TECHNISCHE UNIVERSITÄT DRESDEN

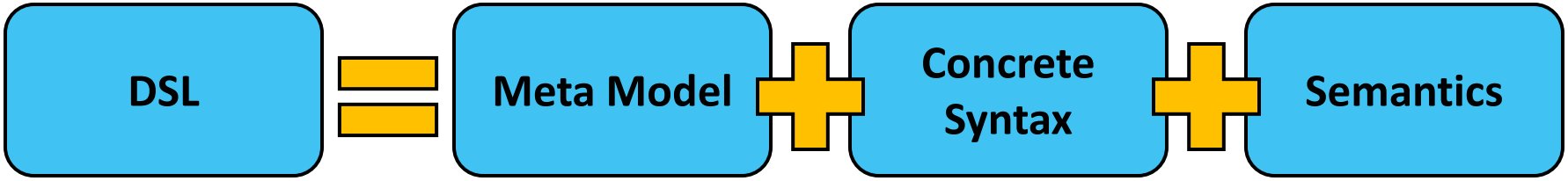
Fakultät Informatik Institut Software- und Multimediatechnik, Lehrstuhl Softwaretechnologie

JastEMF: Reference Attribute Grammars for EMF-based DSLs

Sven Karol, Christoff Bürger

ACSE 17.12.2012

What's in a DSL?



Domain concepts, relations, attributes

Symbols, shapes, text

Code analysis, behavior, meaning, intepretation

Benefits of Metamodelling

Metamodelling is a standardisation process with the following benefits:

- MM 1 Metamodelling Abstraction
- MM 2 Metamodelling Consistency
- MM 3 Metamodel Implementation Generators
- MM 4 Metamodel/Model Compatibility
- MM 5 Tooling Compatibility

However, metamodelling leaks convenient mechanisms for semantics specification.

Benefits of Attribute Grammars (AGs)

AGs are very convenient to specify semantics for tree structure with the following benefits:

- AG 1: Declarative Semantics Abstraction
- AG 2: Semantics Consistency
- AG 3: Semantics Generators
- AG 4: Semantics Modularity

Claim: A combination of MM and AGs enables *semantics integrated metamodelling* and leads to more successful and reliable tool implementations.

Attribute Grammars

Formalism to compute static semantics over syntax trees [Knuth68]

- Basis: context-free grammars + attributes + semantic functions
- Evaluation by tree visitors with different visiting strategies
  - Static: ordered attribute grammars (OAGs)
  - Dynamic: demand-driven evaluation
- AGs are modular and extensible

Improvements

- Higher order attribute grammars (HOAGs) [Vogt+89]
- Reference attributed grammars (RAGs) [Hedin00,Boyland05]

Formal Definition

(Short) Definition (attribute grammar): An attribute grammar (AG) is an 8-tuple $G=(\mathbf{G_0},\mathbf{Syn},\mathbf{Inh},\mathbf{Syn_X},\mathbf{Inh_X},\mathbf{K},\mathbf{\Omega},\mathbf{\Phi})$ with the following components

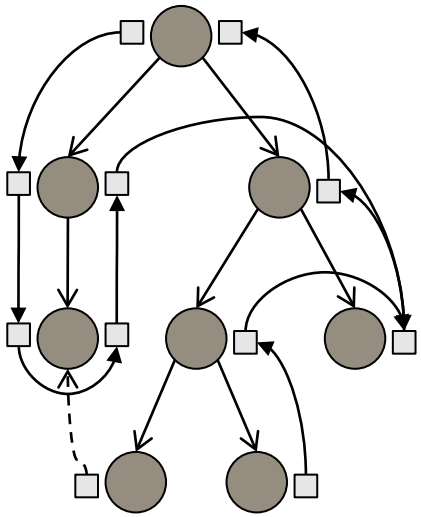
- $\mathbf{G_0} = (N,\Sigma,P,S)$ a CFG,
- **Syn** and **Inh** the finite, disjoint sets of synthesized and inherited attributes,
- $\mathbf{Syn_X} : N \rightarrow P(Syn)$ a function that assigns a set of synthesized attributes to each nonterminal in G_0,
- $\mathbf{Inh_X} : N \rightarrow P(Inh)$ a function that assigns a set of inherited attributes to each nonterminal in G_0,
- **K** a set of attribute types/sorts,
- $\mathbf{\Omega} : Inh \cup Syn \rightarrow K$ a function assigning each attribute a $\kappa \in K$,
- **Φ** a set of semantic functions $\varphi_{(p,i,a)}$ with $p \in P$, $i \in \{0,\ldots,n_p\}$, $a \in Syn_X(p_i) \cup Inh_X(p_i)$.

Formal Definition

Kind of AG depends on K and Φ

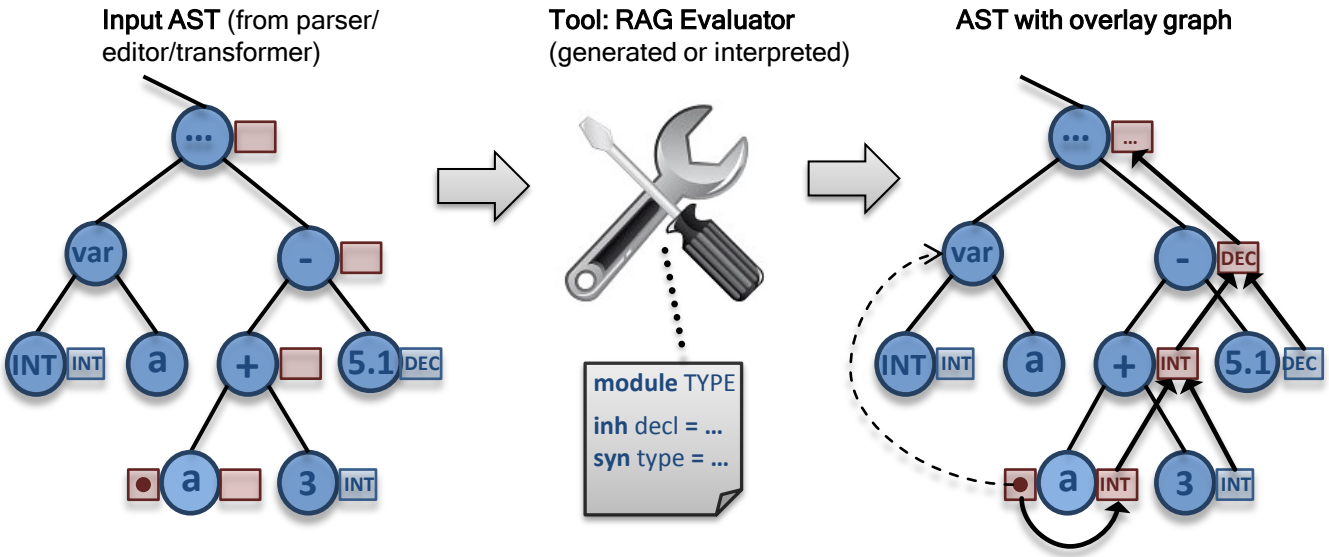
- Standard AG
  - $K \cap \mathbf{N} = \{\}$
- Higher Order AG (HOAG)
  - $K \cap \mathbf{N} \neq \{\}$
  - all $\varphi \in \Phi$ only yield copied or fresh values
- Reference Attribute Grammar (RAG)
  - $K \cap \mathbf{N} \neq \{\}$
  - $\varphi \in \Phi$ yield references (pointers) to nonterminals (or value copies)

