I217: Functional Programming

3. Term Rewriting

Kazuhiro Ogata, Canh Minh Do

Roadmap

- Pattern Match
  - Substitution
- Sub-terms
  - Positions in terms
- Rewrite rules
  - Redexes & Contracts
- Rewriting
  - One step rewrite, Reduction & Trace

Pattern Match

Let us consider the module LISTNAT:

```
mod! NATLIST {
 -- imports
 pr(NAT-ERR)
 -- signature
 [Nil NnNatList < NatList]
 op nil : -> Nil {constr} .
 op _|_ : Nat NatList -> NnNatList {constr} .
 op hd : Nil -> ErrNat .
 op hd : NnNatList -> Nat .
 op hd : NatList -> Nat&Err .
 op tl : NatList -> NatList .
 op _@_ : NatList NatList -> NatList .
 ...
-- CafeOBJ vars
 vars X Y : Nat .
 vars L L2 : NatList .
 -- equations
 -- hd
 eq hd(nil) = errNat .
 eq hd(X | L) = X .
 -- tl
 eq tl(nil) = nil .
 eq tl(X | L) = L .
 -- _@_
 eq nil @ L2 = L2 .
 eq (X | L) @ L2 = X | (L @ L2) .
 ... }
```


Pattern Match

hd(X | L) is a term whose least sort is Nat.

hd(2 | 1 | 0 | nil) is a term whose least sort is Nat.

Seemingly, the two terms are different.

By replacing X that is a term of Nat and L that is a term of NatList with 2 that is a term of Nat as well as NzNat and 1 | 0 | nil that is a term of NatList as well as NnNatList, however, hd(X | L) becomes hd(2 | 1 | 0 | nil).

hd(2 | 1 | 0 | nil) is called an instance of hd(X | L) or can match hd(X | L) with the replacement of the variables with the terms.

Pattern Match

Such a replacement is called a *substitution*.

A substitution is a function from variables to terms that preserves sorts.

The substitution σ_{ex} used as the example is the function from $\{X, Y, L, L2\}$ to the disjoint union of the sets of terms of Nat and terms of NatList such that it maps X, Y, L and $L2$ to 2, Y, $1 \mid 0 \mid nil$ and $L2$.

$$\sigma_{ex}(X) = 2 \quad \sigma_{ex}(Y) = Y \quad \sigma_{ex}(L) = 1 \mid 0 \mid nil \quad \sigma_{ex}(L2) = L2$$

σ_{ex} may be expressed as follows:

$$\{X \leftarrow 2, L \leftarrow 1 \mid 0 \mid nil\}$$

Pattern Match

A substitution σ can be naturally extended as a function from terms to terms as follows:

for a non-variable term $f(t_1, \ldots, t_n)$,

$$\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$$

$$\begin{aligned}\sigma_{ex}(hd(X \mid L)) &= hd(\sigma_{ex}(X \mid L)) \\ &= hd(\sigma_{ex}(X) \mid \sigma_{ex}(L)) \\ &= hd(2 \mid 1 \mid 0 \mid nil)\end{aligned}$$

Pattern Match

Given a term t and a ground term s, the *pattern match* between t and s is the problem to decide whether there exists a substitution σ such that $\sigma(t) = s$.

t may be called a pattern.

If that is the case, s is called an instance of the pattern t and can match the pattern t with the substitution σ.

Pattern Match

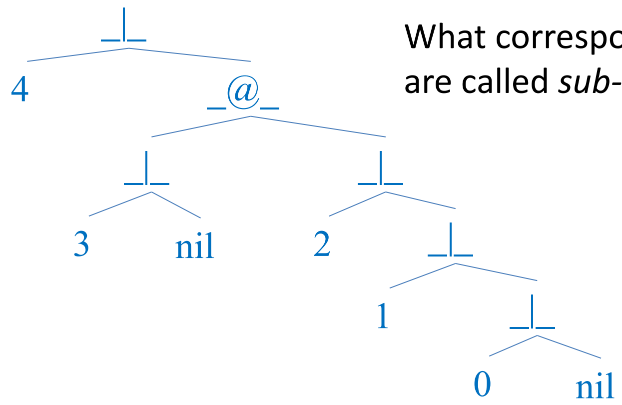
Can the ground term match the pattern? If yes, what is the substitution?

