The Inner Workings of Overture-core

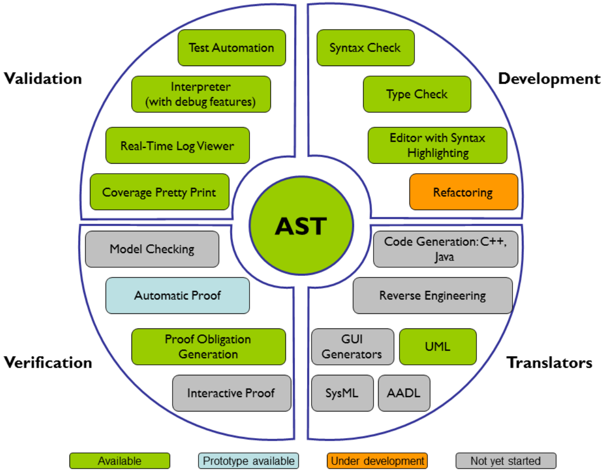
Kenneth Lausdahl

Department of Engineering, Aarhus University, Denmark

28 August 2012 / 10th International Workshop Overture/VDM

Outline

Features


Validation
Test Automation
Interpreter (with debug features)
Real-Time Log Viewer
Coverage Pretty Print
Development
Syntax Check
Type Check
Editor with Syntax Highlighting
Refactoring
AST
Model Checking
Automatic Proof
Proof Obligation Generation
Interactive Proof
Verification
Code Generation: C++, Java
Reverse Engineering
GUI Generators
UML
SysML
AADL
Translators
Available
Prototype available
Under development
Not yet started

Overture Core

- AST
- Parser
- Type Checker
- Interpreter
- Proof Obligation Generator

Overture Core

- AST
- Parser
- Type Checker
- Interpreter
- Proof Obligation Generator

Visitor based

Visitor Types

1. Standard visitor
2. Question visitor
    - Parse an argument
3. Answer visitor
    - Return a result
4. Question-Answer visitor
    - Parse an argument
    - Return a result

Visitor Types

1. Standard visitor
2. Question visitor
    - Parse an argument
3. Answer visitor
    - Return a result
4. Question-Answer visitor
    - Parse an argument
    - Return a result

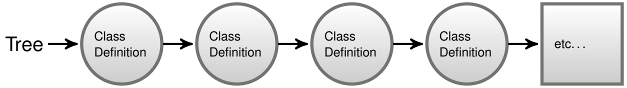
Question-Answer: is used by the Type Checker and Interpreter

Outline

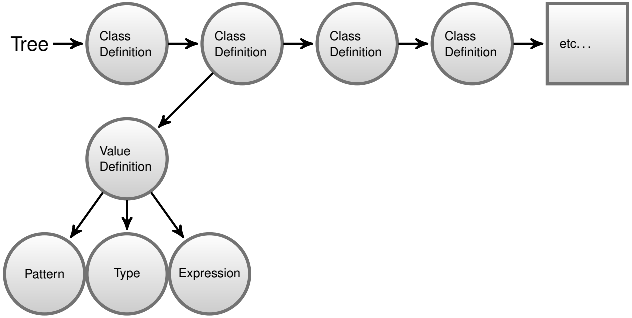
Type Check

1. Make sure there are no duplicate class definitions.
2. Create a public class environment with all classes
3. For each class:
    1. Generate implicit definitions (class and definition). Including:
        - Class type hierarchy
        - Implicit local names (access to inherited definitions)
    2. Create private class env and resolve types
    3. Check overloading and overriding definitions.
    4. In the order [types, values, definitions]:
        - type check with a private env
    5. Produce “unused” warnings for unused definitions.