Kinds of Attributes



- **Inherited attributes** (inh): Top-down value dataflow/computation
- **Synthesized attributes** (syn): Bottom-up value dataflow/computation
- **Collection attributes** (coll): Collect values freely distributed over the AST
- **Reference attributes**: Compute references to exisiting nodes in the AST

Basic Working Principle of RAG Tools



How do (R)AGs Relate with Metamodeling?

Models are graphs!?

- Ecore (EMOF) metamodels are commonly build around a **tree**-based abstract syntax used by
  - **Tree** iterators, **tree** editors, transformation tools, textual editors
- Syntax mappers like *EMFText* use the **tree** structure to compute all other information (e.g., resolving cross references)
- Graphical editors use the **tree** structure to manage user created object hierarchies, cross references and values therein and to compute read-only information (e.g., cross references, derived values)

➔ Ecore (EMOF) models are ASTs with cross-references (overlay graphs) and derived information!

Mapping Between Both Worlds [Bürger+11]

Syntax in Ecore	Syntax in RAGs
EClass	AST Node Type
EReference[containment]	Non Terminal
EAttribute[non-derived]	Terminal

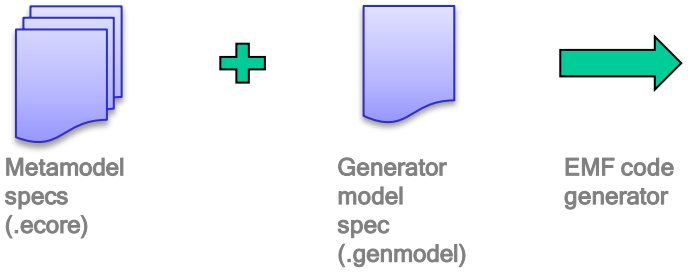
E_{Syn}

Semantics Interface in Ecore	Semantics in RAGs
EAttribute[derived]	[synthesized|inherited] attribute
EAttribute[derived,multiple]	collection attribute
EReference[non-containment]	collection attribute, reference attribute
EOperation[side-effect free]	[synthesized|inherited] attribute

E_{Sem}

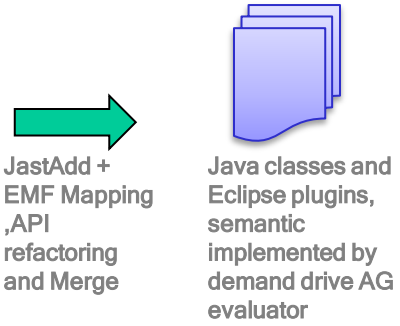
The JastEMF Tool

EMF basic generation process



JastAdd basic generation process

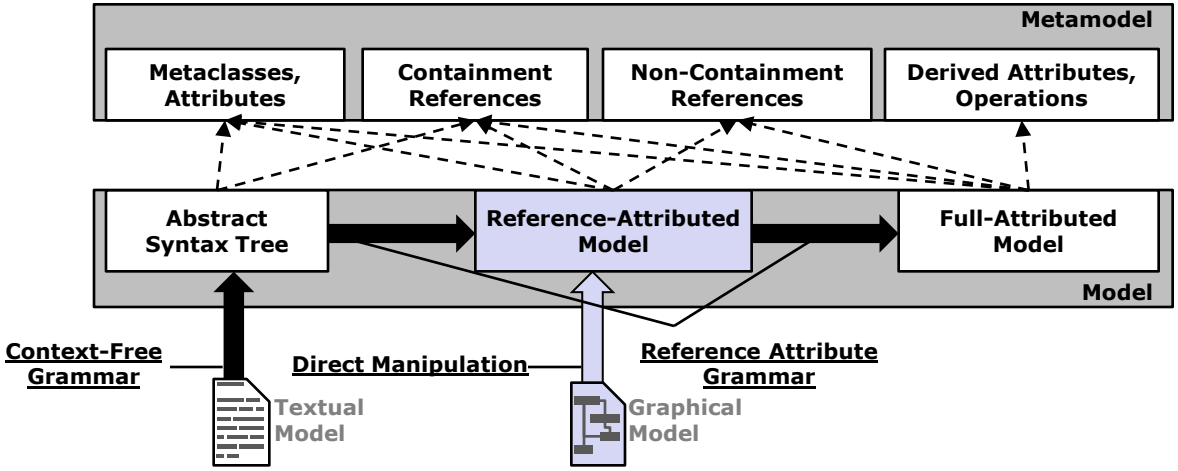
JastEMF meta integration



The JastEMF Tool

Semantic evaluation can start from (partly) reference-attributed models

- Non-containment references can have predefined values (e.g., specified by users using a diagram editor)

