プログラミング 4－6
（1995. 11. 30）

Towards a Foundation of Computational Reflection based on Abstract Rewriting

(Preliminary Result)

Takuo Watanabe

School of Information Science,
Japan Advanced Institute of Science and Technology

15 Asahidai, Tatsunokuchi, Ishikawa 923-12, Japan
Phone: +81-761-51-1256, Facsimile: +81-761-51-1149
E-Mail: takuo@jaist.ac.jp

The question we are considering is whether a *representation-independent* model of computational reflection exists. We use abstract rewriting systems as general operational models of computation. We define reflection based on the notion of *implementation morphisms* on ARSs. This paper presents our preliminary results.

Keywords: Reflection, Computational Reflection, Abstract Rewriting

抽象書換えにもとづく自己反映計算の基礎付け

渡部卓雄

北陸先端科学技術大学院大学
情報科学研究科

〒923-12 石川県能美郡辰口町旭台 15
電話: 0761-51-1256, ファクシミリ: 0761-51-1149
電子メール: takuo@jaist.ac.jp

抽象書換え系を用いて，自己反映計算に対する形式的な定義を試みる．これは自己反映計算およびそれに関連した諸概念 (例えばリフレクティブ・タワー等) に対する，特定の計算モデル・言語に依存しない形式的かつクリアーな定義方法を与えることを目的としている．本論文では，抽象書換え系に対する実現 (implementation) の関係を定義し，それによって自己反映計算の定義を行っている．

キーワード: 自己反映計算，リフレクション，抽象書換え系

1 Introduction

Computational reflection[4] (or simply, *reflection*) is the process in which a computational system can deal with itself, in the same ways that the system deals with its primary subject domain.

The question we are considering is whether a *representation-independent* model of computational reflection exists. We use abstract rewriting systems as general operational models of computation. We define reflection based on the notion of *implementation morphisms* on ARSs.

This paper presents our preliminary results; In Section 2, we first introduce basic tools — *abstract rewriting systems* (ARSs) and *implementation morphisms* on ARSs — for our construction. In Section 3, we formally define the notion of computational reflection using implementation morphisms.

2 Abstract Rewriting Systems

Abstract rewriting systems are commonly used as general operational models of computation. In this section, we introduce *implementation morphisms* between abstract rewriting systems, with which later we define the notion of computational reflection.

2.1 Basic Definitions

Definition 1 (Abstract Rewriting System) An *abstract rewriting system* (ARS) is a structure[1]

$$\langle \Sigma, \rightarrow \rangle$$

consisting of

- a set Σ and,
- an irreflexive binary relation $\rightarrow$ on Σ.

[1]We define an ARS with just one rewriting relation. Usually (for example, [1]), an ARS is defined as $\langle \Sigma, (\rightarrow_\alpha)_{\alpha \in I} \rangle$.

We write $s \rightarrow s'$ to indicate that $(s, s') \in \rightarrow$. Elements of Σ are called *states*. ■

The transitive closure (transitive reflexive closure) of $\rightarrow$ is written as $\rightarrow^+$ ($\rightarrow^*$); *i.e.*, we write $a \rightarrow^+ b$ ($a \rightarrow^* b$) if there is a sequence of rewriting steps $a = a_0 \rightarrow a_1 \rightarrow \cdots \rightarrow a_n = b$ for $n > 0$ ($n \geq 0$).

Definition 2 For any $s, s' \in \Sigma_A$, we say that s' is *reachable* from s if $s \rightarrow^* s'$. $\mathcal{R}_A(s)$ indicates all the elements of Σ_A reachable from s; *i.e.*,

$$\mathcal{R}_A(s) = \{s' \in \Sigma_A \mid s \rightarrow^* s'\}.$$

Further, if $\mathcal{R}_A(s) = \{s\}$, we call s a *normal form* of A. We write $NF_A(s)$ to indicate that s is a normal form of A. ■

