Executing Specifications

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.jku.at

Term Rewriting

Term rewriting can be used for “executing” initial specifications.

- Reduction system $(R, \to)$.
  - Set R, relation $\to \subseteq R \times R$.
- Reduction sequence $r_1, r_2, \ldots$
  - Finite or infinite sequence of elements $r_i \in R$ with $r_i \to r_{i+1}$, for every i.
    - Transitive closure: $r_1 \to^* r_k$, for every k.
- Equivalence sequence $r_1, r_2, \ldots$
  - Finite or infinite sequence of elements $r_i \in R$ with $r_i \to r_{i+1}$ or $r_{i+1} \to r_i$, for every i.
    - Transitive closure: $r_1 \simeq r_k$, for every k.

We will now investigate various properties of these notions.

Properties of Reduction Systems

Take reduction system $(R, \to)$.

- $(R, \to)$ is Noetherian:

  $(R, \to)$ does not have any infinite reduction sequence.
- $(R, \to)$ is locally confluent:

  $\forall r, s, t \in R : r \to s \land r \to t \Rightarrow \exists u \in R : s \to^* u \land t \to^* u.$

$$\begin{tikzcd} & r \arrow[dl] \arrow[dr] & \\ s \arrow[dr, "*"] & & t \arrow[dl, "*"'] \\ & u & \end{tikzcd}$$

- $(R, \to)$ is confluent:

  $\forall r, s, t \in R : r \to^* s \land r \to^* t \Rightarrow \exists u \in R : s \to^* u \land t \to^* u.$
- $s \in R$ is a normal form of $r \in R$:

  $r \to^* s \land \neg\exists t \in R : s \to t.$

Example

- Take reduction system $(\mathbb{N}^3, \rightarrow)$:

  $$(m_1, m_2, m_3) \rightarrow (n_1, n_2, n_3) :\Leftrightarrow$$
  $$m_3 > 0 \wedge n_1 = m_1 \wedge n_2 = m_2 + 1 \wedge n_3 = m_3 - 1 \vee$$
  $$m_2 > 0 \wedge n_3 = m_3 \wedge n_1 = m_1 + 1 \wedge n_2 = m_2 - 1.$$
- Possible reduction sequences:

  $$(4,1,2) \rightarrow (5,0,2) \rightarrow (5,1,1) \rightarrow (6,0,1) \rightarrow (6,1,0) \rightarrow (7,0,0)$$
  $$(4,1,2) \rightarrow (4,2,1) \rightarrow (5,1,1) \rightarrow (5,2,0) \rightarrow (6,1,0) \rightarrow (7,0,0)$$
- Reduction system is Noetherian, locally confluent, and confluent.
  - Normal forms are $(n, 0, 0)$ with $n \in \mathbb{N}$.

A reduction system may be viewed as a non-deterministic program (provided that R is decidable and $\rightarrow$ is computable).

Church-Rosser Property

- Lemma: for every confluent reduction system $(R, \rightarrow)$, we have

  $$\forall r, s \in R : r \simeq s \Leftrightarrow \exists t \in R : r \rightarrow^* t \wedge s \rightarrow^* t.$$
- Proof of "$\Rightarrow$" ("$\Leftarrow$ is trivial):
  - Take arbitrary $r \simeq s$ with equivalence sequence $r = r_1, r_2, \ldots, r_k = s$. Proof proceeds by induction on k.
  - $k = 1$: take $t = r$.
  - $k > 1$: by induction hypothesis, we have u with $r \rightarrow^* u$ and $r_{k-1} \rightarrow^* u$. Now either $r_{k-1} \rightarrow s$ or $s \rightarrow r_{k-1}$.
    - Case $r_{k-1} \rightarrow s$: by confluence, we have v with $s \rightarrow^* v$ and $u \rightarrow^* v$, hence $r \rightarrow^* v$. Take $t = v$.
    - Case $s \rightarrow r_{k-1}$: we have $s \rightarrow^* u$. Take $t = u$.

Graphical representation guides intuition in proof.