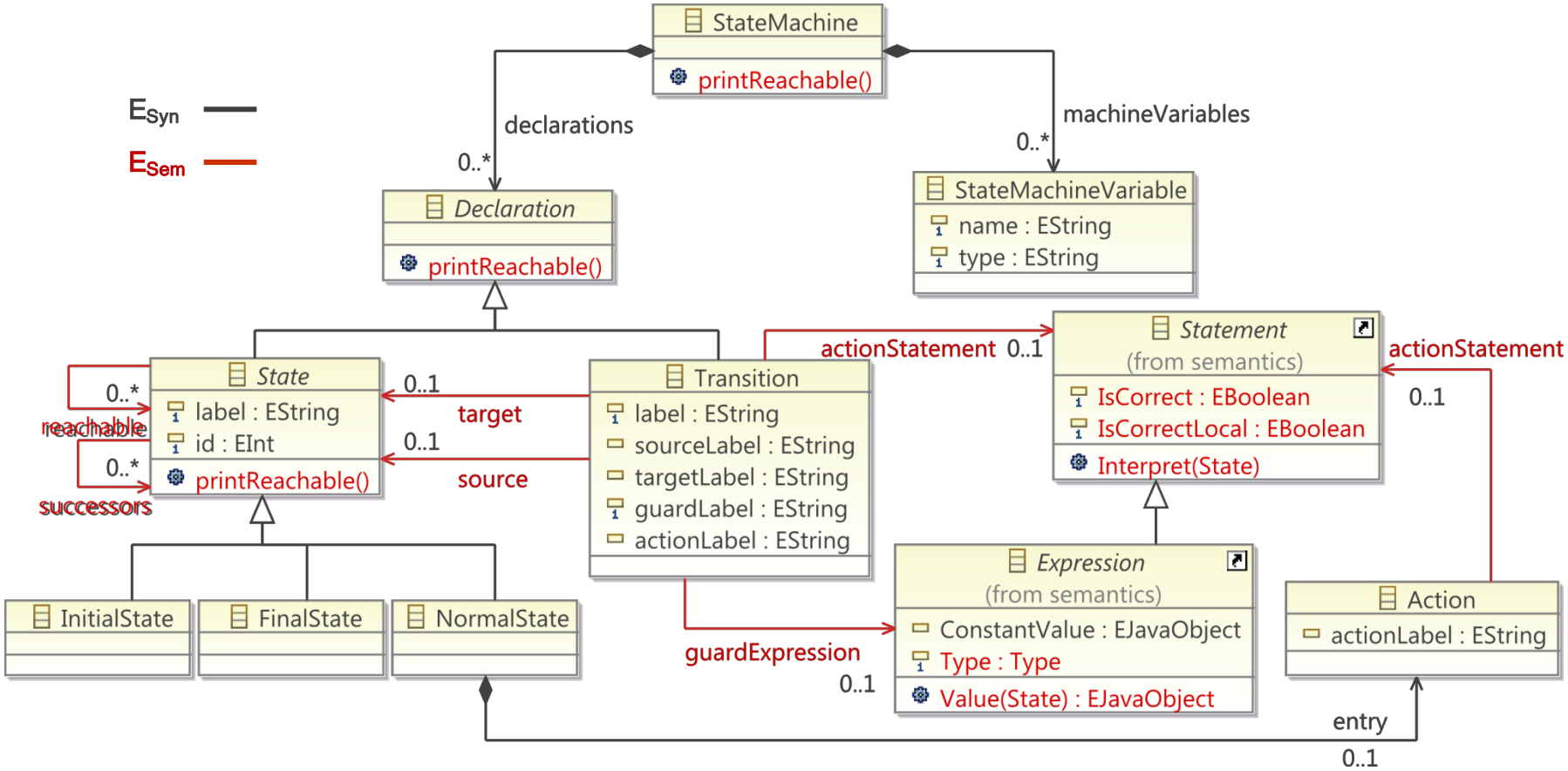


- If a value is given: Use it instead of attribute equation

Examples

1. Graphical DSL [Bürger+11]
   - Statecharts
   - Syntax in GMF
2. Textual DSL [Heidenreich+12]
   - Forms DSL (see ACSE tutorial of 04.12.12)
   - Syntax in EMFText
3. Imperative Programming Language [Bürger+11]
   - SiPLE (**Si**mple **P**rogramming **L**anguage **E**xample)
   - Syntax in xText

Example 1: Statechart Metamodel

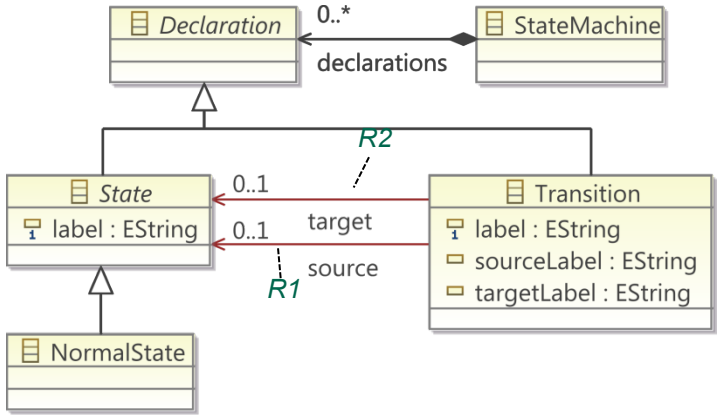


(Ecore-based, extended version of Statemachine example in Hedin, G.: Generating Language Tools with JastAdd. In: GTTSE '09. LNCS,Springer (2010))

Example 1: Statechart Attributes

AST specification (partial):

```
abstract State:Declaration ::= <label:String>;
NormalState:State;
Transition:Declaration ::=<label:String>
   <sourceLabel:String><targetLabel:String>;
```



Attribution example:

