Hypergraph Representation of λ-Terms

Alimjan Yasin and Kazunori Ueda

Waseda University, Tokyo

July 18, 2016

Content

- A technique to represent variable binding (binders) and manipulate such representation
- The λ-calculus as an example
  - $\lambda x.t$ where λx is a binder
  - $\lambda x.x$ the same as $\lambda y.y$
  - Substitution:
    $(\lambda x.x\,y)\,\lambda y.y$
    $\equiv (x\,y)[x := \lambda y.y]$ leads to variable capture without renaming of y
    $\not\equiv \lambda y.y\,y$ Wrong!
    $\equiv (\lambda z.z)\,y$ Right!
    $\equiv y$
- Many formal systems involve variable binding, such as programs, logical formulas, types and proofs

Existing techniques for representing binders

Well-known techniques

- de-Bruijn representation
  - $\lambda x.\lambda y.\lambda z.x\,z\,(y\,z)$ as $\lambda\,\lambda\,\lambda\,3\,1\,(2\,1)$
  - poor readability and shifting operation in substitution
- Higher-order abstract syntax
  - representation is exact
  - implements a variant of λ-calculus and higher-order unification
- Locally nameless representation
  - numbers for bound variables and fresh names for free variables
  - $\lambda x.x\,y$ as $abs(app(bvar\,0)(fvar\,y))$
  - extra operations: variable opening and variable closing,
- Nominal logic
  - freshness constraint $\alpha\#t$ ($atom\ \alpha$ *free for term* t)

Motivation

- Can we define the substitution without side-conditions on such terms which look like λ-terms exactly?

The untyped λ-calculus

syntax

$$t \equiv x \mid \lambda x.t \mid t\ t.$$

β-reduction

$$(\lambda x.m)n \equiv m[x := n]$$

substitution (no side conditions on freeness of variables)

$$x[x := n] \equiv n$$
$$y[x := n] \equiv y\,, (x \neq y)$$
$$(m_1\ m_2)[x := n] \equiv (m_1[x := n])(m_2[x := n])$$
$$(\lambda y.m)[x := n] \equiv \lambda y.(m[x := n])\,, (x \neq y)$$

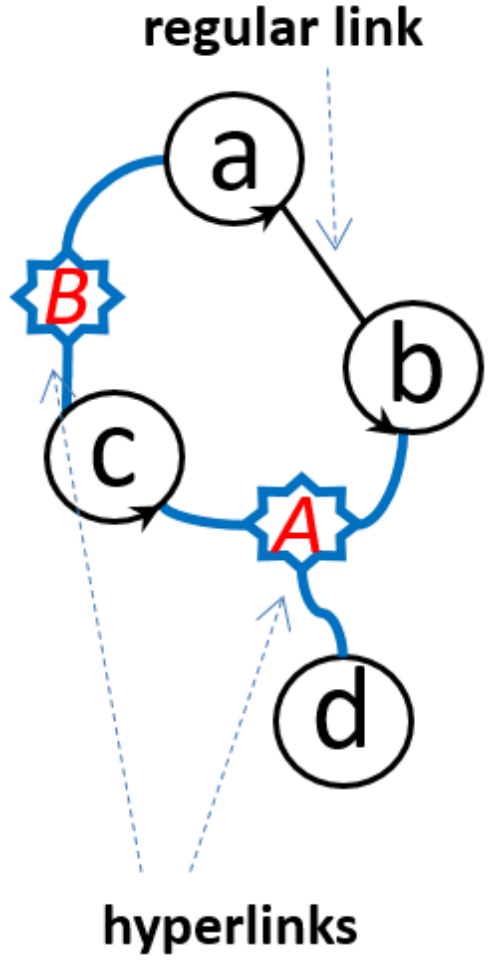
- Barendregt's variable convention is assumed

HyperLMNtal: a hypergraph rewriting language

- Graph elements
  - atom: a node in the graph
  - freshly created links
    - regular link: an edge between two atoms
    - hyperlink: edges to one or more atoms
    - A hyperlink could have an attribute
- Rewrite rules

```
Head :- Guard |  Body.
```

