The rewriting calculus

Horatiu Cirstea and Claude Kirchner and Luigi Liquori

In close collaboration with
Clara Bertolissi, Germain Faure, Benjamin Wack

Thanks to the Protheo group@Nancy

ESSLLI Nancy August, 2004

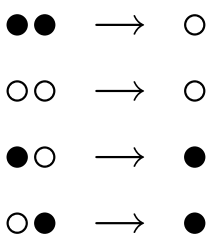
1– Introduction

Mathematics is frequently described as "the science of pattern," a characterisation that makes more sense than most, both of pure mathematics, but also of the ability of mathematics to connect to the world teeming with patterns, symmetries, regularities, and uniformities

Jon Barwise
Lawrence Moss

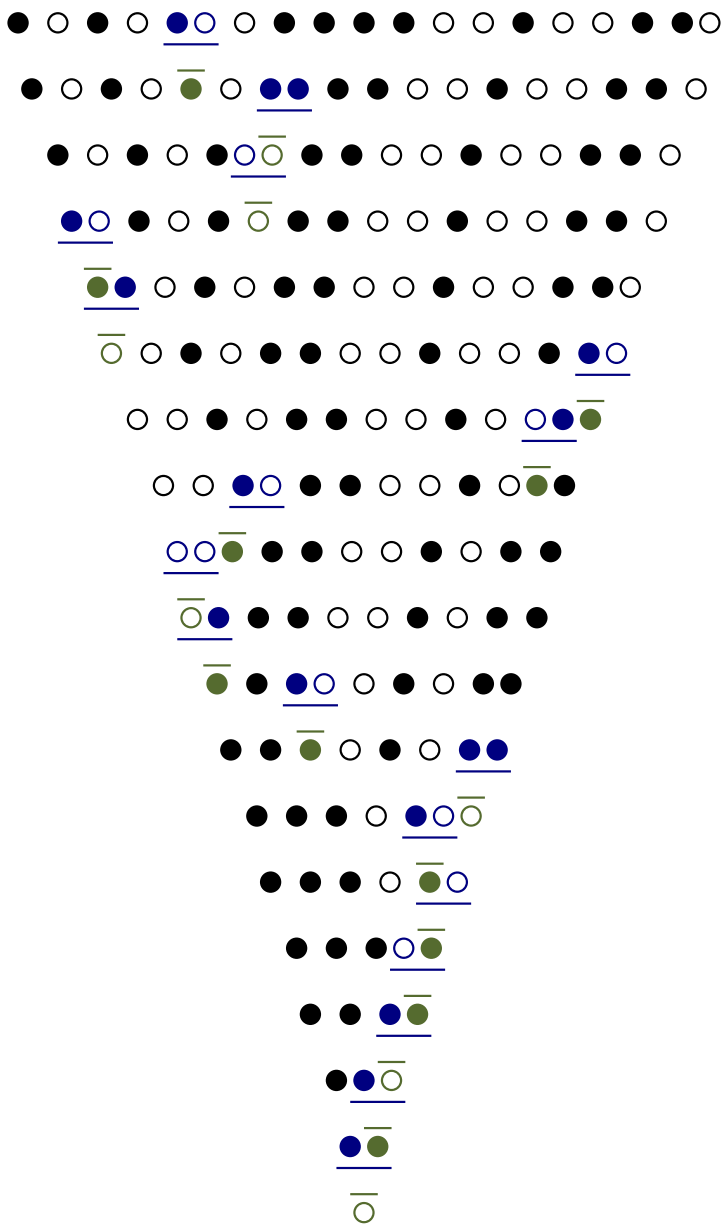
A simple game

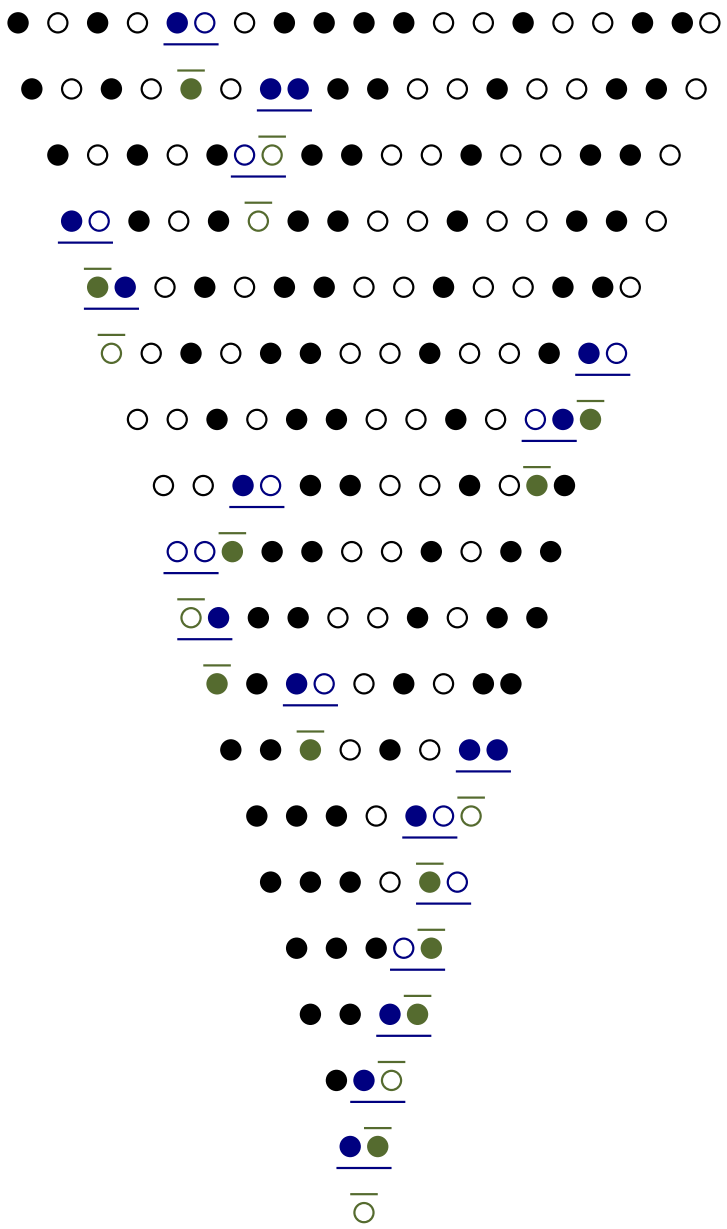
The rules of the game:



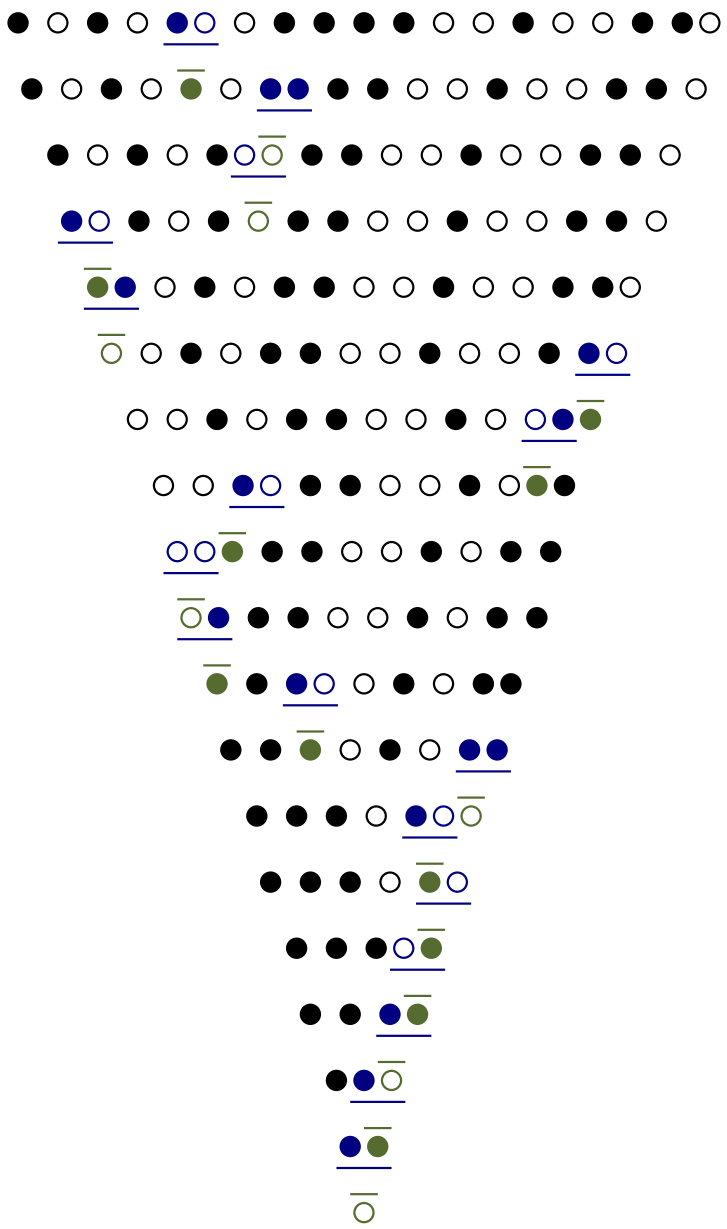
A starting point:

Who wins? (i.e. put the last white)

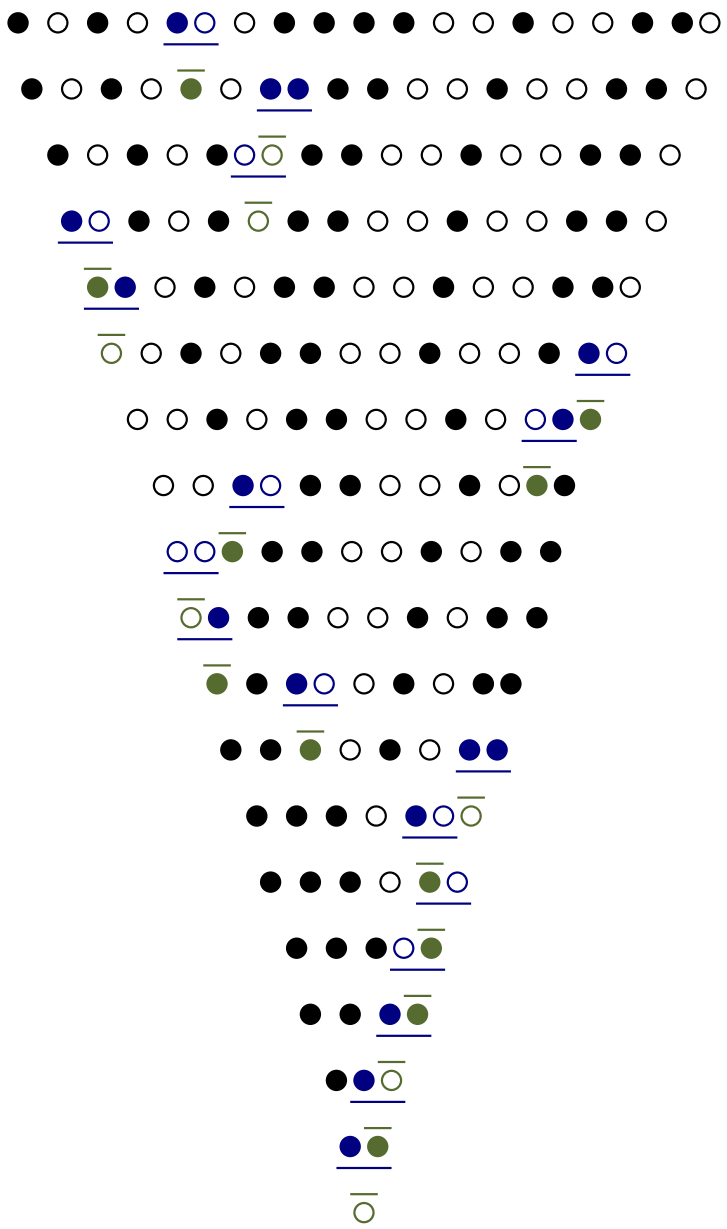




May I always win?



May I always win? Do we get always the same result?



May I always win? Do we get always the same result? Does the game terminate?

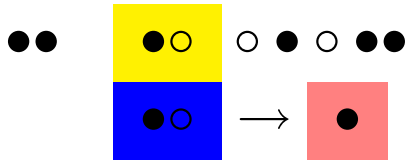
What are the basic operations that have been used?

What are the basic operations that have been used?

1– Matching

The data:

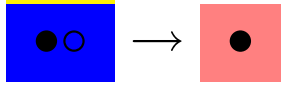
The rewrite rule:



What are the basic operations that have been used?

1– Matching

The data: 

The rewrite rule: 

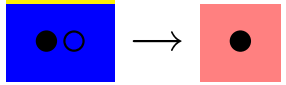
2– Compute what should be substituted

The lefthand side: 

What are the basic operations that have been used?

1– Matching

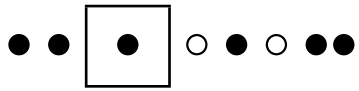
The data: 

The rewrite rule: 

2– Compute what should be substituted

The lefthand side: 

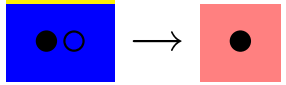
3– Replacement

The new generated data: 

What are the basic operations that have been used?

1– Matching

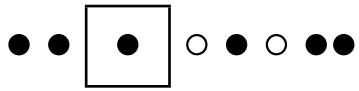
The data: 

The rewrite rule: 

2– Compute what should be substituted

The lefthand side: 

3– Replacement

The new generated data: 

Note that, that last list is a NEW object

Let us do simple math: just addition in Peano arithmetic (1)

Let us do simple math: just addition in Peano arithmetic (1)

How can we compute $2 + 2$?

Let us do simple math: just addition in Peano arithmetic (1)

How can we compute $2 + 2$?

What are the natural?

Let us do simple math: just addition in Peano arithmetic (1)

How can we compute $2 + 2$?

What are the natural?

Objects build from a constant called 0 and a unary operator s (successor).
Some example

Let us do simple math: just addition in Peano arithmetic (1)

How can we compute $2 + 2$?

What are the natural?

Objects build from a constant called 0 and a unary operator s (successor).
Some example
$s(s(s(0)))$ often denoted 3, $s(s(s(s(s(s(s(s(s(s(s(0))))))))))$ often denoted 11

Let us do simple math: just addition in Peano arithmetic (1)

How can we compute $2 + 2$?

What are the natural?

Objects build from a constant called 0 and a unary operator s (successor).
Some example
$s(s(s(0)))$ often denoted 3, $s(s(s(s(s(s(s(s(s(s(s(0))))))))))$ often denoted 11

We can now define a binary operator $+$

Let us do simple math: just addition in Peano arithmetic (1)

How can we compute $2 + 2$?

What are the natural?

Objects build from a constant called 0 and a unary operator s (successor).
Some example
$s(s(s(0)))$ often denoted 3, $s(s(s(s(s(s(s(s(s(s(s(0))))))))))$ often denoted 11

We can now define a binary operator $+$

For example we can write $s(s(0)) + s(s(0))$, $0 + s(s(s(s(s(s(s(s(s(s(s(0))))))))))$

Let us do simple math: just addition in Peano arithmetic (1)

How can we compute $2 + 2$?

What are the natural?

Objects build from a constant called 0 and a unary operator s (successor).
Some example
$s(s(s(0)))$ often denoted 3, $s(s(s(s(s(s(s(s(s(s(s(0))))))))))$ often denoted 11

We can now define a binary operator $+$

For example we can write $s(s(0)) + s(s(0))$, $0 + s(s(s(s(s(s(s(s(s(s(0))))))))))$

how do we express the $result$?

Let us do simple math: just addition in Peano arithmetic (1)

How can we compute $2 + 2$?

What are the natural?

Objects build from a constant called 0 and a unary operator s (successor).
Some example
$s(s(s(0)))$ often denoted 3, $s(s(s(s(s(s(s(s(s(s(s(0))))))))))$ often denoted 11

We can now define a binary operator $+$

For example we can write $s(s(0)) + s(s(0))$, $0 + s(s(s(s(s(s(s(s(s(s(s(0))))))))))$

how do we express the $result$?

by the way . . . , what is the $result$?

Let us do simple math: just addition in Peano arithmetic (2)

Peano gives a meaning to addition by using the following axioms:

$$\begin{aligned} 0 + x &= x \\ s(x) + y &= s(x + y) \end{aligned}$$

Let us do simple math: just addition in Peano arithmetic (2)

Peano gives a meaning to addition by using the following axioms:

$$\begin{aligned} 0 + x &= x \\ s(x) + y &= s(x + y) \end{aligned}$$

Thus

$$\begin{aligned} s(s(0)) + s(s(0)) &= s(s(0) + s(s(0)) \\ &= s(s(0 + s(s(0)))) \\ &= s(s(s(s(0)))) \\ &= s(0) + s(s(s(0))) \\ &= 0 + 0 + 0 + s(s(s(s(0)))) \\ &= \ldots \end{aligned}$$

Let us do simple math: just addition in Peano arithmetic (2)

Peano gives a meaning to addition by using the following axioms:

$$\begin{aligned} 0 + x &= x \\ s(x) + y &= s(x + y) \end{aligned}$$

Thus

$$\begin{aligned} s(s(0)) + s(s(0)) &= s(s(0) + s(s(0))) \\ &= s(s(0 + s(s(0)))) \\ &= s(s(s(s(0)))) \\ &= s(0) + s(s(s(0))) \\ &= 0 + 0 + 0 + s(s(s(s(0)))) \\ &= \ldots \end{aligned}$$

is there a *better* result?

Let us do simple math: just addition in Peano arithmetic (3)

Let us *compute* a result by turning the equalities into rewrite rules:

$$\begin{aligned} 0 + x &\rightarrow x \\ s(x) + y &\rightarrow s(x + y) \end{aligned}$$

Let us do simple math: just addition in Peano arithmetic (3)

Let us *compute* a result by turning the equalities into rewrite rules:

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x + y) \end{array}$$

Thus $s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0)) \rightarrow s(s(0 + s(s(0)))) \rightarrow s(s(s(s(0))))$

Let us do simple math: just addition in Peano arithmetic (3)

Let us *compute* a result by turning the equalities into rewrite rules:

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x + y) \end{array}$$

Thus $s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0)) \rightarrow s(s(0 + s(s(0)))) \rightarrow s(s(s(s(0))))$

Is this computation *terminating*,

Let us do simple math: just addition in Peano arithmetic (3)

Let us *compute* a result by turning the equalities into rewrite rules:

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x + y) \end{array}$$

Thus $s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0)) \rightarrow s(s(0 + s(s(0)))) \rightarrow s(s(s(s(0))))$

Is this computation *terminating*,

is there always a *result* (e.g. an expression without $+$)

Let us do simple math: just addition in Peano arithmetic (3)

Let us *compute* a result by turning the equalities into rewrite rules:

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x + y) \end{array}$$

Thus $s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0)) \rightarrow s(s(0 + s(s(0)))) \rightarrow s(s(s(s(0))))$

Is this computation *terminating*,

is there always a *result* (e.g. an expression without $+$)

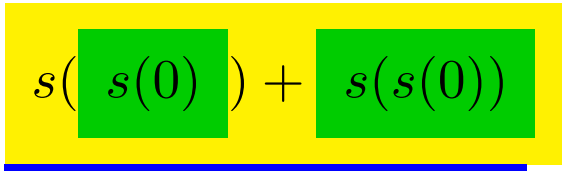
is such a result *unique*? ? ?

What are the basic operations that have been used? (2)

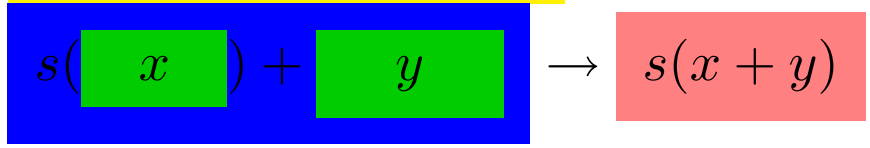
What are the basic operations that have been used? (2)

1– Matching

The data:



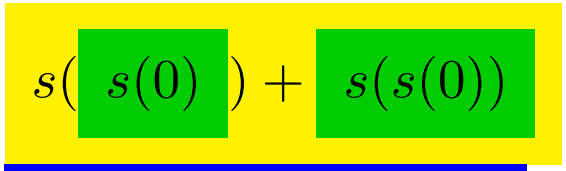
The rewrite rule:



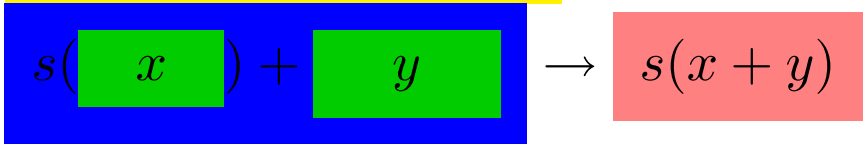
What are the basic operations that have been used? (2)

1– Matching

The data:

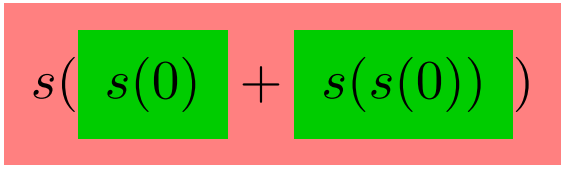


The rewrite rule:



2– Compute what should be substituted

The instanciated lhs:



What are the basic operations that have been used? (2)

1– Matching

The data: $s(\ s(0)\ ) + s(s(0))$

The rewrite rule: $s(\ x\ ) + y \rightarrow s(x+y)$

2– Compute what should be substituted

The instanciated lhs: $s(\ s(0)\ + s(s(0))\ )$

3– Replacement

The new generated data: s(s(0)+s(s(0)))

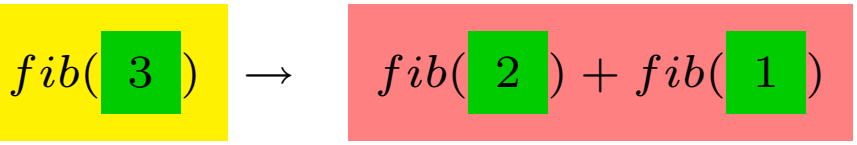
fibonacci

$$
\begin{array}{lrcl}
[\alpha] & fib(0) & \rightarrow & 1 \\
[\beta] & fib(1) & \rightarrow & 1 \\
[\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$

$fib(\ 3\ )$

fibonacci

$$
\begin{array}{lllll}
[\alpha] & fib(0) & \rightarrow & 1 \\
[\beta] & fib(1) & \rightarrow & 1 \\
[\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$



fibonacci

$$
\begin{array}{lllll}
[\alpha] & fib(0) & \rightarrow & 1 \\
[\beta] & fib(1) & \rightarrow & 1 \\
[\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$

$fib(3) \rightarrow fib(2) + fib(1)$

$fib(2) + fib(1)$

fibonacci

$$
\begin{array}{lll}
[\alpha] & fib(0) & \rightarrow \quad 1 \\
[\beta] & fib(1) & \rightarrow \quad 1 \\
[\gamma] & fib(n) & \rightarrow \quad fib(n-1) + fib(n-2)
\end{array}
$$

$$fib(3) \rightarrow fib(2) + fib(1)$$

$$fib(2) + fib(1) \rightarrow fib(2) + 1$$

fibonacci

$$
\begin{array}{lll}
[\alpha] \quad fib(0) & \rightarrow & 1 \\
[\beta] \quad fib(1) & \rightarrow & 1 \\
[\gamma] \quad fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$

$$fib(3) \rightarrow fib(2) + fib(1)$$

$$fib(2) + fib(1) \rightarrow fib(2) + 1$$

$$fib(2) + 1 \rightarrow fib(2) + 1$$

fibonacci

$$
\begin{array}{lll}
[\alpha] \quad fib(0) & \rightarrow & 1 \\
[\beta] \quad fib(1) & \rightarrow & 1 \\
[\gamma] \quad fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$

$$fib(3) \rightarrow fib(2) + fib(1)$$

$$fib(2) + fib(1) \rightarrow fib(2) + 1$$

$$fib(2) + 1 \rightarrow fib(1) + fib(0) + 1$$

fibonacci

$$
\begin{array}{lll}
[\alpha] & fib(0) & \rightarrow \quad 1 \\
[\beta] & fib(1) & \rightarrow \quad 1 \\
[\gamma] & fib(n) & \rightarrow \quad fib(n-1) + fib(n-2)
\end{array}
$$

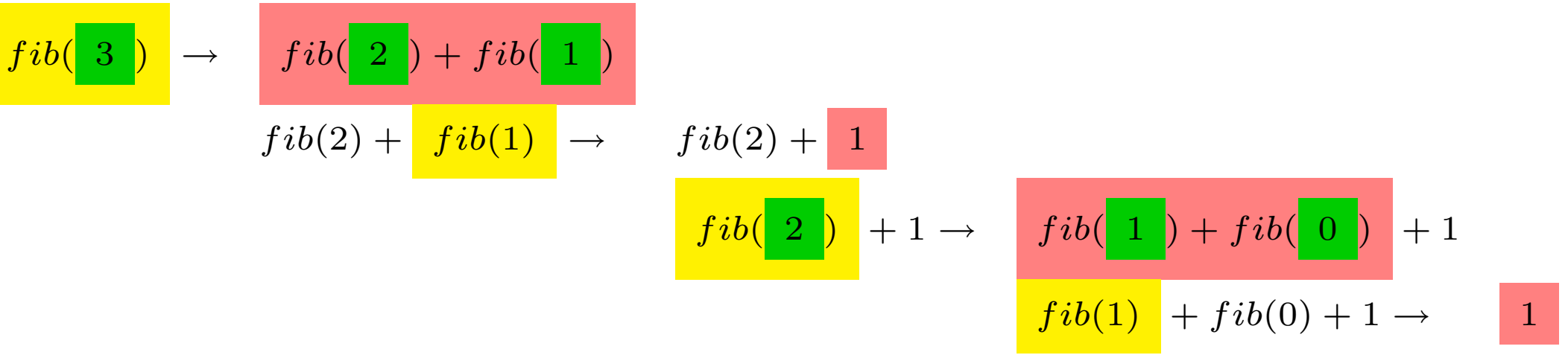
$$fib(3) \rightarrow fib(2) + fib(1)$$

$$fib(2) + fib(1) \rightarrow fib(2) + 1$$

$$fib(2) + 1 \rightarrow fib(1) + fib(0) + 1$$

fibonacci

$$
\begin{array}{lll}
[\alpha] \quad fib(0) & \rightarrow & 1 \\
[\beta] \quad fib(1) & \rightarrow & 1 \\
[\gamma] \quad fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$



fibonacci

$$
\begin{array}{lll}
[\alpha] & fib(0) & \rightarrow \quad 1 \\
[\beta] & fib(1) & \rightarrow \quad 1 \\
[\gamma] & fib(n) & \rightarrow \quad fib(n-1) + fib(n-2)
\end{array}
$$

$$fib(3) \rightarrow fib(2) + fib(1)$$

$$fib(2) + fib(1) \rightarrow fib(2) + 1$$

$$fib(2) + 1 \rightarrow fib(1) + fib(0) + 1$$

$$fib(1) + fib(0) + 1 \rightarrow 1 + 1$$

$$1 + 1 \rightarrow \ldots$$

Finally $fib(3) = 3$, $fib(4) = 5$, ...

Graphical Rewriting

$$F \longrightarrow F + F - F - FF + F + F - F$$

Graphical Rewriting

$$F \longrightarrow F + F - F - FF + F + F - F$$

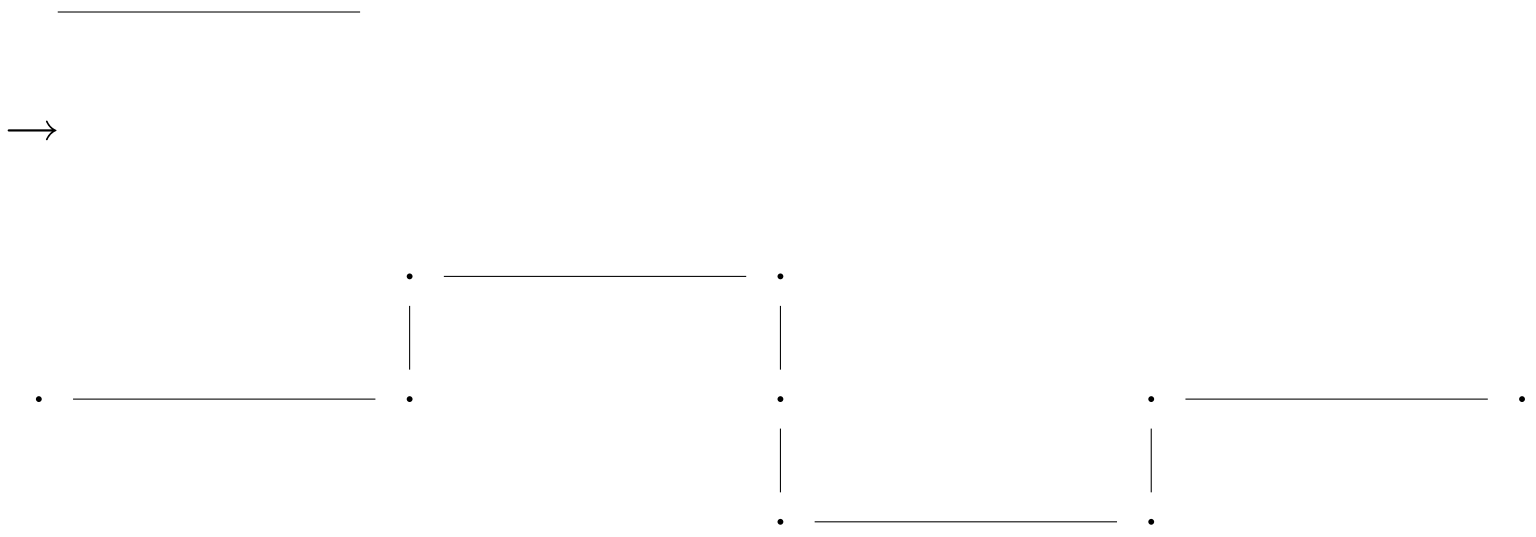
Graphical Rewriting

$$F \longrightarrow F + F - F - FF + F + F - F$$

$\longrightarrow$

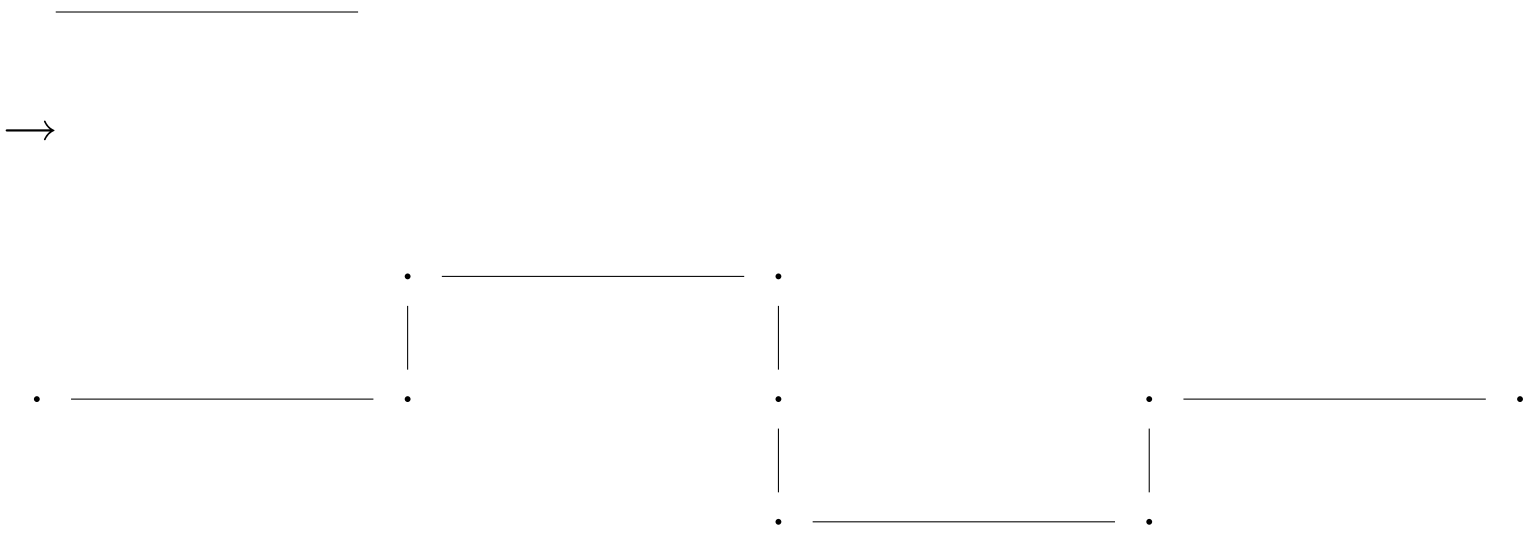
Graphical Rewriting

$$F \;\longrightarrow\; F + F - F - FF + F + F - F$$



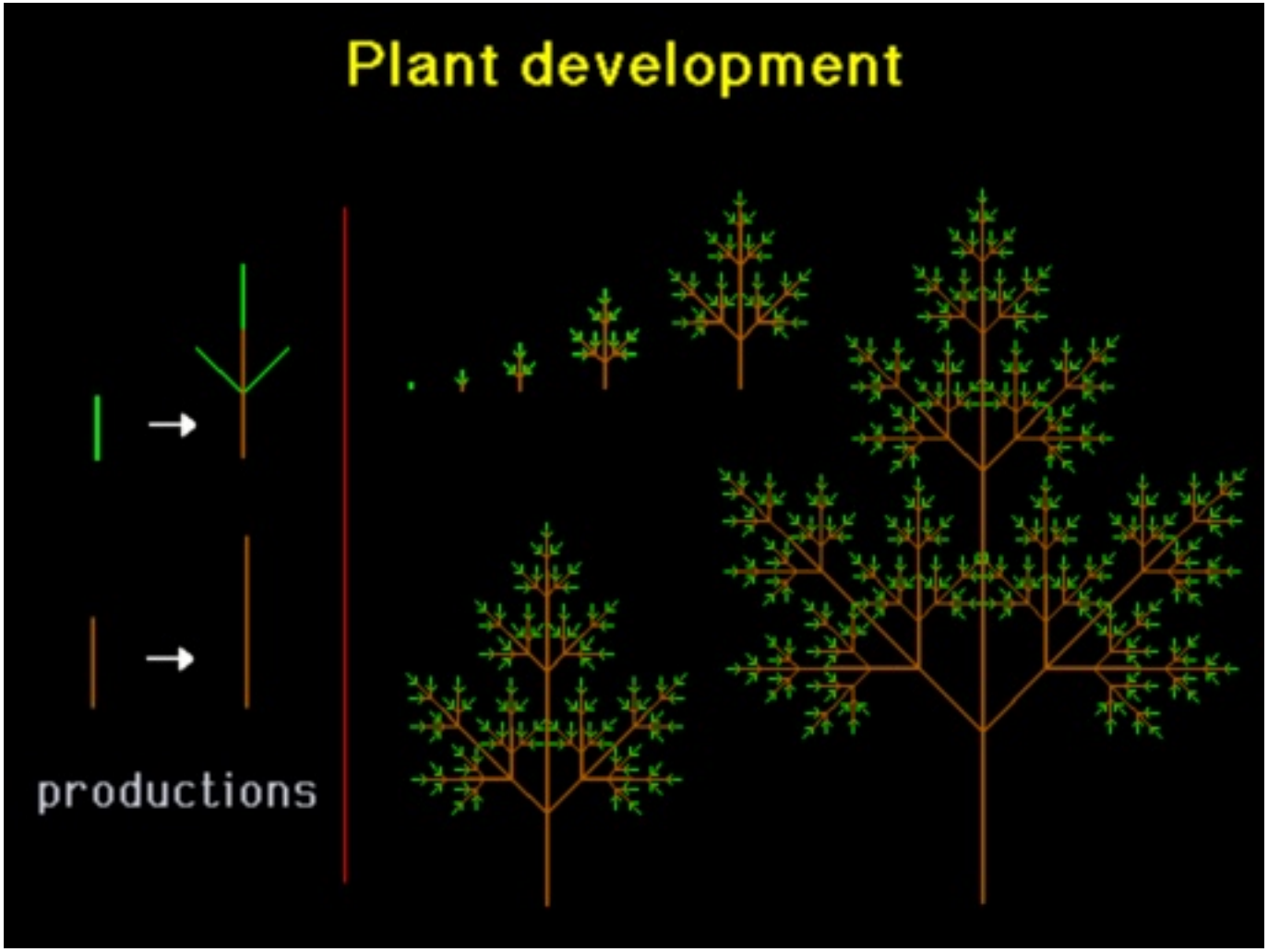
Graphical Rewriting

$$F \rightarrow F + F - F - FF + F + F - F$$



L-systems (Lindenmeier)

Ecological Rewriting



http://www.cpsc.ucalgary.ca/Redirect/bmv/vmm-deluxe/Section-08.html

/etc/sendmail.cf

```
#####  @(#)nullrelay.m4 8.19 (Berkeley) 5/19/1998  #####

#
#  This configuration applies only to relay-only hosts.  They send
#  all mail to a hub without consideration of the address syntax
#  or semantics, except for adding the hub qualification to the
#  addresses.
#
#       This is based on a prototype done by Bryan Costales of ICSI.
#

##########################################################################
##########################################################################
#####
#####                     REWRITING RULES
#####
##########################################################################
##########################################################################

########################################
###  Ruleset 3 -- Name Canonicalization  ###
```

```
##############################################
S3

# handle null input
R$@                     $@ <@>

# strip group: syntax (not inside angle brackets!) and trailing semicolon
R$*                     $: $1 <@>                       mark addresses
R$* < $* > $* <@>       $: $1 < $2 > $3                 unmark <addr>
R$* :: $* <@>           $: $1 :: $2                     unmark node::addr
R:include: $* <@>       $: :include: $1                 unmark :include:...
R$* : $* <@>            $: $2                           strip colon if marked
.
.
.
.
.
.
.
.

##################################################
###  Ruleset 4 -- Final Output Post-rewriting  ###
```

```
##########################################################
S4

R$* <@>                 $@                                  handle <> and list:;

# strip trailing dot off before passing to nullclient relay
R$* @ $+ .              $1 @ $2
```

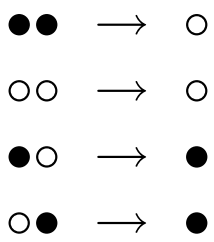
Equational description of a sorting algorithm

```
  sorts NeList List ;   subsorts Nat < NeList < List ;
operators
  nil : List ;
  @ @ : (List List) List       [associative id: nil] ;
  @ @ : (NeList List) NeList   [associative] ;
  hd @ : (NeList) Nat ;
  tl @ : (NeList) List ;
  sort @ : (List) List ;
end
rules for List
  X, Y : Nat ; L L' L'' : List;
   hd (X L) => X ;                      tl (X L) => L ;
   sort nil => nil .
   sort (L X L' Y L'') =>  sort (L Y L' X L'')  if Y < X .
end

sort (6 5 4 3 2 1) => ...
```

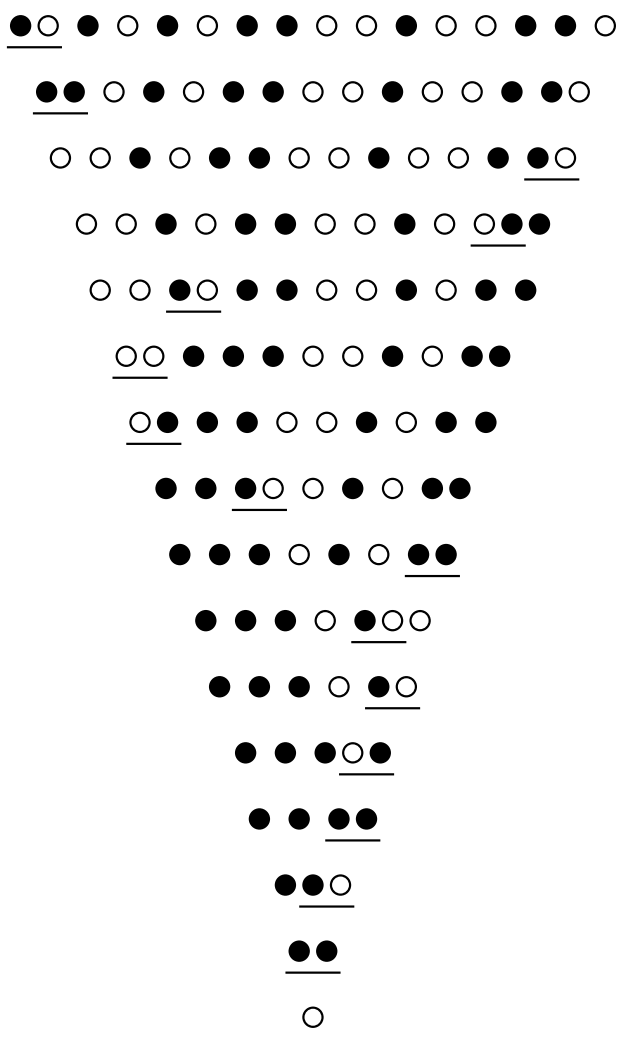
Back with the simple game

The rules of the game:



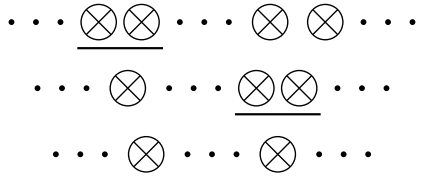
A starting point:

From a given start, is the result determinist?

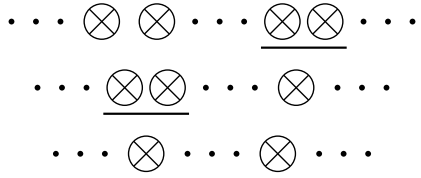


Analysing the different cases

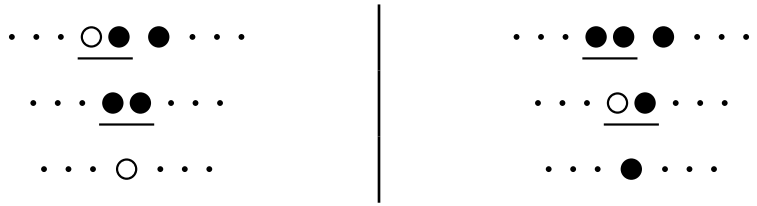
Disjoint redexes:



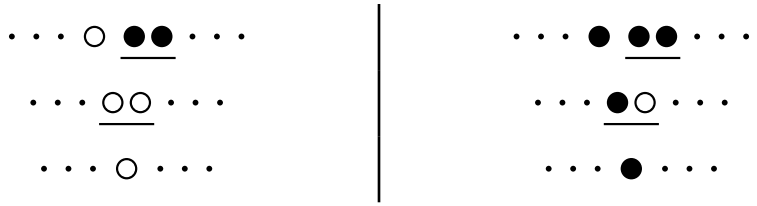
is the same as:



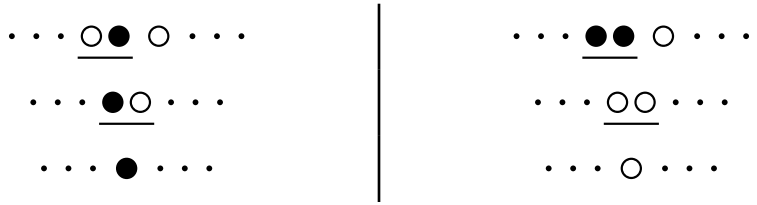
No disjoint redexes (central black):



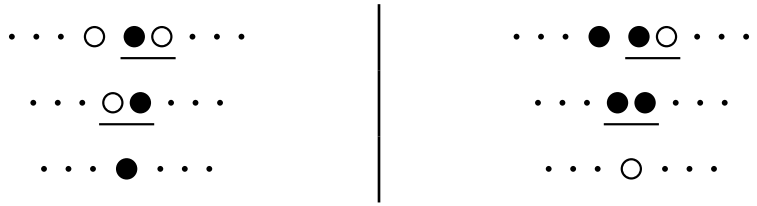
but



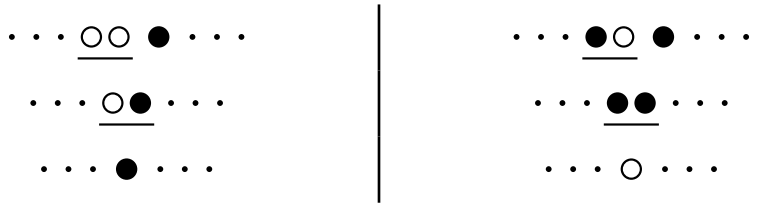
or



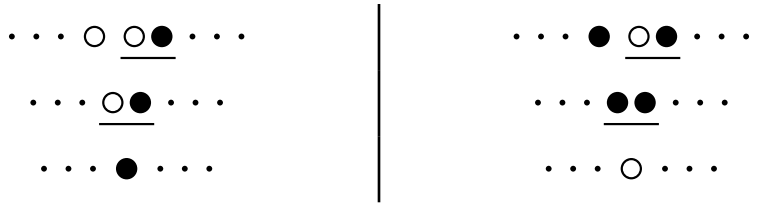
but



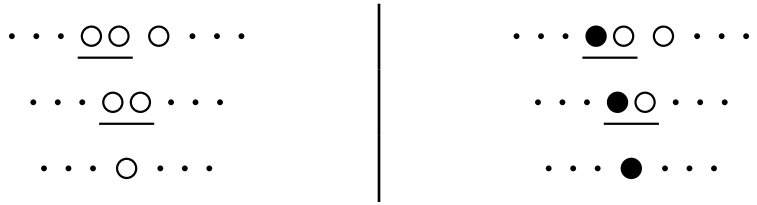
No disjoint redexes (central white):



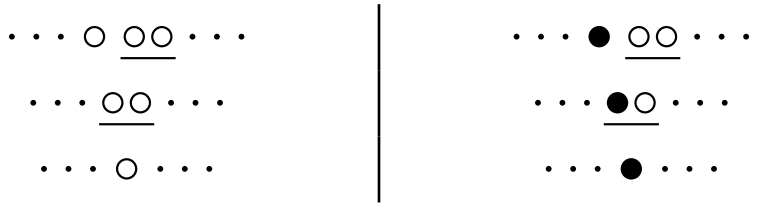
but



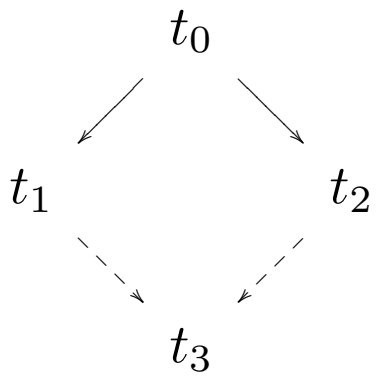
or



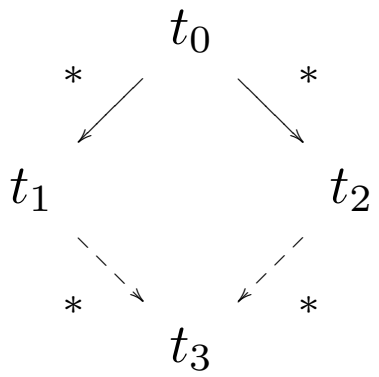
but



Thus in all the cases:



but what about:



Alternative Rings

$$
\begin{aligned}
0 + x &= x \\
x * 0 &= 0 \\
i(x + y) &= i(x) + i(y) \\
x * (y + z) &= (x * y) + (x * z) \\
(x * y) * y &= x * (y * y) \\
i(x) * y &= i(x * y) \\
i(0) &= 0 \\
x + y &= y + x
\end{aligned}
\qquad
\begin{aligned}
0 * x &= 0 \\
i(x) + x &= 0 \\
i(i(x)) &= x \\
(x + y) * z &= (x * z) + (y * z) \\
(x * x) * y &= x * (x * y) \\
x * i(y) &= i(x * y) \\
(x + y) + z &= x + (y + z)
\end{aligned}
$$

Can we prove by rewriting the Moufang Identities?

$$
\begin{aligned}
(x * y) * x &= x * (y * x) \\
x * ((y * z) * x) &= (x * (y * z)) * x \\
x * (y * (x * z)) &= ((x * y) * x)) * z \\
((z * x) * y) * x) &= z * (x * (y * x)) \\
(x * y) * (z * x) &= (x * (y * z)) * x
\end{aligned}
$$

(R.Moufang, 1933)
(S.Anantharaman and J.Hsiang, JAR 6, 1990)

An additive group G is defined by the set of equalities

$$\begin{aligned} x + e &= x \\ x + (y + z) &= (x + y) + z \\ x + i(x) &= e \end{aligned}$$

How to check that two elements of the group are the same?

$$i(x + y) =? = i(y) + i(x)$$

An equivalent deterministic term rewrite system

$$
\begin{array}{rcl}
x + e & \rightarrow & x \\
e + x & \rightarrow & x \\
x + (y + z) & \rightarrow & (x + y) + z \\
x + i(x) & \rightarrow & e \\
i(x) + x & \rightarrow & e \\
i(e) & \rightarrow & e \\
(y + i(x)) + x & \rightarrow & y \\
(y + x) + i(x) & \rightarrow & y \\
i(i(x)) & \rightarrow & x \\
i(x + y) & \rightarrow & i(y) + i(x)
\end{array}
$$

Then

$$i(x + y) =? = i(y) + i(x) \quad \Leftrightarrow \quad i(x + y) \rightarrow \ldots \rightarrow \bullet \leftarrow \ldots \leftarrow i(y) + i(x)$$

XSLT

Note that XSLT is just a (special) kind of rewriting language,
acting on XML documents

XSLT: example 1/3

Document Example D.1 in the Appendix of XSLT specification

```
<xsl:stylesheet
     version="1.0"   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns="http://www.w3.org/TR/xhtml1/strict">

<xsl:template match=doc>
   <html>
     <head>
        <title> <xsl:value-of select=title /> </title>
     </head>
     <body> <xsl:apply-templates/> </body>
   </html>
</xsl:template>

...

<xsl:template match=chapter/title>
   <h2> <xsl:apply-templates/> </h2>
</xsl:template>
```

XSLT: example 2/3

When applied to

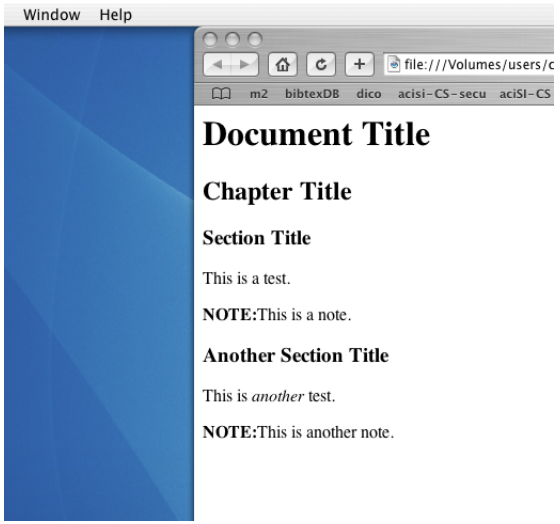
```
<doc>
    <title>Document Title</title>
    <chapter>
        <title>Chapter Title</title>
        <section>
            <title>Section Title</title>
            <para>This is a test.</para>
            <note>This is a note.</note>
        </section>
        <section>
            <title>Another Section Title</title>
            <para>
                This is
                <emph> another </emph>
                 test.
            </para>
            <note> This is another note. </note>
        </section>
    </chapter> </doc>
```

XSLT: example 3/3

we get as result the HTML document

```
<html><head><title>Document Title</title></head><body>
<h1>Document Title</h1><h2>Chapter Title</h2><h3>Section Title</h3>
<p>This is a test.</p><p class=note>
<b>NOTE:</b>This is a note.</p><h3>Another Section
 Title</h3><p>This is <em>another</em> test.</p><p class=note>
<b>NOTE:</b>This is another note.</p>
</body></html>
```

Which we usualy prefer to see as



The 4 basic ingredients of a rewriting step: 1- The rewritten objects

They can be:

- terms like $2 + i(3)$ or XML documents
- strings like "What is rewriting?" `sed` performs string rewriting
- graphs
- sets
- multisets
- higher-order terms
- . . .

