A THEORETICAL APPROACH TO PROGRAM INVERSION

Á.Fóthi and J.Nyéky-Gaizler (Budapest, Hungary)

Abstract. The aim of this paper is to demonstrate how the mathematical model of programming presented in [2,3,9] can be applied. The problem of program inversion, a well-known and widely used program transformation, described informally by M.Jackson as a solution to the problem of structure clashes, is adapted here to this model. Using the notion of elementwise processing, the concepts of elementwise producer and elementwise consumer programs are introduced. With the help of these it is proved how one can eliminate the intermediate sequential files needed in the original solution of the problem while creating an "inverted" program as a result.

1. Introduction

At the present time the importance of reliability of the programs became underlined by the improving quality of hardware and by the increasing number of different fields where computer science takes place. Formal methods were defined where techniques are available to prove that a system satisfies a given specification [1,2,3,4,5,8,9]. The solution of a programming task is almost always based on abstraction. The depth of this abstraction depends on the level of generality of the concept, created by the programmer for modeling the problem.

The methods, similar to the approach of F. Vivarès [8], which propose practical strategies to help in building a system, are relatively rare. Our model serves this purpose, this means we are interested in giving a formal method for designing and constructing correct programs.

In the present paper our abstract approach is presented by proving the theorem of program inversion, a well-known and widely used program transformation, described only informally by M.Jackson as possible solution to "structure clashes" [6].

2. Preliminary definitions and notions

The basic definitions and concepts of our mathematical model of programming is given in [9]. For the sake of brevity we do not repeat definitions given in [9] (printed in this volume). The definitions of binary relation, domain and range of a relation, $(\mathcal{D}_R, \mathcal{R}_R)$, strict composition of relations $(Q \odot P)$, function, reduced sequence($\text{red}(\alpha)$) are given in [9]. The notions and definitions of the mathematical model used, i.e. state space (A), variable $(v_i : A \rightarrow A_i)$, problem, program, program function $(p(S))$, solution, weakest precondition, program constructions (sequence, alternative construction, loop), type specification, type, type constructions (direct product, union, iteration), some functions defined on sequence types (e.g. dom, lov, hiext, lorem), and the notion of elementwise processable functions are also defined in [9].

Some supplements which will be needed during the solution of the problem described above:

a) The program function $p(\mathbf{S})$ of the program $\mathbf{S}$ is said to be **independent** from the state space component A_i, if

$$\forall a, b \in \mathcal{D}_{p(\mathbf{S})} : (\forall k \in ([1, i-1] \cup [i+1, n]) : a_k = b_k) \rightarrow p(\mathbf{S})(a) = p(\mathbf{S})(b)$$

b) $\overline{R}$ is called the **closure** of the relation $R \subseteq A \times A$, if

$$\mathcal{D}_{\overline{R}} \stackrel{\text{def}}{=} \{a \in A | \not\exists L_R(a_0, \ldots\ldots) : a_0 = a\},$$

$$\overline{R}(a) \stackrel{\text{def}}{=} \{a' \in A | \exists L_R(a_0, \ldots, a_q) : a = a_0 \wedge a' = a_q \wedge a_q \notin \mathcal{D}_R\},$$

where $L_R(a_0, \ldots, a_q)$ is a *length q point-chain* of the relation $R \subseteq A \times A$, if $\forall i \in [1, q] : a_i \in R(a_{i-1})$. The a_q is called the *end-point* of the point-chain, if $a_q \notin \mathcal{D}_R$.

$L_R(a_0, \ldots\ldots)$ is an *infinite point-chain* of the relation $R \subseteq A \times A$, if $\forall i \in N : a_i \in R(a_{i-1})$.

c) Let $R \subseteq A \times A$ be a relation and π be a logical statement over A.

