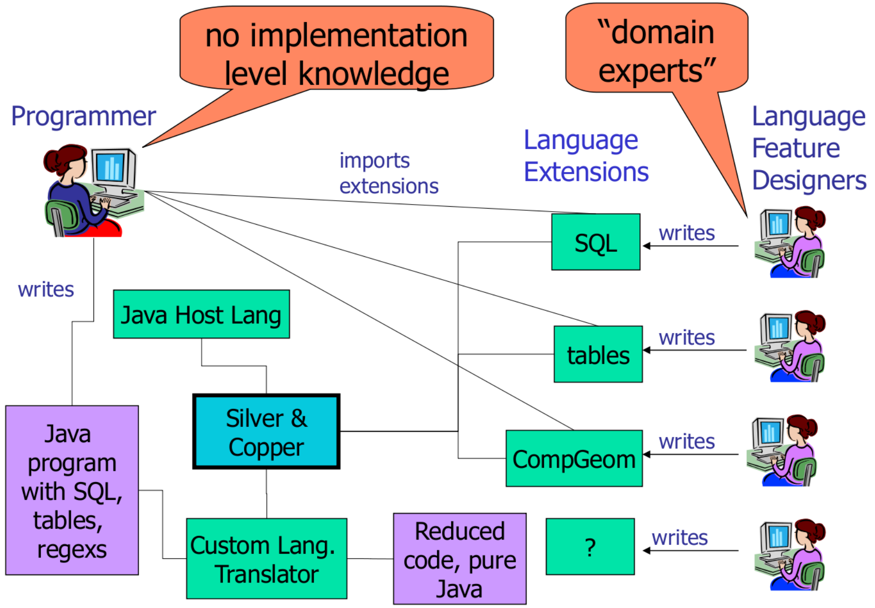
Modular Analysis of Attribute Grammars

Ted Kaminski and Eric Van Wyk

University of Minnesota

WG 2.11 - June, 2012, Halmstad

Extensible language frameworks and modular analysis


no implementation level knowledge
"domain experts"
Programmer
Language Extensions
Language Feature Designers
imports extensions
writes
SQL
writes
Java Host Lang
tables
writes
Silver & Copper
CompGeom
writes
Java program with SQL, tables, regexs
Custom Lang. Translator
Reduced code, pure Java
?
writes

- natural syntax
- semantic analysis
- composable extensions

```
class Demo {
  int demoMethod ( ) {
    List<List<Integer>> dlist ;
    int T ;
    int SELECT ;
    connection c "jdbc:derby:/home/derby/db/testdb"
      with table person [ person_id INTEGER,
                          first_name VARCHAR,
                          last_name VARCHAR ] ,
           table details [ person_id INTEGER,
                           age INTEGER ] ;
    Integer limit = 18 ;
    ResultSet rs = using c query {
          SELECT age, gender, last_name
           FROM person , details
          WHERE person.person_id = details.person_id
           AND details.age > limit } ;
    Integer = rs.getInteger("age");
    String gender = rs.getString("gender");
    boolean b ;
    b = table ( age > 40      :  T * ,
                gender == "M" :  T F ) ;
  }
}
```

Composing language specifications

- It should "just work".
- Many formalisms naturally compose.
- Take the set union of the component sets.
  - context free grammars
  - attribute grammars
  - term rewriting systems
- But certain desirable properties may not hold.

Context free grammars

$$G_H \quad \cup \quad G_E^1 \quad \cup \quad G_E^2 \quad \cup \quad \dots \quad \cup \quad G_E^i$$

- $\cup$ of sets of nonterminals, terminals, productions
- Composition of all is an context free grammar.
- Is it ambiguous, useful for deterministic (LR) parsing?

