Manipulating Trees with Hidden Labels

Luca Cardelli
Philippa Gardner
Giorgio Ghelli

FOSSACS, 2003-04-03

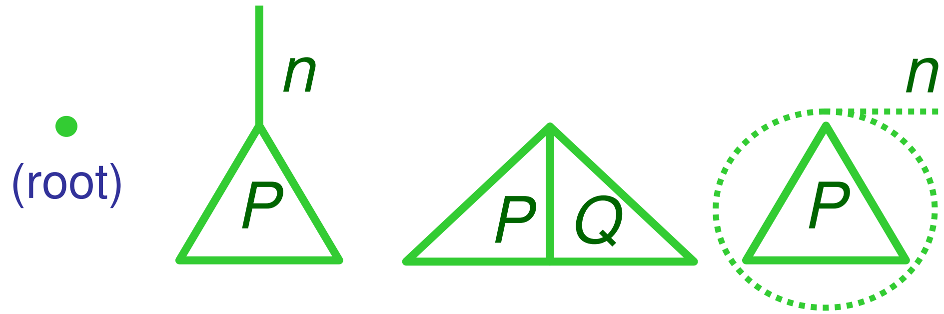
Tree Manipulation

- Well-typed Tree Manipulation
  - ML: pattern matching
  - XDuce: regular patterns + run-time type matching
  - CDuce: ... + negative types, higher order types
  - FreshML: pattern matching with user defined binders
  - This talk: pattern matching with scope extrusion + run-time type matching
- We study two basic techniques:
  - Pattern Matching for hidden labels
  - Transpositions

Data Model: Trees with Hidden Labels

$P,Q ::=$

- 0
- $n[P]$
- $P \mid Q$
- $(\nu n)P$



$P \equiv Q$

P and *Q* represent the same (unordered) tree (up to renaming)

$P \bullet (n \leftrightarrow m)$ (actual) transposition

$n \bullet (n \leftrightarrow m) = m$

$m \bullet (n \leftrightarrow m) = n$

$p \bullet (n \leftrightarrow m) = p \qquad (p \neq n,m)$

$0 \bullet \tau = 0 \qquad (\text{for } \tau = (m \leftrightarrow p))$

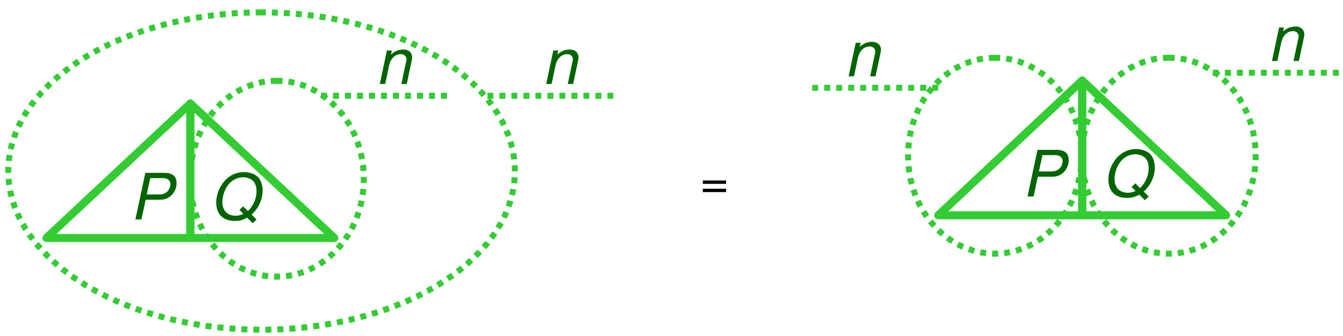
$n[P] \bullet \tau = n \bullet \tau [P \bullet \tau]$

$(P \mid Q) \bullet \tau = (P \bullet \tau) \mid (Q \bullet \tau)$

$((\nu n)P) \bullet \tau = (\nu n \bullet \tau)(P \bullet \tau)$

Tree Equivalence (Structural Congruence)