The relation $R|_\pi \stackrel{\text{def}}{=} (R \cap ([\pi] \times A)) \cup \{(a, a)|[\pi] \setminus \mathcal{D}_R\}$ is the **restriction of the relation R to π** .

d) $\overline{R|_\pi}$ is the closure of the relation R with respect to the condition π.

e) The theorem describing the program function of loops: $p(DO(\pi; S_0)) =$
$= \overline{p(S_0)|_\pi}$ is proved in [2].

f) In this paper the simplest form of elementwise processing will only be used, this is the so called "first version", where the function $f : \mathbf{X} \to$
$\to \mathbf{Y}$, $\mathbf{X}, \mathbf{Y} \subseteq 2^H$ is single-valued and depends only on one variable:

$$A = \mathbf{X} \times \mathbf{Y}, \quad x : \mathbf{X}, \; y : \mathbf{Y},$$
$$B = \mathbf{X}, \quad x' : \mathbf{X},$$
$$Q_{x'} = (x = x'),$$
$$R_{x'} = (y = f(x'))$$

It can be proved, that the program $\mathbf{S}$ is the solution of this specifiation:
$\mathbf{S} ::= (\; y := \emptyset; \; DO((f(x) \neq \emptyset) : (e : mem(x); \; y := y \tilde{\cup} f(\{e\}); \; x := x \tilde{-} \{e\})\;))$,
with the invariant $P_{x'} = (y \cup f(x) = f(x') \wedge y \cap f(x) = \emptyset)$.

g) Using the notion of the weakest precondition, the appropriate operations which achieve the functions defined on type-constructions in [9], can be specified. The specifications of the operations needed later are:

Let us suppose that $T = seq(T_0)$.

Let $A = T \times N$ and $m : N$, $t : T$.

i) $wp(m := t.dom, R) = R^{m \leftarrow dom(t)}$

Let $A = T \times T_0$ and $t_0 : T_0$, $t : T$.

ii) $wp(t_0 := t.lov, R) = dom(t) \neq 0 \wedge R^{t_0 \leftarrow lov(t)}$

iii) $wp(t : hiext(t_0), R) = R^{t \leftarrow hiext(t,t_0)}$

Let $A = T$ and $t : T$.

iv) $wp(t : lorem, R) = dom(t) \neq 0 \wedge R^{t \leftarrow lorem(t)}$

h) Supposing $A = \mathbf{U} \times \mathbf{V}$, the problem $F \subseteq A \times A$ describes the evaluation of a function $f : \mathbf{U} \to \mathbf{V}$ at some $u' \in \mathbf{U}$ if $F = \{((u', v), (u, f(u'))) \mid \forall u, u' \in \mathbf{U}, \forall v \in \mathbf{V}\}$.

3. The Jackson method

Jackson's method [6,7] allows to build programs from the logical structure of their inputs and outputs. The steps are:

- Describe precisely the structure of program inputs and outputs by means of Jackson's trees.

- Merge both structures in order to obtain the structure of the program (this is the so called structure text).

- Build the list of elementary functions allowing to compute all values of elementary outputs.

- Assign a list of operations to every leaf of the structure text.

- Add the boolean expressions used for selection and iteration and translate the structure text into the target language.

In our model this means that:

- Determine the state space of the problem, realize the internal structure of data types (to which type constructions they belong), and refine them then step-by-step until the level of elementary data types. The obvious correspondance between Jackson's trees and type constructions defined above can be given by:

i) A leaf representing an atomic data corresponds to an elementary data type.

ii) The selection constructor corresponds to the union of data types.

iii) The sequence constructor corresponds to the direct product of data types.

iv) The iteration constructor corresponds to the iteration of a data type.

- Check at each level whether an elementwise processable function, which results the values of elementary outputs can be determined, and if this is the case, then define it.

- Create the solving program using the correspondance between type constructions and program constructions:

i) The direct product of data types will be handled with a composition of functions, this yields a sequence of programs.

ii) The case of union of data types will be handled with functions defined by alternative construction, i.e. an alternative construction of programs will solve it.

iii) Handling an iteration of a data type yields a loop construction with a loop condition depending on the length of the iteration.

- Code the result into the target language.

It is easy to justify that these correspondances hold.

4. The theorem of program inversion

The Jackson method however is only applicable if an elementwise processing function, which describes the connection between the input and output data structures, can be defined. Problems arise only if we cannot determine an elementwise processable function which results in the values of elementary outputs. This is the case of "structure clashes". Jackson distinguishes the cases of "boundary clash" and "ordering clash".

The solution to these problems is given with a sequence of programs, using intermediate file(s), produced by the first program and consumed by the second one. The program using intermediate file(s) is in some cases - the precise conditions will be investigated now - convertable to a program, which produces an element of the intermediate file, and then instead of writing it into the file, consumes it immediately while producing the result of the problem.

