Scopes as Types
Hendrik van Antwerpen, Casper Bach Poulsen, Arjen Rouvoet, Eelco Visser
Delft University of Technology

Declarative, executable specifications of type systems are complicated by name binding.
- Different representations for different binding patterns are bad for reuse of concepts, code, and tools.
- Executability introduces algorithmic concerns such as premise ordering and rule splitting.

<table>
<thead>
<tr>
<th>Binding Pattern</th>
<th>Representation</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexical</td>
<td>Ordered environment Name-type list</td>
<td>Top-down environment construction</td>
</tr>
<tr>
<td>Structural Records</td>
<td>Unordered fields Label-type map</td>
<td>Interleaving of name resolution and type checking</td>
</tr>
<tr>
<td>Modules</td>
<td>Module table Name-interface map</td>
<td>Staged module table construction and module body checking</td>
</tr>
</tbody>
</table>

Scope Graph
- Scope graphs represent binding patterns as a graph of scopes and declarations, connected by labeled edges.
- Names are resolved by querying the graph. Query parameters (a regular expression and an order on edge labels) determine visibility and shadowing.

Statix
- Statix is a constraint language to specify type systems with syntax-directed rules.
- Rules specify assertions and queries on the (implicit) scope graph.
- Statix specifications have a declarative semantics, and are executable.

Can we soundly resolve queries in an incomplete scope graph?
- A query result on an intermediate graph is sound if it also holds in the final graph.
- This is true if remaining constraints do not add data that shadows the query result.
- We (over)approximate which labeled edges may be added to the scopes in the intermediate graph.
- This approximation uses static rule information and dynamic information on the remaining constraints.
- A dynamic check ensures that scope graph queries are delayed if an invalidating graph extension may occur.

Executing Statix Specifications
Specifications are executed by rewriting a constraint set and a solution:

- Constraints are solved by unification, building the scope graph, querying the scope graph, and rule-based simplification.
- Intermediate scope graphs may be incomplete, because of remaining scope graph assertions in the constraint set.
- Resolving queries in an incomplete scope graph is essential to support type-dependent name resolution or binding in types.