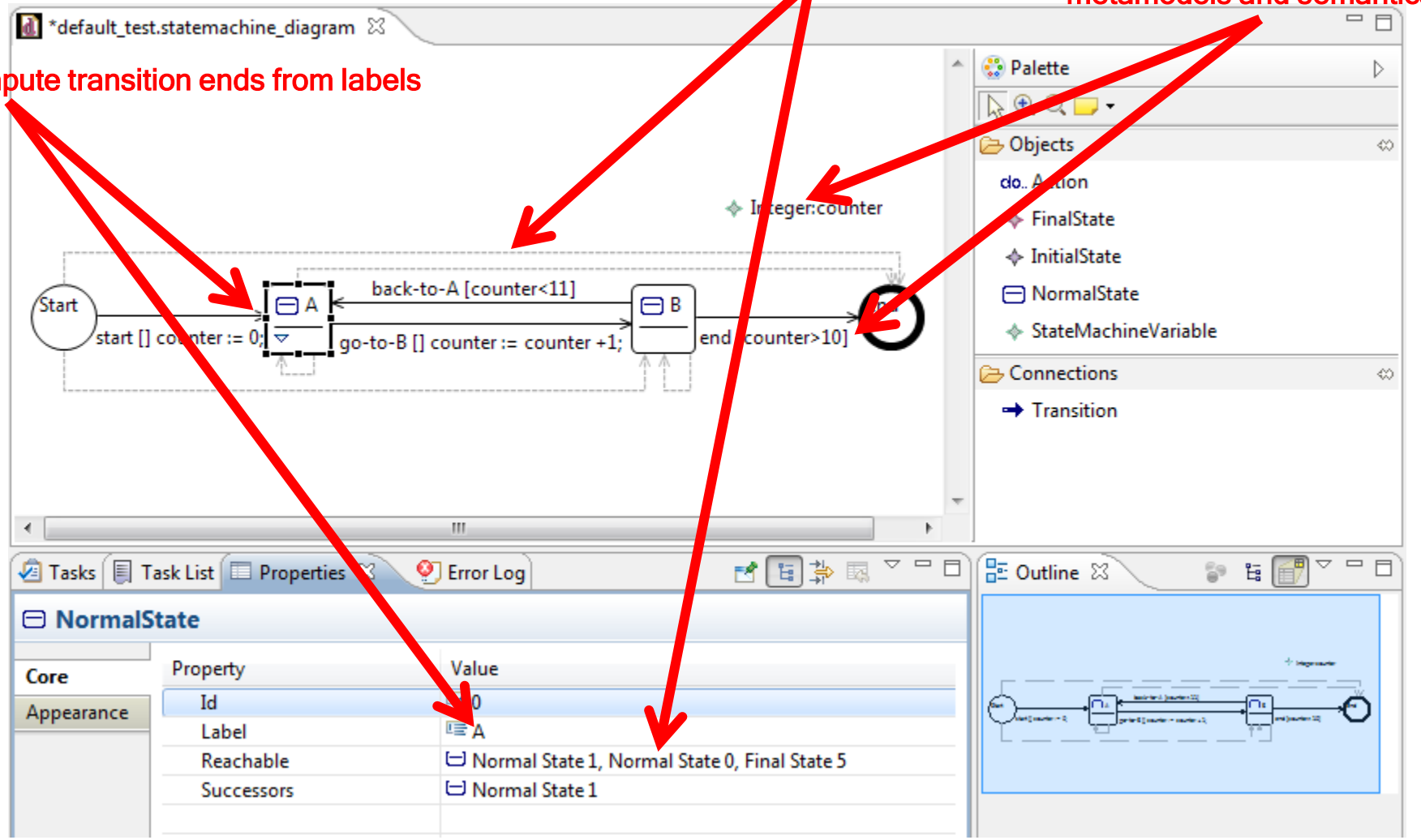
```
syn lazy State Transition.source() = lookup(getSourceLabel()); // R1
syn lazy State Transition.target() = lookup(getTargetLabel()); // R2
inh State Declaration.lookup(String label); // R3
eq StateMachine.getDeclarations(int i).lookup(String label) { … } // R4
syn State Declaration.localLookup(String label) =
      (label==getLabel()) ? this : null; // R5
```

(Ecore-based, extended version of Statemachine example in Hedin, G.: Generating Language Tools with JastAdd. In: GTTSE '09. LNCS,Springer (2010))

