Metacomputation as a Tool for Formal Linguistic Modeling

Robert Glück[1] Andrei Klimov[2]

Department of Computer Science
University of Copenhagen
Universitetsparken 1
DK-2100 Copenhagen, Denmark

Keldysh Institute of Applied Mathematics
Russian Academy of Sciences
Miusskaya Square 4
125047 Moscow, Russia

Abstract

We consider the principle "a new model is a model of an existing one" as the main scheme for deriving new linguistic models by metacomputation. We derive the basic requirements for metacomputation by a structural analysis of different model definitions, and show that in order to automate the creation of linguistic models the following operations on linguistic models have to be performed by metacomputation effectively and efficiently: composition, inversion, and specialization of algorithms. This may also serve as a unifying paradigm for different program transformation approaches.

1. INTRODUCTION

During the last decades we have witnessed tremendous technological breakthroughs in the development and application of computers. The introduction of the computer was an evolutionary step in the control of formal linguistic models, a *metasystem transition* (MST). As a result the number of linguistic models created and used has significantly increased.

The method of modern science is, in its essence, the creation of *linguistic models* [1]. Informally, a *model* is a process which somehow mimics, or simulates, another primary process. Using a model it becomes possible for a system S to predict and know something about the primary process before it actually happens, or without performing it. As does every branch of science, computer science has its own types of objects, namely *formal linguistic models*, and the models of the models computer science creates itself. Linguistic models that can be executed on a computer, at least in principle, are referred to as *algorithms*, or *programs*.

The computer was really necessary before one could start to learn more about formal linguistic modeling on a large scale (human beings are neither precise enough, nor fast enough to carry out any but the simplest procedures). Just as mastering the general principle of using tools gives rise to the creation of industrial systems, mastering the principle of linguistic modeling gives rise to the creation of hierarchical systems of formal languages (on which modern science is based).

---

[1] Supported by the Austrian Science Foundation (grant no. J0780). On leave from the Univ. of Technology Vienna, Inst. für Computersprachen, Argentinierstr. 8, A-1040 Vienna, Austria. E-mail: glueck@diku.dk.

[2] Supported by the Russian Foundation for Fundamental Research (grant no. 93-012-628) and in part by the 'Österr. Forschungsgemeinschaft' (grant no. 06/1789). E-mail: And.Klimov@refal.msk.su.

In: Trappl R. (ed.), Cybernetics and Systems '94. Vol. 2, 1563-1570, World Scientific 1994.

The next evolutionary step in linguistic modeling is the control over the creation of formal linguistic models by the computer. In this paper we consider the principle "a new model is a model of an existing one" as the main scheme for the derivation of new models. This is performed by a process referred to as *metacomputation*. This viewpoint determines which operations metacomputation has to perform efficiently on formal linguistic models in order to evolve linguistic modeling: *composition*, *inversion*, and *specialization* of algorithms. Indeed, many known problems in computer science can be placed in these categories.

2. Linguistic Modeling and Metacomputation

In this paper, two aspects of linguistic modeling are studied (in both cases, *control* means the automation / mechanization of the respective activity):

control of executing models	=	computation
control of creating models	=	metacomputation

The use of the computer to execute models was the first step in controlling linguistic modeling, but it is not the last step. Indeed, the creation of linguistic models was not directly affected by the introduction of the computer. At the beginning, this activity was fully performed by the human ('programming'). Later, computer science has developed various methods to achieve more control over these activities, such as the development of new language paradigms, the construction of tools for generating special-purpose programs (e.g., scanners, parsers), and a variety of approaches for the verification, transformation, and compilation of programs. However, the basic problem still exists: how are we to achieve control over the creation of linguistic models, in general, and of programs, in particular?

Our approach is to use the method of linguistic modeling itself. *Metacomputation* is the creation of new models from existing models. This term underlines the fact that model creation is one metasystem level higher than computation. From now on we will refer to formal linguistic models as linguistic models, or simply as models.