Central Theorem

- Newman's Lemma: a Noetherian and locally confluent reduction system is confluent.
  - It suffices to check local confluence.
- Theorem: take Noetherian and confluent reduction system $(R, \rightarrow)$.
  - Each element of R has exactly one normal form.
  - Let $r, s \in R$. Then $r \simeq s$ iff r and s have the same normal form.

Central theorem for reduction systems.

Term Rewriting Systems

Take initial specification (Σ, Φ).

- The term rewriting system for (Σ, Φ) is the reduction system $(T_\Sigma, \rightarrow)$ where $\rightarrow$ is inductively defined as follows:
  - $v\sigma \rightarrow w\sigma$
    - for each equation $\forall X.v = w \in \Phi$ and ground substitution $\sigma : X \rightarrow T_\Sigma$.
  - If $t \rightarrow u$, then $s[t/y] \rightarrow s[u/y]$
    - for every term $s \in T_{\Sigma(\{y\})}$ containing at least one occurrence of variable y.
- If $t \rightarrow u$, we call "$t \rightarrow u$" a rewrite rule.

Initial specifications give rise to reduction systems.

Example

initial spec
  sorts *nat*
  opns
    $0 :\to nat$
    $Succ : nat \to nat$
    $\_ + \_ : nat \times nat \to nat$
  vars $m, n : nat$
  eqns
    $n + 0 = n$
    $n + Succ(m) = Succ(n + m)$
endspec

- Some rewrite rules:
  $(0 + Succ(0)) + 0 \to 0 + Succ(0)$
  $Succ((0 + Succ(0))+0)+Succ(0) \to Succ(0+Succ(0))+Succ(0)$
- The normal forms are exactly the terms $Succ^n(0)$, for every $n \geq 0$.
  - If term contains $+$, a rewrite rule can be applied.
- The resulting term rewriting system is Noetherian and confluent.

Properties of Term Rewriting Systems

Take term rewriting system $(T_\Sigma, \to)$ for initial specification (Σ, Φ) with signature $\Sigma = (S, \Omega)$.

- $\forall s \in S : \forall t, u \in T_{\Sigma,s} : t \to^* u \Rightarrow \Phi \models t = u.$
  - If there is a reduction sequence from t to u, then t equals u.
- $\forall s \in S : \forall t, u \in T_{\Sigma,s} : t \simeq u \Rightarrow \Phi \models t = u.$
  - If there is an equivalence sequence between t and u, then t equals u.
- $\forall s \in S : \forall t, u \in T_{\Sigma,s} : \Phi \models t = u \Rightarrow t \simeq u.$
  - If t equals u, then there is an equivalence sequence between t and u.

The notion of equality in a specification coincides with the existence of an equivalence sequence in the specification's term rewriting system.

Proofs by Term Rewriting

- Simple proof method for equality proofs:
  - To prove $\Phi \models t = u$, it suffices to prove $t \simeq u$.
  - To prove $\Phi \models \forall X.v = w$, it suffices to prove $v\sigma \simeq w\sigma$ for all ground substitutions σ.
- Example: prove for initial specification (Σ, Φ) of the natural numbers
  $\Phi \models n + Succ(Succ(0)) = Succ(Succ(n) + 0)$
  - Take arbitrary ground term $t \in T_\Sigma$ and prove
    $t + Succ(Succ(0)) \simeq Succ(Succ(t) + 0)$
  - $t + Succ(Succ(0))$
    $\to Succ(t + Succ(0))$
    $\to Succ(Succ(t + 0))$
    $\to Succ(Succ(t))$
    $\leftarrow Succ(Succ(t) + 0)$

Difficult to find equivalence sequence between terms.

Proofs by Term Rewriting (Contd)

- Assume that term rewriting system is Noetherian and confluent.
- Assume that normal forms are terms $Succ^n(0), n \geq 0$.
  - Both needs proof.
- Take arbitrary ground term $t \in T_\Sigma$ and prove
  $t + Succ(Succ(0)) \simeq Succ(Succ(t) + 0)$
  - Take $k \geq 0$ such that $t \simeq Succ^k(0)$.
  - $t + Succ(Succ(0))$
    $\to Succ(t + Succ(0))$
    $\to Succ(Succ(t + 0))$
    $\to Succ(Succ(t))$
    $\to^* Succ^{k+2}(0)$.
  - $Succ(Succ(t) + 0)$
    $\to Succ(Succ(t))$
    $\to^* Succ^{k+2}(0)$.