Definition 3 (sub-ARS) Let

$$\begin{aligned} A &= \langle \Sigma_A, \rightarrow_A \rangle \quad \text{and} \\ B &= \langle \Sigma_B, \rightarrow_B \rangle \end{aligned}$$

be ARSs. Then A is a *sub-ARS* of B if

- $\Sigma_A \subseteq \Sigma_B$,
- $\rightarrow_A$ is the restriction of $\rightarrow_B$ to Σ_A; *i.e.*,

$$\forall s, s' \in \Sigma_A \,.\, s \rightarrow_B s' \Leftrightarrow s \rightarrow_A s'$$

and

- A is closed under $\rightarrow_B$; *i.e.*,

$$\forall s \in \Sigma_A, s' \in \Sigma_B \,.\, s \rightarrow_B s' \Rightarrow s' \in \Sigma_A.$$

We write $A \subseteq B$ when A is a sub-ARS of B. ■

2.2 Implementation Morphisms

Definition 4 (ARS-morphism) Let A and B be ARSs. A function $f : \Sigma_A \mapsto \Sigma_B$ is called an *ARS-morphism* if

$$s \rightarrow_A s' \;\Rightarrow\; f(s) \rightarrow_B f(s') \;\vee\; f(s) = f(s')$$

for any $s, s' \in \Sigma_A$. We write $f : A \mapsto B$ when f is an ARS-morphism. ■

Obviously, the identity function on Σ specifies the identity ARS-morphism. The composition of two ARS-morphisms is also an ARS-morphism. We say an ARS-morphism f is *partial* when f is a partial function on the states.

An ARS-morphism preserves reachable states:

Proposition 1 Let A, B be ARSs and $f : A \mapsto B$ be an ARS-morphism. For each $s \in \Sigma_A$,

$$f(\mathcal{R}_A(s)) \subseteq \mathcal{R}_B(f(s))$$

i.e., $\forall s' \in \mathcal{R}_A(s) \,.\, f(s') \in \mathcal{R}_B(s)$.

proof It is easily checked from the definitions of reachability (Definition 2) and ARS-morphisms (Definition 4). ■

There exist some trivial ARS-morphisms between any pair of ARSs. For example, a function which maps every element of Σ_A to an element of Σ_B should be an ARS-morphism. To model the *meta-relation* on ARSs properly, we pay our attention to the special ARS-morphisms which preserves *meaningful* computation.

Definition 5 (implementation morphism) Let A and B be ARSs. An ARS-morphism $f : A \mapsto B$ is an *implementation morphism* if

- $NF_A(s) \Rightarrow NF_B(f(s))$ and
- $NF_B(f(s)) \Rightarrow [\forall s' \in \mathcal{R}_A(s) \,.\, f(s') = f(s)]$

for any $s, s' \in \Sigma_A$. ■

Proposition 2 Let A, B and C be ARSs. Then the following holds.

1. The identity ARS-morphism (specified by the identity function on the set of states) is an implementation morphism.
2. If $f : A \mapsto B$ and $g : B \mapsto C$ are implementation morphisms, then the composition $g \circ f : A \mapsto C$ is also an implementation morphism.

proof

1. Obvious.
2. By Definition 5, $NF_A(s)$ implies $NF_B(f(s))$. So $NF_C(g(f(s)))$ holds. Conversely, $NF_C(g(f(s)))$ implies that g maps all the elements of $\mathcal{R}_B(f(s))$ to $g(f(s))$ (by Definition 5). By Proposition 1, $f(\mathcal{R}_A(s)) \subseteq \mathcal{R}_B(f(s))$. So $g \circ f$ maps all the elements of $\mathcal{R}_A(s)$ to $g(f(s))$. ■

Definition 6 An ARS-morphism $f : A \mapsto B$ is a *strong implementation morphism* if:

- $NF_A(s) \Rightarrow NF_B(f(s))$ and
- $\mathcal{R}_B(f(s)) = f(\mathcal{R}_A(s))$

for any $s \in \Sigma_A$. ■

It is easily checked that strong implementation morphisms are implementation morphisms.

