Specializing Scope Graph Resolution Queries

Aron Zwaan

Dec 7, 2022

Overview: Synthesizing Type Checkers

- Implementing type checkers is hard: generate using Statix

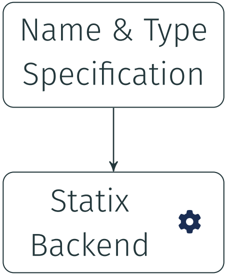
Overview: Synthesizing Type Checkers

* Implementing type checkers is hard: generate using Statix

Name & Type Specification

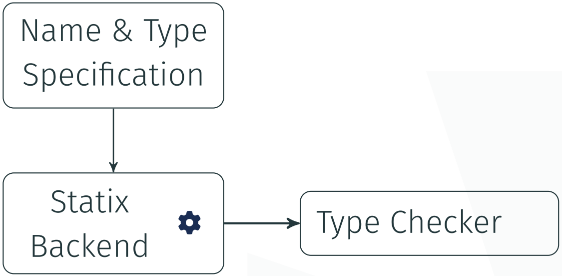
Overview: Synthesizing Type Checkers

- Implementing type checkers is hard: generate using Statix



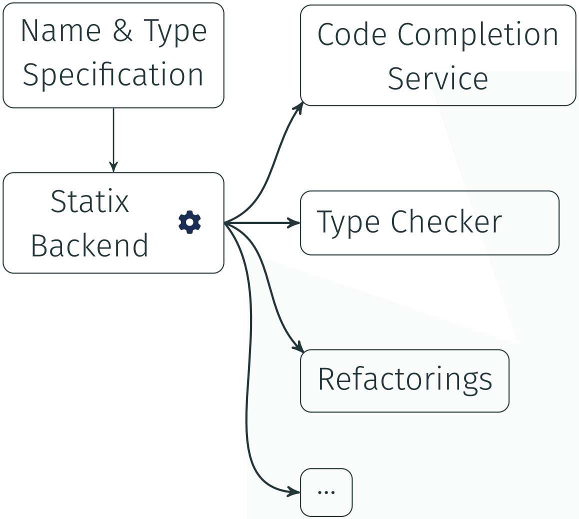
Overview: Synthesizing Type Checkers

* Implementing type checkers is hard: generate using Statix



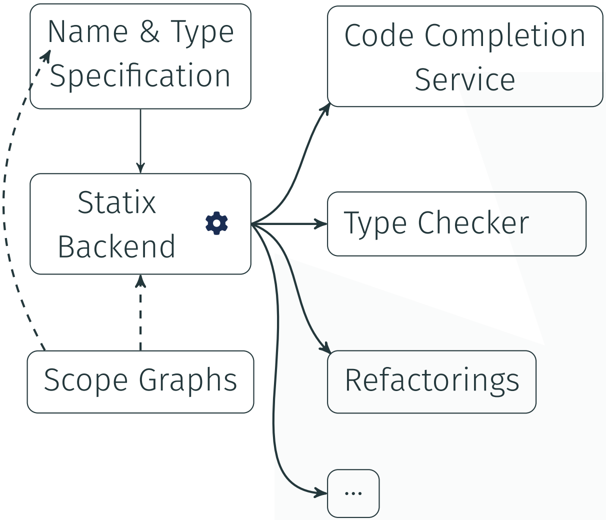
Overview: Synthesizing Type Checkers

- Implementing type checkers is hard: generate using Statix



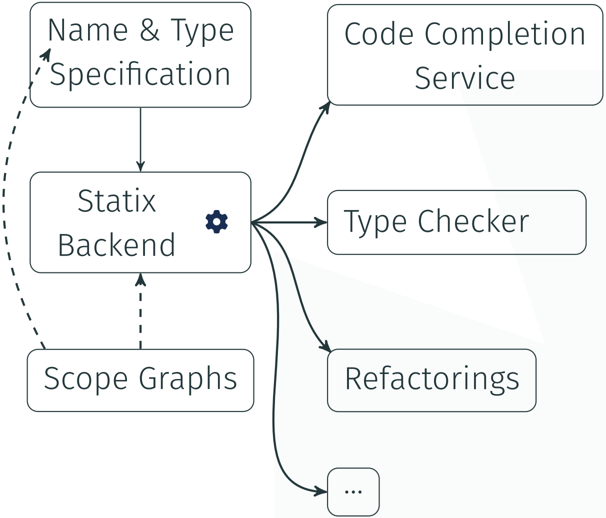
Overview: Synthesizing Type Checkers

- Implementing type checkers is hard: generate using Statix



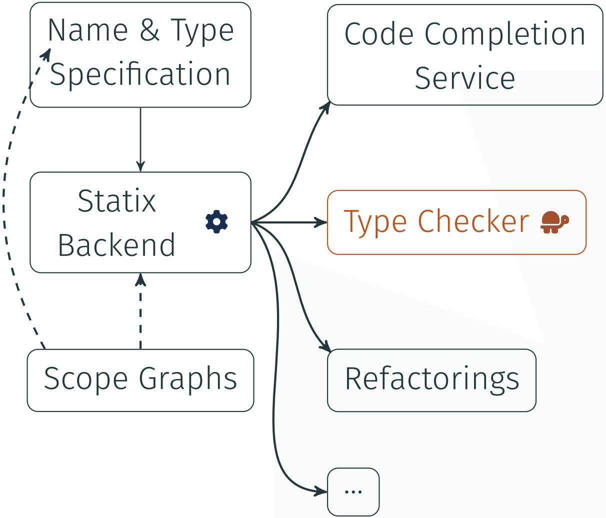
Overview: Synthesizing Type Checkers

- Implementing type checkers is hard: generate using Statix
- Advantages of generating
  1. Easy
  2. Consistent
  3. Allows reasoning



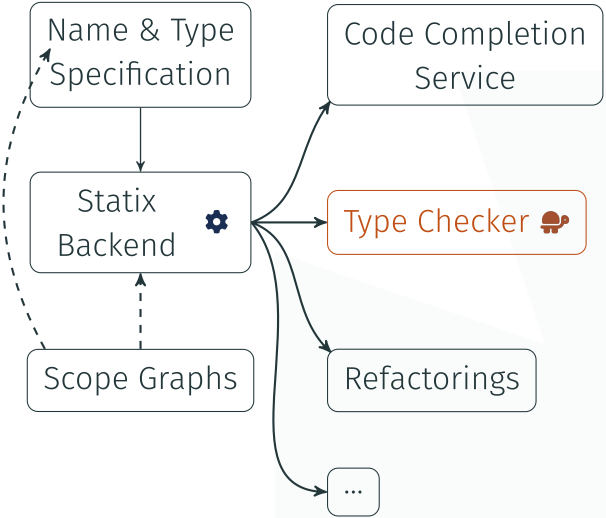
Overview: Synthesizing Type Checkers

- Implementing type checkers is hard: generate using Statix
- Advantages of generating
  1. Easy
  2. Consistent
  3. Allows reasoning
- Problem: 50% overhead in name resolution algorithm