Modeling The common notion of modeling, often referred to as the *modeling scheme*, is as follows (Fig. 1). Let an object o be in some state, which is characterized by the information x_o, and suppose the object performs an action. We shall denote by $y_o = F_o(x_o)$ the information about the ensuing state.[3] Suppose we want to make a prediction about y_o. Modeling introduces another object m, a model, for making predictions about the object o, considering the model 'similar', in some sense, to the object. A model m is an abstraction of the object o, that is, a model contains less information than the object.[1]

The mappings H_{in} and H_{out} are two abstraction functions, often referred to as homomorphisms. They map the information x_o, y_o about the object o to the information x_m, y_m in the model m (since we consider only linguistic models, we may safely assume that the functions H_{in}, H_{out} do not interfere with the behavior of the object o). Having full information x_o, y_o, we can deduce the corresponding information x_m, y_m in the model, but

---

[3] We do not assume that these actions are deterministic. $F_o(x_o)$ denotes any of the possible states after some action. We can think of F_o as a non-deterministic function, or a relation.

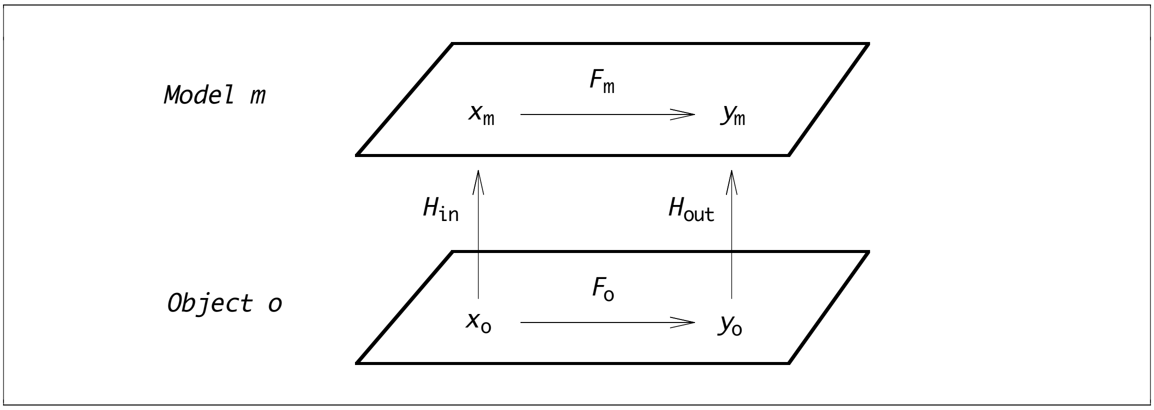

Fig. 1. Modeling scheme.

not vice versa. Having information y_m, we cannot, in general, predict a precise fact y_o about the object, but only that $H_{out}(y_o) = y_m$. That is, the possible outcome belongs to a set $\{y_o \mid H_{out}(y_o) = y_m\}$. Thus by using F_m one can predict, to some extent, the state of the object resulting from F_o:

$$y_m = F_m(H_{in}(x_o)) = H_{out}(F_o(x_o))$$

Creating models from models Let the object o itself be a model and assume that we want to create another model m being a model of the first model o. Initially we are provided with the object o: that is, a description of the domains X_o and Y_o over which x_o and y_o range, and a description of the function $F_o\colon \mathrm{X}_o \to \mathrm{Y}_o$. We need to define the domains X_m and Y_m which x_m and y_m range over, and the function $F_m\colon \mathrm{X}_m \to \mathrm{Y}_m$ of the derived model m.

What can be automated in the creation of a new model? The choice of the information available for building the new model is a creative step that depends on the external goals of the user. Consequently, we will not address the problem of how to choose homomorphisms H_{in} and H_{out}; this choice will be left to the user.