The 4 basic ingredients of a rewriting step: 2- Substitution

Graphting = First-order substitution
Replace all the instanciated variables by their values

Example: Apply $\sigma = \{x \mapsto b, y \mapsto a\}$ on $f(x, g(x, y))$

Denoted either $\sigma(g(x, g(x, y)))$ or $(g(x, g(x, y)))\sigma$

The 4 basic ingredients of a rewriting step: 3- Matching

Finding a substitution σ such that

$$l\sigma \overset{\mathbb{T}}{=} t$$

is called the matching problem from l to t, modulo the theory $\mathbb{T}$
When $\mathbb{T} = \emptyset$, it is decidable in linear time in the size of t
It induces a relation on terms called subsumption

Example: $f(x, g(x, y)) \ll_{\emptyset} f(b, g(b, a))$

The 4 basic ingredients of a rewriting step: 4- Replacement

At a given occurrence of a term, replace the existing subterm by another one

Example:
Put $f(x, a)$ at occurence 1.1 in $g(g(a, x), f(b, b))$

How does (first-order) rewriting work?

It relies on the 4 notions:
(first-order) terms, substitution, matching, replacement
and, given a set of rules $\mathcal{R}$ and a term t to be rewritten, it consists to:

How does (first-order) rewriting work?

It relies on the 4 notions:
(first-order) terms, substitution, matching, replacement
and, given a set of rules $\mathcal{R}$ and a term t to be rewritten, it consists to:

– find a rule in $\mathcal{R}$

How does (first-order) rewriting work?

It relies on the 4 notions:
(first-order) terms, substitution, matching, replacement
and, given a set of rules $\mathcal{R}$ and a term t to be rewritten, it consists to:

– find a rule in $\mathcal{R}$

– find a subterm of t

How does (first-order) rewriting work?

It relies on the 4 notions:
(first-order) terms, substitution, matching, replacement
and, given a set of rules $\mathcal{R}$ and a term t to be rewritten, it consists to:

– find a rule in $\mathcal{R}$

– find a subterm of t

– that matches the left hand side of the rule

How does (first-order) rewriting work?

It relies on the 4 notions:
(first-order) terms, substitution, matching, replacement
and, given a set of rules $\mathcal{R}$ and a term t to be rewritten, it consists to:

– find a rule in $\mathcal{R}$

– find a subterm of t

– that matches the left hand side of the rule

– and replace that subterm by the right hand side of the rule instanciated by the match

Denoted $t \Longrightarrow_{\mathcal{R}} t'$

How does (first-order) rewriting work?

It relies on the 4 notions:
(first-order) terms, substitution, matching, replacement
and, given a set of rules $\mathcal{R}$ and a term t to be rewritten, it consists to:

– find a rule in $\mathcal{R}$

– find a subterm of t

– that matches the left hand side of the rule

– and replace that subterm by the right hand side of the rule instanciated by the match

Denoted $t \Longrightarrow_{\mathcal{R}} t'$

Simple?. . .

How does (first-order) rewriting work?

It relies on the 4 notions:
(first-order) terms, substitution, matching, replacement
and, given a set of rules $\mathcal{R}$ and a term t to be rewritten, it consists to:

– find a rule in $\mathcal{R}$

– find a subterm of t

– that matches the left hand side of the rule

– and replace that subterm by the right hand side of the rule instanciated by the match

Denoted $t \Longrightarrow_{\mathcal{R}} t'$

Simple?. . . let's sum-up

What is rewriting? (1/2)

1- Discriminate

to give the possibility to discriminate directly

"one image is better than ten explanations"

What is rewriting? (1/2)

1- Discriminate

to give the possibility to discriminate directly
"one image is better than ten explanations"

What is rewriting? (1/2)

1- Discriminate

to give the possibility to discriminate directly
"one image is better than ten explanations"

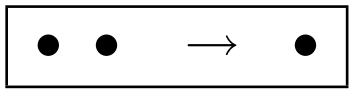
lambda-calculus is not discriminating: one needs to encode matching

What is rewriting? (2/2)

1- Discriminate
2- Transform what has been discriminated

What is rewriting? (2/2)

1- Discriminate
2- Transform what has been discriminated

-  discriminates the repetition of two black bullets

What is rewriting? (2/2)

1- Discriminate
2- Transform what has been discriminated

- $\boxed{\bullet\ \bullet \quad \longrightarrow \quad \bullet}$ discriminates the repetition of two black bullets

  $\bullet\ \circ\ \bullet\ \bullet\ \circ\ \circ\ \bullet\ \bullet$ is rewritten, for example, into $\bullet\ \circ\ \bullet\ \circ\ \circ\ \bullet\ \bullet$

- $\boxed{x \times 0 \quad \longrightarrow \quad 0}$ discriminates objects where x is arbitrary

  3×0 is rewritten into 0

What is rewriting? (2/2)

1- Discriminate
2- Transform what has been discriminated

- $\boxed{\bullet\ \bullet \quad \longrightarrow \quad \bullet}$ discriminates the repetition of two black bullets

  $\bullet\ \circ\ \bullet\ \bullet\ \circ\ \circ\ \bullet\ \bullet$ is rewritten, for example, into $\bullet\ \circ\ \bullet\ \circ\ \circ\ \bullet\ \bullet$

- $\boxed{x \times 0 \quad \longrightarrow \quad 0}$ discriminates objects where x is arbitrary

  3×0 is rewritten into 0

- $\boxed{x \text{ et } x \quad \longrightarrow \quad x}$ discriminates objects where x is repeated

  $x = 3$ et $x = 3$ is rewritten into en $x = 3$

Important: Rewriting is always in need of a strategy

- to chose the appropriate rule
- to find an appropriate occurrence
- to chose the appropriate result

Examples using ELAN

ELAN, a system for clever rewriting: *deduction modulo* at work

$$\text{ELAN} = \text{computation rules} + (\text{deduction rules} + \text{strategies})$$

Implements rewriting for

- computation — Fibonacci
- proving — Prop. Seq. Calculus
- solving — 8 queens

Example 1: Very simple ...

```
module fib_builtin
import global builtinInt;
  end

operators global
        fib(@)  : (builtinInt) builtinInt ;
  end

rules for builtinInt
        n : builtinInt ;
global
    [] fib(0) => 1 end
    [] fib(1) => 1 end
    [] fib(n) => fib(n - 1) + fib(n - 2) if greater_builtinInt(n,1) end
  end
end
```

fib(33) = 5702887 11405773 rewrite steps in 0.695 s 16.411.184 rewrite/s

Digital 500/500, 128Mo

Example 2: propositional sequent calculus

$$\frac{H, P \vdash Q}{H \vdash \neg P, Q}\ \text{neg} - \text{r}$$

```
rules for Seq
        P, Q, R : Pred;   H : Pred;   S1, S2 : Seq;
global
  [negd]   H |- ^P : Q   => S1
                                  where S1 := (dedstrat) H : P |- Q
  end

...
```

The true code

Built (for later use) the proof term:

```
[negr]  H |- ^P : Q   =>  [#negd,H |- ^P : Q] <> S1
                                  where S1 := (dedstrat) H : P |- Q
end
```

Strategies

```
strategies  for Seq
implicit

        [] SetRules  =>  first one(
                                 axio
                                 ,negd ,disjd
                                 ,impd ,negg ,conjg
                                 ,disjg ,conjd ,impg)
        end
end

strategies  for Seq
implicit
        [] dedstrat  =>  first one( Start );
                         repeat*( SetRules )
        end
end
```

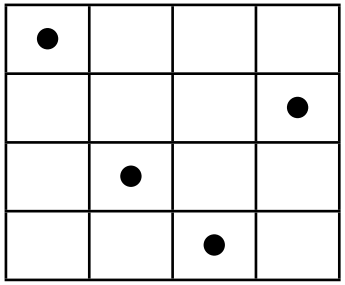
The resulting proof term

```
[dedstrat](A |=> B |- ^(B) |=> ^(A))
```

evaluates to:

```
#infer[#impd]<(A#to B)#vdash(#neg(B)#to#neg(A))>
<#infer[#negd]<(A#to B),#neg(B)#vdash#neg(A)>
<#infer[#negg]<A,(A#to B),#neg(B)#vdash EmptyP>
<#infer[#impg]<A,(A#to B)#vdash B>
<#infer[#axiom]<A,B#vdash B><#mbox<>>&
#infer[#axiom]<A#vdash A,B><#mbox<>>>>>>
end
```

Example 3: 8 queens



is represented by the list (3,1,0,2)

Module nqueensAC

```
operators
 global
    queensAC(@) : (int) list[int] ;
 local
    queens(@,@) : (set list[int]) list[int] ;
    ok(@,@,@)   : ( int int list[int] ) bool;
    @ U @       : (set set) set (AC);
    (@)         : (int) set ;
    [@ U @]     : (int set) set ;
    Set(@)      : (int) set ;
    Empty       : set ;
end
```

```
rules for list[int]
  n:int;
global
  [] queensAC(n) => queens(Set(n-1),nil)

rules for list[int]
  p1: int;  s,s1:set ;  l,l1: list[int];
local
  [final] queens(Empty,l) => l     end

  [queensrule]  queens(s,l)     => queens(s1,p1.l)
                  where (set) [p1 U s1] :=(extractPos) s
                  if ok(1,p1,l)

strategies for list[int]
implicit
  [] queens => repeat*(dk(queensrule)); first(final)
```

Related systems

- TOM (`tom.loria.fr`)
- ASF+SDF
- OBJ, MAUDE
- STRATEGO
- LPG
- ...

Some applications

- XML and XSLT
- Program transformation (e.g. compilation)
- Simplification (e.g. computer algebra)
- Computation
- Production rules
- Model checking
- Proof search —Cariboo—
- Chemistry

. . .

Syntactic Matching: A rule based description

Delete

$$t \lessdot t \wedge P \rightarrow P$$

Decomposition

$$f(t_1, \ldots, t_n) \lessdot f(t'_1, \ldots, t'_n) \wedge P \rightarrow \bigwedge_{i=1,\ldots,n} t_i \lessdot t'_i \wedge P$$

SymbolClash

$$f(t_1, \ldots, t_n) \lessdot g(t'_1, \ldots, t'_m) \wedge P \rightarrow \texttt{fail} \quad \text{if } f \neq g$$

MergingClash

$$x \lessdot t \wedge x \lessdot t' \wedge P \rightarrow \texttt{fail} \quad \text{if } t \neq t'$$

SymbolVariableClash

$$f(t_1, \ldots, t_n) \lessdot x \wedge P \rightarrow \texttt{fail} \quad \text{if } x \in \mathcal{X}$$

Find a match

$$x+(y*y) \ll 1+(4*3)$$

$$\Rightarrow_{\text{Decomposition}} x \ll 1 \ \wedge \ y*y = 4*3$$

$$\Rightarrow_{\text{Decomposition}} x \ll 1 \ \wedge \ y \ll 4 \ \wedge \ y \ll 3$$

$$\Rightarrow_{\text{MergingClash}} \texttt{fail}$$

Find a match

$$x+(y*y) \lessdot 1+(4*3)$$
$$\Rightarrow_{\text{Decomposition}} x \lessdot 1 \ \wedge \ y*y = 4*3$$
$$\Rightarrow_{\text{Decomposition}} x \lessdot 1 \ \wedge \ y \lessdot 4 \ \wedge \ y \lessdot 3$$
$$\Rightarrow_{\text{MergingClash}} \texttt{fail}$$

$$x+(y*3) \lessdot 1+(4*3)$$
$$\Rightarrow_{\text{Decomposition}} x \lessdot 1 \ \wedge \ y*3 = 4*3$$
$$\Rightarrow_{\text{Decomposition}} x \lessdot 1 \ \wedge \ y \lessdot 4 \ \wedge \ 3 \lessdot 3$$
$$\Rightarrow_{\text{Delete}} x \lessdot 1 \ \wedge \ y \lessdot 4$$

Theorem

The normal form by the rules in **Match**, of any matching problem $t \ll t'$ such that $\mathcal{V}ar(t) \cap \mathcal{V}ar(t') = \emptyset$, exists and is unique.

1. If it is `fail`, then there is no match from t to t'.

2. If it is of the form $\bigwedge_{i \in I} x_i \ll t_i$ with $I \neq \emptyset$, the substitution $\sigma = \{x_i \mapsto t_i\}_{i \in I}$ is the unique match from t to t'.

3. If it is empty then t and t' are identical: $t = t'$.

Aims of the ρ-calculus

- To define at the same level
  - rewrite rules
  - rewriting strategies
  - applications of rules and strategies
  - results

Aims of the ρ-calculus

- To define at the same level
  - rewrite rules
  - rewriting strategies
  - applications of rules and strategies
  - results
- To unify:
  - first-order rewriting (ELAN, Maude . . .)
  - λ-calculus

For the rewriting RELATION

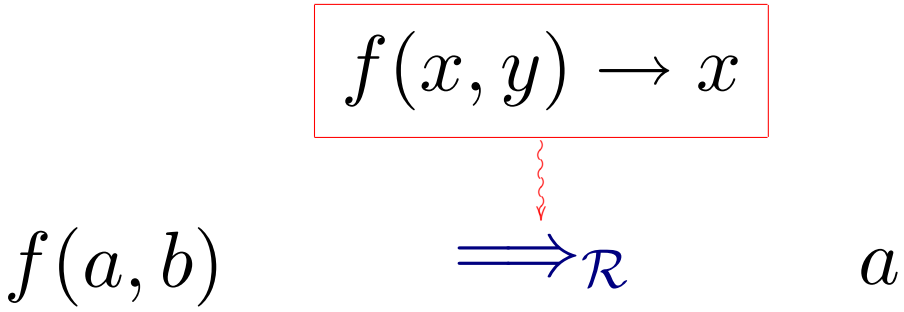
$$f(x, y) \longrightarrow x$$

For the rewriting RELATION

$$f(x, y) \longrightarrow x$$

$$f(a, b)$$

For the rewriting RELATION



For the rewriting CALCULUS

$$f(X, Y) \quad \rightarrow \quad X$$

➥ Abstraction Operator

For the rewriting calculus

$$f(X,Y) \quad \twoheadrightarrow \quad X \qquad f(a,b)$$

For the rewriting calculus

$$\left(f(X,Y) \quad \twoheadrightarrow \quad X \right) \bullet f(a,b)$$

↪ Application Operator

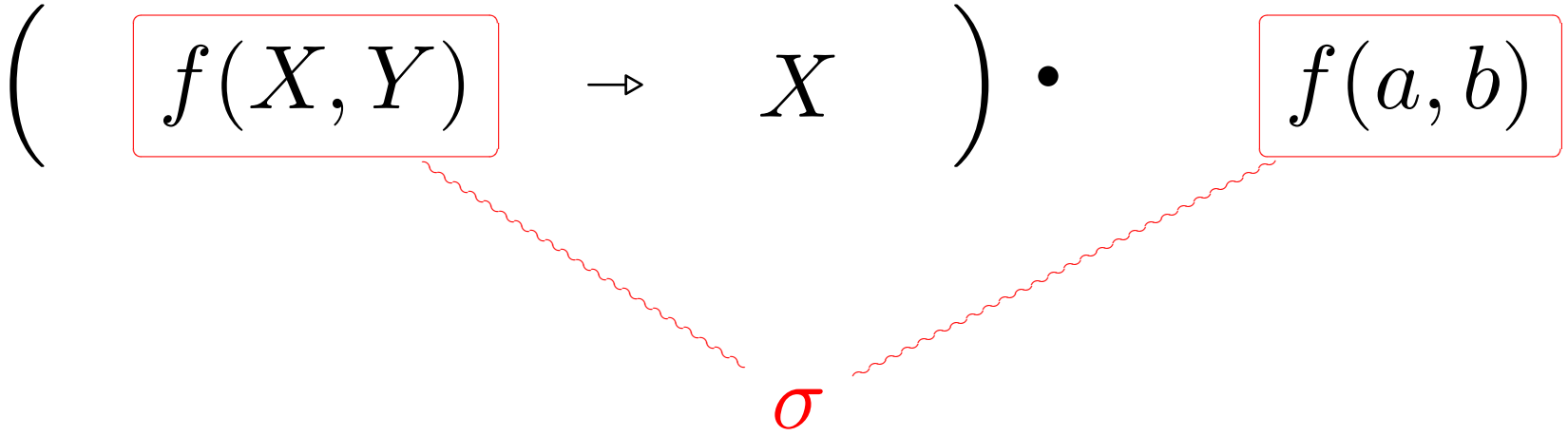
For the rewriting calculus

$$\left(f(X,Y) \quad \twoheadrightarrow \quad X \right) \bullet f(a,b)$$

↪ Application Operator

Rule to evaluate such a term: (ρ)

For the rewriting calculus



For the rewriting calculus

$$\left(\quad f(X,Y) \quad \twoheadrightarrow \quad X \quad \right) \bullet \quad f(a,b)$$

$$\sigma = \{X \mapsto a, Y \mapsto b\}$$

For the rewriting calculus

$$\left(\quad f(X,Y) \quad \twoheadrightarrow \quad X \quad \right) \bullet \quad f(a,b)$$

σ

$$= \{X \mapsto a, Y \mapsto b\}$$

For the rewriting calculus

$$\left(\quad f(X,Y) \quad \twoheadrightarrow \quad X \quad \right) \bullet \quad f(a,b)$$

$$\longmapsto_{\rho} \quad \sigma(X)$$

For the rewriting calculus

$$\left(\quad f(X,Y) \quad \twoheadrightarrow \quad X \quad \right) \bullet \quad f(a,b)$$

$\longmapsto_\rho \ \sigma(X)$ *i.e.* a

For the rewriting relation

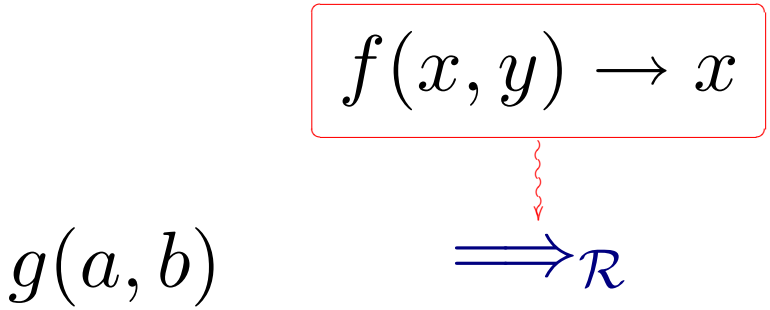
$$f(x, y) \longrightarrow x$$

For the rewriting relation

$$f(x, y) \longrightarrow x$$

$$g(a, b)$$

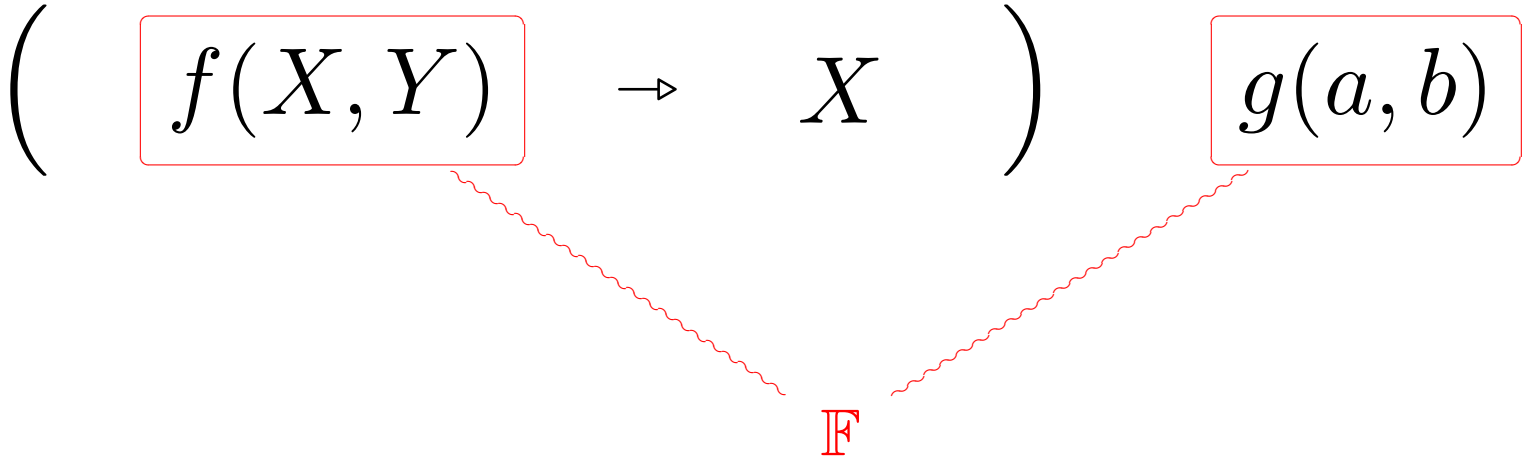
For the rewriting relation



For the rewriting calculus

$$\Big(\ f(X,Y) \quad \twoheadrightarrow \quad X\Big) \quad g(a,b)$$

For the rewriting calculus



Related topics and works

- pattern lambda calculus (Peyton-Jones, Van Oostrom, Colson)
- pattern calculus (Jay)
- combination of rewriting and HO features
  - ⋆ CRS (Klop)
  - ⋆ lambda calculus and rewriting (Breazu Tannen & Gallier, Okada)
  - ⋆ CC and rewriting (Blanqui, Jouannaud, Okada)
  - ⋆ HO rewriting (Nipkow)
- ML, Haskel, Rogue

(Some) Recommended Readings

- The Rewriting Calculus Home page
  `http://www.loria.fr/~faure/TheRhoCalculusHomePage/`

- Repository of Lectures on Rewriting and Related Topics
  `www.loria.fr/~ckirchne/`

- Online book on rewriting `www.loria.fr/~ckirchne/rsp.ps.gz`

- L'intelligence et le calcul (may be translated to English?)
  Jean-Paul Delahaye
  Look also at his web page

- Term Rewriting Systems
  Terese (M. Bezem, J. W. Klop and R. de Vrijer, eds.)
  Cambridge Univerty press, 2002

- Term *Re*writing and *all That*
  Franz Baader and Tobias Nipkow
  Cambridge Univerty press, 1998

The syntax and semantics of the untyped rewriting calculus

The Untyped Syntax

$$\mathcal{P} ::= \mathcal{T} \qquad \text{Patterns}$$

$$\mathcal{T} ::= \mathcal{X} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid [\mathcal{P} \ll \mathcal{T}]\mathcal{T} \mid \mathcal{T}, \mathcal{T} \qquad \text{Terms}$$

1. $T_1 \twoheadrightarrow T_2$ denotes a *rule abstraction* with pattern T_1 and body T_2
   ... the free variables of T_1 are bound in T_2

2. $[T_1 \ll T_2]T_3$ denotes a *delayed matching constraint*
   ... the free variables of T_1 are bound in T_3 but not in T_2

3. The terms can be also *structures* built using the symbol ","

4. We work modulo the *α-convention* and the *hygiene-convention*

Some ρ-terms

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a$ similar to the λ-term $(\lambda x.x)\ a$

$(\mathcal{X} \twoheadrightarrow \mathcal{X}\ \mathcal{X})\ (\mathcal{X} \twoheadrightarrow \mathcal{X}\ \mathcal{X})$ the well-known λ-term $(\omega\omega)$

Some ρ-terms

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a$ similar to the λ-term $(\lambda x.x)\ a$

$(\mathcal{X} \twoheadrightarrow \mathcal{X}\ \mathcal{X})\ (\mathcal{X} \twoheadrightarrow \mathcal{X}\ \mathcal{X})$ the well-known λ-term $(\omega\omega)$

$(a \twoheadrightarrow b)\ a$ the application of the rule $a \twoheadrightarrow b$ to the term a

$(f(\mathcal{X}, \mathcal{Y}) \twoheadrightarrow g(\mathcal{X}, \mathcal{Y}))\ f(a, b)$ a classical rewrite rule application

Some ρ-terms

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a$ similar to the λ-term $(\lambda x.x)\ a$

$(\mathcal{X} \twoheadrightarrow \mathcal{X}\ \mathcal{X})\ (\mathcal{X} \twoheadrightarrow \mathcal{X}\ \mathcal{X})$ the well-known λ-term $(\omega\omega)$

$(a \twoheadrightarrow b)\ a$ the application of the rule $a \twoheadrightarrow b$ to the term a

$(f(\mathcal{X}, \mathcal{Y}) \twoheadrightarrow g(\mathcal{X}, \mathcal{Y}))\ f(a, b)$ a classical rewrite rule application

$(a \twoheadrightarrow b, a \twoheadrightarrow c)\ a$ "non-deterministic" application

Some abbreviations

$(T_i)^{i=1\ldots n} \triangleq T_1, \ldots, T_n$ structure/object ($n \in$ Nat)

$T_1.T_2 \triangleq T_1\ T_2\ T_1$ Kamin's self-application

Operator	Associate	Priority
$\_\ \_$	Left	$>$
$[\_ \ll \_]\_$	Right	$>$
$\_ \rightarrowtriangle \_$	Right	$>$
$\_, \_$	Right	$>$

Matching Equations and Solutions

1. A *match equation* is a formula of the form $T_1 \lessdot\!\ll T_2$.

2. A *matching system* $\mathsf{T} \stackrel{\triangle}{=} \bigwedge_{i=0\ldots n} T_i \lessdot\!\ll T_i'$ is a conjunction of match equations, where $\wedge$ is associative, commutative, and idempotent.

3. A matching system T is *successful* if it is empty or:
   (a) has the shape $\bigwedge_{i=0\ldots n} \mathcal{X}_i \lessdot\!\ll T_i \bigwedge_{j=0\ldots m} \mathcal{K}_j \lessdot\!\ll \mathcal{K}_j$;
   (b) for all $h, k = 0 \ldots n$, such that $h \neq k$, we have $\mathcal{X}_h \neq \mathcal{X}_k$;

4. A substitution $\sigma_{\mathsf{T}} = \{T_1/\mathcal{X}_1 \cdots T_n/\mathcal{X}_n\}$ is the solution of a successful matching system T. The set of solutions of T is denoted by $\mathcal{S}ol(\mathsf{T})$.

Reduction relies on matching power

Matching is parametrized over a theory $\mathbb{T}$ and an order $\prec$ on substitutions

$$\mathcal{S}ol(T_1 \prec\!\!\prec_{\mathbb{T}} T_2) = \sigma_1, \ldots, \sigma_n, \ldots$$

$$\sigma \in \mathcal{S}ol(T_1 \prec\!\!\prec_{\mathbb{T}} T_2) \;\Leftrightarrow\; \models_{\mathbb{T}} \sigma(T_1) = T_2$$

$$\sigma_1 \prec \ldots \prec \sigma_n \qquad (n \leq \infty)$$

THEORIES ALGORITHM

The Small-step Reduction Semantics

$$(P \twoheadrightarrow A)\ B \ \rightarrow_{\rho}\ [P \ll B]A$$

$$[P \ll B]A \ \rightarrow_{\sigma}\ A\theta_{(P \lessdot B)} \qquad \text{if } \exists\theta.\ P\theta =_{\mathbb{T}} B$$

$$(A, B)\ C \ \rightarrow_{\delta}\ A\ C, B\ C$$

The Small-step Reduction Semantics

$$(P \twoheadrightarrow A)\ B \ \rightarrow_{\rho}\ [P \ll B]A$$

$$[P \ll B]A \ \rightarrow_{\sigma}\ A\theta_1, \ldots, A\theta_n, \ldots$$

$$\text{with } \{\theta_1, \ldots, \theta_n, \ldots\} = \mathcal{S}ol(P \prec\!\!\ll_{\mathbb{T}} B)$$

$$(A, B)\ C \ \rightarrow_{\delta}\ A\ C, B\ C$$

Intuition on the small-step Semantics

$$(P \twoheadrightarrow A)\ B \ \rightarrow_\rho \ [P \ll B]A$$
$$\rightarrow_\sigma \ A\theta$$

if $\exists\theta.\ P\theta =_{\mathbb{T}} B$

$$(P \twoheadrightarrow A)\ B \ \rightarrow_\rho \ [P \ll B]A$$

STOP!

if $\nexists\theta.\ P\theta =_{\mathbb{T}} B$

Some ρ-reductions

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a$

$(\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X}))\ (\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X}))$

$(a \twoheadrightarrow b)\ a$

$(f(\mathcal{X}, \mathcal{Y}) \twoheadrightarrow g(\mathcal{X}, \mathcal{Y}))\ (f(a, b))$

$(f(\mathcal{X}, \mathcal{Y}) \twoheadrightarrow g(\mathcal{X}, \mathcal{Y}))\ (g(a, b))$

$(a \twoheadrightarrow b, a \twoheadrightarrow c)\ a$

Some ρ-reductions

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a \qquad \longmapsto\!\!\!\!\!\rightarrow_{\rho\infty\delta}\ a$

$(\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X}))\ (\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X}))$

$(a \twoheadrightarrow b)\ a$

$(f(\mathcal{X}, \mathcal{Y}) \twoheadrightarrow g(\mathcal{X}, \mathcal{Y}))\ (f(a, b))$

$(f(\mathcal{X}, \mathcal{Y}) \twoheadrightarrow g(\mathcal{X}, \mathcal{Y}))\ (g(a, b))$

$(a \twoheadrightarrow b, a \twoheadrightarrow c)\ a$

Some ρ-reductions

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a \qquad \longmapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ a$

$(\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X}))\ (\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X})) \qquad \longmapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ \omega\ \omega \longmapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ \ldots$

$(a \twoheadrightarrow b)\ a$

$(f(\mathcal{X}, \mathcal{Y}) \twoheadrightarrow g(\mathcal{X}, \mathcal{Y}))\ (f(a, b))$

$(f(\mathcal{X}, \mathcal{Y}) \twoheadrightarrow g(\mathcal{X}, \mathcal{Y}))\ (g(a, b))$

$(a \twoheadrightarrow b, a \twoheadrightarrow c)\ a$

Some ρ-reductions

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ a$

$(\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X}))\ (\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X})) \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ \omega\ \omega \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ \ldots$

$(a \twoheadrightarrow b)\ a \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} b$

$(f(\mathcal{X},\mathcal{Y}) \twoheadrightarrow g(\mathcal{X},\mathcal{Y}))\ (f(a,b))$

$(f(\mathcal{X},\mathcal{Y}) \twoheadrightarrow g(\mathcal{X},\mathcal{Y}))\ (g(a,b))$

$(a \twoheadrightarrow b, a \twoheadrightarrow c)\ a$

Some ρ-reductions

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ a$

$(\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X}))\ (\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X})) \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ \omega\ \omega \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ \ldots$

$(a \twoheadrightarrow b)\ a \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} \ b$

$(f(\mathcal{X},\mathcal{Y}) \twoheadrightarrow g(\mathcal{X},\mathcal{Y}))\ (f(a,b)) \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} [f(\mathcal{X},\mathcal{Y}) \ll f(a,b)]g(\mathcal{X},\mathcal{Y}) \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta} g(a,b)$

$(f(\mathcal{X},\mathcal{Y}) \twoheadrightarrow g(\mathcal{X},\mathcal{Y}))\ (g(a,b))$

$(a \twoheadrightarrow b, a \twoheadrightarrow c)\ a$

Some ρ-reductions

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a \qquad \mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta}\ a$

$(\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X}))\ (\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X})) \qquad \mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta}\ \omega\ \omega \mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta}\ \ldots$

$(a \twoheadrightarrow b)\ a \qquad \mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta}\ b$

$(f(\mathcal{X},\mathcal{Y}) \twoheadrightarrow g(\mathcal{X},\mathcal{Y}))\ (f(a,b))$
$\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta}\ [f(\mathcal{X},\mathcal{Y}) \ll f(a,b)]g(\mathcal{X},\mathcal{Y}) \mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta}\ g(a,b)$

$(f(\mathcal{X},\mathcal{Y}) \twoheadrightarrow g(\mathcal{X},\mathcal{Y}))\ (g(a,b)) \qquad \mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta}\ [f(\mathcal{X},\mathcal{Y}) \ll g(a,b)]g(\mathcal{X},\mathcal{Y})$

$(a \twoheadrightarrow b, a \twoheadrightarrow c)\ a$

Some ρ-reductions

$(\mathcal{X} \twoheadrightarrow \mathcal{X})\ a \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta}\ a$

$(\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X}))\ (\mathcal{X} \twoheadrightarrow (\mathcal{X}\ \mathcal{X})) \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta}\ \omega\ \omega \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta}\ \ldots$

$(a \twoheadrightarrow b)\ a \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta}\ b$

$(f(\mathcal{X},\mathcal{Y}) \twoheadrightarrow g(\mathcal{X},\mathcal{Y}))\ (f(a,b))$
$\mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta}\ [f(\mathcal{X},\mathcal{Y}) \ll f(a,b)]g(\mathcal{X},\mathcal{Y}) \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta}\ g(a,b)$

$(f(\mathcal{X},\mathcal{Y}) \twoheadrightarrow g(\mathcal{X},\mathcal{Y}))\ (g(a,b)) \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta}\ [f(\mathcal{X},\mathcal{Y}) \ll g(a,b)]g(\mathcal{X},\mathcal{Y})$

$(a \twoheadrightarrow b, a \twoheadrightarrow c)\ a \qquad \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta}\ (a \twoheadrightarrow b)\ a, (a \twoheadrightarrow c)\ a \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\delta}\ b, c$

Simple Success Reduction

$$\underline{(f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})\ f(3)} \mapsto_{\rho} \underline{[f(\mathcal{X}) \ll f(3)]((3 \twoheadrightarrow 3)\ \mathcal{X})}$$
$$\mapsto_{\sigma} \underline{(3 \twoheadrightarrow 3)\ 3}$$
$$\mapsto_{\rho} \underline{[3 \ll 3]3}$$
$$\mapsto_{\sigma} 3$$

Simple Success Reduction

$$
\begin{array}{lll}
\underline{(f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})\ f(3)} & \mapsto_{\rho} & \underline{[f(\mathcal{X}) \ll f(3)]((3 \twoheadrightarrow 3)\ \mathcal{X})} \\
 & \mapsto_{\sigma} & \underline{(3 \twoheadrightarrow 3)\ 3} \\
 & \mapsto_{\rho} & \underline{[3 \ll 3]3} \\
 & \mapsto_{\sigma} & 3
\end{array}
$$

$$
\begin{array}{lll}
(f(\mathcal{X}) \twoheadrightarrow \underline{(3 \twoheadrightarrow 3)\mathcal{X}})\ f(3) & \mapsto_{\rho} & \underline{(f(\mathcal{X}) \twoheadrightarrow [3 \ll \mathcal{X}]3)\ f(3)} \\
 & \mapsto_{\rho} & \underline{[f(\mathcal{X}) \ll f(3)]([3 \ll \mathcal{X}]3)} \\
 & \mapsto_{\sigma} & \underline{[3 \ll 3]3} \\
 & \mapsto_{\sigma} & 3
\end{array}
$$

Simple Failure Reduction

$$\underline{(f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})\ f(4)} \mapsto_{\rho} \underline{[f(\mathcal{X}) \ll f(4)]((3 \twoheadrightarrow 3)\ \mathcal{X})}$$

$$\mapsto_{\sigma} \underline{(3 \twoheadrightarrow 3)\ 4}$$

$$\mapsto_{\rho} \underline{[3 \ll 4]3}$$

Simple Failure Reduction

$$\underline{(f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})\ f(4)} \mapsto_{\rho} \underline{[f(\mathcal{X}) \ll f(4)]((3 \twoheadrightarrow 3)\ \mathcal{X})}$$

$$\mapsto_{\sigma} \underline{(3 \twoheadrightarrow 3)\ 4}$$

$$\mapsto_{\rho} \underline{[3 \ll 4]3}$$

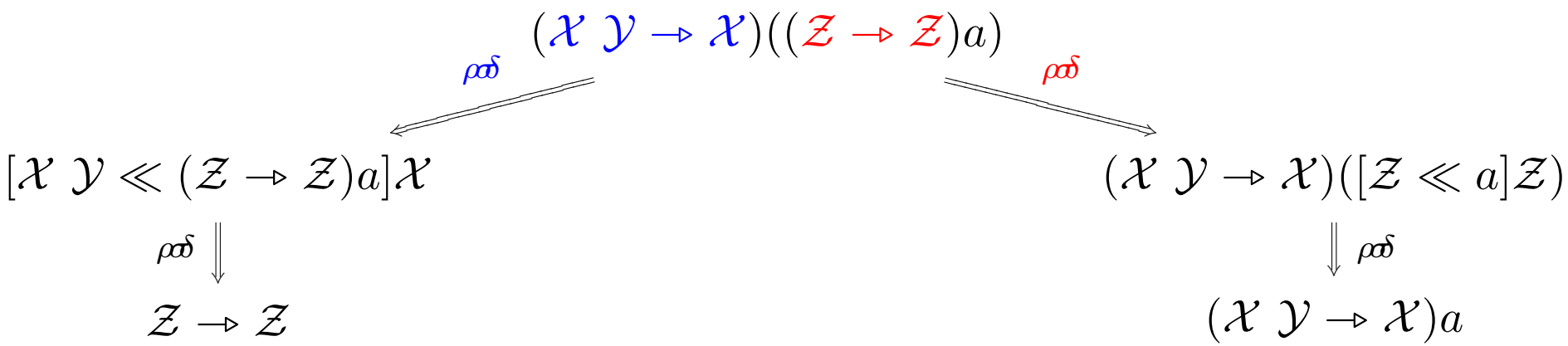
$$(f(\mathcal{X}) \twoheadrightarrow \underline{(3 \twoheadrightarrow 3)\mathcal{X}})\ f(4) \mapsto_{\rho} \underline{(f(\mathcal{X}) \twoheadrightarrow [3 \ll \mathcal{X}]3)\ f(4)}$$

$$\mapsto_{\rho} \underline{[f(\mathcal{X}) \ll f(4)]([3 \ll \mathcal{X}]3)}$$

$$\mapsto_{\sigma} \underline{[3 \ll 4]3}$$

On the (non-)confluence

Variables in applicative position



On the (non-)confluence

Non-linear patterns

$$\mathcal{C} \triangleq rec \twoheadrightarrow S \twoheadrightarrow X \twoheadrightarrow (d(Y,Y) \twoheadrightarrow e)\ d(X, S.rec\ X)$$

$$\mathcal{A} \triangleq rec' \twoheadrightarrow S' \twoheadrightarrow \mathcal{C}.rec\ S'.rec'$$

On the (non-)confluence

Non-linear patterns

$$\mathcal{C} \triangleq rec \twoheadrightarrow S \twoheadrightarrow X \twoheadrightarrow (d(Y,Y) \twoheadrightarrow e)\ d(X, S.rec\ X)$$

$$\mathcal{A} \triangleq rec' \twoheadrightarrow S' \twoheadrightarrow \mathcal{C}.rec\ S'.rec'$$

$$\begin{aligned}
&\mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} \mathcal{C}.rec\ \mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} (d(Y,Y) \twoheadrightarrow e)\ d(\mathcal{A}.rec', \mathcal{C}.rec\ \mathcal{A}.rec') \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} (d(Y,Y) \twoheadrightarrow e)\ d(\mathcal{C}.rec\ \mathcal{A}.rec', \mathcal{C}.rec\ \mathcal{A}.rec') \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} e
\end{aligned}$$

$$\begin{aligned}
&\mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} \mathcal{C}.rec\ \mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} \mathcal{C}.rec\ e
\end{aligned}$$

On the (non-)confluence

Non-linear patterns

$$\mathcal{C} \triangleq rec \twoheadrightarrow S \twoheadrightarrow X \twoheadrightarrow (d(Y,Y) \twoheadrightarrow e)\ d(X, S.rec\ X)$$

$$\mathcal{A} \triangleq rec' \twoheadrightarrow S' \twoheadrightarrow \mathcal{C}.rec\ S'.rec'$$

$$\begin{aligned}
&\mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} \mathcal{C}.rec\ \mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} (d(Y,Y) \twoheadrightarrow e)\ d(\mathcal{A}.rec', \mathcal{C}.rec\ \mathcal{A}.rec') \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} (d(Y,Y) \twoheadrightarrow e)\ d(\mathcal{C}.rec\ \mathcal{A}.rec', \mathcal{C}.rec\ \mathcal{A}.rec') \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} e
\end{aligned}$$

$$\begin{aligned}
&\mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} \mathcal{C}.rec\ \mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} \mathcal{C}.rec\ e
\end{aligned}$$

and

$$\mathcal{C}.rec\ e \mapsto (d(Y,Y) \twoheadrightarrow e)\ d(e, \mathcal{C}.rec\ e)$$

On the (non-)confluence

Non-linear patterns

$$\mathcal{C} \triangleq rec \twoheadrightarrow S \twoheadrightarrow X \twoheadrightarrow (d(Y,Y) \twoheadrightarrow e)\ d(X, S.rec\ X)$$

$$\mathcal{A} \triangleq rec' \twoheadrightarrow S' \twoheadrightarrow \mathcal{C}.rec\ S'.rec'$$

$$\begin{aligned}
&\mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} \mathcal{C}.rec\ \mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} (d(Y,Y) \twoheadrightarrow e)\ d(\mathcal{A}.rec', \mathcal{C}.rec\ \mathcal{A}.rec') \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} (d(Y,Y) \twoheadrightarrow e)\ d(\mathcal{C}.rec\ \mathcal{A}.rec', \mathcal{C}.rec\ \mathcal{A}.rec') \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} e
\end{aligned}$$

$$\begin{aligned}
&\mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} \mathcal{C}.rec\ \mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\alpha\delta} \mathcal{C}.rec\ e
\end{aligned}$$

and

$$\mathcal{C}.rec\ e \mapsto (d(Y,Y) \twoheadrightarrow e)\ d(e, \mathcal{C}.rec\ e)$$

and e and $\mathcal{C}.rec\ e$ have no common reduction (by induction on the supposed length of a reduction from $\mathcal{C}.rec\ e$ to e).

On the (non-)confluence

Non-linear patterns

$$\mathcal{C} \triangleq rec \twoheadrightarrow S \twoheadrightarrow X \twoheadrightarrow (d(Y,Y) \twoheadrightarrow e)\ d(X, S.rec\ X)$$
$$\mathcal{A} \triangleq rec' \twoheadrightarrow S' \twoheadrightarrow \mathcal{C}.rec\ S'.rec'$$

$$\begin{aligned}
&\mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} \mathcal{C}.rec\ \mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} (d(Y,Y) \twoheadrightarrow e)\ d(\mathcal{A}.rec', \mathcal{C}.rec\ \mathcal{A}.rec') \\
&\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} (d(Y,Y) \twoheadrightarrow e)\ d(\mathcal{C}.rec\ \mathcal{A}.rec', \mathcal{C}.rec\ \mathcal{A}.rec') \\
&\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} e
\end{aligned}$$

$$\begin{aligned}
&\mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} \mathcal{C}.rec\ \mathcal{A}.rec' \\
&\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} \mathcal{C}.rec\ e
\end{aligned}$$

and

$$\mathcal{C}.rec\ e \mapsto (d(Y,Y) \twoheadrightarrow e)\ d(e, \mathcal{C}.rec\ e)$$

and e and $\mathcal{C}.rec\ e$ have no common reduction (by induction on the supposed length of a reduction from $\mathcal{C}.rec\ e$ to e).

Rigid Pattern Condition (RPC) [van Oostrom 90]

$\mathcal{P} \triangleq \{T \in NF(\rho\sigma\delta) \mid T$ is "linear" with no "active" variables$\}$

Big-step Operational Semantics

- Natural proof deduction system à la Kahn
- Map every closed expression into a term in *weak head normal form*
- We present a *lazy call-by-name* strategy

Big-step Operational Semantics

- Natural proof deduction system à la Kahn
- Map every closed expression into a term in *weak head normal form*
- We present a *lazy call-by-name* strategy

$$\mathcal{V} ::= \mathcal{K} \mid \mathcal{T} \twoheadrightarrow \mathcal{T} \mid \mathcal{K}\,\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

$$\mathcal{O} ::= \mathcal{V} \mid \mathsf{wrong} \mid \mathcal{O}, \mathcal{O}$$

Big-step Operational Semantics

- Natural proof deduction system à la Kahn
- Map every closed expression into a term in *weak head normal form*
- We present a *lazy call-by-name* strategy

$$\mathcal{V} \ ::= \ \mathcal{K} \mid \mathcal{T} \twoheadrightarrow \mathcal{T} \mid \mathcal{K}\,\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

$$\mathcal{O} \ ::= \ \mathcal{V} \mid \mathsf{wrong} \mid \mathcal{O}, \mathcal{O}$$

- The special output wrong represents the result obtained by a computation involving a "matching equation failure"

Big-step Operational Semantics

- Natural proof deduction system à la Kahn
- Map every closed expression into a term in *weak head normal form*
- We present a *lazy call-by-name* strategy

$$\begin{aligned} \mathcal{V} &::= \mathcal{K} \mid \mathcal{T} \twoheadrightarrow \mathcal{T} \mid \mathcal{K}\,\mathcal{T} \mid \mathcal{T}, \mathcal{T} \\ \mathcal{O} &::= \mathcal{V} \mid \mathsf{wrong} \mid \mathcal{O}, \mathcal{O} \end{aligned}$$

- The special output wrong represents the result obtained by a computation involving a "matching equation failure"
- The semantics is defined via a judgment of the shape $\mathcal{T} \Downarrow \mathcal{O}$

Big-step Operational Semantics

- Natural proof deduction system à la Kahn
- Map every closed expression into a term in *weak head normal form*
- We present a *lazy call-by-name* strategy

$$\mathcal{V} ::= \mathcal{K} \mid \mathcal{T} \twoheadrightarrow \mathcal{T} \mid \mathcal{K}\,\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

$$\mathcal{O} ::= \mathcal{V} \mid \mathsf{wrong} \mid \mathcal{O}, \mathcal{O}$$

- The special output wrong represents the result obtained by a computation involving a "matching equation failure"
- The semantics is defined via a judgment of the shape $\mathcal{T} \Downarrow \mathcal{O}$
- The big-step is deterministic, and immediately suggests how to build an interpreter for the calculus;

Big-step Natural Semantics I

$$\frac{}{\mathcal{V} \Downarrow \mathcal{V}}\,(Red{-}Val)$$

Big-step Natural Semantics I

$$\frac{}{\mathcal{V} \Downarrow \mathcal{V}}\,(Red{-}Val)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathcal{T}_3 \twoheadrightarrow \mathcal{T}_4 \quad [\mathcal{T}_3 \ll \mathcal{T}_2]\mathcal{T}_4 \Downarrow \mathcal{O}}{\mathcal{T}_1\ \mathcal{T}_2 \Downarrow \mathcal{O}}\,(Red{-}\rho_1)$$

Big-step Natural Semantics I

$$\frac{}{\mathcal{V} \Downarrow \mathcal{V}}\,(Red{-}Val)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathcal{T}_3 \twoheadrightarrow \mathcal{T}_4 \quad [\mathcal{T}_3 \ll \mathcal{T}_2]\mathcal{T}_4 \Downarrow \mathcal{O}}{\mathcal{T}_1\ \mathcal{T}_2 \Downarrow \mathcal{O}}\,(Red{-}\rho_1)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathcal{T}_3, \mathcal{T}_4 \quad \mathcal{T}_3\ \mathcal{T}_2 \Downarrow \mathcal{O}_1 \quad \mathcal{T}_4\ \mathcal{T}_2 \Downarrow \mathcal{O}_2}{\mathcal{T}_1\ \mathcal{T}_2 \Downarrow \mathcal{O}_1, \mathcal{O}_2}\,(Red{-}\delta)$$

Big-step Natural Semantics II

$$\frac{\exists\sigma.\ \sigma(\mathcal{T}_1) \equiv \mathcal{T}_2 \quad \sigma(\mathcal{T}_3) \Downarrow \mathcal{O}}{[\mathcal{T}_1 \ll \mathcal{T}_2]\mathcal{T}_3 \Downarrow \mathcal{O}}\,(Red{-}\sigma_1)$$

Big-step Natural Semantics II

$$\frac{\exists\sigma.\ \sigma(\mathcal{T}_1) \equiv \mathcal{T}_2 \quad \sigma(\mathcal{T}_3) \Downarrow \mathcal{O}}{[\mathcal{T}_1 \ll \mathcal{T}_2]\mathcal{T}_3 \Downarrow \mathcal{O}}\,(Red{-}\sigma_1)$$

$$\frac{\nexists\sigma.\ \sigma(\mathcal{T}_1) \equiv \mathcal{T}_2}{[\mathcal{T}_1 \ll \mathcal{T}_2]\mathcal{T}_3 \Downarrow \mathsf{wrong}}\,(Red{-}\sigma_2)$$

Big-step Natural Semantics II

$$\frac{\exists\sigma.\ \sigma(\mathcal{T}_1) \equiv \mathcal{T}_2 \quad \sigma(\mathcal{T}_3) \Downarrow \mathcal{O}}{[\mathcal{T}_1 \ll \mathcal{T}_2]\mathcal{T}_3 \Downarrow \mathcal{O}}\,(Red{-}\sigma_1)$$

$$\frac{\nexists\sigma.\ \sigma(\mathcal{T}_1) \equiv \mathcal{T}_2}{[\mathcal{T}_1 \ll \mathcal{T}_2]\mathcal{T}_3 \Downarrow \mathsf{wrong}}\,(Red{-}\sigma_2)$$

$$\frac{\mathcal{T}_1 \Downarrow \mathsf{wrong}}{\mathcal{T}_1, \mathcal{T}_2 \Downarrow \mathsf{wrong}}\,(Red{-}\rho_2)$$

ρ-ample: Two Natural Deductions

Take the term $(f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})\, f(3)$

$$\dfrac{\dfrac{\sigma \equiv \{3/\mathcal{X}\} \quad (3 \twoheadrightarrow \vdots\, 3)3 \Downarrow 3}{[f(\mathcal{X}) \ll f(3)](3 \twoheadrightarrow 3)\mathcal{X} \Downarrow 3}\, \overline{*}}{(f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})\, f(3) \Downarrow 3}$$

with $* \equiv (f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X}) \Downarrow (f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})$.

ρ-ample: Two Natural Deductions

Take the term $(f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})\, f(4)$

$$\dfrac{\dfrac{\sigma \equiv \{4/\mathcal{X}\} \quad \dfrac{\nexists \sigma.\ \sigma(3) \equiv 4}{(3 \twoheadrightarrow 3)4 \Downarrow \text{wrong}}}{[f(\mathcal{X}) \ll f(4)](3 \twoheadrightarrow 3)\mathcal{X} \Downarrow \text{wrong}}\ *}{(f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})\, f(4) \Downarrow \text{wrong}}$$

with $* \equiv (f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X}) \Downarrow (f(\mathcal{X}) \twoheadrightarrow (3 \twoheadrightarrow 3)\mathcal{X})$.

The Rationale of Optimistic and Pessimistic Machines

Optimistic machine tracks the fact that one computation goes wrong.

$$\frac{\vdots}{(3 \rightarrowtriangle 3, 4 \rightarrowtriangle 4)\, 4 \Downarrow \mathsf{wrong}, 4}$$

The Rationale of Optimistic and Pessimistic Machines

Optimistic machine tracks the fact that one computation goes wrong.

$$\frac{\vdots}{(3 \to 3, 4 \to 4)\, 4 \Downarrow \mathsf{wrong}, 4}$$

- Pessimistic machine "kills" the computation once a wrong value is produced

$$\frac{\vdots}{(3 \to 3, 4 \to 4)\, 4 \Downarrow \mathsf{wrong}}$$

The Rationale of Optimistic and Pessimistic Machines

Optimistic machine tracks the fact that one computation goes wrong.

$$\frac{\vdots}{(3 \to 3, 4 \to 4)\, 4 \Downarrow \mathsf{wrong}, 4}$$

- Pessimistic machine "kills" the computation once a wrong value is produced

$$\frac{\vdots}{(3 \to 3, 4 \to 4)\, 4 \Downarrow \mathsf{wrong}}$$

- Optimistic machine (at least one computation does not go wrong), *vs.* Pessimistic machine (the machine stops if at least one wrong occurs).

