Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

First Class Traversal Idioms with Nested Attribute Grammars

Arie Middelkoop

Dept. of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
Web pages: `http://www.cs.uu.nl/wiki/Center`

15 April 2010 & Software Technology Colloquium

Introduction

Main Question

Software component for name analysis?

Main Question

Software component for name analysis?

Why?

- Pattern that occurs often in compilers
- Software engineering practice (e.g. elimination of code duplication)

Main Question

Software component for name analysis?

Why?

- Pattern that occurs often in compilers
- Software engineering practice (e.g. elimination of code duplication)

Actual question: mechanism for any traversal idiom?

More intriguing idiom: damas-milner inference


my traversal

Warning!

Work in progress! Controversial/not a silver bullet.

I hope this talk leads to:

- References to related material
- Improvements to the concepts
- Improvements to understanding and explanation
- Ideas for a nicer implementation

Related Work

This talk is related to attribute grammars:

- Higher-order Attribute Grammars
- Remote Attributes / Reference Attributes

... and probably many other formalisms/technologies

Background

Name Analysis

Concrete syntax:

$$\begin{aligned}&\textbf{let } x = 1\\ &\quad f = \lambda x.x \textbf{ in } f\ x\end{aligned}$$

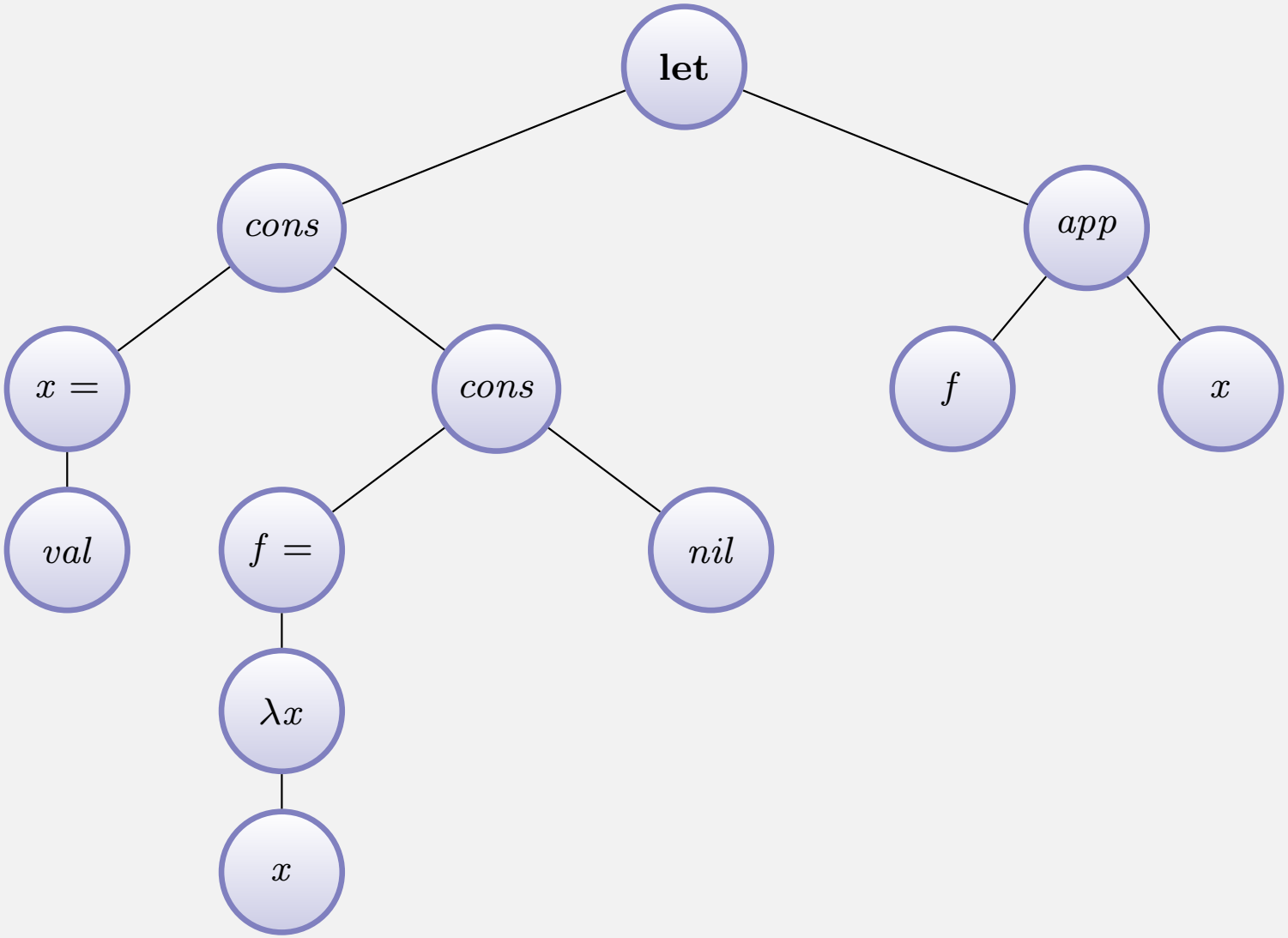
Abstract syntax:

$$\begin{aligned}&\mathit{Let}\ [\mathit{Decl}\ \texttt{"x"}\ (\mathit{Val}\ 1), \mathit{Decl}\ \texttt{"f"}\ (\mathit{Lam}\ \texttt{"x"}\ (\mathit{Var}\ \texttt{"x"}))]\\ &\quad (\mathit{App}\ (\mathit{Var}\ \texttt{"f"})\ (\mathit{Var}\ \texttt{"x"})))\end{aligned}$$

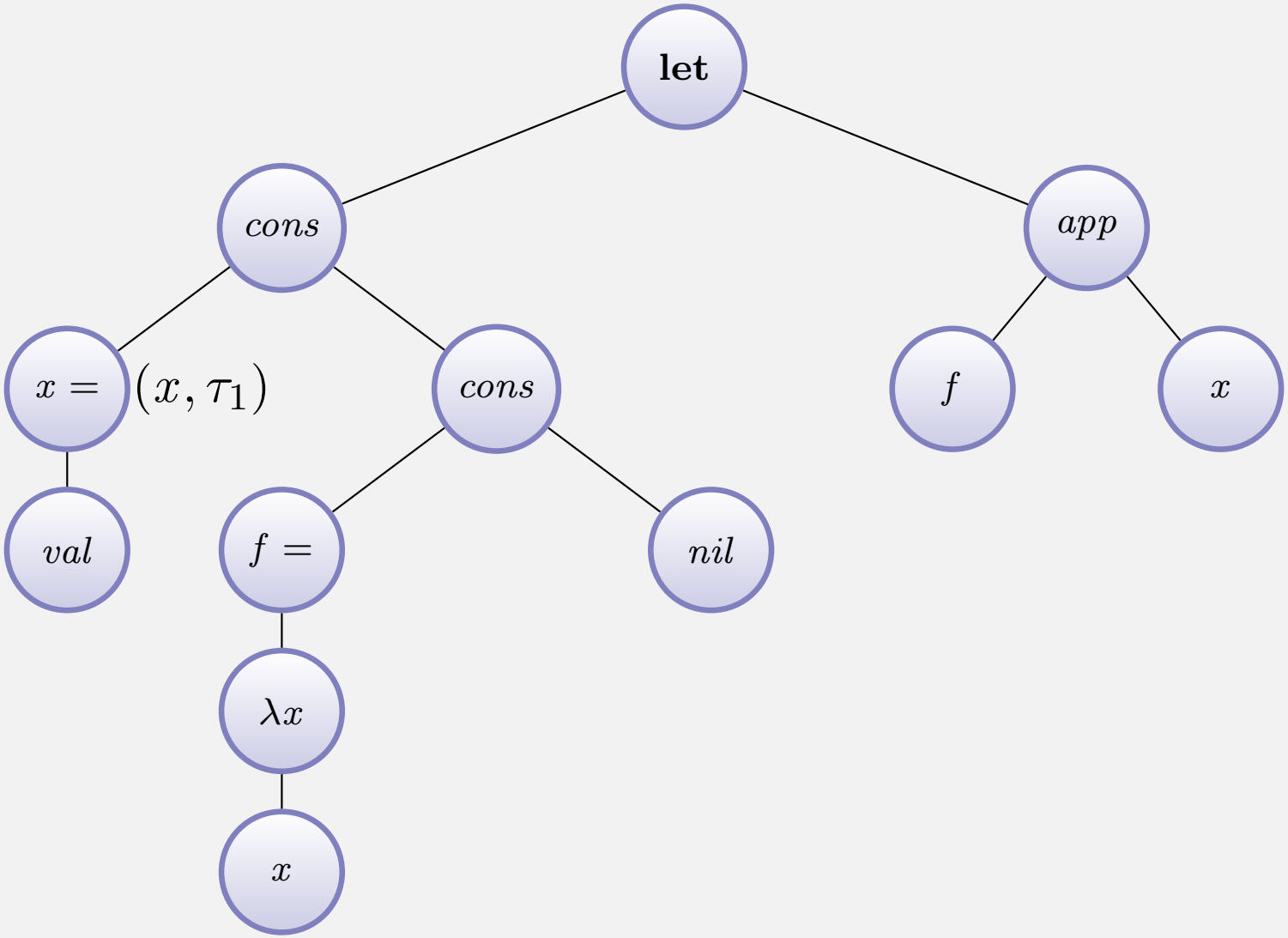
Questions:

- All identifiers defined?
- No duplicate definitions?
- What is the type of an identifier?

Name Analysis applied - gather environment



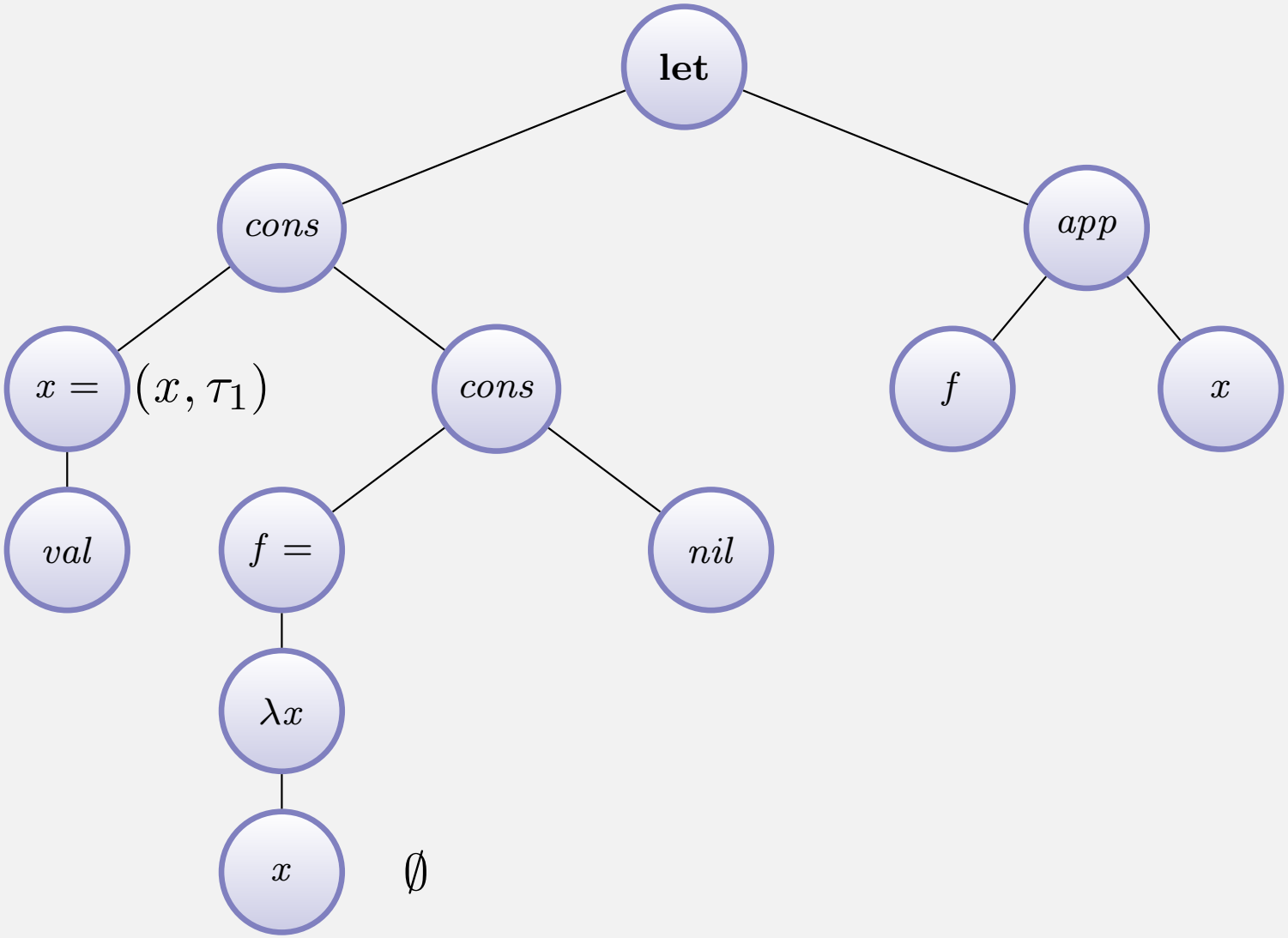
Name Analysis applied - gather environment