The existence of unique normal forms simplifies rewriting proofs.

Execution by Term Rewriting

Take initial specification (Σ, Φ) with a Noetherian and confluent term rewriting system.

- Theorem: let C be the Σ-algebra defined as:
  - $C(s) = \{t \in T_{\Sigma,s} \mid t \text{ is a normal form}\}$
    - for each sort s of Σ.
  - $C(\omega) = \text{the normal form of term } n$
    - for each constant $\omega = (n :\to s)$ of Σ.
  - $C(\omega)(t_1, \ldots, t_k) = \text{the normal form of term } n(t_1, \ldots, t_k)$
    - for each operation $\omega = n : s_1 \times \ldots \times s_k \to s$ of Σ.

  Then we have:
  - $C(t)$ is the normal form of t, for each ground term $t \in T_\Sigma$.
  - C is a characteristic term algebra for (Σ, Φ).
  - C is thus isomorphic to $T(\Sigma, \Phi)$.

The calculation of the value of a ground term may be performed by calculating the normal form of the term.

Systems for Executing Specifications

Based on the previous result, systems like CafeOBJ "execute" initial specifications (i.e. compute the values of ground terms).

- Basic strategy: equations are treated as rewrite rules.
  - Ground terms are rewritten to their normal forms.
- But term rewriting systems may not be Noetherian or confluent.
  - Rewriting may not terminate, normal forms may not be unique.
- Why not check for these properties in advance?
  - We may prove that a certain term rewriting system is Noetherian.
    - Need to find a Noetherian (well-founded) irreflexive partial order of terms that is decreased by the application of every rewrite rule.
  - But also local confluence is undecidable, and so is confluence.
    - The *Knuth-Bendix* completion method tries to construct from a given initial specification sp a specification sp' with $\mathcal{M}(sp) = \mathcal{M}(sp')$ such that, if the term rewriting system for sp is Noetherian, the term rewriting system for sp' is Noetherian and confluent.
    - Semi-algorithm: termination is not guaranteed.

Are there specifications that are *guaranteed* to be executable?

Constructive Specifications

Specifications with an "operational" flavor ("abstract programs").

- Constructive specification $sp = (\Sigma, \Phi, \Omega_c)$
  - Signature $\Sigma = (S, \Omega)$.
  - Set of equations $\Phi \subseteq EL(\Sigma)$.
  - Set of constructors $\Omega_c \subseteq \Omega$.

  Three constraints must be satisfied that can be informally stated as:
  1. The left-hand side of every equation is a "pattern", the right-hand side is the "value" of this pattern.
  2. Every ground term whose outermost operation is not a constructor "matches" exactly one pattern.
  3. Treating the equations as rewrite rules from left to right cannot lead to "infinite recursion" in the evaluation of ground terms.

The conditions ensure that every ground term can be *deterministically* evaluated to a constructor term in a *finite* number of steps.

Constructive Specifications (Contd)

The three constraints are formalized as follows:

1. Each equation of Φ has form $n(v_1, \ldots, v_k) = t$ with

   $(n : s_1 \times \ldots \times s_k \to s) \in \Omega \backslash \Omega_c$,
   $v_i \in T_{\Sigma_c(X),s_i}$, for all i,
   $t \in T_{\Sigma(X),s}$, $Var(t) \subseteq Var(n(v_1, \ldots, v_k))$,
   no variable may occur more than once in $n(v_1, \ldots, v_k)$.

2. For each ground term $n(w_1, \ldots, w_k)$ of T_Σ with

   $(n : s_1 \times \ldots \times s_k \to s) \in \Omega \backslash \Omega_c$,
   $w_i \in T_{\Sigma_c, s_i}$, for all i

   there exists exactly one equation $n(v_1, \ldots, v_k) = t$ in Φ and exactly one ground substitution $\sigma : Var(n(v_1, \ldots, v_k)) \to T_{\Sigma_c}$ such that

   $w_i = v_i\sigma$, for all i.