1. tl(1 | 0 | nil) & tl(X | L)
2. tl(tl(1 | 0 | nil)) & tl(X | L)
3. (4 | 3 | nil) @ (2 | 1 | 0 | nil) & (X | L) @ L2
4. nil @ (2 | 1 | 0 | nil) & nil @ L2
5. 4 | ((3 | nil) @ (2 | 1 | 0 | nil)) & (X | L) @ L2
6. 4 | 3 | (nil @ (2 | 1 | 0 | nil)) & nil @ L2

Sub-terms

A term can be expressed as a tree structure.

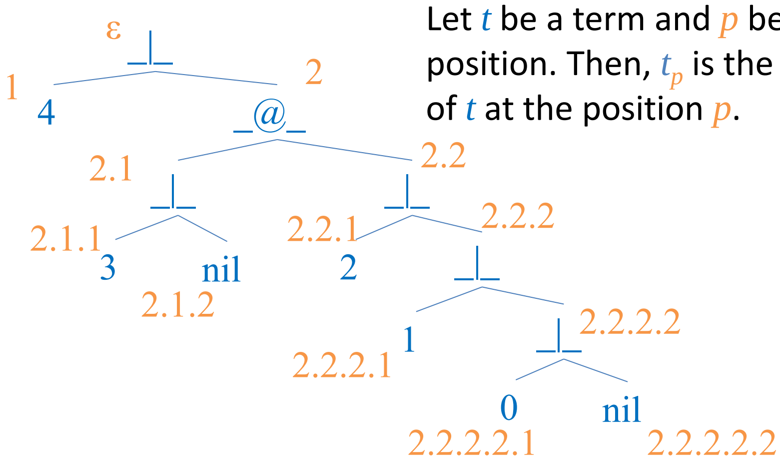
4 | ((3 | nil) @ (2 | 1 | 0 | nil)) is expressed as follows:



What correspond to sub-trees are called *sub-terms* of the term.

Sub-terms

Sub-terms of a term can be identified by *positions*.



Let t be a term and p be a position. Then, t_p is the sub-term of t at the position p.

Sub-terms

Let t be $4 \mid ((3 \mid \text{nil})\ @\ (2 \mid 1 \mid 0 \mid \text{nil}))$.

t_ε is $4 \mid ((3 \mid \text{nil})\ @\ (2 \mid 1 \mid 0 \mid \text{nil}))$.

t_1 is 4. t_2 is $(3 \mid \text{nil})\ @\ (2 \mid 1 \mid 0 \mid \text{nil})$.

$t_{2.1}$ is $3 \mid \text{nil}$. $t_{2.2}$ is $2 \mid 1 \mid 0 \mid \text{nil}$.

$t_{2.1.1}$ is 3. $t_{2.1.2}$ is nil. $t_{2.2.1}$ is 2. $t_{2.2.2}$ is $1 \mid 0 \mid \text{nil}$.

$t_{2.2.2.1}$ is 1. $t_{2.2.2.2}$ is $0 \mid \text{nil}$.

$t_{2.2.2.2.1}$ is 0. $t_{2.2.2.2.2}$ is nil.

Sub-terms

For a term t, a position p and a term s such that the least sort of t_p is a sort of s, $t_p[s]$ is t in which t_p is replaced with s.

Let t be $(4 \mid 3 \mid \text{nil})\ @\ (2 \mid 1 \mid 0 \mid \text{nil})$.

$t_\varepsilon[4 \mid ((3 \mid \text{nil})\ @\ (2 \mid 1 \mid 0 \mid \text{nil}))]$ is $4 \mid ((3 \mid \text{nil})\ @\ (2 \mid 1 \mid 0 \mid \text{nil}))$.