Context free grammars

$$\boxed{G_H} \cup \boxed{G_E^1} \cup G_E^2 \cup \ldots \cup G_E^i$$

- $\cup$ of sets of nonterminals, terminals, productions
- Composition of all is an context free grammar.
- Is it ambiguous, useful for deterministic (LR) parsing?
- $\mathit{conflictFree}(G_H \cup G_E^1)$ holds

Context free grammars

$$\boxed{G_H} \cup G_E^1 \cup \boxed{G_E^2} \cup \ldots \cup G_E^i$$

- $\cup$ of sets of nonterminals, terminals, productions
- Composition of all is an context free grammar.
- Is it ambiguous, useful for deterministic (LR) parsing?
- $\mathit{conflictFree}(G_H \cup G_E^1)$ holds
- $\mathit{conflictFree}(G_H \cup G_E^2)$ holds

Context free grammars

$$\boxed{G_H} \cup G_E^1 \cup G_E^2 \cup \ldots \cup \boxed{G_E^i}$$

- $\cup$ of sets of nonterminals, terminals, productions
- Composition of all is an context free grammar.
- Is it ambiguous, useful for deterministic (LR) parsing?
- *conflictFree*$(G_H \cup G_E^1)$ holds
- *conflictFree*$(G_H \cup G_E^2)$ holds
- *conflictFree*$(G_H \cup G_E^i)$ holds

Context free grammars

$$\boxed{G_H} \cup \boxed{G_E^1} \cup \boxed{G_E^2} \cup \ \dots \ \cup \boxed{G_E^i}$$

- $\cup$ of sets of nonterminals, terminals, productions
- Composition of all is an context free grammar.
- Is it ambiguous, useful for deterministic (LR) parsing?
- $\mathit{conflictFree}(G_H \cup G_E^1)$ holds
- $\mathit{conflictFree}(G_H \cup G_E^2)$ holds
- $\mathit{conflictFree}(G_H \cup G_E^i)$ holds
- $\mathit{conflictFree}(G_H \cup G_E^1 \cup G_E^2 \cup \ \dots \ \cup G_E^i)$ may not hold

Attribute grammars

$$AG_H \;\cup\; AG_E^1 \;\cup\; AG_E^2 \;\cup\; \ldots \;\cup\; AG_E^i$$

- $\cup$ of sets of attributes, attribute equations, occurs-on declarations
- Composition of all is an attribute grammar.
- Completeness: $\forall$ production, $\forall$ attribute, $\exists$ an equation

Attribute grammars

$$\boxed{AG_H} \cup \boxed{AG_E^1} \cup AG_E^2 \cup \ldots \cup AG_E^i$$

- $\cup$ of sets of attributes, attribute equations, occurs-on declarations
- Composition of all is an attribute grammar.
- Completeness: $\forall$ production, $\forall$ attribute, $\exists$ an equation
- $complete(AG_H \cup AG_E^1)$ holds

Attribute grammars

$$\boxed{AG_H} \cup AG_E^1 \cup \boxed{AG_E^2} \cup \ldots \cup AG_E^i$$

- $\cup$ of sets of attributes, attribute equations, occurs-on declarations
- Composition of all is an attribute grammar.
- Completeness: $\forall$ production, $\forall$ attribute, $\exists$ an equation
- $complete(AG_H \cup AG_E^1)$ holds
- $complete(AG_H \cup AG_E^2)$ holds

Attribute grammars

$$\boxed{AG_H} \cup AG_E^1 \cup AG_E^2 \cup \ldots \cup \boxed{AG_E^i}$$

- $\cup$ of sets of attributes, attribute equations, occurs-on declarations
- Composition of all is an attribute grammar.
- Completeness: $\forall$ production, $\forall$ attribute, $\exists$ an equation
- $complete(AG_H \cup AG_E^1)$ holds
- $complete(AG_H \cup AG_E^2)$ holds
- $complete(AG_H \cup AG_E^i)$ holds

Attribute grammars

$$\boxed{AG_H} \cup \boxed{AG_E^1} \cup \boxed{AG_E^2} \cup \ \dots \ \cup \boxed{AG_E^i}$$