3. There exists a reduction ordering $<$ such that

   $t < n(v_1, \ldots, v_k)$, for each equation $n(v_1, \ldots, v_k) = t$ in Φ.

Example

constructive spec
  sorts *nat*
  opns
    constr $0 :\to nat$
    constr $Succ : nat \to nat$
    $\_ + \_ : nat \times nat \to nat$
  vars $m, n : nat$
  eqns
    $m + 0 = m$
    $m + Succ(n) = Succ(m + n)$
endspec

First constraint is clearly satsfied but it is not evident that this is also the case for the last two constraints.

Semantics of Constructive Specifications

Take constructive specification $sp = (\Sigma, \Phi, \Omega_c)$ with signature $\Sigma = (S, \Omega)$ and define $\Sigma_c = (S, \Omega_c)$.

- Specification semantics $\mathcal{M}(sp) = \{A \in Alg(\Sigma) \mid A \simeq C\}$
  where C is the canonical algebra of sp defined as follows:
  - $C(s) = T_{\Sigma_c, s}$, for each sort $s \in S$.
  - $C(\omega) = n$, for each constructor constant $\omega = (n :\to s) \in \Omega_c$.
  - $C(\omega)(w_1, \ldots, w_k) = n(w_1, \ldots, w_k)$,
    for each constructor $\omega = (n : s_1 \times \ldots \times s_k \to s) \in \Omega_c, k \geq 1$,
    and for each constructor term $w_i \in T_{\Sigma_c, s_i}$, for every i.
  - $C(\omega)(w_1, \ldots, w_k) = C(t\sigma)$,
    for each non-constructor (constant)
    $\omega = (n : s_1 \times \ldots \times s_k \to s) \in \Omega \backslash \Omega_c, k \geq 0$,
    and for each constructor term $w_i \in T_{\Sigma_c, s_i}$, for every i

    where $t \in T_{\Sigma(X)}$ and $\sigma : Var(n(v_1, \ldots, v_k)) \to T_{\Sigma_c}$ are such that
    $n(v_1, \ldots, v_k) = t$ is an equation in Φ,
    $v_i\sigma = w_i$, for every i.

It can be proved that C is consistently and uniquely defined.

Example

Take the previous specification of the natural numbers.

- The canonical algebra C of this specification:

  $C(nat) = \{Succ^i(0) \mid i \in \mathbb{N}\}$,
  $C(0) = 0$,
  $C(Succ(w)) = Succ(w)$, for all $w \in C(nat)$,
  $C(+)(w, 0) = C(w)$, for all $w \in C(nat)$,
  $C(+)(w_1, Succ(w_2)) = C(Succ(w_1 + w_2))$,
  for all $w_1, w_2 \in C(nat)$.

- Sample reduction:

  $$\begin{aligned} & C(+)(0, Succ(0)) \\ &= C(Succ(0+0)) \\ &= C(Succ)(C(0+0)) \\ &= Succ(C(+)(C(0), C(0))) \\ &= Succ(C(+)(0,0)) \\ &= Succ(C(0)) \\ &= Succ(0) \end{aligned}$$

Properties of Constructive Specifications

Take constructive specification $sp = (\Sigma, \Phi, \Omega_c)$ with $\Sigma = (S, \Omega)$.

- The canonical algebra C of sp is a model of Φ.
  - It makes sense to take C as the meaning of the specification.
- The term rewriting system for sp is Noetherian and confluent.
  - Ground terms can be mechanically reduced to their normal form.
- Take initial specification $sp_I = (\Sigma, \Phi)$. Then $\mathcal{M}(sp) = \mathcal{M}(sp_I)$.
  - A constructive specification can be viewed as an initial specification.
- Take loose specification with free constructors $sp_L = (\Sigma, \Phi, S, \Omega_c)$. Then $\mathcal{M}(sp) = \mathcal{M}(sp_L)$.
  - A constructive specification can be viewed as a loose specification which is freely generated in all sorts.

