Formal Methods
1. Rewriting

Nachum Dershowitz

February 2000

Consider the following program to re-arrange a list of natural numbers in non-increasing order by inserting elements one-by-one into position:

$$\begin{array}{rcl}
max(0,x) & \rightarrow & x \\
max(x,0) & \rightarrow & x \\
max(s(x),s(y)) & \rightarrow & s(max(x,y)) \\
min(0,x) & \rightarrow & 0 \\
min(x,0) & \rightarrow & 0 \\
min(s(x),s(y)) & \rightarrow & s(min(x,y)) \\
sort(\varepsilon) & \rightarrow & \varepsilon \\
sort(x:y) & \rightarrow & insert(x,sort(y)) \\
insert(x,\varepsilon) & \rightarrow & x:\varepsilon \\
insert(x,y:z) & \rightarrow & max(x,y):insert(min(x,y),z))
\end{array}$$

Lists are represented in "cons" notation (with : and ε, as constructors) and numbers in successor (unary) notation. Each *(rewrite) rule* is an ordered pair of terms, used replace instances of l by corresponding instances of r. For example, $sort(max(0,s(0)):\varepsilon)$ *rewrites* to $sort(s(0):\varepsilon)$. We would like to know that every term constructed from *sort*, :, ε, s, and 0 leads (in zero or more rewrites) to a unique term not containing *sort*, nor the auxiliary symbols, *insert*, *max*, and *min*.

Rewrite systems (sets of rewrite rules) can be used to "interpret" other programming languages: The state of an all-powerful two-counter machine can be represented as a pair $\langle x,\ y\rangle$. The semantics of its instruction set can be defined by the following rules for an interpreter:

$$
\begin{array}{rcl}
\mathit{eval}(\mathbf{zap0}\ , \langle x,\ y\rangle) & \to & \langle 0,\ y\rangle \\
\mathit{eval}(\mathbf{zap1}\ , \langle x,\ y\rangle) & \to & \langle x,\ 0\rangle \\
\mathit{eval}(\mathbf{inc0}, \langle x,\ y\rangle) & \to & \langle s(x),\ y\rangle \\
\mathit{eval}(\mathbf{inc1}, \langle x,\ y\rangle) & \to & \langle x,\ s(y)\rangle \\
\mathit{eval}(\mathbf{dec0}, \langle 0,\ y\rangle) & \to & \langle 0,\ y\rangle \\
\mathit{eval}(\mathbf{dec0}, \langle s(x),\ y\rangle) & \to & \langle x,\ y\rangle \\
\mathit{eval}(\mathbf{dec1}, \langle x,\ 0\rangle) & \to & \langle x,\ 0\rangle \\
\mathit{eval}(\mathbf{dec1}, \langle x,\ s(y)\rangle) & \to & \langle x,\ y\rangle \\
\mathit{eval}(\mathbf{ifpos0}\ p, \langle 0,\ y\rangle) & \to & \langle 0,\ y\rangle \\
\mathit{eval}(\mathbf{ifpos0}\ p, \langle s(x),\ y\rangle) & \to & \mathit{eval}(p, \langle s(x),\ y\rangle) \\
\mathit{eval}(\mathbf{ifpos1}\ p, \langle x,\ 0\rangle) & \to & \langle x,\ 0\rangle \\
\mathit{eval}(\mathbf{ifpos1}\ p, \langle x,\ s(y)\rangle) & \to & \mathit{eval}(p, \langle x,\ s(y)\rangle) \\
\mathbf{whilepos0}\ p & \to & \mathbf{ifpos0}\ (p;\ \mathbf{whilepos0}\ p) \\
\mathbf{whilepos1}\ p & \to & \mathbf{ifpos1}\ (p;\ \mathbf{whilepos1}\ p) \\
\mathit{eval}((p;\ q), u) & \to & \mathit{eval}(q, \mathit{eval}(p, u))
\end{array}
$$

The penultimate rule, for example, can clearly be applied *ad infinitum*. A specific strategy of rule application is needed to guarantee that a normal form is obtained whenever one exists.

In general, rules are usually applied nondeterministically, since, in general, more than one rule can be applied, and any one rule may apply at more than one position within a term.

