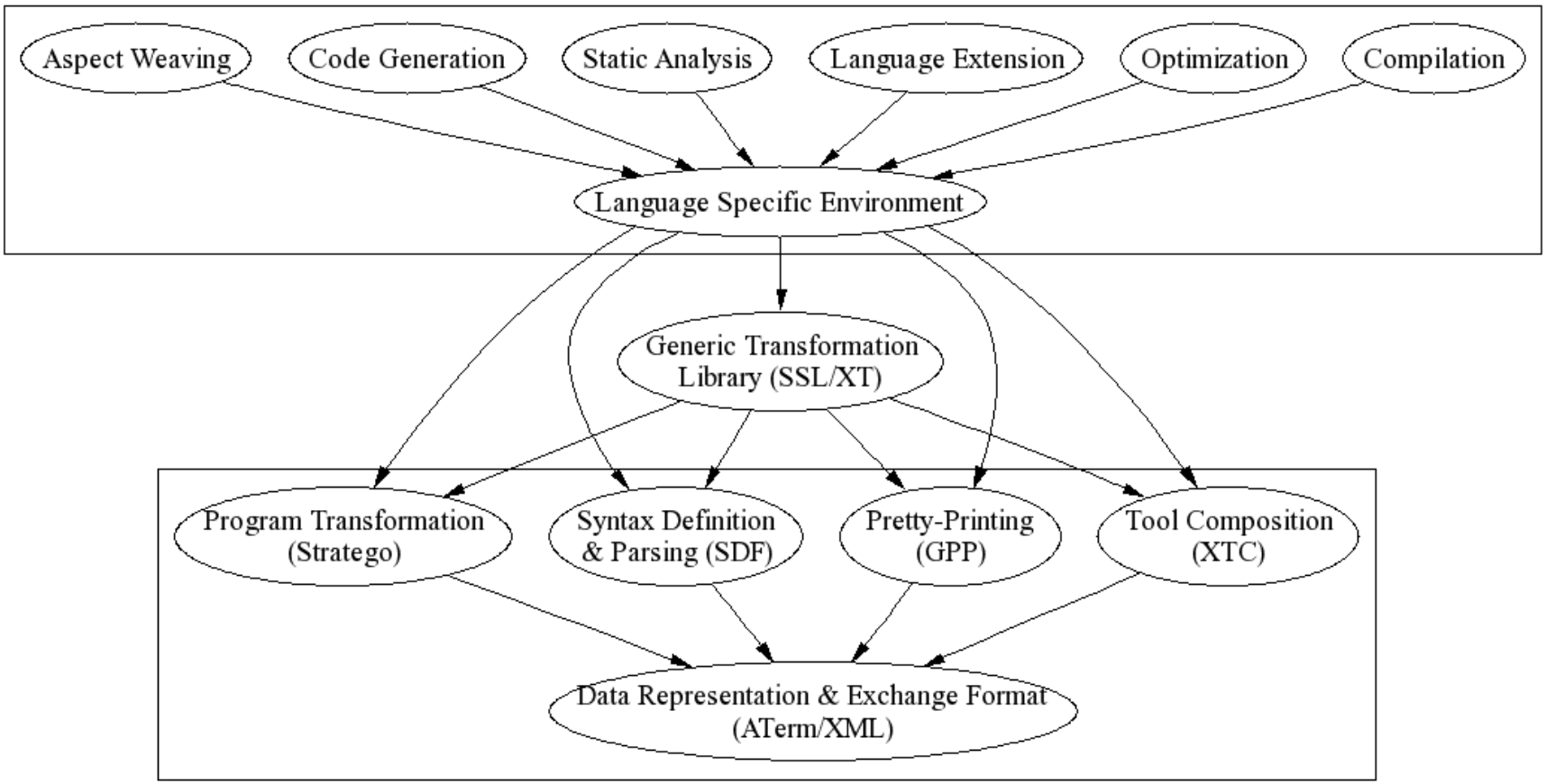
Reusable and Adaptable Strategies for Generative Programming

Martin Bravenboer and Eelco Visser

Center for Software Technology
Institute of Information and Computing Sciences
Utrecht University

October 23, 2004

Stratego/XT: Components for Program Transformation



Producing Generative Programming Tools

Despite the generic infrastructure, creating a generative tool is still too much work. To increase meta-programmer productivity we need:

- Language-specific components
  - front-end with parser, static analysis, ...
- Reusable strategies
  - generic semantic strategies at all levels of granularity
- Adaptable strategies
  - adapt to specific language and/or transformation

Example: data-flow transformations

Constant Propagation

```
prop-const =
  forward-prop(prop-const-transform(prop-const), id, prop-const,
               prop-const-after | ["PropConst"], [], [])

prop-const-transform(recur) =
  EvalFor <+ EvalIf; recur <+ |[ while <recur> do <id> ]|; EvalWhile

prop-const-after =
  try(prop-const-assign <+ PropConst <+ EvalBinOp)

prop-const-assign =
  ? |[ x := e ]|
  ; where( <is-value> e )
  ; rules( PropConst.x : |[ x ]| -> |[ e ]| depends on [(x,x)] )
```

Common-Subexpression Elimination

```
cse =
  forward-prop(fail, id, cse, cse-after | ["CSE"], [], [])

cse-after =
  try(cse-assign <+ CSE)

cse-assign =
  ? |[ x := e ]|
  ; where( <pure-and-not-trivial(|x)> |[ e ]| )
  ; where( get-var-dependencies => xs )
  ; where( innermost-scope-CSE => z )
  ; rules( CSE.z : |[ e ]| -> |[ x ]| depends on xs )
```

Combination of Data-flow Transformations

```
super-opt =
  forward-prop(
    prop-const-transform(super-opt)
    , bvr-before
    , super-opt
    , bvr-after; copy-prop-after; prop-const-after; cse-after
    | ["PropConst", "CopyProp", "CSE"], [], ["RenameVar"]
  )
```

Generic Forward Propagation Strategy

```
forward-prop(transform, before, recur, after | Rs1, Rs2, Rs3) =
  composition-of-traversal-omitted

forward-prop-assign(transform, before, recur, after | Rs) =
  |[ <id> := <recur> ]|
  ; (transform
     <+ before
        ; ?|[ x := e ]|
        ; where( <kill-dynamic-rules(|Rs)>x )
        ; after)

forward-prop-if(transform, before, recur, after | Rs1, Rs2) =
  |[ if <recur> then <id> else <id> ]|
  ; (transform
     <+ before
        ; (|[ if <id> then <recur> else <id> ]|
             /~Rs1\~Rs2/  |[ if <id> then <id> else <recur> ]|)
        ; after)
```

Research Challenges

More semantics-specific strategies without loosing genericity

- High-level declarative specification of front-ends
  - beyond syntax definition
- High-level transformation combinators
  - examples: dynamic rules, regular path queries
- Language-independent strategies
  - example: generic forward propagation, generic constant propagation
- Adaptation mechanisms
  - language binding
  - parameterization is too limited
  - possible directions: aspects, transformation of strategies