Constructive specifications can be "executed"; properties and proof techniques of initial and loose specifications remain valid.

Example

loose spec
  sorts *nat*
  opns
    free constr $0 :\to nat$
    free constr $Succ : nat \to nat$
    $\_ + \_ : nat \times nat \to nat$
  vars $m, n : nat$
  eqns
    $m + 0 = m$
    $m + Succ(n) = Succ(m + n)$
endspec

This loose specification and the corresponding initial and constructive specifications define the same monomorphic abstract datatype.

Constructor Patterns

Take signature $\Sigma = (S, \Omega)$, set of variables X for Σ, set of constructors $\Omega_c \subseteq \Omega$, non-constructor $\omega = (n : s_1 \times \ldots \times s_k \to s) \in \Omega \backslash \Omega_c$ and define signature $\Sigma_c = (S, \Omega_c)$.

- A term $n(v_1, \ldots, v_k)$ is a constructor pattern for ω if:
  $v_i \in T_{\Sigma_c(X), s_i}$, for every i, and
  no variable in $n(v_1, \ldots, v_k)$ occurs more than once.
- A finite set of patterns P for ω is complete if $P \in \mathcal{P}(\omega)$ where $\mathcal{P}(\omega)$ is inductively defined as follows:
  - "Base" rule:
    $$\{n(x_1, \ldots, x_k)\} \in \mathcal{P}(\omega)$$
    where $x_1, \ldots, x_k$ are pair-wise different variables from X.
  - "Variable unfolding" rule: If $P \in \mathcal{P}(\omega)$ and $p \in P$, then any
    $$(P \backslash \{p\} \cup \{p[n_i(x_1, \ldots, x_{k_i})/x] \mid 1 \leq i \leq l\}) \in \mathcal{P}(\omega)$$
    where $x \in Var(p)$, s is the sort of x, l is the number of constructors in Ω_c of form $n_i : s_{i,1} \times \ldots \times s_{i,k_i} \to s$ and the variables $x_1, \ldots, x_{k_i}$ are pairwise different variables from X not in $Var(p) \backslash \{x\}$.

Example

Take the previously stated specification of natural numbers.

- Complete sets of constructor patterns for $\_ + \_ : nat \times nat \to nat$ are, for instance:
  $\{n + m\}$,
  $\{n + 0, n + Succ(m)\}$,
  $\{0 + m, Succ(n) + m\}$,
  $\{0 + 0, Succ(n) + 0, 0 + Succ(m), Succ(n) + Succ(m)\}$,
  $\{n + 0, n + Succ(0), n + Succ(Succ(m))\}$

Every complete set of constructor patterns for an operation "covers all cases" for the application of the operation.

Properties

Take signature $\Sigma = (S, \Omega)$ and set of constructors $\Omega_c \subseteq \Omega$ and define signature $\Sigma_c = (S, \Omega_c)$.

- Lemma: If P is a complete set of constructor patterns for non-constructor $\omega = (n : s_1 \times \ldots \times s_k \to s) \in \Omega \backslash \Omega_c$ and $n(w_1, \ldots, w_k)$ is a term with constructor terms $w_i \in T_{\Sigma_c, s_i}$, then:

  There exists exactly one pattern $p \in P$ and one substitution $\sigma : Var(p) \to T_{\Sigma_c}$ such that $w_i = v_i\sigma$, for every i.
- Theorem: If $\Phi \subseteq EL(\Sigma)$ is a finite set of equations that satisfies constraint (1), then the following is equivalent to constraint (2):
  - The left-hand sides of the equations $n(v1, \ldots, v_k) = t \in \Phi$ represent a complete set of constructor patterns for ω,

    for each operation $\omega = (n : s_1 \times \ldots \times s_k \to s) \in \Omega \backslash \Omega_c$ and $v_i \in T_{\Sigma_c(X), s_i}$, for all i.

A syntactic criterion to check constraint (2).