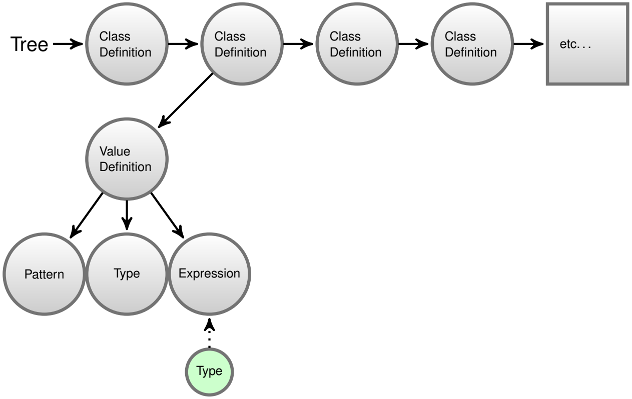
Type Checker Overview



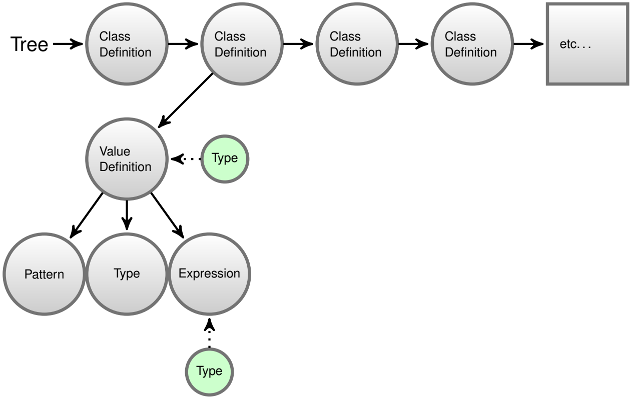
Type Checker Overview



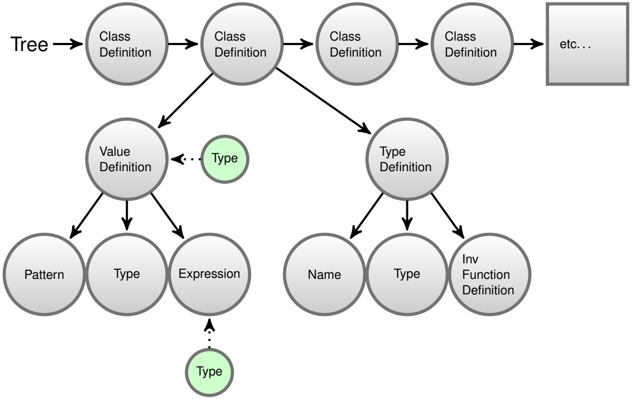
Type Checker Overview



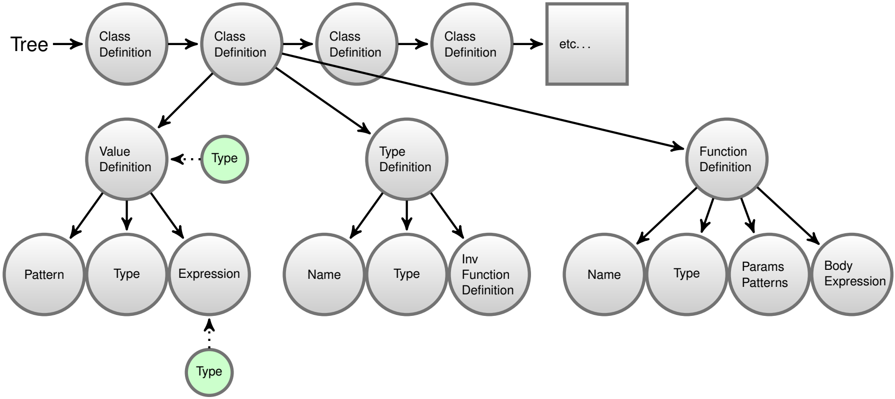
Type Checker Overview



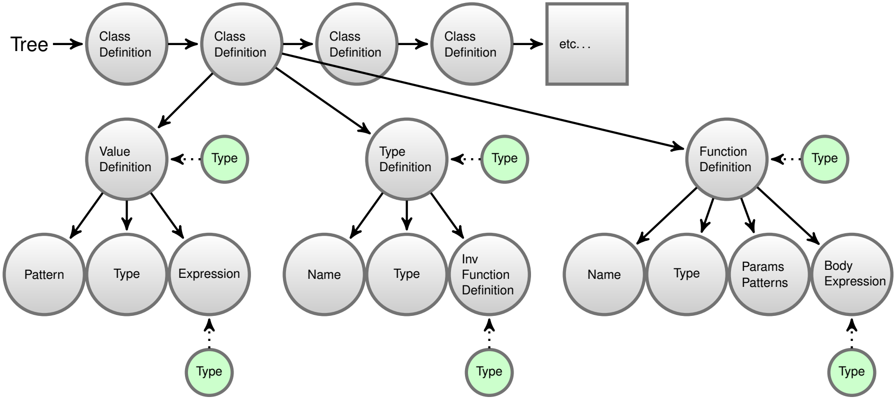
Type Checker Overview



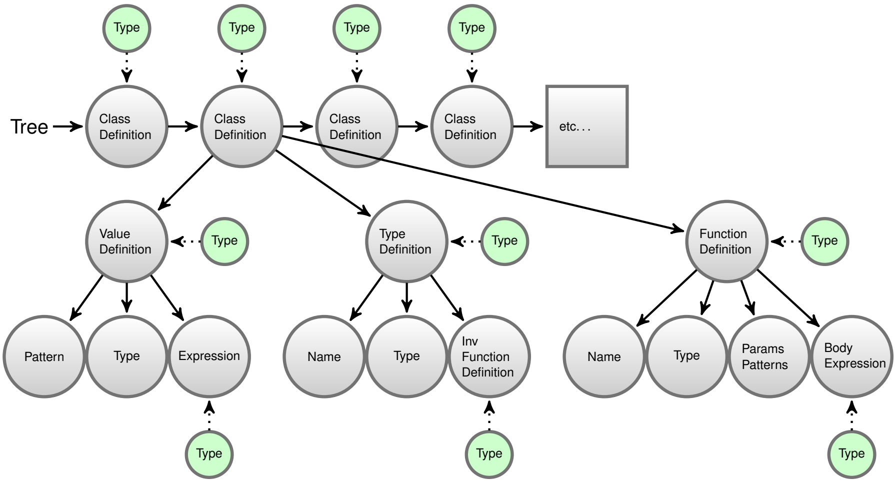
Type Checker Overview



Type Checker Overview



Type Checker Overview



Type Checker Visitor
Expressions

```
public PType defaultSBooleanBinaryExp(SBooleanBinaryExp node,
TypeCheckInfo question) throws AnalysisException{

   node.getLeft().apply(this, question);
   node.getRight().apply(this, question);

   if (!isType(node.getLeft().getType(),expected.getClass()))
   {
     error(3065, "Left hand of" + node.getOp() + "is not"+expected);
   }

   if (!isType(node.getRight().getType(),expected.getClass()))
   {
     error(3066, "Right hand of" + node.getOp() + "is not"+expected);
   }

   node.setType(expected);

   return node.getType();
}
```

Type Checker Visitor

Expressions

```
public PType defaultSBooleanBinaryExp(SBooleanBinaryExp node,
TypeCheckInfo question) throws AnalysisException{

   node.getLeft().apply(this, question);
   no

   if                                                           ()))
   {
                                                                +expected);
   }
```

```
TypeCheckerInfo:
                    Environment env;
                    NameScope scope;
                    List<PType> qualifiers;
```

```
   if (!isType(node.getRight().getType(),expected.getClass()))
   {
     error(3066, "Right hand of" + node.getOp() + "is not"+expected);
   }

   node.setType(expected);

   return node.getType();
}
```

Type Checker Visitor
Expressions

```
public PType defaultSBooleanBinaryExp(SBooleanBinaryExp node,
TypeCheckInfo question) throws AnalysisExc

  node.getLeft().apply(this, question);
  node.getRight().apply(this, question);

  if (!isType(node.getLeft().getType(),expected.getClass()))
  {
    error(3065, "Left hand of" + node.getOp() + "is not"+expected);
  }

  if (!isType(node.getRight().getType(),expected.getClass()))
  {
    error(3066, "Right hand of" + node.getOp() + "is not"+expected);
  }

  node.setType(expected);

  return node.getType();
}
```

