A Translator between Assembly Languages

A Translator between Assembly Languages

Yoneharu Fujita* and Fujio Nishida*

(Received June 15, 1979)

This paper describes a semantic translator between assembly languages and also its experimental results. This translator translates a source programs into an object program by partitioning the source program into segments and constructing somewhat optimized object parts of them using a theorem-proving technique.

1. Introduction

The production of softwares needs many programmers and much time, what is more, the debugging and the maintainance of softwares also need much labour. Therefore, various automatic program-synthesis techniques have been much expected to be developed.[1][2] Unfortunately, however, they have not been developed yet enough to some extent of practical use. On the other hand, there are a large amount of accumulation of useful programs that have been produced for various computers, therefore, it is also useful to find a transformation-technique which automatically converts a completed program into a program for another computer.

In this paper, a semantic transformation method of assembly programs is presented and experimented. A translator presented here partitions a source program into segments by parsing, applies a theorem-proving technique to each segment and yields an object program which consists of somewhat optimized and relevant program pieces for an object computer.

2. Description Equations of Instructions

2.1 A description equation

The function of a sequence of instructions is represented by a change of a state of memory and registers associated with the instruction sequence. In order to represent the functions of instructions, a content function $C(l, s)$ is introduced. It represents a value of a storage cell designated by a specified label l (i.e. a register's name or an address) at the specified state s. The syntactic rules of a content function are described as follows:

⟨content function⟩ :: = C(⟨label⟩, ⟨state⟩)

⟨label⟩ :: = ⟨register name⟩ | ⟨register variable⟩ | ⟨address name⟩ |
⟨address variable⟩ | ⟨content function⟩ | ⟨value⟩

---

* Department of Electrical Engineering, College of Engineering.

⟨ register name ⟩ :: = declared names of registers
⟨ register variable ⟩ :: = $\{\mathrm{Ri} \mid \mathrm{i} = 1, 2, \cdots\}$
⟨ address name ⟩ :: = { names of addresses }
⟨ address variable ⟩ :: = { gothic small letters }
⟨ state ⟩ :: = ⟨ state variable ⟩ | ⟨ state ⟩ ; ⟨ instruction ⟩
⟨ state variable ⟩ :: = $\mathrm{s} \mid \mathrm{s}_1 \mid \mathrm{s}_2 \mid \cdots$
⟨ instruction ⟩ :: = instruction of an assembly language.

By the aid of a content function, a description equation is defined as follows:

⟨ description equation ⟩ :: = ⟨ content function ⟩ = ⟨ value ⟩
⟨ value ⟩ :: = ⟨ function ⟩ (⟨ sequence of arguments ⟩) | ⟨ numerical value ⟩
⟨ address ⟩ | ⟨ content function ⟩
⟨ function ⟩ :: = arithmetic or logical function's name
⟨ sequence of arguments ⟩ :: = ⟨ value ⟩ | ⟨ sequence of arguments ⟩, ⟨ value ⟩

As seen from the above, a description equation defines a value of a label variable at a certain state. If the state part consists of an instruction sequence the state part denotes a state resulted from the application of the instruction sequence to the initial state.

By using content functions, the equivalence of two sequence $S_1^{(1)}, \cdots, S_m^{(1)}$ and $S_1^{(2)}, \cdots, S_n^{(2)}$ of instructions is described as follows:

$$C(y, s; S_1^{(1)}; \cdots; S_m^{(1)}) = C(y, s; S_1^{(2)}; \cdots; S_n^{(2)}).$$

The above equivalence relation is the base of this translation method.

[Example 2.1] The description equation

$$C(R1, s; MV x, R1) = C(x, s)$$

represents that the value of the register $R1$ at the state after execution of the instruction "$MV x, R1$" is equal to the value of an address x at the state before the execution of the instruction.

[Example 2.2] The description equation

$$C(y, s') = C(y, s) + C(x, s)$$

represents that the value of y at a state s' is equal to the sum of values of y and x at a state s.