  - type checking in the Guard



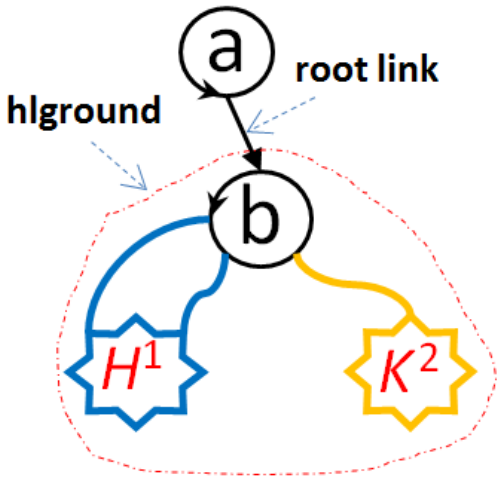
Graph types in HyperLMNtal

- hlink(L,attribute)
  - successful when L is a hyperlink with a particular attribute
- hlground(K,attribute)
  - Identifies a subgraph by the root link K
  - When K is copied (or removed) in a rewrite rule
    - freshly copies (or removes) hyperlinks with matched attribute
    - shares (or removes) hyperlinks with unmatched attribute
    - copies (or removes) atoms

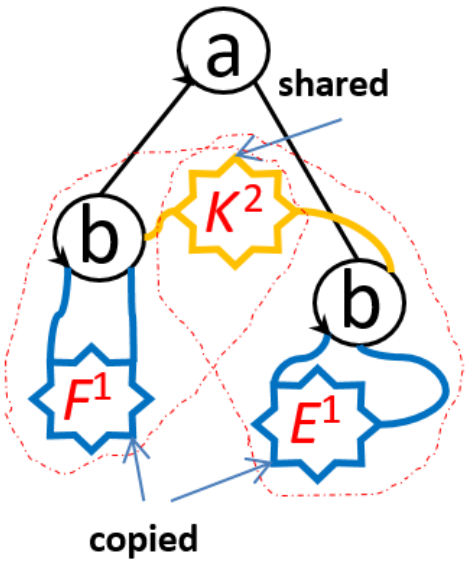
HyperLMNtal: an example

- A term $a(b)$ is the same as $a(A), b(A)$.

```
init.
init       :- new(H,1),new(K,2)   | a(b(H,H,K)).
cp@@ a(A)  :- hlground(A,1)       | a(A,A).
rm@@ a(A)  :- hlground(A,1)       | a.
```



(a) $a(b(H, H, K))$



(b) applying *cp*@@

(c) applying *rm*@@

Representing λ-terms in HyperLMNtal

- Hyperlinks as λ-variables
  - new(H,1) for bound variables
  - new(H,2) for free variables
- Atoms for ..
  - lam(X,Y,R) as $\lambda(x, y)$
  - app(X,Y,R) for $(x\, y)$
  - subs(Z,X,Y,R) for $z[x := y]$
  - Note: R is a regular link

i.e. lam(F,lam(X,app(F,app(F,X))),R)
represents $\lambda f.\lambda x.f\,(f\,x)$.

