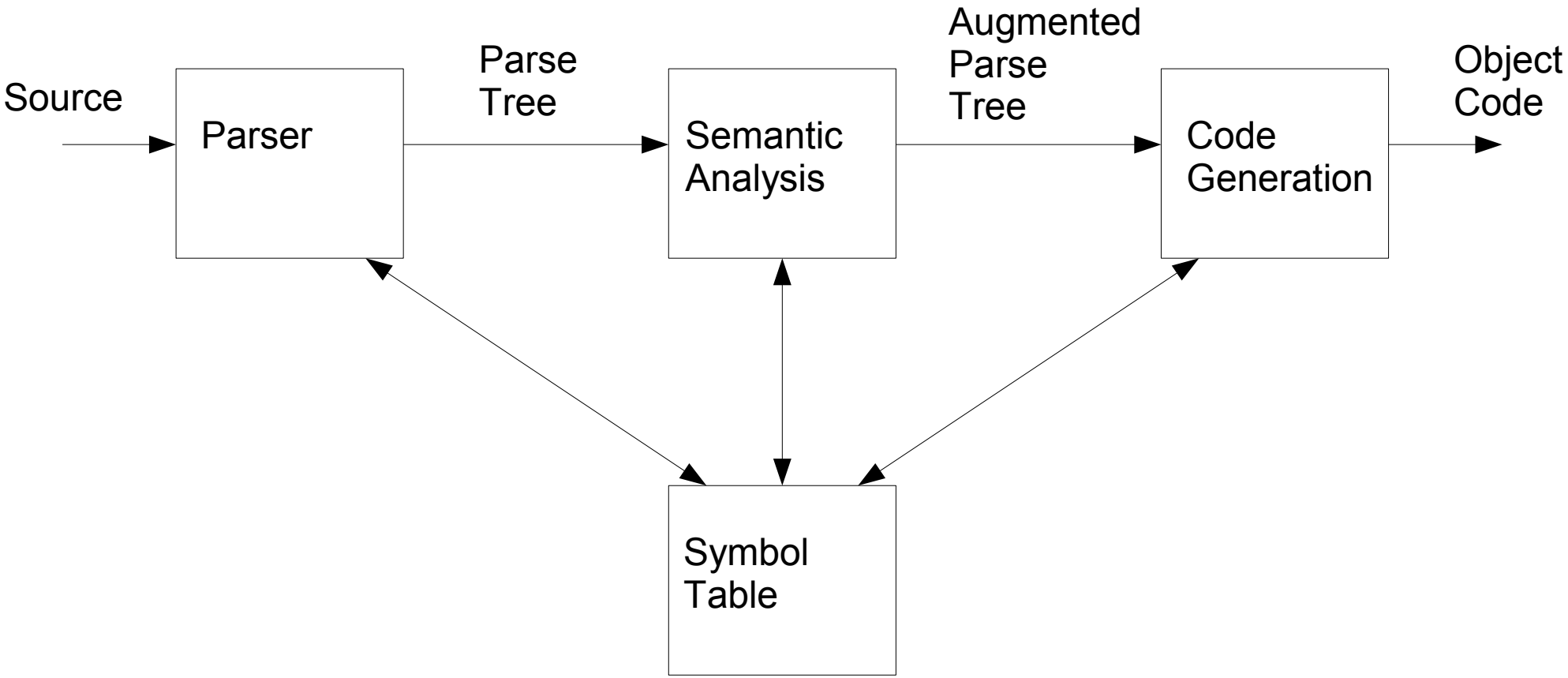
Outline

- Motivation: What is the problem?
- Overview of ASIS-for-GNAT
- An example application: ATC
- ASIS in action
- Conclusions