2.2 Some properties of description equations

Any instruction except for conditional ones is represented in terms of a set of description equations which describes state change of a part of memory and registers caused by the execution of the instruction. These description equations can be simplified in some cases.

First, if an instruction is not a branch instruction, the change of IC is not necessary to be represented for translation of assembly program, and can be represented by an description equation.

Second, if the left side of a description equation D_1 occurs in another description equation D_2 then D_1 and D_2 can be combined into one description equation D_3 by the substitution of the right side of D_1 for each occurence of the left side of D_1 in D_2. Finally, if S does not contain the label z as a destination operand, any term $C(z, s; S)$ having a label z, a state variable s and an instruction S can be simplified into an abbreviated form $C(z, s)$ because S has no effect on the label z.

[Example 2.3] Suppose

$$C(y_1, s; A\, x_1, y_1) = C(y_1, s) + C(x_1, s) \tag{1}$$

and

$$C(z, s; A\, x_1, y_1; S\, y_1, z) = C(z, s; A\, x_1, y_1) - C(y_1, s; A\, x_1, y_1) \tag{2}$$

hold, then substitution of the right side of eq. (1) for the term $C(y_1, s; A\, x_1, y_1)$ in eq. (2) yields,

$$C(z, s; A\, x_1, y_1; S\, y_1, z) = C(z, s; A\, x_1, y_1) - (C(y_1, s) + C(x_1, s)). \tag{3}$$

Furthermore, the term $C(z, s; A\, x_1, y_1)$ is simplified into $C(z, s)$ because the instruction "$A\, x_1, y_1$" has not the label z as a destination operand.

If the left side term of a description equation is contained in another description equation, then the order of the two instructions corresponding to these description equations cannot be changed. In this case, these two descriptions are said to be dependent on each other.

3. Segment

Source programs to be translated can be generally considered to be optimized globally, therefore this paper is not concerned with the optimization of program parts which causes serious changes of original program structure such as loops or branchs, but concerned with generation of an optimized linear portion of the object program. For this purpose, a linear part of source program is divided into some parts each of which is called a segment in order to generate an optimized linear portion of an object program in an efficient way. A segment is defined as a maximum linear portion of a program such that every adjacent description equation is dependent on each other. In other word, a segment is a linear portion that is computable by only one register and a partition to segments is independent of the kind of assembly languages.

[Example 3.1] Consider a linear part of a program which has the following description equations;

(1) $C(R1, s_1; MV x, R1) = C(x, s_1)$

(2) $C(R1, s_2; A\ y, R1) = C(R1, s_2) + C(y, s_2)$ where $s_2 = s_1; MV x, R1$

(3) $C(R1, s_3; S\ z, R1) = C(R1, s_3) + C(z, s_3)$ where $s_3 = s_2; A\ y, R1$

(4) $C(R2, s_4; MV u, R2) = C(R2, s_4)$ where $s_4 = s_3; S\ z, R1$

(5) $C(R2, s_5; A\ v, R2) = C(R2, s_5) - C(v, s_5)$ where $s_5 = s_4; MV u, R2.$

The equations (1), (2) and (3) are dependent on each other and also the equations (4) and (5) are so, but no dependency relation exists between the two groups (1), (2), (3) and (4), (5). Therefore the linear part is divided into two segments (1) – (2) – (3) and (4) – (5).

The segmentation procedure is described as follows:

(1) Generate a description equation of each instruction in a source program, then construct a flow graph of the source program and assign the description equation to each node of the flow graph.

(2) Divide the flow graph into linear pieces.

(3) For a linear piece, a serial nodes starting from the top node and depending on each other is a segment. Remove the obtained segment from the linear pieces and repeat (3) until the linear piece becomes empty.

(4) Repeat (3) for every linear piece.

A segment thus obtained consists of several nodes which are serially numbered and dependent on each other. The description equations at these nodes are combined into one description equation by substitutions as mentioned in section 2.