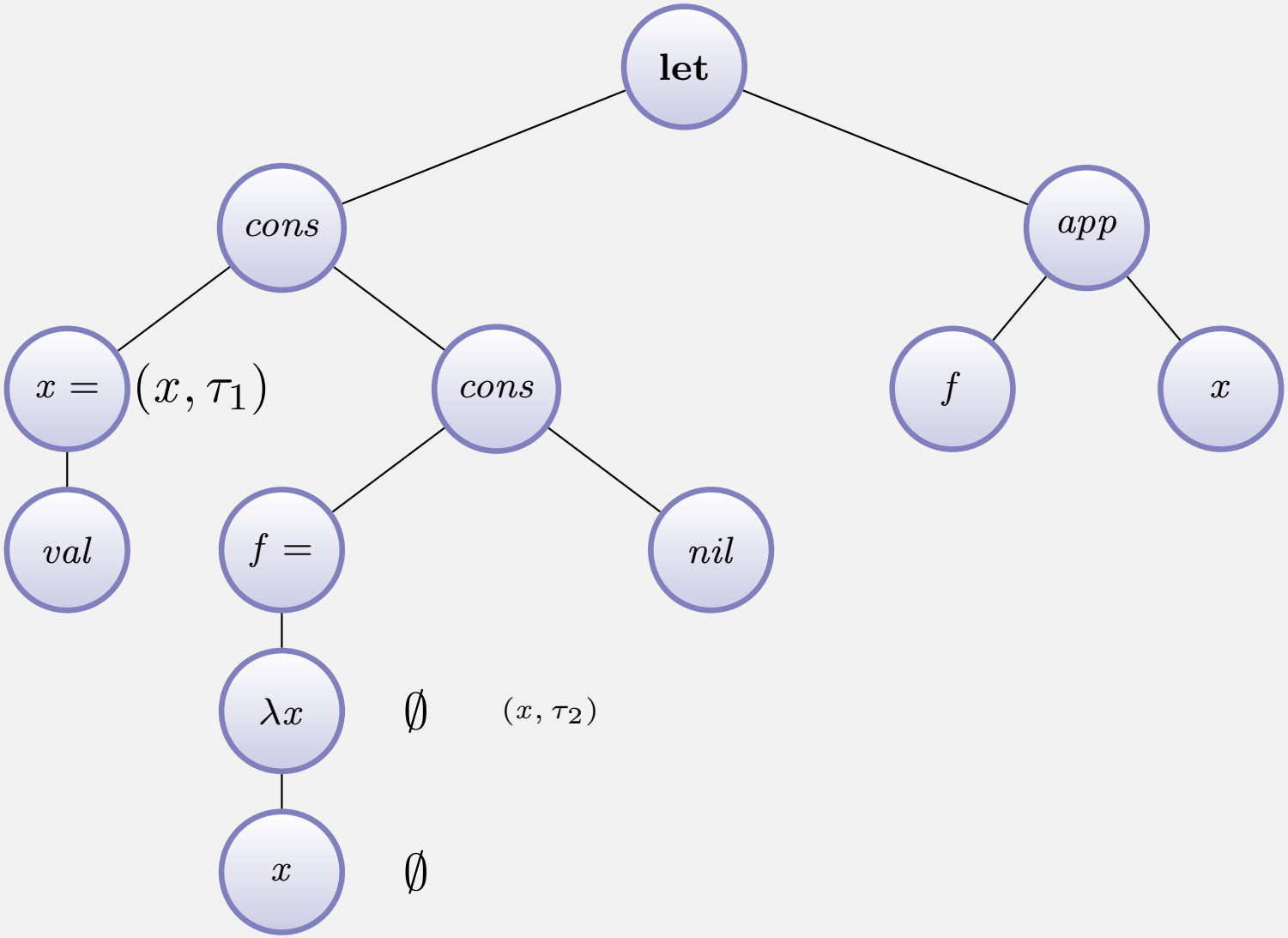
Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

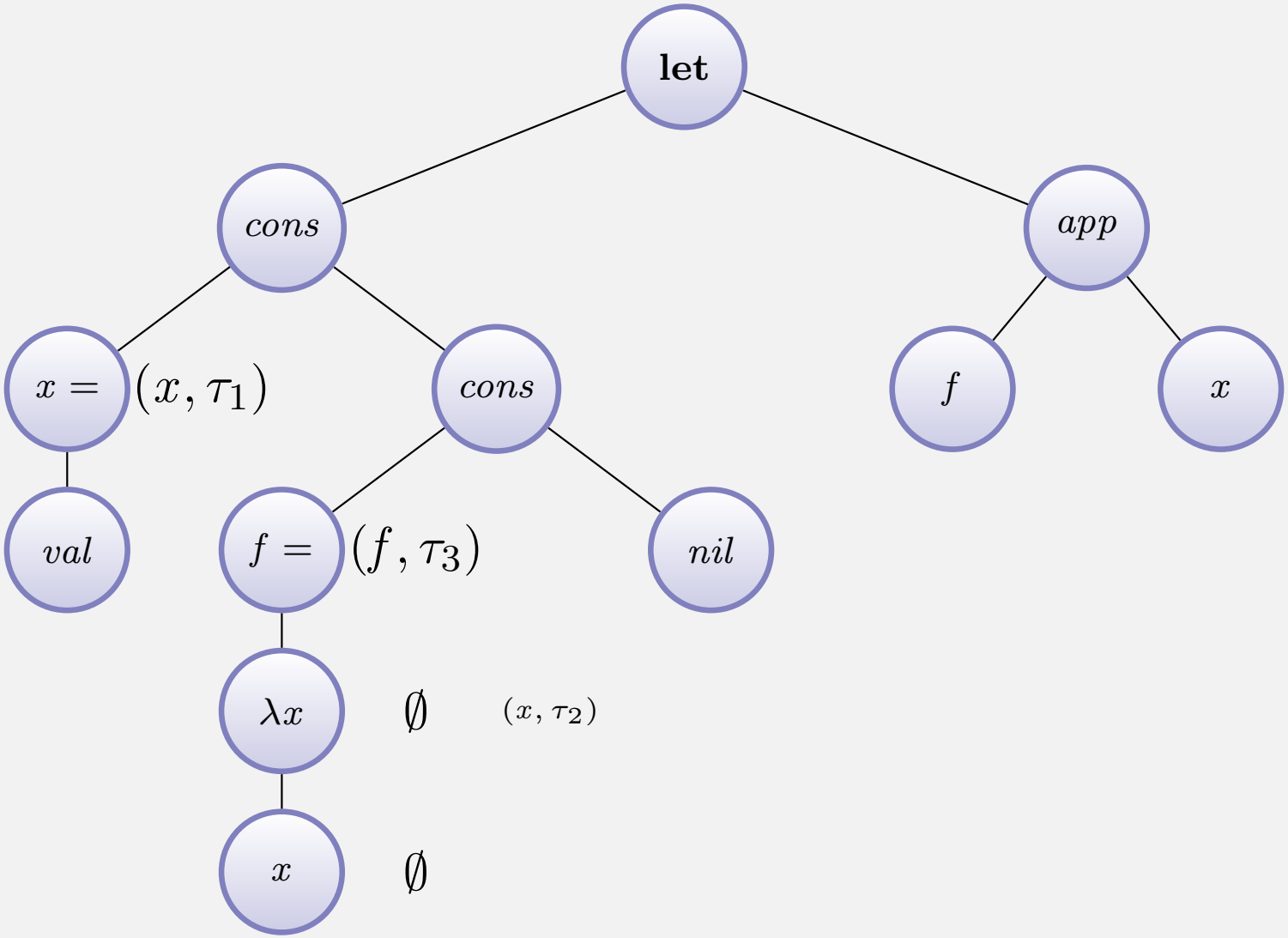
Name Analysis applied - gather environment



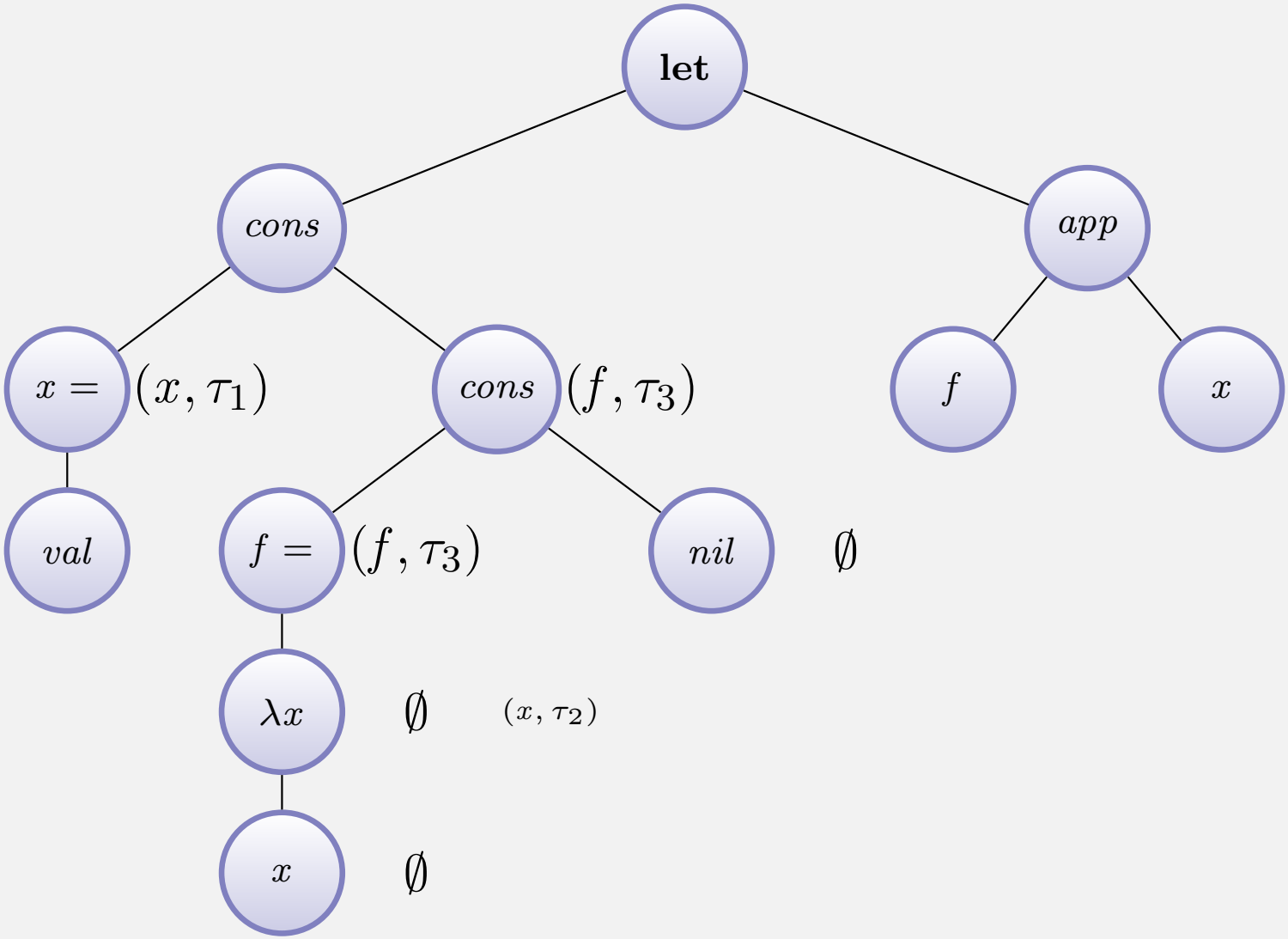
Name Analysis applied - gather environment



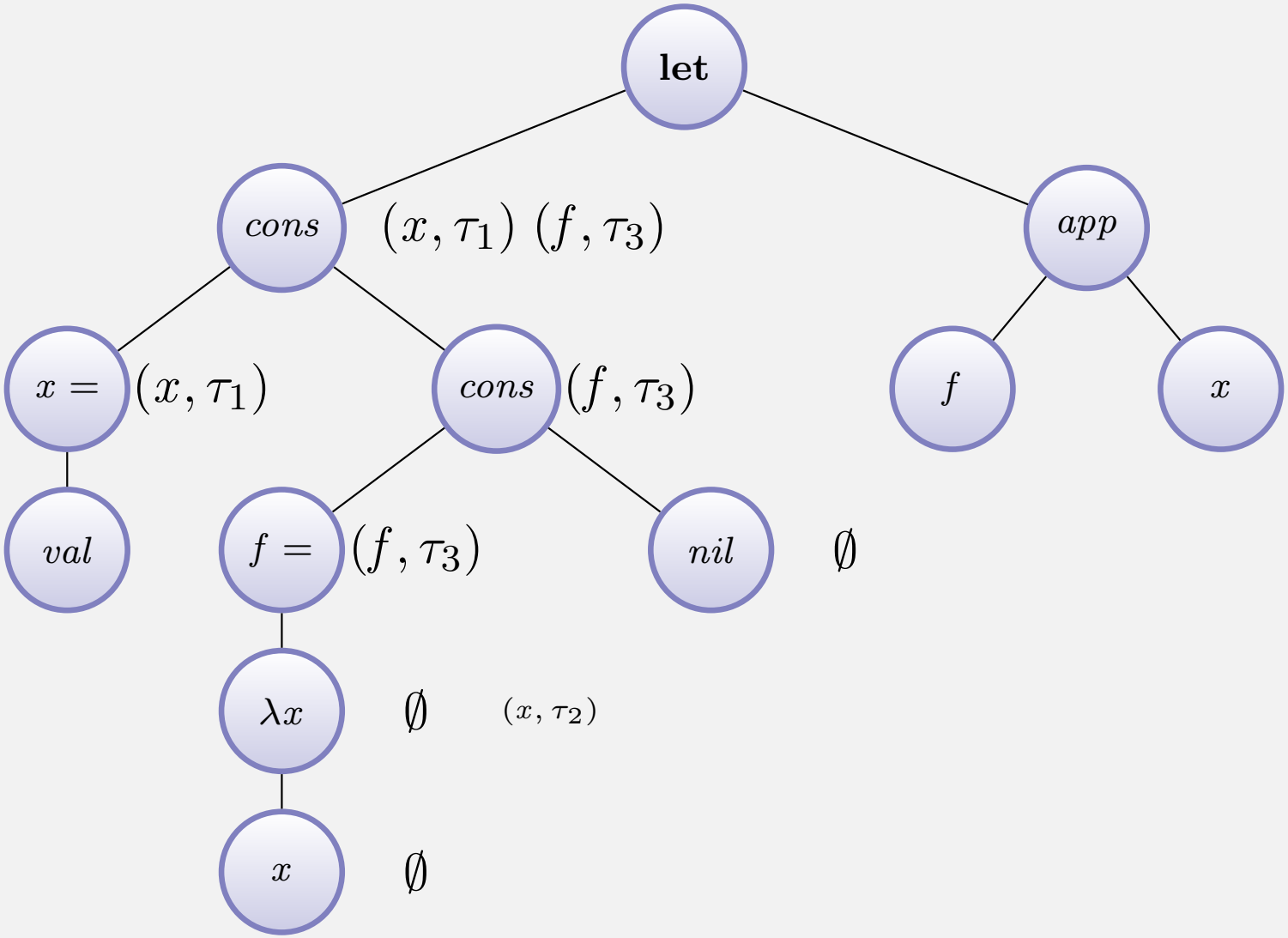
Name Analysis applied - gather environment