Why a new calculus?

Rewriting is nice, but

- the rewrite relation or the rewriting logic are difficult to control
- non-reducibility is impossible to express

Why a new calculus?

Rewriting is nice, but

- the rewrite relation or the rewriting logic are difficult to control
- non-reducibility is impossible to express

Lambda-calculus is great, but

- lacks of discrimination capabilities
- difficult to control

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

In "basic" rewriting calculus,

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

In "basic" rewriting calculus,

- rules are first class object

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

In "basic" rewriting calculus,

- rules are first class object
- application is explicit

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

In "basic" rewriting calculus,

- rules are first class object
- application is explicit
- decision of redex reduction is explicit

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

In "basic" rewriting calculus,

- rules are first class object
- application is explicit
- decision of redex reduction is explicit
- matching is a main explicit parameter

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

In "basic" rewriting calculus,

- rules are first class object
- application is explicit
- decision of redex reduction is explicit
- matching is a main explicit parameter
- results are first class

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

In "basic" rewriting calculus,

- rules are first class object
- application is explicit
- decision of redex reduction is explicit
- matching is a main explicit parameter
- results are first class

This allows for advanced calculi,

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

In "basic" rewriting calculus,

- rules are first class object
- application is explicit
- decision of redex reduction is explicit
- matching is a main explicit parameter
- results are first class

This allows for advanced calculi,

- with explicit constraints (and substitutions),

A calculus with more explicit features

Like for explicit substitution calculi, explicitation of all the ingredients is useful.

In "basic" rewriting calculus,

- rules are first class object
- application is explicit
- decision of redex reduction is explicit
- matching is a main explicit parameter
- results are first class

This allows for advanced calculi,

- with explicit constraints (and substitutions),
- that can dissociate binding from matching when abstracting

Comparing the untyped rewriting calculus to other formalisms

Target formalisms

- Rewriting
- λ-calculus
- Higher-order term rewriting (CRS)
- The Lambda Calculus of Objects $\lambda\mathcal{O}bj$
- The Object Calculus $\varsigma\mathcal{O}bj$

Compiling the λ-calculus into the ρ-calculus

1. $\varphi(X) = X$
2. $\varphi(\lambda X.M) = X \twoheadrightarrow \varphi(M)$
3. $\varphi(M\ N) = \varphi(M)\ \varphi(N)$

Theorem: If $M \mapsto_\beta N$, then $\varphi(M) \mapsto\!\!\!\!\!\twoheadrightarrow_{\rho\sigma\delta} \varphi(N)$.

Compiling the λ-calculus into the ρ-calculus

1. $\varphi(X) = X$
2. $\varphi(\lambda X.M) = X \twoheadrightarrow \varphi(M)$
3. $\varphi(M\ N) = \varphi(M)\ \varphi(N)$

Theorem: If $M \mapsto_\beta N$, then $\varphi(M) \mapsto\!\!\!\!\!\rightarrow_{\rho\mathfrak{dB}} \varphi(N)$.

Example: for Turing's fixpoint combinator [Turing37]

$$\Theta_\lambda = (A_\lambda\ A_\lambda) \quad \text{where} \quad A_\lambda = \lambda xy.y(xxy)$$

we have $\varphi(A_\lambda) = x \twoheadrightarrow (y \twoheadrightarrow y\ (x\ x\ y))$ and thus, to the λ-reduction

$$\Theta_\lambda\ G \longrightarrow_\beta G(\Theta_\lambda\ G)$$

corresponds the following reduction in the rewriting calculus

$\Theta\ G \triangleq (x \twoheadrightarrow (y \twoheadrightarrow y\ (x\ x\ y)))\ A\ G \mapsto\!\!\!\!\!\rightarrow_{\rho\mathfrak{dB}} (y \twoheadrightarrow y\ (A\ A\ y))\ G \mapsto\!\!\!\!\!\rightarrow_{\rho\mathfrak{dB}} G\ (A\ A\ G)$

Rewriting: The relation, the logic and the calculus

Given a set $\mathcal{R}$ of rewrite rules $(l_i \twoheadrightarrow r_i)$, we can define:

Rewriting: The relation, the logic and the calculus

Given a set $\mathcal{R}$ of rewrite rules $(l_i \twoheadrightarrow r_i)$, we can define:

The rewrite relation:

$$t \longrightarrow_{\mathcal{R}} t'$$

i.e. the smallest relation containing the rewrite rules and stable by context et substitution.

Rewriting: The relation, the logic and the calculus

Given a set $\mathcal{R}$ of labeled rewrite rules $([\ell_i]\ l_i \twoheadrightarrow r_i)$, we can define:

Rewriting: The relation, the logic and the calculus

Given a set $\mathcal{R}$ of labeled rewrite rules $([\ell_i]\ l_i \twoheadrightarrow r_i)$, we can define:

The rewriting logic [José Meseguer, TCS1992]:

$$\mathcal{R} \vdash t \Rightarrow t'$$

Rewriting: The relation, the logic and the calculus

Given a set $\mathcal{R}$ of labeled rewrite rules $([\ell_i]\ l_i \twoheadrightarrow r_i)$, we can define:

The rewriting logic [José Meseguer, TCS1992]:

$$\mathcal{R} \vdash t \Rightarrow t'$$

Formulas are sequents of the form

$$\pi : t \Rightarrow t'$$

where π is a proof term, built on $\mathcal{F} \cup L \cup \{;\}$ recording the proof of the sequent.

Rewriting: The relation, the logic and the calculus

$$\mathcal{R} \vdash \pi : t \Rightarrow t'$$

if $\pi : t \Rightarrow t'$ can be obtained by finite application of the following rules:

Reflexivity For any $t \in \mathcal{T}(\mathcal{F})$:

$$t : t \Rightarrow t$$

Transitivity

$$\frac{\pi_1 : t_1 \Rightarrow t_2 \qquad \pi_2 : t_2 \Rightarrow t_3}{\pi_1; \pi_2 \; : \; t_1 \Rightarrow t_3}$$

Congruence For any $f \in \mathcal{F}$ with $arity(f) = n$:

$$\frac{\pi_1 : t_1 \Rightarrow t'_1 \quad \ldots \quad \pi_n : t_n \Rightarrow t'_n}{f(\pi_1, \ldots, \pi_n) : f(t_1, \ldots, t_n) \Rightarrow f(t'_1, \ldots, t'_n)}$$

Replacement For any $\ell : l(x_1, \ldots, x_n) \Rightarrow r(x_1, \ldots, x_n) \in R$,

$$\frac{\pi_1 : t_1 \Rightarrow t'_1 \quad \ldots \quad \pi_n : t_n \Rightarrow t'_n}{\ell(\pi_1, \ldots, \pi_n) : l(t_1, \ldots, t_n) \Rightarrow r(t'_1, \ldots, t'_n)}$$

Rewriting: The relation, the logic and the calculus

Given a set $\mathcal{R}$ of rewrite rules $(l_i \twoheadrightarrow r_i)$, we can define:

Rewriting: The relation, the logic and the calculus

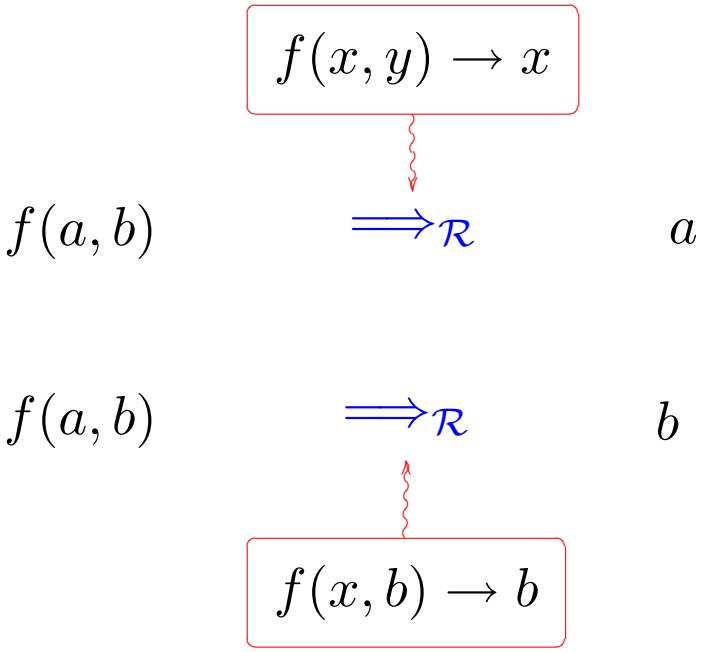
Given a set $\mathcal{R}$ of rewrite rules $(l_i \twoheadrightarrow r_i)$, we can define:

The rewriting calculus:

$$(lmim(\mathcal{R}) \;\; t) \;\; \mapsto\!\!\!\!\!\rightarrow_{\rho\rho\delta} \;\; t'$$

Going from rewrite rule to rewriting system:
For the rewriting reduction

$$\begin{cases} f(x,y) \to x \\ f(x,b) \to b \end{cases}$$



Non Determinism

Going from rewrite rule to rewriting system: In the ρ-calculus

$f(X, Y) \twoheadrightarrow X \qquad f(X, b) \twoheadrightarrow b$

Going from rewrite rule to rewriting system: In the ρ-calculus

$$f(X,Y) \twoheadrightarrow X, \qquad f(X,b) \twoheadrightarrow b$$

Going from rewrite rule to rewriting system: In the ρ-calculus

$f(X,Y) \twoheadrightarrow X, \qquad f(X,b) \twoheadrightarrow b \qquad f(a,b)$

Going from rewrite rule to rewriting system: In the ρ-calculus

$$\Big(f(X,Y) \twoheadrightarrow X, \qquad f(X,b) \twoheadrightarrow b\Big) \qquad f(a,b)$$

Going from rewrite rule to rewriting system: In the ρ-calculus

$$\Big(f(X,Y) \twoheadrightarrow X, \qquad f(X,b) \twoheadrightarrow b\Big) \qquad f(a,b)$$

$$\hookrightarrow \quad (f(X,Y) \twoheadrightarrow X)\ f(a,b), (f(X,b) \twoheadrightarrow b)\ f(a,b)$$

Going from rewrite rule to rewriting system: In the ρ-calculus

$$\Big(f(X,Y) \twoheadrightarrow X, \qquad f(X,b) \twoheadrightarrow b\Big) \qquad f(a,b)$$

$\rightarrowtail$ $(f(X,Y) \twoheadrightarrow X)\ f(a,b), (f(X,b) \twoheadrightarrow b)\ f(a,b)$

$\rightarrowtail$ (a,b)

Encoding Rewriting

Encoding Rewriting

Starting from the proof term, a corresponding ρ-term is obtained:

Theorem: If $T_1 \mapsto_{\mathcal{R}} T_2$, then $\exists\, T_{\mathcal{R}}$ such that $T_{\mathcal{R}}\ T_1 \mapsto\!\!\!\!\!\!\rightarrow_{\rho\partial} T_2$.

Encoding Rewriting

Starting from the proof term, a corresponding ρ-term is obtained:

Theorem: If $T_1 \mapsto_{\mathcal{R}} T_2$, then $\exists\, T_{\mathcal{R}}$ such that $T_{\mathcal{R}}\ T_1 \mapsto\!\!\!\!\!\!\rightarrow_{\rho\partial\delta} T_2$.

Example:
Given $R = \{\ell 1 : a \Rightarrow b, \ell 2 : c \Rightarrow d, \ell 3 : f(x, h(y)) \Rightarrow f(y, h(x))\}$ we have:

$$\ell 3(g(\ell 1), \ell 2) : f(g(a), h(c)) \Rightarrow f(d, h(g(b)))$$

and in rewriting calculus

$$(f(x, h(y)) \twoheadrightarrow f((c \twoheadrightarrow d)\ y, h((g(z) \twoheadrightarrow (a \twoheadrightarrow b)\ z)\ x)))\ f(g(a), h(c)) \mapsto\!\!\!\!\!\!\rightarrow_{\rho\partial\delta} f(d, h(g(b)))$$

Staring from a given system of rewrite rules reduction strategies can be encoded (using $\rho_{\twoheadrightarrow}$)

Encoding Rewriting

Representation of Higher-order term rewriting (CRS)

- Higher-order term rewriting $= Term\ rewriting + Lambda\ calculus$
[Breazu-Tannen,Gallier88],[Okada89],[Klop80],[Nipkow91]
- CRS versus TRS: abstraction and metavariables.

Representation of Higher-order term rewriting (CRS)

- Higher-order term rewriting $=$ $Term\ rewriting + Lambda\ calculus$
[Breazu-Tannen,Gallier88],[Okada89],[Klop80],[Nipkow91]
- CRS versus TRS: abstraction and metavariables.

$App([x]Z(x), Z') \Rightarrow Z(Z')$ $\qquad$ $(\lambda x.t)u \Rightarrow_\beta t\{x/u\}$

$App([x]f(x), a)$ $\qquad$ $(\lambda x.f)\ a$

Representation of Higher-order term rewriting (CRS)

- Higher-order term rewriting $=$ $Term\ rewriting$+$Lambda\ calculus$
[Breazu-Tannen,Gallier88],[Okada89],[Klop80],[Nipkow91]
- CRS versus TRS: abstraction and metavariables.

$App([x]Z(x), Z') \Rightarrow Z(Z')$ $\qquad$ $(\lambda x.t)u \Rightarrow_\beta t\{x/u\}$

$App([x]f(x), a)$ $\qquad$ $(\lambda x.f)\ a$

Reduction : $\qquad \sigma L \longrightarrow_{CRS} \sigma R$

Representation of Higher-order term rewriting (CRS)

- Higher-order term rewriting $=$ $Term\ rewriting$+$Lambda\ calculus$
[Breazu-Tannen,Gallier88],[Okada89],[Klop80],[Nipkow91]
- CRS versus TRS: abstraction and metavariables.

$App([x]Z(x), Z') \Rightarrow Z(Z')$ $\qquad$ $(\lambda x.t)u \Rightarrow_\beta t\{x/u\}$

$App([x]f(x), a)$ $\qquad$ $(\lambda x.f)\ a$

Reduction : $\qquad \sigma L \longrightarrow_{CRS} \sigma R$

$\sigma = \{(Z, \lambda y.fy), (Z', a)\}$

$\sigma L = \sigma(App([x]Z(x), Z')) = App([x](\lambda y.fy)(x), a) \downarrow_\beta = App([x]f(x), a)$

$\sigma R = \sigma(Z(Z')) = (\lambda y.fy)(a) \downarrow_\beta = f(a)$

Representation of Higher-order term rewriting (CRS)

- Higher-order term rewriting $=$ $Term\ rewriting$+$Lambda\ calculus$
[Breazu-Tannen,Gallier88],[Okada89],[Klop80],[Nipkow91]
- CRS versus TRS: abstraction and metavariables.

$App([x]Z(x), Z') \Rightarrow Z(Z')$ $\qquad$ $(\lambda x.t)u \Rightarrow_\beta t\{x/u\}$

$App([x]f(x), a)$ $\qquad$ $(\lambda x.f)\ a$

Reduction : $\qquad \sigma L \longrightarrow_{CRS} \sigma R$

$\sigma = \{(Z, \lambda y.fy), (Z', a)\}$

$\sigma L = \sigma(App([x]Z(x), Z')) = App([x](\lambda y.fy)(x), a) \downarrow_\beta = App([x]f(x), a)$

$\sigma R = \sigma(Z(Z')) = (\lambda y.fy)(a) \downarrow_\beta = f(a)$

$App([x]f(x), a) \longrightarrow_{CRS} f(a)$ $\qquad$ $(\lambda x.f)\ a \longrightarrow_\beta f\{x/a\}$

Translation of CRS in the ρ-calculus

- Metaterms :
  - $\star$ $[\![x]\!] = x$
  - $\star$ $[\![f(t_1, \ldots, t_n)]\!] = f([\![t_1]\!], \ldots, [\![t_n]\!])$
  - $\star$ $[\![[x]t]\!] = x \rightarrowtriangle [\![t]\!]$
  - $\star$ $[\![Z(t_1, \ldots, t_n)]\!] = Z \;\; [\![t_1]\!] \;\; \ldots \;\; [\![t_n]\!]$

Example

Translation of CRS in the ρ-calculus

- Metaterms :
  - $[\![x]\!] = x$
  - $[\![f(t_1, \ldots, t_n)]\!] = f([\![t_1]\!], \ldots, [\![t_n]\!])$
  - $[\![[x]t]\!] = x \to [\![t]\!]$
  - $[\![Z(t_1, \ldots, t_n)]\!] = Z \;\; [\![t_1]\!] \;\; \ldots \;\; [\![t_n]\!]$

- Rewriting rules : $[\![L \;\Rightarrow\; R]\!] = [\![L]\!] \to [\![R]\!]$
- Assignment : $[\![\{\ldots, (Z_i, \xi_i), \ldots\}]\!] = \{\ldots, Z_i/[\![\xi_i]\!], \ldots\}$
- Substitute : $[\![\lambda x_1 \ldots x_n.u]\!] = x_1 \to (x_2 \to (\ldots (x_n \to [\![u]\!]) \ldots))$

Example

Representation of Higher-order term rewriting (CRS)

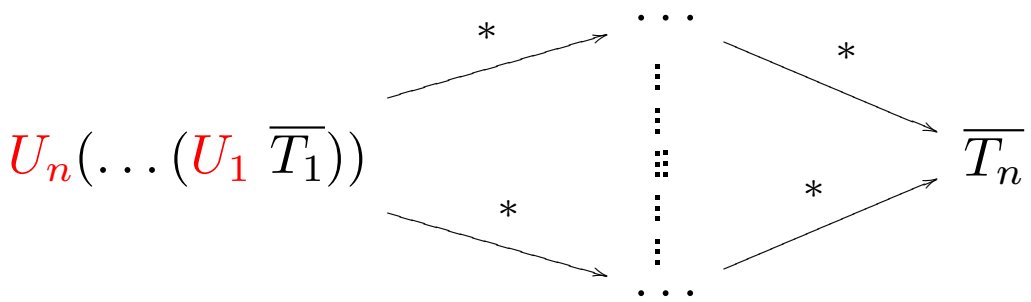
- Definition of the $\rho_{\P}$-calculus (HO matching theory)

$$[P \ll B]A \quad \to_{\sigma} \quad A\theta_{(P \lessdot B)} \qquad \text{if } \exists\theta.\ P\theta =_{\rho_\lambda} B$$

- Translation of CRS-components into RHO-terms

- Correction and completeness of the translation

Theorem: If $T_1 \mapsto_{\mathcal{R}} T_2 \ldots \mapsto_{\mathcal{R}} T_n$, then $\exists\ U_n \ldots U_1$ such that every correspondent RHO-derivation terminates and converges to $\overline{T_n}$



ρ-calculus and records

Record = structure composed of rewriting rules, $i.e.$:

$$
\begin{array}{rcll}
(T_i)^{i=1\ldots n} & \triangleq & T_1, \ldots, T_n & (n \in \mathsf{Nat}) \\
[m_i = T_i]^{i \in I} & \triangleq & (m_i \twoheadrightarrow T_i)^{i \in I} & \\
[cx = 0, cy = 0] & \triangleq & (cx \twoheadrightarrow 0, cy \twoheadrightarrow 0) &
\end{array}
$$

ρ-calculus and records

Record = structure composed of rewriting rules, $i.e.$:

$$
\begin{array}{rcll}
(T_i)^{i=1\ldots n} & \triangleq & T_1, \ldots, T_n & (n \in \mathsf{Nat}) \\
[m_i = T_i]^{i\in I} & \triangleq & (m_i \twoheadrightarrow T_i)^{i\in I} & \\
[cx = 0, cy = 0] & \triangleq & (cx \twoheadrightarrow 0, cy \twoheadrightarrow 0) &
\end{array}
$$

Record selection = the application of the record to the label, $i.e.$ $T_1.T_2$ as $T_1\ T_2$.

$$
\begin{array}{rcll}
(cx \twoheadrightarrow 0, cy \twoheadrightarrow 0)\ cx & \mapsto_{\delta} & (cx \twoheadrightarrow 0)\ cx, (cy \twoheadrightarrow 0)\ cx & \\
& \mapsto_{\rho\varpi\delta} & 0, [cy \ll cx]0 & \\
& =_{\mathbb{T}} & (0, \mathsf{stk}) =_{\mathbb{T}} 0 \quad (\mathbb{T}_{\varsigma\mathcal{O}bj}) & \text{. . . the matching theory}
\end{array}
$$

ρ-calculus and objects

Object $=$ record with an explicit account of self, $i.e.$

$$[m_i = \varsigma(\mathcal{X}_i)T_i]^{i\in I} \quad \triangleq \quad (m_i \twoheadrightarrow \mathcal{X}_i \twoheadrightarrow T_i)^{i\in I}$$

ρ-calculus and objects

Object $=$ record with an explicit account of self, $i.e.$

$$[m_i = \varsigma(\mathcal{X}_i)T_i]^{i\in I} \quad \triangleq \quad (m_i \twoheadrightarrow \mathcal{X}_i \twoheadrightarrow T_i)^{i\in I}$$

Self-application $=$ the application of an object to the object itself, $i.e.$

$$T_1.T_2 \quad \triangleq \quad T_1\ T_2\ T_1$$

ρ-calculus and objects

Object $=$ record with an explicit account of self, $i.e.$

$$[m_i = \varsigma(\mathcal{X}_i)T_i]^{i\in I} \quad \triangleq \quad (m_i \to \mathcal{X}_i \to T_i)^{i\in I}$$

Self-application $=$ the application of an object to the object itself, $i.e.$

$$T_1.T_2 \quad \triangleq \quad T_1\ T_2\ T_1$$

Ex: $T \triangleq a \to S \to b$. Then: $T.a \triangleq T\ a\ T \quad \mapsto\!\!\!\!\!\!\rightarrow_{\rho\partial\delta} \quad (S \to b)\ T \quad \mapsto\!\!\!\!\!\!\rightarrow_{\rho\partial\delta} \quad b$

ρ-calculus and objects

Object $=$ record with an explicit account of self, $i.e.$

$$[m_i = \varsigma(\mathcal{X}_i)T_i]^{i\in I} \quad \triangleq \quad (m_i \twoheadrightarrow \mathcal{X}_i \twoheadrightarrow T_i)^{i\in I}$$

Self-application $=$ the application of an object to the object itself, $i.e.$

$$T_1.T_2 \quad \triangleq \quad T_1\ T_2\ T_1$$

Ex: $T \triangleq a \twoheadrightarrow S \twoheadrightarrow b$. Then: $T.a \triangleq T\ a\ T \quad \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \quad (S \twoheadrightarrow b)\ T \quad \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \quad b$

Ex: $T \triangleq \omega \twoheadrightarrow S \twoheadrightarrow S.\omega$. Then: $T.\omega \quad \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \quad (S \twoheadrightarrow S.\omega)\ T \quad \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \quad T.\omega \quad \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \quad \ldots$

A "ping-pong" object

Let $T \triangleq (ping \twoheadrightarrow S \twoheadrightarrow S.pong, pong \twoheadrightarrow S \twoheadrightarrow S.ping)$
Then:

$$T.ping \quad \triangleq \quad T\ ping\ T$$

A "ping-pong" object

Let $T \triangleq (ping \twoheadrightarrow S \twoheadrightarrow S.pong, pong \twoheadrightarrow S \twoheadrightarrow S.ping)$
Then:

$$\begin{array}{rcl} T.ping & \triangleq & T\ ping\ T \\ & \longmapsto\!\!\!\!\!\rightarrow_{\rho\varpi\delta} & ((ping \twoheadrightarrow S \twoheadrightarrow S.pong)\ ping, \\ & & (pong \twoheadrightarrow S \twoheadrightarrow S.ping)\ ping)\ T \end{array}$$

A "ping-pong" object

Let $T \triangleq (ping \twoheadrightarrow S \twoheadrightarrow S.pong, pong \twoheadrightarrow S \twoheadrightarrow S.ping)$
Then:

$$\begin{array}{rcl} T.ping & \triangleq & T\ ping\ T \\ & \longmapsto_{\rho\omega\delta} & ((ping \twoheadrightarrow S \twoheadrightarrow S.pong)\ ping, \\ & & (pong \twoheadrightarrow S \twoheadrightarrow S.ping)\ ping)\ T \\ & \longmapsto_{\rho\omega\delta} & ((S \twoheadrightarrow S.pong), \mathsf{stk})\ T \end{array}$$

A "ping-pong" object

Let $T \triangleq (ping \twoheadrightarrow S \twoheadrightarrow S.pong, pong \twoheadrightarrow S \twoheadrightarrow S.ping)$
Then:

$$
\begin{array}{rcl}
T.ping & \triangleq & T\ ping\ T \\
& \longmapsto\!\!\!\!\!\rightarrow_{\rho\varpi\delta} & ((ping \twoheadrightarrow S \twoheadrightarrow S.pong)\ ping, \\
& & \ (pong \twoheadrightarrow S \twoheadrightarrow S.ping)\ ping)\ T \\
& \longmapsto\!\!\!\!\!\rightarrow_{\rho\varpi\delta} & ((S \twoheadrightarrow S.pong), \mathsf{stk})\ T \\
& \longmapsto\!\!\!\!\!\rightarrow_{\rho\varpi\delta} & (S \twoheadrightarrow S.pong)\ T, \mathsf{stk}\ T
\end{array}
$$

A "ping-pong" object

Let $T \triangleq (ping \twoheadrightarrow S \twoheadrightarrow S.pong, pong \twoheadrightarrow S \twoheadrightarrow S.ping)$
Then:

$$
\begin{array}{rcl}
T.ping & \triangleq & T\ ping\ T \\
& \longmapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} & ((ping \twoheadrightarrow S \twoheadrightarrow S.pong)\ ping, \\
& & \ (pong \twoheadrightarrow S \twoheadrightarrow S.ping)\ ping)\ T \\
& \longmapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} & ((S \twoheadrightarrow S.pong), \mathsf{stk})\ T \\
& \longmapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} & (S \twoheadrightarrow S.pong)\ T, \mathsf{stk}\ T \\
& \longmapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} & (S \twoheadrightarrow S.pong)\ T, \mathsf{stk}
\end{array}
$$

A "ping-pong" object

Let $T \triangleq (ping \twoheadrightarrow S \twoheadrightarrow S.pong, pong \twoheadrightarrow S \twoheadrightarrow S.ping)$
Then:

$$
\begin{array}{rcl}
T.ping & \triangleq & T\ ping\ T \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} & ((ping \twoheadrightarrow S \twoheadrightarrow S.pong)\ ping, \\
& & \ (pong \twoheadrightarrow S \twoheadrightarrow S.ping)\ ping)\ T \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} & ((S \twoheadrightarrow S.pong), \mathsf{stk})\ T \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} & (S \twoheadrightarrow S.pong)\ T, \mathsf{stk}\ T \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} & (S \twoheadrightarrow S.pong)\ T, \mathsf{stk} \\
& =_{\mathbb{T}} & (S \twoheadrightarrow S.pong)\ T \quad (\mathbb{T}_{\varsigma\mathcal{O}bj})
\end{array}
$$

A "ping-pong" object

Let $T \triangleq (ping \twoheadrightarrow S \twoheadrightarrow S.pong, pong \twoheadrightarrow S \twoheadrightarrow S.ping)$
Then:

$$
\begin{array}{rcl}
T.ping & \triangleq & T\ ping\ T \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\mathsf{stk}} & ((ping \twoheadrightarrow S \twoheadrightarrow S.pong)\ ping, \\
& & \ (pong \twoheadrightarrow S \twoheadrightarrow S.ping)\ ping)\ T \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\mathsf{stk}} & ((S \twoheadrightarrow S.pong), \mathsf{stk})\ T \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\mathsf{stk}} & (S \twoheadrightarrow S.pong)\ T, \mathsf{stk}\ T \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\mathsf{stk}} & (S \twoheadrightarrow S.pong)\ T, \mathsf{stk} \\
& =_{\mathbb{T}} & (S \twoheadrightarrow S.pong)\ T \quad (\mathbb{T}_{\varsigma\mathcal{O}bj}) \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\mathsf{stk}} & T.pong \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\mathsf{stk}} & T.ping \\
& \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\mathsf{stk}} & \ldots
\end{array}
$$

Functional object update

Update $(a.m := b) \stackrel{\triangle}{=} (a, m \twoheadrightarrow b)$

Functional object update

Update $\quad (a.m := b) \triangleq (a, m \twoheadrightarrow b)$

$$
\begin{aligned}
Point \quad \triangleq \quad & val \twoheadrightarrow S \twoheadrightarrow v(1,1), \\
& get \twoheadrightarrow S \twoheadrightarrow S.val, \\
& set \twoheadrightarrow S \twoheadrightarrow v(X,Y) \twoheadrightarrow (S.val := S' \twoheadrightarrow v(X,Y))
\end{aligned}
$$

Functional object update

Update $\quad (a.m := b) \triangleq (a, m \twoheadrightarrow b)$

$$\begin{aligned} Point \quad \triangleq \quad & val \twoheadrightarrow S \twoheadrightarrow v(1,1), \\ & get \twoheadrightarrow S \twoheadrightarrow S.val, \\ & set \twoheadrightarrow S \twoheadrightarrow v(X,Y) \twoheadrightarrow (S.val := S' \twoheadrightarrow v(X,Y)) \end{aligned}$$

Then:

$$Point.get \quad \mapsto\!\!\!\!\!\rightarrow_{\rho\infty\delta} \quad v(1,1)$$

Functional object update

Update $\quad (a.m := b) \triangleq (a, m \twoheadrightarrow b)$

$$\begin{aligned} Point \quad \triangleq \quad & val \twoheadrightarrow S \twoheadrightarrow v(1,1), \\ & get \twoheadrightarrow S \twoheadrightarrow S.val, \\ & set \twoheadrightarrow S \twoheadrightarrow v(X,Y) \twoheadrightarrow (S.val := S' \twoheadrightarrow v(X,Y)) \end{aligned}$$

Then:

$$\begin{aligned} Point.get &\longmapsto\!\!\!\!\!\rightarrow_{\rho\infty\delta} v(1,1) \\ Point.set(v(2,2)) &\longmapsto\!\!\!\!\!\rightarrow_{\rho\infty\delta} Point, (val \twoheadrightarrow S' \twoheadrightarrow v(2,2)) \end{aligned}$$

Functional object update

Update $\quad (a.m := b) \triangleq (a, m \twoheadrightarrow b)$

$$\begin{array}{rcl} Point & \triangleq & val \twoheadrightarrow S \twoheadrightarrow v(1,1), \\ & & get \twoheadrightarrow S \twoheadrightarrow S.val, \\ & & set \twoheadrightarrow S \twoheadrightarrow v(X,Y) \twoheadrightarrow (S.val := S' \twoheadrightarrow v(X,Y)) \end{array}$$

Then:

$$\begin{array}{rcl} Point.get & \mapsto\!\!\!\!\rightarrow_{\rho\infty\delta} & v(1,1) \\ Point.set(v(2,2)) & \mapsto\!\!\!\!\rightarrow_{\rho\infty\delta} & Point, (val \twoheadrightarrow S' \twoheadrightarrow v(2,2)) \\ Point.set(v(2,2)).get & \mapsto\!\!\!\!\rightarrow_{\rho\infty\delta} & v(1,1), v(2,2) \end{array}$$

Imperative object update

$Kill_m$ rule:

$$kill_m \triangleq (m \twoheadrightarrow X, Y) \twoheadrightarrow Y \qquad \text{(in } \mathbb{T}_{\varsigma\mathcal{O}bj}\text{)}$$

Imperative object update

$Kill_m$ rule:

$$kill_m \triangleq (m \twoheadrightarrow X, Y) \twoheadrightarrow Y \qquad \text{(in } \mathbb{T}_{\varsigma\mathcal{O}bj}\text{)}$$

Update $(a.m := b) \triangleq (kill_m(a), m \twoheadrightarrow b)$

Imperative object update

$Kill_m$ rule:

$$kill_m \quad \triangleq \quad (m \twoheadrightarrow X, Y) \twoheadrightarrow Y \qquad\qquad (\text{in } \mathbb{T}_{\varsigma\mathcal{O}bj})$$

Update $\quad (a.m := b) \triangleq (kill_m(a), m \twoheadrightarrow b)$

Then:

$$Point_I.get \quad \mapsto\!\!\!\!\!\!\rightarrow_{\rho\sigma\delta} \quad v(1,1)$$

Imperative object update

$Kill_m$ rule:

$$kill_m \triangleq (m \rightarrow X, Y) \rightarrow Y \qquad \text{(in } \mathbb{T}_{\varsigma\mathcal{O}bj}\text{)}$$

Update $(a.m := b) \triangleq (kill_m(a), m \rightarrow b)$

Then:

$$\begin{aligned} Point_I.get &\mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} v(1,1) \\ Point_I.set(v(2,2)) &\mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} val \rightarrow S' \rightarrow v(2,2), get \rightarrow \ldots, set \rightarrow \ldots \end{aligned}$$

Imperative object update

$Kill_m$ rule:

$$kill_m \triangleq (m \twoheadrightarrow X, Y) \twoheadrightarrow Y \qquad \text{(in } \mathbb{T}_{\varsigma\mathcal{O}bj}\text{)}$$

Update $(a.m := b) \triangleq (kill_m(a), m \twoheadrightarrow b)$

Then:

$$\begin{aligned} Point_I.get &\mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} v(1,1) \\ Point_I.set(v(2,2)) &\mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} val \twoheadrightarrow S' \twoheadrightarrow v(2,2), get \twoheadrightarrow \ldots, set \twoheadrightarrow \ldots \\ Point_I.set(v(2,2)).get &\mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} v(2,2) \end{aligned}$$

The Object Calculus $\varsigma\mathcal{O}bj$
The Lambda Calculus of Objects $\lambda\mathcal{O}bj$

An object with "self-extension"

Let $t_1 \triangleq add_n \twoheadrightarrow S \twoheadrightarrow (S, n \twoheadrightarrow S' \twoheadrightarrow 1)$

$$(t_1.add_n).n \quad \mapsto\!\!\!\!\!\rightarrow_{\rho\omega\delta} \quad ((S \twoheadrightarrow (S, n \twoheadrightarrow S' \twoheadrightarrow 1))\ t_1).n$$

An object with "self-extension"

Let $t_1 \triangleq add_n \to S \to (S, n \to S' \to 1)$

$$
\begin{aligned}
(t_1.add_n).n \quad &\mapsto\!\!\!\!\!\rightarrow_{\rho\omega\delta} \quad ((S \to (S, n \to S' \to 1))\ t_1).n \\
&\mapsto\!\!\!\!\!\rightarrow_{\rho\omega\delta} \quad \underbrace{(t_1, n \to S' \to 1)}_{t_2}.n
\end{aligned}
$$

An object with "self-extension"

Let $t_1 \triangleq add_n \twoheadrightarrow S \twoheadrightarrow (S, n \twoheadrightarrow S' \twoheadrightarrow 1)$

$$
\begin{array}{lll}
(t_1.add_n).n & \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} & ((S \twoheadrightarrow (S, n \twoheadrightarrow S' \twoheadrightarrow 1))\ t_1).n \\
 & \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} & \underbrace{(t_1, n \twoheadrightarrow S' \twoheadrightarrow 1)}_{t_2}.n \\
 & \triangleq & t_2\ n\ t_2
\end{array}
$$

An object with "self-extension"

Let $t_1 \triangleq add_n \to S \to (S, n \to S' \to 1)$

$$
\begin{array}{lll}
(t_1.add_n).n & \mapsto\!\!\!\!\to_{\rho\delta} & ((S \to (S, n \to S' \to 1))\ t_1).n \\
& \mapsto\!\!\!\!\to_{\rho\delta} & \underbrace{(t_1, n \to S' \to 1)}_{t_2}.n \\
& \triangleq & t_2\ n\ t_2 \\
& \mapsto\!\!\!\!\to_{\rho\delta} & ((add_n \to \ldots)\ n, (n \to S' \to 1)\ n)\ t_2
\end{array}
$$

An object with "self-extension"

Let $t_1 \triangleq add_n \to S \to (S, n \to S' \to 1)$

$$
\begin{array}{lll}
(t_1.add_n).n & \mapsto\!\!\!\!\to_{\rho\delta} & ((S \to (S, n \to S' \to 1))\ t_1).n \\
& \mapsto\!\!\!\!\to_{\rho\delta} & \underbrace{(t_1, n \to S' \to 1)}_{t_2}.n \\
& \triangleq & t_2\ n\ t_2 \\
& \mapsto\!\!\!\!\to_{\rho\delta} & ((add_n \to \ldots)\ n, (n \to S' \to 1)\ n)\ t_2 \\
& \mapsto\!\!\!\!\to_{\rho\delta} & \mathsf{stk}, (S' \to 1)\ t_2
\end{array}
$$

An object with "self-extension"

Let $t_1 \triangleq add_n \to S \to (S, n \to S' \to 1)$

$$
\begin{array}{rcl}
(t_1.add_n).n & \mapsto\!\!\!\!\to_{\rho\delta} & ((S \to (S, n \to S' \to 1))\ t_1).n \\
& \mapsto\!\!\!\!\to_{\rho\delta} & \underbrace{(t_1, n \to S' \to 1)}_{t_2}.n \\
& \triangleq & t_2\ n\ t_2 \\
& \mapsto\!\!\!\!\to_{\rho\delta} & ((add_n \to \ldots)\ n, (n \to S' \to 1)\ n)\ t_2 \\
& \mapsto\!\!\!\!\to_{\rho\delta} & \mathsf{stk}, (S' \to 1)\ t_2 \\
& =_{\mathbb{T}} & (S' \to 1)\ t_2 \quad (\mathbb{T}_{\varsigma\mathcal{O}bj})
\end{array}
$$

An object with "self-extension"

Let $t_1 \triangleq add_n \to S \to (S, n \to S' \to 1)$

$$
\begin{array}{rcl}
(t_1.add_n).n & \mapsto\!\!\!\!\to_{\rho\delta} & ((S \to (S, n \to S' \to 1))\ t_1).n \\
& \mapsto\!\!\!\!\to_{\rho\delta} & \underbrace{(t_1, n \to S' \to 1)}_{t_2}.n \\
& \triangleq & t_2\ n\ t_2 \\
& \mapsto\!\!\!\!\to_{\rho\delta} & ((add_n \to \ldots)\ n, (n \to S' \to 1)\ n)\ t_2 \\
& \mapsto\!\!\!\!\to_{\rho\delta} & \mathsf{stk}, (S' \to 1)\ t_2 \\
& =_{\mathbb{T}} & (S' \to 1)\ t_2 \quad (\mathbb{T}_{\varsigma\mathcal{O}bj}) \\
& \mapsto\!\!\!\!\to_{\rho\delta} & 1
\end{array}
$$

Inheritance in the ρ-calculus (Abadi & Cardelli encoding of classes-as-objects)

$$
\begin{aligned}
PClass \triangleq\ & new \to S \to (val \to S' \to (S.preval)\ S', \\
& \qquad get \to S' \to (S.preget)\ S', \\
& \qquad set \to S' \to (S.preset)\ S'), \\
& preval \to S \to S' \to v(1,1), \\
& preget \to S \to S' \to S'.val, \\
& preset \to S \to S' \to v(X,Y) \to (S'.val := S'' \to v(X,Y))
\end{aligned}
$$

Inheritance in the ρ-calculus (Abadi & Cardelli encoding of classes-as-objects)

$$
\begin{aligned}
PClass \quad \triangleq \quad & new \to S \to (val \to S' \to (S.preval)\ S', \\
& \qquad\qquad get \to S' \to (S.preget)\ S', \\
& \qquad\qquad set \to S' \to (S.preset)\ S'), \\
& preval \to S \to S' \to v(1,1), \\
& preget \to S \to S' \to S'.val, \\
& preset \to S \to S' \to v(X,Y) \to (S'.val := S'' \to v(X,Y))
\end{aligned}
$$

Then:

$$PClass.new \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} Point$$

The Para object: labels as first-class entities

$$Para \triangleq (ten \twoheadrightarrow S \twoheadrightarrow 10, par(X) \twoheadrightarrow S \twoheadrightarrow S.X)$$

This object has a method $par(X)$ which seeks for a method name that is assigned to the variable X and then sends this method to the object itself.

The Para object: labels as first-class entities

$$Para \triangleq (ten \twoheadrightarrow S \twoheadrightarrow 10, par(X) \twoheadrightarrow S \twoheadrightarrow S.X)$$

This object has a method $par(X)$ which seeks for a method name that is assigned to the variable X and then sends this method to the object itself.

$$Para.(par(ten)) \quad \triangleq \quad Para\ (par(ten))\ Para$$

The Para object: labels as first-class entities

$$Para \triangleq (ten \rightarrow S \rightarrow 10, par(X) \rightarrow S \rightarrow S.X)$$

This object has a method $par(X)$ which seeks for a method name that is assigned to the variable X and then sends this method to the object itself.

$$\begin{array}{lll} Para.(par(ten)) & \triangleq & Para\ (par(ten))\ Para \\ & \mapsto\!\!\!\!\!\rightarrow_{\beta\delta} & ((ten \rightarrow S \rightarrow 10)\ (par(ten)), \\ & & (par(X) \rightarrow S \rightarrow S.X)\ (par(ten)))\ Para \end{array}$$

The Para object: labels as first-class entities

$$Para \triangleq (ten \twoheadrightarrow S \twoheadrightarrow 10, par(X) \twoheadrightarrow S \twoheadrightarrow S.X)$$

This object has a method $par(X)$ which seeks for a method name that is assigned to the variable X and then sends this method to the object itself.

$$\begin{array}{lll} Para.(par(ten)) & \triangleq & Para\ (par(ten))\ Para \\ & \mapsto_{\rho\varpi\delta} & ((ten \twoheadrightarrow S \twoheadrightarrow 10)\ (par(ten)), \\ & & \ (par(X) \twoheadrightarrow S \twoheadrightarrow S.X)\ (par(ten)))\ Para \\ & \mapsto_{\rho\varpi\delta} & \mathsf{stk}, (S \twoheadrightarrow S.ten)\ Para \end{array}$$

The Para object: labels as first-class entities

$$Para \triangleq (ten \twoheadrightarrow S \twoheadrightarrow 10, par(X) \twoheadrightarrow S \twoheadrightarrow S.X)$$

This object has a method $par(X)$ which seeks for a method name that is assigned to the variable X and then sends this method to the object itself.

$$\begin{array}{lll} Para.(par(ten)) & \triangleq & Para\ (par(ten))\ Para \\ & \mapsto\!\!\!\!\!\rightarrow_{\rho\varpi\delta} & ((ten \twoheadrightarrow S \twoheadrightarrow 10)\ (par(ten)), \\ & & \quad (par(X) \twoheadrightarrow S \twoheadrightarrow S.X)\ (par(ten)))\ Para \\ & \mapsto\!\!\!\!\!\rightarrow_{\rho\varpi\delta} & \mathsf{stk}, (S \twoheadrightarrow S.ten)\ Para \\ & =_{\mathbb{T}} & (S \twoheadrightarrow S.ten)\ Para \quad (\mathbb{T}_{\varsigma\mathcal{O}bj}) \end{array}$$

The Para object: labels as first-class entities

$$Para \triangleq (ten \twoheadrightarrow S \twoheadrightarrow 10, par(X) \twoheadrightarrow S \twoheadrightarrow S.X)$$

This object has a method $par(X)$ which seeks for a method name that is assigned to the variable X and then sends this method to the object itself.

$$\begin{array}{lll} Para.(par(ten)) & \triangleq & Para\ (par(ten))\ Para \\ & \mapsto_{\rho\varsigma\delta} & ((ten \twoheadrightarrow S \twoheadrightarrow 10)\ (par(ten)), \\ & & \quad (par(X) \twoheadrightarrow S \twoheadrightarrow S.X)\ (par(ten)))\ Para \\ & \mapsto_{\rho\varsigma\delta} & \mathsf{stk}, (S \twoheadrightarrow S.ten)\ Para \\ & =_{\mathbb{T}} & (S \twoheadrightarrow S.ten)\ Para \quad (\mathbb{T}_{\varsigma\mathcal{O}bj}) \\ & \mapsto_{\rho\varsigma\delta} & Para.ten \end{array}$$

The Para object: labels as first-class entities

$$Para \triangleq (ten \twoheadrightarrow S \twoheadrightarrow 10, par(X) \twoheadrightarrow S \twoheadrightarrow S.X)$$

This object has a method $par(X)$ which seeks for a method name that is assigned to the variable X and then sends this method to the object itself.

$$\begin{array}{lll} Para.(par(ten)) & \triangleq & Para\ (par(ten))\ Para \\ & \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} & ((ten \twoheadrightarrow S \twoheadrightarrow 10)\ (par(ten)), \\ & & \quad (par(X) \twoheadrightarrow S \twoheadrightarrow S.X)\ (par(ten)))\ Para \\ & \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} & \mathsf{stk}, (S \twoheadrightarrow S.ten)\ Para \\ & =_{\mathbb{T}} & (S \twoheadrightarrow S.ten)\ Para \quad (\mathbb{T}_{\varsigma\mathcal{O}bj}) \\ & \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} & Para.ten \\ & \mapsto\!\!\!\!\!\rightarrow_{\rho\varsigma\delta} & 10 \end{array}$$

The object Daemon: methods as first-class entities

$$Daemon \triangleq set \to S \to X \to (X, set \to S' \to Y \to (Y, S'))$$

The object Daemon: methods as first-class entities

$$Daemon \triangleq set \twoheadrightarrow S \twoheadrightarrow X \twoheadrightarrow (X, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S'))$$

$$Daemon.set(x \twoheadrightarrow S \twoheadrightarrow 3)$$
$$\triangleq \quad Daemon\ set\ Daemon\ (x \twoheadrightarrow S \twoheadrightarrow 3)$$

The object Daemon: methods as first-class entities

$$Daemon \triangleq set \twoheadrightarrow S \twoheadrightarrow X \twoheadrightarrow (X, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S'))$$

$$\begin{aligned} Daemon.set(x \twoheadrightarrow S \twoheadrightarrow 3) & \\ \triangleq \quad & Daemon\ set\ Daemon\ (x \twoheadrightarrow S \twoheadrightarrow 3) \\ \mapsto_{\rho\varpi\delta} \quad & (S \twoheadrightarrow X \twoheadrightarrow (X, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S')))\ Daemon\ (x \twoheadrightarrow S \twoheadrightarrow 3) \end{aligned}$$

The object Daemon: methods as first-class entities

$$Daemon \triangleq set \twoheadrightarrow S \twoheadrightarrow X \twoheadrightarrow (X, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S'))$$

$$\begin{aligned}
&Daemon.set(x \twoheadrightarrow S \twoheadrightarrow 3) \\
&\quad \triangleq \quad Daemon\ set\ Daemon\ (x \twoheadrightarrow S \twoheadrightarrow 3) \\
&\quad \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} \quad (S \twoheadrightarrow X \twoheadrightarrow (X, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S')))\ Daemon\ (x \twoheadrightarrow S \twoheadrightarrow 3) \\
&\quad \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} \quad (X \twoheadrightarrow (X, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S')))\ (x \twoheadrightarrow S \twoheadrightarrow 3)
\end{aligned}$$

The object Daemon: methods as first-class entities

$$Daemon \triangleq set \twoheadrightarrow S \twoheadrightarrow X \twoheadrightarrow (X, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S'))$$

$$\begin{aligned}
&Daemon.set(x \twoheadrightarrow S \twoheadrightarrow 3) \\
&\quad \triangleq \quad Daemon\ set\ Daemon\ (x \twoheadrightarrow S \twoheadrightarrow 3) \\
&\quad \mapsto\!\!\!\!\!\rightarrow_{\rho\varpi\delta} \quad (S \twoheadrightarrow X \twoheadrightarrow (X, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S')))\ Daemon\ (x \twoheadrightarrow S \twoheadrightarrow 3) \\
&\quad \mapsto\!\!\!\!\!\rightarrow_{\rho\varpi\delta} \quad (X \twoheadrightarrow (X, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S')))\ (x \twoheadrightarrow S \twoheadrightarrow 3) \\
&\quad \mapsto\!\!\!\!\!\rightarrow_{\rho\varpi\delta} \quad \underbrace{x \twoheadrightarrow S \twoheadrightarrow 3, set \twoheadrightarrow S' \twoheadrightarrow Y \twoheadrightarrow (Y, S')}_{obj}
\end{aligned}$$

The object Daemon: methods as first-class entities

$$Daemon \triangleq set \to S \to X \to (X, set \to S' \to Y \to (Y, S'))$$

$$\begin{aligned}
&Daemon.set(x \to S \to 3) \\
&\quad \triangleq \quad Daemon\ set\ Daemon\ (x \to S \to 3) \\
&\quad \mapsto\!\!\!\!\!\rightarrow_{\rho\mathfrak{dB}} \quad (S \to X \to (X, set \to S' \to Y \to (Y, S')))\ Daemon\ (x \to S \to 3) \\
&\quad \mapsto\!\!\!\!\!\rightarrow_{\rho\mathfrak{dB}} \quad (X \to (X, set \to S' \to Y \to (Y, S')))\ (x \to S \to 3) \\
&\quad \mapsto\!\!\!\!\!\rightarrow_{\rho\mathfrak{dB}} \quad \underbrace{x \to S \to 3, set \to S' \to Y \to (Y, S')}_{obj}
\end{aligned}$$

$$obj.set(y \to S \to 4) \mapsto\!\!\!\!\!\rightarrow_{\rho\mathfrak{dB}} (y \to S \to 4, x \to S \to 3, set \to S' \to Y \to (Y, S'))$$

The Object Calculus $\varsigma\mathcal{O}bj$

Abstract syntax

$$a, b \quad ::= \quad X \;\mid\; [m_i = \varsigma(X)b_i]^{i\in I} \;\mid\; a.m \;\mid\; a.m := \varsigma(X)b$$

Small-step semantics

Let $a \triangleq [m_i = \varsigma(X)b_i]^{i\in I}$

$$(Select) \qquad a.m_j \rightsquigarrow \{X/a\}b_j \qquad (j \in I)$$

$$(Update) \qquad a.m_j := \varsigma(X)b \rightsquigarrow [m_i = \varsigma(X)b_i, m_j = \varsigma(X)b]^{i\in I\setminus\{j\}} \qquad (j \in I)$$

$$(Extend) \qquad a.m_j := \varsigma(X)b \rightsquigarrow [m_i = \varsigma(X)b_i, m_j = \varsigma(X)b]^{i\in I} \qquad (j \notin I)$$

Compiling $\varsigma\mathcal{O}bj$ in ρ-calculus

$$
\begin{array}{lcl}
[\![X]\!] & \triangleq & X \\
[\![a.m_j]\!] & \triangleq & [\![a]\!].m_j \\
[\![\ [m_i = \varsigma(X)b_i]^{i\in I}\ ]\!] & \triangleq & (m_i \twoheadrightarrow X \twoheadrightarrow [\![b_i]\!])^{i\in I} \\
[\![a.m := \varsigma(X)b]\!] & \triangleq & [\![a]\!].m := X \twoheadrightarrow [\![b]\!]
\end{array}
$$

Theorem:

If $a \rightsquigarrow_{\varsigma\mathcal{O}bj} b$, then $[\![a]\!] \longmapsto\!\!\!\!\!\rightarrow_{\rho\partial\mathcal{S}_{\mathbb{T}_{\varsigma\mathcal{O}bj}}} [\![b]\!]$.

