TYPE-CHECKING MODULES AND IMPORTS USING SCOPE GRAPHS

A Case Study on a Language with Relative, Unordered and Glob Import Semantics

Paul Hübner
p.m.hubner@student.tudelft.nl

Supervisors:
Dr. Casper Bach Poulsen, Aron Zwaan

1. INTRODUCTION

- Type checking using conventional methods such as environments is elegant but often <u>difficult to apply</u> to the real world
- For example due to module systems [3]
- Scope graphs provide a formal definition for type checking
- With them we hope to be able to easily <u>represent real-world programs</u>
- LM is a proof-of-concept language with interesting module/import properties
- MiniStatix is a type-checker using scope graphs
- Its implementation of LM does not always halt, when modules and imports are involved, it "gets stuck" [2]

2. RESEARCH QUESTION

The current MiniStatix representation of LM does not support imports due to its query scheduling [2].

Can scope graphs constructed by a phased Haskell library be used to typecheck a language with relative, unordered and glob imports? How?

3. BACKGROUND (LM)

LM has glob, relative and unordered import semantics [1]. In that regard, it is extremely similar to Rust [1].

```
module A {
  def x = 19
}

module M {
  import A
  def y = x
}
```

Listing 1: Module A is imported in a "glob" fashion, all declarations are visible.

```
module A {
  module B {
    def x = 19
  }
}

module M {
  import B
  import A
  def y = x
}
```

Listing 2: Module B is imported relatively (as opposed to A.B) and the order of imports does not matter (they are unordered).

4. BACKGROUND (SCOPE GRAPHS)

Scope graphs contain nodes for particular scopes, joined by directional edges. The scope graph of Listing 2 is Figure 2.

Edge labels:

- V for variable sinks
- M for module sinks
- P for lexical parent
- I for imports

- When type-checking, x needs to be resolved
- This can be done using <u>querying</u> all sinks along paths that match the RegEx *P*I?V*

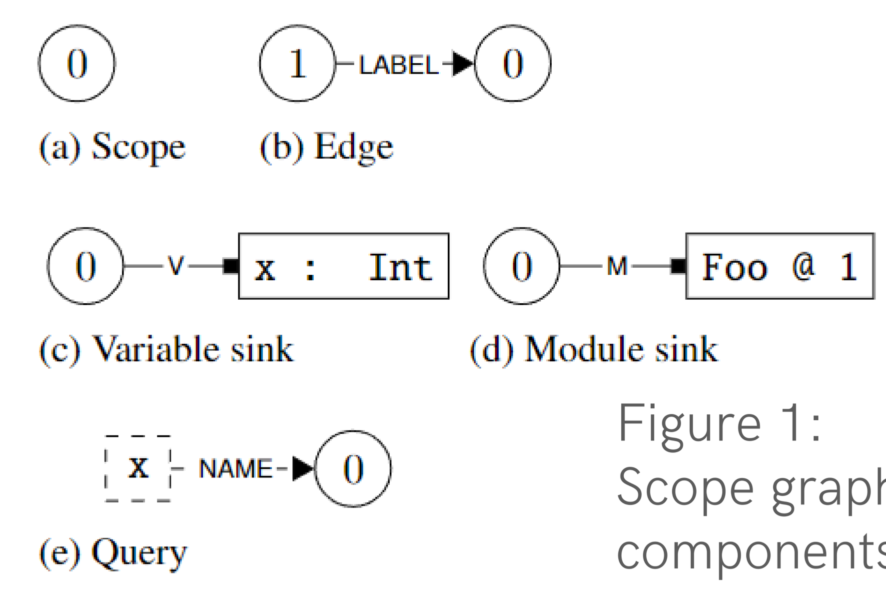

Figure 1: Scope graph components

Core problem: you cannot add an import edge to a scope that has already been queried for import edges (monotonicity violation)!