- $\cup$ of sets of attributes, attribute equations, occurs-on declarations
- Composition of all is an attribute grammar.
- Completeness: $\forall$ production, $\forall$ attribute, $\exists$ an equation
- $complete(AG_H \cup AG_E^1)$ holds
- $complete(AG_H \cup AG_E^2)$ holds
- $complete(AG_H \cup AG_E^i)$ holds
- $complete(AG_H \cup AG_E^1 \cup AG_E^2 \cup \ \dots \ \cup AG_E^i)$ may not hold

Attribute grammars

$$\boxed{AG_H} \cup \boxed{AG_E^1} \cup \boxed{AG_E^2} \cup \ \ldots \ \cup \boxed{AG_E^i}$$

- $\cup$ of sets of attributes, attribute equations, occurs-on declarations
- Composition of all is an attribute grammar.
- Completeness: $\forall$ production, $\forall$ attribute, $\exists$ an equation
- $complete(AG_H \cup AG_E^1)$ holds
- $complete(AG_H \cup AG_E^2)$ holds
- $complete(AG_H \cup AG_E^i)$ holds
- $complete(AG_H \cup AG_E^1 \cup AG_E^2 \cup \ \ldots \ \cup AG_E^i)$ may not hold
- similarly for non-circularity of the AG

Term rewriting systems

... similarly ...

So, they easily compose.

The non-expert program is happy, maybe.

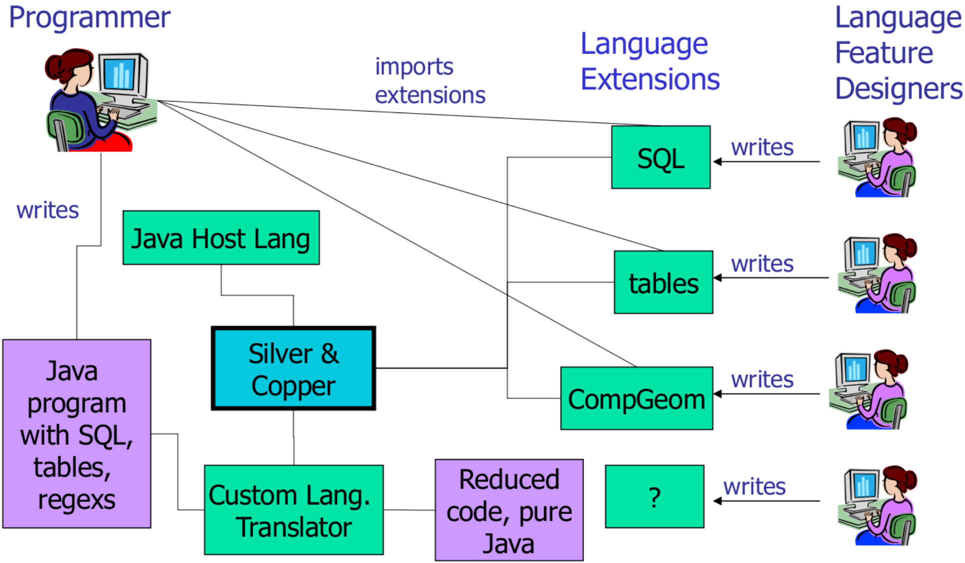
But what assurances can we provide that the composition will be a “good” one?

Detecting problems, ensuring composition

When can some analysis of the language specification be applied?
When ...

1. the host language is developed ?
2. a language extensions is developed ?
3. when the host and extensions are composed ?
4. when the resulting language tools is run ?

When to analyze



Libraries, and modular type checking

- Libraries "just work"
- Type checking is done by the library writer, modularly.
- Language extensions should be like libraries, composition of "verified" extensions should "just work."