The problem can be formulated in general as follows:

Notations: Let $\mathbf{U}$, $\mathbf{V}$, $\mathbf{E}$, $\mathbf{T}$ be arbitrary type value sets, $\mathbf{X} = seq(\mathbf{E})$, $\mathbf{Y} =$

$= seq(\mathbf{T})$ sequences created from $\mathbf{E}$ and $\mathbf{T}$ respectively.

Suppose, there is given a function $f : \mathbf{U} \to \mathbf{V}$, which is a composition: $f = f_1 \odot f_2 \odot f_3$, where $f_1 : \mathbf{U} \to \mathbf{X}$, $f_2 : \mathbf{X} \to \mathbf{Y}$, $f_3 : \mathbf{Y} \to \mathbf{V}$ and the task is to find a program, which solves the problem $F \subseteq A \times A$ given with the specification:

$$A = \mathbf{U} \times \mathbf{V} \quad u : \mathbf{U}, \quad v : \mathbf{V}.$$

$$B = \mathbf{U} \quad u' : \mathbf{U}.$$

$$Q_{u'} = (u = u'),$$

$$R_{u'} = (v = f(u')).$$

Suppose that there exist programs $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$ which solve f_1, f_2, f_3 respectively, then the program $\mathbf{S} ::= (\mathbf{S}_1; \mathbf{S}_2; \mathbf{S}_3)$ will solve f accordingly to the derivation rule of the sequential construction [9], while the original state space $A = \mathbf{U} \times \mathbf{V}$ will temporarily be increased by new components:

$A' = \mathbf{U} \times \mathbf{V} \times \mathbf{X} \times \mathbf{Y}$. If $\mathbf{X}$ and $\mathbf{Y}$ are sequences, then this means, that this program generates intermediate sequences. Now a sufficient condition for the elimination of $\mathbf{X}$ and $\mathbf{Y}$ will be formulated. If this condition holds, the solving program $\mathbf{S}$ can be transformed to an other program $\mathbf{S}'$, which solves the original problem F without creating the intermediate sequences $\mathbf{X}$ and $\mathbf{Y}$.

Definition 4.1. (elementwise producer program) The program $\mathbf{S}_1 \subseteq$ $\subseteq A_1 \times A_1^{**}$,

$$\mathbf{S}_1 ::= (\mathbf{S}_{11};\ DO(\pi_1 :\ \mathbf{S}_{12}))$$

will be called an elementwise producer program if

1) $\mathbf{S}_1$ is a solution to the problem defined by the specification:

$$A_1 = \mathbf{U} \times \mathbf{X} \quad u : \mathbf{U},\quad x : \mathbf{X}.$$
$$B = \mathbf{U} \quad u' : \mathbf{U}.$$
$$Q_{u'} = (u = u'),$$
$$R_{u'} = (x = f_1(u')).$$

2) $\mathbf{S}_{11} ::= (\mathbf{S}_{11}';\ x :=<>)$, where $\mathbf{S}_{11}'$ is a program over $A = \mathbf{U}$ and it is independent from the component $\mathbf{X}$ of the state space.

3) $\mathbf{S}_{12} = (\mathbf{S}_{12}';\ x : hiext(e))$ where $e \in \mathbf{E}$ and $\mathbf{S}_{12}'$ is a program over $A = \mathbf{U} \times \mathbf{E}$ and it is independent from the component $\mathbf{X}$ of the state space.

Denote by n the number of elements in the resulted sequence, which will primarily be determined - as it follows from the derivation rule of the loop [9] - by the loop condition π_1.

In this case the problem solved by $\mathbf{S}_1$ can essentially be described as $p(\mathbf{S}_1) = g_{11} \odot g_{12}^n$, where $g_{11}, g_{12} \quad A_1 \rightarrow A_1$, $g_{11} = p(\mathbf{S}_{11})$ and $g_{12} = p(\mathbf{S}_{12})|_{\pi_1}$.

Remark: Elementwise processable functions can also be solved by elementwise producer programs.