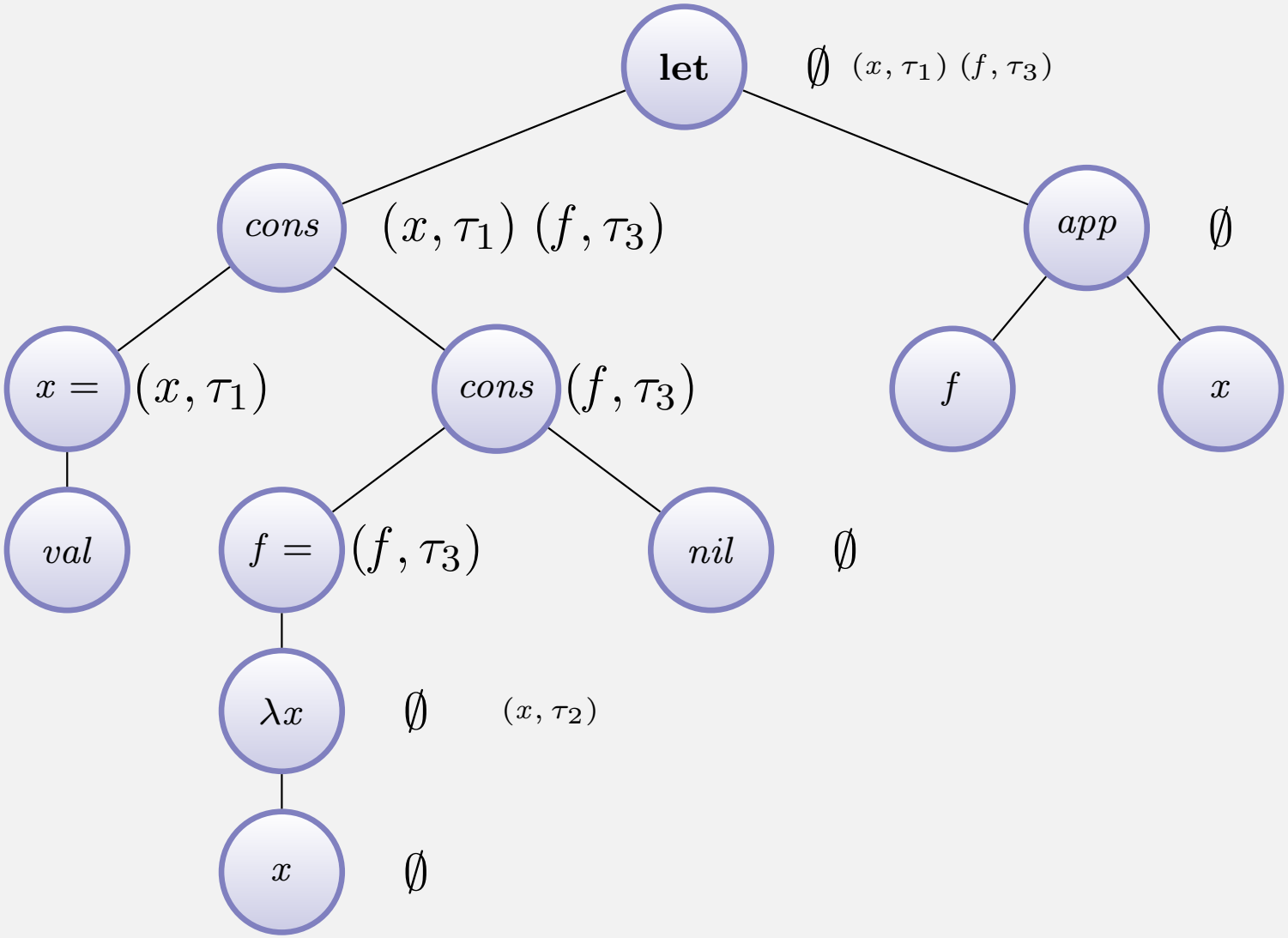
Name Analysis applied - gather environment



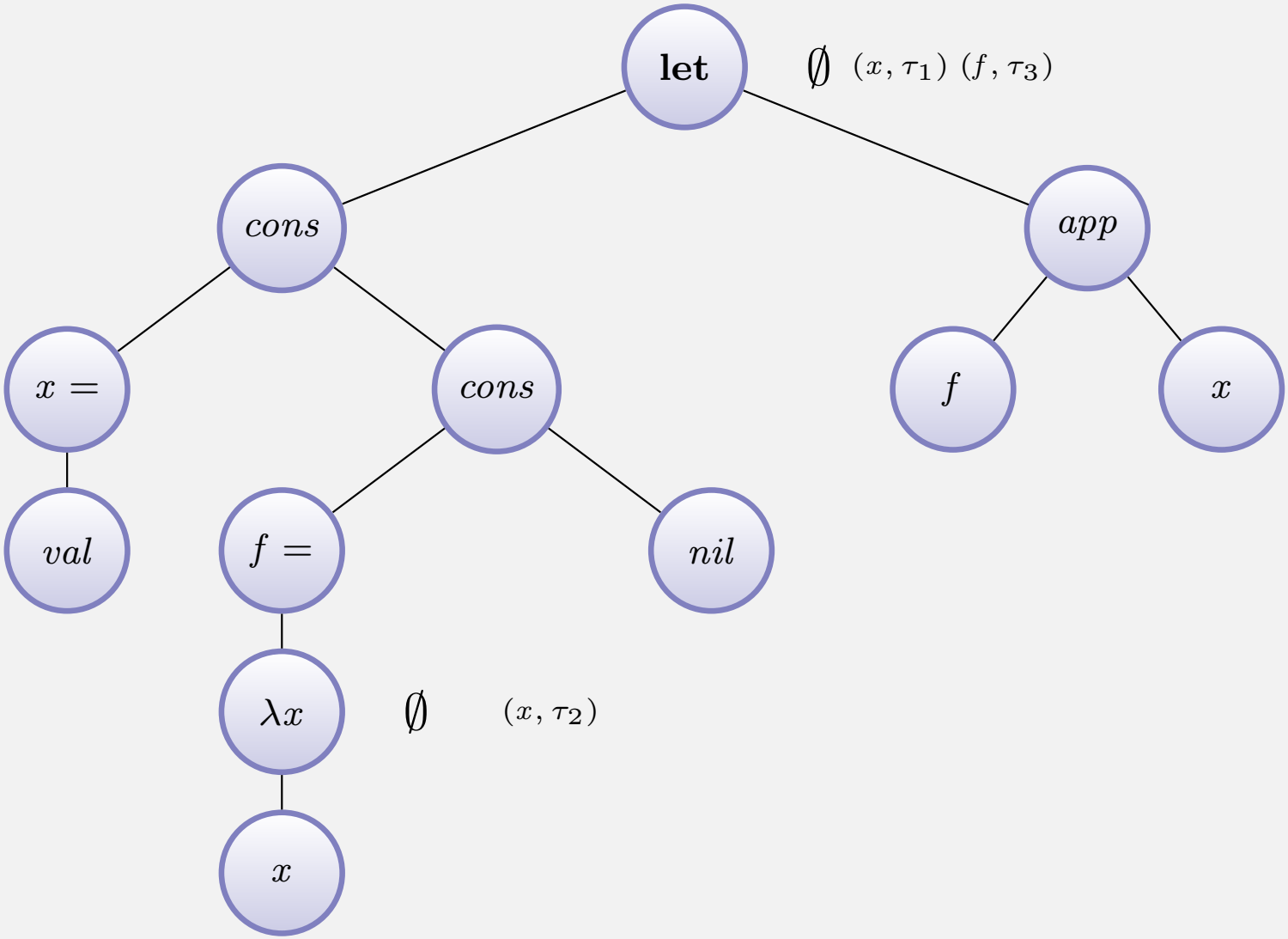
Name Analysis applied - gather environment



Name Analysis applied - gather environment



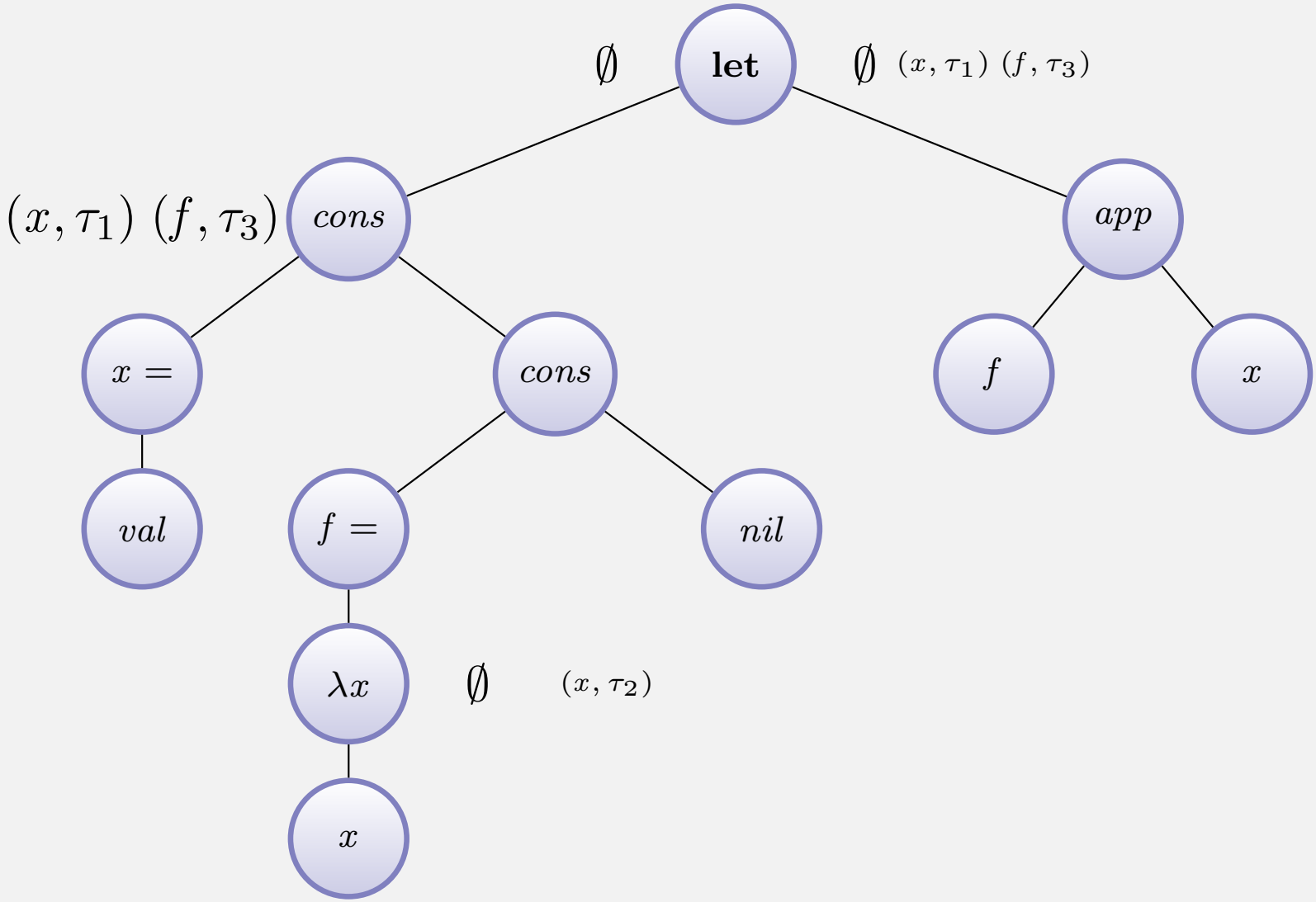
Name Analysis applied - distribute environment



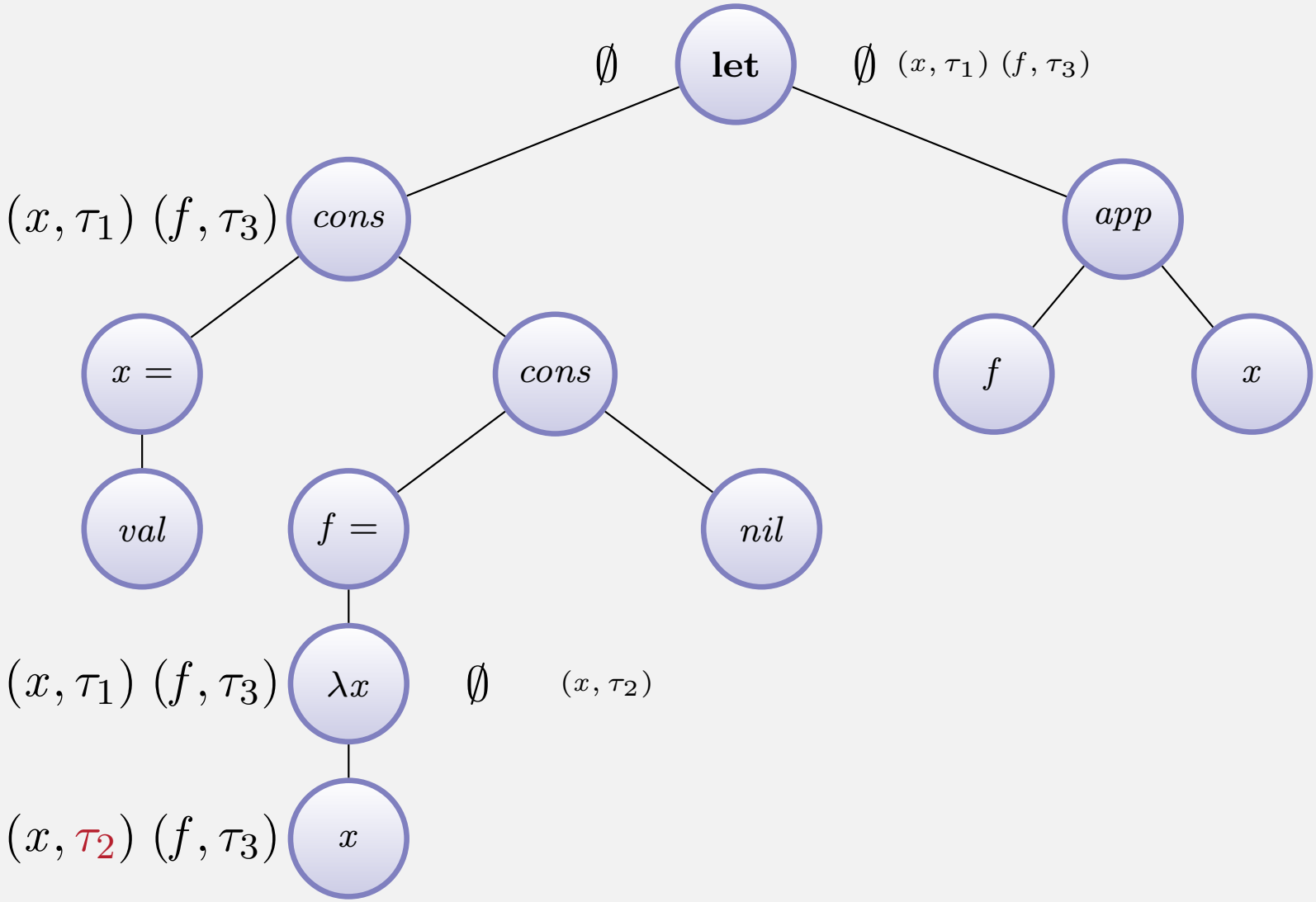
Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

