Formulas with indications for establishing definitional equivalence

Technische Hogeschool Eindhoven.
Onderafdeling der Wiskunde.
Notitie 15, d.d. 21 december 1970.

AUT8

Formulas with indications for establishing definitional equivalence.

by N.G. de Bruijn.

1. Introduction.

We consider a language in which a notion of definitional equivalence is defined by means of a number of rules. A pair of expressions in the language can be called directly definitionally equivalent if the equivalence is established by a single application of a single rule. In the general case we need a chain of intermediate expressions which are pairwise directly equivalent. An essential feature of the definitional equivalence is, that if a subexpression Π_1 of an expression Σ_1 is definitionally equivalent to Π_2, and if the expression Σ_2 is obtained from Σ_1 if we replace Π_1 by Π_2, then Σ_1 and Σ_2 are definitionally equivalent. We shall call this the subexpression rule; all further rules will be called local rules.

We shall be concerned with the development of a notational system for indicating by what sequence of applications of local rules to what subexpressions an expression can be transformed into another one.

Before we start to state things formally, we shall try to express our intention by the following example. The expressions to be considered are of the following kind

$$p\,(\,a\,(\,f\,(\,d\,,\,b\,(\,x\,,\,u\,)\,,\,c\,)\,,\,b\,(\,g\,,\,w\,(\,t\,)\,)\,)\,) \qquad (1.1)$$

Now assume that $b(x,u)$ can be directly transformed into $r(u,x)$ by means of local rule (1). We write $b(x,u) \overset{(1)}{=} r(u,x)$. Assume, furthermore, that $r(u,x) \overset{(2)}{=} e(x)$, that $f(d,e(x),c) \overset{(3)}{=} q$, $b(g,w(t)) \overset{(4)}{=} z$, and $a(q,z) \overset{(5)}{=} v(y,z)$. Then the expression (1.1) is definitionally equivalent to $p(v(y,z))$. We shall now write all this in a single formula, viz.

$$p(a(f(d,\ b(x,u) \overset{(1)}{=} r(u,x) \overset{(2)}{=} e(x),\ c) \overset{(3)}{=} q,\ b(g,\ w(t)) \overset{(4)}{=} z) \overset{(5)}{=} v(y,z)) \qquad (1.2)$$

From this line several formulas can be obtained, if we select a single entry from each chain (the word "chain" refers to structures like $\ldots \overset{(7)}{=} \ldots \overset{(8)}{=} \ldots$) and drop like all the others. If we take the leftmost member of each chain, we get formula (1.1), if we take the rightmost member of each chain, we get $p(v(y,z))$. Thus our formula (1.2) indicates by what sequence of rules (1.1) is transformed into $p(v(y,z))$.

We shall say that (1.1) is a formula of the object language, and that (1.2) is a formula of the reference language.

The object language, to be described in section 2, will be slightly different from the one used in (1.1). In (1.1) we have "terminal letters" (not followed by an opening parenthesis) like d,x,u,c,g,t. We shall now replace these by k(∅), x(∅), u(∅),..., where ∅ is the empty string. In order to cater for the needs of AUTOMATH (see [1]) we admit the possibility of having still more "fake strings". A fake string is a symbol that can occur after an opening parenthesis and nowhere else. It is not, and does not stand for, a non-empty sequence of subexpressions separated by commas. A further alteration is that we do not want to have a single expression between parentheses. If it occurs, we put a symbol ∅ and a comma in front of it. Example: instead of f(g(x)) we write f(∅,g(∅,x(∅))).

These empty strings complicate our formulas, of course, but they keep us closer to the coding system (see [1]), and they simplify our discussion: Now we are no longer troubled by the difference between an expression and the string consisting of just that expression, nor by the difference between an identifier and an expression consisting of just that identifier.

In order to keep parentheses and commas free for ordinary use, we shall put little roofs over the separation marks of the language (e.g. instead of f(∅,g(∅,x(∅))) we write f ‸ ∅ ‸ g ‸ ∅ ‸ x ‸ ∅ ‸ ‸ ‸.

2. The object language.

We have three sets of symbols: "identifiers", "fake string symbols" and "separation marks". There are only three separation marks: the comma, the opening parenthesis and the closing parenthesis. We now form formulas, described in "Backus' normal form" as follows:

<formula> ::= <string> | <expression>
<string> ::= <fakestringsymbol> | <string> ‸, <expression>
<expression> ::= <identifier> ‸(<string> ‸)

Note that a formula cannot be both a string and an expression.

3. The reference language.

In addition to the symbols of the object language, we have a set of local rule numbers. We now construct <refformula>, <refstring>, <refexpression> (abbreviations for "formula in the reference language", etc.) as follows:

<refformula> ::= <refstring> | <refexpression>
<refstring> ::= <fakestringsymbol> | <refstring> ‸, <refexpression>
<refexpression> ::= <identifier> ‸(<refstring> ‸) |
<refexpression> <localrulenumber> <identifier> ‸(<refstring> ‸)

Again, note that a refformula cannot be both a refstring and a refexpression.

4. Our metalanguage.

In our discussion we shall often talk about formulas etc. without specifying them explicitly. For example, we want to say "let F denote a formula", or "let F denote the formula p ⌜ ϕ ⌝ ". Now F is a "metalingual symbol". We want to be able to make a difference between the symbol F and the formula denoted by F. In the latter case we shall underline F. This makes it possible to write concatenation in the form of actual concatenation. Example: if F denotes the formula p ⌜ ϕ ⌝ , and if G denotes q ⌜ ϕ ⌝ , then ϕ ^ F ^ G is the formula ϕ ^ p ⌜ ϕ ⌝ ^ q ⌜ ϕ ⌝.