For any binary relation $\to$, we use $s \leftrightarrow t$ to mean $s \to t$ or $t \to s$. We say s *derives* t and write $s \to^* t$ if $s \to \cdots \to t$ in zero or more steps. We say that s and t are *convertible*, symbolized $s \leftrightarrow^* t$, if $s \leftrightarrow \cdots \leftrightarrow t$ in zero or more steps, and that s and t are *joinable*, if s and t derive the same term. A *normal form* is an element s for which there is no t such that $s \to t$. We write $s \to^! t$ if s derives the normal form t.

Definition 1 (Church-Rosser Property) *A binary relation is* Church-Rosser *if elements are joinable whenever they are convertible.*

Definition 2 (Confluence) *A binary relation is* confluent *if elements are joinable whenever they are derivable from the same term.*

Lemma 1 *The Church-Rosser and confluence properties are equivalent.*

Proof The proof is by induction on the number of "peaks" in the conversion. □

Definition 3 *A binary relation is* locally confluent (weakly confluent) *if for all elements r, s, and t, if $r \to s$ and $r \to t$ then $s \downarrow t$.*

Definition 4 (Termination) *A binary relation is* terminating *if there are no infinite derivations $t_1 \to t_2 \to \cdots$.*

Termination is an undecidable property of rewrite systems.

Theorem 1 (Newman's Lemma) *If a binary relation is locally confluent and terminating, then it is confluent.*

Proof The proof is by well-founded induction. □

The normalizability relation $\to^!$ for uniquely-normalizing systems defines a function.

Definition 5 (Orthogonality) *A rewrite system is* orthogonal *if*

- *no variable appears on the right side of a rule but not on the left;*
- *no variable appears more than once on any left side;*
- *no left side unifies with a (renamed) non-variable subterm of any other left-hand side or with a (renamed) proper subterm of itself.*

Combinatory Logic is a prime example of a (nonterminating) orthogonal system:

$$\begin{aligned} I \circ x &\to x \\ (K \circ x) \circ y &\to x \\ ((S \circ x) \circ y) \circ z &\to (x \circ z) \circ (y \circ z) \end{aligned}$$

This system can be used to implement any recursive function. The combinators K and S were dubbed "kestrel" and "starling" by Smullyan; I is the identity combinator; $\circ$ is composition.

An example of a non-orthogonal non-confluent (non-terminating) system is:

$$\begin{aligned} f(x,x) &\to 0 \\ f(x,s(x)) &\to 1 \\ a &\to s(a) \end{aligned}$$

Definition 6 *A binary relation is* strongly confluent *if for all elements r, s, and t, if $r \to s$ and $r \to t$ then either s and t are identical or $s \to t$ or $t \to s$ or there is a term u such that $s \to u$ and $t \to u$.*

Lemma 2 *Every strongly confluent binary relation is confluent.*

Proof Straightforward. □

The importance of orthogonal systems stems from the following:

Theorem 2 *Every orthogonal system is confluent.*

In particular, the two-counter interpreter is confluent.

Proof The idea is to show that a parallel rewriting relation associated with the system R is strongly confluent. Parallel rewriting is rewriting at one or more disjoint positions at the same time. We need to consider parallel rewriting, because if s rewrites to t_1 and s rewrites to t_2, then a subterm of s may appear many times in t_1 or t_2, and all of these occurrences may have to be rewritten in parallel to find a t to which both t_1 and t_2 rewrite in one (parallel) step. For example, if s is $f(a)$ and R is $\{a \to b, f(x) \to g(f(x), f(x))\}$ then $f(a)$ rewrites to both $f(b)$ and $g(f(a), f(a))$. Both of these terms parallel rewrite to $g(f(b), f(b))$, and this requires parallel rewriting of the two occurrences of a in $g(f(a), f(a))$. □

Definition 7 (Outermost Rewriting) *A rewriting step $s \to t$ is* outermost *if no rule applies at a symbol closer to the root symbol (in the tree representation of terms).*

Theorem 3 *For any orthogonal system, if no outermost step is perpetually ignored, the normal form—if there is one—will be reached.*

Outermost rewriting is used to compute normal forms in Combinatory Logic.

In this way, orthogonal systems provide a simple, pattern-directed (first-order) functional programming language, in which the orthogonal conditional operator

$$\begin{array}{lcl} \mathit{if}(\mathbf{true}, x, y) & \to & x \\ \mathit{if}(\mathbf{false}, x, y) & \to & y \end{array}$$

can also conveniently be incorporated.