Example with object update

Typed rewriting calculi

The new syntax - contexts, etc

τ	$::=$	$\iota \mid \tau \twoheadrightarrow \tau$	Types
Δ	$::=$	$\emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau$	Contexts
$\mathcal{P}$	$::=$	$\mathcal{X} \mid \mathsf{stk} \mid \mathcal{K}\ \overline{P}$ *(variables occur only once in any P)*	Patterns
$\mathcal{T}$	$::=$	$\mathcal{K} \mid \mathsf{stk} \mid \mathcal{X} \mid \mathcal{P} \twoheadrightarrow_\Delta \mathcal{T} \mid [\mathcal{P} \ll_\Delta \mathcal{T}]\mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid \mathcal{T}, \mathcal{T}$	Terms

The Type System

$$\frac{\mathcal{X}{:}\sigma \in \Gamma}{\Gamma \vdash \mathcal{X} : \sigma}\,(Start)$$

The Type System

$$\frac{\mathcal{X}{:}\sigma \in \Gamma}{\Gamma \vdash \mathcal{X} : \sigma}\,(Start)$$

$$\frac{\Gamma \vdash \mathcal{T}_1 : \sigma \quad \Gamma \vdash \mathcal{T}_2 : \sigma}{\Gamma \vdash \mathcal{T}_1, \mathcal{T}_2 : \sigma}\,(Struct)$$

The Type System

$$\frac{\mathcal{X}{:}\sigma \in \Gamma}{\Gamma \vdash \mathcal{X} : \sigma}\,(Start)$$

$$\frac{\Gamma \vdash \mathcal{T}_1 : \sigma \quad \Gamma \vdash \mathcal{T}_2 : \sigma}{\Gamma \vdash \mathcal{T}_1, \mathcal{T}_2 : \sigma}\,(Struct)$$

$$\frac{\Gamma \vdash \mathcal{T}_1 : \sigma \to \tau \quad \Gamma \vdash \mathcal{T}_2 : \sigma}{\Gamma \vdash \mathcal{T}_1\ \mathcal{T}_2 : \tau}\,(Appl)$$

The Type System

$$\frac{\mathcal{X}{:}\sigma \in \Gamma}{\Gamma \vdash \mathcal{X} : \sigma}\ (Start)$$

$$\frac{\Gamma \vdash \mathcal{T}_1 : \sigma \quad \Gamma \vdash \mathcal{T}_2 : \sigma}{\Gamma \vdash \mathcal{T}_1, \mathcal{T}_2 : \sigma}\ (Struct)$$

$$\frac{\Gamma \vdash \mathcal{T}_1 : \sigma \to \tau \quad \Gamma \vdash \mathcal{T}_2 : \sigma}{\Gamma \vdash \mathcal{T}_1\ \mathcal{T}_2 : \tau}\ (Appl)$$

$$\frac{\Gamma, \Delta \vdash \mathcal{T}_1 : \sigma \quad \Gamma, \Delta \vdash \mathcal{T}_2 : \tau}{\Gamma \vdash \mathcal{T}_1 \to_{\Delta} \mathcal{T}_2 : \sigma \to \tau}\ (Abs)$$

The Type System

$$\frac{\mathcal{X}{:}\sigma \in \Gamma}{\Gamma \vdash \mathcal{X} : \sigma}\,(Start)$$

$$\frac{\Gamma \vdash \mathcal{T}_1 : \sigma \quad \Gamma \vdash \mathcal{T}_2 : \sigma}{\Gamma \vdash \mathcal{T}_1, \mathcal{T}_2 : \sigma}\,(Struct)$$

$$\frac{\Gamma \vdash \mathcal{T}_1 : \sigma \to \tau \quad \Gamma \vdash \mathcal{T}_2 : \sigma}{\Gamma \vdash \mathcal{T}_1\ \mathcal{T}_2 : \tau}\,(Appl)$$

$$\frac{\Gamma, \Delta \vdash \mathcal{T}_1 : \sigma \quad \Gamma, \Delta \vdash \mathcal{T}_2 : \tau}{\Gamma \vdash \mathcal{T}_1 \to_{\Delta} \mathcal{T}_2 : \sigma \to \tau}\,(Abs)$$

$$\frac{\Gamma, \Delta \vdash \mathcal{T}_1 : \sigma \quad \Gamma \vdash \mathcal{T}_2 : \sigma \quad \Gamma, \Delta \vdash \mathcal{T}_3 : \tau}{\Gamma \vdash [\mathcal{T}_1 \ll_{\Delta} \mathcal{T}_2]\mathcal{T}_3 : \tau}\,(Match)$$

Typing properties

Well-typed matching:

If $\mathcal{S}ol(\mathcal{P} \ll \mathcal{T}) = \theta$, then $\forall X \in \mathcal{P}, \quad \Gamma \vdash X : \sigma \ \Rightarrow \Gamma \vdash X\theta : \sigma$.

Subject Reduction:

If $\Gamma \vdash \mathcal{T}_1 : \sigma$ and $\mathcal{T}_1 \mapsto\!\!\!\!\!\rightarrow_{\rho\alpha\beta\delta} \mathcal{T}_2$, then $\Gamma \vdash \mathcal{T}_2 : \sigma$.

Uniqueness:

If $\Gamma \vdash \mathcal{T} : \varphi$ and $\Gamma \vdash \mathcal{T} : \psi$, then $\varphi =_\alpha \psi$.

Decidability:

(typechecking) $\Gamma \vdash \mathcal{T} : \varphi\,?$
(type inference) $\Gamma \vdash \mathcal{T} : ?$

} are decidable.

Normalization failure

$f : (\alpha \to \alpha) \to \alpha$ and $\Gamma = X : \alpha \to \alpha$, $\quad \omega \triangleq f\ X \to X\ (f\ X)$

$$
\begin{array}{rcl}
\omega\ (f\ \omega) & \equiv & (f\ X \to X\ (f\ X))\ (f\ \omega) \\
& \mapsto_{\rho} & [f\ X \ll f\ \omega](X\ (f\ X)) \\
& \mapsto\!\!\!\!\mapsto_{\rho\partial} & \omega\ (f\ \omega) \\
& \mapsto_{\rho} & \dots
\end{array}
$$

Normalization failure (cont'd)

$f : (\alpha \to \alpha) \to \alpha$ and $\Gamma = X : \alpha \to \alpha$, $\quad \omega \triangleq f\ X \to X\ (f\ X)$

$$\dfrac{\dfrac{\Gamma \vdash f : (\alpha \to \alpha) \to \alpha \quad \Gamma \vdash X : \alpha \to \alpha}{(b)\ \ \Gamma \vdash f\ X : \alpha} \quad \dfrac{\Gamma \vdash X : \alpha \to \alpha \quad \dfrac{(b)}{\Gamma \vdash f\ X : \alpha}}{\Gamma \vdash X\ (f\ X) : \alpha}}{(a) \vdash \omega \equiv f\ X \to X\ (f\ X) : \alpha \to \alpha}$$

$$\dfrac{\dfrac{(a)}{\vdash \omega : \alpha \to \alpha} \quad \dfrac{\vdash f : (\alpha \to \alpha) \to \alpha \quad \dfrac{(a)}{\vdash \omega : \alpha \to \alpha}}{\vdash f\ \omega : \alpha}}{\vdash \omega\ (f\ \omega) : \alpha}$$

(Well-typed) Encoding of Rewriting in the ρ-calculus

- rewrite rules and their application,
  - ρ-abstractions and applications (Simple Encoding)

(Well-typed) Encoding of Rewriting in the ρ-calculus

- rewrite rules and their application,
  - ➥ ρ-abstractions and applications (Simple Encoding)

- an iteration operator that applies $repeatedly$ a set of rewrite rules,
  - ➥ $\omega\ (f\ \omega)$

(Well-typed) Encoding of Rewriting in the ρ-calculus

- rewrite rules and their application,
  - ➥ ρ-abstractions and applications (Simple Encoding)
- an iteration operator that applies $repeatedly$ a set of rewrite rules,
  - ➥ $\omega\ (f\ \omega)$
- a construction grouping together a set of rewrite rules,
  - ➥ structures and objects

(Well-typed) Encoding of Rewriting in the ρ-calculus

- rewrite rules and their application,
  - ➥ ρ-abstractions and applications (Simple Encoding)
- an iteration operator that applies $repeatedly$ a set of rewrite rules,
  - ➥ $\omega\ (f\ \omega)$
- a construction grouping together a set of rewrite rules,
  - ➥ structures and objects
- an operator testing if a set of rewrite rules is $applicable$ to a term.
  - ➥ the symbol stk

ρ-calculus and objects

- Object $=$ record with an explicit account of self, $i.e.$

$$[m_i = \varsigma(X_i)t_i]^{i\in I} \quad \triangleq \quad (m_i(X_i) \twoheadrightarrow t_i)^{i\in I}$$

ρ-calculus and objects

- Object $=$ record with an explicit account of self, $i.e.$

$$[m_i = \varsigma(X_i)t_i]^{i\in I} \quad \triangleq \quad (m_i(X_i) \twoheadrightarrow t_i)^{i\in I}$$

- Self-application $=$ the application of an object to the object itself, $i.e.$

$$t_1.t_2 \quad \triangleq \quad t_1\ t_2(t_1)$$

ρ-calculus and objects

- Object $=$ record with an explicit account of self, $i.e.$

$$[m_i = \varsigma(X_i)t_i]^{i\in I} \quad \triangleq \quad (m_i(X_i) \twoheadrightarrow t_i)^{i\in I}$$

- Self-application $=$ the application of an object to the object itself, $i.e.$

$$t_1.t_2 \quad \triangleq \quad t_1 \; t_2(t_1)$$

- Ex: $t \triangleq a(S) \twoheadrightarrow b$. Then: $t.a \triangleq t \; a(t) \mapsto_\rho [a(S) \ll a(t)]b \mapsto_\sigma b$

Typed objects

- An object has type:

$$\dfrac{\dfrac{S : lab \rightarrowtriangle \phi \vdash meth : (lab \rightarrowtriangle \phi) \rightarrowtriangle lab}{\vdash meth(S) : lab}\,(Appl) \qquad \vdash \mathcal{T}_{meth} : \phi}{\vdash meth(S) \rightarrowtriangle \mathcal{T}_{meth} : lab \rightarrowtriangle \phi}\,(Abst)$$

Typed objects

- An object has type:

$$\dfrac{\dfrac{S : lab \rightarrowtriangle \phi \vdash meth : (lab \rightarrowtriangle \phi) \rightarrowtriangle lab}{\vdash meth(S) : lab}\,(Appl) \qquad \vdash \mathcal{T}_{meth} : \phi}{\vdash meth(S) \rightarrowtriangle \mathcal{T}_{meth} : lab \rightarrowtriangle \phi}\,(Abst)$$

- $obj.meth \triangleq obj\ meth(obj)$ can be typed as follows:

$$\dfrac{\vdash obj : lab \rightarrowtriangle \phi \qquad \dfrac{\vdash meth : (lab \rightarrowtriangle \phi) \rightarrowtriangle lab \quad \vdash obj : lab \rightarrowtriangle \phi}{\vdash meth(obj) : lab}}{\vdash obj\ meth(obj) : \phi}$$

Detecting matching failures: the symbol stk

1. The relation $P \not\sqsubseteq A$ detects (some) definitive matching failures:

$$
\begin{array}{lcll}
f & \not\sqsubseteq & g & \\
f(\overline{A_n}) & \not\sqsubseteq & B & \text{if } (B \equiv g(\overline{B_m}), f \neq g\,) \ \vee \ (B \equiv f(\overline{B_n}), \exists i, A_i \not\sqsubseteq B_i) \\
P & \not\sqsubseteq & A & \text{if } A \equiv ([Q \ll A_1]A_2 \wedge Q \not\sqsubseteq A_1 \vee P \not\sqsubseteq A_2)
\end{array}
$$

Detecting matching failures: the symbol stk

1. The relation $P \not\sqsubseteq A$ detects (some) definitive matching failures:

$$
\begin{array}{lcll}
f & \not\sqsubseteq & g & \\
f(\overline{A_n}) & \not\sqsubseteq & B & \text{if } (B \equiv g(\overline{B_m}), f \neq g\,) \ \vee \ (B \equiv f(\overline{B_n}), \exists i, A_i \not\sqsubseteq B_i) \\
P & \not\sqsubseteq & A & \text{if } A \equiv ([Q \ll A_1]A_2 \wedge Q \not\sqsubseteq A_1 \vee P \not\sqsubseteq A_2)
\end{array}
$$

2. The relation $\rightarrow_{\mathsf{stk}}$ treats matching failures uniformly:

$$
\begin{array}{rcll}
[P \ll A]B & \rightarrow_{\mathsf{stk}} & \mathsf{stk} & \text{if } P \not\sqsubseteq A \\
\mathsf{stk}, A & \rightarrow_{\mathsf{stk}} & A & \\
A, \mathsf{stk} & \rightarrow_{\mathsf{stk}} & A & \\
\mathsf{stk}\ A & \rightarrow_{\mathsf{stk}} & \mathsf{stk} &
\end{array}
$$

Theory of Stuck $\mathbb{T}_{\mathsf{stk}}$ Encoding Rewriting

Encoding rewriting - Addition over Peano integers

$$plus \triangleq \left[\begin{array}{l} S \twoheadrightarrow add(0, y) \twoheadrightarrow y, \\ S \twoheadrightarrow add(suc(x), y) \twoheadrightarrow suc\big((S\ S)\ add(x, y)\big) \end{array} \right]$$

Encoding rewriting - Addition over Peano integers

$$plus \triangleq \left[\begin{array}{l} S \twoheadrightarrow add(0, y) \twoheadrightarrow y, \\ S \twoheadrightarrow add(suc(x), y) \twoheadrightarrow suc\big((S\ S)\ add(x, y)\big) \end{array}\right]$$

$$\begin{array}{ll} & (plus\ \ plus)\ \ add(N, M) \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\mathcal{S}} & [0 \ll N]M,\ [0 \ll N-1](M{+}1)\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ [suc\ x \ll 0]\ldots \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\mathcal{S}} & \mathsf{stk},\ \mathsf{stk}\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ \mathsf{stk}\ldots \\ \mapsto_{\mathsf{stk}} & M + N \end{array}$$

Encoding rewriting - Addition over Peano integers

$$plus \triangleq \left[\begin{array}{l} S \twoheadrightarrow add(0, y) \twoheadrightarrow y, \\ S \twoheadrightarrow add(suc(x), y) \twoheadrightarrow suc\big((S\ S)\ add(x, y)\big) \end{array} \right]$$

$$\begin{array}{ll} & (plus\ \ plus)\ \ add(N, M) \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} & [0 \ll N]M,\ [0 \ll N-1](M{+}1)\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ [suc\ x \ll 0]\ldots \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} & \mathsf{stk},\ \mathsf{stk}\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ \mathsf{stk}\ldots \\ \mapsto_{\mathsf{stk}} & M + N \end{array}$$

Fill in the blanks with your favorite rewrite system...

$$func \triangleq \left[\begin{array}{ll} S \twoheadrightarrow & \\ S \twoheadrightarrow & (S\ S) \end{array} \right]$$

Encoding rewriting - Addition over Peano integers

$$plus \triangleq \left[\begin{array}{l} S \twoheadrightarrow add(0, y) \twoheadrightarrow y, \\ S \twoheadrightarrow add(suc(x), y) \twoheadrightarrow suc\big((S\ S)\ add(x, y)\big) \end{array} \right]$$

$$\begin{array}{ll} & (plus\ \ plus)\ \ add(N, M) \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\mathcal{S}} & [0 \ll N]M,\ [0 \ll N-1](M{+}1)\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ [suc\ x \ll 0]\ldots \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\mathcal{S}} & \mathsf{stk},\ \mathsf{stk}\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ \mathsf{stk}\ldots \\ \mapsto_{\mathsf{stk}} & M + N \end{array}$$

Fill in the blanks with your favorite rewrite system...

$$func \triangleq \left[\begin{array}{l} S \twoheadrightarrow\ len([\,]) \twoheadrightarrow\ 0\ \ , \\ S \twoheadrightarrow\ len(Cons(x, l)) \twoheadrightarrow\ suc\big((S\ \ S)\ \ \ len(l)\big) \end{array} \right]$$

Encoding rewriting - Addition over Peano integers

$$plus \triangleq \left[\begin{array}{l} S \twoheadrightarrow add(0, y) \twoheadrightarrow y, \\ S \twoheadrightarrow add(suc(x), y) \twoheadrightarrow suc\big((S\ S)\ add(x, y)\big) \end{array} \right]$$

$$\begin{array}{ll} & (plus\ \ plus)\ \ add(N, M) \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial} & [0 \ll N]M,\ [0 \ll N-1](M{+}1)\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ [suc\ x \ll 0]\ldots \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial} & \mathsf{stk},\ \mathsf{stk}\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ \mathsf{stk}\ldots \\ \mapsto_{\mathsf{stk}} & M + N \end{array}$$

Fill in the blanks with your favorite rewrite system... provided it is convergent and ground reducible if you want completeness.

$$func \triangleq \left[\begin{array}{l} S \twoheadrightarrow\ len([\,]) \twoheadrightarrow\ 0\ \ , \\ S \twoheadrightarrow\ len(Cons(x, l)) \twoheadrightarrow\ suc\big((S\ \ S)\ \ \ len(l)\big) \end{array} \right]$$

Encoding rewriting - Addition over Peano integers

$$plus \triangleq \left[\begin{array}{l} S \twoheadrightarrow add(0, y) \twoheadrightarrow y, \\ S \twoheadrightarrow add(suc(x), y) \twoheadrightarrow suc\big((S\ S)\ add(x, y)\big) \end{array} \right]$$

$$\begin{array}{ll} & (plus\ \ plus)\ \ add(N, M) \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial} & [0 \ll N]M,\ [0 \ll N-1](M{+}1)\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ [suc\ x \ll 0]\ldots \\ \mapsto\!\!\!\!\!\rightarrow_{\rho\partial} & \mathsf{stk},\ \mathsf{stk}\ \cdots\ \underline{[0 \ll 0](M{+}N)},\ \mathsf{stk}\ldots \\ \mapsto_{\mathsf{stk}} & M + N \end{array}$$

Fill in the blanks with your favorite rewrite system... provided it is convergent and ground reducible if you want completeness.

$$func \triangleq \left[\begin{array}{l} S \twoheadrightarrow\ len([\,]) \twoheadrightarrow\ 0\ \ , \\ S \twoheadrightarrow\ len(Cons(x, l)) \twoheadrightarrow\ suc\big((S\ \ S)\ \ \ len(l)\big) \end{array} \right]$$

Encoding rewriting in the ρ-calculus

1. The following operator selects the first applicable rule of a set:

$$\mathsf{first}(\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_\mathsf{n}) \quad \stackrel{\triangle}{=} \quad X \twoheadrightarrow ((\mathsf{stk} \twoheadrightarrow A_n\ X, I)\ (\ldots\ (\mathsf{stk} \twoheadrightarrow A_2\ X, I)\ (A_1 X)))$$

$$\mathsf{first}(\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_\mathsf{n})\ \mathsf{B} \quad \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \quad A_{j+1}\ B \ \text{ if } \forall i \leq j,\ A_i\ B \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \mathsf{stk} \ \text{ and } A_{j+1}\ B \not\mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \mathsf{stk}$$

Encoding rewriting in the ρ-calculus

1. The following operator selects the first applicable rule of a set:

$$\mathsf{first}(\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_n) \quad \triangleq \quad X \twoheadrightarrow ((\mathsf{stk} \twoheadrightarrow A_n\ X, I)\ (\ldots\ (\mathsf{stk} \twoheadrightarrow A_2\ X, I)\ (A_1 X)))$$

$$\mathsf{first}(\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_n)\ \mathsf{B} \quad \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \quad A_{j+1}\ B \ \text{ if } \forall i \leq j,\ A_i\ B \mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \mathsf{stk} \ \text{ and } A_{j+1}\ B \not\mapsto\!\!\!\!\!\rightarrow_{\rho\partial\delta} \mathsf{stk}$$

2. The Term Rewrite System $\mathcal{R} = \{t_i \to s_i\}$ with signature $\{a_j\}$ is encoded by:

$$\mathcal{R} \ \rightsquigarrow \ (rec\ S) \twoheadrightarrow \mathsf{first} \left(\begin{array}{l} t_1 \twoheadrightarrow S\ (rec\ S)\ s_1, \\ \ldots, \\ a_1\ \overline{X} \twoheadrightarrow S\ (Rec\ S)\ (a_1\ \overline{S(rec\ S)\ X}), \\ \ldots, \end{array} \right),$$

$$(Rec\ S) \twoheadrightarrow \mathsf{first} \left(\begin{array}{l} t_1 \twoheadrightarrow S\ (rec\ S)\ s_1, \\ \ldots, \\ I \end{array} \right)$$

Example - A simple calculator

$$
\begin{array}{lcl}
calc & \triangleq & rec(S) \to \mathsf{first} \left(\begin{array}{l} add\ 0\ Y \to Y, \\ add\ (suc\ X)\ Y \to S.rec\ (suc\ (add\ X\ Y)), \\ mult\ 0\ Y \to 0, \\ mult\ (suc\ X)\ Y \to S.rec\ (add\ (mult\ X\ Y)\ X), \\ add\ X\ Y \to S.Rec\ (add\ (S.rec\ X)\ (S.rec\ Y)), \\ mult\ X\ Y \to S.Rec\ (mult\ (S.rec\ X)\ (S.rec\ Y)), \\ suc\ X \to S.Rec\ (suc\ (S.rec\ X)) \end{array} \right), \\
 & & Rec(S) \to \mathsf{first} \left(\begin{array}{l} add\ 0\ Y \to Y, \\ add\ (suc\ X)\ Y \to S.rec\ (suc\ (add\ X\ Y)), \\ mult\ 0\ Y \to 0, \\ mult\ (suc\ X)\ Y \to S.rec\ (add\ (mult\ X\ Y)\ X), \\ Y \to Y \end{array} \right)
\end{array}
$$

Computing $(3 + 5) \times 4$

➥ $calc.rec\ (mult\ (add\ \overline{3}\ \overline{5})\ \overline{4}) \mapsto\!\!\!\!\to_{\rho\alpha\delta} \overline{32}$.

Example - Computing the length of a list

$$
\begin{array}{lll}
length & \triangleq & rec\ S \twoheadrightarrow \mathsf{first} \left(\begin{array}{ll} len\ nil & \twoheadrightarrow 0\ , \\ len\ (cons\ X\ L) & \twoheadrightarrow S.rec\ (suc\ (len\ L))\ , \\ suc\ X & \twoheadrightarrow S.Rec\ suc\ (S.rec\ X) \end{array} \right) , \\
 & & Rec\ S \twoheadrightarrow \mathsf{first} \left(\begin{array}{ll} len\ nil & \twoheadrightarrow 0\ \ , \\ len\ (cons\ X\ L) & \twoheadrightarrow S.rec\ (suc\ (len\ L)), \\ Y \twoheadrightarrow Y & \end{array} \right)
\end{array}
$$

Logical inconsistency

- As is, the Curry-Howard isomorphism is not valid:

$$\frac{\Gamma, \Delta \vdash \mathcal{T}_1 : \sigma \quad \Gamma, \Delta \vdash \mathcal{T}_2 : \tau}{\Gamma \vdash \mathcal{T}_1 \rightarrowtriangle_{\Delta} \mathcal{T}_2 : \sigma \rightarrow \tau}\,(Abs) \qquad \frac{\Gamma, \Delta \vdash \sigma \quad \Gamma, \Delta \vdash \tau}{\Gamma \vdash \sigma \rightarrow \tau}\,(\rightarrow I)$$

Logical inconsistency

- As is, the Curry-Howard isomorphism is not valid:

$$\frac{\Gamma, \Delta \vdash \mathcal{T}_1 : \sigma \quad \Gamma, \Delta \vdash \mathcal{T}_2 : \tau}{\Gamma \vdash \mathcal{T}_1 \twoheadrightarrow_{\Delta} \mathcal{T}_2 : \sigma \twoheadrightarrow \tau}\,(Abs) \qquad \frac{\Gamma, \Delta \vdash \sigma \quad \Gamma, \Delta \vdash \tau}{\Gamma \vdash \sigma \twoheadrightarrow \tau}\,(\twoheadrightarrow I)$$

- Thus, for instance, $\dfrac{\vdash (h^{\perp \twoheadrightarrow \alpha \twoheadrightarrow \alpha}(X^{\perp}) \twoheadrightarrow X^{\perp}) : \alpha \twoheadrightarrow \alpha \twoheadrightarrow \perp}{\vdash (h^{\perp \twoheadrightarrow \alpha \twoheadrightarrow \alpha}(X^{\perp}) \twoheadrightarrow X^{\perp})\ (Y^{\alpha} \twoheadrightarrow Y^{\alpha}) : \perp}$

Logical inconsistency

- As is, the Curry-Howard isomorphism is not valid:

$$\frac{\Gamma, \Delta \vdash \mathcal{T}_1 : \sigma \quad \Gamma, \Delta \vdash \mathcal{T}_2 : \tau}{\Gamma \vdash \mathcal{T}_1 \twoheadrightarrow_{\Delta} \mathcal{T}_2 : \sigma \to \tau}\,(Abs) \qquad \frac{\Gamma, \Delta \vdash \sigma \quad \Gamma, \Delta \vdash \tau}{\Gamma \vdash \sigma \to \tau}\,(\to I)$$

- Thus, for instance, $\dfrac{\vdash (h^{\perp \to \alpha \to \alpha}(X^{\perp}) \to X^{\perp}) : \alpha \to \alpha \to \perp}{\vdash (h^{\perp \to \alpha \to \alpha}(X^{\perp}) \to X^{\perp})\ (Y^{\alpha} \to Y^{\alpha}) : \perp}$

- How to fix it ?

$$\frac{\Gamma, X_i : \varphi_i \vdash B : \psi}{\Gamma \vdash A \to B : (\bigwedge \varphi_i) \to \psi}\,(Abs) \quad , \quad FV(A) = \{X_i^{\varphi_i}\}$$

Dependent type discipline

$$\frac{\Gamma, \Delta \vdash \mathcal{T}_1 : \sigma \quad \Gamma, \Delta \vdash \mathcal{T}_2 : \tau}{\Gamma \vdash \mathcal{T}_1 : \Delta \rightarrowtriangle \mathcal{T}_2 : \sigma \rightarrowtriangle \tau} (Abs)$$

$$\frac{\Gamma \vdash \mathcal{T}_1 : \sigma \rightarrowtriangle \tau \quad \Gamma \vdash \mathcal{T}_2 : \sigma}{\Gamma \vdash \mathcal{T}_1\ \mathcal{T}_2 : \tau} (Appl)$$

$$\frac{\Gamma, \Delta \vdash \mathcal{T}_2 : \tau}{\Gamma \vdash (\mathcal{T}_1 : \Delta) \rightarrowtriangle \mathcal{T}_2 : (\mathcal{T}_1 : \Delta) \rightarrowtriangle \tau} (Abs)$$

$$\frac{\Gamma \vdash \mathcal{T}_1 : (\mathcal{T}_{11} : \Delta) \rightarrowtriangle \tau \quad \Gamma \vdash \mathcal{T}_2 : \sigma \quad \Gamma, \Delta \vdash \mathcal{T}_{11} : \sigma}{\Gamma \vdash \mathcal{T}_1\ \mathcal{T}_2 : [\mathcal{T}_{11} \ll_\Delta \mathcal{T}_2]\tau} (Appl)$$

Pure Pattern Type Systems

POPL-03

P^2TS

Motivations and Contributions

Small "λ-Digression"

From Lambda-calculus to Rewriting-calculus

- Lambda-calculus builds upon lambda abstraction:

$$(\lambda \mathbf{X}.\mathbf{X})\ \mathbf{3}$$

From Lambda-calculus to Rewriting-calculus

- Lambda-calculus builds upon lambda abstraction:

$$\mathbf{3}$$

From Lambda-calculus to Rewriting-calculus

- Lambda-calculus builds upon lambda abstraction:

$$\mathbf{3}$$

- Lambda-calculus with patterns builds upon pattern abstraction:

From Lambda-calculus to Rewriting-calculus

- Lambda-calculus builds upon lambda abstraction:

$$\mathbf{3}$$

- Lambda-calculus with patterns builds upon pattern abstraction:

$$(\lambda\langle \mathbf{f}(\mathbf{X}), \mathbf{g}(\mathbf{Y})\rangle.\langle \mathbf{X}, \mathbf{Y}\rangle)\langle \mathbf{3}, \mathbf{4}\rangle$$

From Lambda-calculus to Rewriting-calculus

- Lambda-calculus builds upon lambda abstraction:

$$\mathbf{3}$$

- Lambda-calculus with patterns builds upon pattern abstraction:

$$(\lambda \underbrace{\langle \mathbf{sqr(X)}, \mathbf{trl(X)} \rangle}_{\text{pattern}} . \mathbf{headof(X)}) \ \underbrace{\langle \mathbf{sqr(wood)}, \mathbf{trl(wood)} \rangle}_{\text{argument}}$$

From Lambda-calculus to Rewriting-calculus

- Lambda-calculus builds upon lambda abstraction:

$$\mathbf{3}$$

- Lambda-calculus with patterns builds upon pattern abstraction:

$$(\lambda \underbrace{\langle \mathbf{sqr(X)}, \mathbf{trl(X)} \rangle}_{\text{pattern}} . \mathbf{headof(X)}) \ \underbrace{\langle \mathbf{sqr(wood)}, \mathbf{trl(wood)} \rangle}_{\text{argument}}$$

- Rewriting-calculus builds upon generalised abstraction:

$$(\lambda \underbrace{(\lambda \mathbf{P}.\mathbf{Q})}_{\text{pattern}} . \mathbf{M}) \ \underbrace{\mathbf{N}}_{\text{arg}}$$

From Lambda-calculus to Rewriting-calculus

- Lambda-calculus builds upon lambda abstraction:

$$\mathbf{3}$$

- Lambda-calculus with patterns builds upon pattern abstraction:

$$(\lambda \underbrace{\langle \mathbf{sqr(X), trl(X)} \rangle}_{\text{pattern}} . \mathbf{headof(X)}) \ \underbrace{\langle \mathbf{sqr(wood), trl(wood)} \rangle}_{\text{argument}}$$

- Rewriting-calculus builds upon generalised abstraction:

$$(\lambda \underbrace{(\lambda \mathbf{X.Y})}_{\text{pattern}} . \mathbf{Y\ 3}) \underbrace{(\lambda \mathbf{Z.Z})}_{\text{arg}}$$

PATTERNS

We Want More Patterns!

Pattern Power!

The Uncle Pat

MATCHING

We Want More Matching!

Matching Power!

The Lady Match

P^{2}TS: Notational Convention

- Capital letters, like $A, B, C, \ldots$ range over general terms (metavariables)
- Capital letters, like $X, Y, Z, \ldots$ range over variables
- Small letter, like $a, b, \ldots, f, \ldots$ and strings like $car, cons, nil, int, \ldots$ range over constants
- The application of a constant function, say f, to a term A will be usually denoted by $f(A)$, following the algebraic "folklore"
- We can "curryfy" in order to denote a function taking multiple arguments, $e.g.$ $f(A_1 \cdots A_n) \stackrel{\triangle}{=} f\ A_1\ \cdots A_n$
- Fact: Mixing term rewriting and lambda calculus is a longstanding "shambles": many proposals, many solutions, many problems, ...

P^2TS: **Tricky !**

- Quiz: . . . the below term can have free variables ?

$$\lambda \underbrace{cons(T\ X\ nil(T))}_{\text{pattern}}.cons(T\ X\ cons(T\ X\ nil(T)))$$

P^2TS: Tricky !

- Quiz: . . . the below term can have free variables ?

$$\lambda \underbrace{cons(\mathbf{T}\ X\ nil(\mathbf{T}))}_{\text{pattern}}.cons(\mathbf{T}\ X\ cons(\mathbf{T}\ X\ nil(\mathbf{T})))$$

- yes, and we can even abstract over the variable T

$$\lambda T.\lambda cons(T\ X\ nil(T)).cons(T\ X\ cons(T\ X\ nil(T)))$$

P^{2}T S: **Tricky !**

- ... we will explain why we cannot reduce

$$(\lambda \underbrace{cons(T\ X\ nil(T))}_{\text{pattern}}.cons(T\ X\ cons(T\ X\ nil(T))))$$

$$\underbrace{cons(int\ 3\ nil(int))}_{\text{argument}}$$

P^2TS: Tricky !

- ... we will explain why we cannot reduce

$$(\lambda \underbrace{cons(T\ X\ nil(T))}_{\text{pattern}}.cons(T\ X\ cons(T\ X\ nil(T))))\ \underbrace{cons(int\ 3\ nil(int))}_{\text{argument}}$$

- ... but we can reduce

$$\lambda \underbrace{T}_{\text{patt1}}.\lambda \underbrace{cons(T\ X\ nil(T))}_{\text{patt2}}.cons(T\ X\ cons(T\ X\ nil(T)))\ \underbrace{int}_{\text{arg1}}\ \underbrace{cons(int\ 3\ nil(int))}_{\text{arg2}}$$

TYPES

We Need to Plug Types!

Thanks to TAL's Group (Cornell)

Typed Rho

- Our goal is to use the Rewriting-calculus as a foundation for proof assistants based on Curry-Howard isomorphism $\grave{a}\ la$ Coq, Twelf, Lego, . . .

Typed Rho

- Our goal is to use the Rewriting-calculus as a foundation for proof assistants based on Curry-Howard isomorphism $\grave{a}\ la$ Coq, Twelf, Lego, . . .
- As an intermediate goal, we develop a dependent type theory for the Rewriting-calculus

Typed Rho

- Our goal is to use the Rewriting-calculus as a foundation for proof assistants based on Curry-Howard isomorphism $\grave{a}\ la$ Coq, Twelf, Lego, . . .
- As an intermediate goal, we develop a dependent type theory for the Rewriting-calculus
- We do so by extending the framework of Pure Type Systems (PTS)

Typed Rho

- Our goal is to use the Rewriting-calculus as a foundation for proof assistants based on Curry-Howard isomorphism $\grave{a}\ la$ Coq, Twelf, Lego, . . .
- As an intermediate goal, we develop a dependent type theory for the Rewriting-calculus
- We do so by extending the framework of Pure Type Systems (PTS)
- We develop the basic theory of the resulting framework which we call

Pure Pattern Type Systems

(P^2TS)

Typed Rho

- Our goal is to use the Rewriting-calculus as a foundation for proof assistants based on Curry-Howard isomorphism $\grave{a}\ la$ Coq, Twelf, Lego, . . .
- As an intermediate goal, we develop a dependent type theory for the Rewriting-calculus
- We do so by extending the framework of Pure Type Systems (PTS)
- We develop the basic theory of the resulting framework which we call

Pure Pattern Type Systems

(P^2TS)

- This is not as straightforward as one may imagine

Typed Rho

- Our goal is to use the Rewriting-calculus as a foundation for proof assistants based on Curry-Howard isomorphism $\grave{a}\ la$ Coq, Twelf, Lego, . . .
- As an intermediate goal, we develop a dependent type theory for the Rewriting-calculus
- We do so by extending the framework of Pure Type Systems (PTS)
- We develop the basic theory of the resulting framework which we call

Pure Pattern Type Systems

(P^2TS)

- This is not as straightforward as one may imagine :-)

P^2TS

Free Unrestricted Patterns

Plætora of Serious Problems

P^{2}TS: **Some Problems**

- Confluence can fails for bad patterns

$$(\lambda \underbrace{(\mathbf{X}\ \mathbf{Y})}_{\text{bad!}}.\mathbf{X})((\lambda \mathbf{Z}.\mathbf{Z})\ \mathbf{a}) \mapsto\!\!\!\!\rightarrow$$

P^{2}TS: **Some Problems**

- Confluence can fails for bad patterns

$$(\lambda \underbrace{(\mathbf{X}\ \mathbf{Y})}_{\text{bad!}}.\mathbf{X})((\lambda \mathbf{Z}.\mathbf{Z})\ \mathbf{a}) \twoheadrightarrow (\lambda \mathbf{Z}.\mathbf{Z})$$

P^2TS: Some Problems

- Confluence can fails for bad patterns

$$(\lambda \underbrace{(\mathbf{X}\ \mathbf{Y})}_{\text{bad!}}.\mathbf{X})((\lambda \mathbf{Z}.\mathbf{Z})\ \mathbf{a}) \twoheadrightarrow (\lambda(\mathbf{X}\ \mathbf{Y}).\mathbf{X})\ \mathbf{a}$$

P^2TS: Some Problems

- Confluence can fails for bad patterns

$$(\lambda \underbrace{(\mathbf{X}\ \mathbf{Y})}_{\text{bad!}}.\mathbf{X})((\lambda \mathbf{Z}.\mathbf{Z})\ \mathbf{a}) \twoheadrightarrow (\lambda(\mathbf{X}\ \mathbf{Y}).\mathbf{X})\ \mathbf{a}$$

- Subject Reduction can fails for bad patterns

$$\vdash (\lambda \underbrace{(\mathbf{X}_1^{\sigma_1 \to \tau}\ \mathbf{Y}_1^{\sigma_1})}_{\text{bad patt}}.\ \mathbf{Z}^{\sigma_1 \to \tau}\ \mathbf{Y}_1^{\sigma_1})\ \underbrace{(\mathbf{X}_2^{\sigma_2 \to \tau}\ \mathbf{Y}_2^{\sigma_2})}_{\text{badarg}} : \tau$$

P^{2}T S: **Some Problems**

- Confluence can fails for bad patterns

$$(\lambda \underbrace{(\mathbf{X}\ \mathbf{Y})}_{\text{bad!}}.\mathbf{X})((\lambda \mathbf{Z}.\mathbf{Z})\ \mathbf{a}) \twoheadrightarrow (\lambda(\mathbf{X}\ \mathbf{Y}).\mathbf{X})\ \mathbf{a}$$

- Subject Reduction can fails for bad patterns

$$\not\vdash \mathbf{Z}^{\sigma_1 \to \tau}\ \mathbf{Y}_{\mathbf{2}}^{\sigma_2} : \tau$$

P^{2}TS: **Some Problems**

- Confluence can fails for bad patterns

$$(\lambda \underbrace{(\mathbf{X}\ \mathbf{Y})}_{\text{bad!}}.\mathbf{X})((\lambda \mathbf{Z}.\mathbf{Z})\ \mathbf{a}) \twoheadrightarrow (\lambda(\mathbf{X}\ \mathbf{Y}).\mathbf{X})\ \mathbf{a}$$

- Subject Reduction can fails for bad patterns

$$\not\vdash \mathbf{Z}^{\sigma_1 \to \tau}\ \mathbf{Y}_{\mathbf{2}}^{\sigma_2} : \tau$$

- Shapes of good patterns must be synchronized with a sound static type system!

A Good Recipes for ... Good Patterns

- Good Patterns are in Normal form (no redexes), $i.e.$ the bad pattern

$$(\lambda \mathbf{P}.\mathbf{A})\ \mathbf{B}$$

A Good Recipes for ... Good Patterns

- Good Patterns are in Normal form (no redexes), $i.e.$ the bad pattern

$$(\lambda \mathbf{P}.\mathbf{A})\ \mathbf{B}$$

- Good Patterns are not occurrence of "active" variables, $i.e.$ the bad pattern

$$(\mathbf{X}\ \mathbf{A})$$

A Good Recipes for ... Good Patterns

- Good Patterns are in Normal form (no redexes), $i.e.$ the bad pattern

$$(\lambda \mathbf{P}.\mathbf{A})\ \mathbf{B}$$

- Good Patterns are not occurrence of "active" variables, $i.e.$ the bad pattern

$$(\mathbf{X}\ \mathbf{A})$$

- Good Patterns are linear, $i.e.$ variables occurs only once, $i.e.$ the bad pattern

$$\mathbf{f}(\mathbf{X}, \mathbf{X})$$

A Good Recipes for ... Good Patterns

- Good Patterns are in Normal form (no redexes), $i.e.$ the bad pattern

$$(\lambda \mathbf{P}.\mathbf{A})\ \mathbf{B}$$

- Good Patterns are not occurrence of "active" variables, $i.e.$ the bad pattern

$$(\mathbf{X}\ \mathbf{A})$$

- Good Patterns are linear, $i.e.$ variables occurs only once, $i.e.$ the bad pattern

$$\mathbf{f}(\mathbf{X}, \mathbf{X})$$

- All those recipes can be formalized, enforced by the syntax, checked at run time or statically, or by any reasonable mathematical technique....

The main contribution of this (ongoing) work are ...

- to provide adequate notions of patterns, substitutions and syntactic matching in a typed setting.
  We introduce delayed matching constraint, and the possibility for patterns in abstractions to evolve (by reduction or substitution) during execution

The main contribution of this (ongoing) work are ...

- to provide adequate notions of patterns, substitutions and syntactic matching in a typed setting.
  We introduce delayed matching constraint, and the possibility for patterns in abstractions to evolve (by reduction or substitution) during execution
- to propose an extension of PTSs supporting abstraction over patterns, and enjoying
  - ⋆ confluence
  - ⋆ subject reduction
  - ⋆ conservativity over PTSs
  - ⋆ consistency for normalizing P^2TSs

The main contribution of this (ongoing) work are ...

- to provide adequate notions of patterns, substitutions and syntactic matching in a typed setting.
  We introduce delayed matching constraint, and the possibility for patterns in abstractions to evolve (by reduction or substitution) during execution
- to propose an extension of PTSs supporting abstraction over patterns, and enjoying
  - ⋆ confluence
  - ⋆ subject reduction
  - ⋆ conservativity over PTSs
  - ⋆ consistency for normalizing P^2TSs
- Strong normalization for all P^2TS is an open problem . . . but it is ok for simple P^2TS-types (see Benjamin Wack SN-paper) and it "seems" ok for the simplest dependent-type discipline ($\rightsquigarrow$ *Pattern Logical Framework*)

P^2TS

The Syntax

$\mathsf{P}^2\mathsf{T}\,\mathsf{S}$

its time to be uniform!

$$\lambda A.B \sim A \to B$$

The Typed Syntax

$$\Gamma \ ::= \ \emptyset \mid \Gamma, X{:}A \mid \Gamma, f{:}A$$

$$A \ ::= \ X \mid f \mid P \twoheadrightarrow_{\Delta} B \mid A \ A \mid [P \ll_{\Delta} B]C \mid A, B$$

$$\Pi P{:}\Delta.B$$

The Typed Syntax

$$\Gamma \ ::= \ \emptyset \mid \Gamma, X{:}A \mid \Gamma, f{:}A$$

$$A \ ::= \ X \mid f \mid P \twoheadrightarrow_{\Delta} B \mid A \ A \mid [P \ll_{\Delta} B]C \mid A, B$$

$$\Pi P{:}\Delta.B$$

1. Term $\mathbf{A \twoheadrightarrow_{\Delta} B}$ is an abstraction (resp. product abstraction $i.e.$ $\Pi A{:}\Delta.B$)

The Typed Syntax

$$\Gamma \ ::= \ \emptyset \mid \Gamma, X{:}A \mid \Gamma, f{:}A$$

$$A \ ::= \ X \mid f \mid P \twoheadrightarrow_{\Delta} B \mid A \ A \mid [P \ll_{\Delta} B]C \mid A, B$$

$$\Pi P{:}\Delta.B$$

1. Term $\mathbf{A \twoheadrightarrow_{\Delta} B}$ is an abstraction (resp. product abstraction $i.e.$ $\Pi A{:}\Delta.B$)
2. Term $\mathbf{[A \ll_{\Delta} B]C}$ is a delayed matching constraint

The Typed Syntax

$$\Gamma \ ::= \ \emptyset \mid \Gamma, X{:}A \mid \Gamma, f{:}A$$

$$A \ ::= \ X \mid f \mid P \twoheadrightarrow_\Delta B \mid A \ A \mid [P \ll_\Delta B]C \mid A, B$$

$$\Pi P{:}\Delta.B$$

1. Term $\mathbf{A \twoheadrightarrow_\Delta B}$ is an abstraction (resp. product abstraction $i.e.$ $\Pi A{:}\Delta.B$)
2. Term $\mathbf{[A \ll_\Delta B]C}$ is a delayed matching constraint
3. Term of the form $\mathbf{A, B}$ is called a structure

The Typed Syntax

$$\Gamma \ ::= \ \emptyset \mid \Gamma, X{:}A \mid \Gamma, f{:}A$$

$$A \ ::= \ X \mid f \mid P \twoheadrightarrow_\Delta B \mid A \ A \mid [P \ll_\Delta B]C \mid A, B$$

$$\Pi P{:}\Delta.B$$

1. Term $\mathbf{A \twoheadrightarrow_\Delta B}$ is an abstraction (resp. product abstraction $i.e.$ $\Pi A{:}\Delta.B$)
2. Term $\mathbf{[A \ll_\Delta B]C}$ is a delayed matching constraint
3. Term of the form $\mathbf{A, B}$ is called a structure
4. Term of the form $\mathbf{\Pi P{:}\Delta.B}$ is called a product type

Untyped Rho *vs.* Typed P^{2}T S

Rho

$$\mathcal{T} \ ::= \ \mathcal{V} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid [\mathcal{P} \ll \mathcal{T}].\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

P^{2}T S

$$\mathcal{C} \ ::= \ \emptyset \mid \mathcal{C}, \mathcal{V}{:}\mathcal{T} \mid \mathcal{C}, \mathcal{K}{:}\mathcal{T} \qquad \checkmark \in \{\lambda\ \Pi\}$$

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{K} \mid \checkmark\mathcal{P}{:}\mathcal{C}.\mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid [\mathcal{P} \ll_{\mathcal{C}} \mathcal{T}].\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

Untyped Rho *vs.* Typed P^{2}T S

Rho

$$\mathcal{T} \ ::= \ \mathcal{V} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid [\mathcal{P} \ll \mathcal{T}].\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

P^{2}T S

$$\mathcal{C} \ ::= \ \emptyset \mid \mathcal{C}, \mathcal{V}{:}\mathcal{T} \mid \mathcal{C}, \mathcal{K}{:}\mathcal{T} \qquad \checkmark \in \{\lambda\ \Pi\}$$

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{K} \mid \checkmark\mathcal{P}{:}\mathcal{C}.\mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid [\mathcal{P} \ll_{\mathcal{C}} \mathcal{T}].\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

Untyped Rho *vs.* Typed P^{2}T S

Rho

$$\mathcal{T} \ ::= \ \mathcal{V} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid [\mathcal{P} \ll \mathcal{T}].\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

P^{2}T S

$$\mathcal{C} \ ::= \ \emptyset \mid \mathcal{C}, \mathcal{V}{:}\mathcal{T} \mid \mathcal{C}, \mathcal{K}{:}\mathcal{T} \qquad \checkmark \in \{\lambda\ \Pi\}$$

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{K} \mid \checkmark\mathcal{P}{:}\mathcal{C}.\mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid [\mathcal{P} \ll_{\mathcal{C}} \mathcal{T}].\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

Untyped Rho *vs.* Typed P^{2}T S

Rho

$$\mathcal{T} \ ::= \ \mathcal{V} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid [\mathcal{P} \ll \mathcal{T}].\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

P^{2}T S

$$\mathcal{C} \ ::= \ \emptyset \mid \mathcal{C}, \mathcal{V}{:}\mathcal{T} \mid \mathcal{C}, \mathcal{K}{:}\mathcal{T} \qquad \checkmark \in \{\lambda\ \Pi\}$$

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{K} \mid \checkmark\mathcal{P}{:}\mathcal{C}.\mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid [\mathcal{P} \ll_{\mathcal{C}} \mathcal{T}].\mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

P^2TS

Small-step Semantics

Reduction@glance

$$(P \twoheadrightarrow_{\Delta} B)C$$

Reduction@glance

$$(P \twoheadrightarrow_\Delta B)C \;\mapsto_\rho\; [P \ll_\Delta C].B$$

Reduction@glance

$$(P \twoheadrightarrow_\Delta B)C \mapsto_\rho [P \ll_\Delta C].B \mapsto_\sigma B\sigma$$

Reduction@glance

$$(P \twoheadrightarrow_\Delta B)C \mapsto_\rho [P \ll_\Delta C].B \mapsto_\sigma B\sigma \quad \text{if } \exists\sigma.A\sigma \equiv C$$

Reduction@glance

$$(P \twoheadrightarrow_\Delta B)C \mapsto_\rho [P \ll_\Delta C].B \mapsto_\sigma B\sigma \quad \text{if } \exists\sigma.A\sigma \equiv C$$

$$(P \twoheadrightarrow_\Delta B)C$$

Reduction@glance

$$(P \twoheadrightarrow_\Delta B)C \mapsto_\rho [P \ll_\Delta C].B$$
$$\mapsto_\sigma B\sigma$$
$$\text{if } \exists\sigma.A\sigma \equiv C$$

$$(P \twoheadrightarrow_\Delta B)C \mapsto_\rho [P \ll_\Delta C].B$$

Reduction@glance

$$(P \twoheadrightarrow_\Delta B)C \mapsto_\rho [P \ll_\Delta C].B \mapsto_\sigma B\sigma \quad \text{if } \exists\sigma.A\sigma \equiv C$$

$$(P \twoheadrightarrow_\Delta B)C \mapsto_\rho [P \ll_\Delta C].B$$

STOP!