Modular determinism analysis for grammars, 2009

$$G_H \;\cup\; G_E^1 \;\cup\; G_E^2 \;\cup\; \ldots \;\cup\; G_E^i$$

Modular determinism analysis for grammars, 2009

$$\boxed{G_H} \cup \boxed{G_E^1} \cup G_E^2 \cup \ldots \cup G_E^i$$

- $\mathit{isComposable}(G_H, G_E^1) \wedge \mathit{conflictFree}(G_H \cup G_E^1)$ holds

Modular determinism analysis for grammars, 2009

$$\boxed{G_H} \cup G_E^1 \cup \boxed{G_E^2} \cup \ldots \cup G_E^i$$

- $isComposable(G_H, G_E^1) \wedge conflictFree(G_H \cup G_E^1)$ holds
- $isComposable(G_H, G_E^2) \wedge conflictFree(G_H \cup G_E^2)$ holds

Modular determinism analysis for grammars, 2009

$$\boxed{G_H} \cup G_E^1 \cup G_E^2 \cup \ldots \cup \boxed{G_E^i}$$

- $isComposable(G_H, G_E^1) \wedge conflictFree(G_H \cup G_E^1)$ holds
- $isComposable(G_H, G_E^2) \wedge conflictFree(G_H \cup G_E^2)$ holds
- $isComposable(G_H, G_E^i) \wedge conflictFree(G_H \cup G_E^i)$ holds

Modular determinism analysis for grammars, 2009

$$\boxed{G_H} \cup \boxed{G_E^1} \cup \boxed{G_E^2} \cup \ \ldots \ \cup \boxed{G_E^i}$$

- *isComposable*$(G_H, G_E^1) \wedge$ *conflictFree*$(G_H \cup G_E^1)$ holds
- *isComposable*$(G_H, G_E^2) \wedge$ *conflictFree*$(G_H \cup G_E^2)$ holds
- *isComposable*$(G_H, G_E^i) \wedge$ *conflictFree*$(G_H \cup G_E^i)$ holds
- these imply *conflictFree*$(G_H \cup G_E^1 \cup G_E^2 \cup \ \ldots)$ holds

Modular determinism analysis for grammars, 2009

$$\boxed{G_H} \cup \boxed{G_E^1} \cup \boxed{G_E^2} \cup \ \ldots \ \cup \boxed{G_E^i}$$

- $isComposable(G_H, G_E^1) \wedge conflictFree(G_H \cup G_E^1)$ holds
- $isComposable(G_H, G_E^2) \wedge conflictFree(G_H \cup G_E^2)$ holds
- $isComposable(G_H, G_E^i) \wedge conflictFree(G_H \cup G_E^i)$ holds
- these imply $conflictFree(G_H \cup G_E^1 \cup G_E^2 \cup \ \ldots)$ holds
- $$(\ \forall i \in [1, n]. isComposable(G_H, G_E^i) \ \wedge \ conflictFree(G_H \cup \{G_E^i\}\} \ ) \implies conflictFree(G_H \cup \{G_E^1, \ldots, G_E^n\})$$
- Some restrictions to extension introduced syntax apply, of course.

Modular completeness analysis for attribute grammars

$$AG_H \;\cup\; AG_E^1 \;\cup\; AG_E^2 \;\cup\; \ldots \;\cup\; AG_E^i$$

Modular completeness analysis for attribute grammars

$$\boxed{AG_H} \cup \boxed{AG_E^1} \cup AG_E^2 \cup \ldots \cup AG_E^i$$

- $modComplete(AG_H \cup AG_E^1)$ holds

Modular completeness analysis for attribute grammars

$$\boxed{AG_H} \cup AG_E^1 \cup \boxed{AG_E^2} \cup \ldots \cup AG_E^i$$

- $modComplete(AG_H \cup AG_E^1)$ holds
- $modComplete(AG_H \cup AG_E^2)$ holds

Modular completeness analysis for attribute grammars

$$\boxed{AG_H} \cup AG_E^1 \cup AG_E^2 \cup \ldots \cup \boxed{AG_E^i}$$

- $modComplete(AG_H \cup AG_E^1)$ holds
- $modComplete(AG_H \cup AG_E^2)$ holds
- $modComplete(AG_H \cup AG_E^i)$ holds