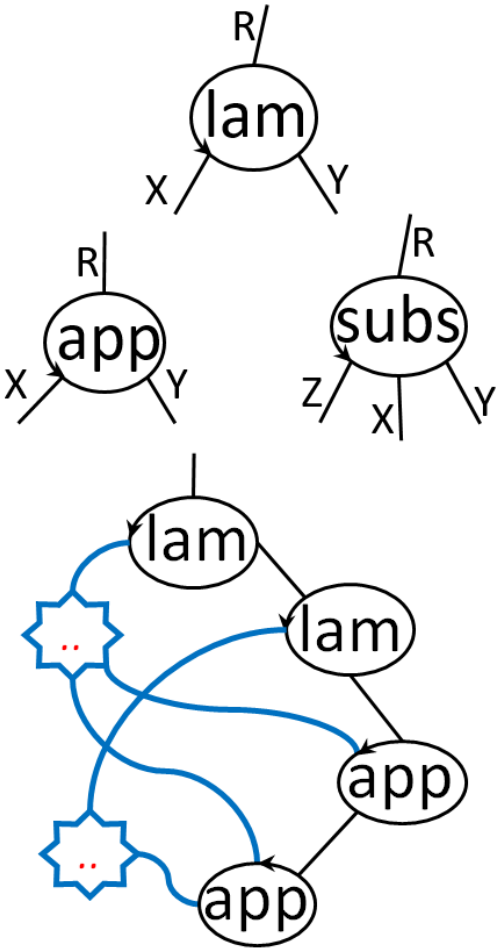


Figure: $\lambda f.\lambda x.f\,(f\,x)$

Defining substitution in HyperLMNtal

- Substitution without side conditions (but with issues)

```
subs(X,X,N,R)             :- hlink(X) | R=N.
subs(X,Y,N,R)             :- hlink(X), X\=Y, hlground(N,1) |
                             R=X.
subs(lam(X,M),Y,N,R)      :- X\=Y |
                             R=lam(X,subs(M,Y,N)).
subs(app(M1,M2),X,N,R) :- hlink(X), hlground(N,1)|
                             R=app(subs(M1,X,N),
                                   subs(M2,X,N)).

beta@@ app(lam(X,A),B,R) :- subs(A,X,B,R).
```

- Exactly corresponds to original definition of substitution

An example of substitution

- *A reduction should look like as shown below*

$(\lambda x.xx)\lambda m\lambda n.mn$

$(xx)[x := \underline{\lambda m\lambda n.mn}]$ **step 2**

$(\underline{\lambda m_1\lambda n_1.m_1n_1})(\underline{\lambda m_2\lambda n_2.m_2n_2})$

$(\lambda n_1.m_1n_1)[m_1 := \lambda m_2\lambda n_2.m_2n_2]$

$\lambda n_1.((m_1n_1)[m_1 := \underline{\lambda m_2\lambda n_2.m_2n_2}])$ **step 5**

$\lambda n_1.((m_1[m_1 := \underline{\lambda m_3\lambda n_3.m_3n_3}])(n_1[m_1 := \underline{\lambda m_4\lambda n_4.m_4n_4}]))$

$\lambda n_1.((\lambda m_3\lambda n_3.m_3n_3)n_1)$

$\lambda n_1.((\lambda n_3.m_3n_3)[m_3 := n_1])$

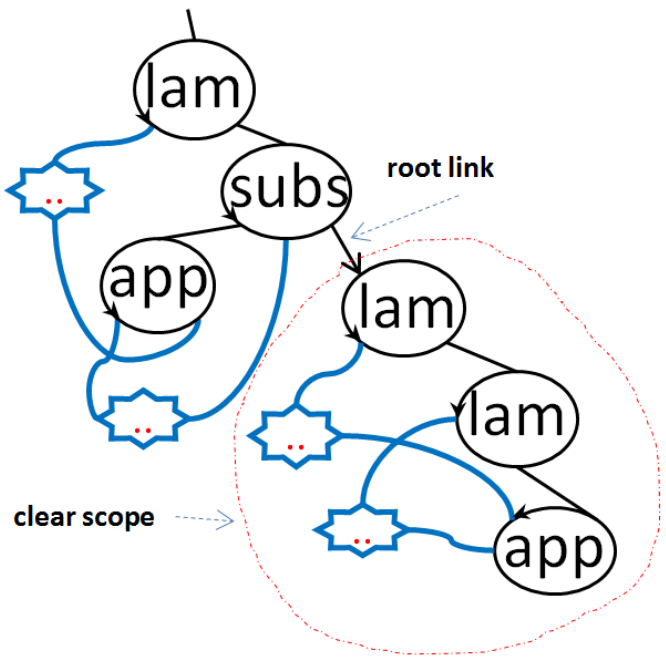
$\lambda n_1.(\lambda n_3.((m_3n_3)[m_3 := \underline{n_1}]))$ **step 9**

$\lambda n_1.(\lambda n_3.((m_3[m_3 := \underline{n_1}])(n_3[m_3 := \underline{n_1}])))$