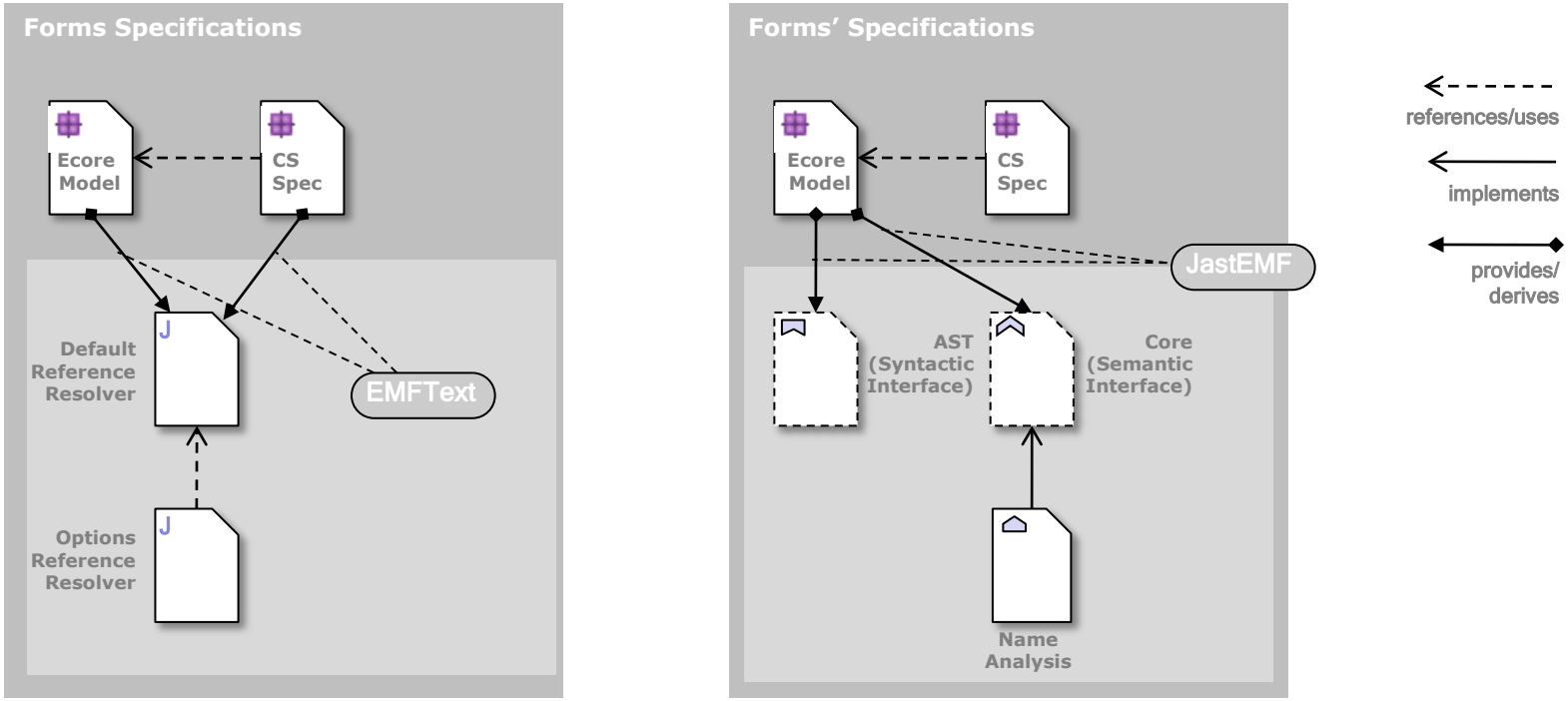
compute closure

reuse of metamodels and semantics

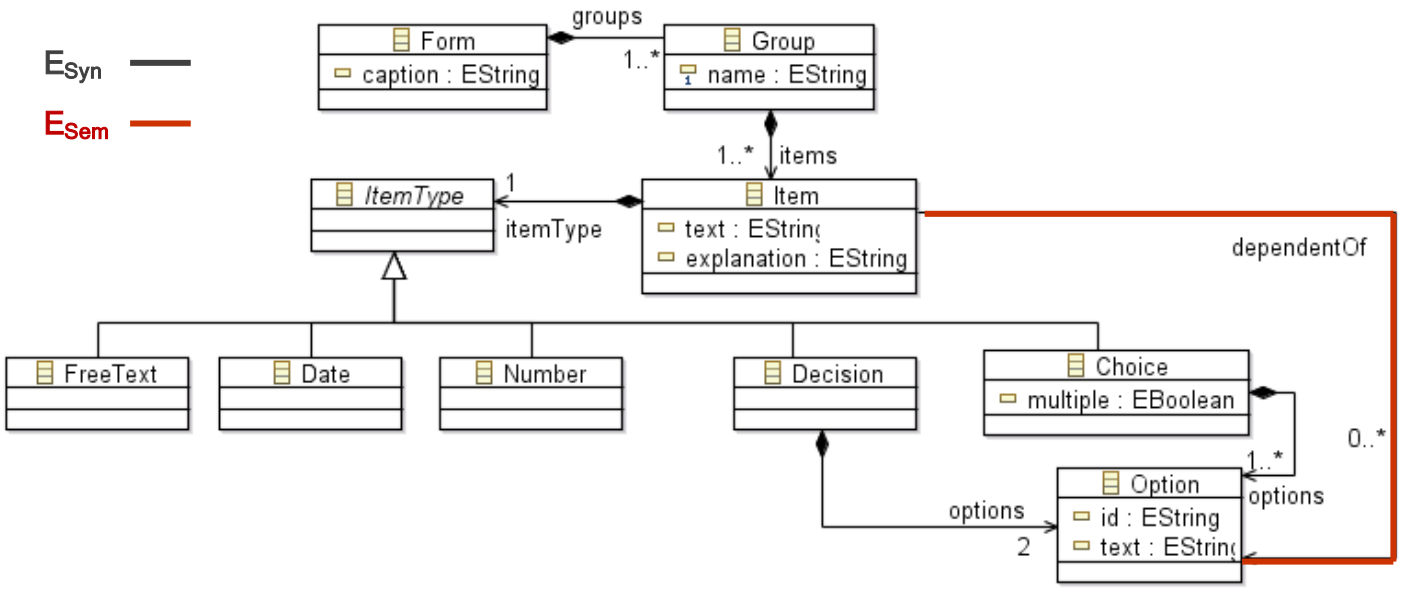
compute transition ends from labels



Example 2: Forms DSL



Example 2: Forms Metamodel



Example 2: Forms as AST Grammar

```
Form ::= <caption:String> groups:Group*;

Item ::= <text:String> <explanation:String> <dependentOfName:String>
         itemType:ItemType;

abstract ItemType;
FreeText:ItemType;
Choice:ItemType ::= <multiple:boolean> options:Option*;
Date:ItemType;
Number:ItemType;
Decision:ItemType ::= options:Option*;

Option ::= <id:String> <text:String>;

Group ::= <name:String> items:Item*;
```