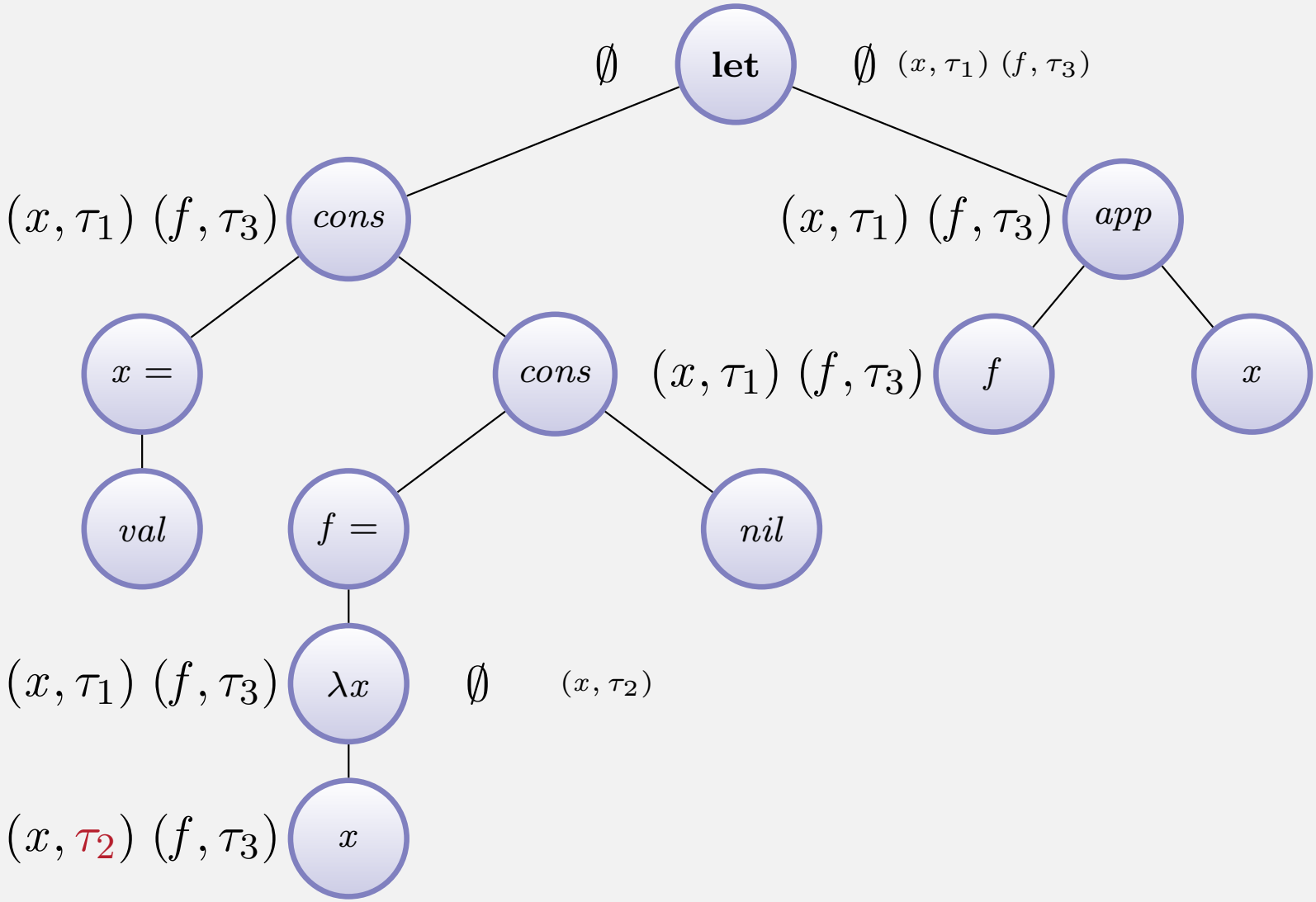
Name Analysis applied - distribute environment



Name Analysis applied - distribute environment



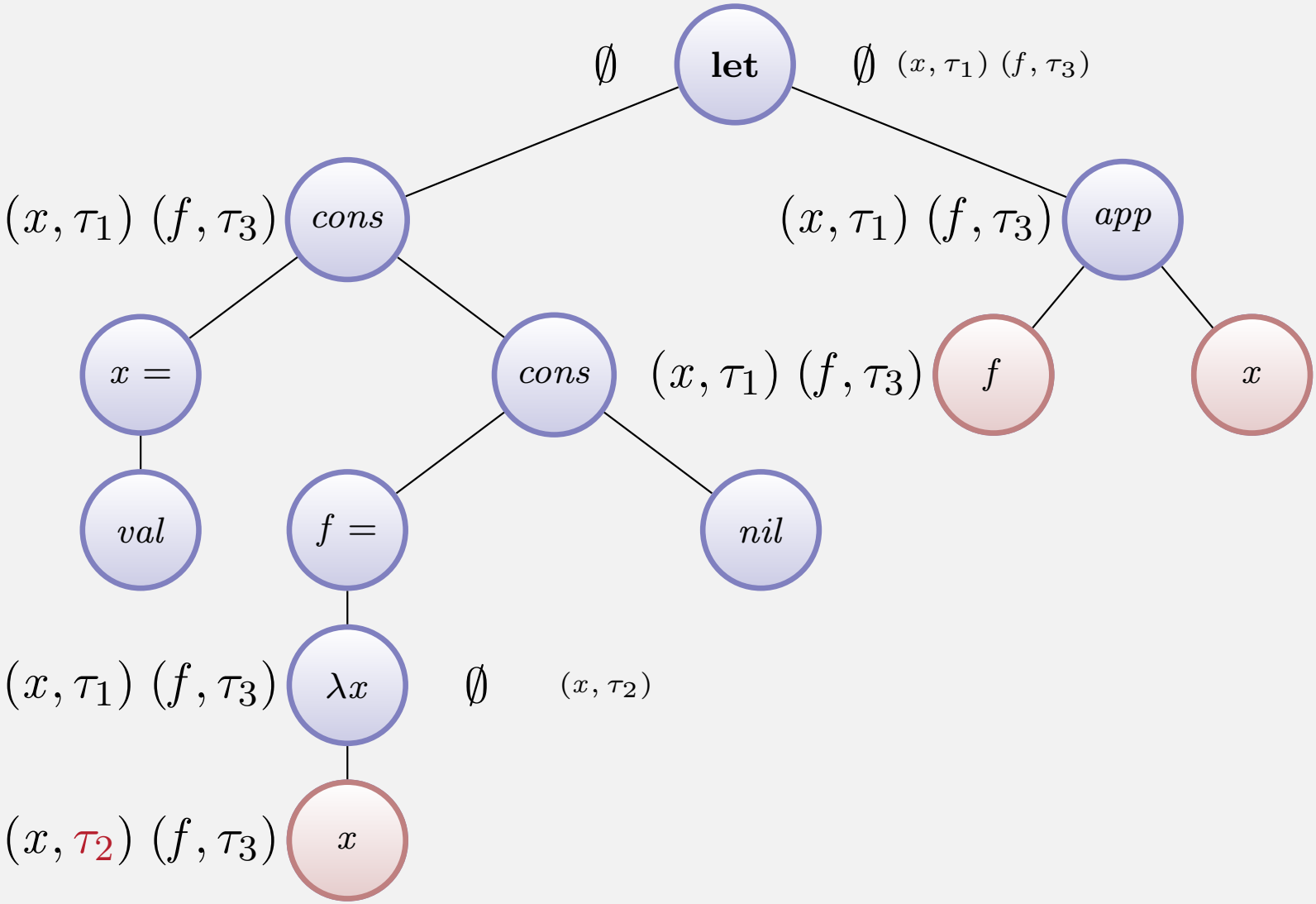
Name Analysis applied - distribute environment



Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

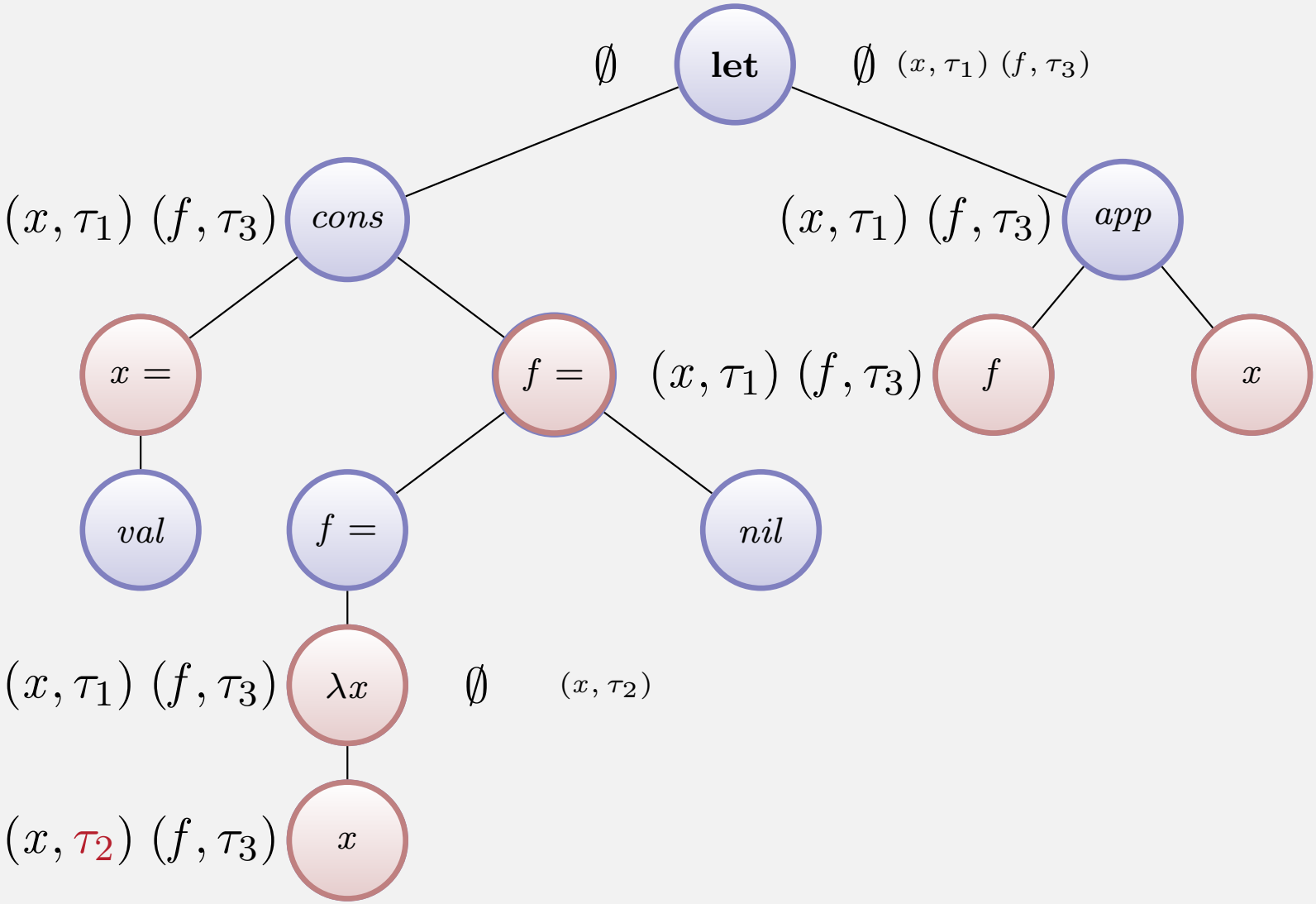
Name Analysis applied - distribute environment



Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Name Analysis applied - distribute environment



Implementation - parser

```
pExpr :: Parser Token T_Expr

pExpr = sem_Expr_Let $ pKey "let" * pDecls * pKey "in" * pExpr
      | sem_Expr_Lam $ pKey "\\" * pIdent * pKey "." * pExpr
      | pExprApps

pExprApps = pChainl sem_Expr_App pExprBase

pExprBase = sem_Expr_Var $ pIdent
          | sem_Expr_Val $ pValue
          | pParens pExpr

pDecls = pFoldr (sem_Decls_Cons, sem_Decls_Nil) pDecl
pDecl = sem_Decl_Decl $ pIdent * pKey "=" * pExpr
```

Implementation - constructors (1)

$$
\begin{aligned}
sem\_Expr\_Var\ nm &= \mathbf{sem} :: Expr\ ... \\
sem\_Expr\_Val\ val &= \mathbf{sem} :: Expr\ ... \\
sem\_Expr\_App\ l\ r &= \mathbf{sem} :: Expr\ ... \\
sem\_Expr\_Lam\ nm\ b &= \mathbf{sem} :: Expr\ ... \\
sem\_Expr\_Let\ ds\ b &= \mathbf{sem} :: Expr\ ... \\
sem\_Decls\_Cons\ d\ ds &= \mathbf{sem} :: Decls\ ... \\
sem\_Decls\_Nil &= \mathbf{sem} :: Decls\ ... \\
sem\_Decl\_Decl\ nm\ e &= \mathbf{sem} :: Decl\ ...
\end{aligned}
$$

Implementation - constructors (1)

$$
\begin{aligned}
sem\_Expr\_Var\ nm &= \mathbf{sem} :: Expr\ ... \\
sem\_Expr\_Val\ val &= \mathbf{sem} :: Expr\ ... \\
sem\_Expr\_App\ l\ r &= \mathbf{sem} :: Expr\ ... \\
sem\_Expr\_Lam\ nm\ b &= \mathbf{sem} :: Expr\ ... \\
sem\_Expr\_Let\ ds\ b &= \mathbf{sem} :: Expr\ ... \\
sem\_Decls\_Cons\ d\ ds &= \mathbf{sem} :: Decls\ ... \\
sem\_Decls\_Nil &= \mathbf{sem} :: Decls\ ... \\
sem\_Decl\_Decl\ nm\ e &= \mathbf{sem} :: Decl\ ...
\end{aligned}
$$

$$
\begin{aligned}
&sem\_Expr\_Lam :: String \to T\_Expr \to T\_Expr \\
&sem\_Expr\_Let :: T\_Decls \to T\_Expr \to T\_Expr
\end{aligned}
$$

Implementation - interfaces

$$
\begin{array}{l}
\textbf{itf}\ \mathit{Expr}\ \mathit{Decls}\ \mathit{Decl} \\
\quad \textbf{visit}\ \mathit{gather} \quad\ \ \textbf{syn}\ \mathit{gathEnv} :: [(\mathit{String}, \mathit{Ty})] \\
\quad \textbf{visit}\ \mathit{distribute}\ \textbf{inh}\ \mathit{finEnv} :: [(\mathit{String}, \mathit{Ty})] \\
\qquad\qquad\qquad\ \ \textbf{syn}\ \mathit{errors}\ \ :: [\mathit{String}] \\
\quad \textbf{order}\ \mathit{gather} < \mathit{distribute}
\end{array}
$$

Implementation - interfaces

$$
\begin{aligned}
&\textbf{itf}\ \mathit{Expr}\ \mathit{Decls}\ \mathit{Decl}\\
&\quad \textbf{visit}\ \mathit{gather} \quad\ \ \textbf{syn}\ \mathit{gathEnv} :: [(\mathit{String}, \mathit{Ty})]\\
&\quad \textbf{visit}\ \mathit{distribute}\ \textbf{inh}\ \mathit{finEnv} :: [(\mathit{String}, \mathit{Ty})]\\
&\qquad\qquad\qquad\ \ \textbf{syn}\ \mathit{errors}\ \ :: [\mathit{String}]\\
&\quad \textbf{order}\ \mathit{gather} < \mathit{distribute}
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{newtype}\ \mathit{T\_Expr}\ \mathit{st} = \ldots\\
&\textbf{data}\ \mathit{Inh\_Expr} = \mathit{Inh\_Expr}\\
&\quad \{\mathit{finEnv\_Inh\_Expr} \quad :: [(\mathit{String}, \mathit{Ty})]\}\\
&\textbf{data}\ \mathit{Syn\_Expr} = \mathit{Syn\_Expr}\\
&\quad \{\mathit{gathEnv\_Syn\_Expr} :: [(\mathit{String}, \mathit{Ty})]\\
&\quad , \mathit{errors\_Syn\_Expr} \quad :: [\mathit{String}]\\
&\quad \}\\
&\mathit{eval\_Expr} :: (\mathit{forall}\ \mathit{st}.\mathit{T\_Expr}\ \mathit{st}) \rightarrow \mathit{Inh\_Expr} \rightarrow \mathit{Syn\_Expr}
\end{aligned}
$$