Overview: Synthesizing Type Checkers

* Implementing type checkers is hard: generate using Statix
* Advantages of generating
  1. Easy
  2. Consistent
  3. Allows reasoning
* Problem: 50% overhead in name resolution algorithm
* Solution: partial evaluation



Specializing Scope Graph Resolution Queries

Scope Graphs

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```

Scope Graphs

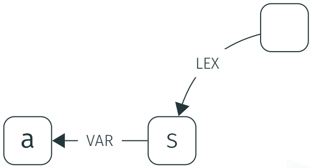
```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```

Scope Graphs

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```

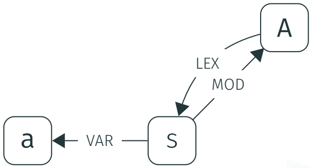
Scope Graphs

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



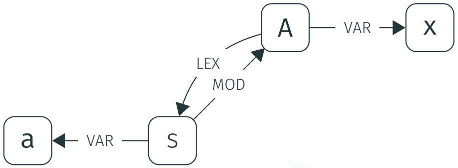
Scope Graphs

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



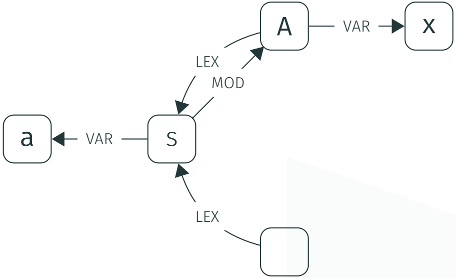
Scope Graphs

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



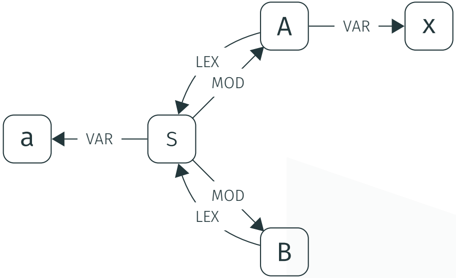
Scope Graphs

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



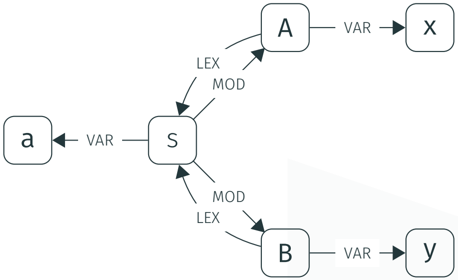
Scope Graphs

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



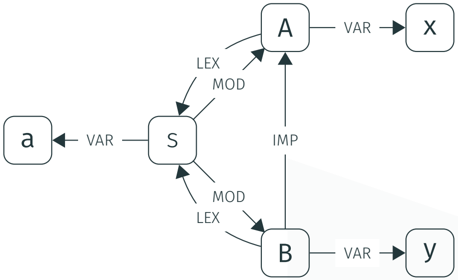
Scope Graphs

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



Scope Graphs

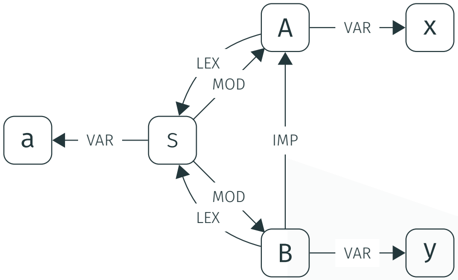
```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



Specializing Scope Graph Resolution Queries

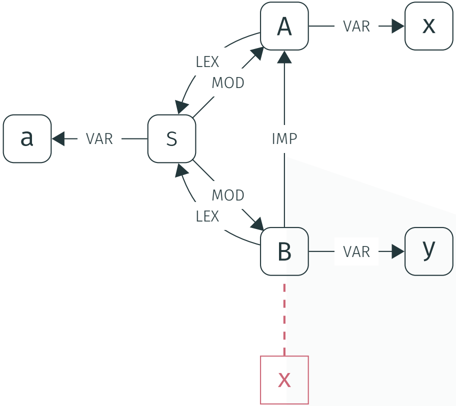
Name Resolution in Scope Graphs: Queries

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



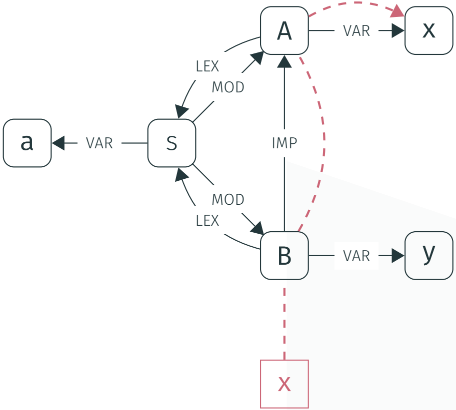
Name Resolution in Scope Graphs: Queries

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



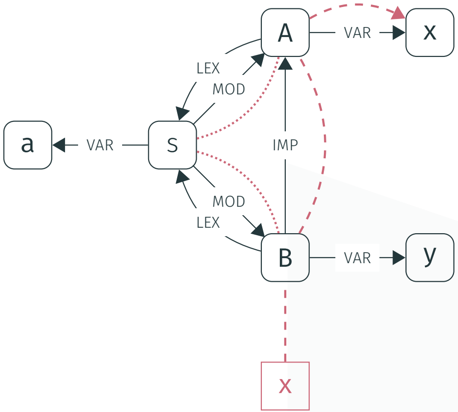
Name Resolution in Scope Graphs: Queries

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



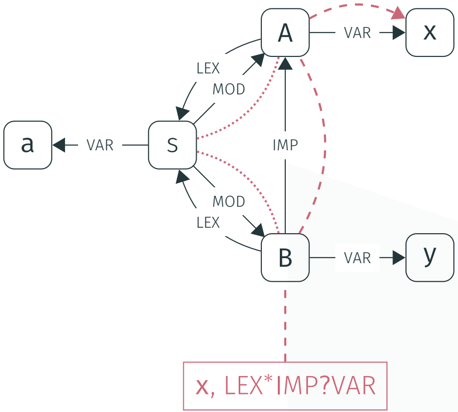
Name Resolution in Scope Graphs: Queries

