Check for updates

# Even More on Advice on Structuring Compilers and Proving them Correct: Changing an Arrow

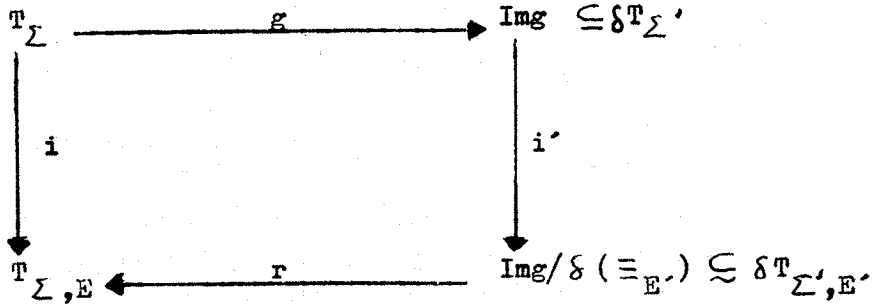
Fernando Orejas
Facultad de Matemáticas
Universidad Complutense
Madrid 3, SPAIN

I had just finished the first draft of the paper "On Representation of Data Types" and I was, once more, thinking on programming languages as data types (see, for example, Gaudel (1980) or Broy and Wirsing (1980)) and considering translation as representation of data types, when, turning my attention to ADJ (1979), I discovered that my concept of representation correctness was clearly connected with the concept of compiler correctness in the tetralogy McCarthy and Painter (1967), Burstall and Landin (1969), Morris (1973) and ADJ (1979). Even diagrams looked similar. But, suddenly, I realized that an arrow had a different direction. At first, I thought that I had to be wrong, as it usually happens in such cases. But, looking at the problem more carefully, I, finally, decided that I was correct and they were wrong.

In my paper (Orejas (1980)), representations were defined in terms of derivors (in a more general sense than ADJ(1976)). Intuitively, a derivor $\delta$ from a signature $\Sigma$ to a signature $\Sigma'$, is a pair (f,d), where f is a function from S to $S'^{*}$ (S and S' are the sets of sorts of $\Sigma$ and $\Sigma'$, respectively), and d is a family of functions associating, to each operation symbol $\sigma \in \Sigma_{w,s}$, a procedure, written in terms of the operations of $\Sigma'$, wit f(w) input parameters and f(s) output results.

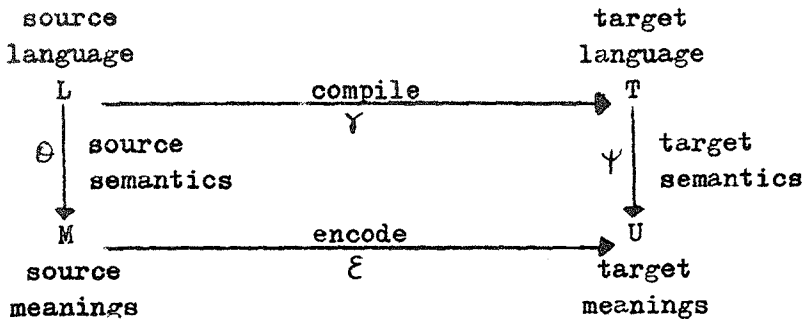
Given a $\Sigma'$-algebra A and a derivor $\delta$ from $\Sigma$ to $\Sigma'$, we may define the derived $\Sigma$-algebra $\delta A$ as the algebra with carriers: $\forall s \in S \ \delta A_s = A^{f(s)}$ and with operations: $\forall \sigma \in \Sigma \quad \sigma_{\delta A}$ is the function computed by the derived procedure $d(\sigma)$.

In these terms, given data types $T_{\Sigma,E}$ and $T_{\Sigma',E'}$, a derivor $\delta$ from $\Sigma$ to $\Sigma'$, is a representation of $T_{\Sigma,E}$ by $T_{\Sigma',E'}$, if and only if there exists a (unique) homomorphism r, making commutative the diagram:




where i and i' are the canonical epimorphisms, mapping every element into its class of equivalence, g is the unique homomorphism from the initial $\Sigma$-algebra $T_\Sigma$ into $\delta T_{\Sigma'}$, and $\delta(\equiv_{E'})$ is the $\Sigma$-congruence derived intuitively from $\equiv_{E'}$. Note that r is the usual representation function of Hoare (1972).

