Jiří Kopřiva

Generalization of well-translation of formal languages

Kybernetika, Vol. 2 (1966), No. 4, (305)--313

Persistent URL: http://dml.cz/dmlcz/125796

KYBERNETIKA ČÍSLO 4, ROČNÍK 2/1966

Generalization of Well-Translation of Formal Languages*

JIŘÍ KOPŘIVA

An algorithm for translation of formal languages is described which combines two different ways of translation. Its purpose is obtaining the target (translated) text having the same semantic content as had the original text, on the one hand (this is ensured by the use of the well-translation algorithm as our algorithm framework - "well-translation on the whole"), and allowing to mark the same semantic content of certain characteristic parts of the input and output text by writing them in the familiar form, on the other hand ("well- translation on a small scale").

1. INTRODUCTION

The concept of *well-translation of grammars and formal languages* (generated by grammars the rules of which can be represented in the Backus' normal form) was introduced in [1] and [2]. Denotation and concepts from these two papers will be used without references here. In section 2, the reader's attention is drawn to a small oversight in [2] and it is shown how it can be removed. In the following section, a little disagreeable consequence of the definition of well-translation is discussed. It is tried to eliminate this consequence by generalization of this concept. In the end, the generalized well-translation algoritm is described.

2.

The condition (2) in [2] (p. 48) contains implicitly (for both languages) the request of knowing the meaning of $S(x)$ for each $x \in G(a)$, where a is any standard symbol of an arbitrary syntactic rule. But, at the same time, a may be also a proper symbol (in V_P), hence a terminal symbol. The meaning of $G(a)$ is not defined in this case. If we put then $G(a) = \{a\}$, which is entirely natural, the meaning of $S(a)$ may still remain undefined. Each of the terminal symbols need not be the value of some metavariable, i.e. it need not belong to the set E of expressions. According to the definition of the function S, no semantic content is given to it.

* Presented at the *Second Conference on Cybernetics*, Prague, November 16th—19th, 1965.

This small oversight may easily be put right by the integration of the definition of $G(x)$ in the way mentioned above and by the integration (also quite natural) of the definition of $S(x)$ with the help of the mapping τ. If we do not wish to integrate these definitions, it is sufficient to reformulate unsubstantially the condition (2) in [2] (and in [1]) as follows:

(2′) *Let* $a_0 :: = b_0 a_1 b_1 \ldots a_k b_k$ *and* $c_0 :: = d_0 c_1 d_1 \ldots c_k d_k$ *be the standard form of the rules* $\mathfrak{r} \in \mathfrak{R}$ *and* $\Phi(\mathfrak{r}) \in \mathfrak{R}'$, *resp. For* $1 \leqq v \leqq m$, *where* $0 \leqq m \leqq k$, *let* a_{j_v} *be all the metavariables from the right hand side of the rule* $\mathfrak{r}$ *and let* $c_{\pi(j_v)} = \tau(a_{j_v})$. *For* $1 \leqq v \leqq m$ *let* $x_v \in G(a_{j_v})$ *and* $y_v \in G'(c_{\pi(j_v)})$ *be such phrases that* $S(x_v) = S'(y_v)$. *Then the functions* S *and* S' *assign the same semantic content to any two phrases arising from the rihgt hand sides of the rules* $\mathfrak{r}$, $\Phi(\mathfrak{r})$, *resp. by the substitution of the phrases* x_v, y_v *resp., for* a_{j_v}, $c_{\pi(j_v)}$ *resp., where* $1 \leqq v \leqq m$.

The condition (1) in [2] must be completed by $c_0 = \tau(a_0)$. Also in [2], in the definition of the $G(x)$ (the third section on page 47), it ought to be only e instead of $e \in E$.

The proof of the fact that the corresponding algorithm (see [1], [2]) affords the well-translation also in this case may be performed easily by induction (e.g. for the rang of the phrase marker, if we define the rang as the length of the longest of its paths).

3.

