Embedding complex arithmetic

The following is an effort to please ALGOL users, more than to please language designers. It is an answer to the undeniable fact that those, who should like to work extensively with complex numbers will find it hard to use ALGOL 60, and for two reasons: it is hard to express the computing process, and after having done so, it becomes equally hard for the computer.

Those who are crying for the inclusion of complex arithmetic will therefore be helped greatly by any reasonable effort in this direction, even if it has some deficiencies and lack of elegance. Those who do not care, being happy with real numbers, should not protest on account of the deficiencies as long as their non-complex programs are processed as efficiently as before.
This,therefore, will be my guiding principle: to introduce complex numbers and complex arithmetic in such a way that, whenever a price in efficiency has to be paid, this price will be paid in the program using the complex numbers. (This with usual machines and implementation techniques in mind.)

Complex variables

With the character "complex", followed by an identifiertist, we can declare a number of complex variables at the beginning of a block. A complex variable will be represented by its real and imaginary part, both as reals, i.e. not e.q. in argument and modulus. (This is of arithmetic importance, because now the "resolving power" in the complex plane has been coupled in a rigid way to that one on the real axis.)

Remark. The complex variable will be declared with the single character "complex" and not by means of "complex real". The reason is that I do not intend to cater for "complex integer".

Complex arrays

They will be declared by "complex array" and will consist of an array of complex variables, just as "real array" introduces one or more arrays of real variable.

Complex procedures

They will be declared by "complex procedure" analogous to the way in which real procedures are declared by "real procedure". When used as a function designator, they represent a complex primary.

The library will contain at least one complex procedure, for which I propose the reservation of the identifier "com"

complex procedure com (u,v); value u,v; real u,v;

Its value will be the complex number with the values of u and v as its real and imaginary parts respectively.

Complex formal parameters

The specifications "complex", "complex array", "complex procedure" are added. The library will contain at least two complex procedures for which I propose to reserve the identifiers "re" and "im".

```
real procedure re(z); value z; complex z;
real procedure im(z); value z; complex z;
```

When used as a function designator they have the value of the real and imaginary part, respectively, of the value of the actual parameter. Due to the representation we can state for any complex variable z

```
z = com(re(z), im(z)).
```

A formal parameter specified "complex", may occur in the value list. If a formal parameter specified "complex array" is also allowed in the value list. I can hardly blame implementers if they do not cater for it. (We should bear in mind that the intention of this proposal is to provide something what is asked for.)

The more difficult decisions to take are concerned with
1. What transfer functions to invoke automatically
2. The semantics of expression evaluation (control of types of intermediate results)
3. Whether the special character i should be introduced and if so, how.

My intention is to see to it that the complexity of all intermediate results is known. The restriction, that all complex expressions should be homogeneously of type complex is too strong, therefore I suggest automatic invoking of the transfer of arithmetic (i.e. integer or real) to complex.

If "op" is one of the five binary operations "+", "-", "*", "/" or "else",

```
if c is an expression of type complex,
if ar is an expression of type arithmetic,
```

then the result will be of type complex, and "c op ar" will be interpreted as "c op com(ar,0)" and "ar op c" as "com(ar,0) op c", assuming the five operations defined for complex arguments.
If c is a complex variable, the assignment statement "c:= ar" will be interpreted as "c:= com(ar,0)"

The net effect of this rule will be that all non-complex sub-expressions will be evaluated in terms of integer and real arithmetic. A program, not using the complex arithmetic at all will proceed at full speed, as desired.

The definition of the powering operator will be extended to include the case with complex base and integer exponent. The result will then be of type complex.

The unary + and - signs in front of a complex primary will have their usual meaning.

I am inclined to restrict the automatic invoking of the transfer function to the cases stated above. For one thing, I feel that automatic invoking of the transfer function from complex to real should never take place.

if "arv" stands for a real or integer variable
if "c" stands for a complex expression
the assignment
"arv:= c" will not be allowed and the translater can already give an error message.
In that case the program should write

"arv:= re(c)" or "arv:= mod(c)", just as he wishes.

The next question to decide is whether we admit an arithmetic actual parameter if the formal one is specified as complex.
I think we should, although there are some dangers lurking!

We consider two procedures

procedure p1(z); complex z; begin ..... (z) ..... end; and

procedure p2(z); complex z; begin ....; z:= ..... end;

the difference being, that inside p1 only interest is shown in the right hand value of the formal parameter, whereas in p2 an assignment is made to it(as well).

If we do not provide for automaticly invoking the transfer function, the call "p1(com(z,0))" would be OK, the call "p2(com(z,0))" would have to be rejected at some stage or another. The implementation of this rejection runs shortly as follows.

At call side the actual parameter is specified in its so-called formal locations. Of these formal locations, one is reserved for the evaluation of the right hand value and one for the left hand value, a specification which is given in the form of an instruction, which at run time will be the argument of an execute operation. At call side the actual parameter "com(z,0)" is recognized as one without a left hand value. As a result the calling sequence will insert an alarmjump at the word reserved for the left hand value.

If we decide, that the transfer function from arithmetic to complex should be involved automatically, then we can just write p1(x) and p2(x).
In the case "p1(x)" the system must provide for automatic transfer from arithmetic to complex, in the case p2(x) the system should give an alarm. The requirement is, that we can devise a mechanism, catering for this, without slowing down the work of the non-complex user. A second requirement is that the calling sequence will be independent of the procedure called.

We can achieve this by two stage definition of the formal locations. On account of the specification "formal" the procedure starts to inspect the formal locations. If the actual happens to be an arithmetic variable it can change the formal location by replacing the word, reserved for the left hand value by an alarm jump. In this way the price is paid by the complex user.

Finally I have come to the conclusion that in a system like this there is no room for i, or any other way of representing complex constants. Instead of "u+iv" the user is kindly requested to write com(u,v). If he wishes to do so, he can declare "complex i"; and assigne "i:= com(0,1)" after which he can write "u+i*v".

My dislike for i is that I do not know how to regard it. Is it a unary operator, of which it has not been decided whether it works to the left or to the right? ("u+iv" and "u+vi"). Or is it a constant, not unlike true and false? If so, we should require the user to write "u+i*v" or "u+v*i". But then the use of the multiplication sign (and the plus sign, for that matter) is a sheer mistification, and there is only a point in this. If the translator recognizes these particular circomstances and is able to introduce a further type, call it"imaginary", at least for anonymous results. Then the question may be raised "i * i", this will equal -1; the real result -1 or the complex one?

All this mess is introduced at the moment one tries to include i. Therefore I propose not to do it, hoping that every user, crying for complex numbers, will be glad with the facilities provided, instead of suffering from an absent i.