Terminal - EAttribute

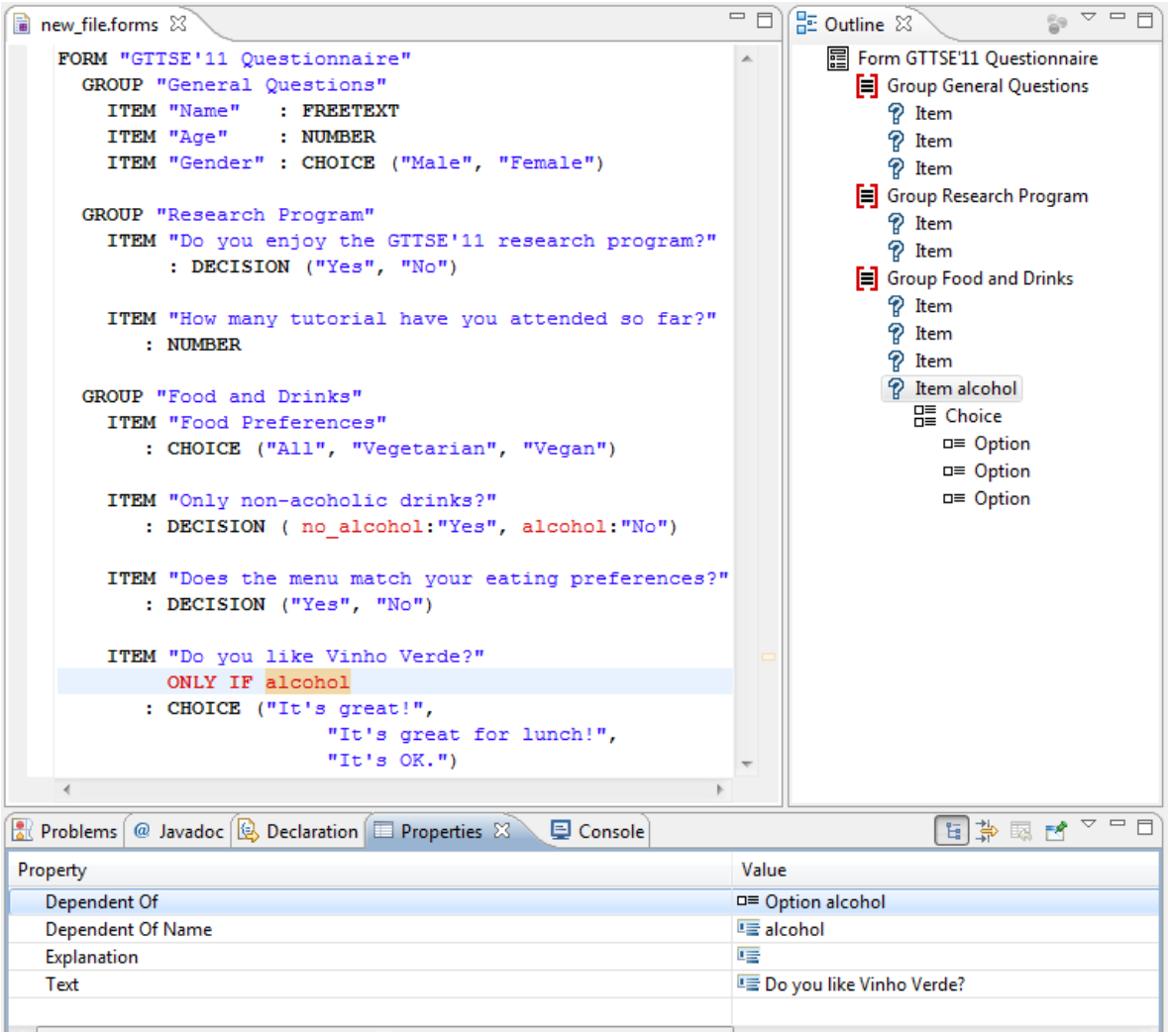
ASTNode - EClass

Nonterminal - EReference (Containment)

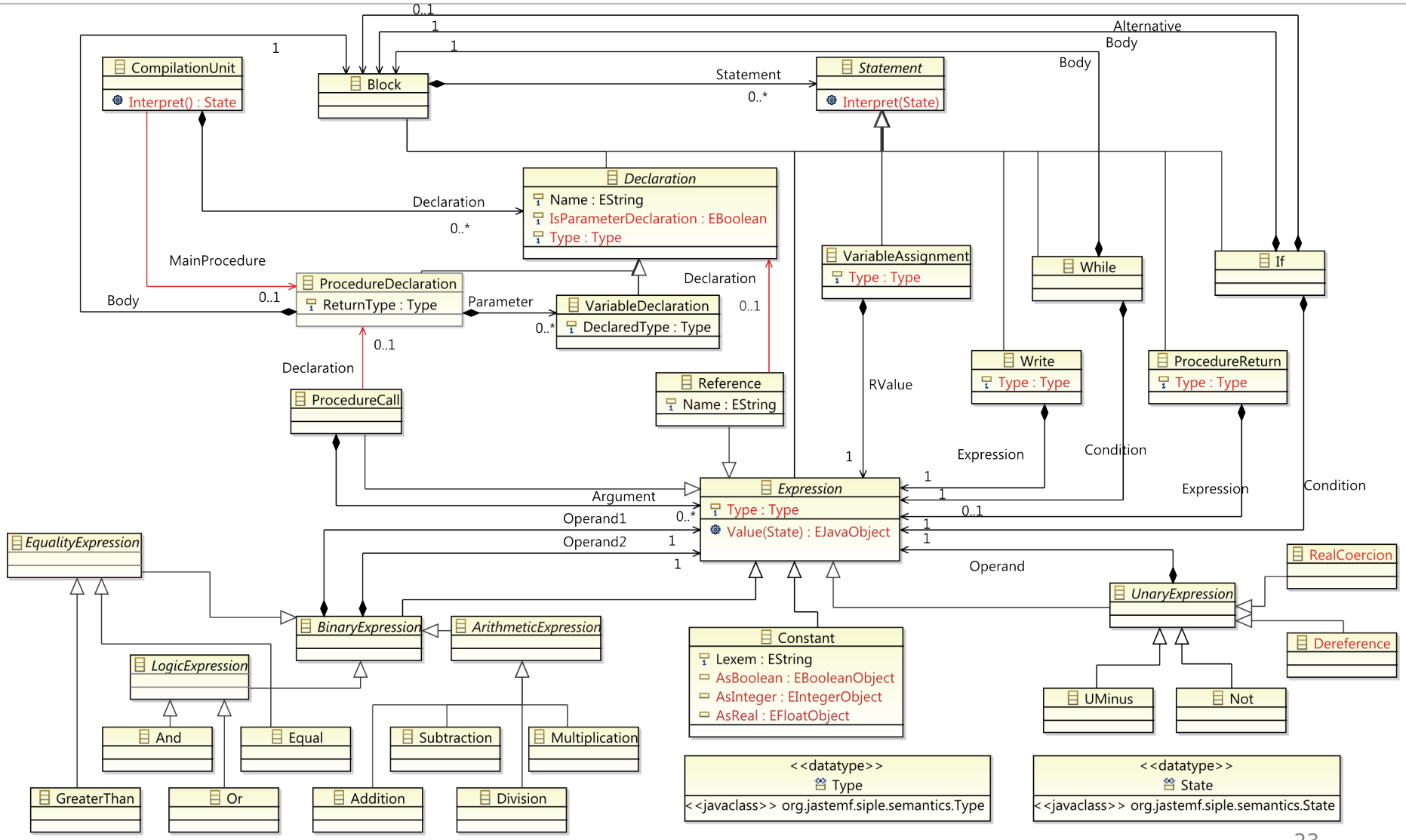
Inheritance

Example 2: Froms Attributes

```
aspect NameAnalysis {
    inh Form ASTNode.form();
    syn EList Item.dependentOf();
    inh EList ASTNode.LookUpOption(String optionName);
    coll EList<Option> Form.Options() [new BasicEList()] with add;

    Option contributes this to Form.Options() for form();

    eq Form.getgroups(int index).form() = this;
    eq Item.dependentOf() = LookUpOption(getdependentOfName());

    eq Form.getgroups(int index).LookUpOption(String optionName){
           EList result = new BasicEList();
           for(Option option:Options()){
                    if(optionName.equals(option.getid()))
                             result.add(option);
           }
           return result;
    }
}
```