Let t be $4 \mid ((3 \mid \text{nil})\ @\ (2 \mid 1 \mid 0 \mid \text{nil}))$.

$t_2[3 \mid (\text{nil}\ @\ (2 \mid 1 \mid 0 \mid \text{nil}))]$ is $4 \mid 3 \mid (\text{nil}\ @\ (2 \mid 1 \mid 0 \mid \text{nil}))$.

Let t be $4 \mid 3 \mid (\text{nil}\ @\ (2 \mid 1 \mid 0 \mid \text{nil}))$.

$t_{2.2}[2 \mid 1 \mid 0 \mid \text{nil}]$ is $4 \mid 3 \mid 2 \mid 1 \mid 0 \mid \text{nil}$.

Rewrite Rules

A rewrite rule is a pair (l,r) of terms l and r such that the least sort of l is a sort of r, l is not a single variable, each variable occurring in r occurs in l.

A rewrite rule (l,r) may be expressed as $l \rightarrow r$.

$$\text{nil} \mathbin{@} L2 \rightarrow L2 \qquad (@1)$$
$$(X \mid L) \mathbin{@} L2 \rightarrow X \mid (L \mathbin{@} L2) \qquad (@2)$$

A *term rewriting system* (*TRS*) is a set of rewrite rules.

Rewrite Rules

For a TRS R,

an instance $\sigma(l)$ of the left-hand side of a rewrite rule $l \rightarrow r \in R$ for some substitution σ is a *redex* (reducible expression) with respect to R;

an instance $\sigma(r)$ of the right-hand side of a rewrite rule $l \rightarrow r \in R$ for some substitution σ is a *contract* with respect to R.

Rewrite Rules

nil @ (2 | 1 | 0 | nil) is a redex with respect to $R_{@}$.

2 | 1 | 0 | nil is a contract with respect to $R_{@}$.

(3 | nil) @ (2 | 1 | 0 | nil) is a redex with respect to $R_{@}$.

3 | (nil @ (2 | 1 | 0 | nil)) is a contract with respect to $R_{@}$.

Let $R_{@}$ be {(@1), (@2)} such that

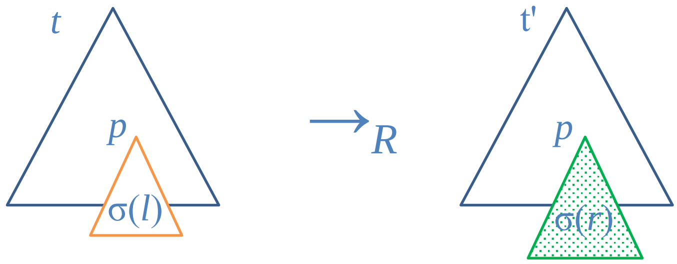
nil @ L2 → L2	(@1)
(X | L) @ L2 → X | (L @ L2)	(@2)

Rewriting

One step rewrite with respect to a TRS R is a pair (t, t') of ground terms t and t' such that there exist a rewrite rule $l \rightarrow r \in R$, a substitution σ and a position p such that t_p is $\sigma(l)$ and t'_p is $\sigma(r)$.

(t, t') may be written as $t \rightarrow_R t'$.

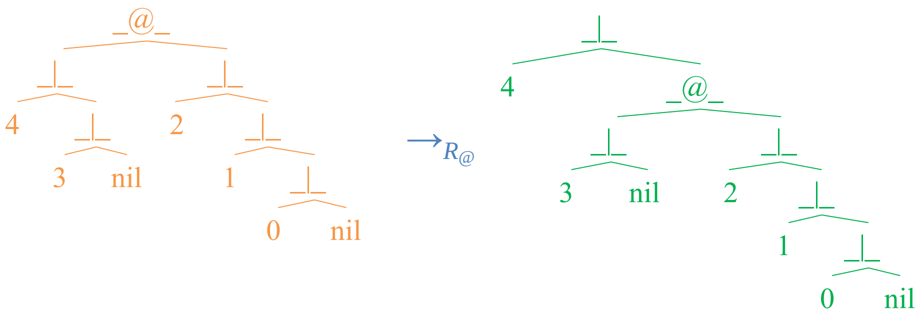
R may be omitted if it is clear from context.



