Scope trees, scope graphs, and reference attribute grammars for name resolution in (domain-specific) languages

... work in progress ...

Luke Bessant , Dawn Michaelson, and Eric Van Wyk

Department of Computer Science & Engineering
University of Minnesota

WG 2.11 Meeting, Delft, April 3-4, 2023

Network virtualization, software-defined networks

Network virtualization, software-defined networks

- Surprisingly, 5G is not 100% meaningless hype!

Network virtualization, software-defined networks

- Surprisingly, 5G is not 100% meaningless hype!
- It is more than just an attempt to sell new phones.
- Networking is not just hardware boxes anymore, much is “software defined.”
- There is a need for dynamic adaptation, easy configuration, security, better performance, etc.
- Network applications manage the network — security/authentication, traffic management (dynamic scaling to match traffic demands), etc.
- Interest in private networks in industrial applications.

A domain-specific language for network functions

- Networks are just connected independent devices.
- DSL inspired by the actor model
- Actors respond to messages by
  - sending more messages
  - modifying local state

  There is no global shared state.
- The aim
  - DSL is a target for program synthesis
  - Analysis to provide performance guarantees
  - Understand scaling in/out as program transformations

A firewall example - message types

```
message FirewallControl {
   bits<2> action;
   bits<128> address;
}

message FirewallInfo {
   int droppedPackets;
}

/* Built in message types: IPv4, IPv6 */
```

A firewall example - actor header and state

```
actor Firewall :
  IPv6|FirewallControl ->
      Drop: IPv6,
      Forward: IPv6,
      Controller: FirewallInfo
{
```

A firewall example - actor header and state

```
actor Firewall :
  IPv6|FirewallControl ->
      Drop: IPv6,
      Forward: IPv6,
      Controller: FirewallInfo
{

  //persistent state held by the actor
  state {
     //table holds 1 for drop, 0 for send
     table<bits<128>, bits<1>> dropTable default 0;
     int droppedPackets;
  }
```

A firewall example - actor initialization and message dispatch

```
init () {
    droppedPackets = 0;
}
```

A firewall example - actor initialization and message dispatch

```
init () {
   droppedPackets = 0;
}

dispatch (msg) {
   match msg with
   | IPv6 { _ } -> data(msg);
   | FirewallControl { _ } -> control(msg);
}
```

A firewall example - actor actions

```
// a regular packet
action data (IPv6 msg) {
    if (dropTable[msg.srcIP] == 1) {
        droppedPackets = droppedPackets + 1;
        send msg to Drop;
    }
    else {
        send msg to Forward;
    }
}
```

A firewall example - actor actions

```
// a SDN controller message
action control (FirewallControl msg) {
    if (msg.action == 0){ //let this address through
        remove msg.address from dropTable;
    }
    else if (msg.action == 1){ //drop packets from this address
        dropTable[msg.address] = 1;
    }
    else { //controller wants information
        send ( FirewallInfo {droppedPackets = droppedPackets;} )
          to Controller;
      }
  }
```