Suppose, that f_2 is an **elementwise processable function for sequences**. This means, that it can be evaluated with the program $\mathbf{S}_2 \subseteq$ $\subseteq A_2 \times A_2^{**}$,

$\mathbf{S}_2 ::= (\ y :=<>\ ;\ DO((f_2(x) \neq <>) :\ \mathbf{S}_{22}\ ))$, where

$\mathbf{S}_{22} ::= (\ e := lov(x);\ t := f_2(\{e\});\ y : hiext(t);\ x : lorem\ )$,

according to the specification:

$$A_2 = \mathbf{X} \times \mathbf{Y} \quad x : \mathbf{X}, \quad y : \mathbf{Y}.$$
$$B = \mathbf{X} \quad x' : \mathbf{X}.$$
$$Q_{x'} = (x = x'),$$
$$R_{x'} = (y = f_2(x')).$$

In this case the problem solved by $\mathbf{S}_2$ can essentially be described as $p(\mathbf{S}_2) = g_{21} \odot g_{22}^n$, where $g_{21}, g_{22} : A_2 \rightarrow A_2$, g_{21} is the problem solved by $(y := <>)$, this means $g_{21} = (y = <>)$, and $g_{22} = p(\mathbf{S}_{22})|_{(f_2(x) \neq <>)}$.

Definition 4.2. (**elementwise consumer program**) The program $\mathbf{S}_3 \subseteq A_3 \times A_3^{**}$,

$$\mathbf{S}_3 ::= (\, \mathbf{S}_{31}; \, DO((y \neq <>) : \mathbf{S}_{32}); \, \mathbf{S}_{33} \,)$$

will be called an elementwise consumer program if

1) $\mathbf{S}_3$ is a solution to the problem defined by the specification:

$$A_3 = \mathbf{Y} \times \mathbf{V} \quad y : \mathbf{Y}, \quad v : \mathbf{V}.$$
$$B = \mathbf{Y} \quad y' : \mathbf{Y}.$$
$$Q_{y'} = (y = y'),$$
$$R_{y'} = (v = f_3(y')).$$

2) $\mathbf{S}_{31}$ and $\mathbf{S}_{33}$ are independent from the component $\mathbf{Y}$ of the state space,

3) $\mathbf{S}_{32} ::= (\, t := lov(y); \, \mathbf{S}_{32}'; \, y : lorem)$ where $\mathbf{S}_{32}'$ is a program over $\mathbf{T} \times \mathbf{V}$ and it is independent from the component $\mathbf{Y}$ of the state space.

In this case the problem solved by $\mathbf{S}_3$ can essentially be described as $p(\mathbf{S}_3) = g_{31} \odot g_{32}^n \odot g_{33}$, where $g_{31}, g_{32}, g_{33} : A_3 \rightarrow A_3$, $g_{31} = p(\mathbf{S}_{31})$, $g_{32} = p(\mathbf{S}_{32})|_{(y \neq <>)}$, and $g_{33} = p(\mathbf{S}_{33})$.

Remark. Elementwise processable functions can also be solved by elementwise consumer programs, but the programs solving the problems of linear or maximum searching, the counting or the summation etc. are also elementwise consumer programs.

Now with the help of these definitions and notations the theorem can be formulated in the following way:

Theorem 4.1. *Let $f : \mathbf{U} \rightarrow \mathbf{V}$ be a function, for which $f = f_1 \odot f_2 \odot f_3$, where $f_1 : \mathbf{U} \rightarrow \mathbf{X}, f_2 : \mathbf{X} \rightarrow \mathbf{Y}, f_3 : \mathbf{Y} \rightarrow \mathbf{V}, f_2$ be an elementwise processable function (thus it can be solved by the elementwise processable*

program S_2), *and* f_1 *and* f_3 *be functions, which can be solved by the elementwise producer program* S_1 *and elementwise consumer program* S_3, *respectively. If this holds, then - using the above notations -* f *can be solved with the program:*
$$S' ::= (S'_{11}; S_{31}; DO(\pi_1 : (S'_{12}; t := f_2(\{e\})); S'_{32}); S_{33}).$$

Proof. Since the program $S ::= (S_1; S_2; S_3)$ solves the problem described with the function f, we have to prove, that if the conditions hold, the program S' also solves it.