Example

Extend the specification of natural numbers as follows:

$$\_ \leq \_ : nat \times nat \to bool,$$
$$Even : nat \to bool.$$

$$0 \leq n = True,$$
$$Succ(m) \leq 0 = False,$$
$$Succ(m) \leq Succ(n) = m \leq n,$$

$$Even(0) = True,$$
$$Even(Succ(0)) = False,$$
$$Even(Succ(Succ(m))) = Even(m).$$

Now it is easy to check that the specification satisfies constraint (2).

Properties

Take signature $\Sigma = (S, \Omega)$ and set of constructors $\Omega_c \subseteq \Omega$ and define signature $\Sigma_c = (S, \Omega_c)$.

- Theorem: If $\Phi \subseteq EL(\Sigma)$ is a finite set of equations that satisfies constraints (1) and (2), then the conjunction of the following two conditions implies constraint (3):
  - The operations $\omega_j = (n_j : s_1 \times \ldots \times s_{k_j} \to s)$ of $\Omega \backslash \Omega_c$ can be ordered as a sequence $\omega_1, \ldots, \omega_d$ such that for each equation $(n_j(v_1, \ldots, v_{k_j}) = t_j) \in \Phi$ the following holds:

    $t_j \in T_{\Sigma_j(X), s}$ where $\Sigma_j = (S, \Omega_c \cup \{\omega_1, \ldots, \omega_j\})$.
    - No mutual recursion among operation definitions.
  - For each operation $n : (s_1 \times \ldots \times s_k \to s) \in \Omega \backslash \Omega_c$, each equation $(n(v_1, \ldots, v_k) = t) \in \Phi$, and each subterm $n(t_1, \ldots, t_k)$ of t:

    Every t_i is a subterm of v_i, and
    at least one t_i is a proper subterm of v_i.
    - In every equation, no argument "grows" and one argument "shrinks".

Syntactic criterion that is sufficient (not necessary) for constraint (3).

Properties

Take signature $\Sigma = (S, \Omega)$ and set of constructors $\Omega_c \subseteq \Omega$ and define signature $\Sigma_c = (S, \Omega_c)$.

- Theorem: If $\Phi \subseteq EL(\Sigma)$ is a finite set of equations that satisfies constraints (1) and (2), then the conjunction of the following two conditions implies constraint (3):
  - ...(as before)
  - For each operation $n : (s_1 \times \ldots \times s_k \to s) \in \Omega \backslash \Omega_c$, there exists an argument position j such that for each equation $(n(v_1, \ldots, v_k) = t) \in \Phi$, and each subterm $n(t_1, \ldots, t_k)$ of t:

    t_j is a proper subterm of v_j.
    - In all equations, the same argument "shrinks" (others may "grow").

Alternative criterion that is sufficient (not necessary) for constraint (3).

A Generalization

- Constructive specifications may use conditional equations

  $\phi_1 \Rightarrow n(v_1, \ldots, v_k) = t_1$
  $\ldots$
  $\phi_l \Rightarrow n(v_1, \ldots, v_k) = t_l$

  where the ϕ_i are first-order predicate formulas without quantifiers
  - that exclude each other mutually: $i \neq j \Rightarrow \neg(\phi_i \wedge \phi_j)$,
  - but whose disjunction holds: $\phi_1 \vee \ldots \vee \phi_l$.
- Example: abstract datatype "list of elements".

  $[\,].l = l$
  $Add(e, l).m = Add(e, l.m)$
  $Isprefix([\,], l) = True$
  $Isprefix(Add(e, l), []) = True$
  $e = e' \Rightarrow Isprefix(Add(e, l), Add(e', m)) = Isprefix(l, m)$
  $e \neq e' \Rightarrow Isprefix(Add(e, l), Add(e', m)) = False$

Summary

- Constructive specifications define monomorphic abstract datatypes.
  - Like initial specifications,
- Constructive specifications define abstract datatypes whose carriers can be represented as term languages.
  - Like loose specifications with free constructors.
- Constructive specifications always possess a model.
  - Unlike loose specifications.
- Model cannot collapse into algebra with singletons as carriers.
  - Unlike initial specifications.
- Constructive specifications can be "executed".
  - Various constraints have to be satisfied.
  - Comparatively "low-level" (less abstract) flavor.

Initial specifications are frequently written in a constructive fashion.