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



Name Resolution in Scope Graphs: Queries

```
def a = 42;
module A {
  def x = 6
}
module B {
  import A;
  def y = 7 * x
}
```



Specializing Scope Graph Resolution Queries

Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂_ℓ ℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂_ℓ ℛ, n)
  return E
}
```

Resolution (simplified)

```
Resolve(𝒢,s,ℛ,n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂_ℓℛ ≠ ∅
      for s′ in getEdges(𝒢,s,ℓ)
        E += Resolve(𝒢,s′,∂_ℓℛ,n)
  return E
}
```

Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

Resolution (simplified)

```
Resolve(𝒢,s,ℛ,n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢,s,ℓ)
        E += Resolve(𝒢,s′,∂ℓℛ,n)
  return E
}
```

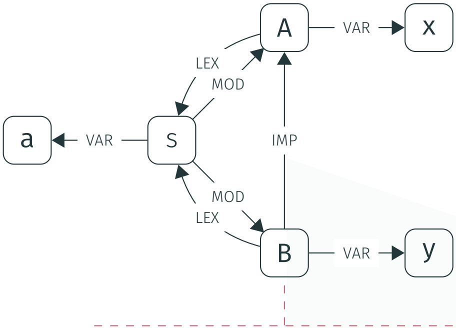
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if [∂ℓℛ] ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, [∂ℓℛ], n)
  return E
}
```

- $\partial_\ell \mathcal{R} = \mathcal{R}' \triangleq \ell\omega \in \mathcal{R} \iff \omega \in \mathcal{R}'$
- Examples
  - $\partial_{L_1} L_1 L_2 = L_2$
  - $\partial_{L_2} L_1 ? L_2^+ = L_2^*$
  - $\partial_{L_3} L_1 ? L_2^+ = \varnothing$

Resolution (simplified)

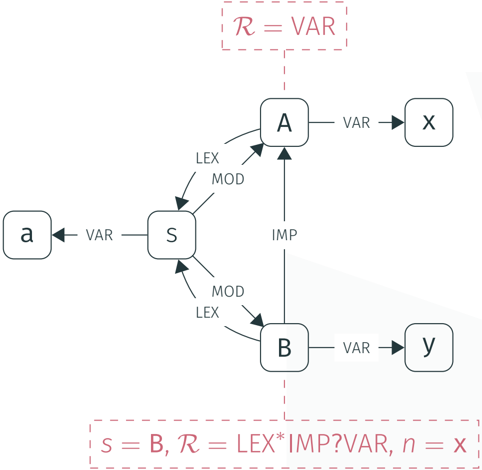
```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



$s = \text{B}, \mathcal{R} = \text{LEX}^*\text{IMP?VAR}, n = \text{x}$

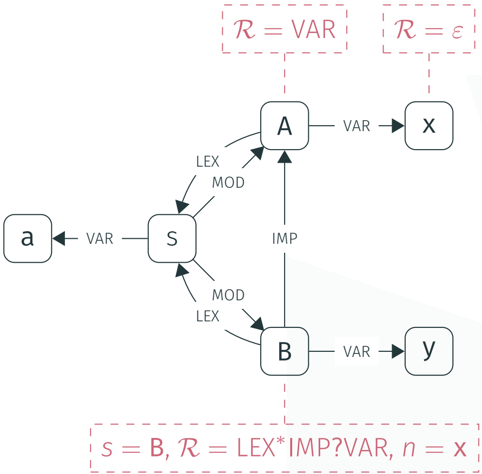
Resolution (simplified)

```
Resolve(𝒢,s,ℛ,n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢,s,ℓ)
        E += Resolve(𝒢,s′,∂ℓℛ,n)
  return E
}
```



Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

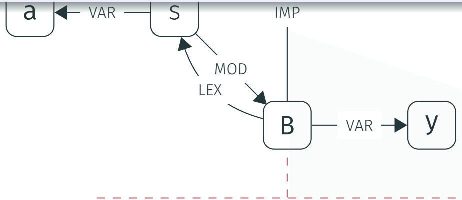


Resolution (simplified)

$\mathcal{R} = \mathsf{VAR}$ $\mathcal{R} = \varepsilon$

Profiling: 12.5% overhead for computing derivatives.

`if` $\partial_\ell \mathcal{R} \neq \varnothing$
  `for` s' `in getEdges(`$\mathcal{G}, s, \ell$`)`
    `E += Resolve(`$\mathcal{G}, s', \partial_\ell \mathcal{R}, n$`)`
`return E`
`}`



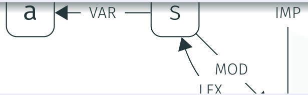
$s = \mathsf{B}, \mathcal{R} = \mathsf{LEX}^*\mathsf{IMP}?\mathsf{VAR}, n = \mathsf{x}$

Resolution (simplified)

$\mathcal{R} = \mathsf{VAR}$ $\mathcal{R} = \varepsilon$

Profiling: 12.5% overhead for computing derivatives.

if $\partial_\ell \mathcal{R} \neq \varnothing$
  for s' in getEdges($\mathcal{G}, s, \ell$)
    E += Resolve($\mathcal{G}, s', \partial_\ell \mathcal{R}, n$)



$\mathcal{R}$ known statically: specialize `Resolve`.

$s = \mathsf{B}$, $\mathcal{R} = \mathsf{LEX^*IMP?VAR}$, $n = \mathsf{x}$

Specializing Scope Graph Resolution Queries

Specialization (simplified)

```
Resolve(𝒢, s, ε, n) {
  E := ∅
  if ϵ ∈ ε & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓε ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓε, n)
  return E
}
```

```
Resolve_ε(𝒢, s, n) {

}
```

Specialization (simplified)

```
Resolve(𝒢, s, ε, n) {
  E := ∅
  if ϵ ∈ ε & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓε ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓε, n)
  return E
}
```

```
Resolve_ε(𝒢, s, n) {

}
```

Specialization (simplified)

```
Resolve(𝒢, s, ε, n) {
  E := ∅
  if ϵ ∈ ε & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓε ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓε, n)
  return E
}
```

```
Resolve_ε(𝒢, s, n) {
  E := ∅

}
```

Specialization (simplified)

```
Resolve(𝒢, s, ε, n) {
  E := ∅
  if ϵ ∈ ε & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓε ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓε, n)
  return E
}
```

```
Resolve_ε(𝒢, s, n) {
  E := ∅



}
```

Specialization (simplified)

