**Software Transformation**

2 questions in transformation:

1. what constructs are to be transformed?
2. what are they to be transformed into?

**Our Position:** Transformations can be made more expressive and useful when they are informed by semantic information of the source.

- This can be general-purpose information like a constructs type
- or domain-specific information like the space required for unbounded integers in a computational geometry application.
Context: Extensible Languages

Programmer

selects

writes

Program with SQL and CG constructs

Host Language Specification

Language Extensions

Feature Designers

implements

SQL

CG

implements

...

implements

Extensible Compiler Tools

generates

input

Customized Compiler

output

Executable Program
Attribute Grammars with Forwarding

```java
foreach Cow c in herd do c.milk();
⇒
{ Cow c ;
  for ( Iterator iter = herd.iterator(); iter.hasNext(); )
  { c = ( Cow ) iter.next(); c.milk(); } }

foreach:  ⟨St⟩ ::= “foreach” ⟨Type⟩ ⟨Id⟩ “in” ⟨E⟩ “do” ⟨St⟩

St.errors = if not Type.implements(Collection) then ...

forwardsTo “... specification of above for loop ... ”
```

Graphically...

```
for-each forwards to block
Id  Type  Expr  Stmt
... forwardsTo ...
Stmts
Dcl  for
```

Semantic Analysis in Software Transformation, Eric Van Wyk & Eric Johnson
For example: Computational Geometry

- Algorithms based on primitives that make qualitative decision: e.g. is “a point \( x \) to the left or right of line \( l \)”
- Many algorithms are simplified if intermediate values can have unbounded precision.
- We can statically compute their size in bits.
- \texttt{unbounded\_add}: \( \langle E \rangle ::= \langle E \rangle +_u \langle E \rangle \)
  \[ E_0\text{.size} = \max(E_1\text{.size}, E_2\text{.size}) + 1 \]
- \texttt{unbounded\_var}: \( \langle E \rangle ::= \langle Id \rangle \)
  \[ E\text{.size} = 53 \]
- Generate “unrolled loops” that perform these operations
Question(s):

Can we unify semantic analysis and efficient rewriting mechanisms?

- Can we do more than combine incremental attribute grammar evaluators and conditional rewrite rules?

- For example, do some transformations preserve semantic values (such as type) so that attribute re-computation is not necessary?

- Can rewrite rules also define semantics for the rewritten term?