Algorithmic Debugging for Attribute Grammars

Praveen Kambam Sugavanam and Eric Van Wyk

University of Minnesota

WG 2.11 - Minneapolis

History of Computing in the Midwest links

- http://en.wikipedia.org/wiki/Atanasoff-Berry_Computer
- http://upload.wikimedia.org/wikipedia/commons/9/90/Atanasoff-Berry_Computer.jpg
- http://en.wikipedia.org/wiki/Engineering_Research_Associates
- http://en.wikipedia.org/wiki/Control_Data_Corporation
- http://en.wikipedia.org/wiki/Blue_Gene
- http://www.mnhs.org/collections/medTech/individuals.html
- http://www.cbi.umn.edu/
- http://www.cbi.umn.edu/resources/MHHC/index.html
- http://www1.umn.edu/twincities/campus-facts/index.html
- http://www.cs.umn.edu/
- http://en.wikipedia.org/wiki/Gopher_(protocol)
- http://www.movielens.org/login
- http://climatechange.cs.umn.edu/

DSLs are bad, or at least have some challenges...

- External
  - "No recourse to the law."
  - Domain-specific syntax, analysis, and optimization
- Internal/Embedded
  - "just libraries"
  - composable
  - general purpose host language available

Maybe there are alternatives ...

Extensible Languages (Frameworks)

- pluggable domain-specific language extensions
- Domain-specific syntax, analysis, and optimization
- composable
- general purpose host language available
- Extension features translate down to host language

```
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 ) ;
  }
}
```

- natural syntax
- semantic analysis
- composable extensions

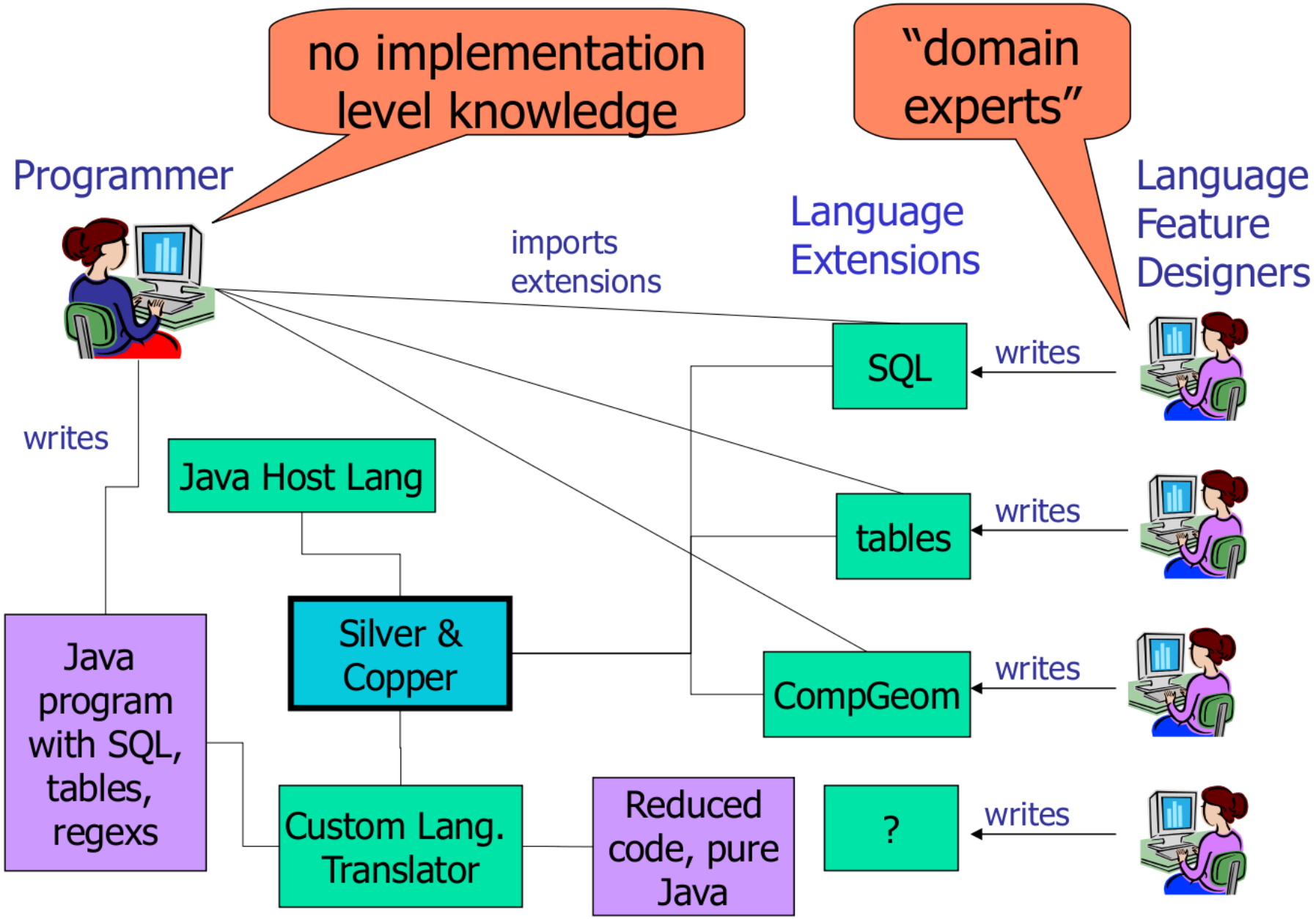
```
#include <sdtio.h>

int main() {

    ... bits of SAC ...

    ... stencil specifications ...

    ... computational geometry optimizations
        robustness transformations ...
}
```

