

```
(define (pcset_o state ffstate inputs)
  (list
    (logical-and2 inputs[1]
      (logical-and2 inputs[2]
        inputs[3])))
```

Figure 4: Code for circuit in Figure 2 automatically produced by partial evaluation, returning just the final value of the output

The complete PC-set code (of which there is a large amount, as mentioned in [4]) is generated only when the simulator returns the complete history of the circuit. Changing the simulator to only return the final values of the output nodes (a three line change) dramatically reduces the number of gate evaluations and the size of the produced code. An example is shown in Figure 4, which presents the compiled simulator for the sample circuit. It is much simpler than the previous compiled simulator. We have measured 4 fold to 14 fold fewer gate evaluations using this improvement (Figure 10).

The style of code generated when the simulator only reports final values lies somewhere between the LCC method and the PC-set method, so we refer to it as the LPC-set method. Indeed, using this strategy, the compiled simulator for a combinational circuit is identical to the one produced via LCC. The difference occurs for sequential circuits, where timing matters: all transitions on the outputs of combinational circuits that generate clocks (ie, clock flipflops) are computed, whereas (in a well designed circuit) only the final values of combinational circuits that produce data are computed.

6 The BACKSIM Algorithm

There is yet another source of inefficiency in the improved PC-set method: gate evaluations occur that *may* contribute to the final result, but don't *actually* contribute to the final result due to data dependencies. For example, consider a 2-input or-gate whose first input has already been determined to be *true*. We may *cut off* the evaluation of its second input because this value is not needed. Gate evaluations would be saved if gates were only evaluated when provably needed. We say that a simulator has a *selective-trace* property if it uses data-dependencies to prune unneeded evaluations. (For example, event-driven simulators are selective-trace.)

BACKSIM, a selective-trace, variable delay simulator that only evaluates gates as they are needed, has been implemented [3]. Starting from the output request of the user, it works backward through the circuit, only evaluating those gates that are needed to determine the outputs. Because of fanout, it may request the value of a gate more than once. The gate (and all the gates it depends on) are only evaluated once regardless of the number of times its value is requested.

Because of cutoff, BACKSIM performs even fewer gate evaluations than the LPC-set method. Consider the sample circuit again. If the input C is known to be 1, the signal value at node D need not be requested. This early cutoff is useful when one input to a gate is inexpensive to evaluate, e.g., a primary input, and the other input is expensive to compute.

BACKSIM's increased efficiency (w.r.t. gate evaluations) makes it a good candidate for specialization. However, there is unavoidable overhead in the method: remembering and testing whether a gate has been evaluated during the simulation. During an interpreted simulation, this overhead is

```
(define (demand state ffstate inputs)
  (list
    (if inputs[1]
      (if inputs[2]
        inputs[3]
        0)
      0)))
```

Figure 5: Demand-driven code produced by partial evaluation.

relatively small, and BACKSIM has been shown to be more efficient than event-driven simulation. For compiled simulation, the relative overhead is much larger.

We made one very simple change to the experiment for the improved PC-set method to generate a compiled simulator based on the BACKSIM algorithm: the primitive boolean functions (such as not, and, etc.) were defined in terms of if expressions, instead of as primitive functions. When logical operations are expressed as functions all inputs must be known before the logical function is computed, there is no chance for cutoff. When logical operations are expressed as conditionals, cutoff will occur, as the inputs to a gate need now not be fully computed. For example, consider the rendering of an OR gate with inputs A and B: if <expression computing A> then true else <expression computing B>. The B expression is evaluated only when the A expression evaluates to false. The simulation program for the sample circuit when logical functions are expressed as conditionals (Figure 5) clearly shows the desired cutoff. Using conditionals rather than functions loses flexibility, and increases code size, limiting us to a 2-level signal model.

The hard part of a compiled simulator based on the BACKSIM algorithm is handling gates with fanout greater than one, as the expression computing the value of the gate would appear in the arms of many if-expressions, causing duplicate code and computation. To avoid this, the expression is textually isolated, and storage is reserved for both its value, and for remembering whether the value has been computed.

The partial evaluator automatically produces the code to this. When producing Scheme code, it uses Scheme's *delay* and *force* constructs. The delay operation is given an expression and returns a an object, called a *thunk* that can be *forced*. When the thunk is forced for the first time, the expression is evaluated and the value remembered. On each subsequent force, the remembered value is returned. Figure 6 shows the use of delay and force to avoid computing shared values unless absolutely needed. Node D is shared by the two and-gates. The value at D corresponds to the expression *shared-value*, which appear as a separate binding since it is shared by the last 2 lines in the code. Without the delay-force construct, *shared-value* will be evaluated every time the code is executed. This is obviously not necessary when P & Q are both 0. Early cutoff without unnecessary computation is achieved by the delay in the binding of *shared-value* and the force in the last 2 lines.

When we ran this experiment, we found that, when the compiled simulator was expressed in Scheme code, the costs of delays far outweighed their benefit. The compiled simulators ran (expressed in Scheme) 2 to 3 times slower than the LPC-set method. When expressed in C code, they ran roughly 10% slower (Figure 11) than the LPC-set method. To explain these results, we measured the number times each