Prototyping the DSL

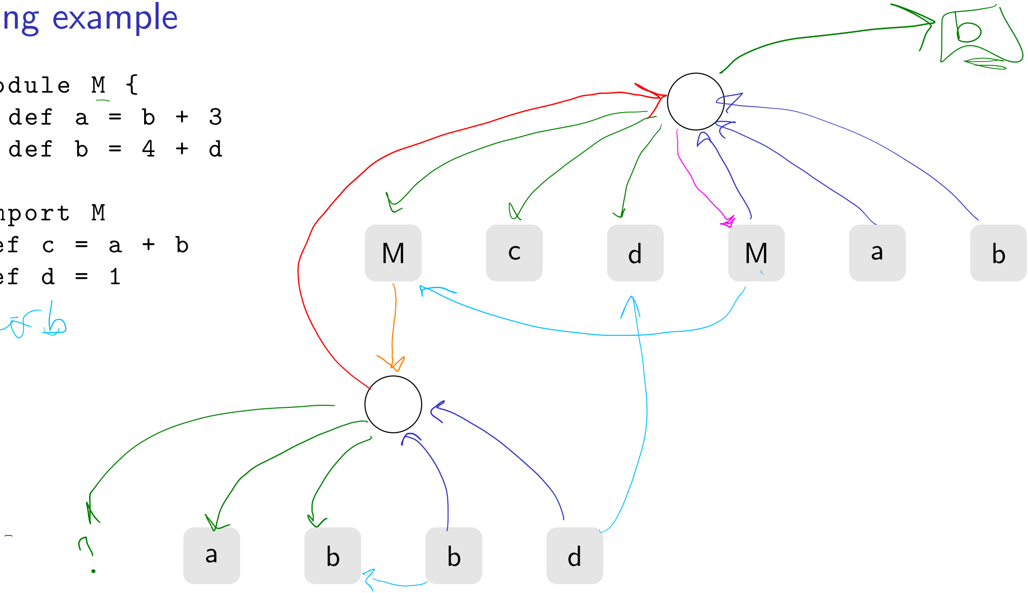
- We want domain user involvement as soon as possible.
- One way to do this is use language-independent formalisms like Eelco's scope graphs for
  - name resolution in the compiler
  - and also for IDE support.

  The intent for scope graphs is to be the "BNF" for name resolution.
- Nodes for scopes, name declarations, and references.
  Edges indicating scoping structure.
  A similar shape to the AST, but limited to information useful for name resolution.
- We did not have an implementation of scope graphs so this was a good excuse to work on one.
- It is work in progress, but this week is a good time for topics related to Eelco's work.

Working example

```
module M {
  def a = b + 3
  def b = 4 + d
}
import M
def c = a + b
def d = 1
```

def b



Reference/Remote AGs

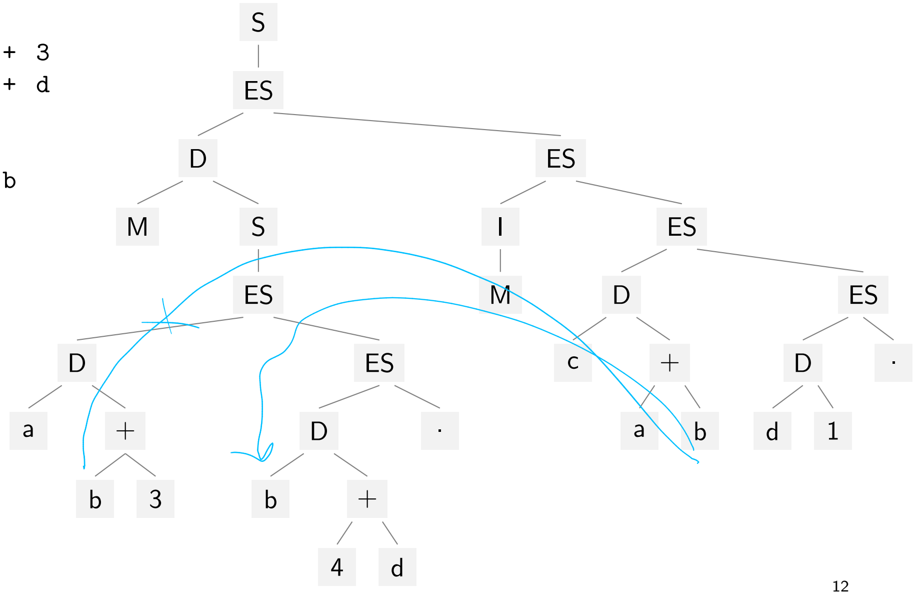
- à la Görel Hedin and John Boyland
- Syntax trees with extra edges, making them into graphs.

  These edges are attributes whose value are references / pointers to remote nodes somewhere in the syntax tree.
- Often in RAGs this is to create edges from name reference to declarations.