We shall use the following standard metalingual symbols as variables, (possibly used with index)

s for strings,
e for expressions,
S for refstrings,
E for refexpressions,
f for fake string symbols,
i for identifiers,
l for local rule numbers.

We can now say that a refformula has one of the following four forms, according to section 3:

$$\underline{f} \tag{4.1}$$
$$\underline{S} \,\hat{}\, \underline{E} \tag{4.2}$$
$$\underline{i} \,\ulcorner\, \underline{S} \,\urcorner \tag{4.3}$$
$$\underline{E}\,\underline{l}\,\underline{i} \,\ulcorner\, \underline{S} \,\urcorner \tag{4.4}$$

5. The operators leftform and rightform.

The operators leftform and rightform are mappings of the set of all refformulas (of the reference language) into the set of formulas (of the object language).

We shall define them recursively by saying what they do to each of the forms (4.1)-(4.4). The following table describes the recursion:

refformula	leftform	rightform
f	f	f
S ; E	leftform (S) ; leftform (E)	rightform (S) ; rightform (E)
i (S)	i (leftform (S))	i (rightform (S))
E 1 i (S)	leftform (E)	i (rightform (S))

Note that as a consequence of our conventions of section 4, we underline in order to pass from metalanguage to language. E.g., leftform (S) is a name for the expression that is the leftform of the expression named S; it is not the expression itself. If we want to have the expression itself as a component of a concatenation in the object language, we have to underline it.

It is easy to verify by recursive reasoning that leftform and rightform map refstrings into strings, and refexpressions into expressions.

6. Definitional equivalence.

As we explained in the introduction, a refformula F of the reference language will be used to express how definitional equivalence of its leftform and its rightform can be established by means of successive application of the substitution rule and the local rules. Every local rule symbol in F occurs between an E and an i (S) (see (4.4)), and the thing that matters is, that the rightform of E and the leftform of i (S) are indeed directly definitionally equivalent according to local rule 1. If this condition is satisfied for every local rule symbol in F , then F will be called suitable.

This definition of suitability can be given recursively by splitting according to (4.1)-(4.4). Concerning (4.1.), we agree that f is suitable. Concerning case (4.2), we agree that if both S and E are suitable, then S ; E is suitable. Concerning case (4.3), we agree that if S is suitable, then i (S) is suitable. Concerning case (4.4), we agree that if

(i) E is suitable,

(ii) S is suitable,

(iii) the rightform of E is directly definitionally equivalent to the leftform of i (S) according to local rule 1

then E 1 i (S) is suitable.

Owing to the power of the subexpression rule (see section 1) we can say:

If F is a refformula and if F is suitable, then the leftform and the rightform of F are definitionally equivalent.

7. Procedure for testing definitional equivalence.

We shall describe a procedure that does the following. It investigates whether a given refformula is suitable, and, if so, it produces leftform and rightform of that refformula.

The procedure call

suitable (F , L , R , failure) (7.1)

can be written in any program where L and R are declared as formula, under the assumption that in this program F is a refformula, failure is a label, and that the procedure statement of "suitable" is available in the program. The effect of (7.1) is the following: If F is suitable, then L and R get as value the leftform of F and the rightform of F, respectively; if F is not suitable, however, then the only effect is a jump to the label failure.

Our procedure "suitable" is written in a language that differs from ALGOL 60 in a few respects.

(i) We use refformula, formula, fakestringsymbol, indentifier, localrulenumber as types (the use of label is standard in ALGOL 60).

(ii) A clause like

if F has the form i (S) then P; (7.2)

is executed as follows. If the expression represented by the variable F does not have the form i (S) then nothing happens. If it does have that form, then the variables i and S get the values they have in F, and, after that, P is executed. Example: if the procedure call was

suitable(a (ϕ ; b (ϕ ; x (ϕ)) , L , R , label);

then (7.2) has the following effect

i := a; S := ϕ ; b (ϕ ; x (ϕ)); P;

(iii) The program contains a boolean expression

defeq (l , R_l , i (L_2)).

It means "R_l is directly definitionally equivalent to i (L_2) according to the local rule indicated by l". This can be either true or false.

```
procedure suitable (F, L, R, lab); value F, lab; refformula F;
          formula L, R; label lab;
    begin refformula S, E; formula L1, R1, L2, R2; fakestringsymbol f;
          identifier i; localrulenumber l;
       if F has the form f then
          begin L := f; R := f; goto end end;
       if F has the form i ( S ) then
          begin suitable (S, L1, R1, lab);
                L := i ( L1 ) ; R :=  i ( R1 )  ; goto end
          end;
       if F has the form S ; E then
          begin suitable ( S, L1, R1, lab);
                suitable (E, L2, R2, lab);
                L := L1 ; L2 ; R := R1 , R2 ; goto end
          end;
       if F has the form E 1 i ( S ) then
          begin suitable (E, L1, R1, lab); suitable (S, L2, R2, lab);
             if ¬defeq ( 1 , R1 , i ( L2 ) ) then goto lab;
                L := L1; R := i ( R2 )
          end;
end:
    end suitable;
```

Reference

[1] N.G. de Bruijn, Coding system for AUT-QE, Dept. Math., Technological University, Eindhoven, Internal Report, Notitie 11 (10 December 1970).