Implementation - Constructors (2)

$$
\begin{aligned}
&sem\_Expr\_Var\ nm = \\
&\quad \mathbf{sem} :: Expr \\
&\qquad loc.mbVal \quad = lookup\ nm\ lhs.finEnv \\
&\qquad lhs.errors \quad = maybe\ [\texttt{"missing: "} +\!\!+ nm]\ (const\ [])\ loc.mbVal \\
&\qquad lhs.gathEnv = \emptyset
\end{aligned}
$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Implementation - Constructors (2)

$$
\begin{aligned}
&sem\_Expr\_Var\ nm = \\
&\quad \mathbf{sem} :: Expr \\
&\qquad loc.mbVal \;\; = lookup\ nm\ lhs.finEnv \\
&\qquad lhs.errors \;\; = maybe\ [\texttt{"missing: "} +\!\!+ nm]\ (const\ [])\ loc.mbVal \\
&\qquad lhs.gathEnv = \emptyset
\end{aligned}
$$

$$
\begin{aligned}
&sem\_Expr\_Lam\ nm\ b = \\
&\quad \mathbf{sem} :: Expr \\
&\qquad \mathbf{child}\ body :: Expr = b \\
&\qquad loc.(envWith, envWithout) = partition\ ((\equiv nm).fst)\ lhs.finEnv \\
&\qquad loc.dupErrs = \mathbf{if}\ length\ loc.envWith > 1\ \mathbf{then}\ [\texttt{"dup: "} +\!\!+ nm] \\
&\qquad lhs.errors \;\; = loc.dupErrs +\!\!+ body.errors \\
&\qquad loc.binding \;\; = [(nm, loc.argType)] \\
&\qquad body.finEnv = loc.binding +\!\!+ loc.envWithout \\
&\qquad lhs.gathEnv = filter\ ((\not\equiv nm).fst)\ body.gathEnv
\end{aligned}
$$

Implementation - Constructors (2)

$$
\begin{aligned}
&sem\_Expr\_App\ l\ r = \\
&\quad \textbf{sem} :: Expr \\
&\qquad \textbf{child}\ left\ \ :: Expr = l \\
&\qquad \textbf{child}\ right :: Expr = r \\
&\qquad left.finEnv\ \ = lhs.finEnv \\
&\qquad right.finEnv = lhs.finEnv \\
&\qquad lhs.gathEnv\ = left.gathEnv +\!\!+ right.gathEnv \\
&\qquad lhs.errors\ \ = left.errors +\!\!+ right.errors
\end{aligned}
$$

Implementation - Constructors (2)

$$
\begin{aligned}
&sem\_Expr\_App\ l\ r = \\
&\quad \textbf{sem} :: Expr \\
&\qquad \textbf{child}\ left\ \ :: Expr = l \\
&\qquad \textbf{child}\ right :: Expr = r \\
&\qquad left.finEnv\ \ = lhs.finEnv \\
&\qquad right.finEnv = lhs.finEnv \\
&\qquad lhs.gathEnv\ = left.gathEnv +\!\!+ right.gathEnv \\
&\qquad lhs.errors\ \ = left.errors +\!\!+ right.errors
\end{aligned}
$$

Other cases: similar.....

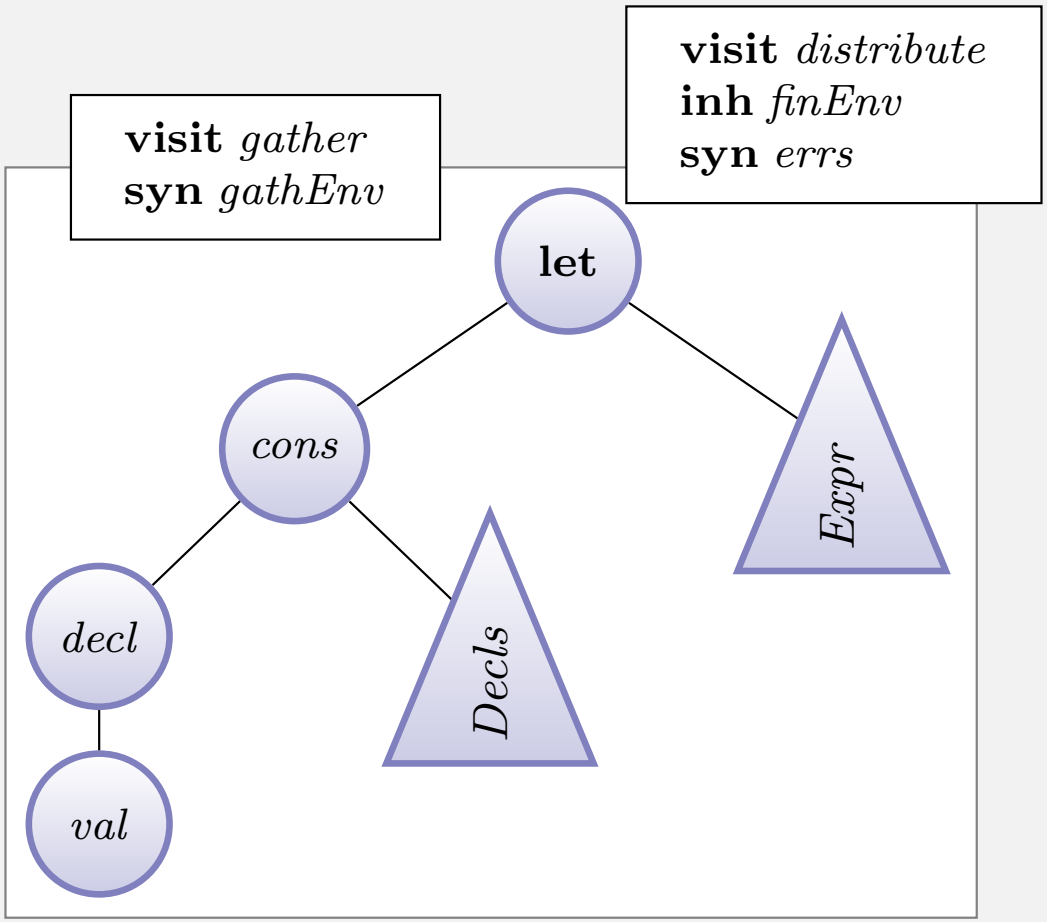
Name Analysis in UHC

- EH AST has 39 constructors that deal with names
- $Core$ AST about 16 constructors
- Lost the count in the HS AST
- Many more ASTs in UHC
- ... think of all other compilers

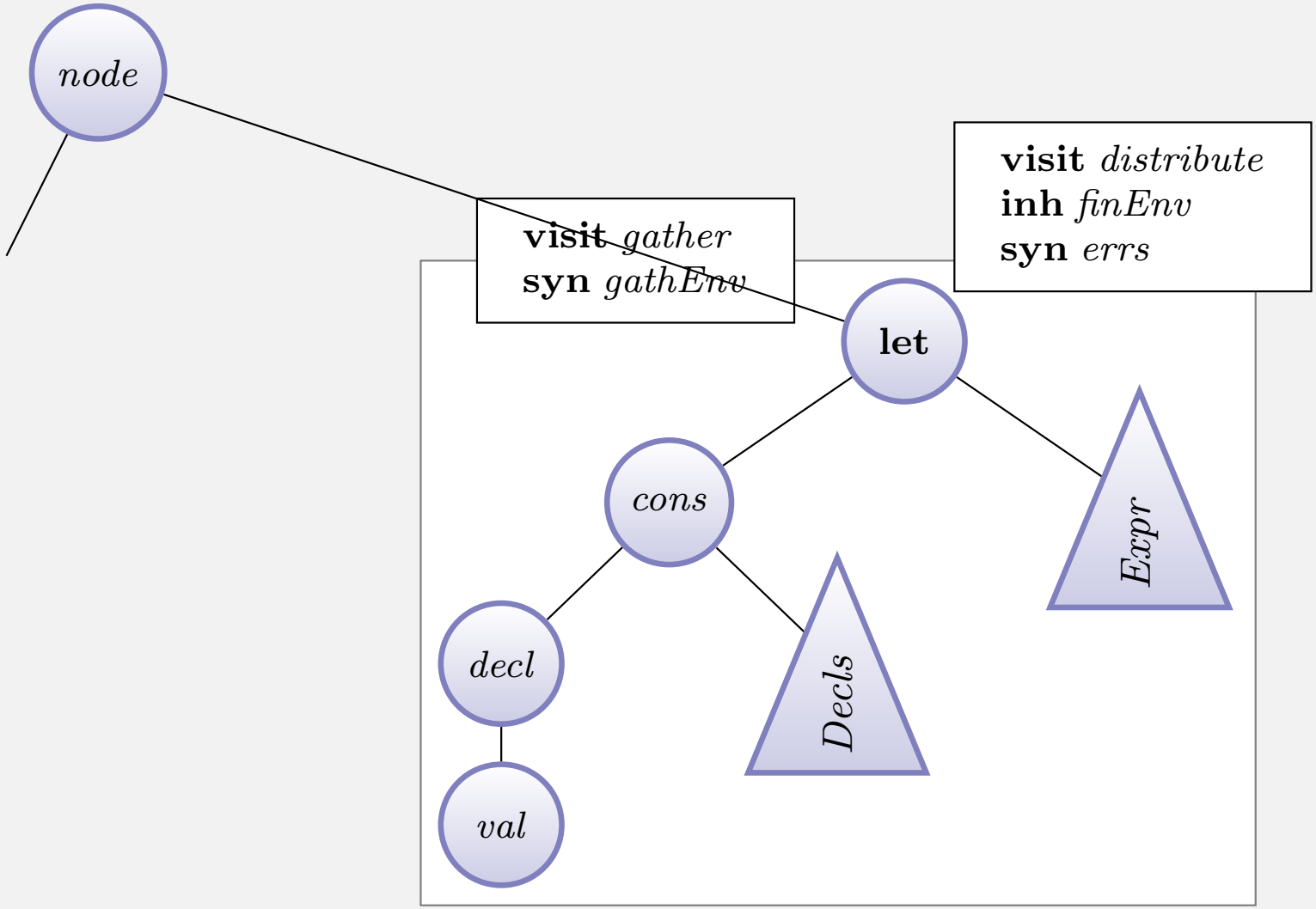
Code duplication!

Components

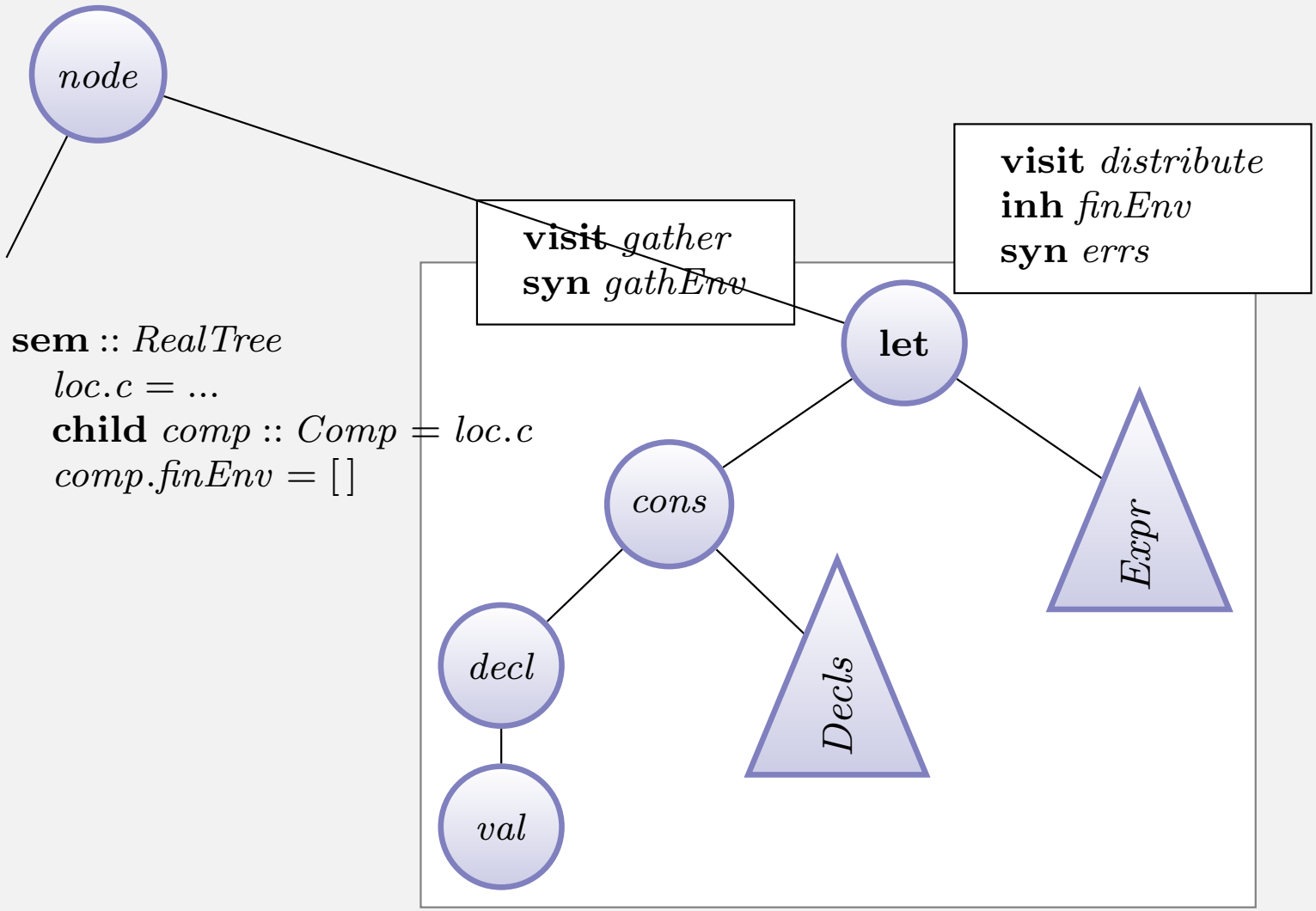
Towards Attributed Trees as Components (1)



Towards Attributed Trees as Components (1)



Towards Attributed Trees as Components (1)

