Introduction to Rewriting

Main concepts of this unit:

Signature
- variable
- symbol, arity

Term
- root, arguments
- ground

Position

Substitution
- domain, range

Rule

Rewrite System
- redex, normal form
- step
- constructor, many sorted

Confluence

Termination
- strong, weak

Orthogonality
- linearity, ambiguity

Signature

Rewriting is a model of computation. It has been used in theorem proving and to abstract the execution of programs in declarative languages.

In this lecture, you will learn this model of computation. Several preliminary concepts are necessary, in particular, **term** and **rewriting.**

An *alphabet* or *signature,* denoted Σ, consists of:

- A denumerable set of *variables* $x_0, x_1, x_2, \ldots$ also denoted $x, y, z, x', y', \ldots$
- A non-empty, finite set of *symbols* $f, g, h, \ldots$ also denoted by expressive names such as *pop*, *push*, $+, 0, \ldots$ Each symbol has an *arity,* a natural number intended as the number of arguments a symbol takes.

Example 2. The symbols of the signature and their arities are:

z	0
s	1
a	2
m	2

Term

A *term* or *expression* over a signature Σ, denoted $\text{TER}(\Sigma)$, is defined inductively as follows:

$x_0, x_1, x_2, \ldots \in \text{TER}(\Sigma)$

If f is a symbol of arity n, $n \geqslant 0$,
and $t_1, t_2, \ldots, t_n \in \text{TER}(\Sigma)$,
then $f(t_1, t_2, \ldots, t_n) \in \text{TER}(\Sigma)$.

In the second case, f is called the *root* of $t_1, t_2, \ldots, t_n$ and $t_1, t_2, \ldots, t_n$ are called the *arguments* of f.

Terms without variables are called *ground* terms.

Example 3. A few terms:

z
x
$s(z)$
$a(s(z), m(s(z), s(z)))$
$a(z, z)$

Exercise 3. Code a Curry program for representing terms. Consider two options: (simple) represent only ground terms of the signature implied by Example 2; (harder) represent terms of any arbitrary signature.

Position

Terms are naturally represented as, and sometimes called, trees. The tree representation of the term $a(s(z), z)$ is:

$$\begin{tikzcd}
 & a \arrow[dl, no head] \arrow[dr, no head] & \\
s \arrow[d, no head] & & z \\
z & &
\end{tikzcd}$$

A position is a sequence of natural numbers defining a path in a tree, therefore identifying a subterm of a term.

Example 4. The position $\langle 1 \cdot 1 \rangle$ identifies the leftmost z in the above term, $\langle 2 \rangle$ identifies the rightmost z.

The positions in the above term are:

$$\begin{tikzcd}
 & \langle\rangle \arrow[dl, no head] \arrow[dr, no head] & \\
\langle 1 \rangle \arrow[d, no head] & & \langle 2 \rangle \\
\langle 1 \cdot 1 \rangle & &
\end{tikzcd}$$

Brackets and dots separating the numbers in a position are often omitted. The empty position is also denoted by Λ.

Position cont'd

The subterm of a term t at the *position* or *occurrence* p, denoted $t|_p$, is defined inductively as follows:

$t|_\Lambda = t$,
for all t

$f(t_1, t_2, \ldots, t_n)|_{iq} = t_i|_q$,
if $1 \leqslant i \leqslant n$ and q is a position

Example 5. Trace of the computation of the subterm at a given position:

$$a(s(z), z)|_{\langle 1 \cdot 1 \rangle} = s(z)|_{\langle 1 \rangle} = z|_{\langle\rangle} = z$$

Exercise 5a. Code a Curry function that takes a term t and a position p of t and returns $t|_p$. Hint: use the representation of terms you chose for Exercise 2.

Exercise 5b. Code a Curry function that takes two terms, t and u, and returns all the positions of u in t.