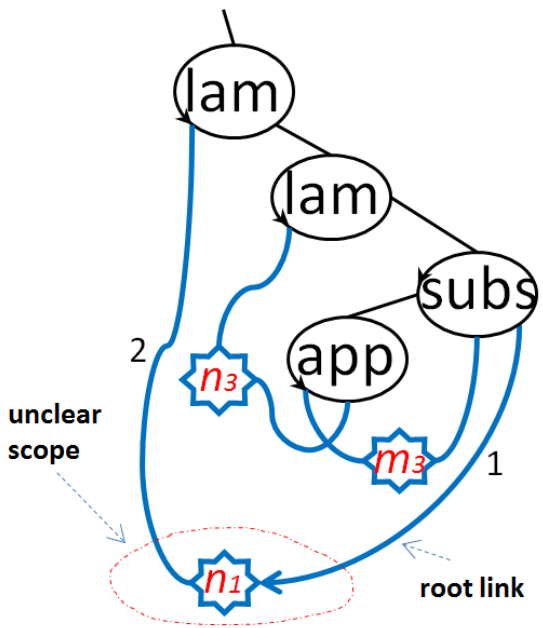
$\lambda n_1.(\lambda n_3.(n_1n_3))$

Issues: finding scope and sharing of bound variables

- $\lambda n_1.(\lambda n_3.((m_3\, n_3)[m_3 := \underline{n_1}]))$ **step 2**, scope is clear
- $\lambda n_1.(\lambda n_3.((m_3\, n_3)[m_3 := \underline{n_1}]))$ **step 9** presents a cycle
- Previous hlground accepts no cycle
- Bound variables are not copied always



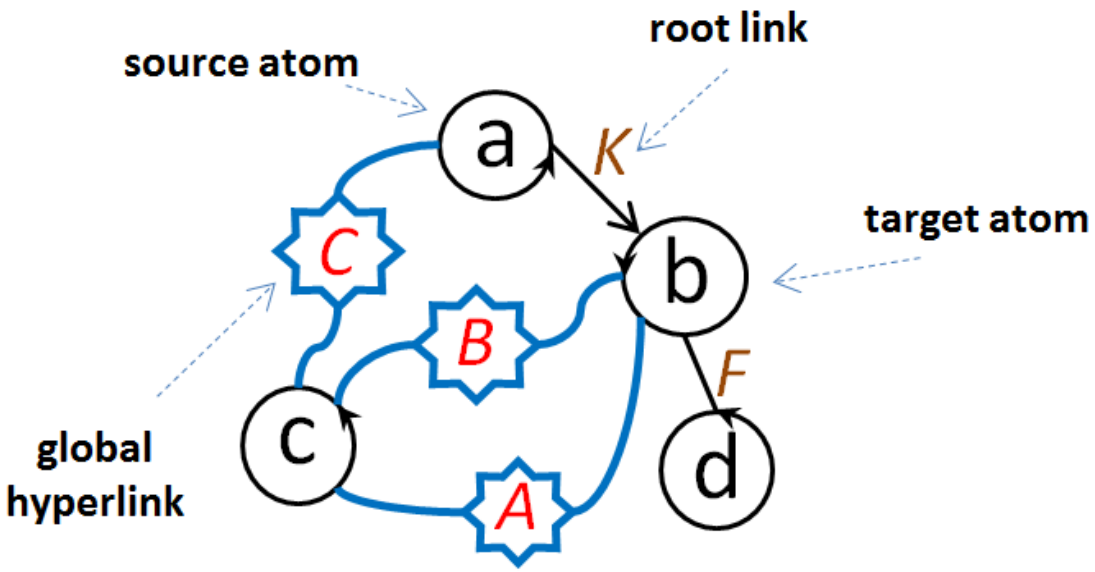
(a) step 2



(b) step 9

Solution (step 1): Global hyperlinks

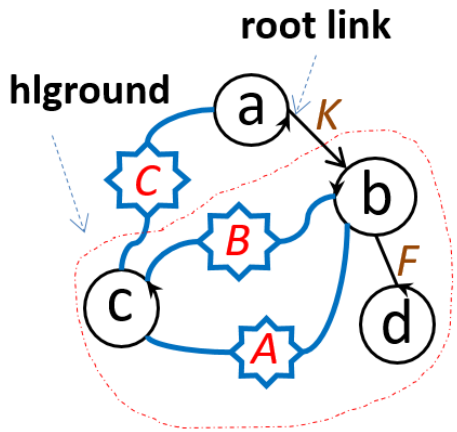
- Define Global hyperlinks for *hlground*(K, *attribute*)
  - Break cycles by cutting hyperlinks (called global hyperlinks)
  - Root link K and global hyperlinks define a subgraph
- Global hyperlinks are s-t min-cut edges
  - Root link defines a source atom and a target atom
  - Find minimum number of hyperlinks whose removal disconnects the source and target atom
  - Never cut regular links