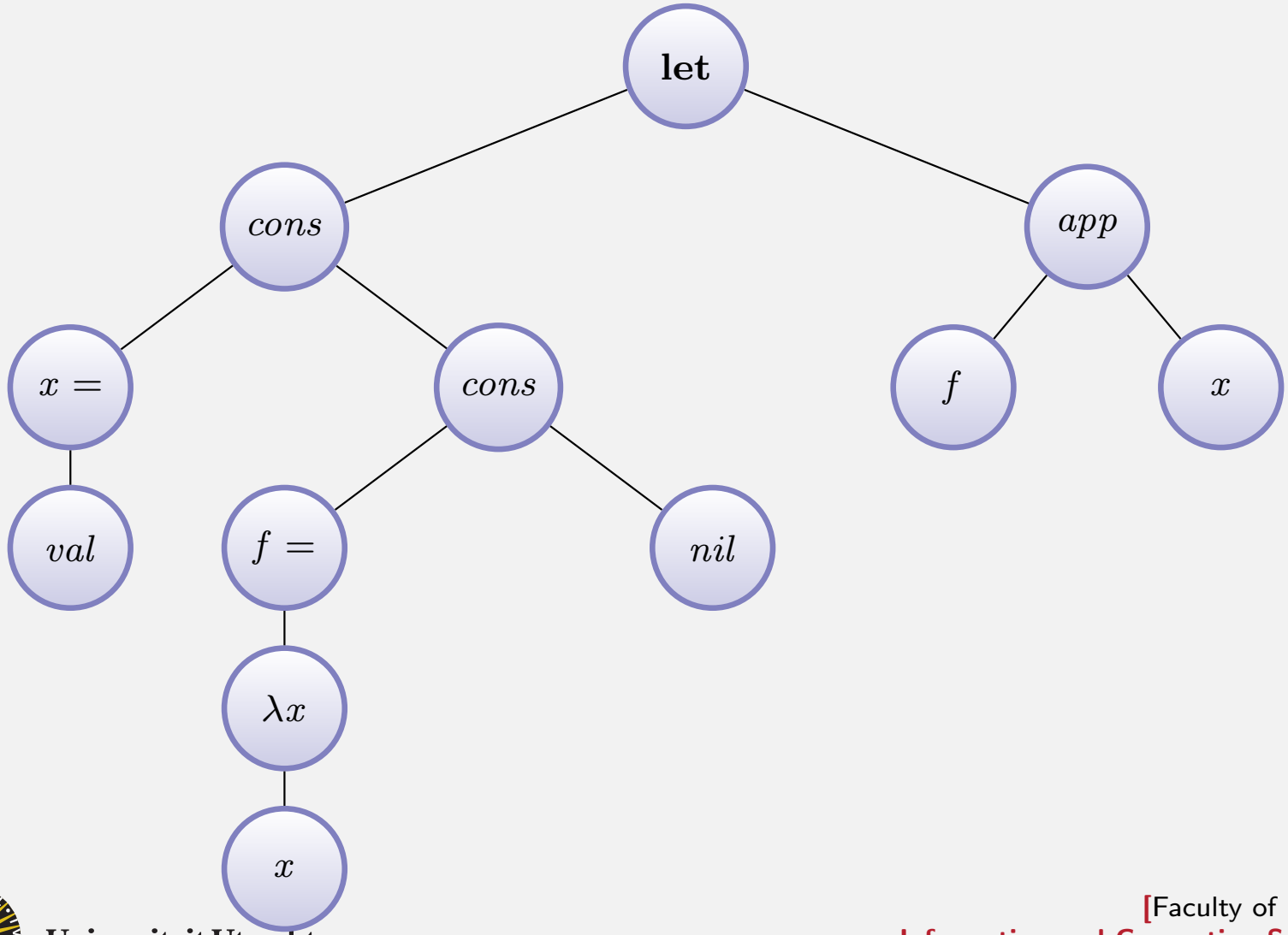


Show stoppers

1. Verbose AST
   - Expressions, declarations: too fine grained
   - Essence of name analysis: binding sites, use sites, scoping
2. Interfacing only with root insufficient
   - Access to the values found for identifiers
   - Access to the error messages at one particular node
3. Extend a component?
   - Mismatch between offered and needed functionality
   - Opening up the component, without affecting the original
   - Overriding attributes, adding new attributes
4. Intuitiveness and predictability of code?
   - Delegation of work: finished when results are needed?
   - Functional code (side effect free/order independent)

More abstract AST

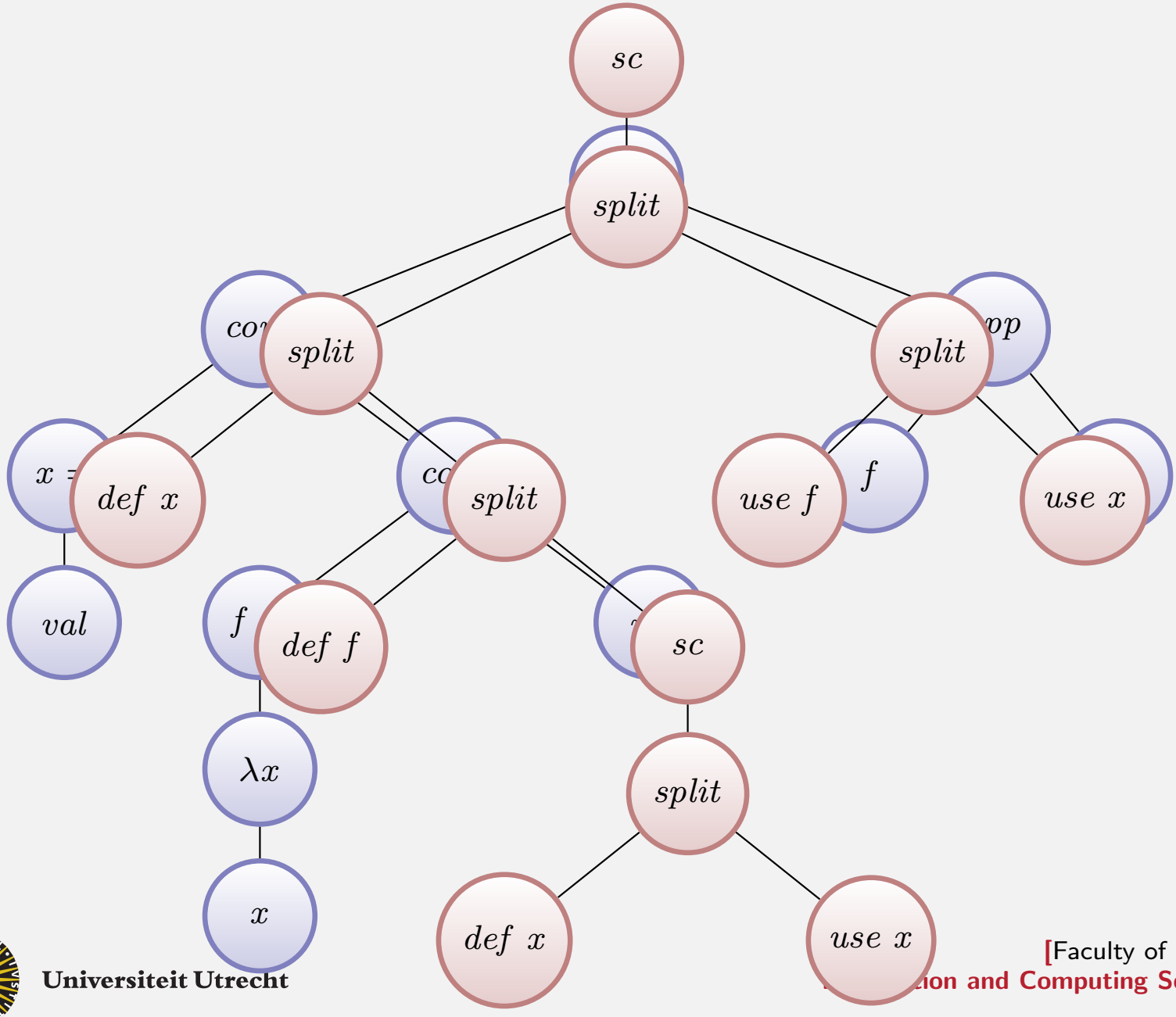
Coarser AST



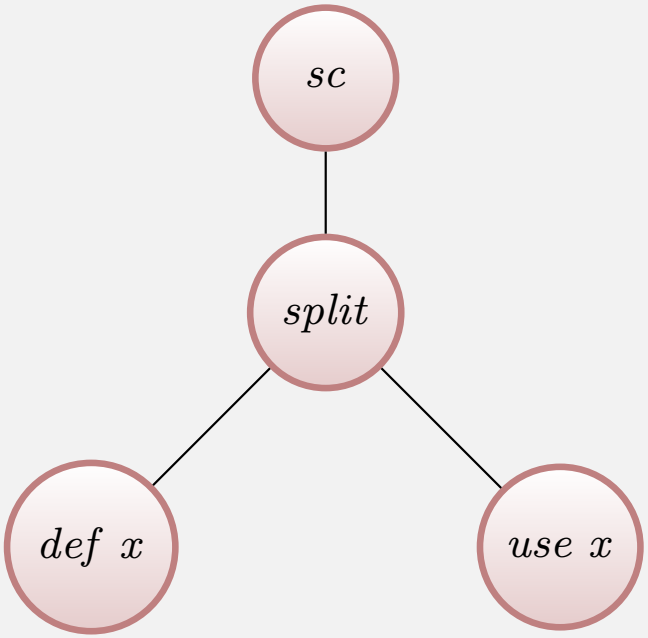
Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Coarser AST


sc
split
split
split
def x
split
use f
f
use x
val
f
def f
sc
λx
split
x
def x
use x

Coarser AST



More Abstract Code

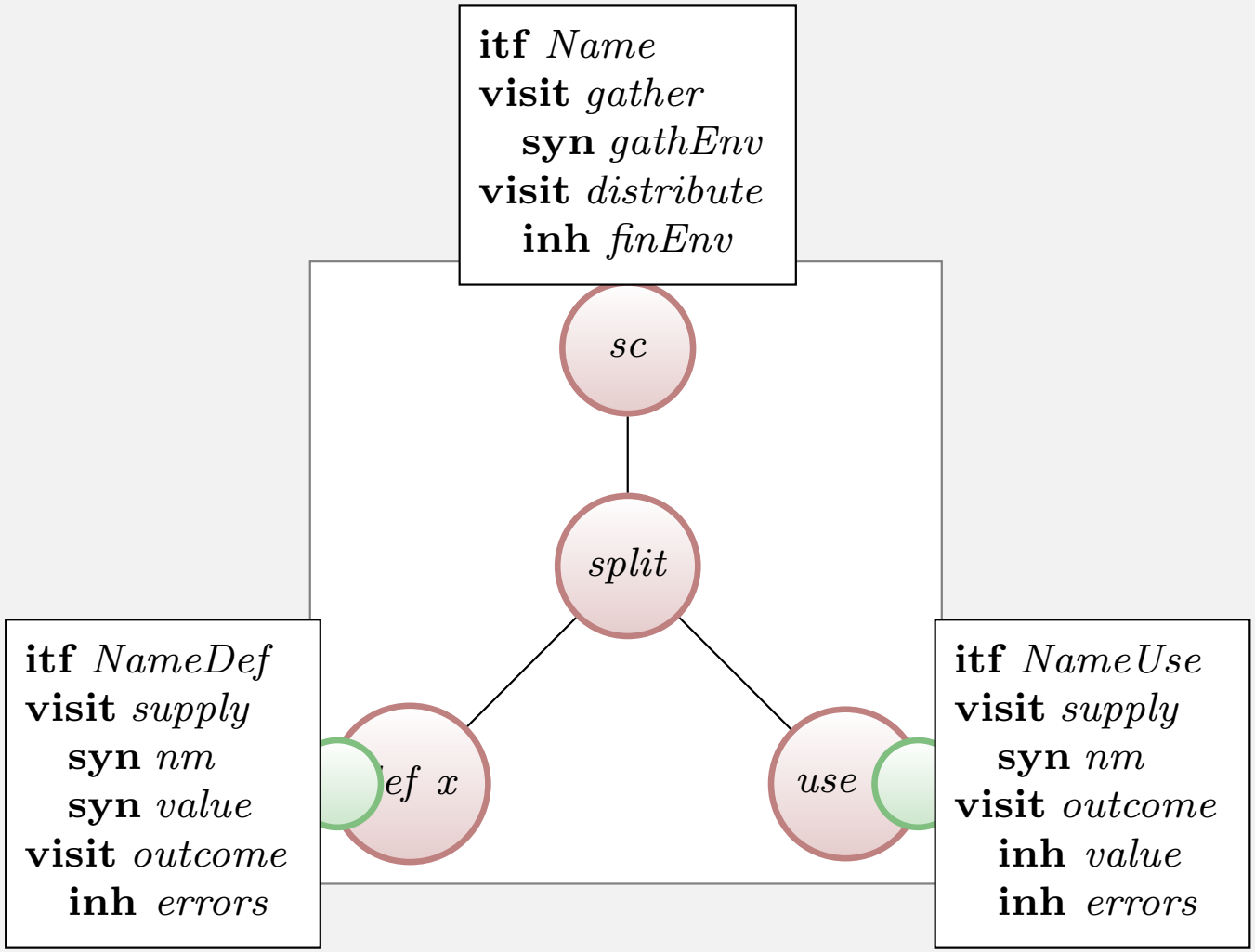
```
itf Name α      visit gather     syn gathEnv use {++} {[]} :: [(String, α)]
                visit distribute inh finEnv :: [(String, α)]
                                 syn errors use {++} {[]} :: [String]

sem_Name_Scope b =
  sem :: Name α
    child body :: Name α = b
    body.finEnv = (lhs.finEnv \\ (map fst body.gathEnv)) ++ body.gathEnv

sem_Name_Split l r =
  sem :: Name α
    child left  :: Name α = l
    child right :: Name α = r

sem_Name_Def x v =
  sem :: Name α
    lhs.gathEnv = [(x, v)]
    lhs.errors  = if length (filter ((≡ x).fst) lhs.finEnv) > 1
                  then ["dup: " ++ x] else []

sem_Name_Use x =
  sem :: Name α
    loc.mbVal = lookup x lhs.finEnv
    lhs.errors = if isNothing loc.mbVal then ["missing: " ++ x] else []
    loc.val    = maybe ⊥ id loc.mbVal   -- how to access this??
```