  But the resolution is ad-hoc and done on a per-language basis.

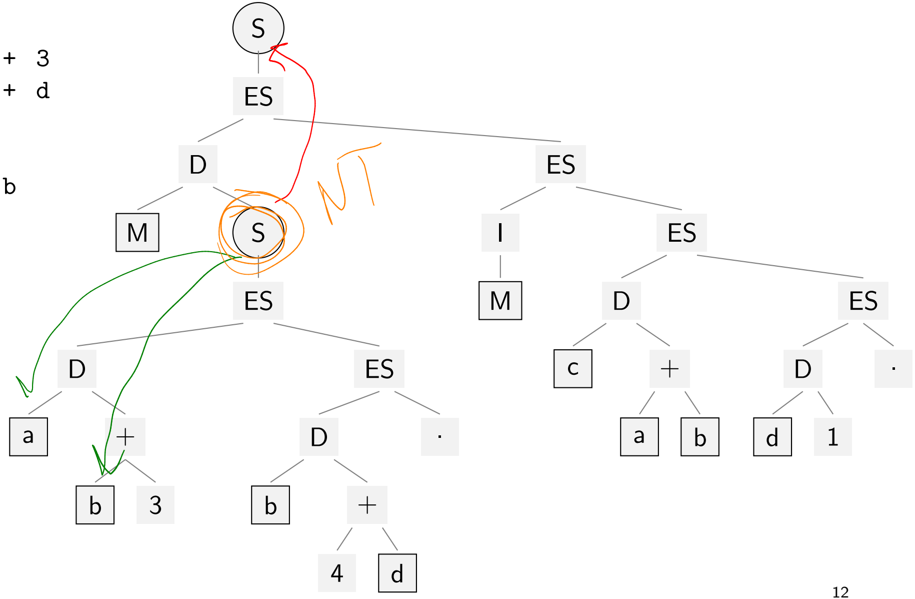
Abstract Syntax Tree

```
module M {
  def a = b + 3
  def b = 4 + d
}
import M
def c = a + b
def d = 1
```



Abstract Syntax Tree

```
module M {
  def a = b + 3
  def b = 4 + d
}
import M
def c = a + b
def d = 1
```



Scope Trees - a RAGs implementation of Visser's Scope Graphs

- We can overlay a scope graph edges over the AST, but we cannot add the generic name resolution computations to AST productions.

  Thus, resolution would still be ad-hoc and language specific.

Scope Trees - a RAGs implementation of Visser's Scope Graphs

- We can overlay a scope graph edges over the AST, but we cannot add the generic name resolution computations to AST productions.

  Thus, resolution would still be ad-hoc and language specific.

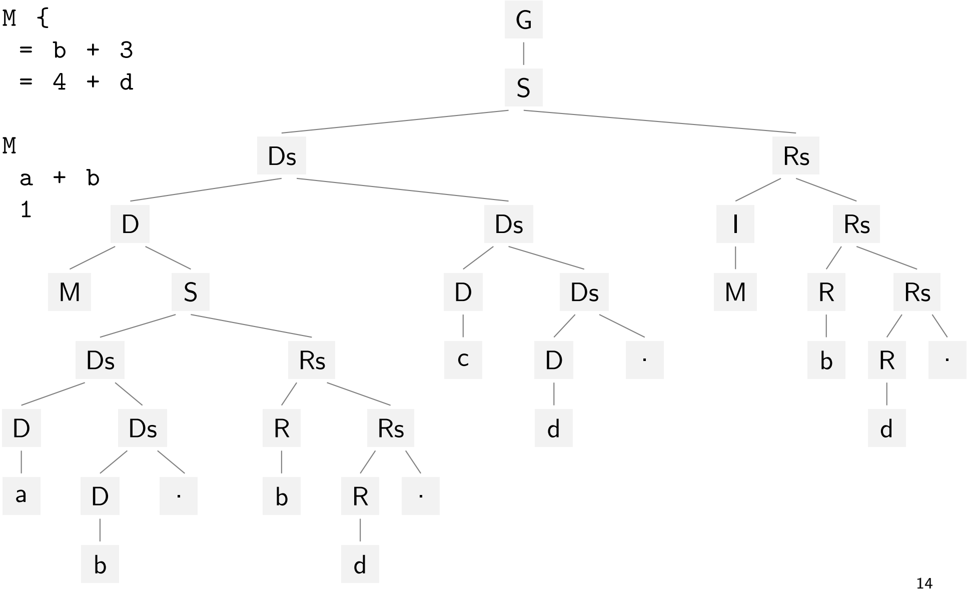
- Instead, a generic scope tree is constructed by the object language specification.