The main task is to automate the construction of the new function F_m. From the modeling scheme (Fig. 1) we see that F_m can be defined by using the mappings H_{in}^{-1} and H_{out} (where H_{in}^{-1} is an inverse of H_{in}). A full definition of F_m is provided by F_o, H_{in}^{-1} and H_{out}:[4]

$$\mathbf{def}\ \ F_m(x_m) = H_{out}(F_o(H_{in}^{-1}(x_m)))$$

Metacomputation The goal of metacomputation is to derive new models from such formal definitions. Let us denote the process of performing metacomputation by Mc. In order to express that metacomputation is applied to the textual definition of the model rather than to its denotation, we move the expression downwards (filling the remaining space with a line):

$$Mc(\underset{H_{out}(F_o(H_{in}^{-1}(x_m)))}{\underline{\hspace{12em}}}) \;\Rightarrow\; F_m$$

[4] To distinguish definitions from statements and equations, we use the keyword **def**. The function on the left hand side (e.g. F_m) is defined by the expression on the right-hand side.

We refer to such a formula as *MST-formula* because it describes the activity of creating models, which is a meta-activity as compared with just executing them. The essence of metacomputation is considering models as material that can be transformed and manipulated in various ways.

Requirements for metacomputation Two operations are involved in this formula:

- *composition:* F_o composed with H_{in}^{-1}, H_{out} composed with F_o
- *inversion:* H_{in}^{-1}

If metacomputation is capable of deriving efficient models defined by these operations, then the creation of formal linguistic models is automated to a large extent. In other words, performing these operations effectively and efficiently is a prerequisite for a successful application of metacomputation.

3. Selected Problems of Metacomputation

3.1 Problem of Program Composition

Consider the case when the information x_o on the object model o is identical to the information x_m on the new model m (Fig. 2). That is, the homomorphism H_{in} is the identity function: $x_m = H_{in}(x_o) = x_o$. The function of the new model is then defined as:

$$\textbf{def}\ \ F_m(x_m) = H_{out}(F_o(x_m))$$

Assume that we are interested in a small part of the output y_o. To define the new model, we provide the homomorphism H_{out} selecting the parts of y_o we are interested in. However, directly computing the above definition of the new model does not decrease the amount of computer resources needed to obtain the result y_m. In this case it might pay off to create a new model by metacomputation:

$$Mc(\underline{\qquad\qquad}) \Rightarrow F_m$$
$$H_{out}(F_o(x))$$

The new model may be more efficient and drastically reduced in size. One would expect that redundant computations in F_o are removed during metacomputation, and only those computations that are needed to produce the information selected by H_{out} are present in F_m. This problem has been studied in connection with *program slicing* [2].

Example A particular application of the metacomputation of composition is deriving an efficient interpreter from a compiler. *Compilation* is the process of translating programs from one language, say L, to another language, say M, where p_L and p_M are programs written in L and M respectively:

$$CompLM(p_L) \Rightarrow p_M$$

Interpretation is performing the activity implied by a program, say p_M, using another, universal program, say $IntM$, called the interpreter (below x is the initial information used in p_M, and y is the result):

$$IntM(p_M, x) \Rightarrow y$$

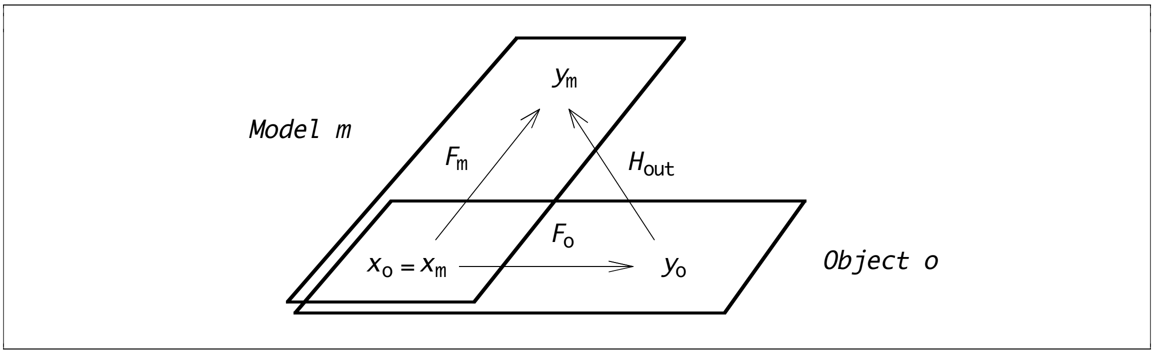

Fig. 2. Deriving a model by abstracting only the output information.