Instantiation of the component

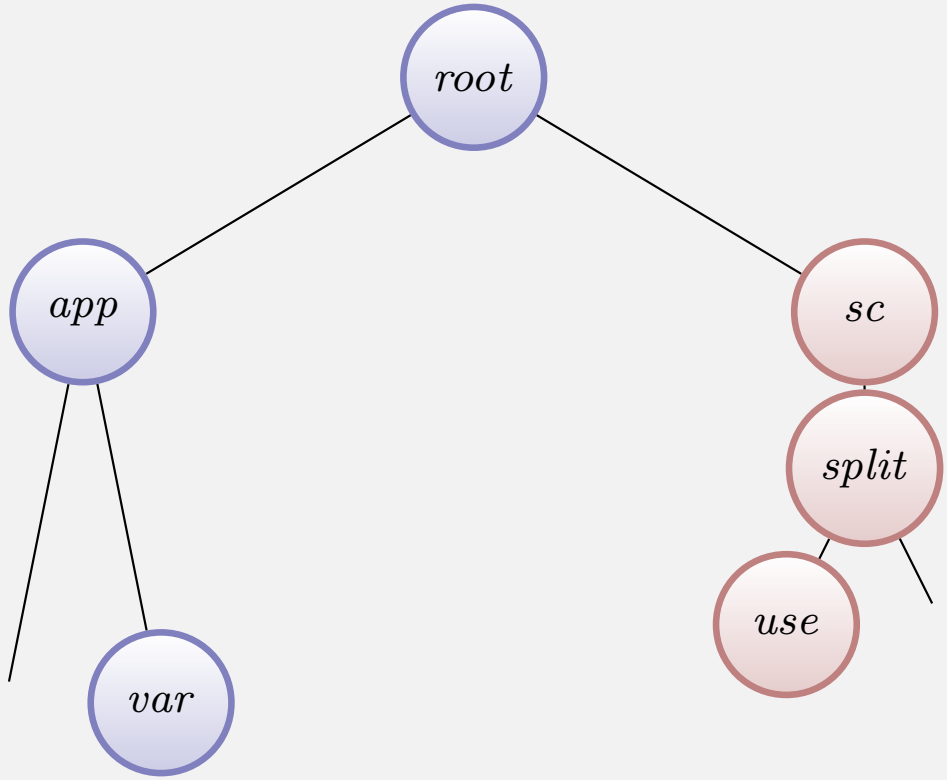
```
itf Expr syn nmTree :: T_Name st Ty

sem_Expr_Lam x b =
  sem :: Expr
    child body :: Expr = b
    loc.τ = ...
    lhs.nmTree = sem_Name_Scope
                 $ sem_Name_Split (sem_Name_Def x loc.τ)
                 $ body.nmTree

sem_Expr_Var x =
  sem :: Expr
    lhs.nmTree = sem_Name_Use x

sem_Root_Root b =
  sem :: Root
    child body :: Expr = b
    child name :: Name Ty = body.nmTree
    lhs.errors = name.errors
```

Interface to Subtrees

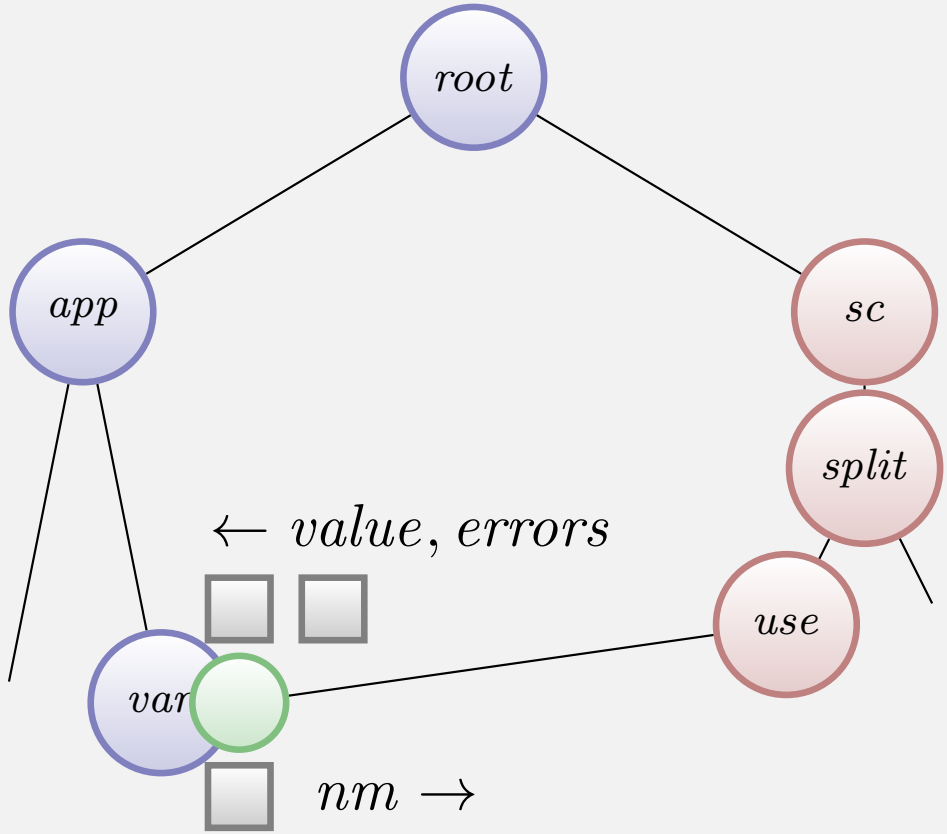
Towards Attributed Trees as Components (2)



Instantiation and Exchange



Instantiation and Exchange



Nested AG Code

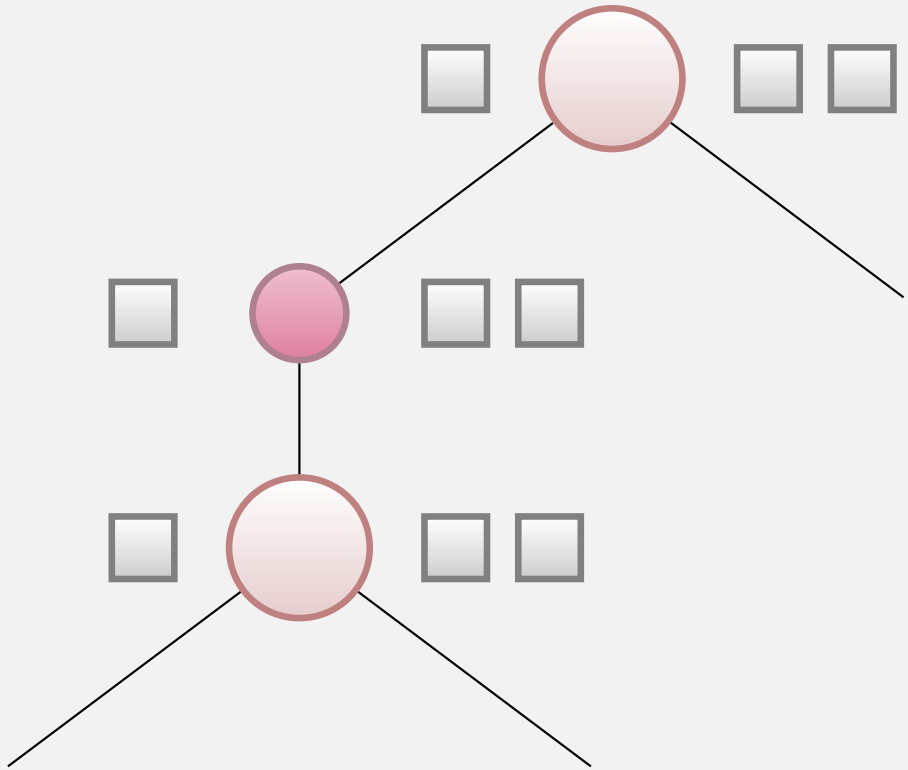
```
itf NameUse α        visit supply   syn nm     :: String
                     visit outcome  inh value  :: α
                                    inh errors :: [String]

sem_Name_Use extTree =
  sem :: Name α
    child ext :: NameUse α = extTree
    loc.mbVal = lookup ext.nm lhs.finEnv
    ext.value  = maybe ⊥ id loc.mbVal
    ext.errors = if isNothing loc.mbVal then ["missing: " ++ ext.nm]

sem_Expr_Var nm =
  sem :: Expr
    lhs.nmTree = sem_Name_Use $
                       sem :: NameUse Type   -- nested sem
                         lhs.nm    = nm
                         loc.errors = lhs.errors
                         loc.value = lhs.value
    ... = loc.errors   -- local attributes
    ... = loc.value    -- shared with nested sem
```

Referentially transparent?

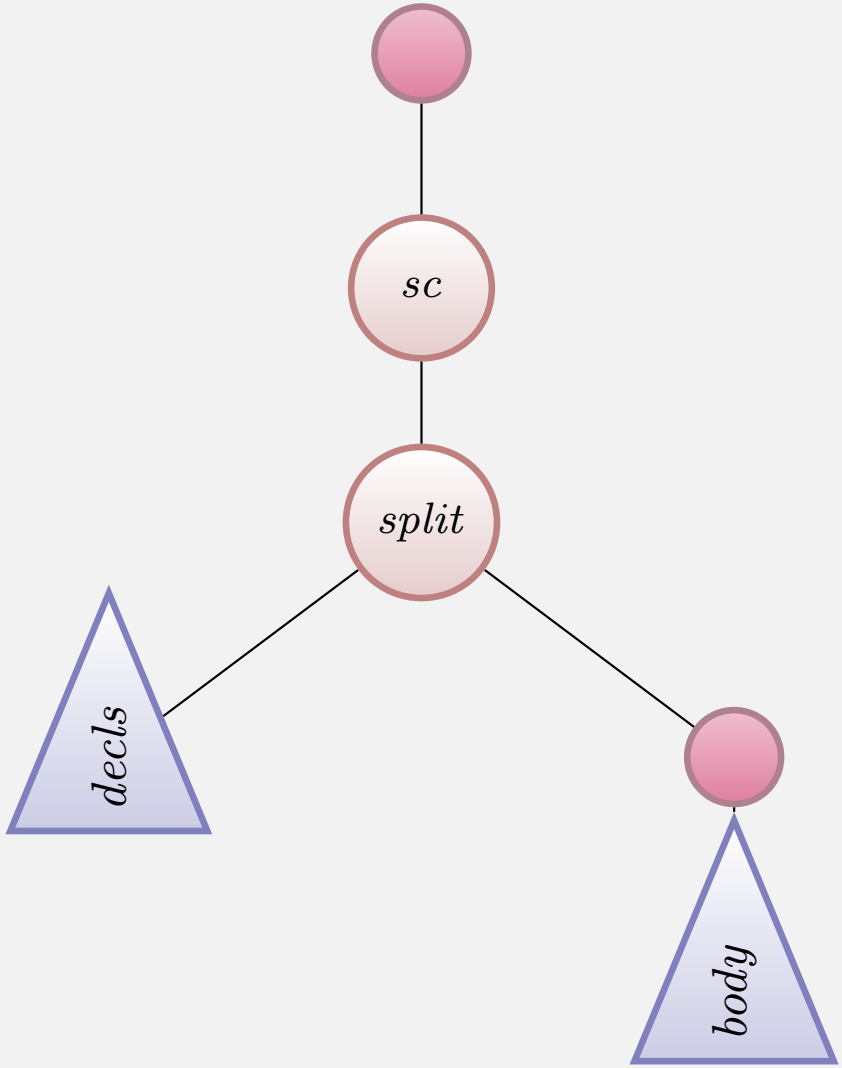
- Unable to guarantee that nested sem is executed exactly once
- We can guarantee referential transparency
  - Requirement: code is "orderable"
  - Then: "an expression for an attribute can be replaced with its value, without affecting the program"
- Referential transparency is important!