```
Resolve(𝒢, s, ε, n) {
  E := ∅
  if ϵ ∈ ε & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓε ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓε, n)
  return E
}
```

```
Resolve_ε(𝒢, s, n) {
  E := ∅
  if s = n
    E += { s }

}
```

Specialization (simplified)

```
Resolve(𝒢, s, ε, n) {
  E := ∅
  if ϵ ∈ ε & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂_ℓ ε ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂_ℓ ε, n)
  return E
}
```

```
Resolve_ε(𝒢, s, n) {
  E := ∅
  if s = n
    E += { s }

}
```

Specialization (simplified)

```
Resolve(𝒢, s, ε, n) {
  E := ∅
  if ϵ ∈ ε & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓε ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓε, n)
  return E
}
```

```
Resolveε(𝒢, s, n) {
  E := ∅
  if s = n
    E += { s }

}
```

Specialization (simplified)

```
Resolve(𝒢, s, ε, n) {
  E := ∅
  if ϵ ∈ ε & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓε ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓε, n)
  return E
}
```

```
Resolve_ε(𝒢, s, n) {
  E := ∅
  if s = n
    E += { s }
  return E
}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {

}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {

}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {
  E := ∅

}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {
  E := ∅
}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {
  E := ∅
}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ                                  ℓ = VAR
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {
  E := ∅
  for s′ ∈ getEdges(𝒢, s, VAR)
    E += Resolve_ε(𝒢, s′, n)
}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ                                    ℓ = LEX
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {
  E := ∅
  for s′ ∈ getEdges(𝒢, s, VAR)
    E += Resolve_ε(𝒢, s′, n)
  for s′ ∈ getEdges(𝒢, s, LEX)
    E += Resolve_LEX*IMP?VAR(𝒢, s′, n)
}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ                                    ℓ = IMP
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {
  E := ∅
  for s′ ∈ getEdges(𝒢, s, VAR)
    E += Resolve_ε(𝒢, s′, n)
  for s′ ∈ getEdges(𝒢, s, LEX)
    E += Resolve_LEX*IMP?VAR(𝒢, s′, n)
  for s′ ∈ getEdges(𝒢, s, IMP)
    E += Resolve_VAR(𝒢, s′, n)

}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ                                   ℓ = MOD
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {
  E := ∅
  for s′ ∈ getEdges(𝒢, s, VAR)
    E += Resolve_ε(𝒢, s′, n)
  for s′ ∈ getEdges(𝒢, s, LEX)
    E += Resolve_LEX*IMP?VAR(𝒢, s′, n)
  for s′ ∈ getEdges(𝒢, s, IMP)
    E += Resolve_VAR(𝒢, s′, n)

}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓ LEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓ LEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {
  E := ∅
  for s′ ∈ getEdges(𝒢, s, VAR)
    E += Resolve_ε(𝒢, s′, n)
  for s′ ∈ getEdges(𝒢, s, LEX)
    E += Resolve_LEX*IMP?VAR(𝒢, s′, n)
  for s′ ∈ getEdges(𝒢, s, IMP)
    E += Resolve_VAR(𝒢, s′, n)

}
```

Specialization (simplified)

```
Resolve(𝒢, s, LEX*IMP?VAR, n) {
  E := ∅
  if ϵ ∈ LEX*IMP?VAR & s = n
    E += { s }
  for ℓ ∈ ℒ
    if ∂ℓLEX*IMP?VAR ≠ ∅
      for s′ ∈ getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓLEX*IMP?VAR, n)
  return E
}
```

```
Resolve_LEX*IMP?VAR(𝒢, s, n) {
  E := ∅
  for s′ ∈ getEdges(𝒢, s, VAR)
    E += Resolve_ε(𝒢, s′, n)
  for s′ ∈ getEdges(𝒢, s, LEX)
    E += Resolve_LEX*IMP?VAR(𝒢, s′, n)
  for s′ ∈ getEdges(𝒢, s, IMP)
    E += Resolve_VAR(𝒢, s′, n)
  return E
}
```

Summary

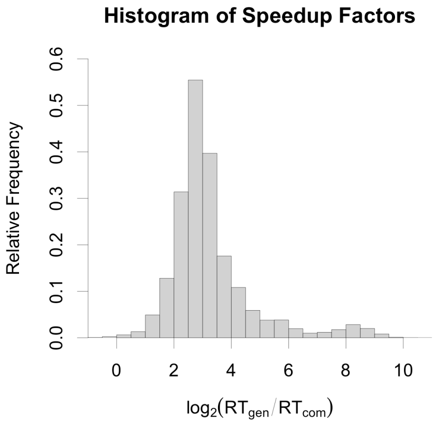
Also in the paper

- Encoding shadowing using partial order on labels
- Resolution algorithm *with shadowing*
- Shadowing: 35% overhead
- Specializing queries shadowing
- Optimizations on IR

Evaluation: Setup

- Java Specification
- Apache Commons CSV, IO and Lang3 projects
- Micro-benchmarks: Speedup of individual queries (CSV)
- Macro-benchmarks: Speedup of type checkers

Evaluation: Results



Project	#Queries	$RT_{gen}(s)$	$RT_{com}(s)$	Speedup
CSV	14328	7.3	4.5	39%
IO	73843	19	12	38%
Lang3	288883	88	46	48%

Bigger Picture

* What made this work so well?
  1. Complex algorithm to interpret language construct (`Resolve`)
  2. Statically known parameters
* Common for more declarative languages

Bigger Picture

* What made this work so well?
  1. Complex algorithm to interpret language construct (`Resolve`)
  2. Statically known parameters
* Common for more declarative languages

Suggests specialization is especially powerful for declarative languages.

Conclusion

Conclusion

Scope graphs allow declarative but executable specification of name resolution.

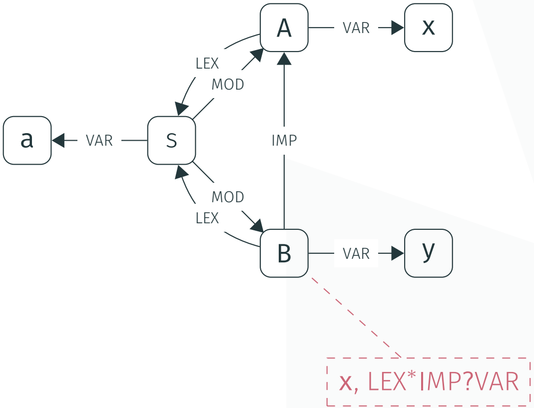
Conclusion

Scope graphs allow declarative but executable specification of name resolution.

Specialization especially speeds up interpreters of declarative languages.

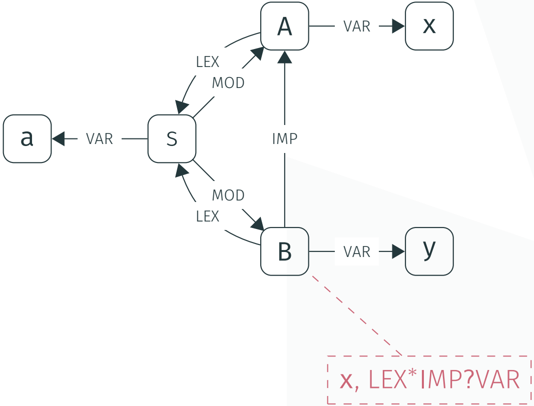
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



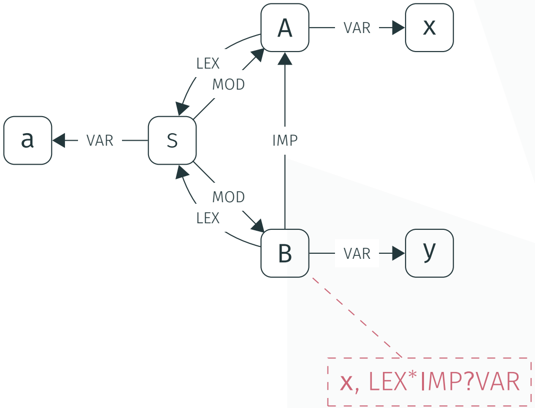
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂_ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂_ℓℛ, n)
  return E
}
```