Reduction@glance

$$(P \twoheadrightarrow_\Delta B)C \mapsto_\rho [P \ll_\Delta C].B \mapsto_\sigma B\sigma \quad \text{if } \exists\sigma.A\sigma \equiv C$$

$$(P \twoheadrightarrow_\Delta B)C \mapsto_\rho [P \ll_\Delta C].B \quad \text{STOP!} \quad \text{if } \nexists\sigma.P\sigma \equiv C$$

The Small-step Reduction Semantics

$$(\rho) \quad (P \twoheadrightarrow_\Delta B)\ C \quad \mapsto_\rho \quad [P \ll_\Delta C].B$$

$$(\sigma) \quad [P \ll_\Delta C].B \quad \mapsto_\sigma \quad B\sigma_{(P \prec\!\!\prec^{\Delta}_{\emptyset} C)}$$

$$(\delta) \quad (A, B)\ C \quad \mapsto_\delta \quad A\ C, B\ C$$

The Small-step Reduction Semantics

$$(\rho)\quad (P \twoheadrightarrow_\Delta B)\ C \quad \mapsto_\rho \quad [P \ll_\Delta C].B$$

$$(\sigma)\quad [P \ll_\Delta C].B \quad \mapsto_\sigma \quad B\sigma_{(P \overset{\Delta}{\ll}_\emptyset C)}$$

$$(\delta)\quad (A, B)\,C \quad \mapsto_\delta \quad A\ C, B\ C$$

- (ρ): applying $\mathbf{P \twoheadrightarrow_\Delta B}$ to C reduces to the delayed matching constraint $\mathbf{[P \ll_\Delta C].B}$

The Small-step Reduction Semantics

$$(\rho) \quad (P \twoheadrightarrow_{\Delta} B)\ C \quad \mapsto_{\rho} \quad [P \ll_{\Delta} C].B$$

$$(\sigma) \quad [P \ll_{\Delta} C].B \quad \mapsto_{\sigma} \quad B\sigma_{(P \prec\!\!\prec^{\Delta}_{\emptyset} C)}$$

$$(\delta) \qquad (A, B)\,C \quad \mapsto_{\delta} \quad A\ C, B\ C$$

- (ρ): applying $\mathbf{P} \twoheadrightarrow_{\mathbf{\Delta}} \mathbf{B}$ to C reduces to the delayed matching constraint $[\mathbf{P} \ll_{\mathbf{\Delta}} \mathbf{C}].\mathbf{B}$

- (σ): run successfully $\mathcal{A}\mathbf{lg}(\mathbf{P} \prec\!\!\prec^{\mathbf{\Delta}}_{\emptyset} \mathbf{C})$, and applying the result σ to the term $\mathbf{B}$

The Small-step Reduction Semantics

$$(\rho) \quad (P \twoheadrightarrow_\Delta B)\ C \quad \mapsto_\rho \quad [P \ll_\Delta C].B$$

$$(\sigma) \quad [P \ll_\Delta C].B \quad \mapsto_\sigma \quad B\sigma_{(P \prec\!\!\prec^\Delta_\emptyset C)}$$

$$(\delta) \quad (A, B)\,C \quad \mapsto_\delta \quad A\ C, B\ C$$

- (ρ): applying $\mathbf{P} \twoheadrightarrow_{\mathbf{\Delta}} \mathbf{B}$ to C reduces to the delayed matching constraint $[\mathbf{P} \ll_{\mathbf{\Delta}} \mathbf{C}].\mathbf{B}$
- (σ): run successfully $\mathcal{A}\mathbf{lg}(\mathbf{P} \prec\!\!\prec^{\mathbf{\Delta}}_\emptyset \mathbf{C})$, and applying the result σ to the term $\mathbf{B}$
- (δ): deals with the distributivity of the application on the structures built with the "," constructor

One-Many-Congruence-As-Usual

P^2TS

Galleria & Glance

Galleria I: The Pattern Abstraction $\mathbf{A} \twoheadrightarrow_{\Delta} \mathbf{B}$

- Generalisation of the λ-abstraction in PTSs. The rationale is:

$$\mathbf{X} \twoheadrightarrow_{\underbrace{(\mathbf{X}:\sigma)}_{\Delta}} \mathbf{A} \sim \lambda \mathbf{X}:\sigma.\mathbf{A}$$

Galleria I: The Pattern Abstraction $\mathbf{A} \twoheadrightarrow_{\Delta} \mathbf{B}$

- Generalisation of the λ-abstraction in PTSs. The rationale is:

$$\mathbf{f}(\mathbf{X}\,\mathbf{Y}) \twoheadrightarrow_{\underbrace{(\mathbf{X}{:}\sigma,\,\mathbf{Y}{:}\tau)}_{\Delta}} \mathbf{A} \sim \lambda\mathbf{f}(\mathbf{X}\,\mathbf{Y}){:}(\mathbf{X}{:}\sigma,\,\mathbf{Y}{:}\tau).\mathbf{A}$$

Galleria I: The Pattern Abstraction $\mathbf{A} \twoheadrightarrow_{\Delta} \mathbf{B}$

- Generalisation of the λ-abstraction in PTSs. The rationale is:

$$\mathbf{f}(\mathbf{X}\,\mathbf{Y}) \twoheadrightarrow_{\underbrace{(\mathbf{X}{:}\sigma,\,\mathbf{Y}{:}\tau)}_{\Delta}} \mathbf{A} \sim \lambda\mathbf{f}(\mathbf{X}\,\mathbf{Y}){:}(\mathbf{X}{:}\sigma,\,\mathbf{Y}{:}\tau).\mathbf{A}$$

- Instead of simple variables we abstract over sophisticated patterns
- The free variables of A (bound in B) are declared in the context Δ, $i.e.$

$$\mathcal{FV}\mathbf{ar}(\mathbf{A} \twoheadrightarrow_{\mathbf{\Delta}} \mathbf{B}) \triangleq (\mathcal{FV}\mathbf{ar}(\mathbf{A}) \cup \mathcal{FV}\mathbf{ar}(\mathbf{B}) \cup \mathcal{FV}\mathbf{ar}(\mathbf{\Delta})) \setminus \mathcal{D}\mathbf{om}(\mathbf{\Delta})$$

Galleria I: The Pattern Abstraction $\mathbf{A} \twoheadrightarrow_{\Delta} \mathbf{B}$

- Generalisation of the λ-abstraction in PTSs. The rationale is:

$$\mathbf{f(X\,Y)} \twoheadrightarrow_{\underbrace{\mathbf{(X{:}\sigma,\,Y{:}\tau)}}_{\Delta}} \mathbf{A} \sim \lambda\mathbf{f(X\,Y){:}(X{:}\sigma,\,Y{:}\tau).A}$$

- Instead of simple variables we abstract over sophisticated patterns
- The free variables of A (bound in B) are declared in the context Δ, $i.e.$

$$\mathcal{FV}\mathbf{ar}(\mathbf{A} \twoheadrightarrow_{\mathbf{\Delta}} \mathbf{B}) \triangleq (\mathcal{FV}\mathbf{ar}(\mathbf{A}) \cup \mathcal{FV}\mathbf{ar}(\mathbf{B}) \cup \mathcal{FV}\mathbf{ar}(\mathbf{\Delta})) \setminus \mathcal{D}\mathbf{om}(\mathbf{\Delta})$$

- Δ discriminates on which $\mathcal{FV}ar(A)$ will be bound in B and which not

$$\mathbf{cons(T\,X\,nil(T))} \twoheadrightarrow_{\mathbf{(X{:}T)}} \mathbf{cons(T\,X\,cons(T\,X\,nil(T)))}$$

Galleria I: The Pattern Abstraction $\mathbf{A} \twoheadrightarrow_{\Delta} \mathbf{B}$

- Generalisation of the λ-abstraction in PTSs. The rationale is:

$$\mathbf{f(X\,Y)} \twoheadrightarrow_{\underbrace{(\mathbf{X}:\sigma,\,\mathbf{Y}:\tau)}_{\Delta}} \mathbf{A} \sim \lambda \mathbf{f(X\,Y)}:(\mathbf{X}:\sigma,\,\mathbf{Y}:\tau).\mathbf{A}$$

- Instead of simple variables we abstract over sophisticated patterns
- The free variables of A (bound in B) are declared in the context Δ, $i.e.$

$$\mathcal{FV}\mathbf{ar}(\mathbf{A} \twoheadrightarrow_{\mathbf{\Delta}} \mathbf{B}) \triangleq (\mathcal{FV}\mathbf{ar}(\mathbf{A}) \cup \mathcal{FV}\mathbf{ar}(\mathbf{B}) \cup \mathcal{FV}\mathbf{ar}(\mathbf{\Delta})) \setminus \mathcal{D}\mathbf{om}(\mathbf{\Delta})$$

- Δ discriminates on which $\mathcal{FV}ar(A)$ will be bound in B and which not

$$\mathbf{cons(T\ X\ nil(T))} \twoheadrightarrow_{\mathbf{(X:T)}} \mathbf{cons(T\ X\ cons(T\ X\ nil(T)))}$$

Galleria II: The Matching Constraint $[\mathbf{A} \ll_{\mathbf{\Delta}} \mathbf{B}]\mathbf{C}$

- In the term

$$[\mathbf{A} \ll_{\mathbf{\Delta}} \mathbf{B}]\mathbf{C}$$

  the matching equation $[A \ll_{\Delta} B]$ is put on the stack, hence constraints and *"de facto"* blocks the evaluation of C

- The body C will be evaluated (in case a matching solution exists) or delayed (in case no solution exists at this stage of the evaluation)

- If a solution exists, the delayed matching constraint self-evaluates to $C\sigma$, otherwise the evaluation is delayed to a later stage

- The free variables of A declared in Δ are bound in B but not in C, *i.e.*

$$\mathcal{FV}\mathbf{ar}([\mathbf{A} \ll_{\mathbf{\Delta}} \mathbf{B}]\mathbf{C}) \triangleq \mathcal{FV}\mathbf{ar}((\mathbf{A} \twoheadrightarrow_{\mathbf{\Delta}} \mathbf{C})\,\mathbf{B})$$

P^2TS

Matching Algorithm

$$\exists \sigma . \mathcal{A}lg(P \lessdot^{\Delta}_{\emptyset} A) \ ?$$

INSIDE ALGO HARD RUN EASY RUN SKIP

Matching Systems

1. A *matching system* $\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} P_i \prec\!\!\prec^{\Delta}_{\Gamma_i} A_i$ is a conjunction of match equations, where $\wedge$ is idempotent, associative and commutative.

Matching Systems

1. A *matching system* $\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} P_i \prec\!\!\prec^{\Delta}_{\Gamma_i} A_i$ is a conjunction of match equations, where $\wedge$ is idempotent, associative and commutative.
2. A matching system T is solved by the substitution σ if $\mathcal{D}om(\sigma) \subseteq \mathcal{D}om(\Sigma) \setminus \mathcal{D}om(\Delta)$ and for all $i = 0 \ldots n$, $P_i\sigma \equiv B_i$

Matching Systems

1. A *matching system* $\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} P_i \prec\!\!\prec^{\Delta}_{\Gamma_i} A_i$ is a conjunction of match equations, where $\wedge$ is idempotent, associative and commutative.
2. A matching system T is solved by the substitution σ if $\mathcal{D}om(\sigma) \subseteq \mathcal{D}om(\Sigma) \setminus \mathcal{D}om(\Delta)$ and for all $i = 0 \ldots n$, $P_i\sigma \equiv B_i$
3. A matching system T is in normal form when it satisfies the following conditions:

Matching Systems

1. A *matching system* $\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} P_i \prec\!\!\prec^{\Delta}_{\Gamma_i} A_i$ is a conjunction of match equations, where $\wedge$ is idempotent, associative and commutative.
2. A matching system T is solved by the substitution σ if $\mathcal{D}om(\sigma) \subseteq \mathcal{D}om(\Sigma) \setminus \mathcal{D}om(\Delta)$ and for all $i = 0 \ldots n$, $P_i\sigma \equiv B_i$
3. A matching system T is in normal form when it satisfies the following conditions:

   (a) $\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} X_i \prec\!\!\prec^{\Sigma}_{\Delta_i} C_i \bigwedge_{j=0\ldots m} f_j \prec\!\!\prec^{\Sigma}_{\Delta_j} f_j$

Matching Systems

1. A *matching system* $\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} P_i \prec\!\!\prec^{\Delta}_{\Gamma_i} A_i$ is a conjunction of match equations, where $\wedge$ is idempotent, associative and commutative.
2. A matching system T is solved by the substitution σ if $\mathcal{D}om(\sigma) \subseteq \mathcal{D}om(\Sigma) \setminus \mathcal{D}om(\Delta)$ and for all $i = 0 \ldots n$, $P_i\sigma \equiv B_i$
3. A matching system T is in normal form when it satisfies the following conditions:
   (a) $\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} X_i \prec\!\!\prec^{\Sigma}_{\Delta_i} C_i \bigwedge_{j=0\ldots m} f_j \prec\!\!\prec^{\Sigma}_{\Delta_j} f_j$
   (b) for all $h, k = 0 \ldots n$, $X_h \equiv X_k$ implies $C_h \equiv C_k$
   (c) for all $i = 0 \ldots n$, $X_i \in \mathcal{D}om(\Delta_i)$ or $X_i \notin \mathcal{D}om(\Sigma)$ implies $X_i \equiv C_i$
   (d) for all $i = 0 \ldots n$, $\mathcal{FV}ar(C_i) \cap \mathcal{D}om(\Delta_i) \neq \emptyset$ implies $X_i \equiv C_i$

Matching Systems

1. A *matching system* $\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} P_i \lessdot^{\Delta}_{\Gamma_i} A_i$ is a conjunction of match equations, where $\wedge$ is idempotent, associative and commutative.
2. A matching system T is solved by the substitution σ if $\mathcal{D}om(\sigma) \subseteq \mathcal{D}om(\Sigma) \setminus \mathcal{D}om(\Delta)$ and for all $i = 0 \ldots n$, $P_i\sigma \equiv B_i$
3. A matching system T is in normal form when it satisfies the following conditions:
   (a) $\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} X_i \lessdot^{\Sigma}_{\Delta_i} C_i \bigwedge_{j=0\ldots m} f_j \lessdot^{\Sigma}_{\Delta_j} f_j$
   (b) for all $h, k = 0 \ldots n$, $X_h \equiv X_k$ implies $C_h \equiv C_k$
   (c) for all $i = 0 \ldots n$, $X_i \in \mathcal{D}om(\Delta_i)$ or $X_i \notin \mathcal{D}om(\Sigma)$ implies $X_i \equiv C_i$
   (d) for all $i = 0 \ldots n$, $\mathcal{FV}ar(C_i) \cap \mathcal{D}om(\Delta_i) \neq \emptyset$ implies $X_i \equiv C_i$
4. if condition 3 is satisfied the matching system T produces the substitution $\{C_1/X_1 \cdots C_n/X_n\}$, otherwise the matching fails

The Algorithm $\mathcal{A}lg$

$$(Lbd/Prod) \qquad (P_1 \twoheadrightarrow_\Delta B_1) \lessdot^\Sigma_\Gamma (P_2 \twoheadrightarrow_\Delta B_2)$$

$$\rightsquigarrow \quad P_1 \lessdot^\Sigma_{\Gamma,\Delta} P_2 \wedge B_1 \lessdot^\Sigma_{\Gamma,\Delta} B_2$$

The Algorithm $\mathcal{A}lg$

$$(Lbd/Prod) \qquad (P_1 \rightarrowtriangle_\Delta B_1) \lessdot\!\!\!\cdot^{\Sigma}_{\Gamma} (P_2 \rightarrowtriangle_\Delta B_2)$$

$$\rightsquigarrow \quad P_1 \lessdot\!\!\!\cdot^{\Sigma}_{\Gamma,\Delta} P_2 \wedge B_1 \lessdot\!\!\!\cdot^{\Sigma}_{\Gamma,\Delta} B_2$$

$$(Delay) \qquad [P_1 \ll_\Delta C_1].B_1 \lessdot\!\!\!\cdot^{\Sigma}_{\Gamma} [P_2 \ll_\Delta C_2].B_2$$

$$\rightsquigarrow \quad P_1 \lessdot\!\!\!\cdot^{\Sigma}_{\Gamma,\Delta} P_2 \wedge B_1 \lessdot\!\!\!\cdot^{\Sigma}_{\Gamma,\Delta} B_2 \wedge C_1 \lessdot\!\!\!\cdot^{\Sigma}_{\Gamma} C_2$$

The Algorithm $\mathcal{A}lg$

$$(Lbd/Prod) \qquad (P_1 \rightarrowtriangle_\Delta B_1) \lessdot\!\!\!\cdot^\Sigma_\Gamma (P_2 \rightarrowtriangle_\Delta B_2)$$

$$\rightsquigarrow \quad P_1 \lessdot\!\!\!\cdot^\Sigma_{\Gamma,\Delta} P_2 \wedge B_1 \lessdot\!\!\!\cdot^\Sigma_{\Gamma,\Delta} B_2$$

$$(Delay) \qquad [P_1 \ll_\Delta C_1].B_1 \lessdot\!\!\!\cdot^\Sigma_\Gamma [P_2 \ll_\Delta C_2].B_2$$

$$\rightsquigarrow \quad P_1 \lessdot\!\!\!\cdot^\Sigma_{\Gamma,\Delta} P_2 \wedge B_1 \lessdot\!\!\!\cdot^\Sigma_{\Gamma,\Delta} B_2 \wedge C_1 \lessdot\!\!\!\cdot^\Sigma_\Gamma C_2$$

$$(Appl/Struct) \qquad (A_1 \fatsemi B_1) \lessdot\!\!\!\cdot^\Sigma_\Gamma (A_2 \fatsemi B_2)$$

$$\rightsquigarrow \quad A_1 \lessdot\!\!\!\cdot^\Sigma_\Gamma A_2 \wedge B_1 \lessdot\!\!\!\cdot^\Sigma_\Gamma B_2$$

The Algorithm $\mathcal{A}lg$

$$(Lbd/Prod) \qquad (P_1 \rightarrowtriangle_\Delta B_1) \lessdot^\Sigma_\Gamma (P_2 \rightarrowtriangle_\Delta B_2)$$

$$\rightsquigarrow \quad P_1 \lessdot^\Sigma_{\Gamma,\Delta} P_2 \wedge B_1 \lessdot^\Sigma_{\Gamma,\Delta} B_2$$

$$(Delay) \qquad [P_1 \ll_\Delta C_1].B_1 \lessdot^\Sigma_\Gamma [P_2 \ll_\Delta C_2].B_2$$

$$\rightsquigarrow \quad P_1 \lessdot^\Sigma_{\Gamma,\Delta} P_2 \wedge B_1 \lessdot^\Sigma_{\Gamma,\Delta} B_2 \wedge C_1 \lessdot^\Sigma_\Gamma C_2$$

$$(Appl/Struct) \qquad (A_1 \fatsemi B_1) \lessdot^\Sigma_\Gamma (A_2 \fatsemi B_2)$$

$$\rightsquigarrow \quad A_1 \lessdot^\Sigma_\Gamma A_2 \wedge B_1 \lessdot^\Sigma_\Gamma B_2$$

recall the reduction rule $[\mathbf{P} \ll_{\mathbf{\Theta}} \mathbf{C}].\mathbf{B} \mapsto_\sigma \mathbf{B}\sigma_{(\mathbf{P} \lessdot^{\mathbf{\Theta}}_{\emptyset} \mathbf{C})}$

Termination of $\mathcal{A}lg$

- The relation $\rightsquigarrow\!\!\!\!\rightarrow$ is defined as the reflexive, transitive and compatible closure of $\rightsquigarrow$
- If T $\rightsquigarrow\!\!\!\!\rightarrow$ T′, with T′ a matching system in solved form then, we say that the matching algorithm $\mathcal{A}lg$ (taking as input the system T) succeeds
- The matching algorithm is clearly terminating (since all rules decrease the size of terms) and deterministic (no critical pairs), and of course, it works modulo α-conversion and Barendregt's hygiene-convention
- Starting form a given solved matching system of the form

$$\mathsf{T} \stackrel{\triangle}{=} \bigwedge_{\mathbf{i=0\ldots n}} \mathbf{X_i} \lessdot^{\mathbf{\Sigma}}_{\mathbf{\Delta_i}} \mathbf{A_i} \bigwedge_{\mathbf{j=0\ldots m}} \mathbf{a_j} \lessdot^{\mathbf{\Sigma}}_{\mathbf{\Delta_j}} \mathbf{a_j}$$

the corresponding substitution $\{A_1/X_1 \cdots A_n/X_n\}$ is exhibited.

Less *Easy Running*

- $X \twoheadrightarrow_{(X:i)} X \twoheadleftarrow^{?}_{\emptyset} X \twoheadrightarrow_{(X:i)} X$

Less *Easy Running*

- $X \twoheadrightarrow_{(X:i)} X \lessdot^{?}_{\emptyset} X \twoheadrightarrow_{(X:i)} X \rightsquigarrow X \lessdot^{?}_{X} X$ OK!!

Less *Easy Running*

- $X \twoheadrightarrow_{(X:i)} X \lessdot^{?}_{\emptyset} X \twoheadrightarrow_{(X:i)} X \rightsquigarrow X \lessdot^{?}_{X} X$ OK!!
- $X \twoheadrightarrow_{(X:i)} X \lessdot^{?}_{\emptyset} X \twoheadrightarrow_{(X:i)} Y$

Less *Easy Running*

- $X \twoheadrightarrow_{(X:i)} X \lessdot^{?}_{\emptyset} X \twoheadrightarrow_{(X:i)} X \rightsquigarrow X \lessdot^{?}_{X} X$ OK!!
- $X \twoheadrightarrow_{(X:i)} X \lessdot^{?}_{\emptyset} X \twoheadrightarrow_{(X:i)} Y \rightsquigarrow X \lessdot^{?}_{X} X \wedge X \lessdot^{?}_{X} Y$ KO!!

Less *Easy Running*

- $X \to_{(X:i)} X \lessdot^{?}_{\emptyset} X \to_{(X:i)} X \rightsquigarrow X \lessdot^{?}_{X} X$ OK!!
- $X \to_{(X:i)} X \lessdot^{?}_{\emptyset} X \to_{(X:i)} Y \rightsquigarrow X \lessdot^{?}_{X} X \wedge X \lessdot^{?}_{X} Y$ KO!!
- $X \to_{(X:i)} f(X\,Y) \lessdot^{Y}_{\emptyset} X \to_{(X:i)} f(X\,3) \rightsquigarrow\!\!\!\rightarrow$

Less *Easy Running*

- $X \twoheadrightarrow_{(X:i)} X \lessdot^{?}_{\emptyset} X \twoheadrightarrow_{(X:i)} X \rightsquigarrow X \lessdot^{?}_{X} X$ OK!!
- $X \twoheadrightarrow_{(X:i)} X \lessdot^{?}_{\emptyset} X \twoheadrightarrow_{(X:i)} Y \rightsquigarrow X \lessdot^{?}_{X} X \wedge X \lessdot^{?}_{X} Y$ KO!!
- $X \twoheadrightarrow_{(X:i)} f(X\,Y) \lessdot^{Y}_{\emptyset} X \twoheadrightarrow_{(X:i)} f(X\,3) \rightsquigarrow\!\!\!\twoheadrightarrow$

  $X \lessdot^{Y}_{X} X \wedge f \lessdot^{Y}_{X} f \wedge X \lessdot^{Y}_{X} X \wedge Y \lessdot^{Y}_{X} 3$ OK!!

Less *Easy Running*

- $X \to_{(X:i)} X \lessdot^{?}_{\emptyset} X \to_{(X:i)} X \rightsquigarrow X \lessdot^{?}_{X} X$ OK!!
- $X \to_{(X:i)} X \lessdot^{?}_{\emptyset} X \to_{(X:i)} Y \rightsquigarrow X \lessdot^{?}_{X} X \wedge X \lessdot^{?}_{X} Y$ KO!!
- $X \to_{(X:i)} f(X\,Y) \lessdot^{Y}_{\emptyset} X \to_{(X:i)} f(X\,3) \rightsquigarrow\!\!\!\rightarrow$

  $X \lessdot^{Y}_{X} X \wedge f \lessdot^{Y}_{X} f \wedge X \lessdot^{Y}_{X} X \wedge Y \lessdot^{Y}_{X} 3$ OK!!
- $[f(X) \ll_{(X:i)} f(Y)]X \lessdot^{Y}_{\emptyset} [f(X) \ll_{(X:i)} f(3)].X \rightsquigarrow\!\!\!\rightarrow$

Less *Easy Running*

- $X \to_{(X:i)} X \lessdot^{?}_{\emptyset} X \to_{(X:i)} X \rightsquigarrow X \lessdot^{?}_{X} X$ OK!!
- $X \to_{(X:i)} X \lessdot^{?}_{\emptyset} X \to_{(X:i)} Y \rightsquigarrow X \lessdot^{?}_{X} X \wedge X \lessdot^{?}_{X} Y$ KO!!
- $X \to_{(X:i)} f(X\,Y) \lessdot^{Y}_{\emptyset} X \to_{(X:i)} f(X\,3) \rightsquigarrow\!\!\!\rightarrow$

  $X \lessdot^{Y}_{X} X \wedge f \lessdot^{Y}_{X} f \wedge X \lessdot^{Y}_{X} X \wedge Y \lessdot^{Y}_{X} 3$ OK!!
- $[f(X) \ll_{(X:i)} f(Y)]X \lessdot^{Y}_{\emptyset} [f(X) \ll_{(X:i)} f(3)].X \rightsquigarrow\!\!\!\rightarrow$

  $f \lessdot^{Y}_{X} f \wedge X \lessdot^{Y}_{X} X \wedge X \lessdot^{Y}_{X} X \wedge f \lessdot^{Y}_{\emptyset} f \wedge Y \lessdot^{Y}_{\emptyset} 3$ OK!!

Two *Easy Running*

- $(\mathbf{cons(T\ X\ nil(T))} \twoheadrightarrow_{(\mathbf{X:i})} \mathbf{X})\ \mathbf{cons(T\ 3\ nil(T))}$

Two *Easy Running*

- $(\mathbf{cons(T\ X\ nil(T))} \twoheadrightarrow_{(\mathrm{X:i})} \mathbf{X})\ \mathbf{cons(T\ 3\ nil(T))}$

Solve $\mathbf{cons(T\ X\ nil(T))} \prec\!\!\prec_{\emptyset}^{\mathbf{X}} \mathbf{cons(T\ 3\ nil(T))}$

Two *Easy Running*

- $(\mathbf{cons(T\ X\ nil(T))} \rightarrowtriangle_{(\mathbf{X:i})} \mathbf{X})\ \mathbf{cons(T\ 3\ nil(T))}$

Solve $\mathbf{cons(T\ X\ nil(T))} \lessdot^{\mathbf{X}}_{\emptyset} \mathbf{cons(T\ 3\ nil(T))} \rightsquigarrow$

$\mathbf{X} \lessdot^{\mathbf{X}}_{\emptyset} \mathbf{3} \wedge \mathbf{T} \lessdot^{\mathbf{X}}_{\emptyset} \mathbf{T}$ **OK!! with** $\sigma = \{\mathbf{3/X}\}$

Two *Easy Running*

- $(\mathbf{cons(T\ X\ nil(T))} \rightarrowtriangle_{(\mathrm{X:i})} \mathbf{X})\ \mathbf{cons(T\ 3\ nil(T))}$

  Solve $\mathbf{cons(T\ X\ nil(T))} \lessdot_{\emptyset}^{\mathbf{X}} \mathbf{cons(T\ 3\ nil(T))} \rightsquigarrow$

  $\mathbf{X} \lessdot_{\emptyset}^{\mathbf{X}} \mathbf{3} \wedge \mathbf{T} \lessdot_{\emptyset}^{\mathbf{X}} \mathbf{T}$ **OK!! with** $\sigma = \{\mathbf{3/X}\}$

- $(\mathbf{cons(T\ X\ nil(T))} \rightarrowtriangle_{(\mathrm{X:i})} \mathbf{X})\ \mathbf{cons(i\ 3\ nil(i))}$

Two *Easy Running*

- $(\mathbf{cons(T\ X\ nil(T))} \rightarrow_{(\mathrm{X:i})} \mathbf{X})\ \mathbf{cons(T\ 3\ nil(T))}$

  Solve $\mathbf{cons(T\ X\ nil(T))} \prec^{\mathbf{X}}_{\emptyset} \mathbf{cons(T\ 3\ nil(T))} \rightsquigarrow$

  $\mathbf{X} \prec^{\mathbf{X}}_{\emptyset} \mathbf{3} \wedge \mathbf{T} \prec^{\mathbf{X}}_{\emptyset} \mathbf{T}$ **OK!! with** $\sigma = \{\mathbf{3/X}\}$

- $(\mathbf{cons(T\ X\ nil(T))} \rightarrow_{(\mathrm{X:i})} \mathbf{X})\ \mathbf{cons(i\ 3\ nil(i))}$

  Solve $\mathbf{cons(T\ X\ nil(T))} \prec^{\mathbf{X}}_{\emptyset} \mathbf{cons(i\ 3\ nil(i))}$

Two *Easy Running*

- $(\mathbf{cons(T\ X\ nil(T))} \twoheadrightarrow_{(\mathbf{X:i})} \mathbf{X})\ \mathbf{cons(T\ 3\ nil(T))}$

  Solve $\mathbf{cons(T\ X\ nil(T))} \lessdot^{\mathbf{X}}_{\emptyset} \mathbf{cons(T\ 3\ nil(T))} \rightsquigarrow$

  $\mathbf{X} \lessdot^{\mathbf{X}}_{\emptyset} \mathbf{3} \wedge \mathbf{T} \lessdot^{\mathbf{X}}_{\emptyset} \mathbf{T}$ **OK!! with** $\sigma = \{\mathbf{3/X}\}$

- $(\mathbf{cons(T\ X\ nil(T))} \twoheadrightarrow_{(\mathbf{X:i})} \mathbf{X})\ \mathbf{cons(i\ 3\ nil(i))}$

  Solve $\mathbf{cons(T\ X\ nil(T))} \lessdot^{\mathbf{X}}_{\emptyset} \mathbf{cons(i\ 3\ nil(i))} \rightsquigarrow$

  $\mathbf{X} \lessdot^{\mathbf{X}}_{\emptyset} \mathbf{3} \wedge \mathbf{T} \lessdot^{\mathbf{X}}_{\emptyset} \mathbf{i}$ **KO!!**

P^2TS

Type System

The Type System I

$$\frac{(s_1, s_2) \in \rangle A}{\emptyset \vdash s_1 : s_2}\,(Axioms)$$

The Type System I

$$\frac{(s_1, s_2) \in \rangle A}{\emptyset \vdash s_1 : s_2}\,(Axioms)$$

$$\frac{\Gamma \vdash A : C \qquad \Gamma \vdash B : C}{\Gamma \vdash A, B : C}\,(Struct)$$

The Type System I

$$\frac{(s_1, s_2) \in \rangle A}{\emptyset \vdash s_1 : s_2}\,(Axioms)$$

$$\frac{\Gamma \vdash A : C \qquad \Gamma \vdash B : C}{\Gamma \vdash A, B : C}\,(Struct)$$

$$\frac{\Gamma \vdash A : s \qquad \alpha \notin Dom(\Gamma)}{\Gamma, \alpha{:}A \vdash \alpha : A}\,(Start)$$

The Type System I

$$\frac{(s_1, s_2) \in \rangle A}{\emptyset \vdash s_1 : s_2}\,(Axioms)$$

$$\frac{\Gamma \vdash A : C \qquad \Gamma \vdash B : C}{\Gamma \vdash A, B : C}\,(Struct)$$

$$\frac{\Gamma \vdash A : s \qquad \alpha \notin Dom(\Gamma)}{\Gamma, \alpha{:}A \vdash \alpha : A}\,(Start)$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s \qquad \alpha \notin Dom(\Gamma)}{\Gamma, \alpha{:}C \vdash A : B}\,(Weak)$$

The Type System I

$$\frac{(s_1, s_2) \in\rangle A}{\emptyset \vdash s_1 : s_2}\,(Axioms)$$

$$\frac{\Gamma \vdash A : C \qquad \Gamma \vdash B : C}{\Gamma \vdash A, B : C}\,(Struct)$$

$$\frac{\Gamma \vdash A : s \qquad \alpha \notin Dom(\Gamma)}{\Gamma, \alpha{:}A \vdash \alpha : A}\,(Start)$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s \qquad \alpha \notin Dom(\Gamma)}{\Gamma, \alpha{:}C \vdash A : B}\,(Weak)$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : D \qquad B =_{\beta\delta} C}{\Gamma \vdash A : C}\,(Conv)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_{\Delta} B : \Pi A{:}\Delta.C}\,(Abs)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_{\Delta} B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{c}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{l}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\Gamma \vdash A : \Pi C{:}\Delta.D \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}{\Gamma \vdash A\ B : [C \ll_\Delta B]D}\,(Appl)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{c}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\Gamma \vdash A : \Pi C{:}\Delta.D \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}{\Gamma \vdash A\ B : [C \ll_\Delta B]D}\,(Appl)$$

$$\frac{\Gamma, \Delta \vdash A : E \qquad \Gamma, \Delta \vdash C : D \qquad \Gamma \vdash B : D}{\Gamma \vdash [C \ll_\Delta B]A : [C \ll_\Delta B]^\top E}\,(Subst)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{l}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\mathbf{\Gamma \vdash A : \Pi C{:}\Delta.D} \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}{\Gamma \vdash A\ B : [C \ll_\Delta B]D}\,(Appl)$$

$$\frac{\Gamma, \Delta \vdash A : E \qquad \Gamma, \Delta \vdash C : D \qquad \Gamma \vdash B : D}{\Gamma \vdash [C \ll_\Delta B]A : [C \ll_\Delta B]^\top E}\,(Subst)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{c}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\mathbf{\Gamma \vdash A : \Pi C{:}\Delta.D} \qquad \mathbf{\Gamma, \Delta \vdash C : E} \qquad \Gamma \vdash B : E}{\Gamma \vdash A\ B : [C \ll_\Delta B]D}\,(Appl)$$

$$\frac{\Gamma, \Delta \vdash A : E \qquad \Gamma, \Delta \vdash C : D \qquad \Gamma \vdash B : D}{\Gamma \vdash [C \ll_\Delta B]A : [C \ll_\Delta B]^\top E}\,(Subst)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{c}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\mathbf{\Gamma \vdash A : \Pi C{:}\Delta.D \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}}{\Gamma \vdash A\ B : [C \ll_\Delta B]D}\,(Appl)$$

$$\frac{\Gamma, \Delta \vdash A : E \qquad \Gamma, \Delta \vdash C : D \qquad \Gamma \vdash B : D}{\Gamma \vdash [C \ll_\Delta B]A : [C \ll_\Delta B]^\top E}\,(Subst)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{c}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\mathbf{\Gamma \vdash A : \Pi C{:}\Delta.D \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}}{\mathbf{\Gamma \vdash A\ B : [C \ll_\Delta B]D}}\,(Appl)$$

$$\frac{\Gamma, \Delta \vdash A : E \qquad \Gamma, \Delta \vdash C : D \qquad \Gamma \vdash B : D}{\Gamma \vdash [C \ll_\Delta B]A : [C \ll_\Delta B]^\top E}\,(Subst)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{l}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\mathbf{\Gamma \vdash A : \Pi C{:}\Delta.D \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}}{\mathbf{\Gamma \vdash A\ B : [C \ll_\Delta B]D}}\,(Appl)$$

$$\frac{\mathbf{\Gamma, \Delta \vdash A : E} \qquad \Gamma, \Delta \vdash C : D \qquad \Gamma \vdash B : D}{\Gamma \vdash [C \ll_\Delta B]A : [C \ll_\Delta B]^\top E}\,(Subst)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{l}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\mathbf{\Gamma \vdash A : \Pi C{:}\Delta.D \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}}{\mathbf{\Gamma \vdash A\ B : [C \ll_\Delta B]D}}\,(Appl)$$

$$\frac{\mathbf{\Gamma, \Delta \vdash A : E \qquad \Gamma, \Delta \vdash C : D} \qquad \Gamma \vdash B : D}{\Gamma \vdash [C \ll_\Delta B]A : [C \ll_\Delta B]^\top E}\,(Subst)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{l}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\mathbf{\Gamma \vdash A : \Pi C{:}\Delta.D \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}}{\mathbf{\Gamma \vdash A\ B : [C \ll_\Delta B]D}}\,(Appl)$$

$$\frac{\mathbf{\Gamma, \Delta \vdash A : E \qquad \Gamma, \Delta \vdash C : D \qquad \Gamma \vdash B : D}}{\Gamma \vdash [C \ll_\Delta B]A : [C \ll_\Delta B]^\top E}\,(Subst)$$

The Type System II

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash A \twoheadrightarrow_\Delta B : \Pi A{:}\Delta.C}\,(Abs)$$

$$\frac{\begin{array}{l}\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\,(Prod)$$

$$\frac{\mathbf{\Gamma \vdash A : \Pi C{:}\Delta.D \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}}{\mathbf{\Gamma \vdash A\ B : [C \ll_\Delta B]D}}\,(Appl)$$

$$\frac{\mathbf{\Gamma, \Delta \vdash A : E \qquad \Gamma, \Delta \vdash C : D \qquad \Gamma \vdash B : D}}{\mathbf{\Gamma \vdash [C \ll_\Delta B]A : [C \ll_\Delta B]^\top E}}\,(Subst)$$

Fetch Your System

$$\frac{\Gamma \vdash C : s_1 \qquad (s_1, s_2, s_3) \in \mathcal{R} \qquad \Gamma, \Delta \vdash A : C \qquad \Gamma, \Delta \vdash B : s_2}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\ (Prod)$$

System	Rules			
$\rho_{\rightarrow}$	$(*,*,*)$			
$\rho 2$	$(*,*,*)$	$(\Box,*,*)$		
$\rho\underline{\omega}$	$(*,*,*)$			$(\Box,\Box,\Box)$
$\rho\omega$	$(*,*,*)$		$(*,\Box,\Box)$	$(\Box,\Box,\Box)$
ρLF	$(*,*,*)$		$(*,\Box,\Box)$	
$\rho P2$	$(*,*,*)$	$(\Box,*,*)$	$(*,\Box,\Box)$	
$\rho P\underline{\omega}$	$(*,*,*)$		$(*,\Box,\Box)$	$(\Box,\Box,\Box)$
$\rho P\omega$	$(*,*,*)$	$(\Box,*,*)$	$(*,\Box,\Box)$	$(\Box,\Box,\Box)$

P^2TS

Typed Examples

Example: Simple Type Derivation

Let $\Gamma \triangleq i{:}*, f{:}\Pi Z{:}\Sigma.i, 3{:}i, 4{:}i$, and $\Delta \triangleq X{:}i$, and $\Sigma \triangleq Z{:}i$,

$$\dfrac{\dfrac{\Gamma, \Delta \vdash (\lambda 3{:}\emptyset.3) : \Pi 3{:}\emptyset.i \quad \boxed{5\ 6} \qquad \dfrac{\boxed{3} \qquad (*,*,*) \in \mathcal{R}}{\Gamma \vdash [Z \ll_\Sigma X].i : *}}{\Gamma, \Delta \vdash (\lambda 3{:}\emptyset.3)\, X : [3 \ll_\emptyset X].i} \qquad \dfrac{\Gamma, \Delta \vdash [3 \ll_\emptyset X].i : *}{\Gamma \vdash \Pi f(X){:}\Delta.[3 \ll_\emptyset X].i : *}}{\dfrac{\Gamma \vdash \lambda f(X){:}\Delta.(\lambda 3{:}\emptyset.3)X : \Pi f(X){:}\Delta.[3 \ll_\emptyset X].i \qquad \boxed{3\ 4}}{\dfrac{\Gamma \vdash (\lambda f(X){:}\Delta.(\lambda 3{:}\emptyset.3)X)\, f(3) : [f(X) \ll_\Delta f(3)].[3 \ll_\emptyset X].i \qquad \boxed{1\ 2}}{\Gamma \vdash (\lambda \mathbf{f}(\mathbf{X}){:}\boldsymbol{\Delta}.(\lambda \mathbf{3}{:}\emptyset.\mathbf{3})\mathbf{X})\, \mathbf{f}(\mathbf{3}) : \mathbf{i}}}$$

where $\boxed{1} \triangleq [f(X) \ll_\Delta f(3)].[3 \ll_\emptyset X].i =_{\rho\partial\delta} i$, and $\boxed{2} \triangleq \Gamma \vdash i : *$, and $\boxed{3} \triangleq \Gamma, \Delta \vdash f(X) : [Z \ll_\Sigma X].i$, and $\boxed{4} \triangleq \Gamma \vdash f(3) : [Z \ll_\Sigma 3].i$, and $\boxed{5} \triangleq \Gamma, \Delta \vdash X : i$, and $\boxed{6} \triangleq \Gamma, \Delta \vdash 3 : i$.

Example: Simple Type Derivation

Let $\Gamma \triangleq i{:}*, f{:}\Pi Z{:}\Sigma.i, 3{:}i, 4{:}i$, and $\Delta \triangleq X{:}i$, and $\Sigma \triangleq Z{:}i$,

$$\dfrac{\dfrac{\Gamma, \Delta \vdash (\lambda 3{:}\emptyset.3) : \Pi 3{:}\emptyset.i \quad \boxed{5\ 6} \qquad \dfrac{\boxed{3} \qquad (*,*,*) \in \mathcal{R}}{\Gamma \vdash [Z \ll_\Sigma X].i : *}}{\Gamma, \Delta \vdash (\lambda 3{:}\emptyset.3)\, X : [3 \ll_\emptyset X].i} \qquad \dfrac{\Gamma, \Delta \vdash [3 \ll_\emptyset X].i : *}{\Gamma \vdash \Pi f(X){:}\Delta.[3 \ll_\emptyset X].i : *}}{\dfrac{\Gamma \vdash \lambda f(X){:}\Delta.(\lambda 3{:}\emptyset.3)X : \Pi f(X){:}\Delta.[3 \ll_\emptyset X].i \quad \boxed{3\ 4}}{\dfrac{\Gamma \vdash (\lambda f(X){:}\Delta.(\lambda 3{:}\emptyset.3)X)\, f(3) : [f(X) \ll_\Delta f(3)].[3 \ll_\emptyset X].i \quad \boxed{1\ 2}}{\Gamma \vdash \mathbf{(\lambda f(X){:}\Delta.(\lambda 3{:}\emptyset.3)X)\, f(3) : i}}}$$

where $\boxed{1} \triangleq [f(X) \ll_\Delta f(3)].[3 \ll_\emptyset X].i =_{\rho\partial\delta} i$, and $\boxed{2} \triangleq \Gamma \vdash i : *$, and $\boxed{3} \triangleq \Gamma, \Delta \vdash f(X) : [Z \ll_\Sigma X].i$, and $\boxed{4} \triangleq \Gamma \vdash f(3) : [Z \ll_\Sigma 3].i$, and $\boxed{5} \triangleq \Gamma, \Delta \vdash X : i$, and $\boxed{6} \triangleq \Gamma, \Delta \vdash 3 : i$.

Playing with the Rho-cube: ρLF

- Let $\Gamma \triangleq i{:}*, f{:}\Pi X{:}(X{:}i).*, 3{:}i$

$$\dfrac{\dfrac{\dfrac{\Gamma, X{:}i \vdash X : i \quad \Gamma \vdash i : * \quad \Gamma, X{:}i \vdash * : \Box}{\Gamma \vdash \Pi X{:}(X{:}i).* : \Box}}{\Gamma \vdash f : \Pi X{:}(X{:}i).*} \qquad \Gamma, X{:}i \vdash X : i \quad \Gamma \vdash 3 : i}{\Gamma \vdash f(3) : [X \ll_{(X:i)} 3]^{\top} * \equiv *}$$

- $\Gamma \vdash \Pi X{:}(X{:}i).* : \Box$ can be derived thanks to the specific rule $(*, \Box, \Box)$

Playing with the Rho-cube: $\rho 2$

- In this system the following polymorphic identity with pattern f" can be derived (where f denotes $f^{X^* \to X^*}$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *}}{\vdash Y^{X^*} : X^*} \quad \dfrac{\dfrac{(Conv{+}Appl)}{\vdots}}{\vdash f(Y^{X^*}) : X^*} \quad \dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *} \quad \dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *}}{X{:}*, Y{:}X \vdash f(Y^{X^*}) \to X : *} \quad \dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *} \quad \dfrac{}{\vdash * : \Box} \quad \dfrac{\dfrac{ok!}{\vdots}}{\vdash f(Y^{X^*}) \to X^* : *}}{X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to X : *}$$

$$\dfrac{\vdash f(Y^{X^*}) \to Y^{X^*} : f(Y^{X^*}) \to X^*}{X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to Y^{X^*} : X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to X : *}$$

Playing with the Rho-cube: $\rho 2$

- In this system the following polymorphic identity with pattern f" can be derived (where f denotes $f^{X^* \to X^*}$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *}}{\vdash Y^{X^*} : X^*} \quad \dfrac{\dfrac{\vdots}{\vdash f(Y^{X^*}) : X^*} \quad \dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *} \quad \dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *}}{X{:}*, Y{:}X \vdash f(Y^{X^*}) \to X : *}\ (Conv{+}Appl)}{\vdash f(Y^{X^*}) \to Y^{X^*} : f(Y^{X^*}) \to X^*} \quad \dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *} \quad \dfrac{}{\vdash * : \Box}}{X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to X : *} \quad \dfrac{\dfrac{ok!}{\vdots}}{\vdash f(Y^{X^*}) \to X^* : *}$$

$$X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to Y^{X^*} : X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to X : *$$

- $X{:}*, Y{:}X \vdash f(Y^{X^*}) \to X : *$ can be derived thanks to $(*, *, *)$

Playing with the Rho-cube: $\rho 2$

- In this system the following polymorphic identity with pattern f" can be derived (where f denotes $f^{X^* \to X^*}$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *}}{\vdash Y^{X^*} : X^*} \quad \dfrac{\dfrac{\vdots}{\vdash f(Y^{X^*}) : X^*}\ (Conv{+}Appl) \quad \dfrac{}{\vdash * : \Box}}{\vdash X^* : *} \quad \dfrac{}{\vdash * : \Box}}{X{:}*, Y{:}X \vdash f(Y^{X^*}) \to X : *}}{\vdash f(Y^{X^*}) \to Y^{X^*} : f(Y^{X^*}) \to X^*} \quad \dfrac{\dfrac{}{\vdash * : \Box}}{\vdash X^* : *} \quad \dfrac{}{\vdash * : \Box} \quad \dfrac{\dfrac{ok!}{\vdots}}{\vdash f(Y^{X^*}) \to X^* : *}}{X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to X : *}}{X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to Y^{X^*} : X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to X : *}$$

- $X{:}*, Y{:}X \vdash f(Y^{X^*}) \to X : *$ can be derived thanks to $(*, *, *)$

- $X{:}*, Y{:}X \vdash X^* \to f(Y^{X^*}) \to X : *$ can be derived thanks to $(\Box, *, *)$

P^2TS

Metatheory

CONCLUSIONS

P^{2}T S: Some Results

- **Confluence** The relation $\mapsto_{\rho\partial\mathcal{S}}$ is confluent

- **Subject Reduction.** If $\Gamma \vdash A : B$, and $A \mapsto_{\rho\partial\mathcal{S}} C$, then $\Gamma \vdash C : B$

P^{2}T S: Some Results

- **Confluence** The relation $\mapsto_{\rho\partial\mathcal{S}}$ is confluent
- **Subject Reduction.** If $\Gamma \vdash A : B$, and $A \mapsto_{\rho\partial\mathcal{S}} C$, then $\Gamma \vdash C : B$
- **Consistency.** Any normalizing P^{2}T S is logically consistent, $i.e.$ for every sort $s \in \mathcal{S}$, $X{:}s \not\vdash A : X$
- **Conservativity.** P^{2}T Ss are a conservative extension of PTSs:

$$\Gamma \vdash_{PTS} A : B \quad \Longleftrightarrow \quad \Gamma^\dagger \vdash_{\mathsf{P^2T\,S}} A^\dagger : B^\dagger$$

Open Tracks

- strong normalization, we conjecture that standard model construction techniques can be used to prove strong normalization of the $\lambda^{\ll}$-cube;
- type checking/inference, we conjecture that existing algorithms for PTSs adapt readily to P^2TSs;
- it would be interesting to study P^2TSs with a limited form of decidable higher-order unification, in the style of λ-Prolog;
- encoding dependent case analysis, dependent sum types (records) $\grave{a}\ la$ Coquand-Pollack-Luo;
- explicit substitutions. The extension is not trivial, because of delayed matching constraints, but the resulting formalism could serve as the core $engine$ of a little type-checker underneath of a powerful proof assistant;

Challenge ... *Extending the Curry-Howard Isomorphism*

- The extension can be considered from the point of view of sequent calculi, deduction modulo, and natural deduction respectively;
- From the point of view of sequent calculi, it remains to investigate how $\mathsf{P}^2\mathsf{TSs}$ can be used to extend previous results on term calculi for sequent calculi, and how their extension with matching theories can be used to provide suitable term calculi for deduction modulo;
- From the point of view of natural deduction, $\mathsf{P}^2\mathsf{TSs}$ correspond to an extension of natural deduction where parts of proof trees are discharged instead of assumptions;
- To our best knowledge, such an extended form of natural deduction has not been considered previously, but it seems interesting to investigate whether such an extended natural deduction could find some applications in proof assistants, *e.g.* for transforming and optimizing proofs.

Logics and Rho *à la* Church

The relation with (intuitionistic) logic through the so-called Curry-Howard isomorphism, or '*formulae-as-types*' principle, has been profoundly studied for Lambda. However, for Rho *à la*Church, this relation is less clear, as demonstrated by the authors. The principle could be adapted as follows:

> *Given a typed term A, if we can derive for A a type τ in the typed system Rho, with a derivation $\mathcal{D}er_{\mathsf{T}}$, then the term A can be seen as the coding of a logical proof, proving the formula φ that can be interpreted as the type τ assigned to A.*

Curry from Church

For those systems, if $\mathcal{D}er_{\mathsf{T}}$ is a typed derivation, and $(\!|-|\!)$ is the above meant erasing function, then by applying $(\!|-|\!)$ to the "subject" of every judgment in $\mathcal{D}er_{\mathsf{T}}$, we obtain a valid type assignment derivation $\mathcal{D}er_{\mathsf{U}}$ with the same structure of the typed one. Vice versa, every type assignment derivation can be viewed as the result of an application of $(\!|-|\!)$ to a typed one. In particular, the erasing function $(\!|-|\!)$ induces an *isomorphism* between every typed system and the corresponding type assignment system.

$$
\begin{array}{lcl}
(\!|f|\!) & \triangleq & f \\
(\!|X|\!) & \triangleq & X \\
(\!|A\ \tau|\!) & \triangleq & (\!|A|\!)
\end{array}
\qquad
\begin{array}{lcl}
(\!|[P \ll_{\Delta} A].B|\!) & \triangleq & [P \ll (\!|A|\!)].(\!|B|\!) \\
(\!|P \twoheadrightarrow_{\Delta} A|\!) & \triangleq & P \twoheadrightarrow (\!|A|\!) \\
(\!|A\ B|\!) & \triangleq & (\!|A|\!)\ (\!|B|\!) \\
(\!|A, B|\!) & \triangleq & (\!|A|\!), (\!|B|\!)
\end{array}
$$

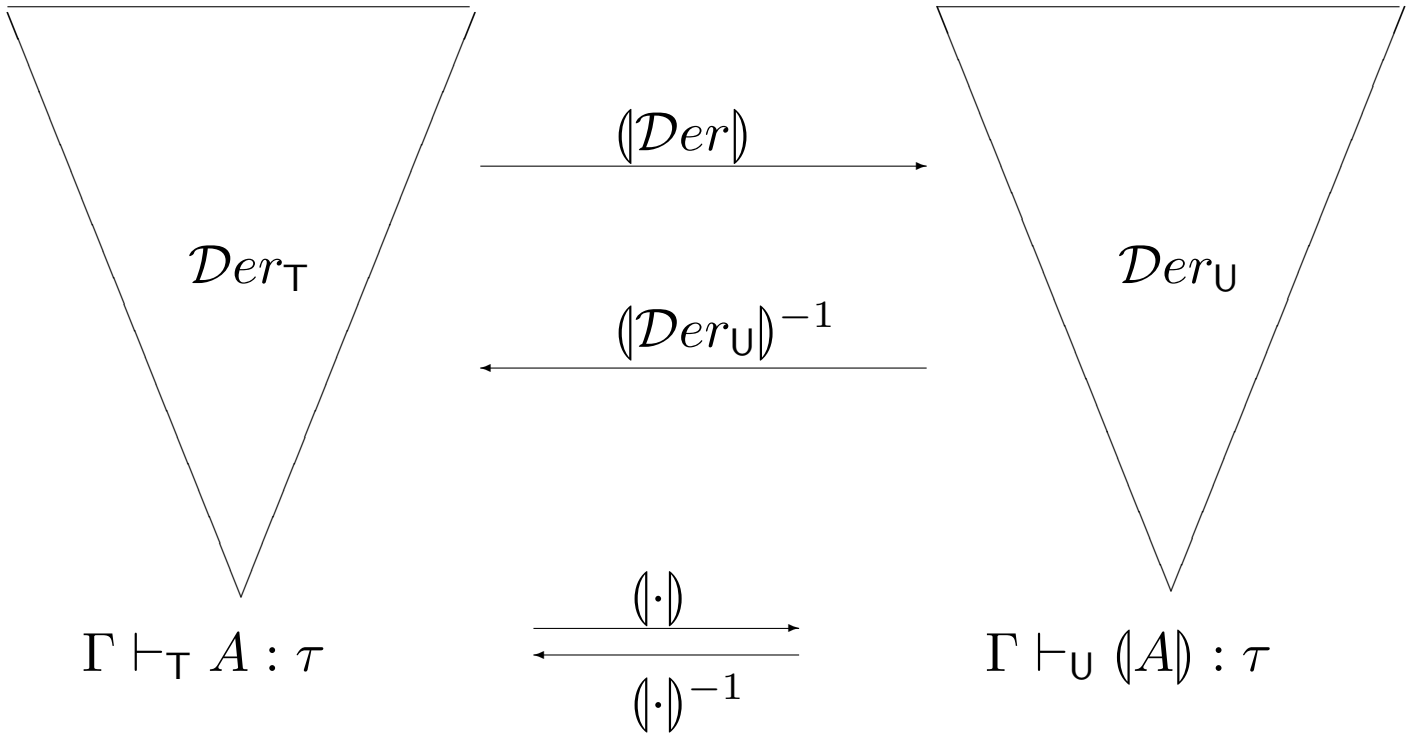
Logics and Rho *à la* Curry

For the type assignment system Rho the relation with logic is less clear even for the corresponding type assignments for the Lambda. The 'formulae-as-types' principle of Curry and Howard could be extended to the above type assignment systems as follows:

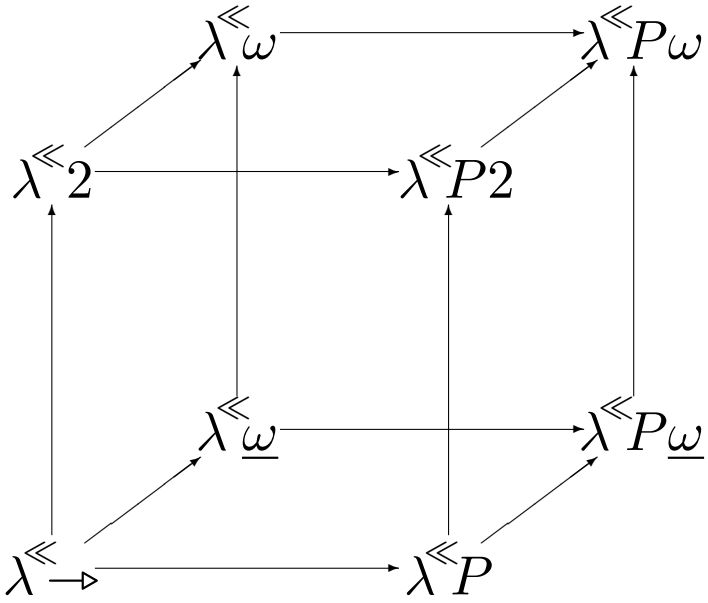
Given an untyped term U, if we can assign a type τ in the type assignment system Rho, with a derivation $\mathcal{D}er_{\mathsf{U}}$, then:

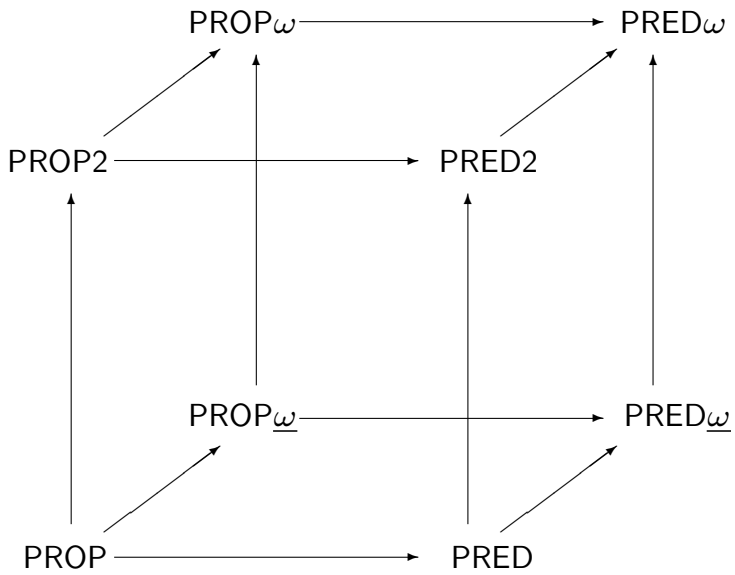
- *$\mathcal{D}er_{\mathsf{U}}$ can be interpreted as the coding of a proof for the logic formulas φ which corresponds to the interpretation of the type τ assigned to U;*
- *U can be interpreted as the coding of a "logical proof schema", whose instances (of the schema) prove, respectively, all the logic formulas φ_i's that can be interpreted as the types τ_i's that can be assigned to U.*

Typed and Untyped Judgments and Derivations



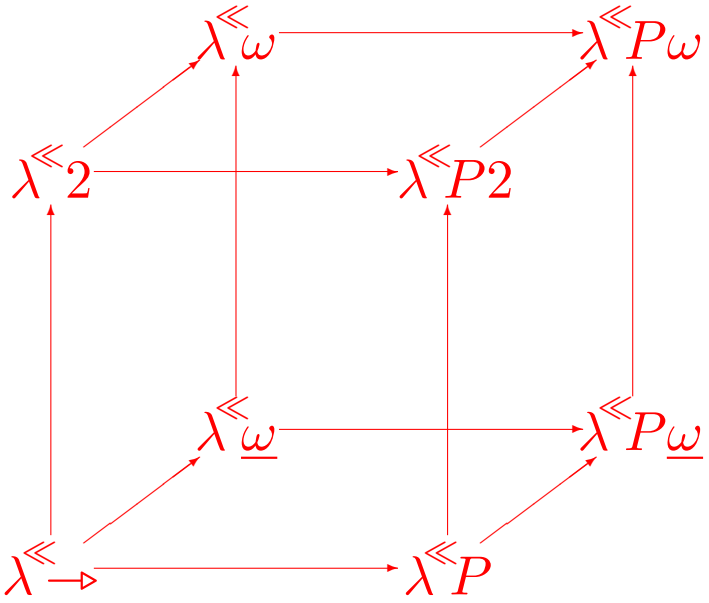
Thanks for youρ attention ...