Roles people play ...

1. host language designer

2. language extension designer

3. programmer
   - language extension user


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

A product line of compilers ...

but we don't know what the variants are.

Building a parser from composed specifications.

$$\ldots \; CFG^{H} \cup \{CFG^{E_1}, \ldots, CFG^{E_n}\}$$

$$\begin{aligned} \forall i \in [1, n].\mathit{isComposable}(CFG^{H}, CFG^{E_i}) \wedge \\ \mathit{conflictFree}(CFG^{H} \cup CFG^{E_i}) \\ \Rightarrow \quad \mathit{conflictFree}(CFG^{H} \cup \{CFG^{E_1}, \ldots, CFG^{E_n}\}) \end{aligned}$$

- Monolithic analysis - not too hard, but not too useful.
- Modular analysis - harder, but required.

Building an attribute grammar evaluator from composed specifications.

$$\ldots\ AG^H \cup \{AG^{E_1}, \ldots, AG^{E_n}\}$$

$$\forall i \in [1, n].modComplete(AG^H, AG^{E_i})$$
$$\Rightarrow \quad complete(AG^H \cup \{AG_1^E, \ldots, AG_n^E\})$$

- Monolithic analysis - not too hard, but not too useful.
- Modular analysis - harder, but required.

Analyses impose some restrictions.