new_file.forms
FORM "GTTSE'11 Questionnaire"
GROUP "General Questions"
ITEM "Name" : FREETEXT
ITEM "Age" : NUMBER
ITEM "Gender" : CHOICE ("Male", "Female")
GROUP "Research Program"
ITEM "Do you enjoy the GTTSE'11 research program?"
: DECISION ("Yes", "No")
ITEM "How many tutorial have you attended so far?"
: NUMBER
GROUP "Food and Drinks"
ITEM "Food Preferences"
: CHOICE ("All", "Vegetarian", "Vegan")
ITEM "Only non-acoholic drinks?"
: DECISION (no_alcohol:"Yes", alcohol:"No")
ITEM "Does the menu match your eating preferences?"
: DECISION ("Yes", "No")
ITEM "Do you like Vinho Verde?"
ONLY IF alcohol
: CHOICE ("It's great!",
"It's great for lunch!",
"It's OK.")
Outline
Form GTTSE'11 Questionnaire
Group General Questions
Item
Item
Item
Group Research Program
Item
Item
Group Food and Drinks
Item
Item
Item
Item alcohol
Choice
Option
Option
Option
Problems
Javadoc
Declaration
Properties
Console
Property
Value
Dependent Of
Option alcohol
Dependent Of Name
alcohol
Explanation
Text
Do you like Vinho Verde?

Example 3: SiPLE Metamodel

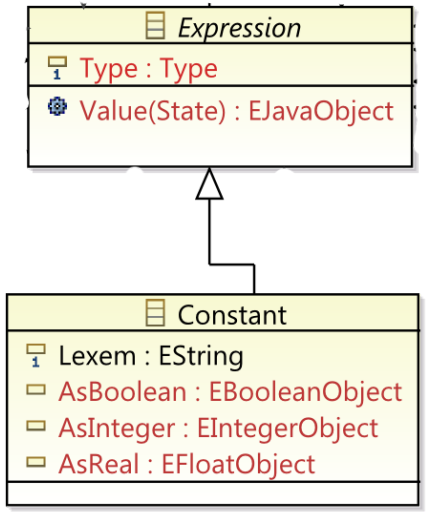


Example 3: SiPLE Types (Excerpt from Semantic Interface)

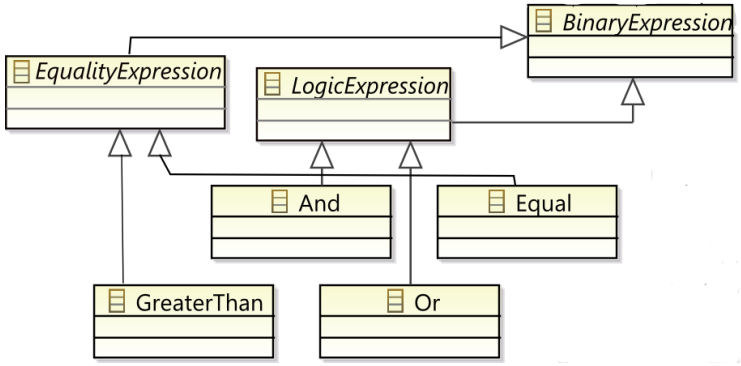
```
aspect TypeAnalysis {
    syn Type Declaration.Type();
    syn Type VariableAssignment.Type();
    syn Type ProcedureReturn.Type();
    syn Type Write.Type();
    syn Type Read.Type();
    syn Type Expression.Type();
}

aspect NameAnalysis {
    // Ordinary name space:
    inh LinkedList<Declaration> ASTNode.LookUp(String name);
    syn ProcedureDeclaration CompilationUnit.MainProcedure();
    syn Declaration Reference.Declaration();
}
```

Example 3: SiPLE Types (Excerpt from Definitions)

```
/** Expressions' Type **/

eq Constant.Type() {
    if (AsBoolean() != null)
          return Type.Boolean;
    if (AsReal() != null)
          return Type.Real;
    if (AsInteger() != null)
          return Type.Integer;
    return Type.ERROR_TYPE;
}
```



Example 3: SiPLE Types (Excerpt from Definitions)