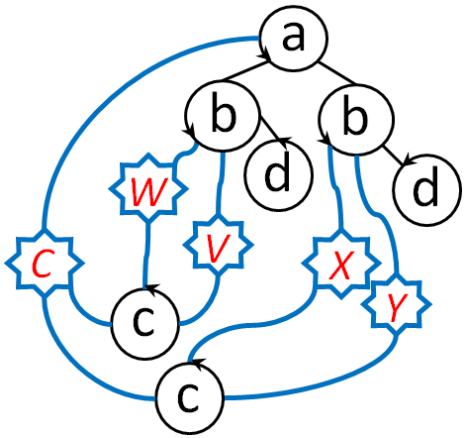
Solution (step 2): hlground extension

- Operation for *hlground*(*K*, *attribute*)
- When *K* is copied (or removed) in a rewrite rule
  - A hyperlink with matched attribute
    - freshly copied (or removed) if it is not a global hyperlink
    - shared (or partly removed) if it is a global hyperlink
  - unmatched attributed hyperlinks are always shared

```
cp@@ a(A,B):- hlground(A)|a(A,A,B).
rm@@ a(A,B):- hlground(A)|a(B).
```



(a) a term



(b) applying *cp*@@

(c) applying *rm*@@

Conclusion

- Straightforward encoding of λ-terms
- Straightforward encoding of substitution
  - free from side-conditions

```
subs(X,X,N,R)             :- hlink(X) | R=N.
subs(X,Y,N,R)             :- hlink(X), X\=Y, hlground(N,1) |
                             R=X.
subs(lam(X,M),Y,N,R)      :- X\=Y |
                             R=lam(X,subs(M,Y,N)).
subs(app(M1,M2),X,N,R)    :- hlink(X), hlground(N,1)|
                             R=app(subs(M1,X,N),
                                   subs(M2,X,N)).
```

- The untyped λ-calculus is encoded and worked

Thanks for your attention!

Substitution through hlground

- Hlground fails if there is a cycle without hyperlinks for the root link, such cycle path is called pure path
- Both rules, *cp*@@ and @@*rm*, cannot rewrite the term $a(b(L), c(L))$
  - because existence of a pure path for the given root link

```
init.
init          :-a(b(L),c(L)).
cp@@ a(A,B):-hlground(A,1)|a(A,A,B).
rm@@ a(A,B):-hlground(A,1)|a(B).
```

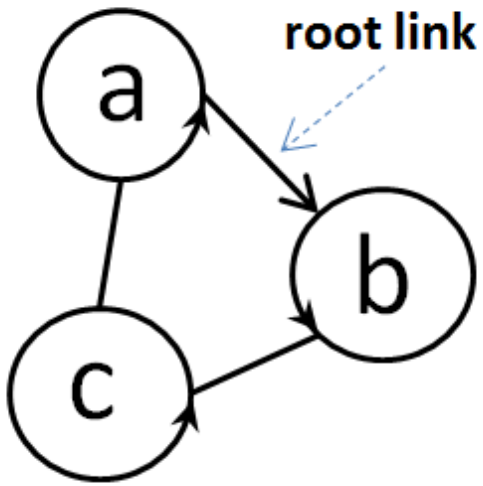


Figure: $a(b(L), c(L))$

- No pure path occurs in hypergraph represented λ-terms for any root link

Solution (step 2): hlground extension

- Operation for *hlground*(K, *attribute*)
- When K is copied (or removed) in a rewrite rule
  - A hyperlink with matched attribute
    - freshly copied (or removed) if it is not a global hyperlink
    - shared (or partly removed) if it is a global hyperlink
  - unmatched attributed hyperlinks
    - always shared