Substitution

A *substitution* is a mapping from the set of variables $\textrm{V}\small{\textrm{AR}}(\Sigma)$ to the set of terms $\textrm{T}\small{\textrm{ER}}(\Sigma)$. The set of variables "affected" by a substitution, σ, namely:

$$\textrm{D}\small{\textrm{OM}}(\sigma) = \{x \mid \sigma(x) \neq x\}$$

is called the *domain* of σ. The set:

$$\textrm{I}\small{\textrm{MG}}(\sigma) = \{y \mid y \text{ occurs in } \sigma(x) \text{ for some } x \in \textrm{D}\small{\textrm{OM}}(\sigma)\}$$

is called the *image* or *range* of σ

Most often in logic programming it is required that $\textrm{D}\small{\textrm{OM}}(\sigma)$ is a finite set and that $\textrm{D}\small{\textrm{OM}}(\sigma) \cap \textrm{I}\small{\textrm{MG}}(\sigma) = \varnothing$.

A substitution is denoted by:

$$\{v_1 \mapsto t_1, \ldots v_n \mapsto t_n\}$$

A substitution σ is extended to terms as follows:

$$\sigma(f(t_1, t_2, \ldots, t_n)) = f(\sigma(t_1), \sigma(t_2), \ldots, \sigma(t_n))$$

Example 6. If $\sigma = \{x \mapsto 0, y \mapsto 1\}$ and $t = x + y$, then $\sigma(t) = 0 + 1$.

Exercise 6. Define a *substitution* and code a Curry function that takes a substitution and a term and returns the result of applying the substitution to the term.

Rule

A *rewrite rule* for a signature Σ is a pair of terms of $\textrm{T}\small{\textrm{ER}}(\Sigma)$, denoted $l \to r$, with the conditions:

l is not a variable,

every variable in r is also in l.

Later, the second condition will be dropped, but other conditions will be added.

Example 7. The following rules refer to Example 2. Variables are in upper case.

$$\begin{array}{rcl} a(z, Y) & \to & Y \\ a(s(X), Y) & \to & s(a(X, Y)) \\ m(z, Y) & \to & z \\ m(s(X), Y) & \to & a(Y, m(X, Y)) \end{array}$$

Rewrite rules compute by replacing in a term an instance of a rule's lhs with the corresponding instance of the rhs.

Example 7 con't.

$$a(s(z), s(z)) \to s(a(z, s(z))) \to s(s(z))$$

The last element of the above sequence has no replacements. It is called a *normal form.*

Rewrite System

A *rewrite system* is a pair $\langle \Sigma, \mathcal{R} \rangle$, where Σ is a signature and $\mathcal{R}$ is a set of rewrite rules over Σ.

Let t be a term, $l \to r$ a rule, p a position of t, and σ a substitution such that $\sigma(l) = t|_p$, i.e., the subterm of t at position p is a *redex* or an instance of l.

A *rewrite* (step) is a pair of terms $t \to t[\sigma(r)]_p$, where the latter denotes the term obtained by replacing the subterm of t at position p with $\sigma(r)$. The rewrite relation, denoted by "$\to$," of $\langle \Sigma, \mathcal{R} \rangle$ is the set of all the rewrite steps. "$\overset{*}{\to}$" denotes the reflexive transitive closure of "$\to$."

A rewrite relation is:

Confluent if $t \overset{*}{\to} t_1$ and $t \overset{*}{\to} t_2$ imply the existence of a u such that $t_1 \overset{*}{\to} u$ and $t_2 \overset{*}{\to} u$, for all t, t_1 and t_2.

Strongly terminating if there is no infinite sequence of rewrite steps.

Weakly terminating if every term has a normal form.

Rewrite System cont'd

Example 9. The system of Example 7 is confluent and strongly terminating. How can one become convinced of this claim?