The function solved by S can essentially be described as

$$f = g_{11} \odot g_{12}^n \odot g_{21} \odot g_{22}^n \odot g_{31} \odot g_{32}^n \odot g_{33}.$$

Investigating these functions we get the following:

$g_{11} = f_{11} \odot f'_{11}$, where

$\quad$ $f_{11} : U \rightarrow U$ is the function evaluated by S'_{11}, and it is independent from X,

$\quad$ $f'_{11} = (x = <>)$;

$g_{12} = f_{12} \odot f'_{12}$, where

$\quad$ $f_{12} : U \rightarrow U \times E$ is the function evaluated by S'_{12}, and it can be described as

$\quad$ $f_{12} = (f_{121}, f_{122})$ where $f_{121} : U \rightarrow U$, $f_{122} : U \rightarrow E$;

$\quad$ $f'_{12} : X \times E \rightarrow X$; $f'_{12} = hiext_{(x,e)}$;

Thus the resulting sequence is:
$$x = < f_{122}(f_{11}(u')), f_{122}(f_{121}(f_{11}(u'))), \ldots f_{122}(f_{121}^{n-1}(f_{11}(u'))) >.$$

$g_{21} = (y = <>)$,

$g_{22} = f_{221} \odot f_2 \odot f_{222} \odot f_{223}$, where

$\quad$ $f_{221} : X \rightarrow E$; $f_{221} = lov_x$;

$\quad$ $f_2 : E \rightarrow T$ is the elementwise processable function;

$\quad$ $f_{222} : Y \times T \rightarrow Y$; $f_{222} = hiext_{(y,t)}$;

$\quad$ $f_{223} : X \rightarrow X$; $f_{223} = lorem_x$;

Therefore for each element of the resulting sequence y holds: $y_i = f_2(x_i)$.

$g_{31} = f_{31}$, $f_{31} : V \rightarrow V$ is the function evaluated by S_{31};

$g_{32} = f_{321} \odot f'_{32} \odot f_{322}$;

$\quad$ $f_{321} : Y \rightarrow T$; $f_{321} = lov_y$;

$\quad$ $f'_{32} : V \times T \rightarrow V$; f'_{32} is the function evaluated by S'_{32};

$\quad$ $f_{322} : Y \rightarrow Y$; $f_{322} = lorem_y$;

$g_{33} = f_{33}$, $f_{33} : V \rightarrow V$ is the function solved by S_{33}.

The transformation performed by $(\mathbf{S}_{31}; DO((y \neq <>) : \mathbf{S}_{32}))$ can be described with the recursive function: $G_0 = f_{31}$, $G_{i+1} = f'_{32}(y_{i+1}, G_i)$.

Using these notations, the problem solved by $\mathbf{S}'$ can be described as

$$f' = f_{11} \odot f_{31} \odot (f_{12} \odot f_2 \odot f'_{32})^n \odot f_{33}.$$

Since the composition of functions working on different subspaces of a state space is always commutative, taking into account the appropriate domains and ranges of the above functions, we might conclude, that it is allowed to commute the functions g_{12}^n with g_{21}, g_{31} with g_{22}^n and with g_{12}^n in the sequence of compositions in f. Thus we get

$f = g_{11} \odot g_{12}^n \odot g_{21} \odot g_{22}^n \odot g_{31} \odot g_{32}^n \odot g_{33} \quad \Rightarrow$

$f = g_{11} \odot g_{21} \odot g_{31} \odot g_{12}^n \odot g_{22}^n \odot g_{32}^n \odot g_{33}.$

Using the above transformations, we get that for each i:

$g_{12}^i \odot g_{22}^i \odot g_{32}^i = (g_{12} \odot g_{22} \odot g_{32})^i.$

Consequently:

$f = g_{11} \odot g_{21} \odot g_{31} \odot (g_{12} \odot g_{22} \odot g_{32})^n \odot g_{33}.$

Using the refinements of the functions:

$f = (f_{11} \odot f'_{11}) \odot (y = <>) \odot f_{31} \odot ((f_{12} \odot f'_{12}) \odot (f_{221} \odot f_2 \odot f_{222} \odot f_{223}) \odot$
$\odot (f_{321} \odot f'_{32} \odot f_{322}))^n \odot f_{33}.$

Substituting the definitions of the functions appropriately, it is easy to see, that one can distunguish between functions working with $\mathbf{X}$ or $\mathbf{Y}$, and those, which have no connection with these state space components.