- $(\nu n)(P \mid (\nu n)Q) \equiv ((\nu n)P) \mid ((\nu n)Q)$

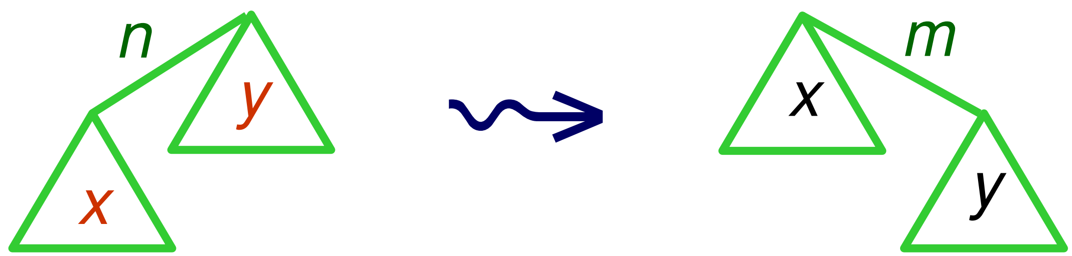


- $(\nu n)m[P] \equiv m[(\nu n)P]$ if $n \neq m$

Ex: Matching Public Labels

- **match** *t* **as** $(n[x{:}\mathbf{T}] \mid y{:}\mathbf{T})$ **then** $(x \mid m[y])$

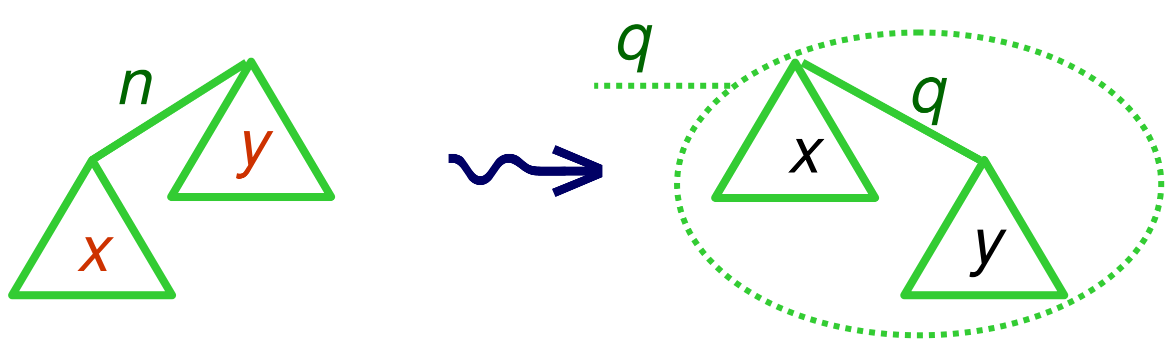
red = binding occurrences



public to public

$$: (n[\mathbf{T}] \mid \mathbf{T}) \rightarrow (\mathbf{T} \mid m[\mathbf{T}])$$

- **match** *t* **as** $(n[x{:}\mathbf{T}] \mid y{:}\mathbf{T})$ **then** $(\nu z)(x \mid z[y])$

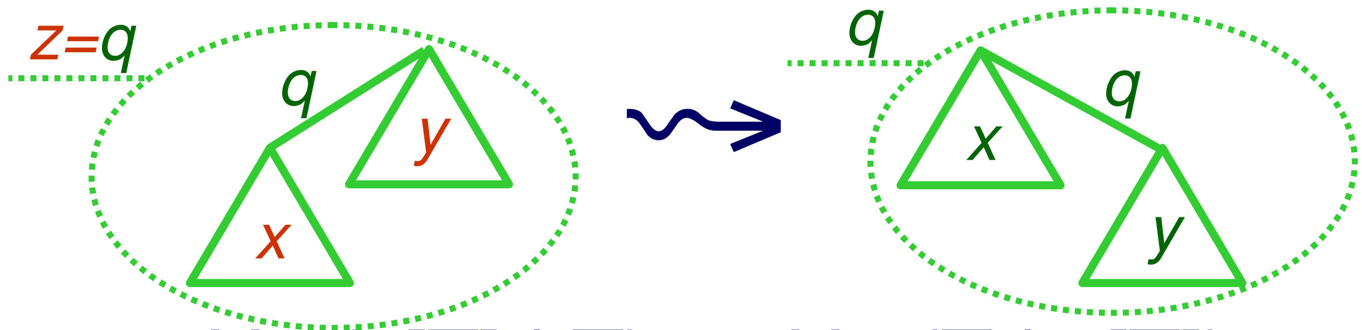


public to private

$$: (n[\mathbf{T}] \mid \mathbf{T}) \rightarrow \mathsf{H}z.(\mathbf{T} \mid z[\mathbf{T}])$$

