Completeness and Minimality of Rule-based Languages

Detlef Plump
University of York

Annegret Habel
Universität Oldenburg
Abstract objects and rules

Assumption 1

\( \mathcal{O} \) and \( \mathcal{R} \) are sets of objects and rules such that each \( S \subseteq \mathcal{R} \) induces a binary relation \( \Rightarrow_S \) on \( \mathcal{O} \).
Examples: objects and rules

**Graph rewriting:** Double-pushout approach ([HP 01], previous talk)

**String rewriting:**
- $O = \Sigma^*$ for some finite alphabet $\Sigma$
- $R = \{ l \rightarrow r \mid l, r \in \Sigma^* \}$
- $v \Rightarrow_S w$ if $v = xly$ and $w = xry$ for some $(l \rightarrow r) \in S$ and $x, y \in \Sigma^*$

**Term rewriting:**
- $O = T(F)$, set of closed terms over a finite set $F$ of function symbols
- $R = \{ l \rightarrow r \mid l, r \in T(F, X), l \notin X, \text{Var}(r) \subseteq \text{Var}(l) \}$
- $t \Rightarrow_S u$ if $u$ results from $t$ by replacing any subterm $\sigma(l)$ with $\sigma(r)$, where $(l \rightarrow r) \in S$ and $\sigma: T(F, X) \rightarrow T(F)$
Abstract programs

Syntax

(1) For every finite set $S$ of rules, $S$ and $S\downarrow$ are programs.

(2) If $P$ and $Q$ are programs, then $P; Q$ is a program.

Semantics

The semantics of a program $P$ is a binary relation $\llbracket P \rrbracket$ on $O$:

(1) $\llbracket S \rrbracket = \Rightarrow_S$ for every finite set $S$ of rules

(2) $\llbracket S\downarrow \rrbracket = \{ \langle x, y \rangle \mid x \Rightarrow^*_S y \text{ and } y \notin \text{Dom}(\Rightarrow_S) \}$

(3) $\llbracket P; Q \rrbracket = \llbracket P \rrbracket \circ \llbracket Q \rrbracket$  (relation composition)
Example: term rewriting program

Negate = Eval↓; \{true \to false, false \to true\}

where

\[
\text{Eval} = \begin{cases}
\neg(\text{true}) \to \text{false} & \text{true} \land x \to x & \text{true} \lor x \to \text{true} \\
\neg(\text{false}) \to \text{true} & \text{false} \land x \to \text{false} & \text{false} \lor x \to x
\end{cases}
\]

Note:

\begin{itemize}
\item \(F = \{\text{true, false, } \neg, \land, \lor\}\)
\item \([\text{Negate}] : T(F) \to T(F)\) satisfies
\end{itemize}

\[
[\text{Negate}](t) = \begin{cases}
\text{true} & \text{if } t \Rightarrow^*_{\text{Eval}} \text{false} \\
\text{false} & \text{if } t \Rightarrow^*_{\text{Eval}} \text{true}
\end{cases}
\]
Representation of objects as strings

**Assumption 2**

$\Sigma$ is a finite alphabet and $\alpha: \Sigma^* \rightarrow \mathcal{O}$ a surjective partial function.

**Graph rewriting:** $\alpha = \text{gra}: \Sigma^* \rightarrow A_C$ [HP 01]

**String rewriting:** $\alpha: \Sigma^* \rightarrow \Sigma^*$ is the identity function on $\Sigma^*$

**Term rewriting:**

- let $\Sigma \supseteq F$ and $[t]$ be the result of removing all parentheses and commas in a term $t$; then $[\cdot]$ is an injective function [Cohn 81]
- $\alpha: \Sigma^* \rightarrow T(F)$ is defined by

$$
\alpha(w) = \begin{cases} 
  t & \text{if there exists } t \in T(F) \text{ with } [t] = w, \\
  \text{undefined} & \text{otherwise}
\end{cases}
$$
Example: representation of terms

\[ g(f(a), h(b, c)) \]
Abstract computability

Let $O_1, O_2 \subseteq \mathcal{O}$. A partial function $f: O_1 \to O_2$ is *computable* if there is a partial computable string function $f': \alpha^{-1}(O_1) \to \alpha^{-1}(O_2)$ such that for all $w \in \alpha^{-1}(O_1)$:

1. $\alpha(w) \in \text{Dom}(f)$ implies $f(\alpha(w)) = \alpha(f'(w))$.
2. $\alpha(w) \notin \text{Dom}(f)$ implies $w \notin \text{Dom}(f')$.

\[
\begin{array}{ccc}
O_1 & \xrightarrow{f} & O_2 \\
\alpha\downarrow & & \alpha\downarrow \\
\alpha^{-1}(O_1) & \xrightarrow{f'} & \alpha^{-1}(O_2)
\end{array}
\]
Computational completeness

Completeness Condition
There is a total function \((\_)^* : \Sigma^* \rightarrow \mathcal{O}\) such that:

C1 (String functions) For every partial computable function \(f : S_1 \rightarrow S_2\) with \(S_1, S_2 \subseteq \Sigma^*\), there is a program \(P_f\) such that for all \(w \in S_1\) and \(x \in \mathcal{O}\),
\[w^*[P_f]x\] if and only if \(f(w)^* = x\).