Now, the compiler correctness diagram (as in ADJ (1979)) is:




and it is said that the compiler $\gamma$ is correct if the diagram commutes.

If we consider languages as data types, the analogy compiler-representation is evident. $T_{\Sigma}$ and $T_{\Sigma'}$ are the sintax of the source and target languages and $T_{\Sigma,E}$ and $T_{\Sigma',E'}$ are their semantics. Thus, the only problem is the direction of the encode arrow.

ADJ,in their paper, point out a problem: there are degenerate cases in which the diagram is commutative and $\gamma$ is not a correct compiler, for example, if T and U are one-point algebras. They suggest to require $\varepsilon$ to be injective, but that is equivalent to ask for the existence of an inverse $\varepsilon^{-1}$ (or r) from a subalgebra of U into M.

Of course, that is not the only reason for changing the arrow, we may prove that there are cases (probably, many) in which $\gamma$ is a correct compiler and the diagram fails to be correct because $\varepsilon$ is not a function,since two or more values of the target meanings are associated to a single value of the source meanings.

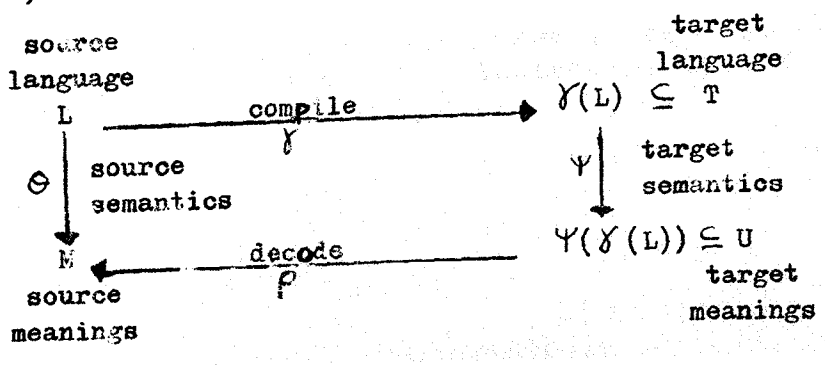
Here is a quite reasonable example. Suppose the source language is Pascal, it should be obvious that the following programs have the same meaning:

```
var x,y: integer;
begin
  x:=1;
  y:=2;
  write(x,y)
end.
```

```
var y,x: integer;
begin
  x:=1;
  y:=2;
  write(x,y)
end.
```

now, suppose the compiler assigns memory locations to variables following the order of declaration, then, if we consider the meaning of a machine program as a function,mapping storage to storage (central plus secondary), it should also be obvious that the meaning of the translations of those programs is different.

Thus, the correct diagram would be:



For more details, as soon it is written, see the paper "Compilers as data Type Representations" (Orejas (198?)).

REFERENCES

ADJ (J.A. Goguen, J.W. Thatcher, E.G. Wright), 1976
"An initial algebra approach to the specification, correctness, and implementation of abstract data types", IBM Research Report RC-6487.

ADJ (J.W. Thatcher, E.G. Wagner, J.B. Wright), 1979,
"More on advice on structuring compilers and proving them correct", IBM Research Report RC-7588.

M. Broy, M. Wirsing, 1980
"Programming Languages as abstract data types", Proc. of the 5th Colloquium on "Arbres en Algèbre et en Programmation".

R.M. Burstall, P.J. Landin, 1969
"Programs and their proofs: an algebraic approach", Machine Intelligence 4.

M.C. Gaudel, 1980
"Génération et preuve de compilateurs basées sur une sémantique formelle des langages de programmation", These d'Etat.

C.A.R. Hoare, 1972
"Proofs of correctness of data representations", Acta Informatica 1, pp.271-281.

J. McCarthy, J. Painter, 1967
"Correctness of a compiler for arithmetic expressions", Mathematical aspects of Computer Science, Proc. of Symposia in Applied Mathematics, vol. 19 (J.T. Schwartz, Ed.) American Math. Soc., Providence R.I.

F.L. Morris, 1973
"Advice on structuring compilers and proving them correct", Proc. ACM Symposium on Principles of Programming Languages, Boston.

F. Orejas, 1980
"On the representation of data types", unpublished draft.

F. Orejas, 198?
"Compilers as data type representations", in preparation.