Let us look into two grammars G and G' having the same sets $V_T = V_T'$ and $V_N = V_N'$ of the terminal symbols and of metavariables resp. and the following sets of rules. $\mathfrak{R}$ contains the rules

$$\langle \text{digit} \rangle \quad :: = 0 \mid 1$$
$$\langle \text{number} \rangle :: = \langle \text{digit} \rangle \mid \langle \text{number} \rangle \langle \text{digit} \rangle$$

and $\mathfrak{R}'$ contains the rules

$$\langle \text{digit} \rangle \quad :: = 0 \mid 1$$
$$\langle \text{number} \rangle :: = \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{number} \rangle .$$

Both grammars generate the same set E (of unsigned dyadic integers). It is natural to define the mapping τ as the identical mapping of the sets V_N and V_T into themselves. Then the mapping Φ coordinates inevitably the rules from $\mathfrak{R}$ to the rules from $\mathfrak{R}'$ in the order as they are written above. This coordination forces the permutation $\pi_4(1) = 2$, $\pi_4(2) = 1$ for the fourth rule. According to the condition (2) in [2] (and [1]) eventually (2′) above it would be e.g. $S(01) = S(10)$. Generally, the same semantic content would be given to two dyadic integers such that each of them is the reflection of the other one (cf. [1], example I). That does not agree with expected

work of the function S. It should yield the same semantic content for the strings expressing the numbers having the same numerical value.

Now, let us look into the following sets of syntactic definitions. The first of them contains two rules, viz

$$\langle \text{type list} \rangle :: = \langle \text{simple variable} \rangle \mid \langle \text{simple variable} \rangle , \quad \langle \text{type list} \rangle$$

(an ALGOL 60 syntax fragment, cf. [3], Section 5.1.1 of the Chapter 5.1. Type Declaration); the second one contains also two rules, viz

$$\langle \text{type list} \rangle :: = \langle \text{simple variable} \rangle \mid \langle \text{type list} \rangle , \quad \langle \text{simple variable} \rangle .$$

If we consider the strings generated by the two grammar fragments in the whole language ALGOL 60 context (together with its semantics), then the condition (2′) having been brought to advantage does not cause any awkward consequences in spite of the fact that the resulting situation is formally analogous to that in the case of the grammars G and G' above.

It is possible to find more such examples. E.g. the ALGOL 60 syntax fragment generating *identifiers* ([3], Section 2.4.1.) and the same set of rules but with the corresponding change on the right hand side of the rule ⟨identifier⟩ :: = ⟨identifier⟩ ⟨letter⟩ corresponds to the first case. Under certain conditions (cf. [3], Section 4.2.3.1.) the ALGOL 60 syntax fragment generating the left part of the *assignment statement* ([3], Section 4.2.1) corresponds to the second case.

The following section will be opened with another important example of well-translation.

4.

The *push-down memory* is often used for compilation of various program parts expressed with the help of some formal programming language. For this purpose, the translated expression is temporarily represented in *reverse Polish notation*. A simple example will show that the translation from mathematical notation (of certain sort of expressions) into reverse Polish notation is the well-translation. (See also example I in [1].)

Let us consider the following sets of rules. The first one contains the rules

(1) $\langle \text{variable} \rangle :: = a_1 \mid a_2 \mid \ldots \mid a_n$

(2) $\langle \text{unary operator} \rangle :: = \Delta_1 \mid \Delta_2 \mid \ldots \mid \Delta_k$

(3) $\langle \text{binary operator} \rangle :: = \Theta_1 \mid \Theta_2 \mid \ldots \mid \Theta_m$

$\langle \text{expression} \rangle :: = \langle \text{variable} \rangle \mid \langle \text{unary operator} \rangle \langle \text{expression} \rangle \mid$
$(\langle \text{expression} \rangle \langle \text{binary operator} \rangle \langle \text{expression} \rangle)$

The second one contains the rules (1), (2), (3), and the rules

$$\langle \text{expression} \rangle :: = \langle \text{variable} \rangle \mid \langle \text{expression} \rangle \langle \text{unary operator} \rangle \mid$$
$$\langle \text{expression} \rangle \langle \text{expression} \rangle \langle \text{binary operator} \rangle$$

(cf. [4]). Now, it is sufficient to define τ as the identical mapping of the set $V_N \cup V_P$ into itself (the brackets (and) are auxiliary symbols) and Φ in such a way that it coordinates the rules of the two considered grammars in the same order as they are written above. The permutations π_1 and π_2 for the last but one and the last rule resp. are

$$\pi_1(1) = 2\,, \quad \pi_1(2) = 1\,;$$
$$\pi_2(1) = 2\,, \quad \pi_2(2) = 3\,, \quad \pi_2(3) = 1\,.$$