Component Extensions

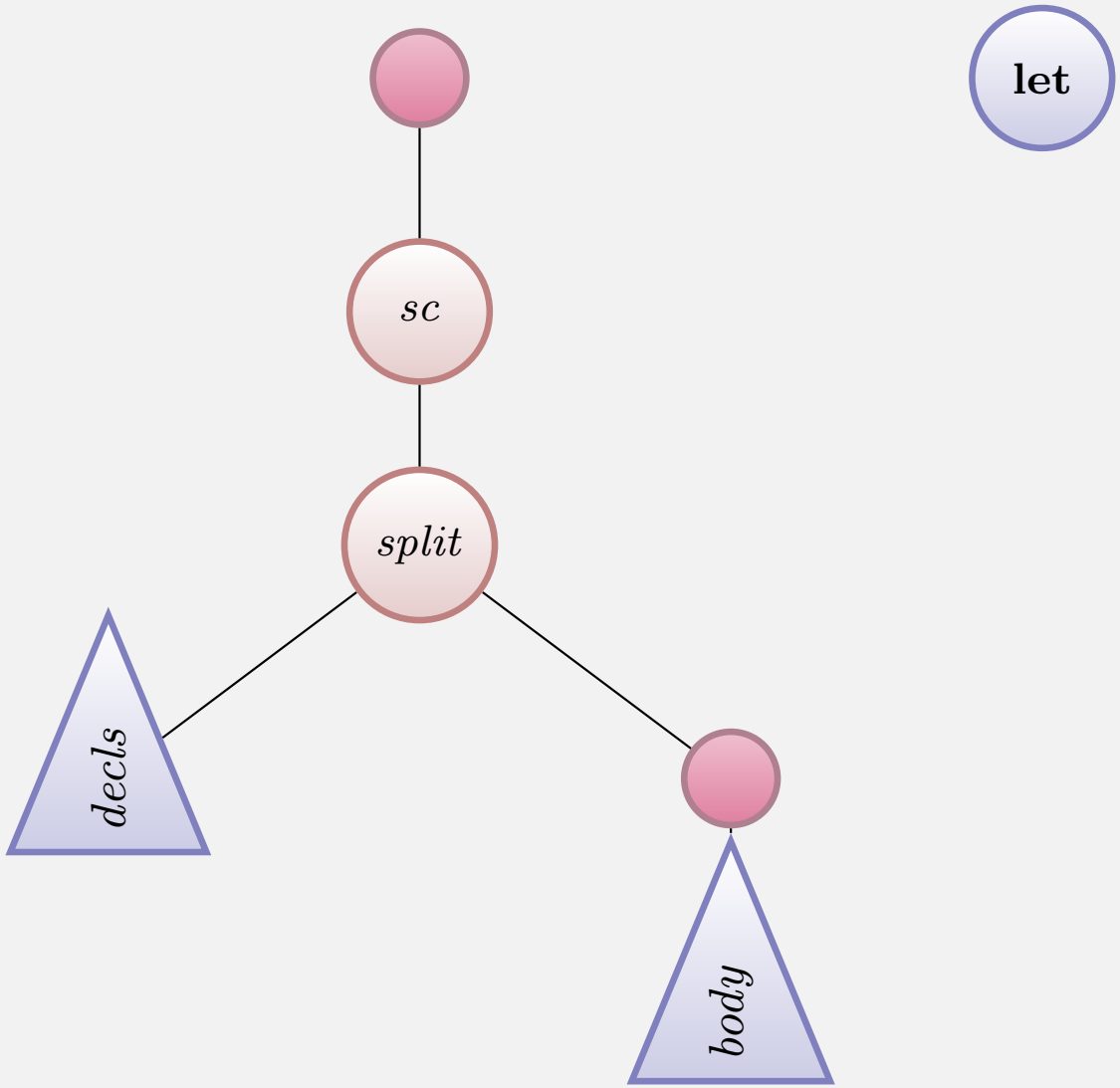
Extend via extra filter nodes



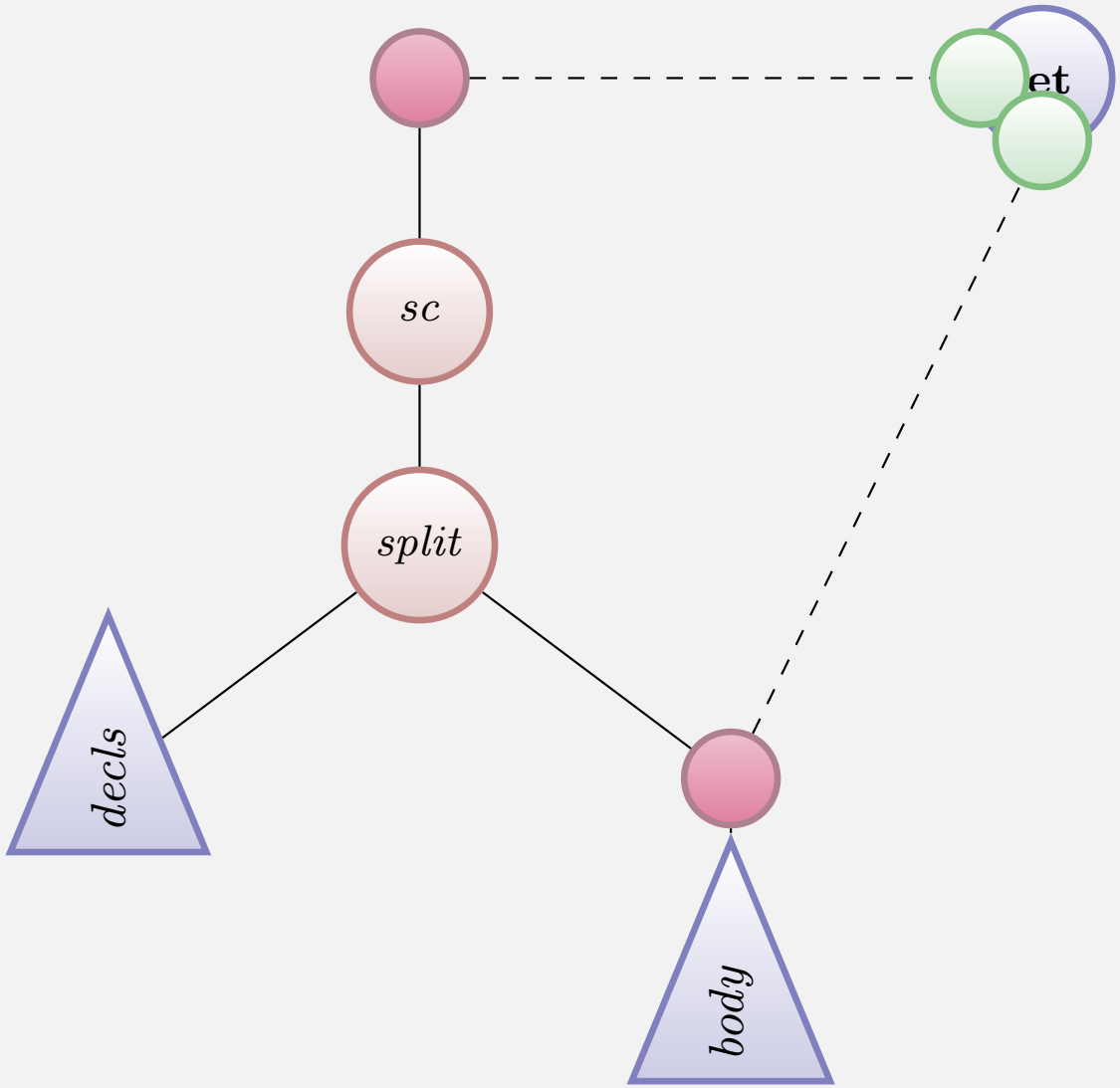
Example: let-bound polymorphism



Example: let-bound polymorphism



Example: let-bound polymorphism



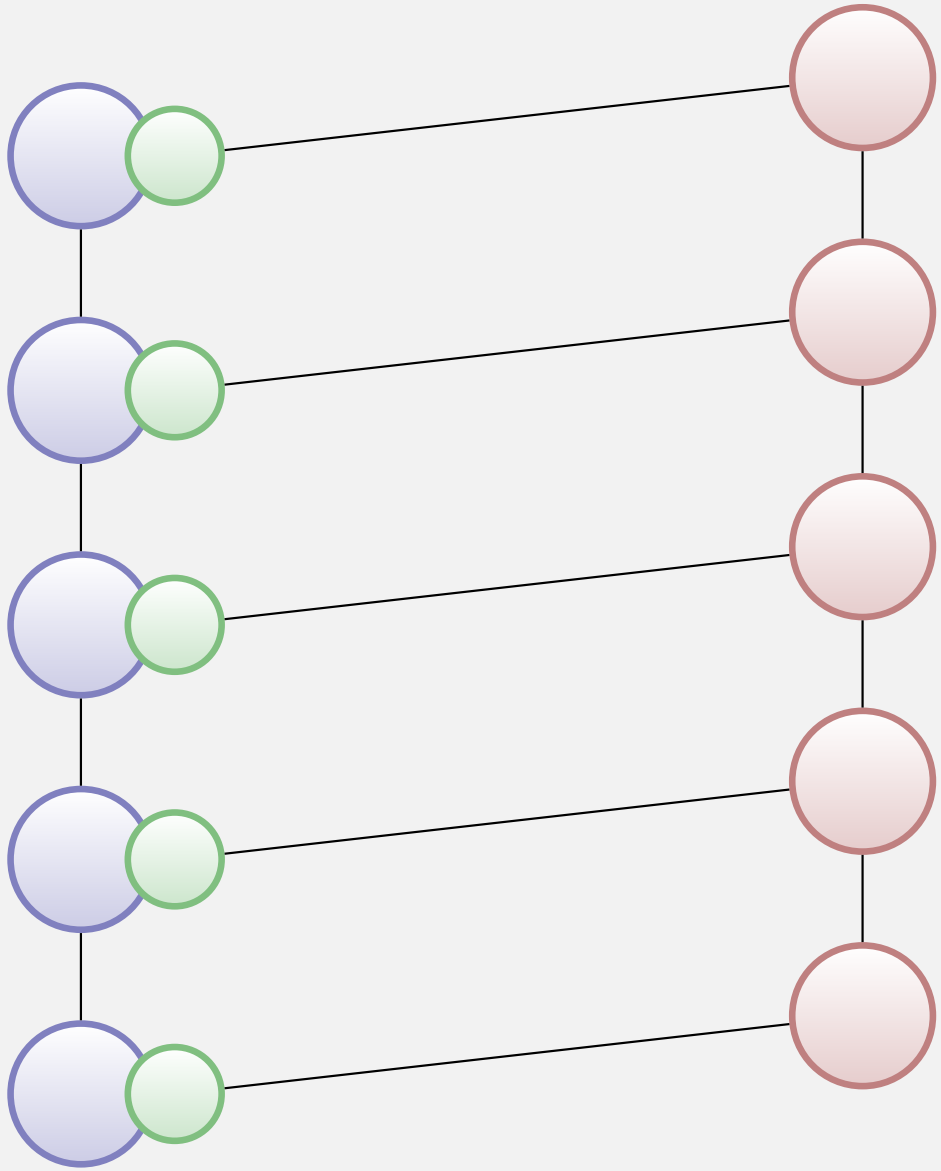
Example: let-bound polymorphism - code

```
sem_Expr_Let ds b
  = sem :: Expr
      lhs.nmTree = loc.semTop $ sem_Name_Scope $
                   sem_Name_Split decls.nmTree $
                   loc.semBody body.nmTree

      loc.semTop t =
        sem :: Name Ty
          child body :: Name Ty = t
          loc.parentEnv = lhs.finEnv

      loc.semBody t =
        sem :: Name Ty
          child body :: Name Ty = t
          body.finEnv = generalize lhs.finEnv
                                   loc.parentEnv
```

AG convenience

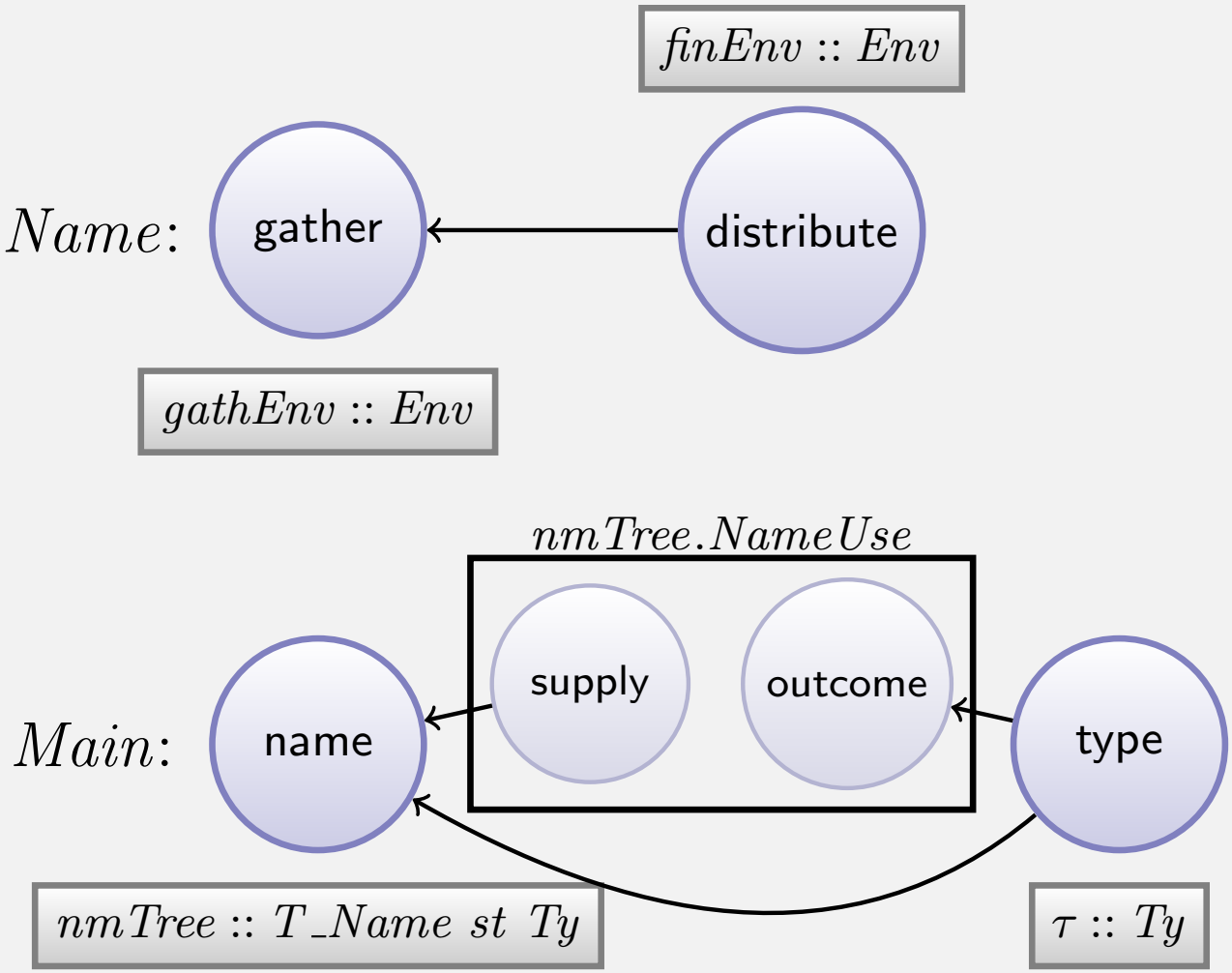
- Override one or two attributes with filter nodes
- Copy rules take care of the rest

Limitations



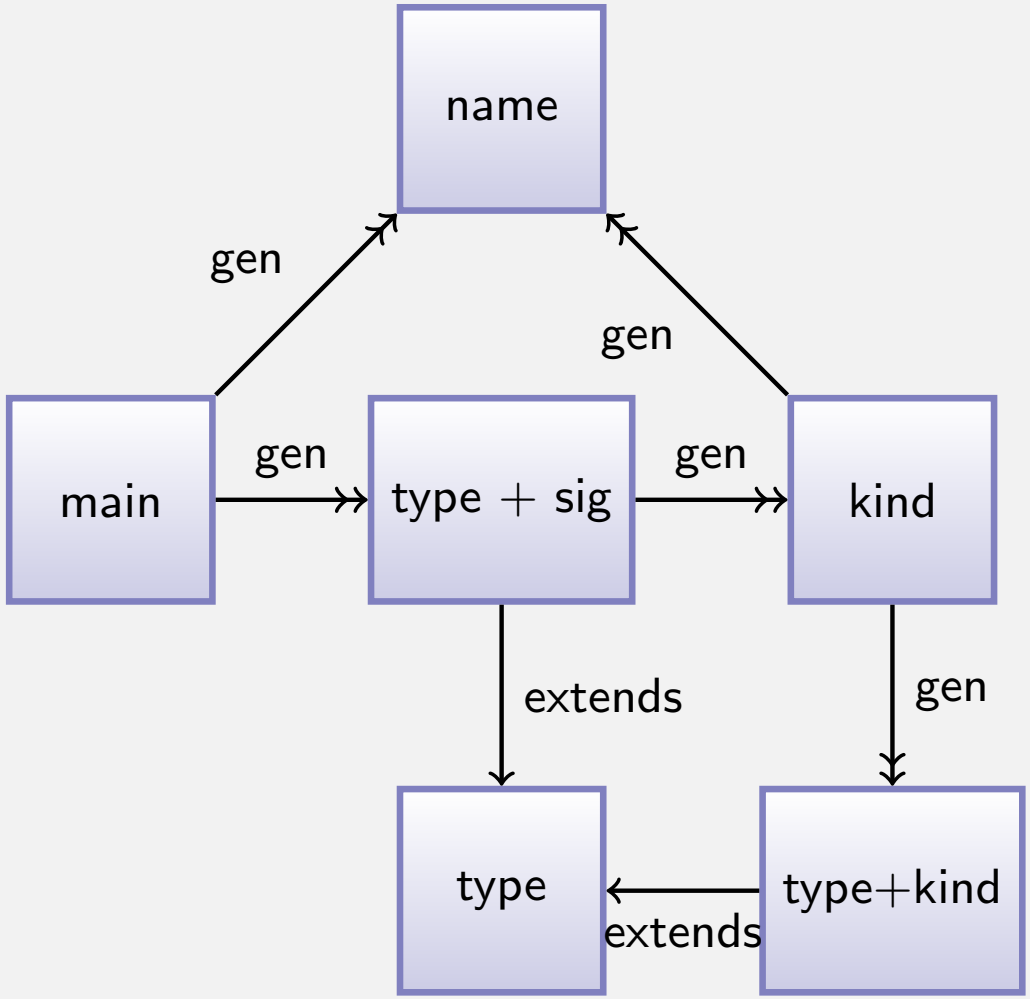
Order Analysis

Interfaces and Dependencies



Conclusions

Type Check Idiom



Discussion

performance views
idioms
generics
proofs functional backend
examples UUAG
tradeoffs
validation usability

Conclusion

Traversal components:

- Produce an attributed tree
- Interface with nodes of this attributed tree
- Extend functionality with filter nodes