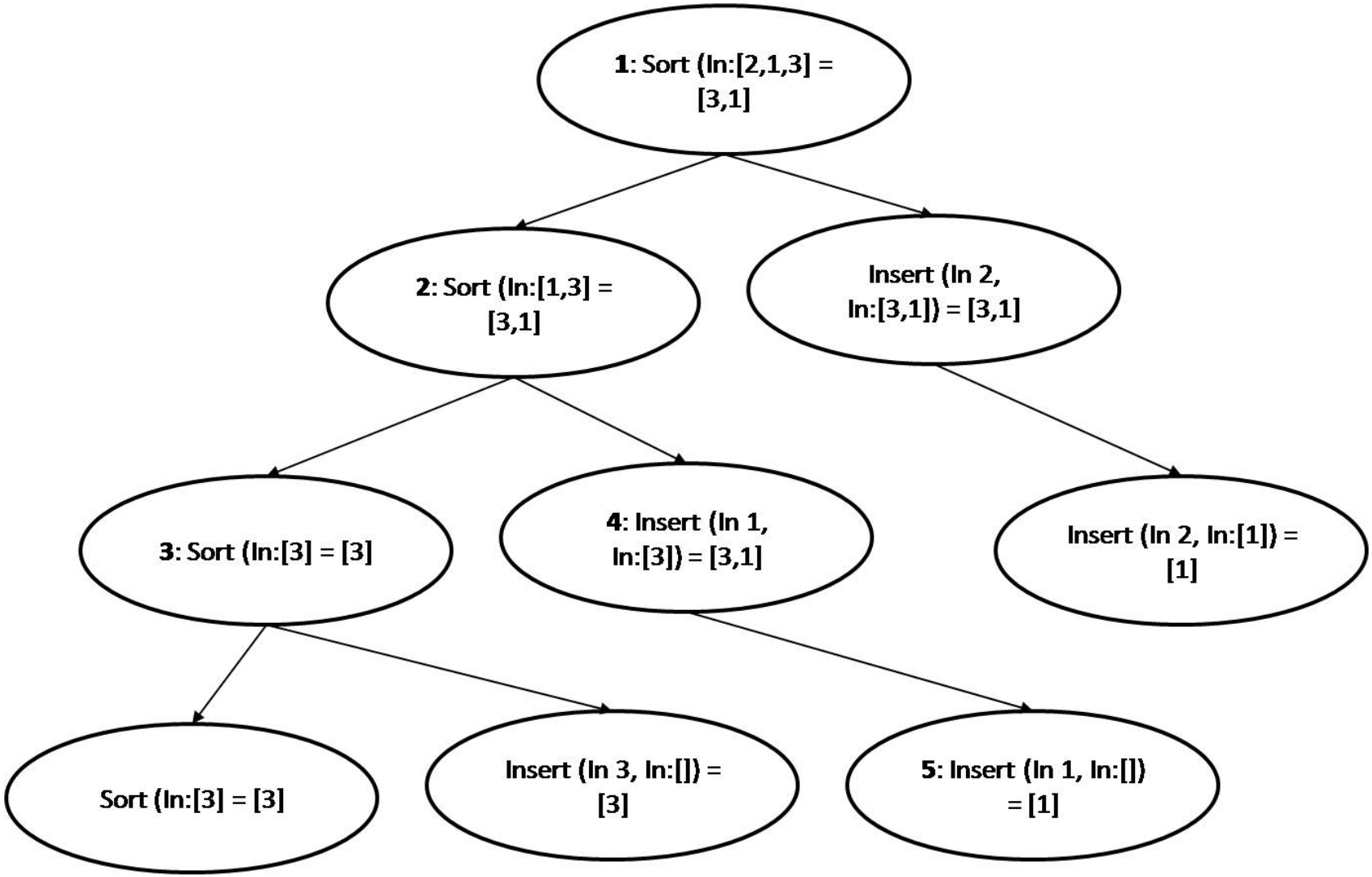
We don't care about you (the language or extension designer).

We care about the programmer.