Ex: Matching Private Labels

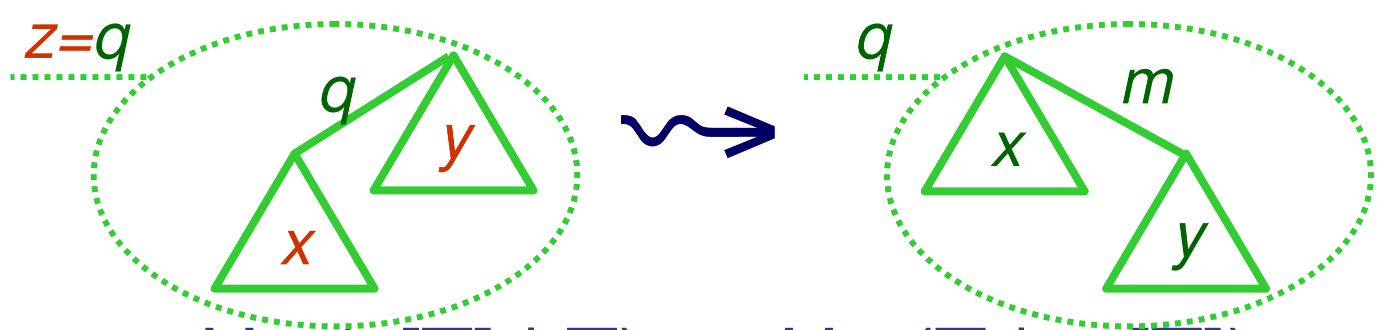
- **match** *t* **as** $(\nu z)(z[x{:}\mathbf{T}] \mid y{:}\mathbf{T})$ **then** $^{(\nu z)}(x \mid z[y])$



private to private

$: \mathsf{H}z.(z[\mathbf{T}] \mid \mathbf{T}) \rightarrow \mathsf{H}z.(\mathbf{T} \mid z[\mathbf{T}])$

- **match** *t* **as** $(\nu z)(z[x{:}\mathbf{T}] \mid y{:}\mathbf{T})$ **then** $^{(\nu z)}(x \mid m[y])$



private to public

$: \mathsf{H}z.(z[\mathbf{T}] \mid \mathbf{T}) \rightarrow \mathsf{H}z.(\mathbf{T} \mid m[\mathbf{T}])$

Formal Transpositions

- Suppose we want to swap *n* with *m* in *t*. We can do it by recursion and pattern matching, but the type would be just $\mathbf{T} \to \mathbf{T}$.
- We want a more informative typing:

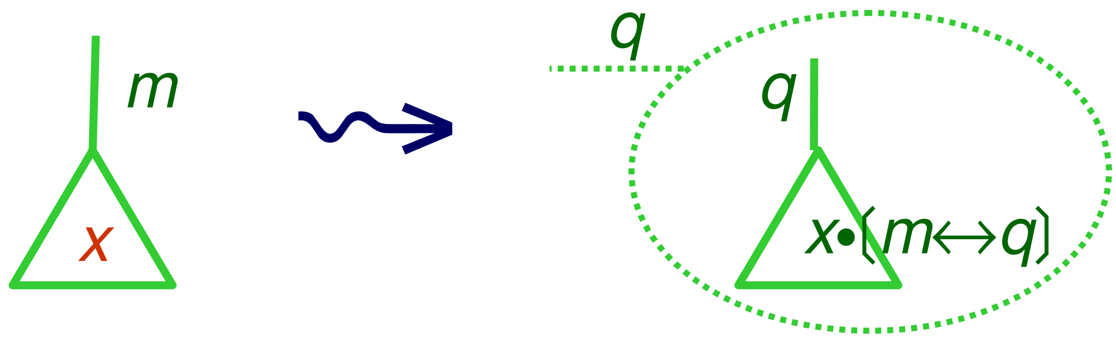
$$\lambda x{:}m[\mathbf{T}].\ x\lbag m \leftrightarrow n \rbag$$
$$: m[\mathbf{T}] \to m[\mathbf{T}]\lbag m \leftrightarrow n \rbag$$
$$= m[\mathbf{T}] \to n[\mathbf{T}]$$

- For this, we need *transposition types*, $A\lbag m \leftrightarrow n \rbag$, and a notion of *transposition equivalence* over types.