The numerical constants and identifiers stand instead of the variables a_i in the programs. Let us imagine that the rules generating numbers of digits (as in section 3) and identifiers of letters and digits (as in ALGOL 60) are substituted for the rules (1). We can then observe that the well-translation yields a convenient form of the resulting expression on the whole. But the small formal difference of the corresponding syntactic rules allowing the use of the well-translation algorithm may cause a disagreeable notation of some important substrings (e.g. of numbers and identifiers). (The suitable notation question has a deeper sense here, of course, and depends intrinsically on the compiler and perhaps on the performance of the operations in the computer.) At the same time, the relevant changes of the rules are sometimes forced by the used compilation manner (see e.g. [5]). The decision of whether we have an analogy to the grammars G and G' case (section 3) or an analogy to the case following that one cannot be done within the syntax. It is necessary to consider the semantic aspect of languages, their semantics.

It is often possible to find such disjoint parts of the input text (i.e. of the program written in some programming language) that the following conditions hold:

a) *A certain subphrase of the output text corresponds in one to-one-way to each such subphrase of the input text.*

b) *These corresponding subphrases of the output text are disjoint in it.*

c) *Each two corresponding subphrases have the same semantic content.*

The mutual relation between two corresponding subphrases need not be the well-translatability (with regard to the grammars of both languages). Examples of such input program parts are numbers, identifiers etc. in ALGOL 60. The analysis of these parts may often be performed by a simpler technique than that chosen for performing the whole program analysis (or its major parts). The idea of the *Turing machine* or the *transition table* may be used here (see e.g. [7], [6]).

It seems to be natural to use here two different methods of translation. For the first one we find out the relevant subphrases of the input text, translate them into the

output language with the help of some suitable translation algorithm. The output text parts prepared in this way are then stored for later use. (In some cases, these found subphrases of the input text may be placed at the relevant places of the output text without any changes.) The relevant metavariables are substituted for the found subphrases in the input text which is now prepared for the use of the second translation method, viz the well-translation process.

The well-translation algorithm constructs only the framework of the output text. The definitive form of the output text is then obtained by placing the prepared subphrases on the relevant places of the framework.

We formalize the described process in the following section.

5.

We suppose that the subphrases which are not to be translated with the help of the well-translation algorithm are values of certain metavariables from V_N. We call them auxiliary metavariables.

The translation process may be decomposed into the following nine phases. The more detailed description of the processes performed in the separated phases is given further. The denotation used in the following item II differs a little from that used in [1] and [2].

I. *Syntactic analysis* of the input text; its result is the sequence $\{(p_i, q_i)\}_m^1$ of number pairs.

II. *Construction of the phrase marker*, i.e. construction of the sets $P = \{(i, \mathfrak{r}_i) \mid 1 \leqq i \leqq m\}$ and $Q = \{(a_i, b_i, c_i) \mid i = 1, \ldots, m-1\}$.

III. *Quest of such elements* $(i, \mathfrak{r}_i) \in P$ *that the left hand side of* $\mathfrak{r}_i$ *is an auxiliary metavariable.* Among all such vertices being placed on the same path we take that one whose distance from the root is minimal.

IV. The *subtrees* having their roots in the vertices found in III are the *phrase markers of the subphrases mentioned above.* We find these subphrases. Simultaneously, we omit from P and from Q the elements corresponding to the vertices of these subtrees.

V. *Translation* of the found input text subphrases.

VI. *Construction of the new rules* having the auxiliary variables images on their left hand sides and the relevant subphrases translations on their right hand sides.

VII. *Translation* of the rests P_1 and Q_1 of the sets P and Q with the help of the well-translation algorithm; we get P_1' and Q_1'.

VIII. *Adding to* P_1' *new pairs* containing the new rules.

IX. *Synthesis* of the output text.

Some explanation to the individual items:

I. The syntactic analysis may be performed e.g. with the help of the algorithm described in [1]. p_i is the index of the rule used at the $(m - i + 1)$-st derivation step

consisting in replacing the q_i-th symbol of the string obtained as yet by the right hand side of the rule $\mathfrak{r}_{p_i}$. The rule $\mathfrak{r}_{p_m}$ is used at the first step, then is used $\mathfrak{r}_{p_{m-1}}$ etc.

