Programming Languages (CS302 2007S)

Liskov et al. - Abstraction Mechanisms in CLU

Comments on:

Liskov, B., Snyder, A., Atkinson, R., and Schaffert, C. (1977).Abstraction Mechanisms in CLU. *Communications of the ACM* **20**(8), August 1977.

Except for some minor differences in syntax, the CLU code samples look a whole lot like Java.

This is the first time I have heard of the programming language CLU. The syntax for CLU seemed a bit unusual, perhaps because it was developed in the 1970s. Does CLU support inheritance and polymorphism? I read online that CLU supports parametric polymorphism, what does this mean?

CLU is ugly.

The authors said "Thus variables are completely private to the procedure in which they are declared and cannot be accessed or modified by any other procedure". Is that a design choice to limit mistakes of the programmer or was that done for a different reason?

"Note that when the CLU compiler does type checking, it is, in addition to enforcing the constraint that permits the proof to be decomposed, also performing a small part of the actual proof". I like the idea that the compiler can and should prove part of the correctness of a program.

On pg565, fourth paragraph of second column, Liskov et al says that "the behavior of the data objects must be completely characterized by the set of operations." On pg570, first paragraph of first column, they say that "CLU objects exist independently of procedure activations." My question: The existence of (data) objects seems to be useless without procedure activations since the objects can be manipulated only through the operations provided with them. Hence, should the existence of the objects be dependent on procedure activation? Could you also clarify what they mean by "procedure activation"?

One thing in the paper is not clear to me. That is put_n(r,x) method, which the paper says "makes x the n component of the record r (568)". I understand this as that this method can update a component of an object during running time. We have seen the similar thing in Ruby (we can update an object's method). However, CLU does type checking in compiling time, which sounds contradictory to what I understand. For example, if a program declares an integer as the type of a component and pass the type checking in compiling time, then during running time, it invokes the put_n(r,x) to change the type to float. What will happen then?

In terms of formal vs informal, CLU appears to err on the side of formal. Despite the flexibility to be able to define your own types and procedures there is a specified format for using the abstractions that CLU has built in. My question is, how helpful is it really to formalize these abstractions? To some degree it creates a standard that you assume a function will follow, but so long as the methods that you define for any given object are allowed to vary so long as your implementation for a given thing isn't standardized it seems like you'd still run into problems. The example of the hashtable comes to mind. Perhaps in CLU you've got it defined to have a create, retrieve, remove, and add set of functions associated with it... but so long as the

implementation of the functions is unknown, or the exact behavior is unclear, (which it seems like its still possible to do with CLU) I'd think you'd run into the same problems.