Ex: Transposition and Constant Types

- $\lambda x{:}m[\mathbf{T}].\ (\nu z)\ x\llparenthesis m \leftrightarrow z \rrparenthesis$

  replace public m with private $[\![z]\!]$ in input $[\![x]\!]$ of shape $m[P]$



$: m[\mathbf{T}] \rightarrow \mathrm{H}z.\ m[\mathbf{T}]\llparenthesis m \leftrightarrow z \rrparenthesis$

the “obvious” syntax-driven type

$= \ldots\ \mathrm{H}z.\ z[\mathbf{T}\llparenthesis m \leftrightarrow z \rrparenthesis]$

distributing, and swapping m with $[\![z]\!]$

$= \ldots\ \mathrm{H}z.\ z[\mathbf{T}]$

permuting two names in the set of all data, **T**, gives **T**.

$= m[\mathbf{T}] \rightarrow \mathrm{H}z.z[\mathbf{T}]$

Ex: Transposition and Dependent Types

- $\lambda w{:}\mathbf{N}.\ \lambda x{:}w[\mathbf{T}].\ (\nu z)\ x\llparenthesis m \leftrightarrow z \rrparenthesis$

  replace public $[\![w]\!]$ with private $[\![z]\!]$ in $[\![x]\!]$ of shape $[\![w]\!][P]$

  Turned into a parameter, which might be *m*

  $: \Pi w.\ w[\mathbf{T}] \rightarrow \mathsf{H}z.\ w[\mathbf{T}]\llparenthesis m \leftrightarrow z \rrparenthesis$

  the “obvious” syntax-driven type

  $= \ldots\ \mathsf{H}z.\ w\llparenthesis m \leftrightarrow z \rrparenthesis[\mathbf{T}\llparenthesis m \leftrightarrow z \rrparenthesis]$

  distributing; swapping $[\![w]\!]$ with $[\![z]\!]$ cannot be eliminated

  $= \ldots\ \mathsf{H}z.\ w\llparenthesis m \leftrightarrow z \rrparenthesis[\mathbf{T}]$

  permuting two names in the set of all data, **T**, gives **T**.

  $= \Pi w.\ w[\mathbf{T}] \rightarrow \mathsf{H}z.\ w\llparenthesis m \leftrightarrow z \rrparenthesis[\mathbf{T}]$

  no further simplification because the $[\![w]\!]$ passed as a parameter can in fact be *m*.

New typing technology

- Semistructured types
  - From tree automata theory
  - From spatial logic
- Freshness
  - Fresh quantifier
  - Specialized Hiding quantifier
    - Fresh quantifier + name restriction operator
  - Typing contexts carrying names and freshness info
- Transpositions
- Name-dependent types
  - Name expressions (name constants or name variables, or name transpositions).

Syntax

- Name Expressions:

  $\mathcal{N},\mathcal{M} ::= \; x,\; n,\; \mathcal{N}\langle\mathcal{M}\leftrightarrow\mathcal{M}\rangle$

- Terms:

  $t,u,v ::= \; 0,\; \mathcal{N}[t],\; t|t,\; (\nu x)t,\; t\langle\mathcal{M}\leftrightarrow\mathcal{M}\rangle,\; t?(x{:}\mathcal{A}).u,v,$

  $t\div(\mathcal{N}[y{:}\mathcal{A}]).u,\; t\div(x{:}\mathcal{A}|y{:}\mathcal{B}).u,\; t\div((\nu x)y{:}\mathcal{A}).u,$

  $x,\; \mathcal{N},\; \lambda x{:}\mathcal{F}.t,\; t(u)$

  Trees — Run-time type test — Pattern matching — Lambda

- Low Types:

  $\mathcal{A},\mathcal{B} ::= \; \mathbf{0},\; \mathcal{N}[\mathcal{A}],\; \mathcal{A}|\mathcal{B},\; \mathsf{H}x.\mathcal{A},\; ©\mathcal{N},$

  $\mathbf{F},\; \mathcal{A}\wedge\mathcal{B},\; \mathcal{A}\Rightarrow\mathcal{B},\; \mathcal{A}\langle\mathcal{M}\leftrightarrow\mathcal{M}\rangle$

  Tree types — Transposition types — Propositional types

- High types:

  $\mathcal{F},\mathcal{G},\mathcal{H} ::= \; \mathcal{A},\; \mathbf{N},\; \mathcal{F}\rightarrow\mathcal{G},\; \Pi x.\mathcal{G}$

Satisfaction (Tree Types)

$P \vDash_T \mathcal{A}$ for trees P and tree types $\mathcal{A}$

$P \vDash \mathbf{F}$	*never*	$(\mathbf{T} \triangleq \mathbf{F} \Rightarrow \mathbf{F})$
$P \vDash \mathcal{A} \wedge \mathcal{B}$	$\triangleq$	$P \vDash \mathcal{A} \wedge P \vDash \mathcal{B}$
$P \vDash \mathcal{A} \Rightarrow \mathcal{B}$	$\triangleq$	$P \vDash \mathcal{A} \Rightarrow P \vDash \mathcal{B}$
$P \vDash \mathbf{0}$	$\triangleq$	$P \equiv \mathbf{0}$
$P \vDash \mathcal{N}[\mathcal{A}]$	$\triangleq$	$\exists n,P'.\ n \vDash \mathcal{N} \wedge P \equiv n[P'] \wedge P' \vDash \mathcal{A}$
$P \vDash \mathcal{A} \mid \mathcal{B}$	$\triangleq$	$\exists P',P''.\ P \equiv P' \mid P'' \wedge P' \vDash \mathcal{A} \wedge P'' \vDash \mathcal{B}$
$P \vDash \mathrm{H}x.\mathcal{A}$	$\triangleq$	$\exists n,P'.\ n \notin na(\mathcal{A}) \wedge P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A}\{x \leftarrow n\}$
$P \vDash \copyright\mathcal{N}$	$\triangleq$	$\exists n.\ n \vDash \mathcal{N} \wedge n \in fn(P)$
$P \vDash \mathcal{A}(\mathcal{M}_1 \leftrightarrow \mathcal{M}_2)$	$\triangleq$	$\exists m_1,m_2.\ m_1 \vDash \mathcal{M}_1 \wedge m_2 \vDash \mathcal{M}_2 \wedge P \bullet (m_1 \leftrightarrow m_2) \vDash \mathcal{A}$

$n \vDash_N \mathcal{N}$ for names n and name expressions $\mathcal{N}$

$n \vDash m$	$\triangleq$	$n = m$
$n \vDash \mathcal{N}(\mathcal{M}_1 \leftrightarrow \mathcal{M}_2)$	$\triangleq$	$\exists m_1,m_2.\ m_1 \vDash \mathcal{M}_1 \wedge m_2 \vDash \mathcal{M}_2 \wedge n \bullet (m_1 \leftrightarrow m_2) \vDash \mathcal{N}$

Satisfaction (High Types)

$F \vDash_H \mathcal{G}$ for (high) values F and (high) types $\mathcal{G}$

$$F \vDash \mathbf{N} \quad \triangleq \quad \exists \mathcal{N}.\ F \vDash_N \mathcal{N}$$

$$F \vDash \mathcal{A} \quad \triangleq \quad F \vDash_T \mathcal{A}$$

$$F \vDash \mathcal{G} \rightarrow \mathcal{H} \quad \triangleq \quad \mathcal{G} \neq \mathbf{N} \wedge F = \langle \rho, z, t \rangle \wedge \forall G,H.\ (G \vDash \mathcal{G} \wedge t \Downarrow_{\rho[z \leftarrow G]} H) \Rightarrow H \vDash \mathcal{H}$$

$$F \vDash \Pi x.\ \mathcal{H} \quad \triangleq \quad F = \langle \rho, z, t \rangle \wedge \forall n,H.\ t \Downarrow_{\rho[z \leftarrow n]} H \Rightarrow H \vDash \mathcal{H}\{x \leftarrow n\}$$

Closure

Operational Reduction

- Values F are either names, trees, or closures.
- Closures are triples stack-parameter-body.
- Stacks ρ map variables to high values.

- Closures have function type $\mathcal{G} \rightarrow \mathcal{H}$ if the input type $\mathcal{G}$ is *not* the type of names **N**; otherwise they have name-dependent function type $\Pi x.\mathcal{H}$, where x:**N**.

