第三章笔记

2019年9月7日 星期六　　下午7:08

One powerful design strategy, which is particularly approprite to the constrution of programs for modeling physical systems, is to base the structure of programs on the structure of the system being modeled. For each object in the system, we construct a corresponding computational object. For each system action, we define a symbolic operation in our computational model. Our hope in using this strategy is that extending the model to accommodate new objects or new actions will require no strategic changes to the program, only the addition of the new symbolic analogs of those objects or actions. If we have been successfully in our system organization, then to add a new feature or debug an old one we will have to work on only a localized part of the system.

The difficulties of dealing with objects, change, and identity are a fundamental consequence of the need of grapple with time in our computational models. These difficuties become even greater when we allow the possibility of the concurrent execution of programs.

An object is said to "have state" if its behavior is influenced by its history.

Encapusulation reflects the general system-design principle known as the hiding principle: hiding principle: one can make a system more modular and robust by protecting parts of the system from each other; that is, by providing information access only to those parts of the system that have a "need to know".

Soon as we intrduce assignment into our language, substition is no longer an adequete model of procedure application.

We have alreadly used begin implicity in our programs, because in Scheme the body of a procedure can be a sequence of expressions. Also, the <consequent> part of each clause in a cond expression can be a sequence of expressions rather than a single expression.

Two calls to the same procedure with the same arguments always produced the same result.

Actually, this is not quite true. One exception is random number generator. Another exception involved the operation /type tables we introduced in section 2.4.3, where the values of two calls to get the same arguments depended on intervening calls to put. On the other hand, until we introduce assignment, we have no way to create such procedures ourselves.