

There should be no need for the programmer to give directions concerning details of scheduling and communication, but the programmer must still design the algorithm with concurrency in mind; we can only get the full benefits of parallelism if the algorithm is coded to give gross parallel structure (e.g. using a divide-and-conquer approach).

An automatic compile-time analysis of the source (called "strictness analysis") will often detect much of the inherent parallelism [Clack85], but this is still a research area and programmer annotations may be used in preference or to give additional hints to the compiler.

The resultant parallel tasks can be managed automatically at run-time, and our aim is that the entire business of administering parallel tasks should be hidden from the programmer.

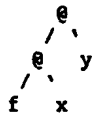
1.2 Graph reduction

A functional program is a single expression which has a natural representation as a syntax tree. In general, there will be sharing of nodes, and so the syntax tree will be a graph, which may be cyclic.

A functional program may be evaluated by manipulating the syntax graph. The evaluation proceeds by means of simple steps, each of which performs a local transformation to the graph. Each step is called a reduction, and the process is known as "graph reduction" [Turn79]. A reducible expression is often referred to as a redex.

Reductions may take place concurrently, since they cannot interfere with each other, and evaluation is complete when there are no further reducible expressions (normal form).

The curried application of a function "f" to two arguments x and y is represented like this:



where "e" represents an application cell, containing pointers to function and argument.

Consider the following functional program:

```

LET  f x = AND x x
IN   f (NOT TRUE)
  
```

(the LET introduces a definition of the function f, which takes a single argument x. Function application is denoted by Juxtaposition, thus (NOT TRUE) denotes the function NOT applied to the argument TRUE).

The figure below shows how it would be evaluated. Notice that after each reduction the root of the redex is overwritten with the result of the reduction.