[Example 3.2] The description equations in Example 3.1 are combined into two description equations by substitutions and then simplified, as mentioned in section 2, as follows:

$$C(R1, s_1; MV x, R1; A\ y, R1; S\ z, R1) = C(x, s_1) + C(y, s_1) - C(z, s_1)$$
$$C(R2, s_4; MV u, R2; S\ v, R2) = C(u, s_4) - C(v, s_4).$$

Each description equation of a segment is used for generating a sequence of instructions written in an objective assembly language. For this purpose, the state part of the left side of the description equation is replaced by an undetermined state variable s, then an instruction sequence of the object program is found which satisfies the undetermined state variable in terms of the object assembly language.

The following properties of a segment are used to remove redundant inferences in a theorem-proving process at generation of an object program from each segment.

[Property 3.1] If the label part of a description equation D of a segment contains no function, the sequence of instructions which satisfies the undetermined state variable in the description equation D needs only at most two transfer instructions for loading a data on a register and for storing the content of the register into an address.

(Proof) The input-output relation of a segment can be realized by using only one register. Therefore, this segment requires only the initial transfer of data to a register and the final transfer of the data from the register to a memory address.

[Property 3.2] If the description equation D of a segment contains no content function in the label part of the left side and contains a nesting of the n-th degree of the content functions in the right side, the sequence of instructions satisfying the description equation D contains at most $2n+2$ transfer operations.

(Proof) One nesting of a content function is realized at most two transfer instructions, therefore the realization of a nesting of the n-th degree content functions needs at most $2n$ transfer instructions. Furthermore, the content function in the left side of the description equation D needs at most two transfer instructions. Therefore the total number of transfer instructions is at most $2n+2$.

[Corrollary 3.1] Suppose D is a description equation which has one content function in the left side and also a nesting of the n-th degree of content functions and m operations in its right side. Then the realization of a sequence of instructions satisfying the undetermined state part of D needs at most $2n+m+2$ instructions.

(Proof) The corrollary is evident from Property 3.1 and Property 3.2.

4. Transformation of Segments

In order to obtain an instruction sequence written in an object language equivalent to an instruction sequence contained in a source program, a theorem-proving technique is applied to a set of description equations which are written in the object language and involve some undetermined state variables. Let D be a description equation containing an undetermined state variable s in the left side and $A(s)$ be an answer clause, and construct a clause $\sim D \vee A(s)$ called an objective clause. Then the state part of the derived literal of the answer clause A from the objective clause contains the object sequence of instructions.

In derivation of the answer clause A, the form of unification applied to a pair of terms in description equations is somewhat different from the usual one at the respects that state part of the content function has a kind of a right operation form and that there are several kinds of variables such as registers and addresses of memory. Therefore, the unification applied here, first, rewrites the state part of $C(l, s; S_1; \cdots; S_n)$ temporalily into the corresponding left operator form $C(l, S_n(S_{n-1}(\cdots(S_1(s))\cdots)))$ then takes the operands of each $S_i\,(i = 1, \cdots, n)$ as variables or constant terms and then performs the usual unification under the condition of inhibition of the substitution between the different kind variables.

From the above, the unification procedures are summarized as follows:

(1) Transform the state parts of given description equations into left operator forms.

(2) Apply the usual unification to the description equations. If the substitution is permitted then go to (3), otherwise search the other substitution and repeat (2). If no permitted substitution exists the description equations can not be unified.

(3) Apply the inverse transformation of (1) to the unified description equations and output the result.

The derivation of the answer clause A is based on the set of support strategy which has an objective clause $\sim D \vee A$ as the support set. Since all clauses except the support set are unit clauses, the above strategy reduces to a unit resolution or an input resolution. Furthermore, the theorem proving routine obtains an optimized derivation of the answer clause A by rejecting derivations which exceed the bound mentioned in corrollary 3.1 and by searching a shorter derivation.