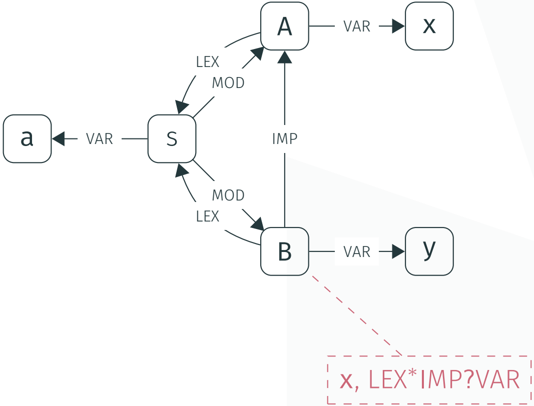
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



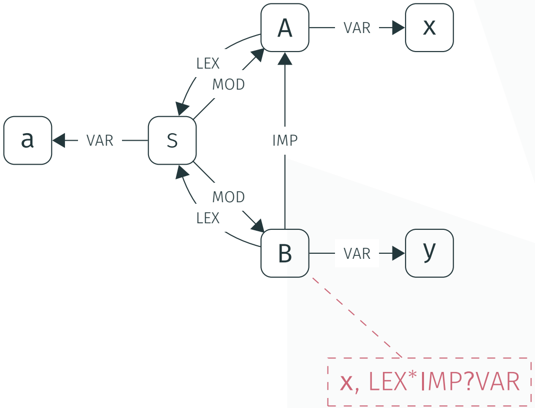
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



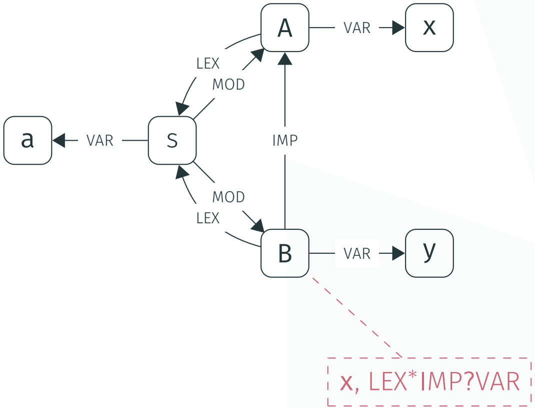
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



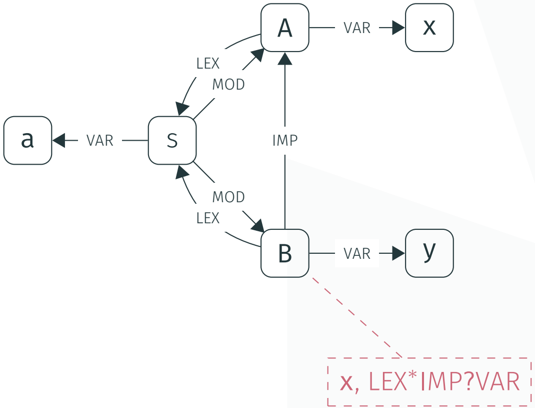
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂_ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂_ℓℛ, n)
  return E
}
```



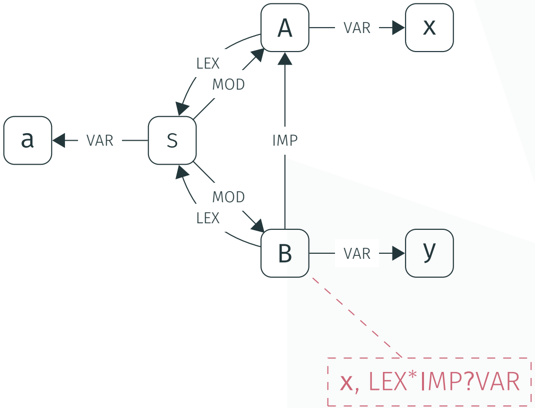
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂_ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂_ℓℛ, n)
  return E
}
```



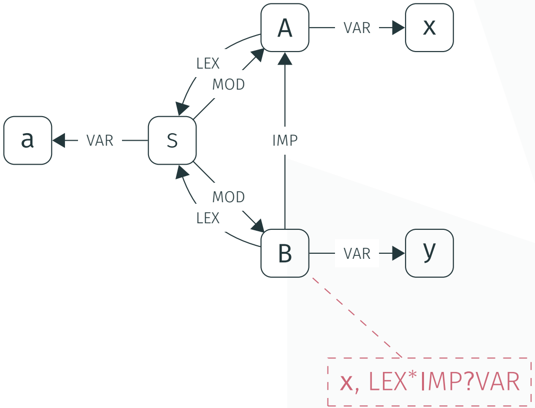
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