Modular completeness analysis for attribute grammars

$$\boxed{AG_H} \cup \boxed{AG_E^1} \cup \boxed{AG_E^2} \cup \ \ldots \ \cup \boxed{AG_E^i}$$

- $modComplete(AG_H \cup AG_E^1)$ holds
- $modComplete(AG_H \cup AG_E^2)$ holds
- $modComplete(AG_H \cup AG_E^i)$ holds
- these imply $complete(AG_H \cup AG_E^1 \cup AG_E^2 \cup \ \ldots)$ holds

Modular completeness analysis for attribute grammars

$$\boxed{AG_H} \cup \boxed{AG_E^1} \cup \boxed{AG_E^2} \cup \ \ldots \ \cup \boxed{AG_E^i}$$

- $modComplete(AG_H \cup AG_E^1)$ holds
- $modComplete(AG_H \cup AG_E^2)$ holds
- $modComplete(AG_H \cup AG_E^i)$ holds
- these imply $complete(AG_H \cup AG_E^1 \cup AG_E^2 \cup \ \ldots)$ holds
- $(\forall i \in [1, n].modComplete(AG_H, AG_E^i)) \implies complete(AG_H \cup \{AG_E^1, \ldots, AG_E^n\}).$

Modular completeness analysis for attribute grammars

$$\boxed{AG_H} \cup \boxed{AG_E^1} \cup \boxed{AG_E^2} \cup \ \dots \ \cup \boxed{AG_E^i}$$

- $modComplete(AG_H \cup AG_E^1)$ holds
- $modComplete(AG_H \cup AG_E^2)$ holds
- $modComplete(AG_H \cup AG_E^i)$ holds
- these imply $complete(AG_H \cup AG_E^1 \cup AG_E^2 \cup \ \dots)$ holds
- $(\forall i \in [1, n].modComplete(AG_H, AG_E^i))$ $\implies complete(AG_H \cup \{AG_E^1, ..., AG_E^n\})$.
- similarly for non-circularity of the AG
- Again, some restrictions on extensions.

The details ...

Attribute grammars (AGs), quick refresher

AGs add two things to context free grammars

1. attributes: named values, with specified types.
   - These "decorate" nonterminals.
   - Synthesized attributes propagate information up the tree.
   - Inherited attributes ... down the tree.
   - *e.g.* `errors :: [ String ] occurs on Expr`
   - *e.g.* `cCodeTrans :: String occurs on Stmt ...`
   - *e.g.* `type :: TypeRep`
   - *e.g.* `env :: [ Map<String, Declaration> ]`

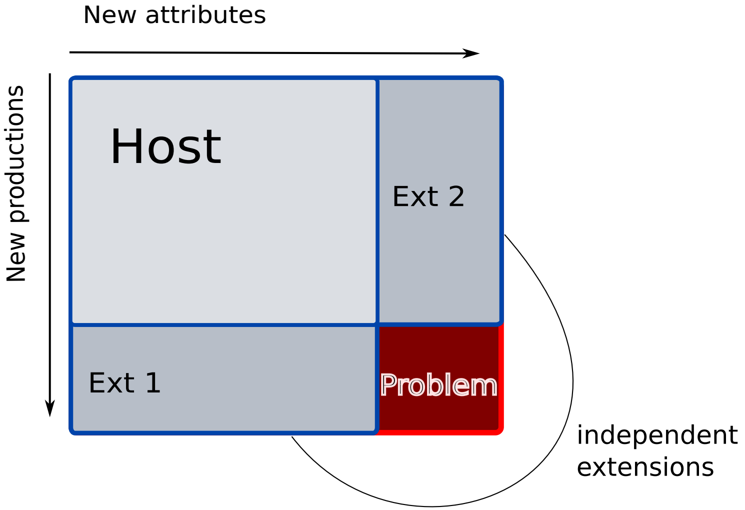
Attribute grammars (AGs), quick refresher