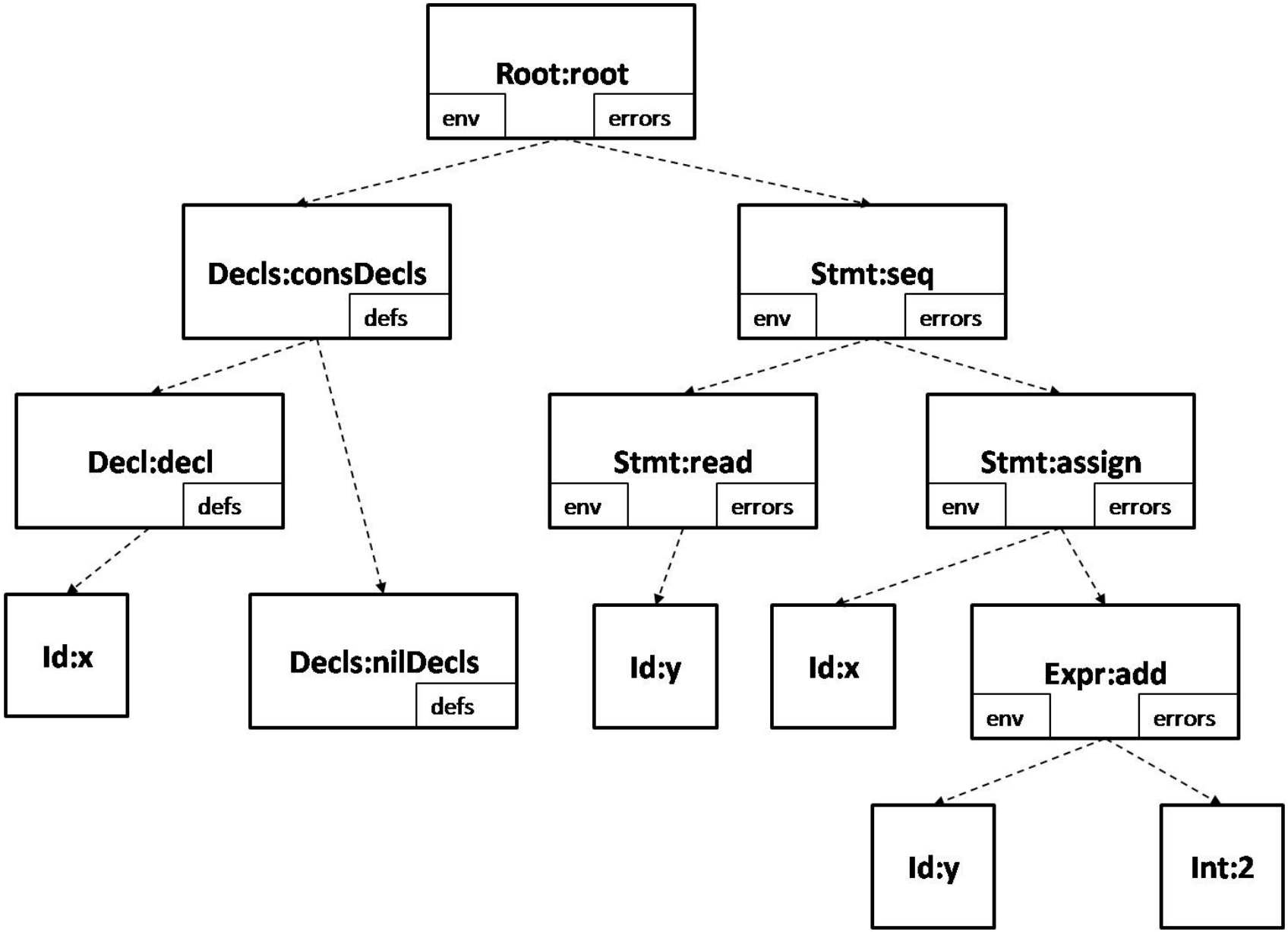
Well, we care about you a little...

