other. Usually B will be code, but the rule applies to all text substitutions.

As an example, suppose that a module must guarantee to increment from zero the observed variable x at least once and at most twice. Any program using the module may examine but not modify x's value. One way is to use module A, which has hidden variable j. When A runs it initially chooses j's value at random (either 1 or 2), and then increments x until it equals j. Alternatively the module B with hidden variable b could be used. B first increments x once. It then decides whether to increment again by choosing b's value (either 0 or 1) arbitrarily. It increments again if b is 0 and not otherwise.

The challenge is to show that B is a valid substitute for A, but with arbitrary runtime behaviour this is not so easy. It boils down to comparing their behaviour. In technical terms it means that any program using the module B algorithmically refines any program using A. To make the comparison precise, imagine that a program's behaviour can be broken down into individual steps, each one observed as a transformation of states (values of program variables). The permitted transformations can be represented in a diagram as arrows connecting pairs of states (see figure). Where choices can be made this will be observed as a branching. Algorithmic refinement just means that B's diagram can be superimposed into A's — in other words any transition step made by B can be made by A.

In the case of data refinement this definition cannot be used directly, because the state spaces are not the same — both B and A have their own hidden variables which must be accounted for. However, we can recover the idea behind algorithmic refinement by considering only the observed behaviour — remember that a program using the module never sees the hidden variables. What we need is the state diagram of the observed variables, and this is obtained from the original diagram by relabelling the states only by the observed variables whilst preserving its shape. B is said to simulate A if we can superimpose B's relabelled diagram into A's, and simulation implies algorithmic refinement.

Unfortunately this analysis is insufficient for all the cases when the diagrams have branching. Recall the example. On initialization A decides, unbeknown to the user, what the final value of x is to be — it is the value assigned to j. The more fickle B delays for as long as possible the decision determining x's final value, and this uncertainty is seen as a branching when x is already 1, making simulation impossible (see figure).

So the analysis must be sneakier. We turn the clock backwards — instead of looking for transitions starting from a particular state, we look for backward transitions that end up at a particular state. As for simulation, diagrams of the backward transitions for the modules can be constructed and compared, giving a new relation called co-simulation. This view of programming solves the cases when random choices are made, because the possible choices that were never made become irrelevant. What's more, it turns out that the co-simulation implies algorithmic refinement. In the example, looking back from any final value of x means that only the choice to increment is observed, and this is the same in both A and B. In terms of diagrams, it means that we can safely identify A's first two states — that is, regard them as identical.

In the theory based on diagrams for programs, simulation and co-simulation are different, and both are needed to prove mathematically that all valid substitutions are correct[1]. But now, thanks to Paul Gardiner and Carroll Morgan[2,3], the story has a far more satisfactory ending.

There is a more general theory of programming (related to the idea of backward transitions) in which programs are associated with predicate transformers, which are more detailed objects than diagrams. The state diagrams are part of this more general setting rather in the way that integers (1, 2...) are embedded into the larger set of rational numbers (1/2, 3/5, 2/1...). The analogy runs deeper. Just as in the rational numbers (but not in the integers) it is possible to divide any number by another, in the more general theory there is an operation analogous to division. Gardiner and Morgan noticed that this 'division' operation makes it possible to combine the notions of simulation and co-simulation — simulation being the 'numerator' and co-simulation the 'denominator'. This compound operator automatically applies both checks for simulation and co-simulation in one go, giving a neat, single characterization of data refinement.

Giving equal status to specification and code brings programming into the mathematical domain in which it truly belongs and out of the wilderness of the 'guess and see' methods. Improvements to refinement rules allow greater freedom of expression whilst maintaining the possibly life-preserving correctness. Placed in this context, the better understanding of program behaviour is likely to turn the process of developing code into a safer ride with no nasty surprises at the end of it. □

Annabelle McIver is in the Computing Laboratory, University of Oxford, Parks Road, Oxford OX1 3QD, UK.

1. Hoare, C. A. R., He Jifend & Sanders, J. W. *Inform. Proc. Lett.* **25**, 71–76 (May 1987).
2. Gardiner, P. H. B. & Morgan, C. C. *Formal Aspects Comput.* **5**, 367–382 (1993).
3. Gardiner, P. H. B. & Morgan, C. C. *A Single Complete Rule for Data Refinement* in *On The Refinement Calculus* (eds Morgan, C. & Vickers, T.) (Springer, Berlin, 1994).

DAEDALUS

Walls have voices

A SUPERMARKET bar-code reader has a special speaker that bleeps when the code is verified. Daedalus muses that the reading process should generate a bleep itself. As the laser beam sweeps across the white label, it encounters a sequence of black bars which absorb its energy and convert it to heat. The scan therefore releases a series of heat pulses which, warming up the local air and increasing its pressure, should launch a sound with the waveform of the bar code. The resulting feeble chirp is presumably lost in the hubbub of the supermarket.

DREADCO's physicists are now trying to increase it. One team is adapting a past triumph, a 'blacker-than-black' printing ink that sets to a microfibrous surface like black velvet. This absorbs laser light totally, and transfers its energy very efficiently to the air trapped between the fibres. A rival team is devising an ink foamed with hydrogen, which conducts heat faster and therefore expands more rapidly than any other gas. One of these schemes, or both in tandem, should make possible a bar code which, when scanned by a laser, emits a loud characteristic chirp of its own.

The new sonic ink should find uses far beyond the banal world of retailing. Daedalus sees it as a source of high-fidelity sound. A laser scanning a sonic-ink soundtrack should generate a perfect sound, with no amplifiers or speakers to contribute noise or distortion. At first he envisaged a sort of ultimately simple gramophone, whose soundtrack-coded disk spun under the reading laser and radiated perfect sound into the air. But he then realized that even the spinning disk could be dispensed with. A sonic-ink soundtrack on any wall or ceiling would speak up when scanned from a distance by a suitable laser.

DREADCO's sonic decor will transform audio technology. Public places such as railway stations, airports and shopping malls will bear on their walls snappy advertisements, regularly scanned, as well as useful information and various emergency announcements to be activated only on demand. Sonic road-markings will be scanned by each passing vehicle by means of a scanning laser pointing down at the road. Home owners will happily hang musical sonic-ink soundtracks on their walls, to be swept at will by a neat steerable laser on a coffee table. But a track stretching right along the wall would generate music which drifted along it with the scanning spot. It might be better to fold the track into a compact zig-zag or spiral pattern, print it on a simple poster, and let the laser trace along it by optical feedback.

David Jones