AGs add two things to context free grammars

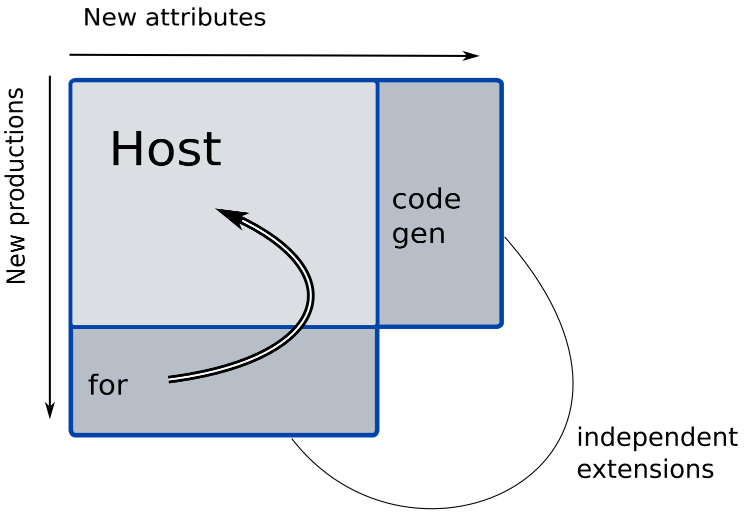
2. attribute equations, associated with productions
    - These assign values to attributes on nodes referenced in the production.
    - For example:

```
production add
e::Expr ::= l::Expr '+' r::Expr
{
 e.errors = l.errors ++ r.errors ++
       if ( ... addition no defined on l.type ...)
       then [ "Error on addition..." ]
       else [ ] ;
 e.type = resolve ( l.type, r.type ) ;
 l.env = e.env ;
 r.env = e.env ;
}
```

Extensibility: safe composability



Extensibility: safe composability



Forwarding

- A production builds another AST and forwards requests for undefined attributes to it.
- Language extension productions forward to their translation in the host language.
  - That is, a tree of host language constructs.
- Forwarding make completeness possible, but doesn't ensure it.

```
production for_loop
s::Stmt ::= var::Name lower::Expr upper:Expr body::Stmt
{ s.errors = ... ;
  forwards to seq ( assign (var, lower) ,
                    while ( lessThanEqual (var, upper),
                        seq ( body ,
                              assign ( var,
                                       plus (var, one) ) )
```

Modular completeness analysis

- The analysis *modComplete* is defined as follows:

$$
\begin{aligned}
&\mathit{modComplete}(AG^H, AG^E) \triangleq \\
&\qquad \mathit{noOrphanOccursOn}(AG^H, AG^E) \ \wedge \\
&\qquad \mathit{noOrphanAttrEqs}(AG^H, AG^E) \wedge \\
&\qquad \mathit{noOrphanProds}(AG^H, AG^E) \wedge \\
&\qquad \mathit{synComplete}(AG^H, AG^E) \wedge \\
&\qquad \mathit{modularFlowTypes}(\mathit{flowTypes}(AG^H), \\
&\qquad\qquad\qquad\qquad \mathit{flowTypes}(AG^H \cup AG^E)) \ \wedge \\
&\qquad \mathit{inhComplete}(AG^H, AG^E, \mathit{flowTypes}(AG^H \cup AG^E))
\end{aligned}
$$

- Some "structural" requirements, some flow-type requirements

- Silver's module system prevents duplicate nonterminal, attribute, production declarations.

No orphan occurs on declarations

"an AG declares `a occurs on nt` only if it declares or exports `a` or `nt`"

$noOrphanOccursOn(AG^H, AG^E)$

holds if and only if each occurs-on declaration
"`attribute` a `occurs on` nt" in $AG^H \cup AG^E$
is exported by the grammar declaring a or the grammar declaring nt.

No orphan attribute equations

"an AG provides an equation `n.a=e` on `p` with l.h.s. nonterminal `nt` only if it declares/exports the production `p` or the occurs-on declaration `a occurs on nt`."