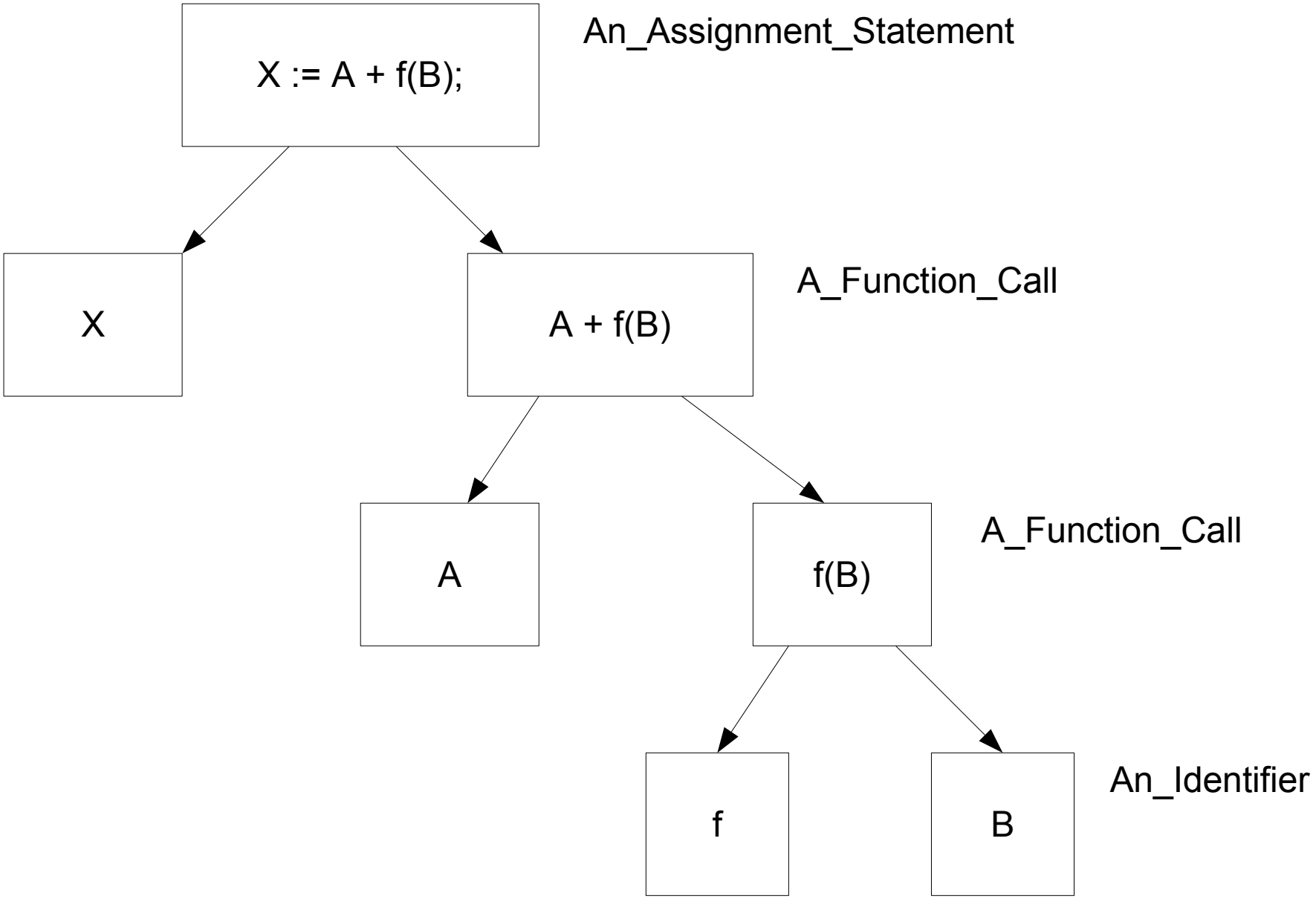
Motivation

- *Software analysis programs are <u>useful</u>.*
  - Style or documentation checkers
  - Specialized translators
  - Optimizers
  - Static analysis of correctness or security properties
- *Software analysis programs are <u>hard to write</u>.*
  - Lexical analysis and parsing
  - Symbol table management
  - Representing type information
  - Tracking modules and scopes

Compiler Overview



Abstract Syntax Tree



ASIS

- What is ASIS?
  - ASIS is an Ada library for analyzing Ada programs.
  - Compiler dumps parse tree and symbol table.
  - Analysis program uses ASIS library to access that information. No need to parse, etc, on its own.
- ASIS is standard.
  - ISO/IEC 15291:1999.
  - Analysis program's source code compatible across compilers.
  - Dumped files have compiler-specific format.

ASIS-for-GNAT

Compilation Command:

```
> gnatmake -gnatc -gnatt sample_1.adb
```

`-gnatc` : Parse and semantically analyze. No code generation.

`-gnatt` : Output “tree” files for use with ASIS.

Resulting Files (size given first):

```
    644 sample_1.adb (Source file)
336,923 sample_1.adt (Tree file)
  1,358 sample_1.ali (“Ada Library Interface” file)
```

ASIS library opens and uses the tree files

Ada Taint Checker (ATC)

- ATC is a program I wrote for a class at UVM*.
- ATC statically analyzes a program to see if “tainted” input data is ever output without first being sanitized.
- ATC uses ASIS. In addition it
  - Constructs control flow graphs
  - Does a data flow analysis
- Using ASIS made ATC much easier to write but ASIS doesn't do everything.