$f = f_{11} \odot (x = <>) \odot (y = <>) \odot f_{31} \odot (f_{12} \odot hiext_{(x,e)} \odot lov_x \odot f_2 \odot$
$\odot hiext_{(y,t)} \odot lorem_x \odot lov_y \odot f'_{32} \odot lorem_y)^n \odot f_{33}$

Considering the following properties of the functions defined on sequences: $lov(hiext(<>, e)) = e$, and $lorem(hiext(<>, e)) = <>$, we get, that the x and y components, which are set to empty at the beginning, are also empty after each iteration-step, this means

$$f = f_{11} \odot f_{31} \odot (f_{12} \circ f_2 \odot f'_{32})^n \odot f_{33} \Rightarrow f = f'.$$

Thus the theorem is proved.

Remark. It often occurs, that one of the functions f_1, f_2, f_3 is the identity.

5. Conclusions

As a summary, we can say, that Jackson's programming method can be well formalized in the frame of our mathematical model. In the case of structure clashes, namely if the problem described with the correspondance between the structures of the input and output data cannot be handled with the help of an elementwise processable function, the solution is, with the aid of the composition of elementwise producer, elementwise processable and elementwise consumer programs, often possible.

This process can also be viewed as an appropriate transformation of the state space. Suppose, that the original problem had to be solved on the state space $A = \mathbf{U} \times \mathbf{V}$, but actually the elementwise processing can be interpreted on the state space $A' = \mathbf{X} \times \mathbf{Y}$. The essential property of an elementwise processable function is, that for evaluation of one element of the result, it uses only one element of the original iteration, this means, if $\mathbf{X} = seq(\mathbf{E})$, $\mathbf{Y} = seq(\mathbf{T})$, it can be viewed as a function $\mathbf{E} \to \mathbf{T}$. In this case, if we are able to transform the original $\mathbf{U}$ elementwise to $\mathbf{X}$ (with the help of a function $\mathbf{U} \to \mathbf{E}$), and then, after the elementwise processing, to transform its result $\mathbf{Y}$ elementwise to $\mathbf{V}$ (with the help of a function $\mathbf{V} \times \mathbf{T} \to \mathbf{V}$), we can easily solve the original, seemingly difficult problem.

We would like to emphasize, that the role of the "kernel" of the elementwise producer program, of the $\mathbf{S}'_{12}$, is a kind of abstract "read": the processable elements are produced with the help of the original state space component $\mathbf{U}$ one after the other. Similarly transforms $\mathbf{S}_3$ the sequence $\mathbf{Y}$ to the resulting $\mathbf{V}$.

The formal description of the state space transformations oversteps the intentions of the present paper, which introduced the way how Jackson's method can be adapted to our mathematical model.

The introduced concepts of elementwise producing and elementwise consuming might be generalized using appropriate state space transformations to the general case of the elementwise processable functions, and similarly the sequence of solving programs can be "inverted". This is the topic of another paper.

References

[1] **Dijkstra E.W.**, *A Discipline of Programming*, Prentice-Hall Inc., Englewood Cliffs, New York, 1976.

[2] **Fóthi Á.**, *Bevezetés a programozáshoz (Introduction into Programming)*, ELTE TTK, Budapest, 1983.

[3] **Fóthi Á.**, A Mathematical Approach to Programming, *Annales Univ. Sci. Bud. Sect. Comp.*, **9** (1988), 105-114.

[4] **Gries D.**, *The Science of Programming*, Springer, Berlin, 1981.

[5] **Hoare C.A.**, Proof of correctness of data representations, *Acta Informatica*, **1** (1972), 271-281.

[6] **Jackson M.A.**, *Principles of Programming Design*, Academic Press, New York, 1975.

[7] **Jackson M.A.**, Structure-oriented programming, *Program Transformation and Programming Environment*, ed. P.Pepper, Springer, Berlin, 1984, 169-180.

[8] **Vivarès F.**, Modelling Jackson's programming method, *Science of Computer Programming*, **20** (1993), 173-204.

[9] **Workgroup on Relational Models of Programming (Fóthi Á. et. al.)**, Some Concepts of a Relational Model of Programming, *The Fourth Symposium on Programming Languages and Software Tools, Visegrád, Hungary, June 8-14, 1995*, 434-446.

Á. Fóthi and J. Nyéky-Gaizler
Department of General Computer Science
Eötvös Loránd University
VIII. Múzeum krt. 6-8.
H-1088 Budapest, Hungary
fa@lngsc2@elte.hu, nyeky@ludens.elte.hu