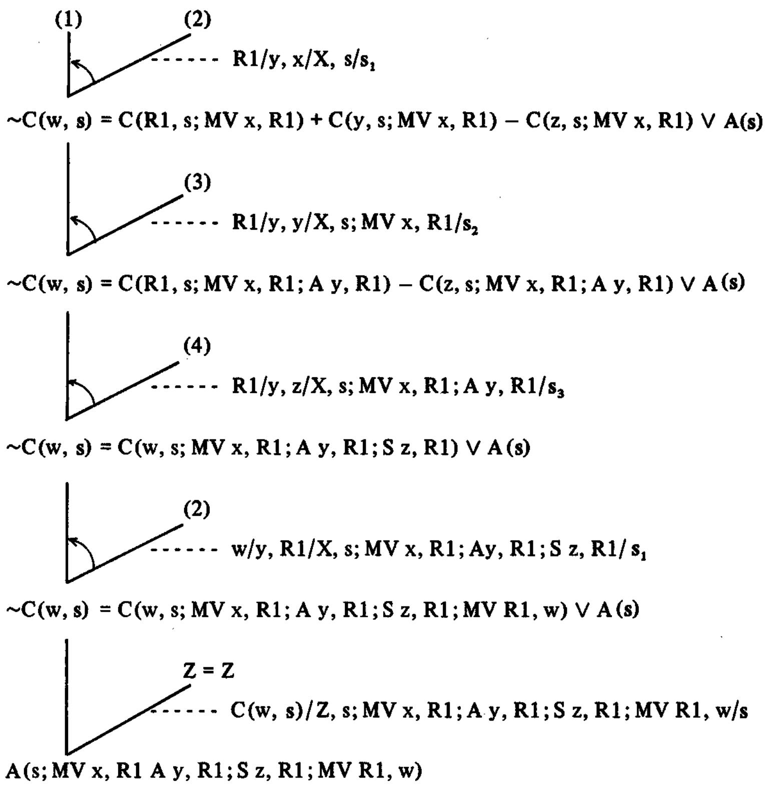
[Example 4.1] $C(w, s) = C(x, s) + C(y, s) - C(z, s) \cdots (1)$ is a clause describing a function of an objective program part. Suppose that the associated instructions of the object assembly language are defined by the following description equations respectively:

$$C(y, s_1;\, MV x, y) = C(x, s_1), \tag{2}$$
$$C(y, s_2;\, A\, x, y) = C(y, s_2) + C(x, s_2) \tag{3}$$

and

$$C(y, s_3;\, S\, x, y) = C(y, s_3) - C(x, s_3). \tag{4}$$

By the aid of the reflexive axiom $z = z$, the answer clause A is derived linearly as shown in Fig. 1.



(1) (2) ------ R1/y, x/X, s/s$_1$

~C(w, s) = C(R1, s; MV x, R1) + C(y, s; MV x, R1) – C(z, s; MV x, R1) V A(s)

(3) ------ R1/y, y/X, s; MV x, R1/s$_2$

~C(w, s) = C(R1, s; MV x, R1; A y, R1) – C(z, s; MV x, R1; A y, R1) V A(s)

(4) ------ R1/y, z/X, s; MV x, R1; A y, R1/s$_3$

~C(w, s) = C(w, s; MV x, R1; A y, R1; S z, R1) V A(s)

(2) ------ w/y, R1/X, s; MV x, R1; Ay, R1; S z, R1/ s$_1$

~C(w, s) = C(w, s; MV x, R1; A y, R1; S z, R1; MV R1, w) V A(s)

Z = Z ------ C(w, s)/Z, s; MV x, R1; A y, R1; S z, R1; MV R1, w/s

A(s; MV x, R1 A y, R1; S z, R1; MV R1, w)

Fig. 1. A derivation of an answer clause.

5. Implementation and Experimental Results

Based on the translation principles described in the preceding sections, a translation system of an assembly program was implemented and a translation experiment was performed on a mini-computer with a magnetic disk.