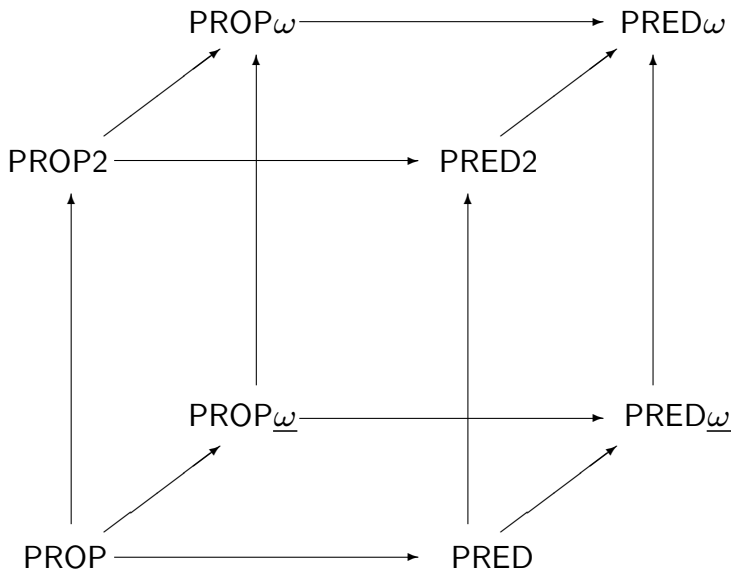




PROP	proposition logic	PRED	predicate logic
PROP2	second-order PROP	PRED2	second-order PRED
PROP$\underline{\omega}$	weakly higher-order PROP	PRED$\underline{\omega}$	weakly higher-order PRED
PROPω	higher-order PROP	PREDω	higher-order PRED

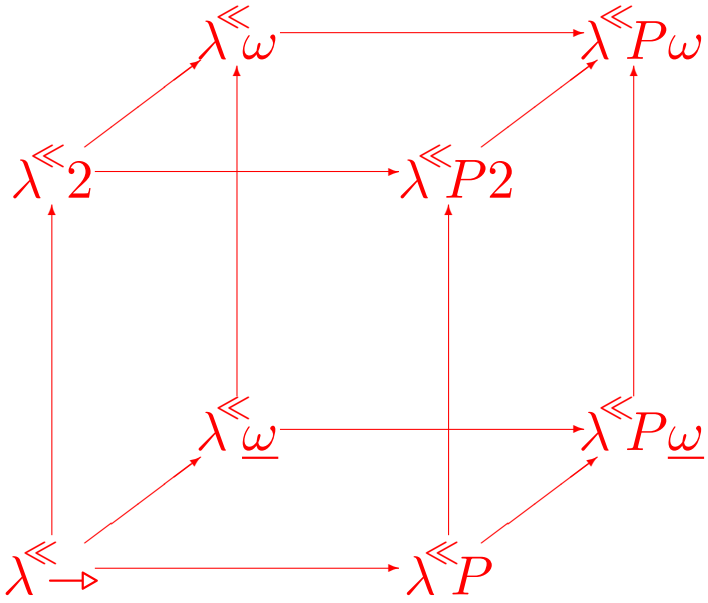
Thanks for youρ attention ...

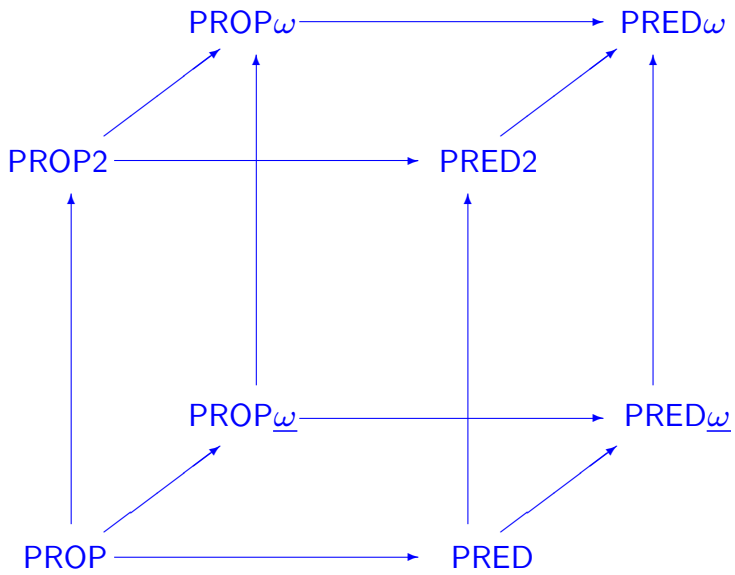




PROP	proposition logic	PRED	predicate logic
PROP2	second-order PROP	PRED2	second-order PRED
PROP$\underline{\omega}$	weakly higher-order PROP	PRED$\underline{\omega}$	weakly higher-order PRED
PROPω	higher-order PROP	PREDω	higher-order PRED

Thanks for youρ attention ...





PROP	proposition logic	PRED	predicate logic
PROP2	second-order PROP	PRED2	second-order PRED
PROP$\underline{\omega}$	weakly higher-order PROP	PRED$\underline{\omega}$	weakly higher-order PRED
PROPω	higher-order PROP	PREDω	higher-order PRED

The Uncle Pat and the Lady Match

+

$= P^2TS$

Imperative Rewriting Calculus

PPDP-04

iRho

Motivations

Multiparadigm

- *Rewriting-based* languages (like, $e.g.$ ELAN, Maude) try to unify the logic paradigm with the functional paradigm.

Multiparadigm

- *Rewriting-based* languages (like, $e.g.$ ELAN, Maude) try to unify the logic paradigm with the functional paradigm.

- Although these languages are less used in common practice than $e.g.$ object-oriented languages (like, $e.g.$, Java, C#, O'Caml, . . .), they can be used also as (formal) common intermediate languages for implementing compilers for rewriting-based, functional, object-oriented, logic, and others "high-level" modern languages and meta languages.

Main *Atout*: *Pattern Matching*.

- *Pattern-matching* allows to discriminate between alternatives. Once a pattern is recognized, a pattern is associated to an action. The corresponding pattern is thus rewritten in an appropriate instance of a new one

- Ability to handle a collection of results: pattern matching need not to be exclusive, $i.e.$ multiple branches can be "fired" simultaneously

- An empty collection of results represents an application failure, a singleton represents a deterministic result, and a collection with more than one element represents a non-deterministic choice between the elements of the collection

Main *Atout*: *Pattern Matching*.

- *Pattern-matching* allows to discriminate between alternatives. Once a pattern is recognized, a pattern is associated to an action. The corresponding pattern is thus rewritten in an appropriate instance of a new one
- Ability to handle a collection of results: pattern matching need not to be exclusive, $i.e.$ multiple branches can be "fired" simultaneously
- An empty collection of results represents an application failure, a singleton represents a deterministic result, and a collection with more than one element represents a non-deterministic choice between the elements of the collection
- Applications: pattern recognition, strings or trees manipulation, etc
- Pattern-matching present in ML, Haskell, Scheme, or Prolog; considered a convenient mechanism for expressing complex requirements about the function's argument, rather than a basis for an $ad\ hoc$ paradigm of computation

Rho's Goodies

- One of the main features of the Rewriting-calculus is its capacity to deal with (de)structuring structures like $e.g.$ lists: we record only the names of the constructor and we discard those of the accessors

- Since structures are built-in the calculus, it follows that the encoding of constructor/accessors is simpler $w.r.t.$ the standard encoding in the Lambda-calculus. The table below (informally) compares the (untyped) encoding of accessors in both formalisms

ops/form	Rewriting-calculus	Lambda-calculus
cons	$X \twoheadrightarrow Y \twoheadrightarrow (\text{cons } X\ Y)$	$\lambda XYZ.\ ZXY$
car	$(\text{cons } X\ Y) \twoheadrightarrow X$	$\lambda Z.\ Z(\lambda XY.X)$
cdr	$(\text{cons } X\ Y) \twoheadrightarrow Y$	$\lambda Z.\ Z(\lambda XY.Y)$

iRho *vs.* Programming

- iRho is an extension of Rho with references, memory allocation, and assignment. It features, all the "idiosyncrasies" of functional/rewriting-based languages with imperative features and modern pattern matching facilities

iRho *vs.* Programming

- iRho is an extension of Rho with references, memory allocation, and assignment. It features, all the "idiosyncrasies" of functional/rewriting-based languages with imperative features and modern pattern matching facilities

- The controlled and conscious use of references, gives to the user the programming comfort and a good expressiveness which we could not a priori expect from a so simple calculus

iRho *vs.* Programming

- iRho is an extension of Rho with references, memory allocation, and assignment. It features, all the "idiosyncrasies" of functional/rewriting-based languages with imperative features and modern pattern matching facilities

- The controlled and conscious use of references, gives to the user the programming comfort and a good expressiveness which we could not a priori expect from a so simple calculus

- The "magic ingredients" of iRho are the combination of modern and safe imperative features (full control over internal data-structure reprs), and of the "matching power" (full Lisp-like operations, like cons/car/cdr)

iRho *vs.* Programming

- iRho is an extension of Rho with references, memory allocation, and assignment. It features, all the "idiosyncrasies" of functional/rewriting-based languages with imperative features and modern pattern matching facilities

- The controlled and conscious use of references, gives to the user the programming comfort and a good expressiveness which we could not a priori expect from a so simple calculus

- The "magic ingredients" of iRho are the combination of modern and safe imperative features (full control over internal data-structure reprs), and of the "matching power" (full Lisp-like operations, like cons/car/cdr)

- $insumma$, iRho as theoretical engine for an family of $ad\ hoc$ languages combining functions, patterns, objects with semi-structured XML-data (XDUCE, CDUCE, HYDROJ, TOM) (*"...O-O pattern matching focuses on the essential information in a msg and is insensitive to inessential information..."*)

A "Fresh" Approach to Rewriting

- We present an imperative fully typed ($e.g.$ $\grave{a}$ la Church) version of the Rho, a pattern-matching based calculus with side-effects, which we call iRho
- We formulate the static and dynamic semantics of iRho
- A call-by-value deterministic Natural Semantics $\grave{a}$ la Kahn: it immediately suggests how to build an interpreter for the calculus
- The static semantics is given via a first-order type system based on a form of product-types reflecting the (non-commutative) structure of the term

A "Fresh" Approach to Rewriting

- We present an imperative fully typed ($e.g.$ $\grave{a}$ la Church) version of the Rho, a pattern-matching based calculus with side-effects, which we call iRho
- We formulate the static and dynamic semantics of iRho
- A call-by-value deterministic Natural Semantics $\grave{a}$ la Kahn: it immediately suggests how to build an interpreter for the calculus
- The static semantics is given via a first-order type system based on a form of product-types reflecting the (non-commutative) structure of the term
- Access and modify a (monomorphic) typed store, and define fixpoints, control structures, ...
- iRho enjoys determinism of the interpreter, subject reduction, and decidability of type-checking (completely checked by a machine assisted approach, using the Coq proof assistant). Progress and decidability of type-checking are proved by pen and paper

Special Emphasis (iRho)

- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor

Special Emphasis (iRho)

- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor
  - ⋆ a "sound" machine (the interpreter)
  - ⋆ a "sound" type system (the type checker), such that
    "well typed programs do not go wrong" [Milner],
  - ⋆ both semantics being suitable to be specified with nice mathematics,
  - ⋆ to be implemented with high-level programming languages,
  - ⋆ and to be certified with modern and semi-automatic theorem provers, like Coq

Special Emphasis (iRho)

- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor
  - ⋆ a "sound" machine (the interpreter)
  - ⋆ a "sound" type system (the type checker), such that
    "well typed programs do not go wrong" [Milner],
  - ⋆ both semantics being suitable to be specified with nice mathematics,
  - ⋆ to be implemented with high-level programming languages,
  - ⋆ and to be certified with modern and semi-automatic theorem provers, like Coq

- Thus, we have encoded in Coq the static and dynamic semantics of iRho

Special Emphasis (iRho)

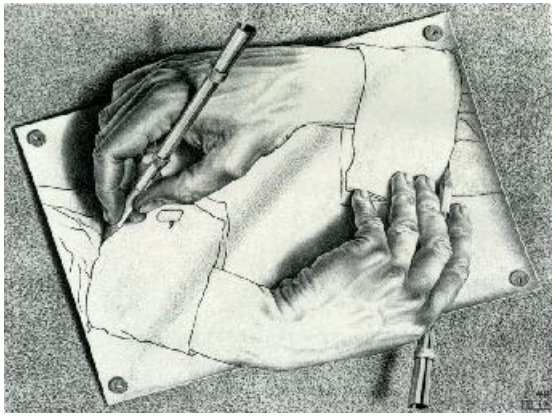
- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor
  - ⋆ a "sound" machine (the interpreter)
  - ⋆ a "sound" type system (the type checker), such that
    "well typed programs do not go wrong" [Milner],
  - ⋆ both semantics being suitable to be specified with nice mathematics,
  - ⋆ to be implemented with high-level programming languages,
  - ⋆ and to be certified with modern and semi-automatic theorem provers, like Coq

- Thus, we have encoded in Coq the static and dynamic semantics of iRho
- All subtle aspects, (usually "swept under the rug") on the paper, are here enlightened by the rigid discipline imposed by the Logical Framework of Coq

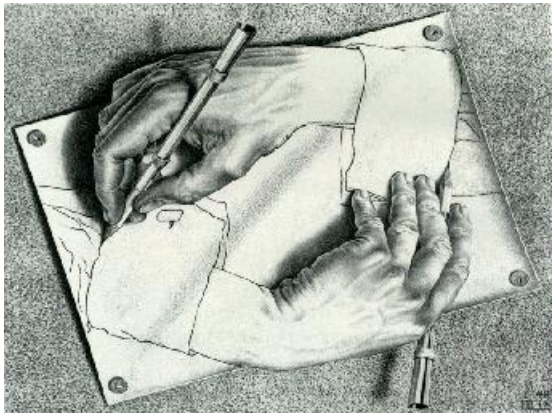
- Often, this process may had a bearing on the design of the static and dynamic semantics

- Often, this process may had a bearing on the design of the static and dynamic semantics
- This (positive) continuous cycle  between mathematics,  manual (*i.e.* pen-and-paper) and  mechanical proofs, and  "toy" implementations using high-level languages such Scheme (and back) has been fruitful since the very beginning of our project

- Often, this process may had a bearing on the design of the static and dynamic semantics
- This (positive) continuous cycle ↻ ↻ between mathematics, ↻ ↻ manual ($i.e.$ pen-and-paper) and ↻ ↻ mechanical proofs, and ↻ ↻ "toy" implementations using high-level languages such Scheme (and back) has been fruitful since the very beginning of our project



- Often, this process may had a bearing on the design of the static and dynamic semantics
- This (positive) continuous cycle ↻ ↻ between mathematics, ↻ ↻ manual ($i.e.$ pen-and-paper) and ↻ ↻ mechanical proofs, and ↻ ↻ "toy" implementations using high-level languages such Scheme (and back) has been fruitful since the very beginning of our project



- Although our calculus is rather simple, it is not impossible, in a near future, to scale-up to larger projects, such as the certified implementation of compilers for a "real" programming language of the C family (Cminor $\subset$ C)
(Action Recherche Coordonnée INRIA, Concert)

iRho

The Syntax

First step: Functional fRho

The symbol τ ranges over types, Γ, Δ range over contexts, P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$), and $A, B, \ldots$ range over terms

$$\tau ::= \mathsf{b} \mid \tau \twoheadrightarrow \tau \mid \tau \wedge \tau \qquad Types$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad Contexts$$

$$P ::= f \mid X \mid f\ \overline{P} \mid P, P \qquad Patterns$$

$$A ::= f \mid X \mid P \twoheadrightarrow_{\Delta} A \mid A\ A \mid A, A \qquad Terms$$

First step: Functional fRho

The symbol τ ranges over types, Γ, Δ range over contexts, P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$), and $A, B, \ldots$ range over terms

$$\tau ::= \mathsf{b} \mid \tau \twoheadrightarrow \tau \mid \tau \wedge \tau \qquad Types$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad Contexts$$

$$P ::= f \mid X \mid f\ \overline{P} \mid P, P \qquad Patterns$$

$$A ::= f \mid X \mid P \twoheadrightarrow_{\Delta} A \mid A\ A \mid A, A \qquad Terms$$

First step: Functional fRho

The symbol τ ranges over types, Γ, Δ range over contexts, P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$), and $A, B, \ldots$ range over terms

$$\tau ::= \mathsf{b} \mid \tau \twoheadrightarrow \tau \mid \tau \wedge \tau \qquad Types$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad Contexts$$

$$P ::= f \mid X \mid f\ \overline{P} \mid P, P \qquad Patterns$$

$$A ::= f \mid X \mid P \twoheadrightarrow_{\Delta} A \mid A\ A \mid A, A \qquad Terms$$

First step: Functional fRho

The symbol τ ranges over types, Γ, Δ range over contexts, P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$), and $A, B, \ldots$ range over terms

$$\tau ::= \mathsf{b} \mid \tau \twoheadrightarrow \tau \mid \tau \wedge \tau \qquad Types$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad Contexts$$

$$P ::= f \mid X \mid f\ \overline{P} \mid P, P \qquad Patterns$$

$$A ::= f \mid X \mid P \twoheadrightarrow_{\Delta} A \mid A\ A \mid A, A \qquad Terms$$

First step: Functional fRho

The symbol τ ranges over types, Γ, Δ range over contexts, P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$), and $A, B, \ldots$ range over terms

$$\tau ::= \mathsf{b} \mid \tau \twoheadrightarrow \tau \mid \tau \wedge \tau \qquad \textit{Types}$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad \textit{Contexts}$$

$$P ::= f \mid X \mid f\ \overline{P} \mid P, P \qquad \textit{Patterns}$$

$$A ::= f \mid X \mid P \twoheadrightarrow_{\Delta} A \mid A\ A \mid A, A \qquad \textit{Terms}$$

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

$\tau ::= \ldots$ as in fRho $\ldots \mid \tau$ ref $\quad Types$

$\Delta ::= \ldots$ as in fRho $\ldots \quad Contexts$

$P ::= \ldots$ as in fRho $\ldots \mid$ ref $P \quad Patterns$

$A ::= \ldots$ as in fRho $\ldots \mid$ ref $A \mid !A \mid A := A \quad Terms$

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

τ ::= . . . as in fRho . . . | τ ref $\quad Types$

Δ ::= . . . as in fRho . . . $\quad Contexts$

P ::= . . . as in fRho . . . | ref P $\quad Patterns$

A ::= . . . as in fRho . . . | ref A | $!A$ | $A := A$ $\quad Terms$

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

τ ::= . . . as in fRho . . . | τ ref — *Types*

Δ ::= . . . as in fRho . . . — *Contexts*

P ::= . . . as in fRho . . . | ref P — *Patterns*

A ::= . . . as in fRho . . . | ref A | !A | $A := A$ — *Terms*

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

τ ::= . . . as in fRho . . . | τ ref — *Types*

Δ ::= . . . as in fRho . . . — *Contexts*

P ::= . . . as in fRho . . . | ref P — *Patterns*

A ::= . . . as in fRho . . . | ref A | !A | $A := A$ — *Terms*

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, $(\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T})$.

τ ::= . . . as in fRho . . . | τ ref — *Types*

Δ ::= . . . as in fRho . . . — *Contexts*

P ::= . . . as in fRho . . . | ref P — *Patterns*

A ::= . . . as in fRho . . . | ref A | $!A$ | $A := A$ — *Terms*

iRho in a glance

Intuitively iRho deals with references $\grave{a}\ la$ Caml $i.e.$:

- **(Deref-types)** The type τ ref is the type of "refs" containing a value of type τ
- **(Deref-op)** The operator ! is a "dereferencing" operator (goto memory)
- **(Ref-op)** The operator ref is a "referencing" operator (`malloc`)
- **(Assign)** The term $A := B$ is an "assignment" operator, which returns as result the value obtained by evaluating B.

We have not (yet) modeled garbage collection: new locations created during reduction will remain in the store forever

Simple, Naïve GC (by Talcott, Mason, Morrissett, *et al.*)

```
for i < n
 letrec x_1 = ref V_1
    and ...
    and x_n = ref V_n
    in A
 -->
 letrec x_1 = ref V_1
    and ...
    and x_i = ref V_i
    in A
 if FV(V_1, ..., V_i)/\ {x_(i+1), ..., x_n} = 0
```

Let-like and conditionals (in a call-by value setting)

As usual, let-like constructs can be generalized with pattern and becomes simple syntactic sugar for applications (types are omitted), $i.e.$

$$\mathsf{let}\ \ P \ll A\ \ \mathsf{in}\ \ B \ \ \triangleq \ \ (P \twoheadrightarrow B)\ A$$

Conditional too can be easily encoded using pair, applications, and constants, $i.e.$

$$\Longleftrightarrow \quad A\ \ \mathsf{then}\, B\ \ \mathsf{else}\ \ C \ \ \triangleq \ \ (\mathsf{true} \twoheadrightarrow B, \mathsf{false} \twoheadrightarrow C)\ A$$

and

$$\mathsf{neg} \ \ \triangleq \ \ (\mathsf{true} \twoheadrightarrow \mathsf{false}, \mathsf{false} \twoheadrightarrow \mathsf{true})$$

Values and Environments in fRho

- We introduce the set $\mathcal{V}al$ of values and the set of environments $\mathcal{E}nv$

$$A_{\mathsf{v}} \ ::= \ f \mid f\ \overline{A}_{\mathsf{v}} \mid A_{\mathsf{v}}, A_{\mathsf{v}} \mid \underbrace{\langle P \twoheadrightarrow_{\Delta} A \cdot \rho \rangle}_{\text{fun closures}} \mid \underbrace{\langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle}_{\text{fail closures}}$$

- Environments are partial functions from the set of variables to the set of values

$$\rho[X \mapsto A_{\mathsf{v}}](Y) \ \triangleq \ \begin{cases} A_{\mathsf{v}} & \text{if } X \equiv Y \\ \rho(Y) & \text{otherwise} \end{cases}$$

- "Failure-values" $\langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle$ denote failure occurring when we cannot found a correct substitution θ on the free variables of P such that $\theta(P) \equiv A_{\mathsf{v}}$; Failure-values are obtained during the computation when a matching failure occurs. It could (in principle) be caught by a suitable exception handler

Values and Environments in fRho

- We introduce the set $\mathcal{V}al$ of values and the set of environments $\mathcal{E}nv$

$$A_{\mathsf{v}} \ ::= \ f \mid f\ \overline{A}_{\mathsf{v}} \mid A_{\mathsf{v}}, A_{\mathsf{v}} \mid \underbrace{\langle P \twoheadrightarrow_{\Delta} A \cdot \rho \rangle}_{\text{fun closures}} \mid \underbrace{\langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle}_{\text{fail closures}}$$

- Environments are partial functions from the set of variables to the set of values

$$\rho[X \mapsto A_{\mathsf{v}}](Y) \ \triangleq \ \begin{cases} A_{\mathsf{v}} & \text{if } X \equiv Y \\ \rho(Y) & \text{otherwise} \end{cases}$$

- "Failure-values" $\langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle$ denote failure occurring when we cannot found a correct substitution θ on the free variables of P such that $\theta(P) \equiv A_{\mathsf{v}}$; Failure-values are obtained during the computation when a matching failure occurs. It could (in principle) be caught by a suitable exception handler

Values and Environments in fRho

- We introduce the set $\mathcal{V}al$ of values and the set of environments $\mathcal{E}nv$

$$A_{\mathsf{v}} \ ::= \ f \mid f\ \overline{A}_{\mathsf{v}} \mid A_{\mathsf{v}}, A_{\mathsf{v}} \mid \underbrace{\langle P \twoheadrightarrow_{\Delta} A \cdot \rho \rangle}_{\text{fun closures}} \mid \underbrace{\langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle}_{\text{fail closures}}$$

- Environments are partial functions from the set of variables to the set of values

$$\rho[X \mapsto A_{\mathsf{v}}](Y) \ \triangleq \ \begin{cases} A_{\mathsf{v}} & \text{if } X \equiv Y \\ \rho(Y) & \text{otherwise} \end{cases}$$

- "Failure-values" $\langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle$ denote failure occurring when we cannot found a correct substitution θ on the free variables of P such that $\theta(P) \equiv A_{\mathsf{v}}$; Failure-values are obtained during the computation when a matching failure occurs. It could (in principle) be caught by a suitable exception handler

Values and Stores in iRho

- The new set of values is enriched by locations. Moreover we define the set of global stores $\mathcal{S}tore$ (the symbol σ ranges over stores)

$$A_{\mathsf{v}} \ ::= \ \ldots \text{as in fRho} \ldots \mid \underbrace{\iota}_{\text{locations}} \quad \text{Imperative Values}$$

- Stores are partial functions from the set $\mathcal{L}$ of locations to the set of values

$$\sigma[\iota_1 \mapsto A_{\mathsf{v}}](\iota_2) \ \triangleq \ \begin{cases} A_{\mathsf{v}} & \text{if } \iota_1 \equiv \iota_2 \\ \sigma(\iota_2) & \text{otherwise} \end{cases}$$

Values and Stores in iRho

- The new set of values is enriched by locations. Moreover we define the set of global stores $\mathcal{S}tore$ (the symbol σ ranges over stores)

$$A_{\mathsf{v}} \ ::= \ \ldots \text{as in fRho} \ldots \mid \underbrace{\iota}_{\text{locations}} \quad \text{Imperative Values}$$

- Stores are partial functions from the set $\mathcal{L}$ of locations to the set of values

$$\sigma[\iota_1 \mapsto A_{\mathsf{v}}](\iota_2) \ \triangleq \ \begin{cases} A_{\mathsf{v}} & \text{if } \iota_1 \equiv \iota_2 \\ \sigma(\iota_2) & \text{otherwise} \end{cases}$$

Values and Stores in iRho

- The new set of values is enriched by locations. Moreover we define the set of global stores $\mathcal{S}tore$ (the symbol σ ranges over stores)

$$A_{\mathsf{v}} \ ::= \ \ldots \text{as in fRho} \ldots \mid \underbrace{\iota}_{\text{locations}} \quad \text{Imperative Values}$$

- Stores are partial functions from the set $\mathcal{L}$ of locations to the set of values

$$\sigma[\iota_1 \mapsto A_{\mathsf{v}}](\iota_2) \ \triangleq \ \begin{cases} A_{\mathsf{v}} & \text{if } \iota_1 \equiv \iota_2 \\ \sigma(\iota_2) & \text{otherwise} \end{cases}$$

Natural Semantics for fRho

- We define a call-by-value optimistic operational semantics via a natural proof deduction system $\grave{a}\ la$ Gilles Kahn (induces quasi directly an "interpreter machine")

- The purpose of the deduction system is to map every expression into a normal form, $i.e.$ an irreducible term in weak head normal form. The present strategy is call-by-value since it does not work under plain abstractions ($i.e.\ P \twoheadrightarrow_{\Delta} A$)

- The present interpreter machine is optimistic since it gives a result if at least one computation does not produce a failure-value: of course other choices are possible, like $e.g.$ a "pessimistic" machine which stops if at least one failure-value occurs

Natural Semantics (NS) in a Nutshell

- Variant of Plotkin's *Structured Operational Semantics* (SOS)
- The semantics induces quasi directly an Interpreter. Judgments are:

$$\frac{\texttt{premises}_i \quad i \geq 0}{\texttt{store} \cdot \texttt{env} \vdash \texttt{expr} \Downarrow_{\mathsf{val}} \texttt{value} \cdot \texttt{store}'} \,(\text{expr}\cdot\text{rule})$$

$$\frac{\texttt{premises}_i \quad i \geq 0}{\texttt{store} \vdash \langle A_{\mathsf{v}} \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} \texttt{value} \cdot \texttt{store}'} \,(\text{appl}\cdot\text{case})$$

$$\frac{\texttt{premises}_i \quad i \geq 0}{\texttt{store} \cdot \texttt{env} \vdash \langle A \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \texttt{env}'} \,(\text{patt}\cdot\text{case})$$

Optimistic- *vs.* Pessimistic-machines

Our machine is optimistic, $i.e.$ it does not abort computations if a matching failure occurs, recording that one computation goes wrong, and hoping that at least one computation succeeds. The choice of killing the computation once a failure-value is produced leads to a pessimistic machine

$$\frac{\cdots}{\cdots \vdash (3 \twoheadrightarrow 3, 4 \twoheadrightarrow 4)\ 4 \Downarrow_{\mathsf{val}} \underbrace{\langle [3 \ll 4].3 \cdot \emptyset \rangle, 4}_{\text{fail+4}} \cdots}\ (\text{Opt})$$

$$\frac{\cdots}{\cdots \vdash (3 \twoheadrightarrow 3, 4 \twoheadrightarrow 4)\ 4 \Downarrow_{\mathsf{val}} \underbrace{\langle [3 \ll 4].3 \cdot \emptyset \rangle}_{\text{just fail}} \cdots}\ (\text{Pex})$$

A pessimistic-machine can be quite easily produced "trucking" an optimistic one by propagating failure-values

Just try before I tell you the full story ...

- Take the imperative term (types omitted)

$$A \triangleq (\underbrace{f(X,Y)}_{\text{pattern}} \twoheadrightarrow (3 \twoheadrightarrow (X := !Y))\ !X)\ \underbrace{f(\mathsf{ref}\ 3, \mathsf{ref}\ 4)}_{\text{argument}}$$

- We evaluate this term in the empty store $\emptyset$ and the empty environment $\emptyset$

- The result would be?

$$\emptyset \cdot \emptyset \ \vdash\ A \Downarrow_{\mathsf{val}}$$

Just try before I tell you the full story ...

- Take the imperative term (types omitted)

$$A \triangleq (\underbrace{f(X,Y)}_{\text{pattern}} \twoheadrightarrow (3 \twoheadrightarrow (X := !Y))\ !X)\ \underbrace{f(\mathsf{ref}\ 3, \mathsf{ref}\ 4)}_{\text{argument}}$$

- We evaluate this term in the empty store $\emptyset$ and the empty environment $\emptyset$

- The result would be?

$$\emptyset \cdot \emptyset \ \vdash\ A \Downarrow_{\mathsf{val}}$$

Just try before I tell you the full story ...

- Take the imperative term (types omitted)

$$A \triangleq (\underbrace{f(X,Y)}_{\text{pattern}} \twoheadrightarrow (3 \twoheadrightarrow (X := !Y))\ !X)\ \underbrace{f(\mathsf{ref}\ 3, \mathsf{ref}\ 4)}_{\text{argument}}$$

- We evaluate this term in the empty store $\emptyset$ and the empty environment $\emptyset$

- The result would be?

$$\emptyset \cdot \emptyset \ \vdash\ A \Downarrow_{\mathsf{val}} 4 \cdot$$

Just try before I tell you the full story ...

- Take the imperative term (types omitted)

$$A \triangleq (\underbrace{f(X,Y)}_{\text{pattern}} \twoheadrightarrow (3 \twoheadrightarrow (X := !Y))\ !X)\ \underbrace{f(\mathsf{ref}\ 3, \mathsf{ref}\ 4)}_{\text{argument}}$$

- We evaluate this term in the empty store $\emptyset$ and the empty environment $\emptyset$

- The result would be?

$$\emptyset \cdot \emptyset \ \vdash\ A \Downarrow_{\mathsf{val}} 4 \cdot [\iota_0 \mapsto 4,$$

Just try before I tell you the full story ...

- Take the imperative term (types omitted)

$$A \triangleq (\underbrace{f(X,Y)}_{\text{pattern}} \twoheadrightarrow (3 \twoheadrightarrow (X := !Y))\ !X)\ \underbrace{f(\mathsf{ref}\ 3, \mathsf{ref}\ 4)}_{\text{argument}}$$

- We evaluate this term in the empty store $\emptyset$ and the empty environment $\emptyset$

- The result would be?

$$\emptyset \cdot \emptyset \ \vdash\ A \Downarrow_{\mathsf{val}} 4 \cdot [\iota_0 \mapsto 4, \iota_1 \mapsto 4]$$

iRho

Natural Semantics

Natural Judgments in iRho

- The semantics is given in terms of three judgments

$$\sigma \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma'$$

$$\sigma \vdash \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} C_{\mathsf{v}} \cdot \sigma'$$

$$\sigma \cdot \rho \vdash \langle A \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'$$

- The first judgment evaluates a term in iRho, the second apply one value to another producing a result value and a new store, and the last judgment updates a correct environment obtained by matching a term against a value

iRho:

$\Downarrow_{\mathsf{val}}$: Application and Structures

$$\frac{\begin{array}{llll} \sigma_0 \cdot \rho & \vdash & A & \Downarrow_{\mathsf{val}} \; A_{\mathsf{v}} \cdot \sigma_1 \\ \sigma_1 \cdot \rho & \vdash & B & \Downarrow_{\mathsf{val}} \; B_{\mathsf{v}} \cdot \sigma_2 \\ \sigma_2 & \vdash & \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle & \Downarrow_{\mathsf{call}} \; C_{\mathsf{v}} \cdot \sigma_3 \end{array}}{\sigma_0 \cdot \rho \vdash A \; B \Downarrow_{\mathsf{val}} C_{\mathsf{v}} \cdot \sigma_3} \; (\mathrm{Red}{-}\rho_{\mathsf{v}})$$

$\Downarrow_{\mathsf{val}}$: Application and Structures

$$\frac{\begin{array}{llll}\sigma_0 \cdot \rho & \vdash & A & \Downarrow_{\mathsf{val}} \; A_{\mathsf{v}} \cdot \sigma_1 \\ \sigma_1 \cdot \rho & \vdash & B & \Downarrow_{\mathsf{val}} \; B_{\mathsf{v}} \cdot \sigma_2 \\ \sigma_2 & \vdash & \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle & \Downarrow_{\mathsf{call}} \; C_{\mathsf{v}} \cdot \sigma_3\end{array}}{\sigma_0 \cdot \rho \vdash A \; B \Downarrow_{\mathsf{val}} C_{\mathsf{v}} \cdot \sigma_3} \; (\mathrm{Red}{-}\rho_{\mathsf{v}})$$

$$\frac{\begin{array}{c}\sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1 \\ \sigma_1 \cdot \rho \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2\end{array}}{\sigma_0 \cdot \rho \vdash A, B \Downarrow_{\mathsf{val}} A_{\mathsf{v}}, B_{\mathsf{v}} \cdot \sigma_2} \; (\mathrm{Red}{-}\mathrm{Struct})$$

$\Downarrow_{\mathsf{val}}$: Functions, Values, Variables

$$\frac{}{\sigma \cdot \rho \vdash P \twoheadrightarrow_{\Delta} A \Downarrow_{\mathsf{val}} \langle P \twoheadrightarrow_{\Delta} A \cdot \rho \rangle \cdot \sigma} \, (\mathrm{Red{-}Fun})$$

$\Downarrow_{\mathsf{val}}$: Functions, Values, Variables

$$\frac{}{\sigma \cdot \rho \vdash P \twoheadrightarrow_{\Delta} A \Downarrow_{\mathsf{val}} \langle P \twoheadrightarrow_{\Delta} A \cdot \rho \rangle \cdot \sigma} \, (\mathrm{Red{-}Fun})$$

$$\frac{}{\sigma \cdot \rho \vdash f \Downarrow_{\mathsf{val}} f \cdot \sigma} \, (\mathrm{Red{-}v})$$

$\Downarrow_{\mathsf{val}}$: Functions, Values, Variables

$$\frac{}{\sigma \cdot \rho \vdash P \twoheadrightarrow_{\Delta} A \Downarrow_{\mathsf{val}} \langle P \twoheadrightarrow_{\Delta} A \cdot \rho \rangle \cdot \sigma} \,(\mathrm{Red{-}Fun})$$

$$\frac{}{\sigma \cdot \rho \vdash f \Downarrow_{\mathsf{val}} f \cdot \sigma} \,(\mathrm{Red{-}v})$$

$$\frac{X \in \mathcal{D}om(\rho)}{\sigma \cdot \rho \vdash X \Downarrow_{\mathsf{val}} \rho(X) \cdot \sigma} \,(\mathrm{Red{-}Var})$$

$\Downarrow_{\mathsf{val}}$: Referencing, Dereferencing, Assignment

$$\frac{\sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1 \qquad \iota \notin \mathcal{D}om(\sigma_1)}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\ A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]}\ (\mathrm{Red}-\mathsf{ref}\ )$$

$\Downarrow_{\mathsf{val}}$: Referencing, Dereferencing, Assignment

$$\frac{\sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1 \qquad \iota \notin \mathcal{D}om(\sigma_1)}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\ A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]}\ (\mathrm{Red}{-}\mathsf{ref}\ )$$

$$\frac{\sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \iota \in \mathcal{D}om(\sigma_1)}{\sigma_0 \cdot \rho \vdash\ !A \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1}\ (\mathrm{Red}{-}!)$$

$\Downarrow_{\mathsf{val}}$: Referencing, Dereferencing, Assignment

$$\frac{\sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1 \qquad \iota \notin \mathcal{D}om(\sigma_1)}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\ A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]}\ (\mathrm{Red}{-}\mathsf{ref}\ )$$

$$\frac{\sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \iota \in \mathcal{D}om(\sigma_1)}{\sigma_0 \cdot \rho \vdash\ !A \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1}\ (\mathrm{Red}{-}!)$$

$$\frac{\begin{array}{l}\iota \in \mathcal{D}om(\sigma_1) \\ \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2\end{array}}{\sigma_0 \cdot \rho \vdash A := B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto B_{\mathsf{v}}]}\ (\mathrm{Red}{-}:=)$$

iRho:

$\Downarrow_{\mathsf{call}}$: FunOk, FunKO

$$\frac{\begin{array}{c}\sigma_0 \cdot \rho_0 \vdash \langle P \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_1 \\ \sigma_0 \cdot \rho_1 \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1\end{array}}{\sigma_0 \vdash \langle \langle P{:}\Delta \twoheadrightarrow A \cdot \rho \rangle \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} A_{\mathsf{v}} \cdot \sigma_1} \text{(Call}-\text{FunOk)}$$

$\Downarrow_{\mathsf{call}}$: FunOk, FunKO

$$\frac{\begin{array}{c}\sigma_0 \cdot \rho_0 \vdash \langle P \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_1 \\ \sigma_0 \cdot \rho_1 \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1\end{array}}{\sigma_0 \vdash \langle \langle P{:}\Delta \twoheadrightarrow A \cdot \rho \rangle \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} A_{\mathsf{v}} \cdot \sigma_1} \text{(Call−FunOk)}$$

$$\frac{\begin{array}{c}\nexists \rho_1.\ \sigma \cdot \rho_0 \vdash \langle P \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_1 \\ A_{\mathsf{v}} \equiv \langle P{:}\Delta \twoheadrightarrow A \cdot \rho \rangle\end{array}}{\sigma \vdash \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} A_{\mathsf{v}} \cdot \sigma} \text{(Call−FunKo)}$$

$\Downarrow_{\mathsf{call}}$: Structure, Algebric, and Wrong

$$\frac{\begin{array}{c}\sigma_0 \vdash \langle A_{\mathsf{v}} \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} D_{\mathsf{v}} \cdot \sigma_1 \\ \sigma_1 \vdash \langle B_{\mathsf{v}} \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} E_{\mathsf{v}} \cdot \sigma_2\end{array}}{\sigma_0 \vdash \langle (A_{\mathsf{v}}, B_{\mathsf{v}}) \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} D_{\mathsf{v}}, E_{\mathsf{v}} \cdot \sigma_2} \text{(Call}-\text{Struct)}$$

$\Downarrow_{\mathsf{call}}$: Structure, Algebric, and Wrong

$$\frac{\begin{array}{c}\sigma_0 \vdash \langle A_{\mathsf{v}} \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} D_{\mathsf{v}} \cdot \sigma_1 \\ \sigma_1 \vdash \langle B_{\mathsf{v}} \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} E_{\mathsf{v}} \cdot \sigma_2\end{array}}{\sigma_0 \vdash \langle (A_{\mathsf{v}}, B_{\mathsf{v}}) \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} D_{\mathsf{v}}, E_{\mathsf{v}} \cdot \sigma_2}\,(\mathrm{Call{-}Struct})$$

$$\frac{}{\sigma \vdash \langle f\ \overline{A}_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} f\ \overline{A}_{\mathsf{v}}\ B_{\mathsf{v}} \cdot \sigma}\,(\mathrm{Call{-}Algbr})$$

$\Downarrow_{\mathsf{call}}$: Structure, Algebric, and Wrong

$$\frac{\begin{array}{c}\sigma_0 \vdash \langle A_{\mathsf{v}} \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} D_{\mathsf{v}} \cdot \sigma_1 \\ \sigma_1 \vdash \langle B_{\mathsf{v}} \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} E_{\mathsf{v}} \cdot \sigma_2\end{array}}{\sigma_0 \vdash \langle (A_{\mathsf{v}}, B_{\mathsf{v}}) \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} D_{\mathsf{v}}, E_{\mathsf{v}} \cdot \sigma_2} \text{(Call}-\text{Struct)}$$

$$\frac{}{\sigma \vdash \langle f\ \overline{A}_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} f\ \overline{A}_{\mathsf{v}}\ B_{\mathsf{v}} \cdot \sigma} \text{(Call}-\text{Algbr)}$$

$$\frac{A_{\mathsf{v}} \equiv \langle [P \ll_{\Delta} B_{\mathsf{v}}].A \cdot \rho \rangle}{\sigma \vdash \langle A_{\mathsf{v}} \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} A_{\mathsf{v}} \cdot \sigma} \text{(Call}-\text{Wrong)}$$

iRho:

$\Downarrow_{\mathsf{match}}$: Variables and Constants

$$\frac{}{\sigma \cdot \rho \vdash \langle f \cdot f \rangle \Downarrow_{\mathsf{match}} \rho}\,(\mathrm{Match{-}Const})$$

$\Downarrow_{\mathsf{match}}$: Variables and Constants

$$\frac{}{\sigma \cdot \rho \vdash \langle f \cdot f \rangle \Downarrow_{\mathsf{match}} \rho} \,(\mathrm{Match{-}Const})$$

$$\frac{}{\sigma \cdot \rho \vdash \langle X \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho[X \mapsto A_{\mathsf{v}}]} \,(\mathrm{Match{-}Var})$$

$\Downarrow_{\mathsf{match}}$: Structures and References

$$\frac{\begin{array}{c}\mathsf{Let}\ \star \in \{\bullet\ ,\} \\ \sigma \cdot \rho_0 \vdash \langle A \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_1 \\ \sigma \cdot \rho_1 \vdash \langle B \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_2\end{array}}{\sigma \cdot \rho_0 \vdash \langle A \star B \cdot A_{\mathsf{v}} \star B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_2}\ (\mathrm{Match{-}Pair})$$

$\Downarrow_{\mathsf{match}}$: Structures and References

$$\frac{\begin{array}{c}\mathsf{Let}\ \star \in \{\bullet\ ,\} \\ \sigma \cdot \rho_0 \vdash \langle A \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_1 \\ \sigma \cdot \rho_1 \vdash \langle B \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_2\end{array}}{\sigma \cdot \rho_0 \vdash \langle A \star B \cdot A_{\mathsf{v}} \star B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_2}\ (\mathrm{Match{-}Pair})$$

$$\frac{\begin{array}{c}\iota \in \mathcal{D}om(\sigma) \qquad \sigma(\iota) \equiv A_{\mathsf{v}} \\ \sigma \cdot \rho_0 \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_1\end{array}}{\sigma \cdot \rho_0 \vdash \langle \mathsf{ref}\ P \cdot \iota \rangle \Downarrow_{\mathsf{match}} \rho_1}\ (\mathrm{Match{-}Ref})$$

Remark: About Linearity

This choice induces also a modification in the classical syntactic pattern matching algorithm, since we "hide" the first binding in favor of the second one. The classical syntactic pattern matching algorithm forces both occurrences to be matchable with the same value. Both solutions are presented in the table below:

patt $\ll$ term	hide/Scheme	force/ML
$f(X,X) \ll f(3,4)$	$\rho \triangleq [X \mapsto 3][X \mapsto 4]$	$\nexists\theta$
$f(X,X) \ll f(4,4)$	$\rho \triangleq [X \mapsto 4][X \mapsto 4]$	$\theta \triangleq \{4/X\}$

Linearity can be easily implemented (much bigger effort in the Coq code)

The Uncle Pat and the Lady Match Still at Work...

iRho:

Playing with NS

An Imperative Natural Derivation

Take the imperative term

$$(f(X,Y) \twoheadrightarrow (3 \twoheadrightarrow (X := !Y))\ !X)\ f(\mathsf{ref}\,3, \mathsf{ref}\,4)$$

with $\sigma_0 \stackrel{\triangle}{=} [\iota_0 \mapsto 3][\iota_1 \mapsto 4]$, and $\sigma_1 \stackrel{\triangle}{=} \sigma_0[\iota_0 \mapsto 4]$, and $\rho_0 \stackrel{\triangle}{=} [X \mapsto \iota_0][Y \mapsto \iota_1]$.