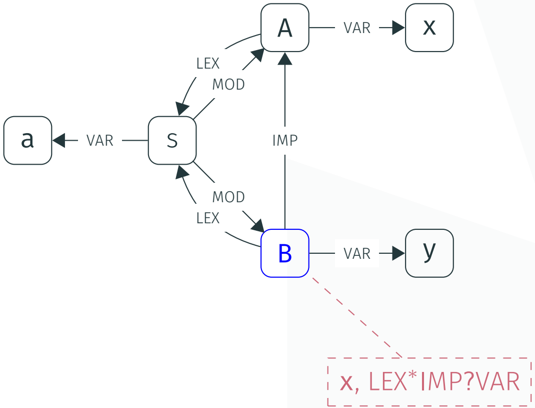
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

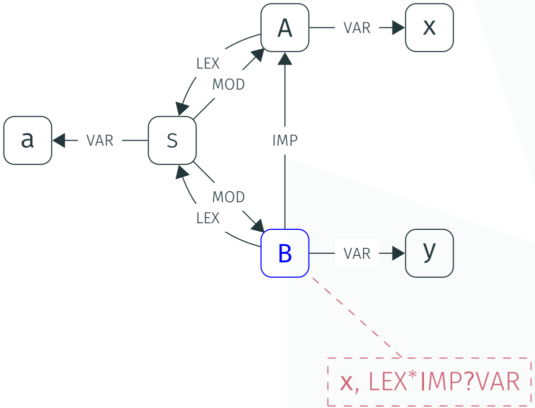


Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂_ℓ ℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂_ℓ ℛ, n)
  return E
}
```

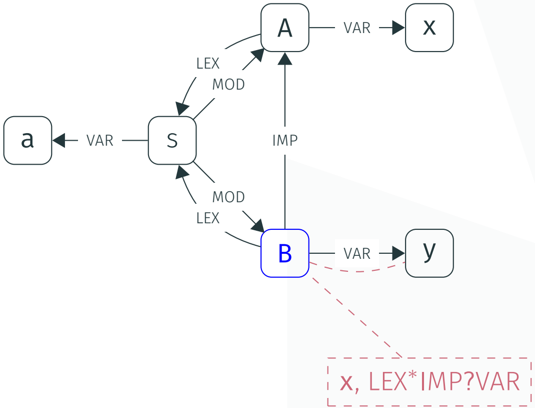
$$\partial_\ell \mathcal{R} = \mathcal{R}' \triangleq \ell\omega \in \mathcal{R} \iff \omega \in \mathcal{R}'$$

E.g.: $\partial_{L_1} L_1?L_2^* = L_2^*$



Resolution (simplified)

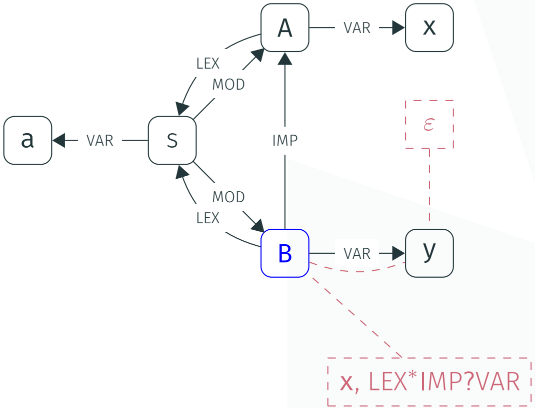
```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ                                ℓ = VAR
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



Resolution (simplified)

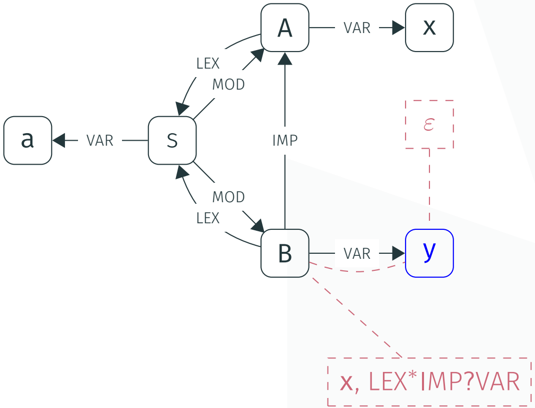
```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

$\ell = \text{VAR}$



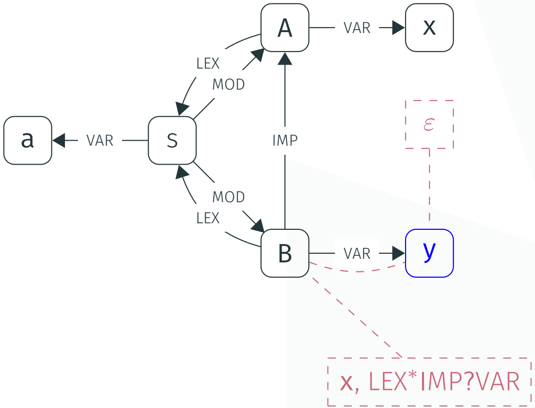
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



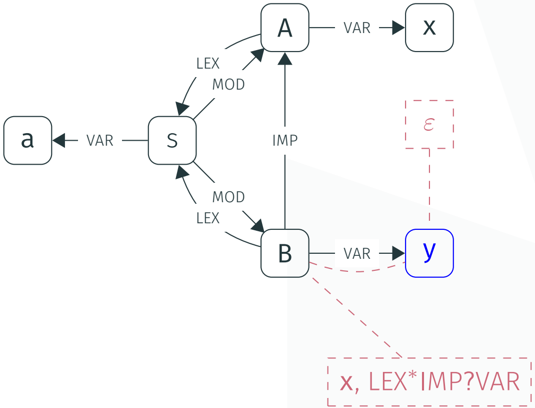
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



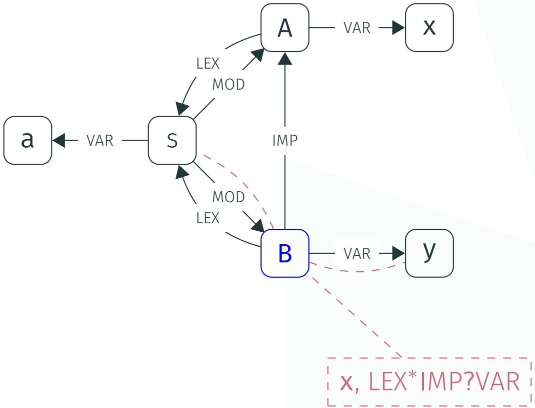
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E      E = ∅
}
```



Resolution (simplified)