The translation system transforms each instruction in a given source program into some description equations using a dictionary which involves description equations of each instruction in the source language. In parsing of a source program, the conditional branch instructions are treated separately from the ordinary instructions and they are represented by a form "***if*** ⟨*branch condition*⟩ ***then*** ⟨*description equation*⟩ ***else*** ⟨*description equation*⟩". Based on the description equations and the above expression of branch instructions, the translation system generates a flow graph which has the line numbers attached to the description equations by the procedure 3.1 at the nodes. Then the translation system divides the flow graph into several segments by the dependency relations of description equations and obtains for each segment a description equation which contains an undetermined state variable in its content function. Next, the translation system generates the object sequences of instructions from the description equations using the theorem-proving technique with the aid of the dictionary for the objective assembly language.

The experimental system up to the present has some constraints in application for its conciseness. First both a source computer and an object computer are assumed byte-machines. Second, the name of an address is assumed to be unique. For example, the address named for *DATA2* in a sequence of instructions such as

'· · · ; *DATA1 DS 2F; DATA2 DS 2F*; · · ·'

can be also represented as *DATA1+4*, but only *DATA2* is adopted as the name. Finally, the optimization of the assignment of registers is not taken into consideration. If an object computer has fewer registers than a source computer, a register of the object computer is used for several registers of the source computer in multiple ways.

Fig. 2 is a schematic diagram of the whole translation process described in section 4, where solid lines and dotted lines denote flows of control and data respectively. The translation system consists of about 26 KB in an assembly language. The translation time excluding transfer time between the main memory and a disk was about *0.2* seconds for every instruction in a segment. Fig. 3 and Fig. 4 show examples of a source program and the translated object program together with some related description equations involved in dictionaries for the translation.

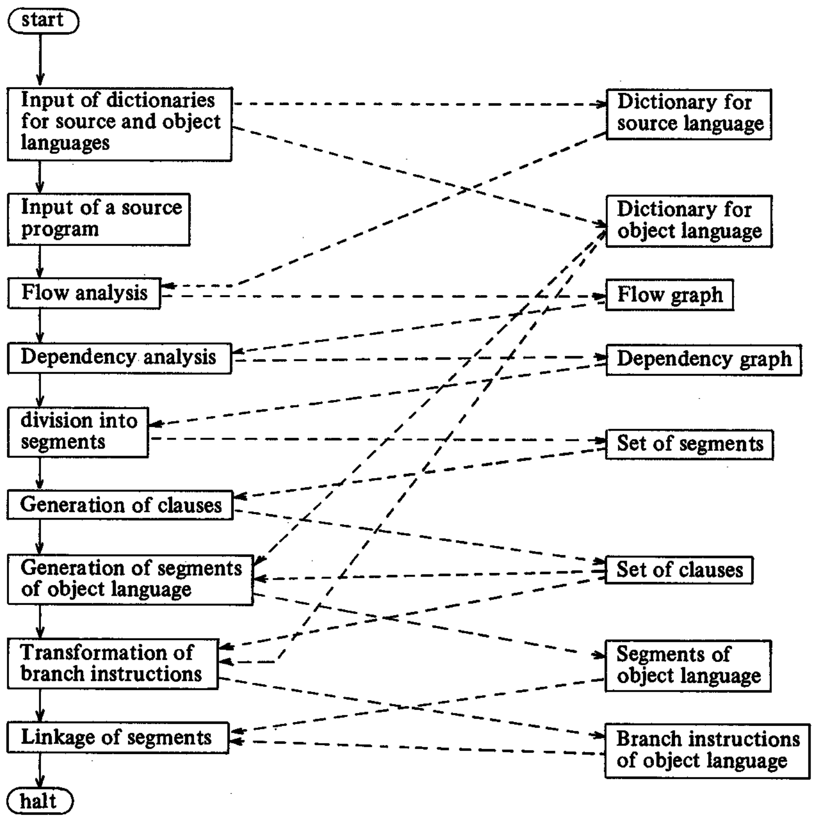

start
Input of dictionaries for source and object languages
Input of a source program
Flow analysis
Dependency analysis
division into segments
Generation of clauses
Generation of segments of object language
Transformation of branch instructions
Linkage of segments
halt
Dictionary for source language
Dictionary for object language
Flow graph
Dependency graph
Set of segments
Set of clauses
Segments of object language
Branch instructions of object language