Exercise 9a. Discuss confluence and termination of:

$$\begin{array}{rcl} a & \to & b \\ f(a) & \to & f(a) \end{array}$$

Exercise 9b. Implementing rewriting for the system of Example 7 in Curry is easy if one is not picky about choosing steps. Code a small program to this aim. Hint: for testing, code two functions as follows:

encd takes a natural number i and return $s(s(\ldots s(z)\ldots))$, where there are exactly i applications of s,

decd is the inverse of encd, i.e., $\mathtt{decd}(\mathtt{encd}(i)) = i$ and vice versa.

and inspect the normal form of

$\mathtt{decd}(a(\mathtt{encd}(i), \mathtt{encd}(j)))$

and likewise for m.

Specialty Systems

Constructor systems: the signature is partitioned into a set of *constructors* and a set of *operations.* The lhs of each rewrite rule is of the form $f(t_1, \ldots, t_n)$, where the root f is an operation and each argument t_i contains only constructors and variables.

Many sorted systems: there is a set of *sorts* (type symbols). The *type* of a signature symbol f is a string of sort symbols denoted $s_1 \times \cdots \times s_n \to s$, for $n \geqslant 0$. The type of a term $f(t_1, \ldots, t_n)$ is s provided that the sort of t_i is s_i

Example 10.
Sorts: $\{Color, Stack\}$
Signature: $\{red, blue, green, empty, push, top, pop\}$
Types:

$$\begin{array}{lcr} red & : & Color \\ blue & : & Color \\ green & : & Color \\ empty & : & Stack \\ push & : & Color \times Stack \to Stack \\ pop & : & Stack \to Stack \\ top & : & Stack \to Color \end{array}$$

Rules:

$$\begin{array}{lcl} pop(push(C,S)) & \to & S \\ top(push(C,S)) & \to & C \end{array}$$

Orthogonality

Two key properties of rewrite systems are:

Left linearity: no repeated variables in the rules' lhss.

Non ambiguity: let $l_1 \to r_1$ and $l_2 \to r_2$ be two rules such that l_1 unifies with a non-variable subterm of l_2, i.e., there exists a substitution σ and a position p of l_2 such that $\sigma(l_2|_p) = \sigma(l_1)$. The term $\sigma(l_2)$ can be rewritten in two ways, namely $\sigma(r_2)$ and $\sigma(l_2[r_1]|_p)$. These two terms form a *critical pair.*

A system is *orthogonal* if it is left-linear and is non-ambiguous (has no critical pairs). Orthogonal systems are confluent.

Exercise 11. Consider the following Curry (and Haskell) program as a rewrite system:

```
insert e xs = e:xs
insert e (x:xs) = x:insert e xs
```

Is the system confluent? Hint: this is a sneaky question! You may get some insight by running the program in each language.

Abstract Systems

Modern treatments of rewrite systems consider *abstract reduction systems,* structures $\langle A, \to \rangle$, where A is a set and $\to$ is a binary relation (or family of relations) on A.

Many key concepts of rewriting, e.g., step, redex, termination, confluence, etc., are independent of terms, hence can be defined for abstract systems.

Many fundamental theorems of rewriting can be proved for abstract systems.

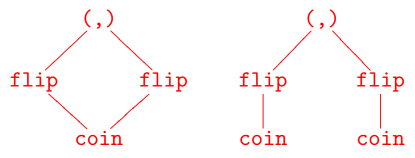
We focus on *term* rewriting systems because they are more interesting for programming.

Graph Rewriting

Formalism similar to *term* rewriting, but expressions are *graphs* rather than *trees*. Consider the system:

$$\begin{array}{lcl} coin & \to & 0 \\ coin & \to & 1 \\ flip\ 0 & \to & 1 \\ flip\ 1 & \to & 0 \end{array}$$

and the expressions :



The Curry (textual) representations are:

```
(flip x, flip x) where x = coin
(flip coin, flip coin)
```

The first one has 2 values only: $(0,0)$ and $(1,1)$.
The second has 4, also: $(0,1)$ and $(1,0)$.

In Curry, variables are always *shared,* multiple occurrences of the same variable refer to the same object.