Typing Environments

- They cover free variables *and* free names:
  - $E \vdash t : \mathscr{A}$ e.g.: $n, \mathrm{H}x{:}\mathbf{N}, \forall y{:}\mathbf{N} \vdash n[x[0] \mid y[0]] : \mathbf{T}$
- Environement satisfaction $\rho \vDash E$, requires freshness satisfaction:
  - $x \mapsto n, y \mapsto m \nvDash n, \mathrm{H}x{:}\mathbf{N}, \forall y{:}\mathbf{N}$ (x <u>not</u> fresh for n)
  - $x \mapsto m, y \mapsto m \vDash n, \mathrm{H}x{:}\mathbf{N}, \forall y{:}\mathbf{N}$ (x fresh for n, y arbitrary)
  - $x \mapsto m, y \mapsto m \nvDash n, \forall y{:}\mathbf{N}, \mathrm{H}x{:}\mathbf{N}$ (x <u>not</u> fresh for y)
- Freshness Signatures of Environments
  - $fs(E)$, e.g. $n, \mathrm{H}x, \forall y$
  - Used in type equivalence and apartness

Semantics and Typing

- Restriction

(Red ν)

$$\frac{n \notin na(t,\rho) \qquad t \Downarrow_{\rho[x \leftarrow n]} P}{(\nu x)t \Downarrow_{\rho} (\nu n)P}$$

(Term ν)

$$\frac{E, \text{H}x{:}\mathbf{N} \vdash t : \mathcal{A}}{E \vdash (\nu x)t : \text{H}x.\mathcal{A}}$$

Freshness signature, H(fresh) or ∀, for vars of type **N**

- Restriction Match

"pull" a restriction

(Red ÷ν)

$$\frac{n \notin na(t,\mathcal{A},u,\rho) \quad x \neq y \quad t \Downarrow_{\rho\equiv} (\nu n)P \qquad P \vDash \rho[x \leftarrow n](\mathcal{A}) \quad u \Downarrow_{\rho[x \leftarrow n][y \leftarrow P]} Q}{t \div ((\nu x)y{:}\mathcal{A}).u \Downarrow_{\rho} (\nu n)Q}$$

(Term ÷ν)

$$\frac{E \vdash t : \text{H}x.\mathcal{A} \qquad E, \text{H}x{:}\mathbf{N}, y{:}\mathcal{A} \vdash u : \mathcal{B}}{E \vdash t \div ((\nu x)y{:}\mathcal{A}).u : \text{H}x.\mathcal{B}}$$

Run-time type match = satisfaction

Rebind result

Subtyping and Equivalence

- All the term typing rules are nice and syntax-driven. Where is the catch?

(Subsumption)

$$\frac{E \vdash t : \mathcal{F} \qquad E \vdash \mathcal{F} <: \mathcal{G}}{E \vdash t : \mathcal{G}}$$

(Sub Equiv)

$$\frac{E \vdash \mathcal{F} \qquad \mathcal{F} \sim_{fs(E)} \mathcal{G}}{E \vdash \mathcal{F} <: \mathcal{G}}$$

(Sub Tree)

$$\frac{E \vdash \mathcal{A} \qquad E \vdash \mathcal{B} \qquad valid_{fs(E)}(\mathcal{A} \Rightarrow \mathcal{B})}{E \vdash \mathcal{A} <: \mathcal{B}}$$