Debugging attribute grammars

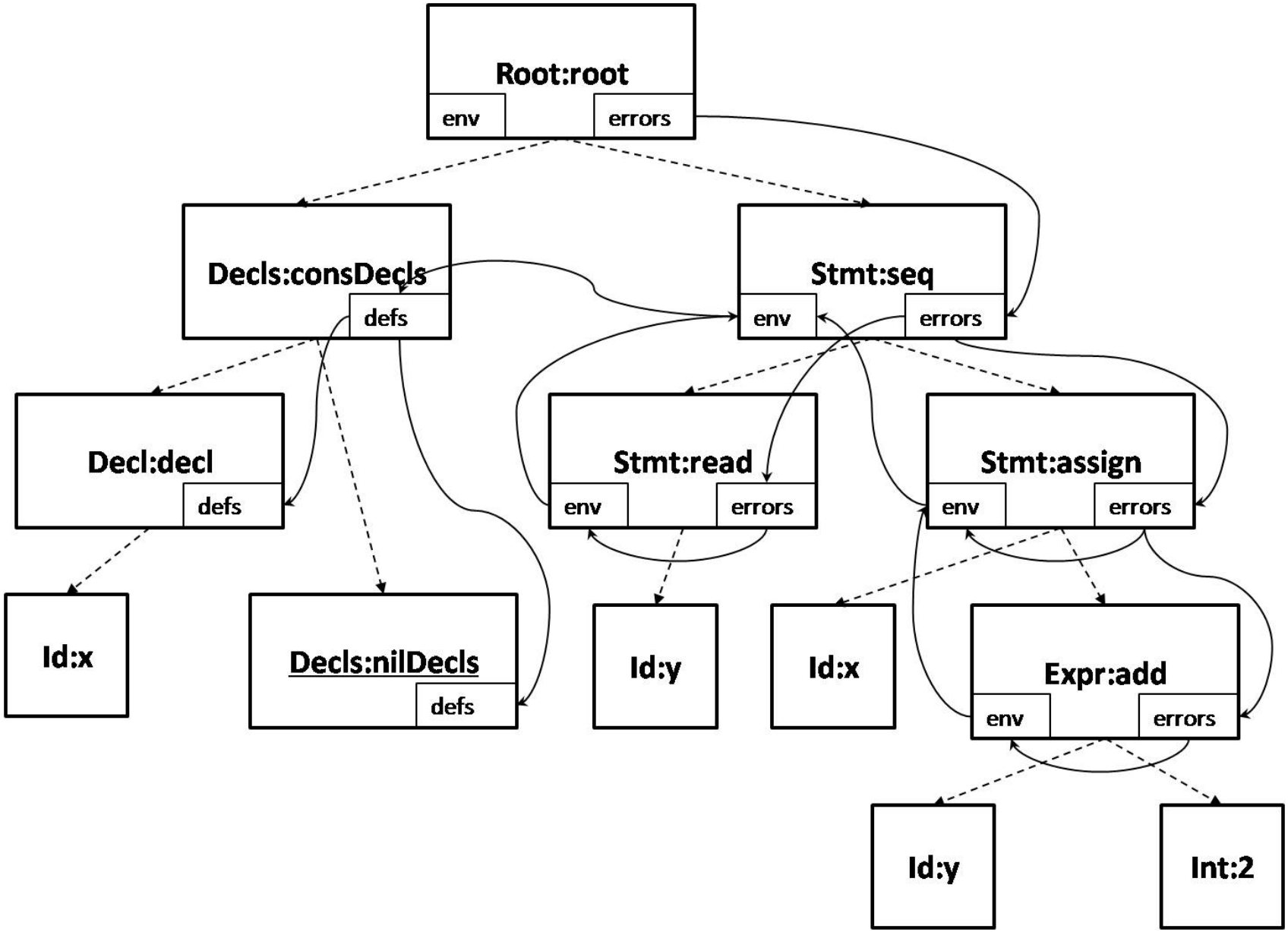
- AGs are a flavor of lazy functional programming...
- Algorithmic debugging
- State of the art: “debugging by staring”


1: Sort (In:[2,1,3] = [3,1]
2: Sort (In:[1,3] = [3,1]
Insert (In 2, In:[3,1]) = [3,1]
3: Sort (In:[3] = [3]
4: Insert (In 1, In:[3]) = [3,1]
Insert (In 2, In:[1]) = [1]
Sort (In:[3] = [3]
Insert (In 3, In:[]) = [3]
5: Insert (In 1, In:[]) = [1]

```
Integer x ;
begin
  read y ;
  x := y + 2
end
```


Root:root
env
errors
Decls:consDecls
defs
Stmt:seq
env
errors
Decl:decl
defs
Stmt:read
env
errors
Stmt:assign
env
errors
Id:x
Decls:nilDecls
defs
Id:y
Id:x
Expr:add
env
errors
Id:y
Int:2

- errors attribute =
  - “line 3: y is not defined”
- This is not correct.


Root:root
env
errors
Decls:consDecls
defs
Stmt:seq
env
errors
Decl:decl
defs
Stmt:read
env
errors
Stmt:assign
env
errors
Id:x
Decls:nilDecls
defs
Id:y
Id:x
Expr:add
env
errors
Id:y
Int:2

```
abstract production assign
s::Stmt ::= id::Id e::Expr
{
 e.env = s.env;
 s.errors
   = if (checkInList(id.lexeme,s.env) == false) then
       id.lexeme ++ " has not been defined.\n" ++
       e.errors
     else
       "";
}
```

Enhancements

- asking fewer questions
- text highlighting instead of tree drawing

Thanks for your attention.

Questions?

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