Assume that an L→M-compiler *CompLM* and an M-interpreter *IntM* are given. Then we can execute a program p_L in two stages: first by translating p_L into M, and then by interpreting the M-program:

$$IntM(CompLM(p_L),x) \quad \Rightarrow \quad y$$

That is, a new interpreter, *IntL*, may be defined by the composition of *CompML* and *IntM*:

$$\textbf{def} \;\; IntL(p,x) \;=\; IntM(CompLM(p),x)$$

However, the step via the intermediate language M may be rather inefficient. If we metacompute this definition, we may obtain a more efficient interpreter.

$$Mc(\underset{IntM(CompLM(p),x)}{\underline{\qquad\qquad\qquad}}) \quad \Rightarrow \quad IntL$$

This is a metasystem transition over the computation process defined by the composition. The importance of effectively metacomputing the composition of models is hard to overestimate, since composition is one of the basic methods for building new models from existing ones.

3.2 Problem of Program Inversion

Consider the case of deriving a new model when the output information y_o on the object model is identical to the output information y_m on the new model (Fig. 3). That is, the homomorphism H_{out} is the identity function: $y_m = H_{out}(y_o) = y_o$. The function F_m of the model is then defined as follows:

$$\textbf{def} \;\; F_m(x_m) \;=\; F_o(H_{in}{}^{-1}(x_m))$$

As in the case of composition, one may derive a new model F_m by metacomputation:

$$Mc(\underset{F_o(H_{in}{}^{-1}(x_m))}{\underline{\qquad\qquad}}) \quad \Rightarrow \quad F_m$$

In this case a combination of composition and inversion is used. We want to derive a new model that can be used to make a prediction y_m using the partial information x_m about x_o. The homomorphism H_{in} defines what is to be known about x_o: $x_m = H_{in}(x_o)$. In some instances, the partial information x_m will be sufficient to produce y_m. Then $F_m(x)$ should

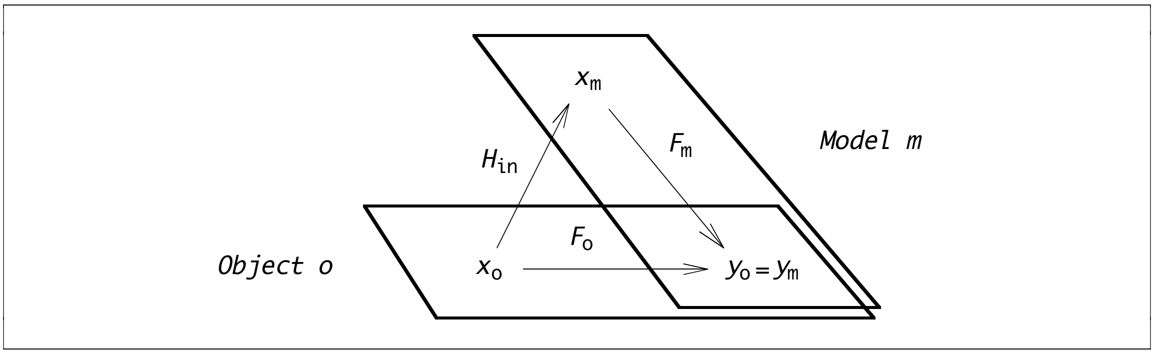

Fig. 3. Deriving a model by abstracting only the input information.

return it. However, generally x_m does not define y_m precisely. In this case, one is interested either in *one* of the possible results, or in the set of *all* possible results. These two choices correspond to two different kinds of inversions.

