Nominal Rewriting: Or . . . talking about functions, without functions.

Murdoch J. Gabbay

Joint work with Maribel Fernández

Thanks to Arnon Avron and Nachum Derschowitz.

Tel-Aviv University, 6 October 2005.

Health warning:

This will not be a formal or technical talk.

This material is back-to-front and consists mostly of lies.

For honesty, read other slides, or the papers. (On my homepage.)

Functions are very useful, but rather weighty beasts.

We want to talk *about* functions, and compute their results, but do we want to meet someone on the street who gives us a function and says '*here, hold this for a moment*'.

For example, the identity function on the natural numbers **Id** is characterised by the rewrite $\mathbf{Id}(x) \longrightarrow x$. This is quite different from its set representation as

$$\mathbf{Id} = \{(0,0),\ (1,1),\ (2,2),\ \ldots\}$$

which is guaranteed to crash any computer which tries to actually build this as a data element.

Computers are good at manipulating *syntax*, and not *mathematics* (or ideas).

So welcome to the wonderful world of formal syntax.

Pick some *term-formers* f_1, f_2, ..., f_n. This is a *signature*.

Typical examples; *the signature of a pocket calculator*:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, \times, /, -.$$

The signature of elementary linguistics:

$$\text{Mary},\ \text{John},\ \text{loves}.$$

A *term* is an *abstract syntax tree* which is a tree with each node labelled by term-formers. Most convenient to represent this as a properly bracketed string with comma-separated subtrees,

$$\text{loves}(\text{Mary}, \text{John}) \quad +(0, 1) \quad +(1, \times(2, 3))$$

or even better as appropriate infix notation:

$$\text{Mary loves John} \quad 0 + 1 \quad 1 + (2 \times 3)$$

Rewriting

Nota bene: this is an *unsorted* system. $loves(John)$ or $0(1,1,3)$ are still terms, just not meaningful ones. $0(1,1,3)$ is a tree with root labelled 0 and three subtrees each consisiting of a single leaf labelled respectively 1, 1, and 3.

They hang around in our universe of terms and probably nobody will ever touch them, but they don't harm anyone. Like books in the shelves of a well-meaning but time-poor professor. Forget about them (or impose sorts).

Also, for the pocket calculator example we might want to let $1(1)$ represent 11. I never use no number larger than 3 in this talk.

(0 to 9 on previous slide are term-formers; $1(1)$ is a term; 11 is binary, of course.)

Rewrite system for arithmetic

Extend the language of terms with *unknowns* X, Y, Z (also called *variable symbols*).

A rewrite rule is a pair of terms $l \longrightarrow r$ where the unknowns mentioned in r are a subset of those mentioned in l.

For example:

$$X \times (Y + Z) \longrightarrow (X \times Y) + (X \times Z) \quad X + Y \longrightarrow Y + X$$

Think of a rewrite rule as being a *directed equality*, parametarised over whatever goes into its unknowns (we say they are *instantiated*).

Typically it is useful to orient rewrites so that terms become simpler, and reduce in finite time to a term which does not reduce any further (a *normal form*).

This models computation; the normal form is our ‘result’.

Rewrite system for combinators

Take as signature $\mathbf{S}$, $\mathbf{K}$, and $\cdot$ (application). Write $\cdot(s,t)$ as st and associate to the left, as usual. Rewrite rules are:

$$\mathbf{S}XYZ \to (XZ)(YZ) \qquad \mathbf{K}XY \to X.$$

An example rewrite is (for t any term):

$$\mathbf{SKK}t \to (\mathbf{K}t)(\mathbf{K}t) \to t$$

($\mathbf{SKK}$ models the identity function.)

Rewrite system for the lambda-calculus

Extend the signature with λ:

- Countably infinitely many *atoms* $a, b, c, d, \ldots$ to represent variable symbols of the logic,
- A term-former *abstraction* abs. Write $abs(a, t)$ as $[a]t$.
- A term-former *substitution* σ. Write $\sigma([a]t, u)$ as $t[a \mapsto u]$.

Additional rewrite rules are:

$$(\lambda[a]X)Y \to X[a \mapsto Y] \qquad \big(app(\lambda[a]X, Y) \to \sigma([a]X, Y)\big)$$

and...

Explicit substitution

$$a[a{\mapsto}X] \to X \quad b[a{\mapsto}X] \to b$$

$$f(X_1,\ldots,X_n)[a{\mapsto}X] \to f(X_1[a{\mapsto}X],\ldots,X_n[a{\mapsto}X])$$

$$b\#X \;\vdash\; ([b]Y)[a{\mapsto}X] \to [b](Y[a{\mapsto}X])$$

For example:

$$(\lambda[a]a)b \to a[a{\mapsto}b] \to b$$

$$(\lambda[a]aab)b \to (aab)[a{\mapsto}b] \to (aa)[a{\mapsto}b](b[a{\mapsto}b]) \to^* bbb$$

$$(\lambda[a]\lambda[b]a)b \to (\lambda[b']a)[a{\mapsto}b] \to \lambda(([b']a)[a{\mapsto}b]) \overset{b'\#b}{\to}$$
$$\lambda[b'](a[a{\mapsto}b]) \to \lambda[b']b.$$

Em . . . what are $b\#X$ and $b'\#b$? How did b turn into b'?

$b\#s$ is read b *is fresh for* s. The intuition is that b can occur in s, but only under an abstraction.

$[a]s$ is read *abstract* a *in* s.

Remember that unknowns get instantiated, so $b\#X$ in a rewrite will turn into $b\#s$ for some appropriate s.