Just a Nice LaTeX Exercise

$$\dfrac{\dfrac{\iota_0 \in \mathcal{D}om(\sigma_0) \qquad \dfrac{\iota_1 \in \mathcal{D}om(\sigma_0)}{\sigma_0 \cdot \rho_0 \vdash Y \Downarrow_{\mathsf{val}} \iota_1 \cdot \sigma_0}}{\sigma_0 \cdot \rho_0 \vdash\ !Y \Downarrow_{\mathsf{val}} 4 \cdot \sigma_0}}{\sigma_0 \cdot \rho_0 \vdash X := !Y \Downarrow_{\mathsf{val}} 4 \cdot \sigma_1} \qquad \dfrac{\sigma_0 \cdot \rho_0 \vdash \langle 3 \cdot 3 \rangle \Downarrow_{\mathsf{match}} \rho_0}{\sigma_0 \vdash \langle \langle 3 \to X := !Y \cdot \rho_0 \rangle \cdot 3 \rangle \Downarrow_{\mathsf{call}} 4 \cdot \sigma_1} \qquad \dfrac{\sigma_0 \cdot \rho_0 \vdash !X \Downarrow_{\mathsf{val}} 3 \cdot \sigma_0 \qquad \sigma_0 \cdot \rho_0 \vdash 3 \to X := !Y \Downarrow_{\mathsf{val}} \langle 3 \to X := !Y \cdot \rho_0 \rangle \cdot \sigma_0}{\sigma_0, \rho_0 \vdash (3 \to X := !Y)\ !X \Downarrow_{\mathsf{call}} 4 \cdot \sigma_1} \qquad \dfrac{\sigma_0 \cdot \emptyset \vdash \langle f(X,Y) \cdot f(\iota_0, \iota_1) \rangle \Downarrow_{\mathsf{match}} \rho_0}{\sigma_0 \cdot \emptyset \vdash \langle \langle (f(X,Y) \to (3 \to X := !Y)\ !X) \cdot \emptyset \rangle \cdot f(\iota_0, \iota_1) \rangle \Downarrow_{\mathsf{call}} 4 \cdot \sigma_1} \qquad \dfrac{\emptyset \cdot \emptyset \vdash f(\mathsf{ref}\, 3, \mathsf{ref}\, 4) \Downarrow_{\mathsf{val}} f(\iota_0, \iota_1) \cdot \sigma_0 \qquad \emptyset \cdot \emptyset \vdash (f(X,Y) \to (3 \to X := !Y)\ !X) \Downarrow_{\mathsf{val}} \langle (f(X,Y) \to (3 \to X := !Y)\ !X) \cdot \emptyset \rangle \cdot \emptyset}{\emptyset \cdot \emptyset \vdash (f(X,Y) \to (3 \to X := !Y)\ !X)\ f(\mathsf{ref}\, 3, \mathsf{ref}\, 4) \Downarrow_{\mathsf{val}} 4 \cdot \sigma_1}$$

Aliases

$$\begin{array}{lcl}
\mathsf{let}\ \ P \ll A\ \ \mathsf{in}\ \ B & \triangleq & (P \twoheadrightarrow B)\ A \\
\iff \quad A\ \ \mathsf{then}\, B\ \ \mathsf{else}\ \ C & \triangleq & (\mathsf{true} \twoheadrightarrow B, \mathsf{false} \twoheadrightarrow C)\ A \\
\mathsf{neg} & \triangleq & (\mathsf{true} \twoheadrightarrow \mathsf{false}, \mathsf{false} \twoheadrightarrow \mathsf{true}) \\
A\,;B & \triangleq & (X \twoheadrightarrow B)\ A \quad X \notin \mathcal{FV}ar(B) \\
(X_1, \ldots X_n) := (A_1, \ldots A_n) & \triangleq & X_1 := A_1\,; \ldots\,; X_n := A_n \\
!A & \triangleq & (\mathsf{ref}\ X \twoheadrightarrow X)\ A
\end{array}$$

Aliases

$$\mathsf{let}\ \ P \ll A\ \ \mathsf{in}\ \ B \quad \triangleq \quad (P \twoheadrightarrow B)\ A$$

$$\iff \quad A\ \ \mathsf{then}\, B\ \ \mathsf{else}\ \ C \quad \triangleq \quad (\mathsf{true} \twoheadrightarrow B, \mathsf{false} \twoheadrightarrow C)\ A$$

$$\mathsf{neg} \quad \triangleq \quad (\mathsf{true} \twoheadrightarrow \mathsf{false}, \mathsf{false} \twoheadrightarrow \mathsf{true})$$

$$A\,;B \quad \triangleq \quad (X \twoheadrightarrow B)\ A \quad X \notin \mathcal{FV}ar(B)$$

$$(X_1, \ldots X_n) := (A_1, \ldots A_n) \quad \triangleq \quad X_1 := A_1\,; \ldots\,; X_n := A_n$$

$$!A \quad \triangleq \quad (\mathsf{ref}\ X \twoheadrightarrow X)\ A$$

Very good, dereferencing is just sugar!

iRho:

iExamples

Computing a Negation Normal Form

This function is used in implementing *decision procedures*, present in almost all model checkers. The processed input is a implication-free languages of formulas with generating grammar:

$$\phi ::= \mathsf{p} \mid \mathsf{and}(\phi, \phi) \mid \mathsf{or}(\phi, \phi) \mid \mathsf{not}(\phi)$$

We present two imperative encodings: in the first, the function is shared via a pointer and recursion is achieved via dereferencing. In the second, formulas are shared too with back-pointers to shared-subtrees. Type decorations are omitted

Imperative, I

This imperative encoding uses a variable SELF which contains a pointer to the recursive code: here the recursion is achieved directly via pointer dereferencing, assignment and classical imperative fixed-point in order to implement recursion. Given the constant dummy, the function nnf1 is defined as

$$\mathsf{nnf1} \triangleq \left(\begin{array}{lcl} \mathsf{p} & \rightarrow & \mathsf{p}, \\ \mathsf{not}(\mathsf{not}(\mathsf{X})) & \rightarrow & !\mathsf{SELF}(\mathsf{X}), \\ \mathsf{not}(\mathsf{or}(\mathsf{X},\mathsf{Y})) & \rightarrow & \mathsf{and}(!\mathsf{SELF}(\mathsf{not}(\mathsf{X})), !\mathsf{SELF}(\mathsf{not}(\mathsf{Y}))), \\ \mathsf{not}(\mathsf{and}(\mathsf{X},\mathsf{Y})) & \rightarrow & \mathsf{or}(!\mathsf{SELF}(\mathsf{not}(\mathsf{X})), !\mathsf{SELF}(\mathsf{not}(\mathsf{Y}))), \\ \mathsf{and}(\mathsf{X},\mathsf{Y}) & \rightarrow & \mathsf{and}(!\mathsf{SELF}(\mathsf{X}), !\mathsf{SELF}(\mathsf{Y})), \\ \mathsf{or}(\mathsf{X},\mathsf{Y}) & \rightarrow & \mathsf{or}(!\mathsf{SELF}(\mathsf{X}), !\mathsf{SELF}(\mathsf{Y})) \end{array} \right)$$

and the imperative encoding is:

$$\mathsf{let}\ \mathsf{SELF} \ll \mathsf{ref}\ \mathsf{dummy}\ \mathsf{in}\ \mathsf{let}\ \mathsf{NNF} \ll \mathsf{nnf1}\ \mathsf{in}\ \mathsf{SELF} := \mathsf{NNF}; \mathsf{NNF}(\phi)$$

Imperative-with-Sharing, IS

This encoding uses a variable SELF which contains a pointer to the recursive code and a flag-pointer to a boolean value associated to each node: all flag-pointers are initially set to false; each time we scan a (possibly) shared-formulas we set the corresponding flag-pointer to true. The grammar of shared-formulas is as follows:

$$
\begin{array}{lcl}
\mathsf{bool} & ::= & \mathsf{true} \mid \mathsf{false} \\
\mathsf{flag} & ::= & \mathsf{bool\ ref} \\
\psi & ::= & \mathsf{ref}\ \phi \\
\phi & ::= & \mathsf{p} \mid \mathsf{and}(\mathsf{flag}, \psi, \psi) \mid \mathsf{or}(\mathsf{flag}, \psi, \psi) \mid \mathsf{not}(\mathsf{flag}, \psi)
\end{array}
$$

Imperative-with-Sharing, IS

Given the constant dummy, the structure nnf2 is defined as follows:

$$
\left(\begin{array}{lcl}
\mathsf{p} & \twoheadrightarrow & \mathsf{p}, \\
\mathsf{not}(B_1, \mathsf{ref\ not}(B_2, \mathsf{X})) & \twoheadrightarrow & !\mathsf{SELF}(!\mathsf{X}), \\
\mathsf{not}(B_1, \mathsf{ref\ or}(B_2, \mathsf{X}, \mathsf{Y})) & \twoheadrightarrow & \mathsf{and}(\mathsf{ref\ false}, !\mathsf{SELF}(\mathsf{ref\ not}(\mathsf{ref\ false}, \mathsf{X})), \\
 & & \qquad !\mathsf{SELF}(\mathsf{ref\ not}(\mathsf{ref\ false}, \mathsf{Y}))), \\
\mathsf{not}(B_1, \mathsf{ref\ and}(B_2, \mathsf{X}, \mathsf{Y})) & \twoheadrightarrow & \mathsf{or}(\mathsf{ref\ false}, !\mathsf{SELF}(\mathsf{ref\ not}(\mathsf{ref\ false}, \mathsf{X})), \\
 & & \qquad !\mathsf{SELF}(\mathsf{ref\ not}(\mathsf{ref\ false}, \mathsf{Y}))), \\
\mathsf{and}(B, \mathsf{X}, \mathsf{Y}) & \twoheadrightarrow & \Longleftrightarrow\ (\mathsf{neg\ ref}\ B) \\
 & & \mathsf{then}\ \ (B, \mathsf{X}, \mathsf{Y}) := (\mathsf{true}, !\mathsf{SELF}(!\mathsf{X}), !\mathsf{SELF}(!\mathsf{Y})) \\
 & & \mathsf{else}\ \ \mathsf{and}(B, \mathsf{X}, \mathsf{Y}), \\
\mathsf{or}(B, \mathsf{X}, \mathsf{Y}) & \twoheadrightarrow & \Longleftrightarrow\ (\mathsf{neg\ ref}\ B) \\
 & & \mathsf{then}\ \ (B, \mathsf{X}, \mathsf{Y}) := (\mathsf{true}, !\mathsf{SELF}(!\mathsf{X}), !\mathsf{SELF}(!\mathsf{Y})) \\
 & & \mathsf{else}\ \ \mathsf{or}(B, \mathsf{X}, \mathsf{Y})
\end{array}\right)
$$

and the imperative encoding is:

$$\mathsf{let\ SELF} \ll \mathsf{ref\ dummy\ in\ let\ NNF} \ll \mathsf{nnf2\ in\ SELF} := \mathsf{NNF}; \mathsf{NNF}(\psi)$$

iRho

Types

Why a type system?

- Our type discipline assigns a semantical meaning to iRho-programs by type-checking and hence, allows to catch some error before run-time
- The type system is powerful enough to ensure a type consistency, and to give a type to a rich collection of interesting examples, namely decision procedures, meaningful objects, fixed-points, term rewriting systems, etc
- This type system is, in principle, suitable to be extended with a *subtyping* relation, or with *bounded-polymorphism*, to capture the behavior of *structures-as-objects*, and object-oriented features

What's new in this type system

- The main novelty, with respect to previous type systems for the (functional) fRho is that term-structures can have different types, $i.e.$

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau_1 \quad \Gamma \vdash_{\mathsf{A}} B : \tau_2}{\Gamma \vdash_{\mathsf{A}} A, B : \tau_1 \wedge \tau_2}\,(\mathrm{Term{-}Struct})$$

- The new kind of type $\tau_1 \wedge \tau_2$ is suitable for heterogeneous (non-commutative) structures, like lists, ordered sets, or objects
- More flexible type discipline, where the structure-type $\tau_1 \wedge \tau_2$ reflects the implicit non-commutative property of "," in the term "A, B", $i.e.$ "A, B" does not behave necessarily as "B, A"
- More expressiveness $w.r.t.$ previous typing disciplines on the fRho, in the sense that it gives a type to terms that will not be stuck at run-time, but it complicates the metatheory and the mechanical proof development

Type Judgments

Recall Types and Contexts

$\tau ::= \mathsf{b} \mid \tau \rightarrowtriangle \tau \mid \tau \wedge \tau \mid \tau \ \mathsf{ref}$

$\Gamma ::= \emptyset \mid \Gamma, X{:}\tau \mid \Gamma, f{:}\tau$

The Type Judgments

$\Gamma \vdash_{\Gamma} ok \qquad \Gamma \vdash_{\tau} \tau : ok \quad \Gamma \vdash_{\mathsf{v}} A_{\mathsf{v}} : \tau$

$\Gamma \vdash_{\rho} \rho : \Gamma' \quad \Gamma \vdash_{\sigma} \sigma : \Gamma' \quad \Gamma \vdash_{\mathsf{P}} P : \tau \qquad \Gamma \vdash_{\mathsf{A}} A : \tau$

$i.e.$ well-typed contexts, types, values, environments, stores, patterns, and terms

iRho:

$$\Gamma \vdash_{\mathsf{P}} P : \tau$$

$\vdash_{\mathsf{A}}$: Pattern Rules

$$\frac{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\Gamma} ok}{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\mathsf{P}} \alpha : \tau}\,(\mathrm{Patt{-}Start})$$

$$\frac{\Gamma \vdash_{\mathsf{P}} P_1 : \tau_1 \quad \Gamma \vdash_{\mathsf{P}} P_2 : \tau_2}{\Gamma \vdash_{\mathsf{P}} P_1, P_2 : \tau_1 \wedge \tau_2}\,(\mathrm{Patt{-}Struct})$$

$\vdash_{\mathsf{A}}$: Pattern Rules

$$\frac{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\Gamma} ok}{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\mathsf{P}} \alpha : \tau}\,(\text{Patt}-\text{Start})$$

$$\frac{\Gamma \vdash_{\mathsf{P}} P_1 : \tau_1 \quad \Gamma \vdash_{\mathsf{P}} P_2 : \tau_2}{\Gamma \vdash_{\mathsf{P}} P_1, P_2 : \tau_1 \wedge \tau_2}\,(\text{Patt}-\text{Struct})$$

$$\frac{\mathsf{arr}(\tau_1) \equiv \tau_2 \twoheadrightarrow \tau_3 \quad \Gamma \vdash_{\mathsf{P}} f\ \overline{P} : \tau_1 \quad \Gamma \vdash_{\mathsf{P}} P : \tau_2}{\Gamma \vdash_{\mathsf{P}} f\ \overline{P}\ P : \tau_3}\,(\text{Patt}-\text{Algbr})$$

$\vdash_{\mathsf{A}}$: Pattern Rules

$$\frac{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\Gamma} ok}{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\mathsf{P}} \alpha : \tau}\,(\mathrm{Patt{-}Start})$$

$$\frac{\Gamma \vdash_{\mathsf{P}} P_1 : \tau_1 \quad \Gamma \vdash_{\mathsf{P}} P_2 : \tau_2}{\Gamma \vdash_{\mathsf{P}} P_1, P_2 : \tau_1 \wedge \tau_2}\,(\mathrm{Patt{-}Struct})$$

$$\frac{\mathsf{arr}(\tau_1) \equiv \tau_2 \twoheadrightarrow \tau_3 \quad \Gamma \vdash_{\mathsf{P}} f\ \overline{P} : \tau_1 \quad \Gamma \vdash_{\mathsf{P}} P : \tau_2}{\Gamma \vdash_{\mathsf{P}} f\ \overline{P}\ P : \tau_3}\,(\mathrm{Patt{-}Algbr)}$$

$$\mathsf{arr}(\tau_1 \twoheadrightarrow \tau_2) \triangleq \tau_1 \twoheadrightarrow \tau_2$$

$$\mathsf{arr}(\tau_1 \wedge \tau_2) \triangleq \tau_3 \twoheadrightarrow (\tau_4 \wedge \tau_5) \left\{ \begin{array}{ll} \Longleftrightarrow & \mathsf{arr}(\tau_1) \equiv \tau_3 \twoheadrightarrow \tau_4 \\ \mathsf{and} & \mathsf{arr}(\tau_2) \equiv \tau_3 \twoheadrightarrow \tau_5 \end{array} \right.$$

iRho:

$$\Gamma \vdash_{\mathsf{A}} A : \tau$$

$\vdash_{\mathsf{A}}$: Variables, Constants, and Structures

$$\frac{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\Gamma} ok}{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\mathsf{A}} \alpha : \tau}\,(\mathrm{Term{-}Start})$$

$\vdash_{\mathsf{A}}$: Variables, Constants, and Structures

$$\frac{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\Gamma} ok}{\Gamma_1, \alpha{:}\tau, \Gamma_2 \vdash_{\mathsf{A}} \alpha : \tau}\,(\text{Term}-\text{Start})$$

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau_1 \quad \Gamma \vdash_{\mathsf{A}} B : \tau_2}{\Gamma \vdash_{\mathsf{A}} A, B : \tau_1 \wedge \tau_2}\,(\text{Term}-\text{Struct})$$

$\vdash_{\mathsf{A}}$: Abstraction and Application

$$\frac{\mathcal{D}om(\Delta) = \mathcal{FV}ar(P) \quad \Gamma, \Delta \vdash_{\mathsf{P}} P : \tau_1 \quad \Gamma, \Delta \vdash_{\mathsf{A}} A : \tau_2}{\Gamma \vdash_{\mathsf{A}} P \twoheadrightarrow_{\Delta} A : \tau_1 \twoheadrightarrow \tau_2}\,(\text{Tern}$$

$\vdash_{\mathsf{A}}$: Abstraction and Application

$$\frac{\mathcal{D}om(\Delta) = \mathcal{FV}ar(P) \quad \Gamma, \Delta \vdash_{\mathsf{P}} P : \tau_1 \quad \Gamma, \Delta \vdash_{\mathsf{A}} A : \tau_2}{\Gamma \vdash_{\mathsf{A}} P \rightarrowtriangle_{\Delta} A : \tau_1 \rightarrowtriangle \tau_2}\,(\mathrm{Tern}$$

$$\frac{\mathsf{arr}(\tau_1) \equiv \tau_2 \rightarrowtriangle \tau_3 \quad \Gamma \vdash_{\mathsf{A}} A : \tau_1 \quad \Gamma \vdash_{\mathsf{A}} B : \tau_2}{\Gamma \vdash_{\mathsf{A}} A\ B : \tau_3}\,(\mathrm{Term{-}Ap}$$

$\vdash_{\mathsf{A}}$: Assign, Referencing, Dereferencing

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau \ \mathsf{ref} \qquad \Gamma \vdash_{\mathsf{A}} B : \tau}{\Gamma \vdash_{\mathsf{A}} A := B : \tau}\,(\mathrm{Term-Assign})$$

$\vdash_{\mathsf{A}}$: Assign, Referencing, Dereferencing

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau \ \mathsf{ref} \qquad \Gamma \vdash_{\mathsf{A}} B : \tau}{\Gamma \vdash_{\mathsf{A}} A := B : \tau} (\mathrm{Term{-}Assign})$$

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau}{\Gamma \vdash_{\mathsf{A}} \mathsf{ref}\, A : \tau \ \mathsf{ref}} (\mathrm{Term{-}Ref})$$

$\vdash_{\mathsf{A}}$: Assign, Referencing, Dereferencing

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau \ \mathsf{ref} \qquad \Gamma \vdash_{\mathsf{A}} B : \tau}{\Gamma \vdash_{\mathsf{A}} A := B : \tau} (\mathrm{Term{-}Assign})$$

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau}{\Gamma \vdash_{\mathsf{A}} \mathsf{ref}\, A : \tau \ \mathsf{ref}} (\mathrm{Term{-}Ref})$$

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau \ \mathsf{ref}}{\Gamma \vdash_{\mathsf{A}} !A : \tau} (\mathrm{Term{-}Deref})$$

iRho:

$$\vdash_{\Gamma} \quad \vdash_{\tau} \quad \vdash_{\mathsf{v}} \quad \vdash_{\rho} \quad \vdash_{\sigma}$$

Almost Routine (Hence Omitted)

Typing the Two Imperative Encodings

If b type of ϕ, and b ref type of ψ, and $\bigwedge^n \tau \triangleq \overbrace{\tau \wedge \cdots \wedge \tau}^{n}$

Typing the Two Imperative Encodings

If b type of ϕ, and b ref type of ψ, and $\bigwedge^n \tau \triangleq \overbrace{\tau \wedge \cdots \wedge \tau}^{n}$

$$\text{let SELF} \ll \text{ref dummy in let NNF} \ll \text{nnf1 in SELF} := \text{NNF}; \text{NNF}(\phi)$$

$$\text{let SELF} \ll \text{ref dummy in let NNF} \ll \text{nnf2 in SELF} := \text{NNF}; \text{NNF}(\psi)$$

Typing the Two Imperative Encodings

If b type of ϕ, and b ref type of ψ, and $\overset{n}{\wedge}\tau \triangleq \overbrace{\tau \wedge \cdots \wedge \tau}^{n}$

let SELF $\ll$ ref dummy in let NNF $\ll$ nnf1 in SELF $:=$ NNF; NNF(ϕ)

let SELF $\ll$ ref dummy in let NNF $\ll$ nnf2 in SELF $:=$ NNF; NNF(ψ)

$\Gamma_1 \triangleq$ dummy: $\overset{n}{\wedge}\tau_1$, SELF: $\overset{n}{\wedge}\tau_1$ ref (with $\tau_1 \triangleq$ b $\rightarrowtriangle$ b)

$\Gamma_2 \triangleq$ dummy: $\overset{n}{\wedge}\tau_2$, SELF: $\overset{n}{\wedge}\tau_2$ ref (with $\tau_2 \triangleq$ b ref $\rightarrowtriangle$ b ref)

Typing the Two Imperative Encodings

If b type of ϕ, and b ref type of ψ, and $\overset{n}{\wedge}\tau \triangleq \overbrace{\tau \wedge \cdots \wedge \tau}^{n}$

let SELF $\ll$ ref dummy in let NNF $\ll$ nnf1 in SELF $:=$ NNF; NNF(ϕ)

let SELF $\ll$ ref dummy in let NNF $\ll$ nnf2 in SELF $:=$ NNF; NNF(ψ)

$\Gamma_1 \triangleq$ dummy: $\overset{n}{\wedge}\tau_1$, SELF: $\overset{n}{\wedge}\tau_1$ ref (with $\tau_1 \triangleq$ b $\rightarrowtriangle$ b)

$\Gamma_2 \triangleq$ dummy: $\overset{n}{\wedge}\tau_2$, SELF: $\overset{n}{\wedge}\tau_2$ ref (with $\tau_2 \triangleq$ b ref $\rightarrowtriangle$ b ref)

(I) Γ_1, X: $\overset{n}{\wedge}\tau_1$, NNF: $\overset{n}{\wedge}\tau_1 \vdash$ NNF(ϕ) : $\overset{n}{\wedge}$b

(IS) Γ_2, X: $\overset{n}{\wedge}\tau_2$, NNF: $\overset{n}{\wedge}\tau_2 \vdash$ NNF(ψ) : $\overset{n}{\wedge}$b ref

iRho:

Properties

Properties (✓= proved by the proof assistant Coq)

(Determinism)✓ If $\sigma \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A'_{\mathsf{v}} \cdot \sigma'$, and $\sigma \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A''_{\mathsf{v}} \cdot \sigma''$, then $A'_{\mathsf{v}} \equiv A''_{\mathsf{v}}$, and $\sigma' \equiv \sigma''$;

(Unique Type)✓ If $\Gamma \vdash_{\mathsf{A}} A : \tau$, then τ is unique;

(Coherence)✓ $\sigma \cdot \rho \vdash_{\mathsf{coh}} \Gamma$ if there exist two sub-contexts Γ_1, and Γ_2, such that $\Gamma_1, \Gamma_2 \equiv \Gamma$, and $\Gamma \vdash_\sigma \sigma : \Gamma_1$, and $\Gamma \vdash_\rho \rho : \Gamma_2$;

(Subject-reduction)✓ If $\emptyset \vdash_{\mathsf{A}} A : \tau$, and $\emptyset \cdot \emptyset \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma$, then there exists Γ' which extend Γ, such that $\Gamma' \vdash_\sigma \sigma : ok$, and $\Gamma' \vdash_{\mathsf{v}} A_{\mathsf{v}} : \tau$.

(Type-soundness) If $\emptyset \vdash_{\mathsf{A}} A : \tau$, then $\emptyset \cdot \emptyset \vdash A \Downarrow_{\mathsf{call}} A_{\mathsf{v}}$;

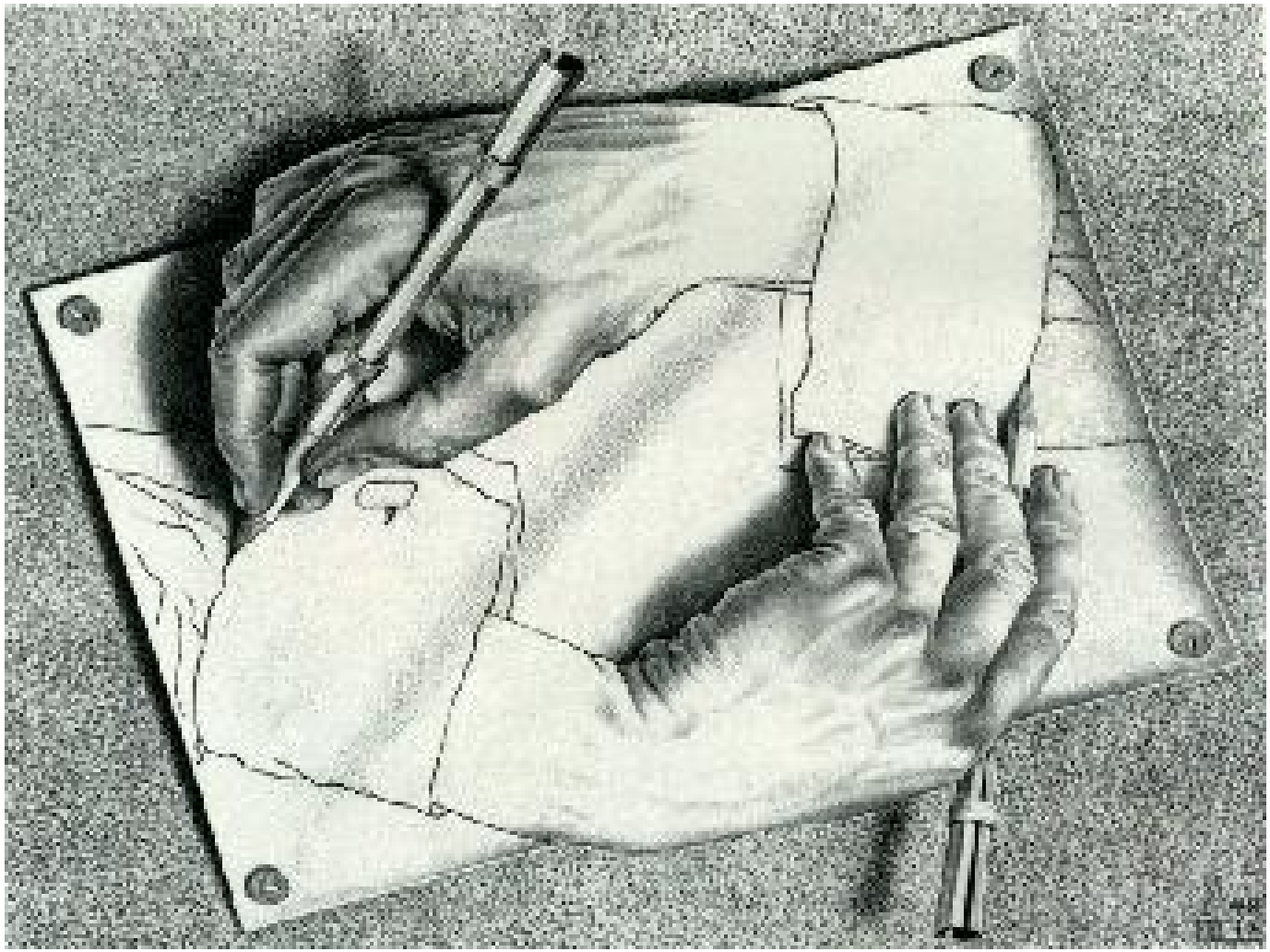
(Type-reconstruction) It is decidable if, for a given τ, is it true that $\emptyset \vdash_{\mathsf{A}} A : \tau$;

(Type-checking) It is decidable if there a type τ such that $\emptyset \vdash_{\mathsf{A}} A : \tau$.

Conclusion

- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor
  - ⋆ a "sound" machine (the interpreter)
  - ⋆ a "sound" type system (the type checker), such that "well typed programs do not go wrong" [Milner],
  - ⋆ both semantics being suitable to be specified with nice mathematics,
  - ⋆ to be implemented with high-level programming languages,
  - ⋆ and to be certified with modern and semi-automatic theorem provers, like Coq
- The paper is quasi a "technical manual" of the softwares (2)
- See `imprho.ps`, and `imprho.scm`, and `TypedImpRho.v`

DIMPRO



The explicit rewriting calculus

Lists in the λ-calculus vs in the ρ-calculus

	λ-calculus	ρ-calculus
`cons`	$\lambda XYZ.ZXY$	$X \twoheadrightarrow Y \twoheadrightarrow \texttt{Cons}(X\ Y)$
`car`	$\lambda Z.Z(\lambda XY.X)$	$\texttt{Cons}(X\ Y) \twoheadrightarrow X$
`cdr`	$\lambda Z.Z(\lambda XY.Y)$	$\texttt{Cons}(X\ Y) \twoheadrightarrow Y$

Run Time errors in O'Caml

```
#let car l = match l with
 x::m -> x;;
```

Run Time errors in O'Caml

```
#let car l = match l with
 x::m -> x;;

#car [];;
Exception: Match_failure (", 12, 42).
```

Run Time errors in O'Caml

```
#let car l = match l with
 x::m -> x;;

#car [];;
Exception: Match_failure (", 12, 42).

#let car l = match l with
  [] -> failwith ``error in car''
 |x::m -> x;;
```

Run Time errors in O'Caml

```
#let car l = match l with
 x::m -> x;;

#car [];;
Exception: Match_failure (", 12, 42).

#let car l = match l with
  [] -> failwith ``error in car''
 |x::m -> x;;
```

➥ Need to deal with error "by hand"

Errors in the ρ-calculus

$$car \triangleq \mathtt{Cons}(X, Y) \twoheadrightarrow X$$

$$\begin{aligned}(\mathtt{Cons}(X, Y) &\twoheadrightarrow X)\ \mathtt{Empty} \\ &\mapsto_{\rho}\ [\mathtt{Cons}(X, Y) \ll \mathtt{Empty}]X\end{aligned}$$

Errors in the ρ-calculus

Checking if two persons are brothers ($i.e.$ they have the same father):

$$\mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(X),\mathtt{Father}(Z)),\mathtt{Person}(\mathtt{Name}(Y),\mathtt{Father}(Z))) \twoheadrightarrow \mathtt{tt}$$

Errors in the ρ-calculus

Checking if two persons are brothers ($i.e.$ they have the same father):

$$\texttt{Brother}(\texttt{Person}(\texttt{Name}(X),\texttt{Father}(Z)),\texttt{Person}(\texttt{Name}(Y),\texttt{Father}(Z))) \twoheadrightarrow \texttt{tt}$$

In plain ρ-calculus when we apply the rule to the term

$$\texttt{Brother}(\texttt{Person}(\texttt{Name}(\texttt{Liz}),\texttt{Father}(\texttt{John})),\texttt{Person}(\texttt{Name}(\texttt{Bob}),\texttt{Father}(\texttt{Jim})))$$

we obtain as result the term

Errors in the ρ-calculus

Checking if two persons are brothers ($i.e.$ they have the same father):

$$\mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(X),\mathtt{Father}(Z)),\mathtt{Person}(\mathtt{Name}(Y),\mathtt{Father}(Z))) \twoheadrightarrow \mathtt{tt}$$

In plain ρ-calculus when we apply the rule to the term

$$\mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(\mathtt{Liz}),\mathtt{Father}(\mathtt{John})),\mathtt{Person}(\mathtt{Name}(\mathtt{Bob}),\mathtt{Father}(\mathtt{Jim})))$$

we obtain as result the term

$$[\mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(X),\mathtt{Father}(Z)),\mathtt{Person}(\mathtt{Name}(Y),\mathtt{Father}(Z))) \ll \mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(\mathtt{Liz}),\mathtt{Father}(\mathtt{John})),\mathtt{Person}(\mathtt{Name}(\mathtt{Bob}),\mathtt{Father}(\mathtt{Jim}))))]\mathtt{tt}$$

Errors in the ρ-calculus

Checking if two persons are brothers ($i.e.$ they have the same father):

$$\mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(X),\mathtt{Father}(Z)),\mathtt{Person}(\mathtt{Name}(Y),\mathtt{Father}(Z))) \twoheadrightarrow \mathtt{tt}$$

In plain ρ-calculus when we apply the rule to the term

$$\mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(\mathtt{Liz}),\mathtt{Father}(\mathtt{John})),\mathtt{Person}(\mathtt{Name}(\mathtt{Bob}),\mathtt{Father}(\mathtt{Jim})))$$

we obtain as result the term

$$[\mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(X),\mathtt{Father}(Z)),\mathtt{Person}(\mathtt{Name}(Y),\mathtt{Father}(Z))) \ll \mathtt{Brother}(\mathtt{Person}(\mathtt{Name}(\mathtt{Liz}),\mathtt{Father}(\mathtt{John})),\mathtt{Person}(\mathtt{Name}(\mathtt{Bob}),\mathtt{Father}(\mathtt{Jim})))]\mathtt{tt}$$

while in $\rho_{\mathtt{x}} - calculus$-calculus the result is

$$(Z \ll \mathtt{John} \wedge Z \ll \mathtt{Jim})\ \mathtt{tt}$$

How to represent programs

- Pattern Matching (possibly non-linear).
- Typed Recursion and Strategies ...to put link
- Imperative features ...to put link

Explicit constraint handling

What for?

- to implement the ρ-calculus [Rogue:Stump]

Explicit constraint handling

What for?

- to implement the ρ-calculus [Rogue:Stump]
- to represent proof-terms of rewriting derivation [Nguyen]

Explicit constraint handling

What for?

- to implement the ρ-calculus [Rogue:Stump]
- to represent proof-terms of rewriting derivation [Nguyen]
- to have a precise control over matching and constraints. Better deal of errors.

Explicit constraint handling

What for?

- to implement the ρ-calculus [Rogue:Stump]
- to represent proof-terms of rewriting derivation [Nguyen]
- to have a precise control over matching and constraints. Better deal of errors.

How?

- by making explicit matching computations
- by making explicit application of substitutions

Explicit matching decomposition

- Decompose functional symbols.
- Decompose the structure ";".
- Do not decompose: Abstraction and Application.

Constraint application

What about?

$$[X \ll a \wedge a \ll b]X$$

Constraint application

What about?

$$[X \ll a \wedge a \ll b]X$$

➥ Do not apply constraint without solution

Constraint application

What about?

$$[X \ll a \wedge a \ll b]X$$

➥ Do not apply constraint without solution

From constraints to substitutions

$$[\ X \ll A \wedge \underbrace{\qquad\qquad}_{\mathfrak{C}}\ ]B$$

Constraint application

What about?

$$[X \ll a \wedge a \ll b]X$$

➥ Do not apply constraint without solution

From constraints to substitutions

$$[\ X \ll A \wedge \underbrace{\qquad}_{\mathfrak{C}}\ ]B \quad \longrightarrow \quad [\ \underbrace{\qquad}_{\mathfrak{C}}\ ]\{\ X \ll A\ \}B$$

$$\texttt{if}\ \ X \notin \mathcal{D}om(\mathfrak{C})$$

Syntax of the ρ_{x}-calculus-calculus

Terms	A, B	$::=$	$\mathcal{X}$	(Variables)
		$\mid$	$\mathcal{K}$	(Constants)
		$\mid$	$A \twoheadrightarrow B$	(Abstraction)
		$\mid$	$A\ B$	(Fonctional application)
		$\mid$	$\mathcal{C}\ B$	(Constraint application)
		$\mid$	A, B	(Structure)
		$\mid$	$\{X \ll A\}B$	(Substitution application on terms)
Constraints	$\mathcal{C}, \mathcal{D}$	$::=$	$A \ll B$	(Match-equation)
		$\mid$	$\mathcal{C} \wedge \mathcal{D}$	(Conjonctions of contraints)
		$\mid$	$\{X \ll A\}\mathcal{C}$	(Substitution application on const.)

Semantics of the ρ_{x}-calculus-calculus

From rewrite rules to constraints

(ρ)	$(A \twoheadrightarrow B)\ C$	$\longrightarrow$	$(A \ll C)\ B$
(δ)	$(A; B)\ C$	$\longrightarrow$	$A\ C; B\ C$

Semantics of the ρ_{x}-calculus-calculus

From rewrite rules to constraints

(ρ)	$(A \twoheadrightarrow B)\ C$	$\rightarrow$	$(A \ll C)\ B$
(δ)	$(A; B)\ C$	$\rightarrow$	$A\ C; B\ C$

From constraints to substitutions

Decomposition

$(Decompose;)$	$A_1; A_2 \ll B_1; B_2$	$\rightarrow$	$A_1 \ll B_1 \wedge\ A_2 \ll B_2$
$(Decompose_{\mathcal{F}})$	$f(A_1 \dots A_n) \ll f(B_1 \dots B_n)$	$\rightarrow$	$A_1 \ll B_1 \wedge \dots \wedge A_n \ll B_n$
$(NGood)$	$f(A_1 \dots A_n) \ll g(B_1 \dots B_n)$	$\rightarrow$	$f(A_1 \dots A_n)\ \ll^{\mathbf{ng}}\ g(B_1 \dots B_n)$

Semantics of the ρ_x-calculus-calculus

From rewrite rules to constraints

(ρ)	$(A \twoheadrightarrow B)\ C$	$\rightarrow$	$(A \ll C)\ B$
(δ)	$(A; B)\ C$	$\rightarrow$	$A\ C; B\ C$

From constraints to substitutions

Decomposition

$(Decompose;)$	$A_1; A_2 \ll B_1; B_2$	$\rightarrow$	$A_1 \ll B_1 \wedge\ A_2 \ll B_2$
$(Decompose_{\mathcal{F}})$	$f(A_1 \dots A_n) \ll f(B_1 \dots B_n)$	$\rightarrow$	$A_1 \ll B_1 \wedge \dots \wedge A_n \ll B_n$
$(NGood)$	$f(A_1 \dots A_n) \ll g(B_1 \dots B_n)$	$\rightarrow$	$f(A_1 \dots A_n)\ \ll^{\mathbf{ng}}\ g(B_1 \dots B_n)$

From constraints to substitutions

$(ToSubst_{\wedge})$	$(X \ll A \wedge \mathcal{C})B$	$\rightarrow$	$(\mathcal{C})(\{X \ll A\}B)$ if $X \notin \mathcal{D}om(\mathcal{C})$

Semantics of the ρ_x-calculus-calculus

From rewrite rules to constraints

(ρ)	$(A \twoheadrightarrow B)\ C$	$\rightarrow$	$(A \ll C)\ B$
(δ)	$(A; B)\ C$	$\rightarrow$	$A\ C; B\ C$

From constraints to substitutions

Decomposition

$(Decompose;)$	$A_1; A_2 \ll B_1; B_2$	$\rightarrow$	$A_1 \ll B_1 \wedge\ A_2 \ll B_2$
$(Decompose_{\mathcal{F}})$	$f(A_1 \dots A_n) \ll f(B_1 \dots B_n)$	$\rightarrow$	$A_1 \ll B_1 \wedge \dots \wedge A_n \ll B_n$
$(NGood)$	$f(A_1 \dots A_n) \ll g(B_1 \dots B_n)$	$\rightarrow$	$f(A_1 \dots A_n)\ \ll^{\mathbf{ng}}\ g(B_1 \dots B_n)$

From constraints to substitutions

$(ToSubst_{\wedge})$	$(X \ll A \wedge \mathcal{C})B$	$\rightarrow$	$(\mathcal{C})(\{X \ll A\}B)$ if $X \notin \mathcal{D}om(\mathcal{C})$

Substitution applications

$(Replace)$	$\{X \ll A\}X$	$\rightarrow$	A
$(Eliminate_{\mathcal{X}})$	$\{X \ll A\}Y$	$\rightarrow$	Y if $X \neq Y$
$(Eliminate_{\mathcal{F}})$	$\{X \ll A\}f$	$\rightarrow$	f
$(Share_{;})$	$\{X \ll A\}(B\ ;\ C)$	$\rightarrow$	$\{X \ll A\}B\ ;\ \{X \ll A\}C$
$(Share_{()})$	$\{X \ll A\}((B)C)$	$\rightarrow$	$(\{X \ll A\}B)\{X \ll A\}C$
$(Share_{\twoheadrightarrow})$	$\{X \ll A\}(B \twoheadrightarrow C)$	$\rightarrow$	$B \twoheadrightarrow \{X \ll A\}C$
$(Share_{\ll})$	$\{X \ll A\}(B \ll C)$	$\rightarrow$	$B \ll \{X \ll A\}C$
$(Share_{\wedge})$	$\{X \ll A\}(\mathcal{C} \wedge \mathcal{D})$	$\rightarrow$	$\{X \ll A\}\mathcal{C}\ \wedge\ \{X \ll A\}\mathcal{D}$

Example

$$(\mathfrak{not}\,(\mathfrak{and}\,(X,Y)) \twoheadrightarrow \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y))\,) \qquad \mathfrak{not}\,(\mathfrak{and}\,(\mathfrak{tt},\mathfrak{ff}))$$

Example

$$(\mathfrak{not}\,(\mathfrak{and}\,(X,Y)) \twoheadrightarrow \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y))\,) \qquad \mathfrak{not}\,(\mathfrak{and}\,(\mathfrak{tt},\mathfrak{ff}))$$

$$\mapsto_{\rho} \qquad \left[\mathfrak{not}\,(\mathfrak{and}\,(X,Y)) \ll \mathfrak{not}\,(\mathfrak{and}\,(\mathfrak{tt},\mathfrak{ff}))\right] \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y))$$

Example

$$(\mathfrak{not}\,(\mathfrak{and}\,(X,Y)) \twoheadrightarrow \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y))\,) \qquad \mathfrak{not}\,(\mathfrak{and}\,(\mathfrak{tt},\mathfrak{ff}))$$

$$\begin{array}{ll} \mapsto_{p} & \left[\mathfrak{not}\,(\mathfrak{and}\,(X,Y)) \ll \mathfrak{not}\,(\mathfrak{and}\,(\mathfrak{tt},\mathfrak{ff}))\right] \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y)) \\ \mapsto_{Decompose} & \left[\mathfrak{and}\,(X,Y) \ll (\mathfrak{and}\,(\mathfrak{tt},\mathfrak{ff})\right] \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y)) \end{array}$$

Example

$$(\mathfrak{not}\,(\mathfrak{and}\,(X,Y)) \twoheadrightarrow \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y))\,) \qquad \mathfrak{not}\,(\mathfrak{and}\,(\mathfrak{tt},\mathfrak{ff}))$$

$$\begin{aligned}
&\mapsto_{\rho} && \left[\mathfrak{not}\,(\mathfrak{and}\,(X,Y)) \ll \mathfrak{not}\,(\mathfrak{and}\,(\mathfrak{tt},\mathfrak{ff}))\right] \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y)) \\
&\mapsto_{Decompose} && \left[\mathfrak{and}\,(X,Y) \ll (\mathfrak{and}\,(\mathfrak{tt},\mathfrak{ff})\right] \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y)) \\
&\mapsto_{Decompose} && \left[X \ll \mathfrak{tt} \wedge Y \ll \mathfrak{ff}\right] \mathfrak{or}\,(\mathfrak{not}\,(X)\,,\mathfrak{not}\,(Y))
\end{aligned}$$

Example

$$(\mathfrak{not}(\mathfrak{and}(X,Y)) \twoheadrightarrow \mathfrak{or}(\mathfrak{not}(X),\mathfrak{not}(Y)))\quad \mathfrak{not}(\mathfrak{and}(\mathfrak{tt},\mathfrak{ff}))$$

$$\begin{array}{ll}
\mapsto_{\rho} & \left[\mathfrak{not}(\mathfrak{and}(X,Y)) \ll \mathfrak{not}(\mathfrak{and}(\mathfrak{tt},\mathfrak{ff}))\right]\mathfrak{or}(\mathfrak{not}(X),\mathfrak{not}(Y)) \\
\mapsto_{Decompose} & \left[\mathfrak{and}(X,Y) \ll (\mathfrak{and}(\mathfrak{tt},\mathfrak{ff})\right]\mathfrak{or}(\mathfrak{not}(X),\mathfrak{not}(Y)) \\
\mapsto_{Decompose} & \left[X \ll \mathfrak{tt} \wedge Y \ll \mathfrak{ff}\right]\mathfrak{or}(\mathfrak{not}(X),\mathfrak{not}(Y)) \\
\mapsto_{Subst} & \left[X \ll \mathfrak{tt}\right]\{Y \ll \mathfrak{ff}\}\mathfrak{or}(\mathfrak{not}(X)\ \ \mathfrak{not}(Y))
\end{array}$$

Example

$$(\mathfrak{not}(\mathfrak{and}(X,Y)) \twoheadrightarrow \mathfrak{or}(\mathfrak{not}(X),\mathfrak{not}(Y)))\quad \mathfrak{not}(\mathfrak{and}(\mathfrak{tt},\mathfrak{ff}))$$

$$\begin{array}{ll}
\mapsto_{\rho} & \left[\mathfrak{not}(\mathfrak{and}(X,Y)) \ll \mathfrak{not}(\mathfrak{and}(\mathfrak{tt},\mathfrak{ff}))\right]\mathfrak{or}(\mathfrak{not}(X),\mathfrak{not}(Y)) \\
\mapsto_{Decompose} & \left[\mathfrak{and}(X,Y) \ll (\mathfrak{and}(\mathfrak{tt},\mathfrak{ff})\right]\mathfrak{or}(\mathfrak{not}(X),\mathfrak{not}(Y)) \\
\mapsto_{Decompose} & \left[X \ll \mathfrak{tt} \wedge Y \ll \mathfrak{ff}\right]\mathfrak{or}(\mathfrak{not}(X),\mathfrak{not}(Y)) \\
\mapsto_{Subst} & \left[X \ll \mathfrak{tt}\right]\{Y \ll \mathfrak{ff}\}\mathfrak{or}(\mathfrak{not}(X)\ \ \mathfrak{not}(Y)) \\
\mapsto^{*}_{Propagation} & (\mathfrak{or}(\mathfrak{not}(\mathfrak{tt}),\mathfrak{not}(\mathfrak{ff})))
\end{array}$$

➥ Need to compose, to able to combine substitutions, to limit term traversal.

Properties of the $\rho_{\mathtt{x}}$-calculus-calculus

- Well behaved properties for substitution application.
- Termination of the constraint handling part.
- Confluence of the calculus.
- Conservativity
- Simulating the λ-calculus$_x$

The $\rho_{\mathtt{xC}}$-calculus

$$(GI) \qquad X \ll A \wedge Y \ll B \quad \rightarrow \quad X \ll A \wedge_{\mathbf{g}} Y \ll B \quad \texttt{if } X \neq Y$$

The $\rho_{\mathtt{xC}}$-calculus

$$
\begin{array}{lll}
(GI) & X \ll A \wedge Y \ll B & \rightarrow \quad X \ll A \wedge_{\mathbf{g}} Y \ll B \\
 & & \quad \mathtt{if}\ \ X \neq Y \\
(Good) & \mathcal{C}^{\mathtt{g}} \wedge \mathcal{D}^{\mathtt{g}} & \rightarrow \quad \mathcal{C}^{\mathtt{g}} \wedge_{\mathbf{g}} \mathcal{D}^{\mathtt{g}} \\
 & & \quad \mathtt{if}\ \ \mathcal{D}om(\mathcal{C}^{\mathtt{g}}) \cap \mathcal{D}om(\mathcal{D}^{\mathtt{g}}) = \emptyset \\
(ToSubst) & [\mathcal{C}^{\mathtt{g}} \wedge \mathcal{D}]A & \rightarrow \quad [\mathcal{D}]\{\mathcal{C}^{\mathtt{g}}\}A
\end{array}
$$

The $\rho_{\mathtt{xC}}$-calculus

$$
\begin{array}{lll}
(GI) & X \ll A \wedge Y \ll B & \rightarrow \quad X \ll A \wedge_{\mathbf{g}} Y \ll B \\
 & & \quad \mathtt{if}\ \ X \neq Y \\
 & & \\
(Good) & \mathcal{C}^{\mathtt{g}} \wedge \mathcal{D}^{\mathtt{g}} & \rightarrow \quad \mathcal{C}^{\mathtt{g}} \wedge_{\mathbf{g}} \mathcal{D}^{\mathtt{g}} \\
 & & \quad \mathtt{if}\ \ \mathcal{D}om(\mathcal{C}^{\mathtt{g}}) \cap \mathcal{D}om(\mathcal{D}^{\mathtt{g}}) = \emptyset \\
(ToSubst) & [\mathcal{C}^{\mathtt{g}} \wedge \mathcal{D}]A & \rightarrow \quad [\mathcal{D}]\{\mathcal{C}^{\mathtt{g}}\}A \\
 & & \quad \mathtt{if}\ \ \mathcal{D}om(\mathcal{C}^{\mathtt{g}}) \cap \mathcal{D}om(\mathcal{D}) = \emptyset
\end{array}
$$

The $\rho_{\mathtt{xC}}$-calculus

To deal with compositions: the $\rho_{\mathtt{xC}}$-calculus

$$(Compose) \quad \{\vartheta\}(\{\varphi\}A) \quad \rightarrow \quad \{\{\vartheta\}\varphi\}(\{\vartheta\}A)$$

The $\rho_{\mathtt{xC}}$-calculus

To deal with compositions: the $\rho_{\mathtt{xC}}$-calculus

$$(Compose) \quad \{\vartheta\}(\{\varphi\}A) \quad \rightarrow \quad \{\{\vartheta\}\varphi\}(\{\vartheta\}A)$$

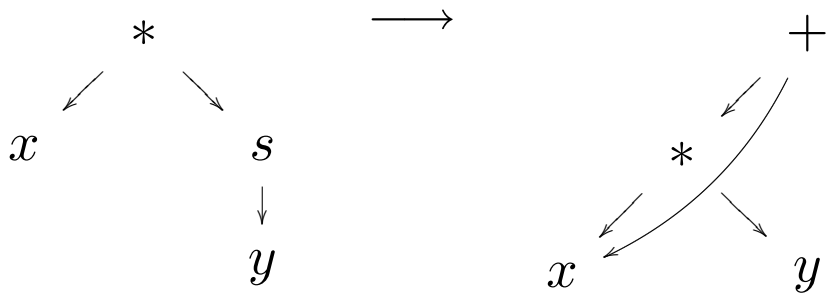
$$(Compose) \quad \{\vartheta\}(\{\varphi\}A) \quad \rightarrow \quad \{\vartheta \wedge_{\mathtt{g}} \{\vartheta\}\varphi\}\ A$$

➥ Properties (confluence AC – CiME)

Graphs in the rewriting calculus

Graphs in the ρ-calculus

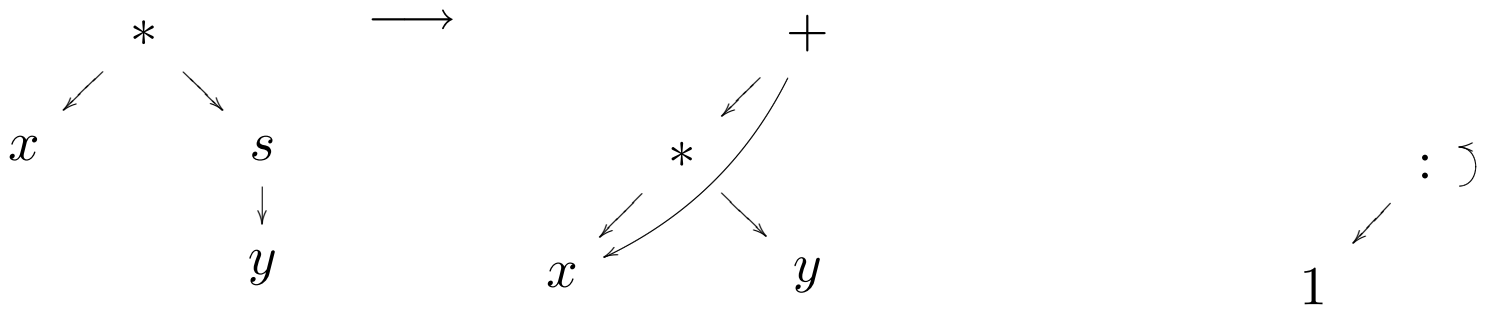
- improve efficiency
  - ➥ save space (sharing subterms)
  - ➥ save time (reduce only once)



Graphs in the ρ-calculus

- improve efficiency
  - ➥ save space (sharing subterms)
  - ➥ save time (reduce only once)

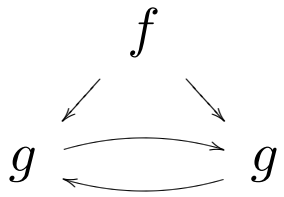
- improve expressiveness
  - ➥ infinite data structures

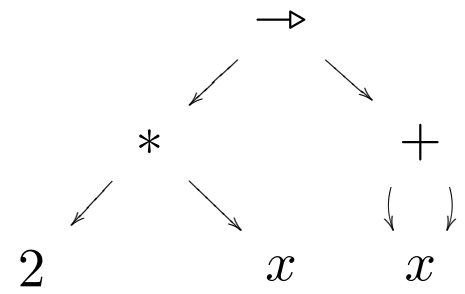


The syntax of ρ_{g}-calculus

Terms	$\mathcal{G}$	$::=$	$\mathcal{X}$	(Variables)
		$\mid$	$\mathcal{K}$	(Constants)
		$\mid$	$\mathcal{G} \twoheadrightarrow \mathcal{G}$	(Abstraction)
		$\mid$	$\mathcal{G}\ \mathcal{G}$	(Functional application)
		$\mid$	$\mathcal{G}, \mathcal{G}$	(Structure)
		$\mid$	$\mathcal{G}\ [\mathcal{C}]$	(Constraint application)
Constraints	$\mathcal{C}$	$::=$	ϵ	(Empty constraint)
		$\mid$	$\mathcal{X} = \mathcal{G}$	(Recursion equation)
		$\mid$	$\mathcal{G} \ll \mathcal{G}$	(Match equation)
		$\mid$	$\mathcal{C}, \mathcal{C}$	(Conjunction of constraints)