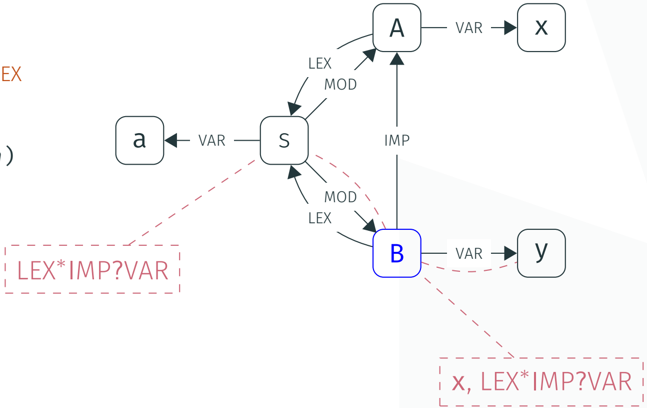
```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ                                ℓ = LEX
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



Resolution (simplified)

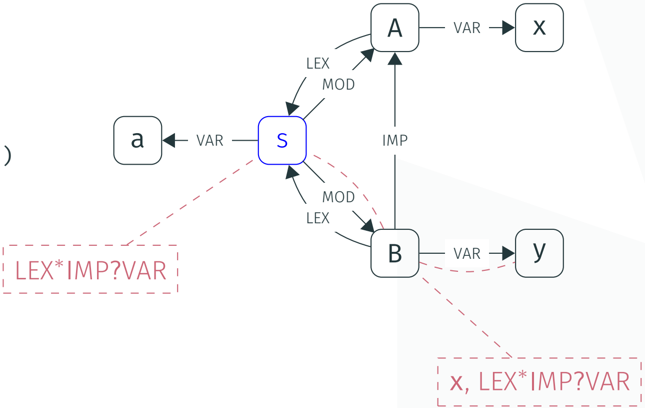
```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

$\ell = \text{LEX}$



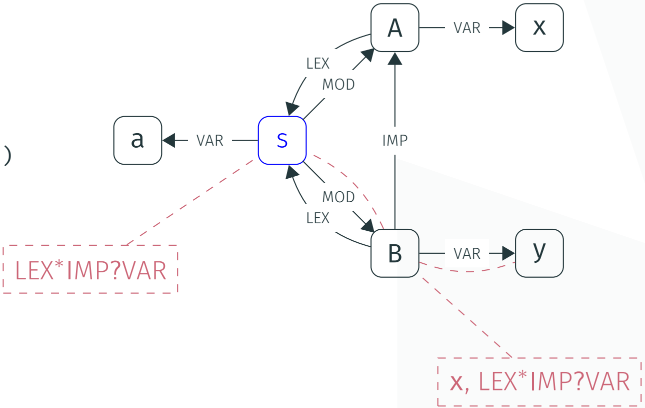
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



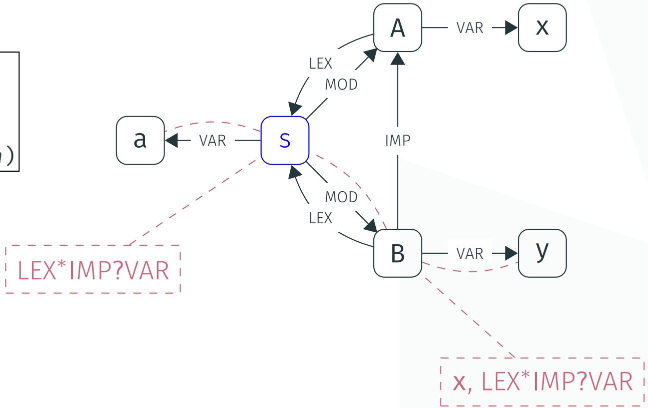
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



Resolution (simplified)

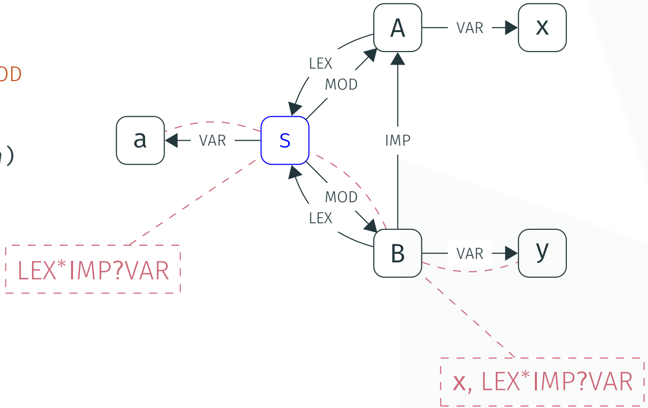
```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂_ℓ ℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂_ℓ ℛ, n)
  return E
}
```