  When the reference attributes are evaluated on this tree, it becomes a scope graph.

- Also create links between the AST nodes and corresponding scope tree nodes.

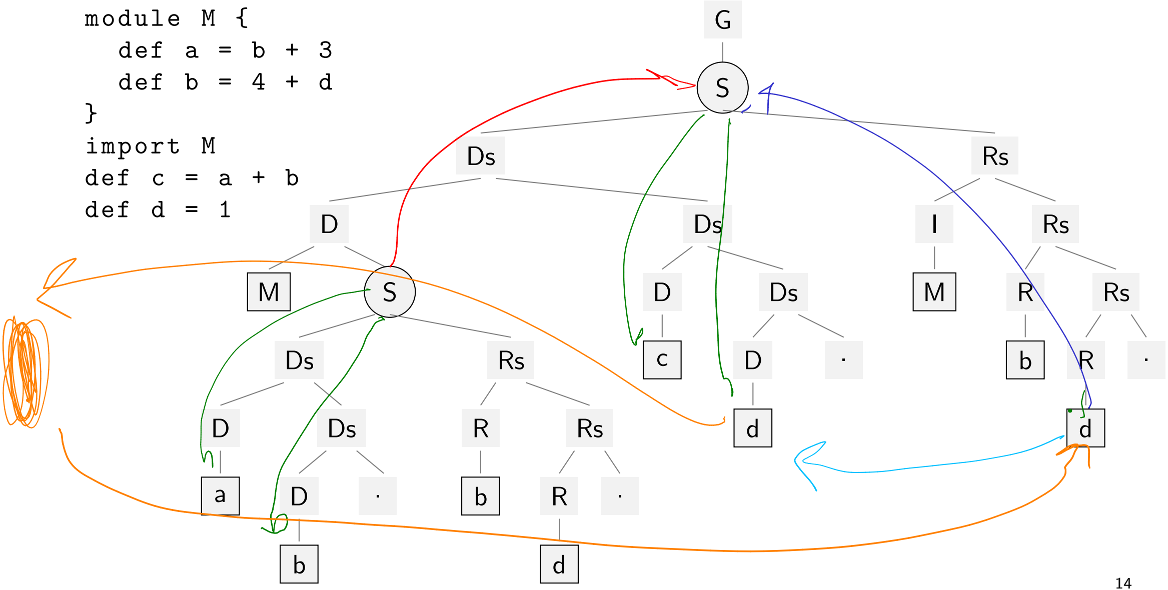
Scope Tree

```
module M {
  def a = b + 3
  def b = 4 + d
}
import M
def c = a + b
def d = 1
```



Scope Tree

```
module M {
  def a = b + 3
  def b = 4 + d
}
import M
def c = a + b
def d = 1
```



Scope trees as a Silver library

- A work in progress.
- Various ways to create trees and links between the AST and the Scope Tree.
- Working on more applications to flesh out the details.

Scope trees as a Silver library

- A work in progress.
- Various ways to create trees and links between the AST and the Scope Tree.
- Working on more applications to flesh out the details.
- Eelco was right - scope graphs are useful things.

Thanks

We thank the NSF for supporting this work under award #2123987.