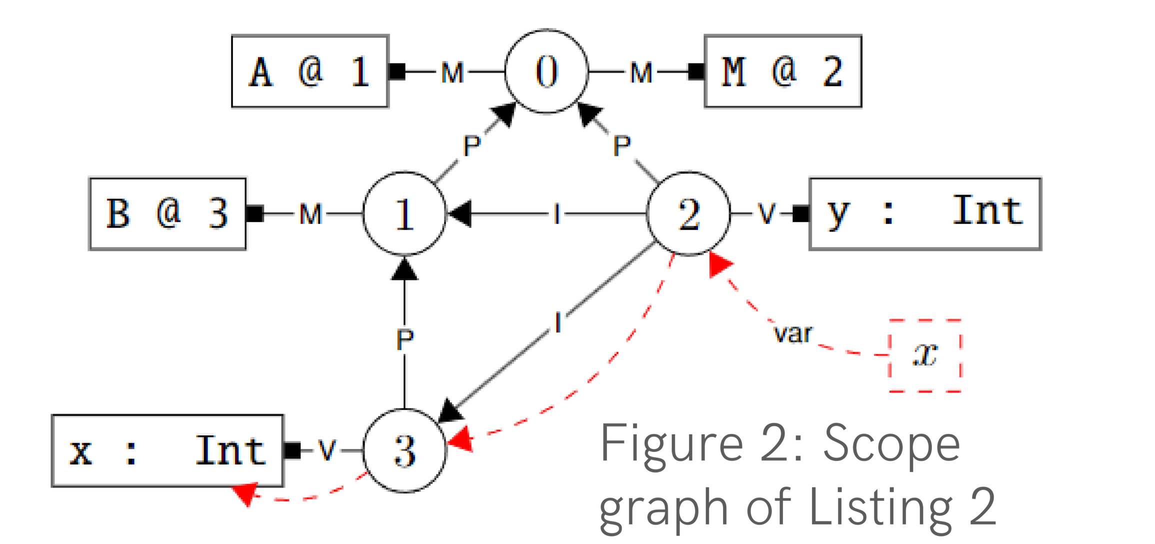

Figure 2: Scope graph of Listing 2

5. CONTRIBUTION

<u>Multiple phases</u> are used to construct the scope graph:

1. Creating the module hierarchy
2. Constructing the module structure scope graph
3. Import resolution (query *P*I?M,* placing I edges)
4. Adding declarations as type variables
5. Type-checking declaration bodies, using inference mostly based on [1]

Algorithm 1 Import resolution algorithm

Require: I, F, m
  while $I \neq \emptyset$ **do**
    $(U, A) \leftarrow$ partition I into unique and shadowing names
    $f \leftarrow$ poll F
    if f is null **then**
      error *"not all imports could be resolved"*
    end if
    $S \leftarrow \emptyset$
    for $u \in U$ **do**
      $r \leftarrow$ query for u from f
      $S \leftarrow S \cup r$
    end for
    $S' \leftarrow \emptyset$
    for $a \in A$ **do**
      $r \leftarrow$ multiple queries for a from all in $\{f\} \cup S$
      $r' \leftarrow$ lowest cost path via label from r or $\emptyset$
      $S' \leftarrow S' \cup r'$
    end for
    $R \leftarrow S \cup S'$
    for $r \in R$ **do**
      place import edge from m to r
    end for
    $F \leftarrow F \cup R$
    $I \leftarrow$ remove modules in R from I
  end while

Algorithm 1: Import resolution algorithm

6. EVALUATION/DISCUSSION

Evaluation using <u>26 test cases</u> based on those in MiniStatix [3]. The results are in Figure 3.

		True behaviour	
		Accept	Reject
Impl.	Accept	20	0
	Reject	1	5

Figure 3: Confusion matrix

New derived scope graph primitives:

- Breadth-first traversal
- Multi-origin querying

Declarativity and feature extensibility:

Less declarable, but much more flexible and extensible than Ministatix.

Reusability:

Languages with relative and glob imports run into similar issues. Reuse for Ruby modules or C++ namespaces.

Limitations:

One test case rejected by ambiguity detection, which is flawed. Similarly, no proof of correctness.

7. CONCLUSION

A five-step stratified approach yields mostly correct behaviour, with the ambiguity checker creating a false negative.

BF-traversal and multi-origin querying were derived as new scope graph primitives in order to facilitate this approach.

Future research recommendations:

- Fix ambiguity checker
- Prove algorithm correctness
- Apply this research to Ruby and C++
- Investigate this approach on transitive imports
- Optimize runtime performance

[1] Hendrik van Antwerpen, Pierre Neron, Andrew Tolmach, Eelco Visser, and Guido Wachsmuth. A constraint language for static semantic analysis based on scope graphs. In *Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, PEPM '16, page 49–60, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Arjen Rouvoet, Hendrik van Antwerpen, Casper Bach Poulsen, Robbert Krebbers, and Eelco Visser. Knowing when to ask: Sound scheduling of name resolution in type checkers derived from declarative specifications. *Proc. ACM Program. Lang., 4* (OOPSLA), November 2020.

[3] Aron Zwaan and Hendrik van Antwerpen. Scope Graphs: The Story so Far. In Ralf Lämmel, Peter D. Mosses, and Friedrich Steimann, editors, *Eelco Visser Commemorative Symposium (EVCS 2023)*, volume 109 of *Open Access Series in Informatics (OASIcs)*, pages 32:1–32:13, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz- Zentrum für Informatik.