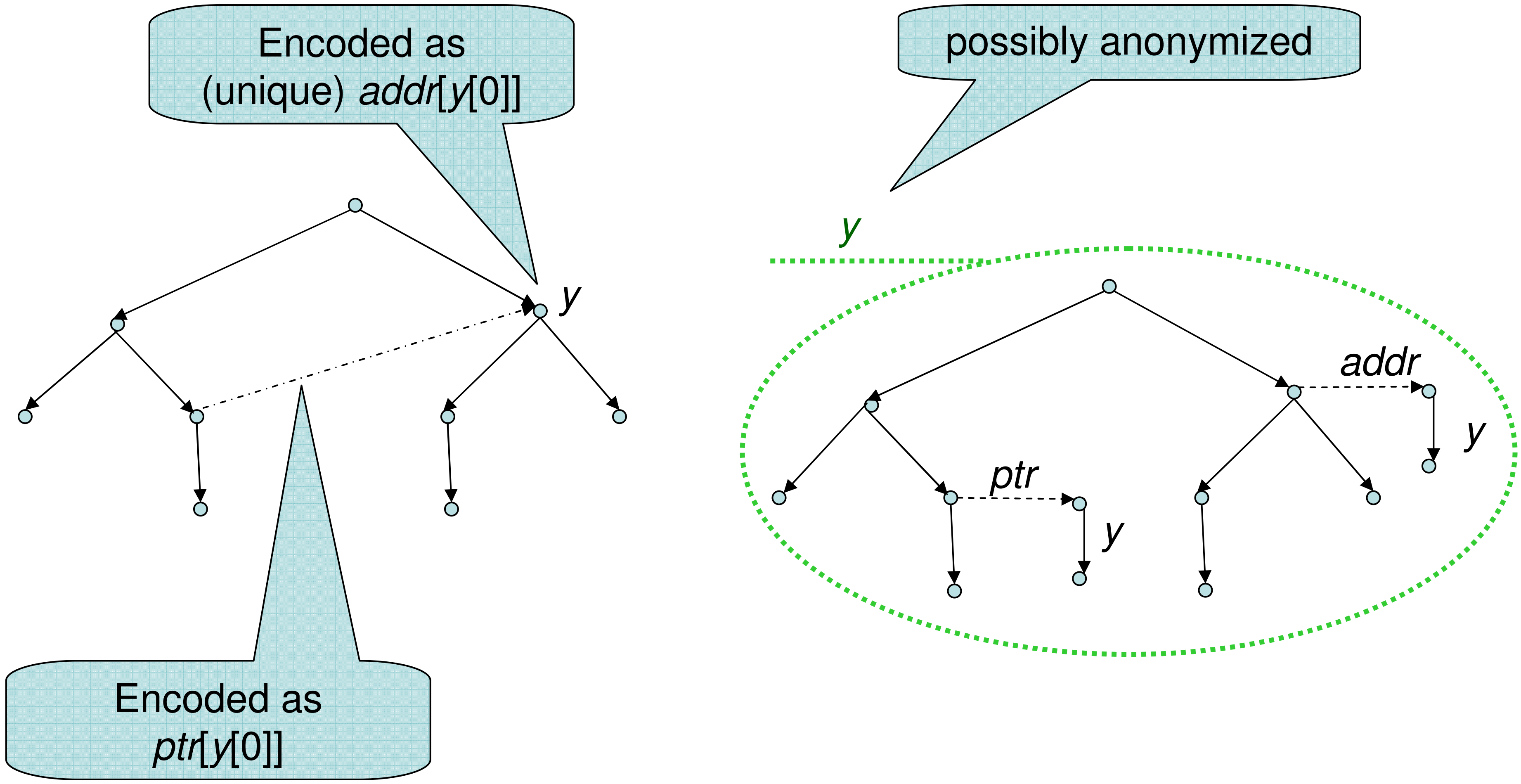
Detailed in the paper, involves *apartness* relation

Left as a "parameter" to the system

(Sub $\rightarrow$), (Sub Π): the usual

Ex: Encoding Local Pointers

- E.g., XML



Ex: Remove Dangling Pointers

- Remove all *ptr*[*y*[0]] that do not have a corresponding *addr*[*y*[0]] in the same tree, whether *y* is public or not.

```
let rec deDangle(x:T): T =
    match x as ((νy)w:©y)
        then (νy)deDangle(w) else r(x,x)
and r(x:T, root:T): T =
    test x as 0 then 0 else
    match x as (y:¬0|w:¬0) then r(y,root) | r(w,root) else
    match x as (ptr[y[0]]) then
        test root as Somewhere(addr[y[0]])
        then ptr[y[0]] else 0   else
    match x as (z[w:T]) then z[r(w,root)] else 0
```

match and strip only "real" restrictions

restrictions preserved in result

remove dangles

search *addr*[*y*[0]] in restriction-stripped root

(Some expected sugar and extensions assumed.)

Conclusions

- New typing technology (freshness, transpositions), possibly of general applicability.
- A different language design than FreshML, but based on the same principles and semantic models.
- A reworking of spatial-logic properties as types. By necessity, these are types of “spatial-like” entities, i.e. data, not computation.
- A pretty nice language for manipulating XML-like data with private labels, and possibly more.
- Decidable subtyping relations (a.k.a. valid entailments in spatial logic) sought.
  - See Calcagno-Cardelli-Gordon.
  - See recent work by Dal Zilio, Lugiez, et. al.
- http://www.luca.demon.co.uk/
  - Fossacs version *with good fonts (!!)*
  - Long version with proofs.