C2 (Encoding) There is a program \(\text{Encode}\) such that for all \(x, y \in \mathcal{O}\),
\[x[\text{Encode}]y\] if and only if \(x = \alpha(w)\) and \(y = w^*\) for some \(w \in \Sigma^*\).

C3 (Decoding) There is a program \(\text{Decode}\) such that for all \(w \in \Sigma^*\) and \(y \in \mathcal{O}\),
\[w^*[\text{Decode}]y\] if and only if \(y = \alpha(w)\).
Completeness Condition in diagrams

\[ C1: (\cdot)^* \quad \xrightarrow{P_f} \quad (\cdot)^* \] Partial computable string functions can be simulated by programs on string-like objects.

\[ C2: \quad \xrightarrow{\text{Encode}} \quad \alpha \quad (\cdot)^* \quad \Sigma^* \] All objects are encoded as string-like objects.

\[ C3: \quad \xrightarrow{\text{Decode}} \quad \alpha \quad (\cdot)^* \quad \Sigma^* \] Decoding is inverse to encoding.
Example: encoding and decoding terms

\[ g(f(a), h(b, c)) \leftrightarrow \text{Encode} \leftarrow g_1(f_1(a_1(h_1(b_1(c_1(\Box)))))) \leftrightarrow \text{Decode} \]

\[ \alpha \]

\[ gfahbc \]
Computational completeness

Theorem (Computational completeness)
If the Completeness Condition is satisfied, then for every partial computable function \( f : O_1 \rightarrow O_2 \) there is a program that computes \( f \).

Corollary (Completeness of string, term and graph programs)
The languages of string, term and graph rewriting programs are computationally complete.
Minimality: graph, string and term programs

Theorem (Graph and string programs)
For graph and string rewriting, omitting any of the three programming constructs results in an incomplete language.

Theorem (Term programs)
For term rewriting, omitting either iteration or sequential composition results in an incomplete language.
Future work

• Can single-step application of rules be omitted
  – in term programs?
  – in string and graph programs computing functions not defined on the empty string/graph?

• Alternative way to guarantee completeness: adding contexts prior to computations
  – string rewriting: enclose input string in special markers
  – term rewriting: add special top symbol
  – graph rewriting: ?

• Applying the framework to other rule-based models of computation
The program $P = \text{Encode}; P_{f'}; \text{Decode}$ computes $f$, i.e. $\llbracket P \rrbracket \cap (O_1 \times O_2) = f$. 
Minimality: cyclic functions

Cyclic functions
A partial function \( f : \mathcal{O} \to \mathcal{O} \) is cyclic if there are \( o \in \mathcal{O} \) and \( n \geq 2 \) such that \( f(o) \neq o \) and \( f^n(o) = o \).

Examples: Reversal of graph edges, string reversal and Boolean term negation are cyclic functions.

Lemma
No cyclic partial function is computable by a program of the form \( P \downarrow \).

Proof: Let \( f : \mathcal{O} \to \mathcal{O} \) be cyclic and consider \( o \in \mathcal{O} \) and \( n \geq 2 \) such that \( f(o) \neq o \) and \( f^n(o) = o \). Suppose that there is a program \( P \downarrow \) such that \( \llbracket P \downarrow \rrbracket = f \). Then \( \llbracket P \downarrow \rrbracket(f^{n-1}(o)) = f^n(o) = o \), hence \( f^{n-1}(o) \Rightarrow^* P o \) and \( o \notin \text{Dom}(\Rightarrow_P) \). But \( \llbracket P \downarrow \rrbracket(o) = f(o) \) and \( o \neq f(o) \) imply \( o \Rightarrow^+_P f(o) \), a contradiction.
Minimality: neutral objects

Neutral objects

$o \in \mathcal{O}$ is neutral if for every finite set of rules $R$ and every $x \in \mathcal{O}$, $o \Rightarrow^+_R x$ implies $x \in \text{Dom}(\Rightarrow_R)$.

Examples: The empty graph and the empty string are neutral objects in graph and string rewriting, respectively.

Lemma

Let $f : \mathcal{O}_1 \rightarrow \mathcal{O}_2$ be a partial function such that there is a neutral element $o \in \text{Dom}(f)$ with $f(o) \neq o$. Then for every program $P_1; \ldots; P_n$ that computes $f$, one of the components $P_i$ is a set of rules.