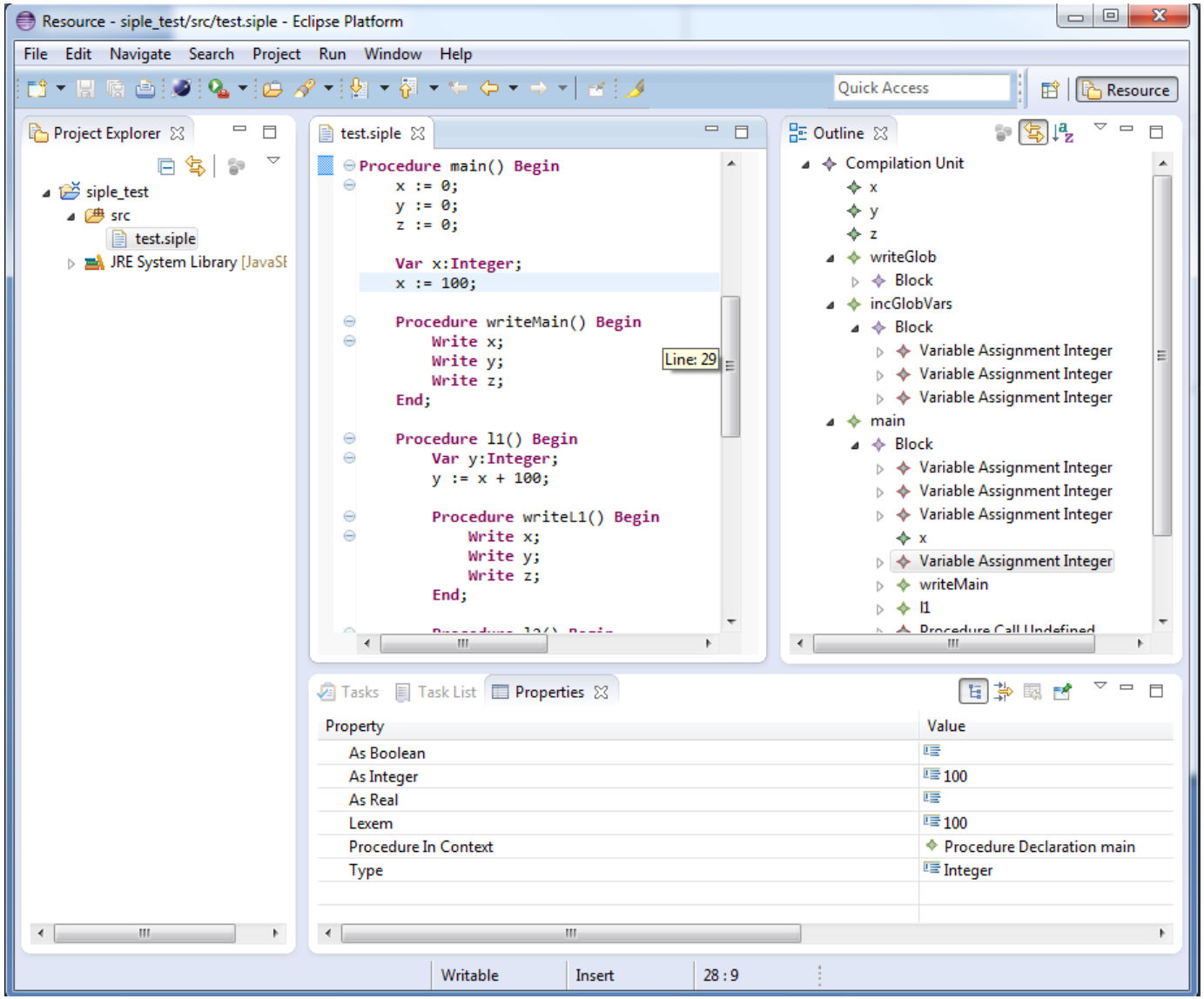



```
eq LogicExpression.Type() =
     getOperand1().Type().isBoolean() &&
     getOperand2().Type().isBoolean() ?
            Type.Boolean :Type.ERROR_TYPE;
```

```
eq EqualityExpression.Type() =
     Type.bequals(getOperand1().Type(), getOperand2().Type()) &&
            (getOperand1().Type().isReal() ||
            getOperand1().Type().isInteger()) ?
                        Type.Boolean : Type.ERROR_TYPE;
```

```
eq Equal.Type() =
     Type.bequals(getOperand1().Type(), getOperand2().Type()) ?
            Type.Boolean : Type.ERROR_TYPE;
```

```
eq ArithmeticExpression.Type() =
     Type.bequals(getOperand1().Type(), getOperand2().Type()) &&
            (getOperand1().Type().isReal() ||
            getOperand1().Type().isInteger()) ?
                        getOperand1().Type() : Type.ERROR_TYPE;
```


Resource - siple_test/src/test.siple - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Quick Access
Resource
Project Explorer
siple_test
src
test.siple
JRE System Library [JavaSE
test.siple
Procedure main() Begin
x := 0;
y := 0;
z := 0;
Var x:Integer;
x := 100;
Procedure writeMain() Begin
Write x;
Write y;
Write z;
End;
Line: 29
Procedure l1() Begin
Var y:Integer;
y := x + 100;
Procedure writeL1() Begin
Write x;
Write y;
Write z;
End;
Outline
Compilation Unit
x
y
z
writeGlob
Block
incGlobVars
Block
Variable Assignment Integer
Variable Assignment Integer
Variable Assignment Integer
main
Block
Variable Assignment Integer
Variable Assignment Integer
Variable Assignment Integer
x
Variable Assignment Integer
writeMain
l1
Tasks
Task List
Properties
Property
Value
As Boolean
As Integer
100
As Real
Lexem
100
Procedure In Context
Procedure Declaration main
Type
Integer
Writable
Insert
28 : 9

Literature

[Hedin00] Hedin, Görel. 2000. Reference Attributed Grammars. *Informatica (Slovenia)* 24, Nr. 3: 301–317.

[Boyland05] Boyland, John T. 2005. Remote attribute grammars. *Journal of the ACM* 52, Nr. 4 (July): 627–687.

[Bürger+11] Bürger, Christoff, Sven Karol, Christian Wende, und Uwe Aßmann. 2011. Reference Attribute Grammars for Metamodel Semantics. In *Software Language Engineering*. Springer Berlin / Heidelberg.

[Knuth68] Knuth, D. E. 1968. Semantics of context-free languages. *Theory of Computing Systems* 2, Nr. 2: 127–145.

[Vogt+89] Vogt, Harald H, Doaitse Swierstra, und Matthijs F Kuiper. 1989. Higher Order Attribute Grammars. In *PLDI '89*, 131–145. ACM.

[Ekman06] Ekman, Torbjörn. 2006. Extensible Compiler Construction. University of Lund.

[Hedin11] An Introductory Tutorial on JastAdd Attribute Grammars. In *Generative and Transformational Techniques in Software Engineering III,* 6491:166-200. Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

[Bürger+10] Bürger, Christoff, Sven Karol, und Christian Wende. 2010. Applying attribute grammars for metamodel semantics. In *Proceedings of the International Workshop on Formalization of Modeling Languages*, 1:1–1:5. FML '10. New York, NY, USA: ACM.

[Kühnemann+97] Kühnemann, Armin, und Heiko Vogler. 1997. *Attributgrammatiken -- Eine grundlegende Einführung*. Braunschweig/Wiesbaden: Vieweg.

[Heidenreich+12] Heidenreich, Florian, Jendrik Johannes, Sven Karol, Mirko Seifert, und Christian Wende. 2012. „Model-based Language Engineering with EMFText“. In Generative and Transformational Techniques in Software Engineering, 7680:322ff. Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

Thank you!