II. The phrase marker vertices are denoted by positive integers. The labelling of the vertex i by the rule $\mathfrak{r}_i$ is performed by putting the pair $(i, \mathfrak{r}_i)$ into the set P. As to the meaning of the triple (a_i, b_i, c_i) from Q, (a_i, b_i) is the *phrase marker branch* and the right hand side of the rule labelling the vertex b_i is substituted for the c_i-th standard symbol of the right hand side of the rule labelling the vertex a_i. The algorithm from [1] constructing the sets P and Q, given the sequence $\{(p_i, q_i)\}$, must here be changed a little, because it does not give c_i as the distance of a standard symbol from the left border but the distance of a symbol in general.

III. We begin with $i = m - 1$ (we suppose that the input text is not the value of an auxiliary variable) and decrease i sequentially. If we find the requested element $(i, \mathfrak{r}_i) \in P$, then we do not investigate the vertices of the subtree having its root in the vertex i. In this way, we ensure that the found pair $(i, \mathfrak{r}_i)$ is one whose distance from the root is minimal. The omission of the elements of P and Q mentioned in IV above may be performed also in connection with the just described process (see also IV below).

IV. The subphrases obtained by the synthesis on the basis of the phrase markers being the subtrees of the main phrase marker and having their roots in the vertices found in III are obviously the values of the auxiliary variables. We omit from P all pairs $(i, \mathfrak{r}_i)$, where i's are the vertices of the just mentioned subtrees including their roots (found in III). From the set Q, we omit the three-tuples (a_i, b_i, c_i), where a_i's are the vertices of the mentioned subtrees (including their roots). After this process, the main phrase marker will contain the branches ending in the roots of the subtrees. We can say (with a certain inaccuracy) that we have formed an *incomplete phrase marker*. The synthesis made on the basis of it does not yield a string of terminal symbols, but a string containing the (auxiliary) metavariables, too.

V. The translation of the subphrases found in IV may be performed with the help of any algorithm translating a context free language into another one. (See also the end of the last but one paragraph of section 4.)

VI. The left hand side of each new rule will be the image of the auxiliary variable which is the left hand side of some rule found in III. The translation of the subphrase found in IV and being the value of this auxiliary variable forms the right hand side of the new rule.

VII. The translation of the sets P_1 and Q_1 means what follows. At first, the rules $\mathfrak{r}_i$ in the pairs $(i, \mathfrak{r}_i)$ are replaced by the rules $\Phi(\mathfrak{r}_i)$. Then the numbers c_i in triples (a_i, b_i, c_i) are replaced by the numbers $\pi_{a_i}(c_i)$. Of course, we must assume the fragment G_1 and G_1' of the grammars G and G' resp. of the input and output language resp. requisite to the translation of the incomplete phrase marker, are in the well-translatability relation. This condition need not hold for the parts of G and G' requisite to the translation of the subphrases from IV.

Let us show a method of finding out in G the rules not used during the application

of the well- translation algorithm. The metavariable β is said to be subordinated to the metavariable α if the following condition holds (see [8] or [9]):

$$(\neg\, \alpha \sim \beta) \wedge (\alpha \in \sum\nolimits_r \supset \beta \in \sum\nolimits_r^{\sim}).$$

Let the metavariables α, β, γ satisfy the following condition. *Let β be an auxiliary one and let β be subordinated to α and γ subordinated to β.* (Then obviously γ is also subordinated to α.) *Let $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n = \gamma$ be any sequence of metavariables such that $\alpha_i \blacktriangleright \alpha_{i+1}$ for $i = 1, \ldots, n-1$* (see [9]). *Then $\alpha_i = \beta$ for some i, $2 \leqq \leqq i < n$.*

Let the input text be the value of the metavariable α. Then it is obvious that *we shall not use the rules containing the metavariables of the type γ (on their left hand or right hand sides). The same affirmation holds for the rules containing the auxiliary variables on their left hand sides.*

The remaining rules (in G_1) must have their images in G_1' in the sense of the well-translation definition.

VIII. The vertices which are the roots of the subtrees described in IV and the ends of certain branches from Q_1' have no corresponding pairs in P_1' as yet. This deficiency will be removed by completing P_1' by the new pairs. The first member of each such pair is the vertex which are the root of a subtree mentioned in IV. The second member is the corresponding new rule found in VI.