Using implementation morphisms, we define the *implementation* relation between ARSs.

Definition 7 (implementation) Let A, B be ARSs. We say that A *implements* B *with* f (notation $A \rhd_f B$), if $f : A \mapsto B$ is an implementation morphism. If f is strong, we say that A *strongly implements* B with f (notation $A \overset{\star}{\rhd}_f B$). ■

We sometimes omit subscripts of $\rhd_f$, $\overset{\star}{\rhd}_f$ when which morphisms are used is not concern.

Proposition 3 $\rhd$, $\overset{\star}{\rhd}$ are reflexive and transitive.

proof By Proposition 2. ■

3 Formalizing Computational Reflection

Following Smith's observation in [2, 3], we can regard a computational system as a pair of a processor (interpreter) and its state (called *structural field* by Smith). We adopt an ARS for the processor and an element of ARS-states for state.

Definition 8 (computational system) A *computational system* is a structure

$$\langle A, s \rangle$$

where A is an ARS and s is an element of Σ_A. ■

A computational system can be regarded as again an ARS. If we take the set $\{A\} \times \Sigma_A$ as a state, we gain an ARS

$$\langle \{A\} \times \Sigma_A, \xrightarrow{I} \rangle$$

where $\xrightarrow{I}$ is defined as

$$\langle A, s \rangle \xrightarrow{I} \langle A, s' \rangle \overset{\text{def}}{\Leftrightarrow} s \rightarrow_A s'.$$

The rewriting rule $\xrightarrow{I}$ is called the *universal rule* of the system $\langle A, s \rangle$. Obviously, the following holds.

Proposition 4 $\langle \{A\} \times \Sigma_A, \xrightarrow{I} \rangle \overset{*}{\rhd} A$ ■

For a reflective system, in addition to the universal rule, there is a computation rule which affect the system itself, such as

$$\langle A, s \rangle \xrightarrow{r} \langle A', s \rangle.$$

where both A and A' are sub-ARS of an ARS. We call $\xrightarrow{r}$ a *reflective rule.*

Definition 9 (reflective ARS) An ARS $A = \langle \Sigma, \rightarrow \rangle$ is *reflective* if

$$A \rhd \langle \mathcal{A}_A \times \Sigma, \xrightarrow{I} \cup \xrightarrow{r} \rangle$$

where $\mathcal{A}_A$ is the set of sub-ARSs of A and $\xrightarrow{r}$ is a reflective rule. ■

4 Concluding Remark

Suppose that A is a reflective ARS. For any A' which is a sub-ARS of A, A implements A' with a non-trivial (non identity) implementation morphism. Because

$$\langle \mathcal{A}_A \times \Sigma, \xrightarrow{I} \cup \xrightarrow{r} \rangle \overset{*}{\rhd}_f A'$$

for any $A' \in \mathcal{A}_A$ and non-trivial f. In addition, $A' \in \mathcal{A}_A$ implements A with an implementation morphism which is a *partial* function on the states of A'. This implies that

Proposition 5 Let A be a reflective ARS. Then $A \rhd_f A$ for non-trivial f. ■

Since $\xrightarrow{r}$ is not empty, f cannot be identity.

The above proposition states the existence of reflective tower[2]; that is

$$\cdots \rhd_f A \rhd_f A \rhd_f A$$

with a non-trivial partial implementation morphism f.

References

[1] Klop, J. W. Term rewriting systems. In Abramsky, S. et al. eds, *HandBook of Logic in Computer Science*, volume 2, pp. 1–116. Oxford Univ. Press, 1992.

[2] Smith, B. C. Reflection and semantics in Lisp. In *Proc. of the ACM Symposium on Principles of Programming Languages (POPL)*, pages 23–35, 1984.

[3] Smith, B. C. What do you mean, **meta**? In *Proc. of ECOOP/OOPSLA '90 Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming*, 1990.

[4] Watanabe, T. A Tutorial Introduction to Computational Reflection. *Computer Software*, 11(3), pp. 5–14, JSSST, 1994 (in Japanese).