* The current version of ATC is “proof of concept” only. Many facilities are unimplemented.

Processing an ASIS Context

```
procedure Process_Context(...) is
   Units : Asis.Compilation_Unit_List :=
      Asis.Compilation_Units.Compilation_Units(The_Context);
   Next_Unit : Asis.Compilation_Unit;
   -- etc...
begin
   for J in Units'Range loop
      Next_Unit        := Units(J);
      Next_Unit_Origin :=
         Asis.Compilation_Units.Unit_Origin(Next_Unit);

      case Next_Unit_Origin is
         when Asis.An_Application_Unit =>
            Unit_Processing.Process_Unit(Next_Unit);

         when Asis.A_Predefined_Unit => ...
         when Asis.An_Implementation_Unit => ...
      end case;
   end loop;
end Process_Context;
```

Traversing the Abstract Syntax Tree

- ASIS represents the AST as a tree of `Asis.Element` objects.
- ASIS provides a generic recursive procedure that traverses the AST.
- You must instantiate that procedure using a processing procedure of your own.
- Your procedure gets to examine each element in parse tree order.

Processing a Unit

```
procedure Process_Unit (The_Unit : Asis.Compilation_Unit) is
   Unit_Decl : Asis.Element := Asis.Unit_Declaration(The_Unit);
begin
   New_Line;
   Put_Line("BUILDING CONTROL FLOW GRAPH");
   Process_Construct(Unit_Decl);
   Dump_CFG;

   New_Line;
   Put_Line("ITERATING DATA FLOW EQUATIONS");
   Compute_Dataflow;
end Process_Unit;
```

Processing Statement Elements

```
-- Various ASIS packages "withed"

procedure Pre_Operation(E : Element; ...) is
  -- etc...
begin
   case Element_Kind(E) is
      when A_Statement =>
         case Statement_Kind(E) is
            when An_Assignment_Statement => ...
            when An_If_Statement => ...
            when A_While_Loop_Statement => ...
            when A_For_Loop_Statement => ...
            when A_Procedure_Call_Statement => ...
            when others =>
               Unsupported_Feature("Unsupported AST Element");
         end case;

      -- This is where declarations are handled.
      when others => null;
   end case;
end Pre_Operation;
```

Ada Expressions

- Statements contain expressions
  - `X := A + f(B);`
- Most expressions are function calls.
  - `function"+"(L : Integer; R : Integer);`
- Other expression kinds include selection (array elements, record members), attribute expressions, etc, etc.

Processing Expression Elements

```
function Is_Tainted(E : Asis.Element; ...) return Boolean is
   -- etc...
begin
   case Expression_Kind(E) is
      -- Literals are not tainted.
      when An_Integer_Literal => return False;
      when An_Identifier =>
         Name : Program_Text := Name_Image(E);
         -- etc...

      -- Unwind one layer of parentheses using recursion.
      when A_Parenthesized_Expression =>
         return Is_Tainted(Expression_Parenthesized(E));

      -- Most expressions are function calls (eg built-in operators).
      when A_Function_Call =>
         return Handle_Function_Call;

      when others =>
         Internal_Error("Unexpected expression kind encountered");
   end case;
   return True;
end Is_Tainted;
```

Semantic Information

- ASIS does more than make the AST available.
  - Given an identifier, ASIS can find the corresponding declaration.
  - ASIS can “untangle” named parameter associations in subprogram calls.
    - `My_Procedure(P2 => Arg2, P1 => Arg1);`
  - ASIS can locate argument modes from procedure declarations.
    - `procedure My_Proc(P1 : in out Integer);`
  - etc.

Processing Subprogram Arguments

```
function Has_Tainted_Input return Boolean is
   Actual_Expression : Expression;
   Mode              : Mode_Kinds;
begin
   for I in Procedure_Arguments'Range loop
      Actual_Expression :=
         Actual_Parameter(Procedure_Arguments(I));
      Mode := Mode_Kind(Procedure_Formals(I));
      if Mode = An_In_Mode or Mode = An_In_Out_Mode then
         if Is_Tainted(Actual_Expression) then
            return True;
         end if;
      end if;
   end loop;
   return False;
end Has_Tainted_Input;
```

Conclusion

- Writing ASIS programs isn't trivial.
  - ASIS requires a fair amount of administrative overhead.
  - Ada is a rich language: Full support for all Ada features requires significant work.
  - ASIS doesn't do everything.
- BUT... it's a lot easier than having to build a full scale parser and symbol table manager.
- Ada's overall clean design helps make ASIS possible. Too bad there's no ASIS for C++.