Type Check expression fields

Type Checker Visitor
Expressions

```
public PType defaultSBooleanBinaryExp(SBooleanBinaryExp node,
TypeCheckInfo question) throws AnalysisException{

   node.getLeft().apply(this, question);
   node.getRight().apply(this, question);

   if (!isType(node.getLeft().getType(),expected.getClass()))
   {
     error(3065, "Left hand of" + node.get
   }

   if (!isType(node.getRight().getType(),e
   {
     error(3066, "Right hand of" + node.getOp() + "is not"+expected);
   }

   node.setType(expected);

   return node.getType();
}
```

Type Check this node

Type Checker Visitor
Expressions

```
public PType defaultSBooleanBinaryExp(SBooleanBinaryExp node,
TypeCheckInfo question) throws AnalysisException{

   node.getLeft().apply(this, question);
   node.getRight().apply(this, question);

   if (!isType(node.getLeft().getType(),expected.getClass()))
   {
     error(3065, "Left hand of" + node.getOp() + "is not"+expected);
   }

   if (!isType(node.getRight().getType(),expected.getClass()))
   {
     error(3066, "Right hand of" + node.getOp() + "is not"+expected);
   }

   node.setType(expected);

   return node.getType();
}
```

Set Type and return it

Outline

Interpreter

- Initial setup
  1. Setup model: Parse, Type Check
  2. Parse initial expression and type check it
- Interpretation
  1. Create Main Context
  2. - Add initial context
     - Add values
  3. Create a main thread with the initial expression
     - Start main
  4. Start the scheduler

Interpreter Visitor
Expressions

```
public Value caseAAndBooleanBinaryExp(AAndBooleanBinaryExp node,
Context ctxt) throws AnalysisException{

   node.getLocation().hit(); // Mark as covered
   Value lv = node.getLeft().apply(this, ctxt);

   if (lv.isUndefined())
     return lv;

   boolean lb = lv.boolValue(ctxt);

   if (!lb)
     return lv; // Stop after LHS

   Value rv = node.getRight().apply(this, ctxt);

   if (lb)
     return rv;

   return new BooleanValue(false);
}
```

Interpreter Visitor
Expressions

```
public Value caseAAndBooleanBinaryExp(AAndBooleanBinaryExp node,
Context ctxt) throws AnalysisException{

   node.getLocation().hit(); // Mark as covered
   Value lv = node.getLeft().apply(this, ctxt);
```

```
   Context:
               Used for lookup. Consists of nested
               contexts to control scopes
```

```
   if (!lb)
     return lv; // Stop after LHS

   Value rv = node.getRight().apply(this, ctxt);

   if (lb)
     return rv;

   return new BooleanValue(false);
}
```

Interpreter Visitor
Expressions

Record coverage

```
public Value caseAAndBooleanBinaryExp(AAndB
Context ctxt) throws AnalysisException{

   node.getLocation().hit(); // Mark as covered
   Value lv = node.getLeft().apply(this, ctxt);

   if (lv.isUndefined())
     return lv;

   boolean lb = lv.boolValue(ctxt);

   if (!lb)
     return lv; // Stop after LHS

   Value rv = node.getRight().apply(this, ctxt);

   if (lb)
     return rv;

   return new BooleanValue(false);
}
```

Interpreter Visitor
Expressions

```
public Value caseAAndBooleanBinaryExp(AAndBooleanBinaryExp node,
Context ctxt) throws AnalysisException{

   node.getLocation().hit(); // Mark as covered
   Value lv = node.getLeft().apply(this, ctxt);

   if (lv.isUndefined())
     return lv;

   boolean lb = lv.boolValue(ctxt);

   if (!lb)
     return lv; // Stop after LHS

   Value rv = node.getRight().apply(this, ctxt);

   if (lb)
     return rv;

   return new BooleanValue(false);
}
```

Evaluate left and right

Thanks

Wiki Overture Workshop 10