Rewriting

$\underline{(4 \mid 3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil})}$

$\rightarrow_{R_@}$ $4 \mid ((3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil}))$ by $(@2)$



Let $R_@$ be $\{(@1), (@2)\}$ such that

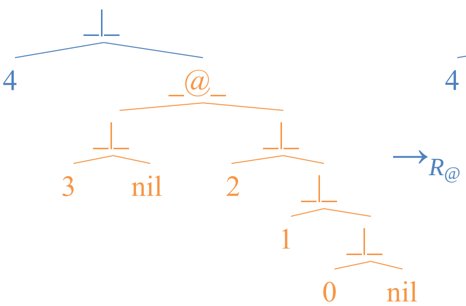
$\text{nil} \mathbin{@} \text{L2} \rightarrow \text{L2}$ $(@1)$

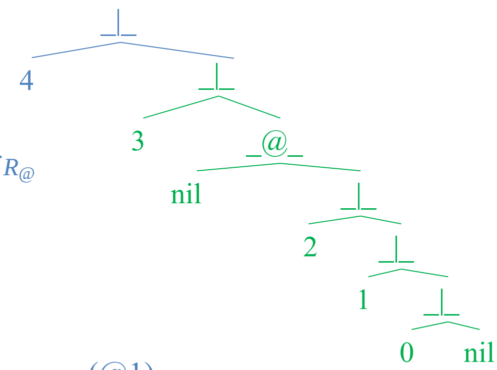
$(\text{X} \mid \text{L}) \mathbin{@} \text{L2} \rightarrow \text{X} \mid (\text{L} \mathbin{@} \text{L2})$ $(@2)$

Rewriting

$4 \mid (\underline{(3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil})})$

$\rightarrow_{R_@}$ $4 \mid 3 \mid (\text{nil} \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil}))$ by $(@2)$





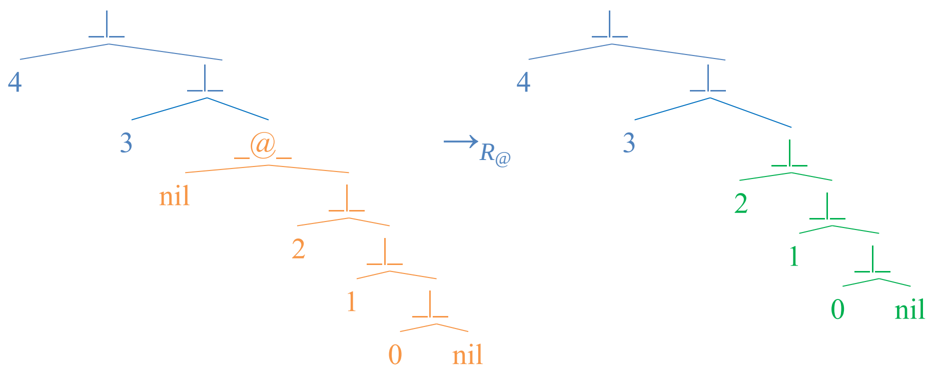
$R_@$ is $\{(@1), (@2)\}$.

$\text{nil} \mathbin{@} \text{L2} \rightarrow \text{L2}$ $(@1)$

$(\text{X} \mid \text{L}) \mathbin{@} \text{L2} \rightarrow \text{X} \mid (\text{L} \mathbin{@} \text{L2})$ $(@2)$

Rewriting

$4 \mid 3 \mid (\underline{\text{nil} \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil})}) \;\rightarrow_{R_{@}}\; 4 \mid 3 \mid 2 \mid 1 \mid 0 \mid \text{nil}$ by $(@1)$



$R_{@}$ is $\{(@1), (@2)\}$.