Fig. 2. Flow chart of the whole process.

```
        LDI   R3, 100         C(R00, s00; LD R00, x00) = C(x00, s00)
        LDI   R1, 0           C(R01, s01; LDI R01, x01) = x01
        LDI   R2, 1           C(R02, s02; LD R02, x02, R03)
LOOP    AD    R1, DATA, R3        = C(x02 + C(R03, s02), s02)
        SB    R1, BBB         C(x03, s03; ST R04, x03) = C(R04, s03)
        AD    R1, XYZ         C(x05 + C(R06, s05), s05; ST R07, x05, R06)
        SBR   R3, R2, SNZ         = C(R07, s05)
        BRN   OWARI           C(R08, s06; AD R08, x06)
        BRN   LOOP                = C(R08, s06) + C(x06, s06)
OWARI   ST    R1, KEKKA       C(R09, s06; AD R09, $x07)
        ;END                      = C(R09, s06) + C(C(x07, s06), s06)
                              C(R10, s07; AD R10, x08, R11)
                                  = C(R10, s08; SB R12, x09)
                              C(R12, s08; SB R12, x09)
                                  = C(R12, s08) – C(x09, s08)
```

```
C(R13, s09; SB R13, $x10)
     = C(R13, s09) - C(C(x10, s09), s09)
C(R14, s10; SB R14, x11, R15)
     = C(R14, s10) - C(x11 + C(R15, s10), s10)
C(R00, s00; ADR R00, R01, SRN)
     = C(R00, s00) + C(R01, s00)
C(R00, s00; SBR R00, R01, SNZ)
     = C(R00, s00) - C(R01, s00)
C(IC, s00; BRN x00) = x00
```

Fig. 3. An example of a source program and related description equations of a source language.

```
        MV    = 100, R3
        MV    = 0, R1
        MV    = 1, R2
LOOP    A     DATA(R3), R1
        S     BBB, R1
        A     XYZ, R1
        S     R2, R3
        BZ    OWARI
        B     LOOP
OWARI   MW    R1, KEKKA
        END
```

```
C(x01, s00; MV x00, x01) = C(x00, s00)
C(C(R00, s02), s02; MV x02, (R00))
     = C(x02, s02)
C(R01, s03; MV = x03; R01) = x03
C(x04, s04; MV (R02), x04) = C(C(R02, s04), s04)
C(R03, s05; MV x05, R03) = C(x05, s03)
C(x07, s06; A x06, x07) = C(x07, s06) + C(x06, s06)
C(x09, s07; A x08(R04), x09)
     = C(x09, s07) + C(x08 + C(R04, s07), s07)
C(R06, s11; A x14, R06) = C(R06, s11) + C(x14, s11)
C(X11, s08; S x10, x11) = C(x11, s08) - C(x10, s08)
C(R04, s09; S x12, R04) = C(R04, s09) - C(x12, s09)
C(x13, s10; S(R05), x13)
     = C(x13, s10) - C(C(R05, s10), s10)
?ZP C(IC, s00; BNM x00) = x00
?ZZ C(IC, s00; BZ x00) = x00
C(IC, s00; B x00) = x00
```

Fig. 4. A translation result and related description equations of an object language.

6. Conclusion

The experimental result shows that the aim of reconstructing an optimized linear pieces of an object program by the aid of a theorem proving technique is achieved under a few restrictions. The restriction on uniqueness of each address name is removable in the practical system because it is not essential. The restriction on the address unit is theoretically removable but removal of it is considered to cause much inefficiency.

The other important problem in practice is the optimization of register assignment.

It needs a global optimization based on a flow chart of an object program and is left in future study.

Reference

1) Z. Manna and R. Waldinger, Comm. ACM, **14**, 151 (1971).
2) Y. Fujita and F. Nishida, Trans. IECE of Japan, **61-D**, 103 (1978).