$noOrphanAttrEqs(AG^H, AG^E)$

holds if and only if each equation $n.a = e$ in a production p is exported by the grammar declaring the (non-aspect) production p or the grammar declaring the occurs-on declaration "`attribute` a `occurs on` nt" (where n has type nt.)

No orphan production declarations

"Productions in extensions that build host language nonterminals must forward."

$noOrphanProds(AG^H, AG^E)$

holds if and only if for each production declaration p in $AG^H \cup AG^E$ with left hand side nonterminal nt, the production p is either exported by the grammar declaring nt, or p forwards.

Completeness of synthesized equations

"A production forwards or has equations for all its attributes, these may be on aspect productions."

$synComplete(AG^H, AG^E)$

holds if and only if for each occurs-on declaration `attribute` a `occurs on` nt, and for each non-forwarding production p that constructs nt, there exists a rule defining the synthesized equation $p : x.a$, where x is the left hand side of the production.

Flow Types

- A flow-type captures how information flow between attributes.
- For a non-terminal, it maps synthesized attributes to the inherited attributes on which it depends.
- $ft_{nt} :: A_s \rightarrow 2^{A_I}$
- *e.g.* $ft_{Expr}(type) = \{env\}$

Modularity of flow types

"Host language attributes on host language nonterminals do not depend on extension-declared inherited attributes."

$$\begin{aligned}&\mathit{modularFlowTypes}(\mathit{flowTypes}(AG^H),\\ &\qquad\qquad\qquad \mathit{flowTypes}(AG^H \cup AG^E))\end{aligned}$$

holds if and only if for each $ft^H_{nt} \in \mathit{flowTypes}(AG^H)$ and $ft^{H\cup E}_{nt} \in \mathit{flowTypes}(AG^H \cup AG^E)$,
for all synthesized attributes s and for all nonterminals nt such that `attribute` s `occurs on` nt is declared in AG^H,
$ft^{H\cup E}_{nt}(s) \subseteq ft^H_{nt}(s)$.

Effective completeness of inherited equations

"All inherited attributes required to compute a synthesized attribute on a node have defining equations."

$inhComplete(AG^H, AG^E, flowTypes(AG^H \cup AG^E))$

holds if and only if for every production p in $AG^H \cup AG^E$ and for every access to a synthesized attribute $n.s$ in an expression within p (where n has type nt,) and for each inherited attribute $i \in ft_{nt}(s)$, there exists an equation $n.i = e$ for p.

A generalization

- Silver does not identify "host" and "extensions."
- The analysis works on `import` grammar relationships.
- A grammar `A` that imports `B` is seen as an extension to `B`.
- So, Silver libraries are hosts, in essence.

Evaluation

- Q: Are these restrictions too overbearing?
  A1: No, they are turned on by default.
  A2: No, except for reference attributes.

- Applied to Silver compiler's Silver specs
  1. We found a few bugs.
  2. We moved some declarations to new grammars.
     These were bad design "smells."
  3. We extended Silver
     - Annotations allowing host language to be modularized without respect to the rules.
       These treat the "whole" host language, spread across many modules, as one for the analysis.
     - Default attributes.

- Reference attributes - rather severe restrictions.
  All inherited attributes must be provided and no more can be added.

Modular circularity analysis

- Analysis extends to check for no cycles, modularly.
- Instead of a flow type for each non-terminal, there is a set of flow-graphs for each.
- Analysis ensures no new patterns of information flow are added by an extension.

Lessons learned ?

- For extensible language frameworks
  - Analysis - modular or don't bother?
- But my undergrads like static completeness detection.

Some questions ...

Some questions ... what to call these?

- We've called this a "modular" analysis.
- Maybe "static composition analysis"
  - analysis that happens before composition
- As opposed to "dynamic composition analysis"
  - analysis that happens during composition
- But "static" and "dynamic" suggest 2 points in time
  - before and during run-time
- We have more than 2 interesting points in time.
  - 1. host development, 2. extension development, 3. composition, 4. translation, 5. execution

Thanks for your attention.

Questions?

http://melt.cs.umn.edu
evw@cs.umn.edu