Some ρ_g-terms





$f(x, y)\ [x = g(y), y = g(x)]$

$(2 * x) \twoheadrightarrow ((y + y)\ [y = x])$

The semantics of ρ_{g}-calculus

$$
\begin{array}{lllll}
(\rho) & (G_1 \twoheadrightarrow G_2)\ G_3 & \to_\rho & G_2\ [G_1 \ll G_3] \\
 & (G_1 \twoheadrightarrow G_2)\ [E]\ G_3 & \to_\rho & G_2\ [G_1 \ll G_3, E] \\
(\delta) & (G_1, G_2)\ G_3 & \to_\delta & G_1\ G_3, G_2\ G_3 \\
 & (G_1, G_2)\ [E]\ G_3 & \to_\delta & (G_1\ G_3, G_2\ G_3)\ [E]
\end{array}
$$

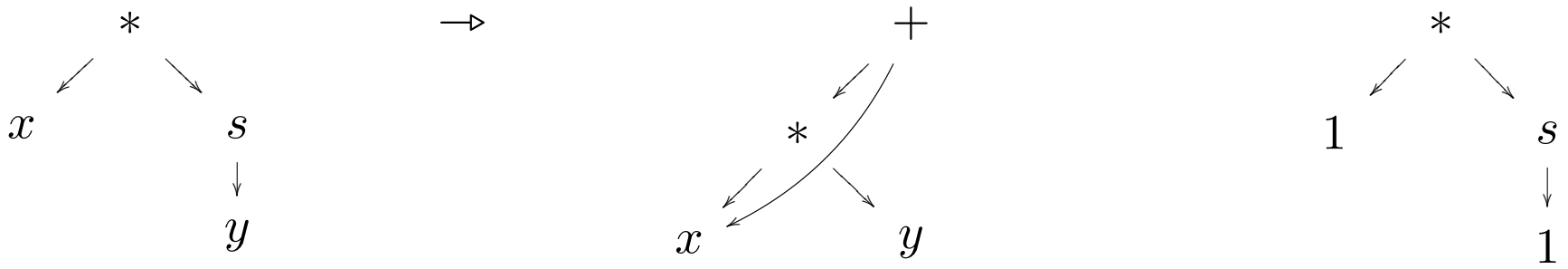
The semantics of ρ_{g}-calculus

$$
\begin{array}{lllll}
(\rho) & (G_1 \twoheadrightarrow G_2)\ G_3 & \to_\rho & G_2\ [G_1 \ll G_3] & \\
 & (G_1 \twoheadrightarrow G_2)\ [E]\ G_3 & \to_\rho & G_2\ [G_1 \ll G_3, E] & \\
(\delta) & (G_1, G_2)\ G_3 & \to_\delta & G_1\ G_3, G_2\ G_3 & \\
 & (G_1, G_2)\ [E]\ G_3 & \to_\delta & (G_1\ G_3, G_2\ G_3)\ [E] & \\
(propagate) & G_1 \ll (G_2\ [E_2]) & & \to_p & G_1 \ll G_2, E_2 \\
(decompose) & K(G_1, \ldots, G_n) \ll K(G'_1, \ldots, G'_n) & & \to_{dk} & G_1 \ll G'_1, \ldots, G_n \ll G'_n \\
(eliminate) & K \ll K, E & & \to_e & E \\
(solved) & x \ll G, E & & \to_s & x = G, E \quad \mathit{if}\ x \notin \mathcal{DV}(E)
\end{array}
$$

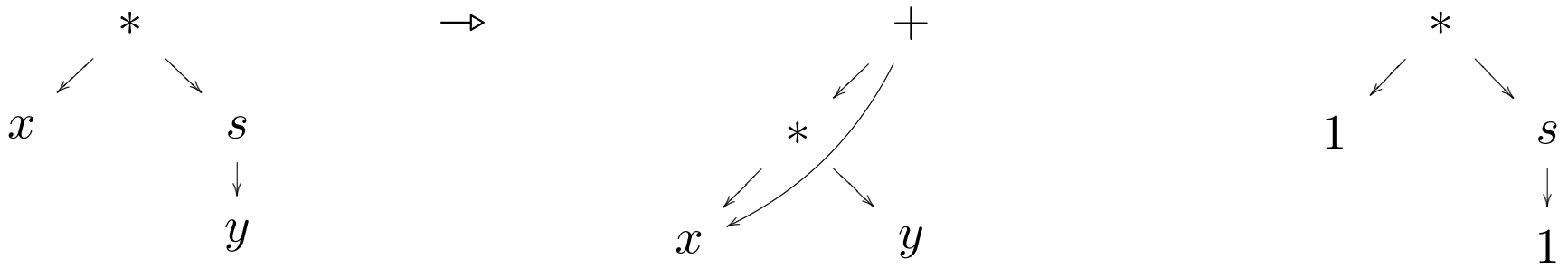
The semantics of ρ_g-calculus

$$
\begin{array}{lllll}
(\rho) & (G_1 \twoheadrightarrow G_2)\ G_3 & \to_\rho & G_2\ [G_1 \ll G_3] & \\
 & (G_1 \twoheadrightarrow G_2)\ [E]\ G_3 & \to_\rho & G_2\ [G_1 \ll G_3, E] & \\
(\delta) & (G_1, G_2)\ G_3 & \to_\delta & G_1\ G_3, G_2\ G_3 & \\
 & (G_1, G_2)\ [E]\ G_3 & \to_\delta & (G_1\ G_3, G_2\ G_3)\ [E] & \\
(propagate) & G_1 \ll (G_2\ [E_2]) & \to_p & G_1 \ll G_2, E_2 & \\
(decompose) & K(G_1, \ldots, G_n) \ll K(G'_1, \ldots, G'_n) & \to_{dk} & G_1 \ll G'_1, \ldots, G_n \ll G'_n & \\
(eliminate) & K \ll K, E & \to_e & E & \\
(solved) & x \ll G, E & \to_s & x = G, E \quad \textit{if } x \notin \mathcal{DV}(E) & \\
(external\ sub) & \mathsf{Ctx}[y]\ [y = G, E] & \to_{es} & \mathsf{Ctx}[\mathcal{G}]\ [y = G, E] & \\
(acyclic\ sub) & G\ [G_0 \lll \mathsf{Ctx}[y], y = G_1, E] & \to_{ac} & G\ [G_0 \lll \mathsf{Ctx}[G_1], y = G_1, E] & \\
 & & & \textit{if } x > y,\ \forall x \in \mathcal{FV}ar(G_0) & \\
 & & & \textit{where } \lll \in \{=, \ll\} & \\
(garbage) & G\ [E, x = G'] & \to_{gc} & G\ [E] & \\
 & & & \textit{if } x \notin \mathcal{FV}ar(E) \cup \mathcal{FV}ar(G) & \\
 & G\ [\epsilon] & \to_{gc} & G & \\
(black\ hole) & \mathsf{Ctx}[x]\ [x =_\circ x, E] & \to_{bh} & \mathsf{Ctx}[\bullet]\ [x =_\circ x, E] & \\
 & G\ [y = \mathsf{Ctx}[x], x =_\circ x, E] & \to_{bh} & G\ [y = \mathsf{Ctx}[\bullet], x =_\circ x, E] & \\
 & & & \textit{if } y > x &
\end{array}
$$

Example - Multiplication

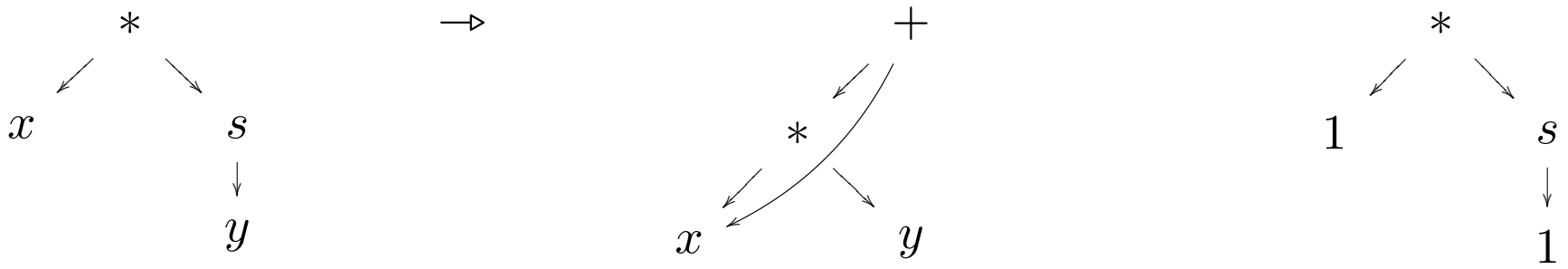


Example - Multiplication



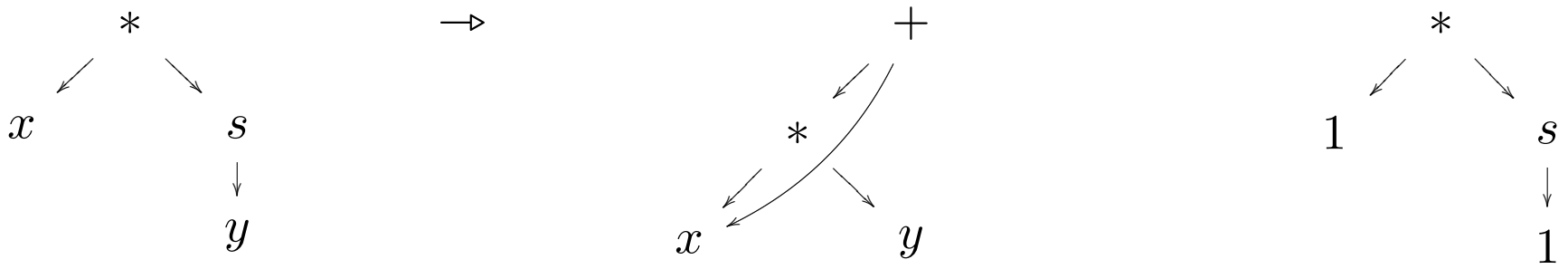
$$
\begin{array}{ll}
 & (x * s(y) \twoheadrightarrow (x_1 * y + x_1\ [x_1 = x]))\ (z * s(z)\ [z = 1]) \\
\mapsto_p & x_1 * y + x_1\ [x_1 = x]\ [x * s(y) \ll (z * s(z)\ [z = 1])] \\
\mapsto_p & x_1 * y + x_1\ [x_1 = x]\ [x * s(y) \ll z * s(z), z = 1]
\end{array}
$$

Example - Multiplication



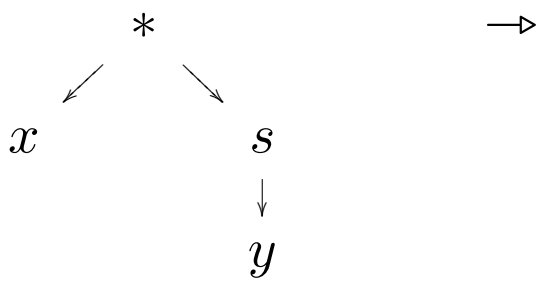
$$
\begin{array}{ll}
 & (x * s(y) \twoheadrightarrow (x_1 * y + x_1\ [x_1 = x]))\ (z * s(z)\ [z = 1]) \\
\mapsto_p & x_1 * y + x_1\ [x_1 = x]\ [x * s(y) \ll (z * s(z)\ [z = 1])] \\
\mapsto_p & x_1 * y + x_1\ [x_1 = x]\ [x * s(y) \ll z * s(z), z = 1] \\
\mapsto\!\!\!\!\rightarrow_{dk} & x_1 * y + x_1\ [x_1 = x]\ [x \ll z, y \ll z, z = 1] \\
\mapsto\!\!\!\!\rightarrow_s & x_1 * y + x_1\ [x_1 = x]\ [x = z, y = z, z = 1]
\end{array}
$$

Example - Multiplication

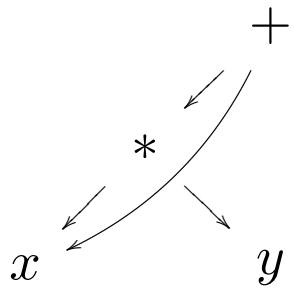


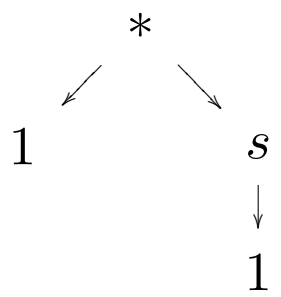
$$
\begin{array}{ll}
 & (x * s(y) \twoheadrightarrow (x_1 * y + x_1\ [x_1 = x]))\ (z * s(z)\ [z = 1]) \\
\mapsto_{p} & x_1 * y + x_1\ [x_1 = x]\ [x * s(y) \ll (z * s(z)\ [z = 1])] \\
\mapsto_{p} & x_1 * y + x_1\ [x_1 = x]\ [x * s(y) \ll z * s(z), z = 1] \\
\mapsto\!\!\!\!\rightarrow_{dk} & x_1 * y + x_1\ [x_1 = x]\ [x \ll z, y \ll z, z = 1] \\
\mapsto\!\!\!\!\rightarrow_{s} & x_1 * y + x_1\ [x_1 = x]\ [x = z, y = z, z = 1] \\
\mapsto\!\!\!\!\rightarrow_{es} & (z * z + z)\ [x_1 = z]\ [x = z, y = z, z = 1] \\
\mapsto\!\!\!\!\rightarrow_{gc} & (z * z + z)\ [x_1 = z]\ [z = 1] \\
\mapsto\!\!\!\!\rightarrow_{gc} & (z * z + z)\ [z = 1]
\end{array}
$$

Example - Multiplication

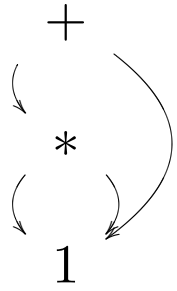


$\rightarrow$





$$
\begin{array}{ll}
 & (x * s(y) \rightarrow (x_1 * y + x_1\ [x_1 = x]))\ (z * s(z)\ [z = 1]) \\
\mapsto_{p} & x_1 * y + x_1\ [x_1 = x]\ [x * s(y) \ll (z * s(z)\ [z = 1])] \\
\mapsto_{p} & x_1 * y + x_1\ [x_1 = x]\ [x * s(y) \ll z * s(z), z = 1] \\
\mapsto\!\!\!\!\rightarrow_{dk} & x_1 * y + x_1\ [x_1 = x]\ [x \ll z, y \ll z, z = 1] \\
\mapsto\!\!\!\!\rightarrow_{s} & x_1 * y + x_1\ [x_1 = x]\ [x = z, y = z, z = 1] \\
\mapsto\!\!\!\!\rightarrow_{es} & (z * z + z)\ [x_1 = z]\ [x = z, y = z, z = 1] \\
\mapsto\!\!\!\!\rightarrow_{gc} & (z * z + z)\ [x_1 = z]\ [z = 1] \\
\mapsto\!\!\!\!\rightarrow_{gc} & (z * z + z)\ [z = 1]
\end{array}
$$



Example - Non-linearity

Success:

$$
\begin{array}{ll}
 & f(x,x)\ [x=y] \ll f(a,a) \\
\mapsto\!\!\!\!\rightarrow_{es} & f(y,y)\ [x=y] \ll f(a,a) \\
\mapsto\!\!\!\!\rightarrow_{gc} & f(y,y) \ll f(a,a) \\
\mapsto_{dk} & y \ll a \quad (by\ idempotency) \\
\mapsto_{s} & y = a
\end{array}
$$

Example - Non-linearity

Success:

$$
\begin{array}{ll}
 & f(x,x)\ [x=y] \ll f(a,a) \\
\mapsto\!\!\!\!\rightarrow_{es} & f(y,y)\ [x=y] \ll f(a,a) \\
\mapsto\!\!\!\!\rightarrow_{gc} & f(y,y) \ll f(a,a) \\
\mapsto_{dk} & y \ll a \quad (by\ idempotency) \\
\mapsto_{s} & y = a
\end{array}
$$

Failure:

$$
\begin{array}{ll}
 & f(x,x)\ [x=y] \ll f(a,b) \\
\mapsto\!\!\!\!\rightarrow_{es} & f(y,y)\ [x=y] \ll f(a,b) \\
\mapsto\!\!\!\!\rightarrow_{gc} & f(y,y) \ll f(a,b) \\
\mapsto_{dk} & y \ll a, y \ll b
\end{array}
$$

ZOOM on matching theories

"Untyped" Matching theories I

Empty theory $\mathbb{T}_\emptyset$ of equality (up to α-conversion)

$$\frac{\Vdash \mathcal{T}_1 = \mathcal{T}_2 \quad \Vdash \mathcal{T}_2 = \mathcal{T}_3}{\Vdash \mathcal{T}_1 = \mathcal{T}_3}\,(Trans) \qquad \frac{\Vdash \mathcal{T}_1 = \mathcal{T}_2}{\Vdash \mathcal{T}_2 = \mathcal{T}_1}\,(Symm)$$

$$\frac{\Vdash \mathcal{T}_1 = \mathcal{T}_2}{\Vdash \mathcal{T}_{3\lceil\mathcal{T}_1\rceil_p} = \mathcal{T}_{3\lceil\mathcal{T}_2\rceil_p}}\,(Context) \qquad \frac{}{\Vdash t = t}\,(Refl)$$

$\mathcal{T}_{1\lceil\mathcal{T}_2\rceil_p}$: term $\mathcal{T}_1$ with term $\mathcal{T}_2$ at position p

BACK

"Untyped" Matching theories II

Theory of Associativity $\mathbb{T}_{A(f)}$ (resp. Commutativity $\mathbb{T}_{C(f)}$) is defined as $\mathbb{T}_{\emptyset}$ plus:

$$\frac{}{\Vdash f(f(\mathcal{T}_1, \mathcal{T}_2), \mathcal{T}_3) = f(\mathcal{T}_1, f(\mathcal{T}_2, \mathcal{T}_3))}\,(Assoc)$$

$$\frac{}{\Vdash f(\mathcal{T}_1, \mathcal{T}_2) = f(\mathcal{T}_2, \mathcal{T}_1)}\,(Comm)$$

BACK

"Untyped" Matching theories III

Theory of Idempotency $\mathbb{T}_{I(f)}$ is defined as $\mathbb{T}_{\emptyset}$ plus the axiom

$$\frac{}{\Vdash f(\mathcal{T},\mathcal{T}) = t}\,(Idem)$$

Theory of Neutral Element $\mathbb{T}_{N(f^0)}$ is defined as $\mathbb{T}_{\emptyset}$ plus

$$\frac{}{\Vdash f(0,\mathcal{T}) = \mathcal{T}}\,(0{-}Left) \qquad \frac{}{\Vdash f(\mathcal{T},0) = \mathcal{T}}\,(0{-}Right)$$

BACK

"Untyped" Matching theories IV

The Theory of Stuck $\mathbb{T}_{\mathsf{stk}}$ is defined as $\mathbb{T}_{N(,^{\mathsf{stk}})}$ plus the axioms

$$\frac{\forall \theta_1, \theta_2,\ \forall C,\ A\theta_2 \mapsto\!\!\!\!\!\rightarrow_{\rho\infty\delta} C \ \Rightarrow\ C \not\equiv P\theta_1}{\Vdash [P \ll A]B = \mathsf{stk}}$$

$$\frac{}{\Vdash \mathsf{stk}\ \mathcal{T} = \mathsf{stk}}$$

Examples

$\Vdash [3 \ll 4]5 = \mathsf{stk}$

$\not\Vdash [3 \ll 3]5 = \mathsf{stk}$

$\not\Vdash [3 \ll \mathcal{X}]5 = \mathsf{stk}$

Detecting matching failures

BACK

"Untyped" Matching theories V

Theory of the Lambda Calculus of Objects $\mathbb{T}_{\lambda\mathcal{O}bj}$ is obtained by considering the symbol "," as associative and stk as its neutral element, $i.e.$:

$$\mathbb{T}_{\lambda\mathcal{O}bj} = \mathbb{T}_{A(,)} \cup \mathbb{T}_{\mathsf{stk}}$$

Theory of the Object Calculus $\mathbb{T}_{\varsigma\mathcal{O}bj}$ is obtained by considering the symbol "," as associative and commutative and stk as its neutral element, $i.e.$:

$$\mathbb{T}_{\varsigma\mathcal{O}bj} = \mathbb{T}_{A(,)} \cup \mathbb{T}_{C(,)} \cup \mathbb{T}_{\mathsf{stk}} = \mathbb{T}_{\lambda\mathcal{O}bj} \cup \mathbb{T}_{C(,)}$$

THEORIES RECORDS

The Matching Algorithm for $\mathbb{T}_\emptyset$

The matching substitution solving a matching equation can be computed by the following *matching reduction system*:

$$(Appl) \qquad (\mathcal{T}_1\ \mathcal{T}_2) \ll (\mathcal{T}_3\ \mathcal{T}_4) \rightsquigarrow \mathcal{T}_1 \ll \mathcal{T}_3 \wedge \mathcal{T}_2 \ll \mathcal{T}_4$$

$$(Struct) \quad (\mathcal{T}_1, \mathcal{T}_2) \ll (\mathcal{T}_3, \mathcal{T}_4) \rightsquigarrow \mathcal{T}_1 \ll \mathcal{T}_3 \wedge \mathcal{T}_2 \ll \mathcal{T}_4$$

Example

$f(\mathcal{X}, \mathcal{Y}) \ll f(a, b) \rightsquigarrow \mathcal{X} \ll a \wedge \mathcal{Y} \ll b$ successful

$f(\mathcal{X}, \mathcal{X}) \ll f(a, b) \rightsquigarrow \mathcal{X} \ll a \wedge \mathcal{Y} \ll b$ unsuccessful

BACK

ZOOM on objects

The Lambda Calculus of Objects $\lambda\mathcal{O}bj$

Abstract syntax

$$M, N \quad ::= \quad c \mid X \mid \lambda X.M \mid MN \mid$$

$$\langle\,\rangle \mid \langle M \longleftarrow n = N\rangle \mid \langle M \longleftarrow\!\!+\, n = N\rangle \mid M \Leftarrow n \mid$$

$$Sel(M, m, N)$$

Small-step semantics $(\leftarrow\!\!* \;=\; \longleftarrow$ or $\longleftarrow\!\!+\,)$

$$(Beta) \qquad (\lambda X.M)\, N \;\rightsquigarrow\; \{X/N\}M$$

$$(Sel) \qquad M \Leftarrow m \;\rightsquigarrow\; Sel(M, m, M)$$

$$(Next) \quad Sel(\langle M \leftarrow\!\!* \, n = N\rangle, m, P) \;\rightsquigarrow\; Sel(M, m, P) \qquad (m \neq n)$$

$$(Succ) \quad Sel(\langle M \leftarrow\!\!* \, n = N\rangle, n, P) \;\rightsquigarrow\; NP$$

Compiling $\lambda\mathcal{O}bj$ in ρ-calculus

$$
\begin{array}{lcl}
[\![c]\!] & \triangleq & c \\
[\![X]\!] & \triangleq & X \\
[\![\lambda X.M]\!] & \triangleq & X \twoheadrightarrow [\![M]\!] \\
[\![MN]\!] & \triangleq & [\![M]\!]\ [\![N]\!] \\
[\![< >]\!] & \triangleq & \mathsf{stk} \\
[\![< M \longleftarrow n = N >]\!] & \triangleq & kill_n([\![M]\!]), n \twoheadrightarrow [\![N]\!] \\
[\![< M \longleftarrow\!\!+ n = N >]\!] & \triangleq & [\![M]\!], n \twoheadrightarrow [\![N]\!] \\
[\![M\text{''} <= \text{''}m]\!] & \triangleq & [\![M]\!].m \triangleq [\![M]\!]\ m\ [\![M]\!] \\
[\![Sel(M,m,N)]\!] & \triangleq & [\![M]\!]\ m\ [\![N]\!]
\end{array}
$$

Theorem: If $M \rightsquigarrow_{\lambda\mathcal{O}bj} N$, then $[\![M]\!] \mapsto\!\!\!\!\twoheadrightarrow_{\rho\mathbf{dB}\mathbb{T}_{\lambda\mathcal{O}bj}} [\![N]\!]$.

Example with object update

ZOOM on Combinatory Reduction Systems

The *Combinatory Reduction Systems*

- The syntax: abstraction and metavariables

$$\mathcal{MT}_{\mathcal{CRS}} ::= \mathcal{X} \mid [\mathcal{X}]\mathcal{MT}_{\mathcal{CRS}} \mid \mathcal{F}(\mathcal{MT}_{\mathcal{CRS}}, \ldots, \mathcal{MT}_{\mathcal{CRS}}) \mid \mathcal{Z}(\mathcal{MT}_{\mathcal{CRS}}, \ldots, \mathcal{MT}_{\mathcal{CRS}})$$

The *Combinatory Reduction Systems*

- The syntax: abstraction and metavariables

$$\mathcal{MT}_{\mathcal{CRS}} ::= \mathcal{X} \mid [\mathcal{X}]\mathcal{MT}_{\mathcal{CRS}} \mid \mathcal{F}(\mathcal{MT}_{\mathcal{CRS}}, \ldots, \mathcal{MT}_{\mathcal{CRS}}) \mid \mathcal{Z}(\mathcal{MT}_{\mathcal{CRS}}, \ldots, \mathcal{MT}_{\mathcal{CRS}})$$

- The set of rewrite rules $\mathcal{R} = \{\ldots, L \Rightarrow R, \ldots\}$
  - L and R are closed metaterms;
  - L is of the form $f(A_1, \ldots, A_n)$ with $A_1, \ldots, A_n$ metatermes;
  - $\mathcal{MV}(L) \supseteq \mathcal{MV}(R)$;
  - L is a *Pattern* : $\forall \omega \; L_{\lceil Z(x_1,\ldots,x_n)\rceil_\omega} \quad x_i$ bound and distinct.

The *Combinatory Reduction Systems*

- The syntax: abstraction and metavariables

$$\mathcal{MT}_{\mathcal{CRS}} ::= \mathcal{X} \mid [\mathcal{X}]\mathcal{MT}_{\mathcal{CRS}} \mid \mathcal{F}(\mathcal{MT}_{\mathcal{CRS}}, \ldots, \mathcal{MT}_{\mathcal{CRS}}) \mid \mathcal{Z}(\mathcal{MT}_{\mathcal{CRS}}, \ldots, \mathcal{MT}_{\mathcal{CRS}})$$

- The set of rewrite rules $\mathcal{R} = \{\ldots, L \Rightarrow R, \ldots\}$
  - L and R are closed metaterms;
  - L is of the form $f(A_1, \ldots, A_n)$ with $A_1, \ldots, A_n$ metatermes;
  - $\mathcal{MV}(L) \supseteq \mathcal{MV}(R)$;
  - L is a *Pattern* : $\forall \omega \; L_{\lceil Z(x_1,\ldots,x_n)\rceil_\omega} \quad x_i$ bound and distinct.

- Assignment : $\sigma = \{(Z1, \xi_1), \ldots, (Z_n, \xi_n)\} \quad s.t. \quad Z_i \in \mathcal{MV}(L), |Z_i| = |\xi_i|$
- Substitute : $\xi = \lambda x_1 \ldots x_n.u \quad s.t \quad u \; is \; a \; CRS\text{-}term$
- Substitution at the meta-level.

Back to encodings

Translation examples

Abstraction:

$[x]Z(x) \quad \rightsquigarrow \quad x \rightarrowtriangle Z\ x$

Patterns:

$L = [x]g([y]Z(x,y)) \quad \rightsquigarrow \quad [\![L]\!] = x \rightarrowtriangle g(y \rightarrowtriangle (Z\ x\ y))$

Assignment into rho-substitution:

$\sigma = \{Z, \lambda x.x\} \quad \rightsquigarrow \quad [\![\sigma]\!] = \{Z/x \rightarrowtriangle x\}$

Rewrite rules into rho-terms:

$Beta :$ $\qquad$ $[\![Beta]\!] :$

$App([x]Z(x), Z') \Rightarrow Z(Z') \quad \rightsquigarrow \quad App(x \rightarrowtriangle Z\ x, Z') \rightarrowtriangle Z\ Z'$

Back to encodings

ZOOM on P^2 S

Lambda Calculi *à la* Church and Logics

- Lambda abstractions are decorated with types, $e.g.\ \lambda x{:}\sigma.M$
- Type Systems λ_i $vs.$ Logic Systems $\mathcal{L}_i$ via the well-known Curry-Howard Isomorphism *"proofs-as-(λ)-terms & propositions-as-types"*
- Each logical system $\mathcal{L}_i$ correspond to a type system λ_i, and for every formula φ

$$\vdash_{\mathcal{L}_i} \varphi \implies \exists M.\ \Gamma \vdash_{\lambda_i} M : [\![\varphi]\!]$$

Lambda Calculi *à la* Church and Logics

- Lambda abstractions are decorated with types, $e.g.\ \lambda x{:}\sigma.M$
- Type Systems λ_i $vs.$ Logic Systems $\mathcal{L}_i$ via the well-known Curry-Howard Isomorphism *"proofs-as-(λ)-terms & propositions-as-types"*
- Each logical system $\mathcal{L}_i$ correspond to a type system λ_i, and for every formula φ

$$\vdash_{\mathcal{L}_i} \varphi \implies \exists M.\ \Gamma \vdash_{\lambda_i} M : [\![\varphi]\!]$$

- Γ contains the types of the free variables of M, and $[\![\varphi]\!]$ is a canonical interpretation of φ in λ_i

Lambda Calculi *à la* Church and Logics

- Lambda abstractions are decorated with types, *e.g.* $\lambda x{:}\sigma.M$
- Type Systems λ_i *vs.* Logic Systems $\mathcal{L}_i$ via the well-known Curry-Howard Isomorphism *"proofs-as-(λ)-terms & propositions-as-types"*
- Each logical system $\mathcal{L}_i$ correspond to a type system λ_i, and for every formula φ

$$\vdash_{\mathcal{L}_i} \varphi \implies \exists M.\ \Gamma \vdash_{\lambda_i} M : [\![\varphi]\!]$$

- Γ contains the types of the free variables of M, and $[\![\varphi]\!]$ is a canonical interpretation of φ in λ_i
- β-reduction $(\lambda x{:}\sigma.M)\,N \twoheadrightarrow_\beta M\{N/x\}$ in λ_i as cut-elimination in $\mathcal{L}_i$

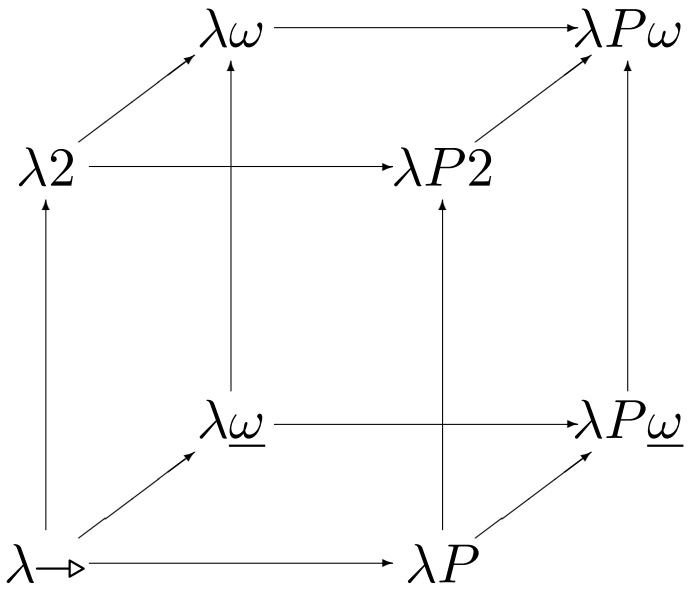
Lambda Calculi *à la* Church and Logics

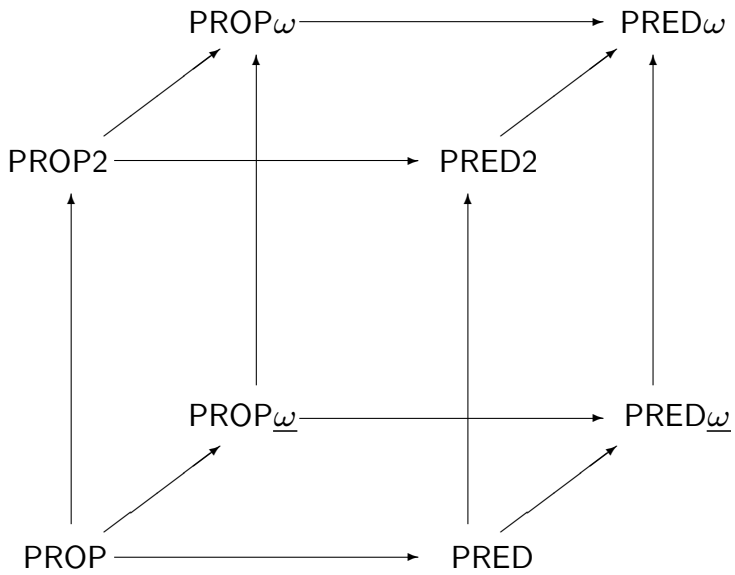
- Lambda abstractions are decorated with types, *e.g.* $\lambda x{:}\sigma.M$
- Type Systems λ_i *vs.* Logic Systems $\mathcal{L}_i$ via the well-known Curry-Howard Isomorphism *"proofs-as-(λ)-terms & propositions-as-types"*
- Each logical system $\mathcal{L}_i$ correspond to a type system λ_i, and for every formula φ

$$\vdash_{\mathcal{L}_i} \varphi \implies \exists M.\ \Gamma \vdash_{\lambda_i} M : [\![\varphi]\!]$$

- Γ contains the types of the free variables of M, and $[\![\varphi]\!]$ is a canonical interpretation of φ in λ_i
- β-reduction $(\lambda x{:}\sigma.M)\,N \twoheadrightarrow_\beta M\{N/x\}$ in λ_i as cut-elimination in $\mathcal{L}_i$
- Subject Reduction Theorem as a correction criterion for cut-elimination BACK

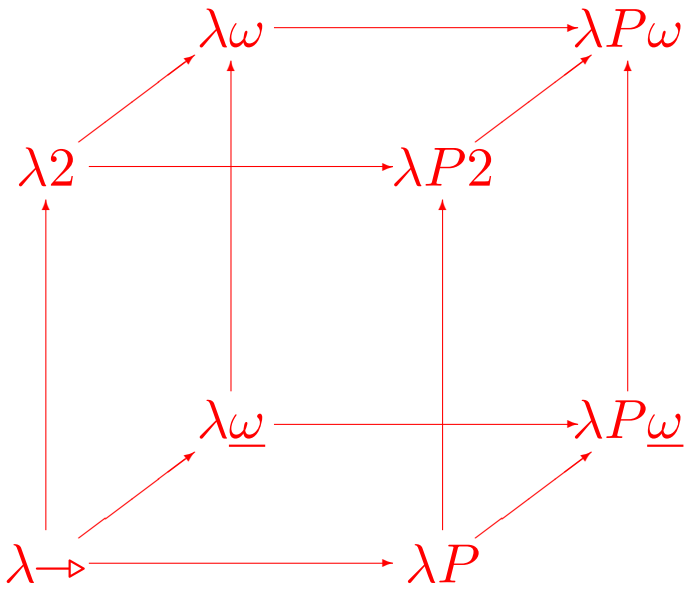
The Famous Barendregt's λ-cube and its 8 logic systems

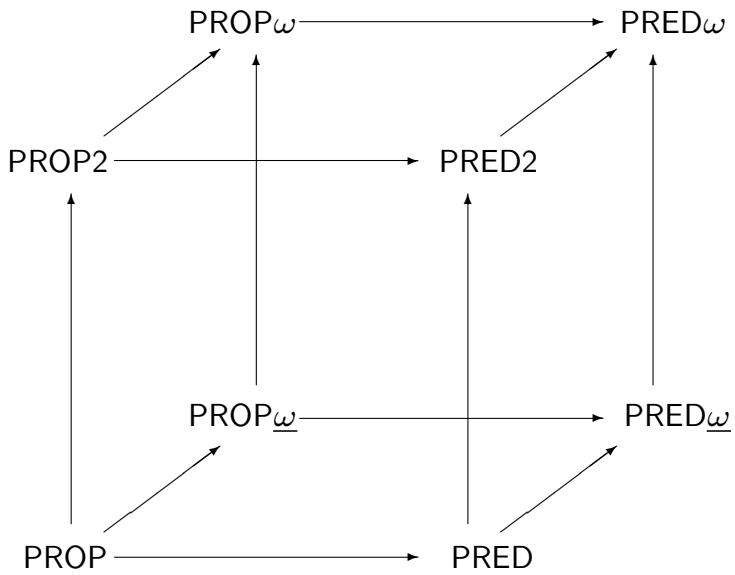




PROP	proposition logic	PRED	predicate logic
PROP2	second-order PROP	PRED2	second-order PRED
PROP$\underline{\omega}$	weakly higher-order PROP	PRED$\underline{\omega}$	weakly higher-order PRED
PROPω	higher-order PROP	PREDω	higher-order PRED

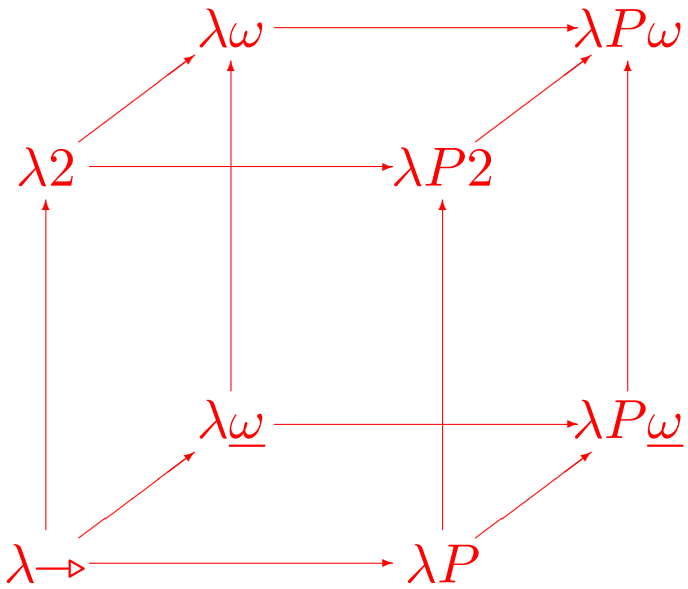
The Famous Barendregt's λ-cube and its 8 logic systems

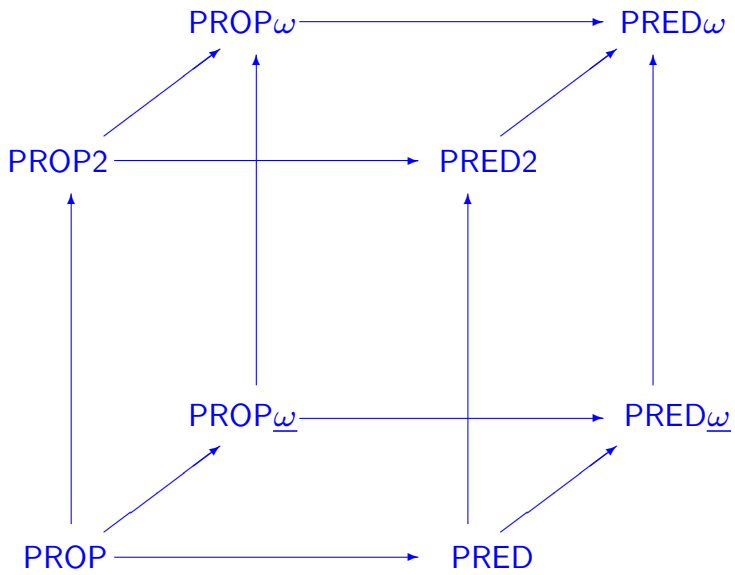




PROP	proposition logic	PRED	predicate logic
PROP2	second-order PROP	PRED2	second-order PRED
PROP$\underline{\omega}$	weakly higher-order PROP	PRED$\underline{\omega}$	weakly higher-order PRED
PROPω	higher-order PROP	PREDω	higher-order PRED

The Famous Barendregt's λ-cube and its 8 logic systems





PROP	proposition logic	PRED	predicate logic
PROP2	second-order PROP	PRED2	second-order PRED
PROP$\underline{\omega}$	weakly higher-order PROP	PRED$\underline{\omega}$	weakly higher-order PRED
PROPω	higher-order PROP	PREDω	higher-order PRED

From Barendregt's λ-cube to Pure Type Systems (PTSs)

- Further generalization of various type systems invented independently by Berardi and Terlouw in '89
- Many systems of typed λ-calculus $\grave{a}\ la$ Church can be seen as PTSs
- One of the success of PTSs is concerned with logics: the 8 logical systems can be described/generalised as a simple unique PTS

BACK

From Barendregt's λ-cube to Pure Type Systems (PTSs)

- Further generalization of various type systems invented independently by Berardi and Terlouw in '89
- Many systems of typed λ-calculus $\grave{a}\ la$ Church can be seen as PTSs
- One of the success of PTSs is concerned with logics: the 8 logical systems can be described/generalised as a simple unique PTS
- Another one is the compactness of the notation of PTSs which greatly allows to factorise and simplify proofs in metatheory, in the style *"one theorem fits all!"*

BACK

From Barendregt's λ-cube to Pure Type Systems (PTSs)

- Further generalization of various type systems invented independently by Berardi and Terlouw in '89
- Many systems of typed λ-calculus $\grave{a}\ la$ Church can be seen as PTSs
- One of the success of PTSs is concerned with logics: the 8 logical systems can be described/generalised as a simple unique PTS
- Another one is the compactness of the notation of PTSs which greatly allows to factorise and simplify proofs in metatheory, in the style *"one theorem fits all!"*
- Examples of well-known PTSs are λHOL, $\lambda PRED$, λCC ($a.k.a.$ the λ-cube)

BACK

More Pragmatically ...

- Almost all proof assistants and relating metalanguages based on the proposition-as-type principle, have a firm theoretical basis in logics represented via PTSs

More Pragmatically ...

- Almost all proof assistants and relating metalanguages based on the proposition-as-type principle, have a firm theoretical basis in logics represented via PTSs

- AUTOMATH, NUPRL, HOL, LEGO, (TW)ELF, AGDA, ISABELLE, COQ, MIZAR, ACL2, PVS ...

More Pragmatically ...

- Almost all proof assistants and relating metalanguages based on the proposition-as-type principle, have a firm theoretical basis in logics represented via PTSs
- AUTOMATH, NUPRL, HOL, LEGO, (TW)ELF, AGDA, ISABELLE, COQ, MIZAR, ACL2, PVS ...
- The degree of automatization of such proof assistants depends also on the capability of simplifying/reduce terms

More Pragmatically ...

- Almost all proof assistants and relating metalanguages based on the proposition-as-type principle, have a firm theoretical basis in logics represented via PTSs
- AUTOMATH, NUPRL, HOL, LEGO, (TW)ELF, AGDA, ISABELLE, COQ, MIZAR, ACL2, PVS ...
- The degree of automatization of such proof assistants depends also on the capability of simplifying/reduce terms
- The "Poincaré principle" can be $(\beta\iota\delta)$-reductions, structural well-founded recursion, provable equality, or some arbitrary notion of reduction

More Pragmatically ...

- Almost all proof assistants and relating metalanguages based on the proposition-as-type principle, have a firm theoretical basis in logics represented via PTSs
- AUTOMATH, NUPRL, HOL, LEGO, (TW)ELF, AGDA, ISABELLE, COQ, MIZAR, ACL2, PVS ...
- The degree of automatization of such proof assistants depends also on the capability of simplifying/reduce terms
- The "Poincaré principle" can be $(\beta\iota\delta)$-reductions, structural well-founded recursion, provable equality, or some arbitrary notion of reduction
- The more reductions principles you have in the metalanguage, the more "powerful" the proof assistant is ... BACK

P^{2}T S: One-step, Many-steps, Congruence

Let Ctx$[-]$ be any term $\mathcal{T}$ with a "single hole" inside, and let Ctx$[A]$ be the result of filling the hole with the term A;

1. the one-step evaluation $\mapsto$ is defined by the following inference rule, where $\rightarrow_{\rho\sigma\delta} \equiv \rightarrow_\rho \cup \rightarrow_\sigma \cup \rightarrow_\delta$:

$$\frac{A \rightarrow_{\rho\sigma\delta} B}{\text{Ctx}[A] \mapsto_{\rho\sigma\delta} \text{Ctx}[B]}\,(\text{Ctx}[-])$$

2. the many-step evaluation $\mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta}$ and congruence relation $=_{\rho\sigma\delta}$ are respectively defined as the reflexive-transitive and reflexive-symmetric-transitive closure of $\mapsto_{\rho\sigma\delta}$

BACK

Abbreviations and Priorities

$(A_i)^{i=1\ldots n} \triangleq A_1, \cdots, A_n$ structure/object

$A.B \triangleq A \bullet B \bullet A$ Kamin's self-application

Abbreviations and Priorities

$(A_i)^{i=1\ldots n} \triangleq A_1, \cdots, A_n$ structure/object

$A.B \triangleq A \bullet B \bullet A$ Kamin's self-application

Operator	Associate	Priority
$\_\,,\,\_$	Right	$>$
$\checkmark\_:\_.\_$	Right	$>$
$[\_ \ll \_\ \_].\_$	Right	$>$
$\_\bullet\_$	Left	$>$

... still substitutions

- We let

$$\mathcal{D}om(\sigma) = \{X_1, \ldots, X_m\}$$

and

$$\mathcal{C}o\mathcal{D}om(\Delta) = \bigcup_{i=1\ldots m} \mathcal{FV}ar(A_i)$$

- A substitution σ is independent from Δ, written $\sigma \not\triangleleft \Delta$ if

$$\mathcal{D}om(\sigma) \cap \mathcal{D}om(\Delta) = \emptyset$$

and

$$\mathcal{C}o\mathcal{D}om(\sigma) \cap \mathcal{D}om(\Delta) = \emptyset$$

Termination of $\mathcal{A}lg$

- The relation $\rightsquigarrow\!\!\!\!\rightarrow$ is defined as the reflexive, transitive and compatible closure of $\rightsquigarrow$
- If T $\rightsquigarrow\!\!\!\!\rightarrow$ T′, with T′ a matching system in solved form then, we say that the matching algorithm $\mathcal{A}lg$ (taking as input the system T) succeeds
- The matching algorithm is clearly terminating (since all rules decrease the size of terms) and deterministic (no critical pairs), and of course, it works modulo α-conversion and Barendregt's hygiene-convention
- Starting form a given solved matching system of the form

$$\mathsf{T} \stackrel{\triangle}{=} \bigwedge_{\mathbf{i=0\ldots n}} \mathbf{X_i} \prec\!\!\!\!\prec^{\mathbf{\Sigma}}_{\mathbf{\Delta_i}} \mathbf{A_i} \bigwedge_{\mathbf{j=0\ldots m}} \mathbf{a_j} \prec\!\!\!\!\prec^{\mathbf{\Sigma}}_{\mathbf{\Delta_j}} \mathbf{a_j}$$

the corresponding substitution $\{A_1/X_1 \cdots A_n/X_n\}$ is exhibited.

Functional P^{2}T S

We require all specifications to be *functional*, *i.e.* for every $s_1, s_2, s_2', s_3, s_3' \in \mathcal{S}$, the following holds:

$$\begin{array}{llllll} (s_1, s_2) \in \rangle A & \text{and} & (s_1, s_2') \in \rangle A & \text{implies} & s_2 \equiv s_2' \\ (s_1, s_2, s_3) \in \mathcal{R} & \text{and} & (s_1, s_2, s_3') \in \mathcal{R} & \text{implies} & s_3 \equiv s_3'. \end{array}$$

Furthermore, we let $\mathcal{S}^\top$ denote the set of *topsorts*, *i.e.*

$$\mathcal{S}^\top = \{s \in \mathcal{S} \mid \forall s' \in \mathcal{S}.\ (s, s') \notin \rangle A\}$$

and define a variant of delayed matching constraint as follows:

$$[A \ll_\Delta C]^\top.B = \begin{cases} B & \text{if } B \in \mathcal{S}^\top \\ [A \ll_\Delta C].B & \text{otherwise} \end{cases}$$

BACK

ZOOM on DIMPRO

Simple Functional Fixpoint

$\Gamma \equiv \mathsf{fix}{:}(\mathsf{b} \rightarrowtriangle \mathsf{b}) \rightarrowtriangle \mathsf{b}$, and $\Delta \equiv X{:}\mathsf{b} \rightarrowtriangle \mathsf{b}$, and
$\Omega \triangleq \mathsf{fix}(X) \rightarrowtriangle_{\Delta} X\ \mathsf{fix}(X)$. We typecheck $\Gamma \vdash_{\mathsf{A}} \Omega\ \mathsf{fix}(\Omega) : \mathsf{b}$

$$\dfrac{\dfrac{\dfrac{\Gamma,\Delta \vdash_{\mathsf{A}} \mathsf{fix} : (\mathsf{b} \rightarrowtriangle \mathsf{b}) \rightarrowtriangle \mathsf{b} \quad \Gamma,\Delta \vdash_{\mathsf{A}} X : \mathsf{b} \rightarrowtriangle \mathsf{b}}{\Gamma,\Delta \vdash_{\mathsf{A}} X : \mathsf{b} \rightarrowtriangle \mathsf{b}} \quad \dfrac{\Gamma,\Delta \vdash_{\mathsf{A}} \mathsf{fix}(X) : \mathsf{b}}{\Gamma,\Delta \vdash_{\mathsf{A}} \mathsf{fix}(X) : \mathsf{b}}}{\Gamma,\Delta \vdash_{\mathsf{A}} X\ \mathsf{fix}(X) : \mathsf{b}} \quad \dfrac{\Gamma \vdash_{\mathsf{A}} \mathsf{fix} : (\mathsf{b} \rightarrowtriangle \mathsf{b}) \rightarrowtriangle \mathsf{b} \quad \Gamma \vdash_{\mathsf{A}} \Omega : \mathsf{b} \rightarrowtriangle \mathsf{b}}{\Gamma \vdash_{\mathsf{A}} \mathsf{fix}(\Omega) : \mathsf{b}}}{\Gamma \vdash_{\mathsf{A}} \Omega\ \mathsf{fix}(\Omega) : \mathsf{b}}$$

The reader can verify that our interpreter diverges, $i.e.$

$$\dfrac{\begin{array}{c}\infty \\ \vdots\end{array}}{\emptyset \vdash \Omega\ \mathsf{fix}(\Omega) \Downarrow_{\mathsf{val}} \mathsf{stack\ overflow}}$$

Encoding Term Rewrite Systems

- However, a suitable recursion operator in the style of the object-oriented encoding allows us to simulate the *global* behavior of a TRS Let the constants rec and add, and let

$$obj.meth \triangleq ((obj\, meth)\, obj)$$

$$plus \triangleq \left(\begin{array}{l} rec(S) \twoheadrightarrow add(0, Y) \twoheadrightarrow Y, \\ rec(S) \twoheadrightarrow add(suc(X), Y) \twoheadrightarrow suc\,(S.rec\ \ add(X, Y)) \end{array} \right)$$

A Nice Circle ... and back

- $$\frac{\sigma_0 \cdot \rho \vdash F \Downarrow_{\mathsf{val}} f \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} a \cdot \sigma_2 \qquad \sigma_2 \vdash \langle f \cdot a \rangle \Downarrow_{\mathsf{call}} v \cdot \sigma_3}{\sigma_0 \cdot \rho \vdash F\ A \Downarrow_{\mathsf{val}} v \cdot \sigma_3}\ (Red{-}\rho_{\mathsf{v}})$$

-
```
(define-method (Eval::value t::App env) ;()
   (with-access::App t (F A)
      (let ( (f (Eval F env)) )
         (let ( (a (Eval A env)) )
            (call f a) ))))
```

-
```
Mutual Inductive eval : expr->env->store->value->store->Prop :=
.....
 | evalApp: (F:expr)(e:env)(s:store)(f:value)(s1:store)
       (eval F e s f s1) -> (A:expr)(a:value)(s2:store)
       (eval A e s1 a s2) -> (v:value)(s3:store)
       (call f a s2 v s3) ->
       (eval (App F A) e s v s3)
.....
```