$\text{nil} \mathbin{@} L2 \rightarrow L2$ $(@1)$

$(X \mid L) \mathbin{@} L2 \rightarrow X \mid (L \mathbin{@} L2)$ $(@2)$

Rewriting

Rewriting $\rightarrow^{*}_{\mathrm{R}}$ with respect to a TRS R is a reflexive and transitive closure of $\rightarrow_{\mathrm{R}}$.

$(4 \mid 3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil}) \;\rightarrow^{*}_{R_{@}}\; (4 \mid 3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil})$

$(4 \mid 3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil})$

$\rightarrow^{*}_{R_{@}}\; 4 \mid ((3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil}))$

$(4 \mid 3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil}) \;\rightarrow^{*}_{R_{@}}\; 4 \mid 3 \mid (\text{nil} \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil}))$

$(4 \mid 3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil}) \;\rightarrow^{*}_{R_{@}}\; 4 \mid 3 \mid 2 \mid 1 \mid 0 \mid \text{nil}$

Rewriting

Reduction of a ground term t with respect to R is rewriting $t \to^*_R t'$ such that t' does not have any redexes with respect to R.

Reduction of a ground term $(4 \mid 3 \mid \text{nil})\ @\ (2 \mid 1 \mid 0 \mid \text{nil})$ with respect to $R_@$ is

$(4 \mid 3 \mid \text{nil})\ @\ (2 \mid 1 \mid 0 \mid \text{nil}) \ \to^*_{R_@}\ 4 \mid 3 \mid 2 \mid 1 \mid 0 \mid \text{nil}$

This is what is done by the command **red** of CafeOBJ, although the result of **red** has the least sort, where equations are used as rewrite rules.

Note that equations should satisfy the conditions for rewrite rules to use the equations as rewrite rules.

Rewriting

The *trace* of reduction of a ground term t_0 with respect to R is a series of one step rewrites.

$$t_0 \xrightarrow{(rl_0)}_R \cdots \xrightarrow{(rl_{i-1})}_R t_i \xrightarrow{(rl_i)}_R t_{i+1} \xrightarrow{(rl_{i+1})}_R \cdots \xrightarrow{(rl_{n-1})}_R t_n$$

such that $t_0 \to^*_R t_n$ is reduction with respect to R and for each one step rewrite $t_i \to_R t_{i+1}$ the redex concerned in t_i is underlined and the rewrite rule (rl_i) used is clearly identified.

Rewriting

The trace of reduction of (4 | 3 | nil) @ (2 | 1 | 0 | nil) with respect to $R_@$ is

$\underline{(4 \mid 3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil})}$

$\rightarrow_{R_@}$	$4 \mid (\underline{(3 \mid \text{nil}) \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil})})$	by (@2)
$\rightarrow_{R_@}$	$4 \mid 3 \mid (\underline{\text{nil} \mathbin{@} (2 \mid 1 \mid 0 \mid \text{nil})})$	by (@2)
$\rightarrow_{R_@}$	$4 \mid 3 \mid 2 \mid 1 \mid 0 \mid \text{nil}$	by (@1)

Let $R_@$ be {(@1), (@2)} such that

nil @ L2 → L2	(@1)
(X | L) @ L2 → X | (L @ L2)	(@2)

Rewriting

We can ask CafeOBJ to display the trace of reduction of a ground term with respect to a TRS as follows:

```
set trace on
open NATLIST .
  red (4 | 3 | nil) @ (2 | 1 | 0 | nil) .
close
set trace off
```

Rewriting

```
-- reduce in %NATLIST : ((4 | (3 | nil)) @ (2 | (1 | (0 | nil)))):NatList
1>[1] rule: eq ((X:Nat | L:NatList) @ L2:NatList) = (X | (L @ L2))
    { X:Nat |-> 4, L2:NatList |-> (2 | (1 | (0 | nil))), L:NatList |-> (3 | nil) }
1<[1] ((4 | (3 | nil)) @ (2 | (1 | (0 | nil)))):NatList --> (4 | ((3 | nil) @ (2 | (1 | (0 | nil))))):NnNatList
```