The problem of constructing an inverse relation is interesting in its own right, since many mathematical problems are stated in the following way: given the description P of some properties of objects, find (at least one) object x such that $P(x)$ holds. This is referred to as the inversion problem. The predicate P may be formulated as an algorithm checking the linguistic object x.

The inversion of programs is a fundamental problem, and a large branch of computer science has been based on solutions emerging from logic and proof theory [3,4]. Direct methods for inverting algorithms have been developed [5-7]. By varying the metaevaluator Mc and the method for solving the inverse problem different linguistic models can be generated by MST-formulas involving composition and inversion.

3.3 Problem of Program Specialization

Automatic inversion by metacomputation is a hard problem, and hence it is important to consider variants of the modeling scheme without inversion. This leads us to the problem of specialization of models.

The relation of x_o and x_m may be established not only by a homomorphism from x_o to x_m, but by a mapping, say G, from some information x_m, given in the model, to x_o. That is, the inverse mapping $G(x_m) = H_{in}^{-1}(x_m)$ is supplied by the user. This corresponds to changing the direction of the arrow marked with H_{in} (Fig. 3). The function F_m of the new model is then defined by:

$$\textbf{def}\ F_m(x_m) = F_o(G(x_m))$$

The problem in this definition has the same structure as the problem of composition. However, since the modeling function F_o is usually much more complex than the mappings H_{out} and G, there is a difference between the two cases: in Section 3.1, the outer function is simpler, and here, the inner function is simpler. Different methods of metacomputation may be advantageous in each case. The problem of specialization falls into the second case.

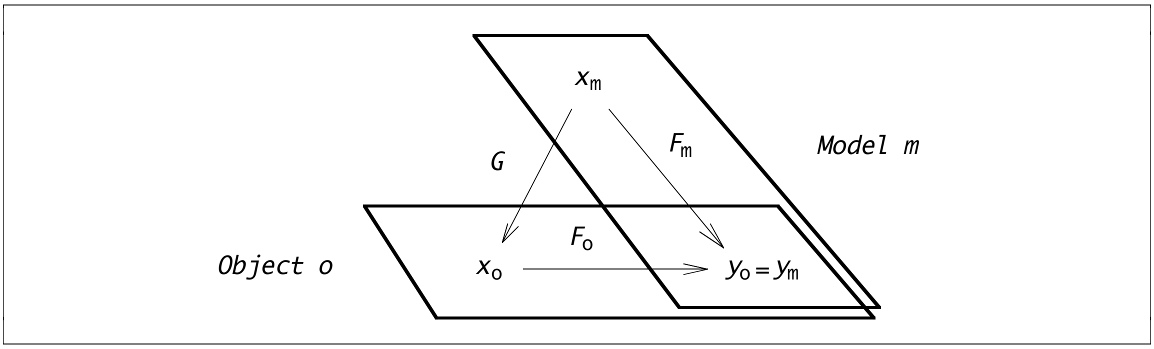

Fig. 4. Deriving a model by restricting the input information.

The problem of specialization arises when the domain for which a model is used is restricted (Fig. 4). The function F_s of the specialized model is the same as the function F_o of the model o, but the input ranges only over a part of the domain of model o:

$$\textbf{def}\ F_s(x) = F_o(x)\ \textbf{if}\ x \in S,\ \text{where } S \text{ is a subset of the domain of model } o$$

Although the definitions of F_s and F_m above are not formally equivalent, in the majority of the cases, the definition of F_s can be replaced by one, where the mapping $G(x_m)$ is a representation of the set S: $S = \{G(x_m) \mid x_m \in X_m\}$.