IX. The output text synthesis will not be described in detail here. Let us note only that under the given circumstances it is advantageous *to replace the last metavariable of the string found up to now.* In this way, the number of the metavariables whose distance from the left border is to be computed at each step is minimal. The reduction which is inverse to such a synthesis is just the canonical reduction described in [10].

6.

The described generalization of the well-translation does not vitiate its main property, viz the *meaning conservation.* That may be proved with the help of induction in a quite similar way as for the original well-translation algorithm (see the last paragraph of section 2). Of course, we must suppose that *the corresponding characteristic subphrases of the input and output text, resp. have the same semantic content.*

A simple example showing the applicability of the described algorithm is that of translating the expressions from the familiar mathematical form into Polish notation. Instead of variables a_i (see section 4) we may consider the numbers to be translated from the decimal into dyadic system and the identifiers whose certain limited initial (or final) part is to be retained in the output text.

The described generalized well-translation algorithm will be tried out on the computer MINSK 22.

(Received December 6th, 1965.)

[1] Čulík K.: Well-translatable grammars and ALGOL-like languages. IFIP working conference "Formal language description languages", Vienna, 14–19 September 1964. To appear.
[2] Čulík K.: Semantics and translation of grammars and ALGOL-like languages. Kybernetika *1* (1965), 1, 47–49.
[3] Backus J. W. et al.: Revised report on the algorithmic language ALGOL 60. Numerische Mathematik *4* (1963), 420–453.
[4] Péter R.: Über die Rekursivität einiger Übersetzungs-transformationen. I. Mitteilung. Publ. Math. Inst. Hung. Acad. of Sciences, Vol. *VII* (1962), Series A, Fasc. 1–2, 69–78.
[5] Morris D.: The use of syntactic analysis in compilers. "Introduction to system programming". Academic Press, London and New York 1964, 249–255.
[6] Naur P.: The design of the Gier ALGOL compiler. BIT *3* (1963), No 2, 124–140; 3, 145 to 166.
[7] Naur P.: State analysis of linear texts. Preliminary report, Regnecentralen Copenhagen, June 1965.
[8] Čulík K.: Formal structure of ALGOL and simplification of its description. In "Symbolic languages in data processing" (Roma 1962). Gordon–Breach, New York 1963, 75–82.
[9] Kopřiva J.: A note on the structure of certain predicates concerning the sublanguages of ALGOL 60. Kybernetika *1* (1965), 2, 122–126.
[10] Eickel J., Paul M., Bauer L. and Samelson K.: A syntax controlled generator of formal language processors. Comm. ACM *6* (1963), 8, 451–455.

VÝTAH

Zobecnění dobrého překládání formálních jazyků

JIŘÍ KOPŘIVA

Některé způsoby kompilace formálních programovacích jazyků si vynucují změny syntaktických pravidel. I když někdy malé formální změny tohoto druhu neporuší vztah „dobré přeložitelnosti" ([1], [2]), a tedy možnost použití algoritmu pro dobré překládání, mají v některých případech nepříjemné důsledky na zápis jistých částí výstupního textu (příklady v odstavcích 3 a 4). Ve snaze vyhnout se těmto důsledkům dobrého překládání „v malém", je vytvořen a popsán algoritmus pro dobré překládání „vcelku", který však ponechává možnost užít jiného způsobu překladu pro jisté charakteristické podfráze vstupního textu. Tím je dána možnost vyznačit stejný význam jistých sobě si odpovídajících částí vstupního a výstupního textu tím, že je zapíšeme způsobem obvyklým v tom kterém jazyku. Při tom vztah mezi nimi nemusí být dobrá přeložitelnost vzhledem ke gramatikám vstupního a výstupního jazyka.

Jsou proto v průběhu překládacího procesu kombinovány dva způsoby překládání formálních jazyků. Libovolným způsobem jsou přeloženy jisté části vstupního textu.

Tyto části jsou pak v něm nahrazeny příslušnými metaproměnnými. Takto vzniklá „kostra" vstupního textu je pomocí algoritmu pro dobré překládání přeložena na „kostru" výstupního textu. Zasazením předem připravených překladů částí na příslušná místa této kostry je dokončeno vytvoření výstupního textu. Přitom není porušena základní vlastnost dobrého překládání, totiž zachování stejného významu překládaného textu.

Doc. Dr. Jiří Kopřiva, CSc., Laboratoř počítacích strojů, Třída Obránců míru 21, Brno.