Also, rewrite rules are applied *up to* renaming unknowns and atoms. So

$$(\lambda[a]X)Y \to X[a{\mapsto}Y] \qquad \big(app(\lambda[a]X,Y) \to \sigma([a]X,Y)\big)$$

induces the same rewrites as

$$(\lambda[c]X)Y \to X[c{\mapsto}Y] \qquad \big(app(\lambda[c]X,Y) \to \sigma([c]X,Y)\big)$$

and

$$(\lambda[c]X')Y' \to X'[c{\mapsto}Y'] \qquad \big(app(\lambda[c]X',Y') \to \sigma([c]X',Y')\big)$$

α-equality and freshness

$$\frac{a\#s_1 \;\cdots\; a\#s_n}{a\#f(s_1,\ldots,s_n)} \qquad \frac{a\#s}{a\#[b]s} \qquad \frac{}{a\#b} \qquad \frac{}{a\#[a]s} \qquad \frac{\pi^{-1}(a)\#X}{a\#\pi\cdot X}$$

$$\frac{s_1 \approx t_1 \;\cdots\; s_n \approx t_n}{f(s_1,\ldots,s_n) \approx f(t_1,\ldots,t_n)} \qquad \frac{}{a \approx a} \qquad \frac{t \approx t'}{t' \approx t}$$

$$\frac{s \approx t}{[a]s \approx [a]t} \qquad \frac{a\#t \qquad s \approx (a\,b)\cdot t}{[a]s \approx [b]t} \qquad \frac{ds(\pi,\pi')\#X}{\pi\cdot X \approx \pi'\cdot X}$$

$$ds(\pi,\pi') \stackrel{\text{def}}{=} \{n \mid \pi(n) \neq \pi'(n)\}.$$

For example, $ds((a\,b),\mathbf{Id}) = \{a,b\}$.

Example derivation

$$\dfrac{\dfrac{}{a\#\lambda[a]ba} \qquad \dfrac{\dfrac{a \approx a \quad b \approx b}{ab \approx ab}}{\lambda[b]ab \approx (b\ a) \cdot (\lambda[a]ba) \equiv \lambda[b]ab}}{\lambda[a]\lambda[b]ab \approx \lambda[b]\lambda[a]ba}$$

$$\dfrac{\dfrac{}{a\#\lambda[a](b\ a) \cdot X} \qquad \dfrac{\dfrac{}{X \approx (b\ a) \circ (b\ a) \cdot X}\,(\#X)}{\lambda[b]X \approx (b\ a) \cdot (\lambda[a](b\ a) \cdot X) \equiv \lambda[b](b\ a) \circ (b\ a) \cdot X}}{\lambda[a]\lambda[b]X \approx \lambda[b]\lambda[a](b\ a) \cdot X}$$

Note permutation allows treatment of *open terms* (terms mentioning unknowns), which allows a parametric treatment of terms with abstraction *and* unknowns.

A *swapping* is a pair of atoms $(a\ b)$. It has an action on atoms and terms as follows:

$$(a\ b)\cdot a \equiv b \qquad (a\ b)\cdot b \equiv a \quad \text{and} \quad (a\ b)\cdot c \equiv c\ (c \neq a, b)$$

and

$$(a\ b)\cdot f(t_1,\ldots,t_n) = f((a\ b)\cdot t_1,\ldots,(a\ b)\cdot t_n)$$
$$(a\ b)\cdot [n]t = [(a\ b)\cdot n](a\ b)t$$
$$(a\ b)\cdot(\pi\cdot X) = (a\ b)\circ\pi\cdot X.$$

Permutations $\pi ::= \mathbf{Id} \mid (a\ b)\circ\pi$ act as the composition of their component swappings: $(ab)\circ(b\ c)\cdot c \equiv a$.

Note that permutations act on abstractions just like any other term-former (distributing into it). Any kind of substitution would have to worry about avoiding capture, thus inducing side-conditions which make for a far less ‘algebraic’ definition.

The recipe is:

1. Separate meta-variables (unknowns X) and object-variables (atoms a).
2. Introduce a term-former $[a]X$ to model abstraction (like λ but:).
3. β-reduction *not* a structural congruence of terms — implement with rewrites.
4. α-equivalence is also *not* a structural congruence of terms — so $[a]a \not\equiv [b]b$.
5. Handle rewriting of open terms using freshness assumptions $a\#X$.
6. *Do* introduce $\approx$ and make it primitive, in that by definition of rewriting (omitted!) if $\Gamma \vdash s \rightarrow t$ and if $\Gamma \vdash t \approx t'$ then $\Gamma \vdash s \rightarrow t'$.

Nominal Rewriting [PPDP'04]

It is a theorem that if $\Gamma \vdash s \to t$ and $\Gamma \vdash s \approx s'$ then $\Gamma \vdash s' \to t$.

Importantly, this is not built into the definition. This is a basic 'correctness' property which also guarantees that rewrites are efficiently computable.

Summary

Rewriting says: take a world of syntax and give it a dynamics based on a notion of partial terms which is formalised by unknowns. The only property of a term relevant for rewriting is its outermost term-formers.

Nominal rewriting says: take a world of syntax as above but with the additional intensional property of freshness $\#$ (for a set of atoms $a, b, c, \ldots$). Make rewrites conditional on satisfying freshness conditions on the unknowns, in the sense that the conditions should be actually valid for whatever the unknown is instantiated to.

$$b\#X \vdash ([b]Y)[a\mapsto X] \rightarrow [b](Y[a\mapsto X]).$$

In this way, we get to talk about functions, without ever meeting them in person.