The rewrite rule used

```
1>[2] rule: eq ((X:Nat | L:NatList) @ L2:NatList) = (X | (L @ L2))
    { X:Nat |-> 3, L2:NatList |-> (2 | (1 | (0 | nil))), L:NatList |-> nil }
```

The substitution

```
1<[2] ((3 | nil) @ (2 | (1 | (0 | nil)))):NatList --> (3 | (nil @ (2 | (1 | (0 | nil))))):NnNatList
```

The redex The contract

```
1>[3] rule: eq (nil @ L2:NatList) = L2
    { L2:NatList |-> (2 | (1 | (0 | nil))) }
1<[3] (nil @ (2 | (1 | (0 | nil)))):NatList --> (2 | (1 | (0 | nil))):NnNatList

(4 | (3 | (2 | (1 | (0 | nil))))):NnNatList
```

Appearances are different, but this contains all information about the trace. Moreover, the substitution used fro each one step rewrite and the least sort of each term are shown.

Rewriting

We can ask CafeOBJ to partially display the trace of reduction of a ground term with respect to a TRS as follows:

```
set trace whole on
open NATLIST .
  red (4 | 3 | nil) @ (2 | 1 | 0 | nil) .
close
set trace whole off
```

Rewriting

-- reduce in %NATLIST : ((4 | (3 | nil)) @ (2 | (1 | (0 | nil)))):NatList
[1]: ((4 | (3 | nil)) @ (2 | (1 | (0 | nil)))):NatList
---> (4 | ((3 | nil) @ (2 | (1 | (0 | nil))))):NnNatList
[2]: (4 | ((3 | nil) @ (2 | (1 | (0 | nil))))):NnNatList
---> (4 | (3 | (nil @ (2 | (1 | (0 | nil)))))):NnNatList
[3]: (4 | (3 | (nil @ (2 | (1 | (0 | nil)))))):NnNatList
---> (4 | (3 | (2 | (1 | (0 | nil))))):NnNatList
(4 | (3 | (2 | (1 | (0 | nil))))):NnNatList

In which for each one step rewrite the redex is not underlined and the rewrite rule used is not shown.

Exercises

1. Let us consider the module NATLIST. Write the traces of reductions of the following terms:
    1) hd(0 | 1 | nil)
    2) tl(0 | 1 | nil)
    3) [2 .. 5]
2. Let us consider the module GCD. Write the trace of reduction of gcd(24,36).
3. Let us consider the module FACT. Write the trace of reduction of fact(5).
4. Let us consider the module OEDC-FACT. Write the trace of reduction of oedc-fact(5).

Exercises

5. Let us consider the module QSORT. Write the trace of reduction of qsort(2 | 1 | 0 | 3 | 4 | nil).
6. Let us consider the module ERATOSTHENES-SIEVE. Write the trace of reduction of primesUpto(6).

Note that each equation used should be given a unique name and you can use the following pseudo-equations as rewrite rules.

eq X rem NzX = remainder of dividing X by NzX . (rem)
eq X < Y = true if so & false otherwise . (<)
eq X > Y = true if so & false otherwise . (>)
eq NzX divides X = true if X is a multiple of NzX & false otherwise . (divides)
eq X * Y = multiplication of X and Y . (*)
eq sd(X,Y) = symmetric difference (| X – Y |) between X and Y . (sd)
eq p NzX = the previous number of NzX . (p)
eq X + Y = addition of X and Y . (+)

Exercises

7. Investigate lambda calculus, which is often used as the basis of many functional programming languages and make a comparison of it with term rewriting.
8. Investigate higher-order functions and how to implement them.
9. Investigate how to implement term rewriting and CafeOBJ.
10. Investigate how to implement lambda calculus and some other functional programming languages based on the calculus, for example, by reading the book “Daniel P. Friedman, Mitchell Wand: Essentials of programming languages (3. ed.). MIT Press 2008.”

Exercises

11. Investigate term rewriting furthermore, for example, by reading the book “Terese: Term Rewriting Systems. Cambridge University Press 2003.”