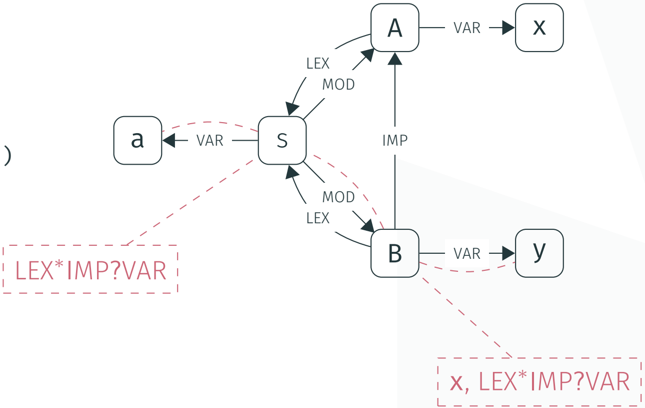
$\ell = \text{MOD}$



Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

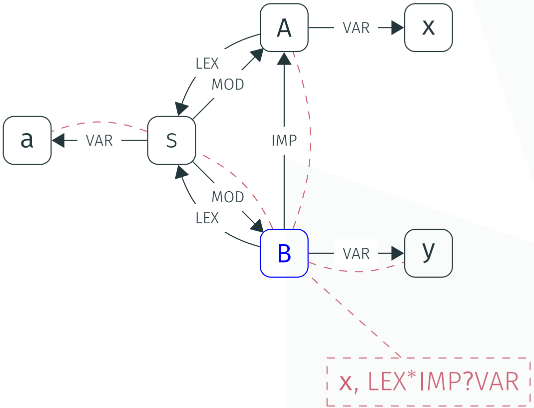
$E = \emptyset$



Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

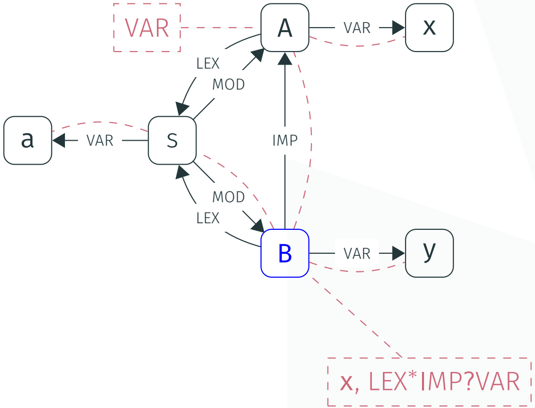
$\ell = \text{IMP}$



Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, [∂ℓℛ], n)
  return E
}
```

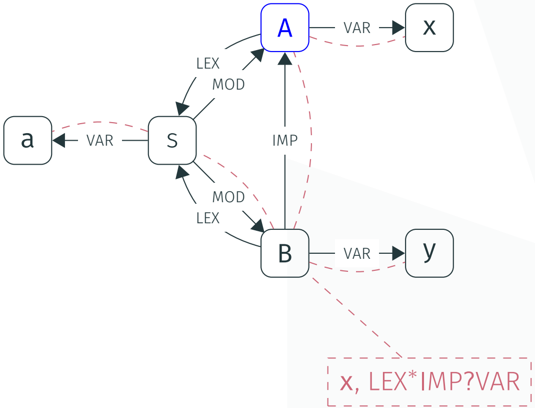
$\ell = \text{IMP}$



Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

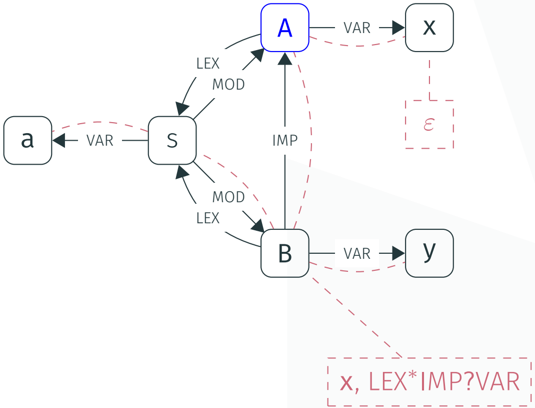
$\ell = \text{VAR}$



Resolution (simplified)

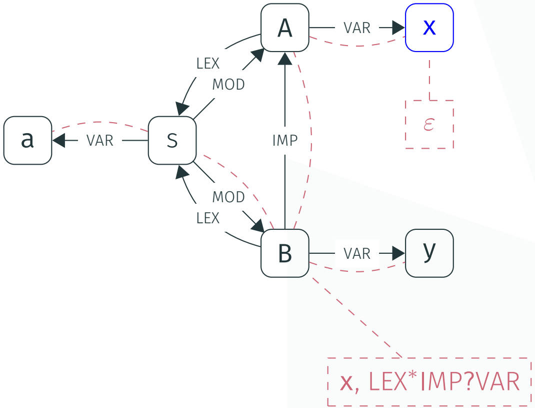
```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

$\ell = \text{VAR}$



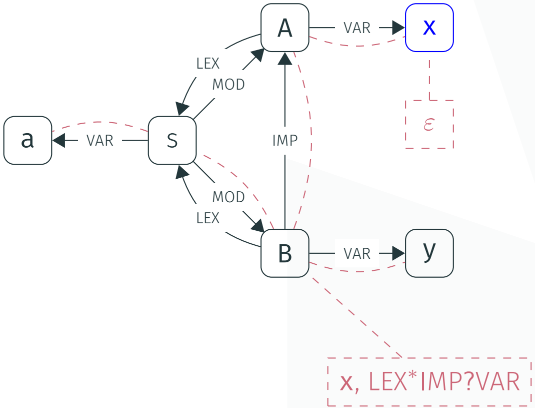
Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂_ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂_ℓℛ, n)
  return E
}
```



Resolution (simplified)

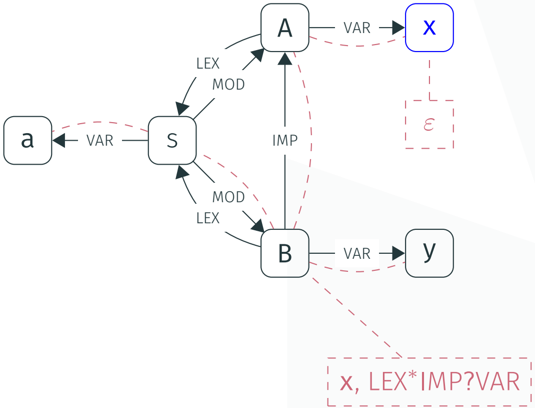
```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ϵ ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```



Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
  E := ∅
  if s = n & ε ∈ ℛ
    E += { s }
  for ℓ in ℒ
    if ∂_ℓℛ ≠ ∅
      for s′ in getEdges(𝒢, s, ℓ)
        E += Resolve(𝒢, s′, ∂_ℓℛ, n)
  return E
}
```

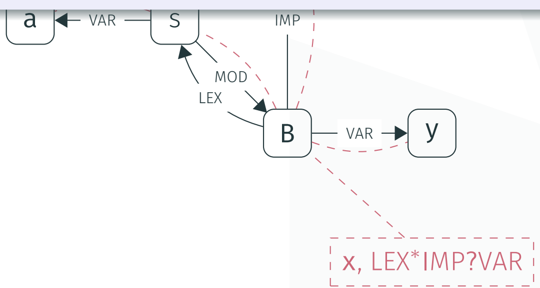
$A = \{ x \}$



Resolution (simplified)

```
Resolve(𝒢,s,ℛ,n) {
    for s′ in getEdges(𝒢,s,ℓ)
        E += Resolve(𝒢,s′,∂ℓℛ,n)
  return E
}
```

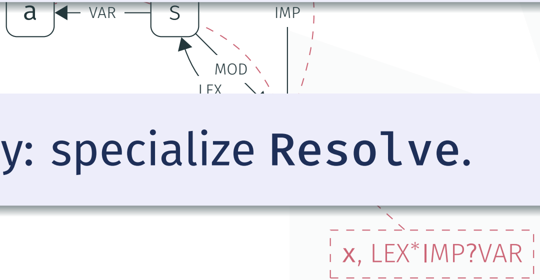
Profiling: 12.5% overhead for computing derivatives.



Resolution (simplified)

```
Resolve(𝒢, s, ℛ, n) {
      E += Resolve(𝒢, s′, ∂ℓℛ, n)
  return E
}
```

Profiling: 12.5% overhead for computing derivatives.



$\mathcal{R}$ known statically: specialize `Resolve`.