Example Consider a model whose function F_o has several arguments, such as $F_o(x_1,x_2)$ (formally it takes a tuple). Then the mapping G may return a tuple in which parts of the arguments are fixed to some information. Assume that x_1 is always mapped to the same information I. Then x_2 is the remaining parameter, and $G(x_2) = (I,x_2)$:

$$\textbf{def}\ F_m(x_2) = F_o(I,x_2)$$

As usually, one may derive a more efficient model F_m by metacomputing the definition:

$$Mc(\underline{\hspace{4em}}) \Rightarrow F_m$$
$$F_o(I,x_2)$$

The motivation to metacompute the definition of a model whose input domain is restricted is to remove redundant computations that may be present in the object model but are not necessary for the narrowed domain. This can give substantial savings, e.g., when one parameter, say x_1, changes less frequently than another.

Surprisingly, many problems in computer science reduce to the problem of specialization, including the central problem of metacomputation: the problem of *self-applying* metacomputation. It was found [8] that the solution to the problem of generating compilers from interpreters requires neither composition nor inversion of programs, just fixing some of the arguments is sufficient (as in the example above).

Although the problem of specialization is a special case of the problem of composition, it is worth considering it separately because of the large number of practical problems that require specialization. This gave rise to a new branch in computer science, called *partial evaluation*, which has been developing rapidly during the past decade due to advances achieved both in theory and practice [9].

4. CONCLUSION

Computer science, as compared to other disciplines, appears as a rather diverse field lacking a clear focus and a notion of what has to be achieved. In this contribution we tried to emphasize common roots of different problems. We discussed how the notion of *linguistic modeling* may serve as a unifying viewpoint and derived the requirements for *metacomputation* by a structural analysis of the problem. We saw that to solve the problem of linguistic modeling, metacomputation must perform the following operations efficiently: *composition*, *inversion*, and *specialization* (the latter being a special, though important, case of composition). This task may serve as a clear guideline for research in metacomputation.

The goal is to make the automatic derivation of models by metacomputation a practical tool. Since the 60s many solutions have been tried and some progress has been achieved, but the basic problems still remain open. We say that the next evolutionary step in formal linguistic modeling, the next large-scale metasystem transition, is achieved if efficient linguistic models can be created by the computer and it suffices for the human to make initial formal definitions. The ultimate goal is to achieve the ability for an arbitrary series of metacomputations over linguistic models to be just an ordinary, mechanical process. In this sense, we are actually working towards the next metasystem transition in linguistic modeling. We believe that this is one of the most challenging tasks of computer science.

REFERENCES

1. V. F. Turchin, *The Phenomenon of Science.* Columbia University Press: New York (1977).
2. S. Horwitz, T. Reps and D. Binkley, "Interprocedural slicing using dependence graphs", *ACM TOPLAS* **12**, 26-60 (1990).
3. R. Kowalski, "Algorithm = logic + control", *Communications of the ACM* **22**, 424-436 (1979).
4. S. M. Abramov, "Metacomputation and logic programming", *Programmirovanie* (3), 31-44 (1991), in Russian.
5. V. F. Turchin, "Equivalent transformations of recursive functions defined in Refal", *Teorija Jazykov i Metody Programmirovanija (Proceedings of the Symposium on the Theory of Languages and Programming Methods)*, 31-42, (1972), in Russian.
6. A. Y. Romanenko, "Inversion and metacomputation", *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, 12-22, ACM Press (1991).
7. P. G. Harrison, "Function inversion", D. Bjørner, A. P. Ershov and N. D. Jones (ed.), *Partial Evaluation and Mixed Computation*, 153-166, North-Holland (1988).
8. Y. Futamura, "Partial evaluation of computation process - an approach to a compiler-compiler", *Systems, Computers, Controls* **2**, 45-50 (1971).
9. N. D. Jones, C. K. Gomard and P. Sestoft, *Partial Evaluation and Automatic Program Generation.* Prentice Hall International Series in Computer Science. Prentice Hall: New York, London, Toronto (1993).

— *Dedicated to the memory of Alexander Romanenko* —