Exercise 5: Programmable Optimizations Using POET
July 11, 2014

1. Look at following automatically generated POET optimization script. Can you identify all the necessary components of a source-to-source optimizer?

```
include opt.pi

<parameter inputFile message="input file name"/>
<parameter outputFile  message="output file name"/>
<parameter tp1 type=1.._ default=2 message="number of threads to parallelize loop Nest2" />
<parameter tp2 type=INT default=(32) message="Blocking factor for loop nestNest2" />
<parameter tp3 type=1.._ default=256
                message="number of loop iterations to run by different threads for Nest2" />
<trace inputCode,Nest1,Nest2,Nest3,Nest4 />

<input from=inputFile syntax="Cfront.code" to=inputCode />

<trace Nest4_grp2 = (Nest4) />
<trace Nest2_grp2 = (Nest2) />
<trace Nest2_grp3 = (Nest2_grp2) />
<trace Nest2_ParallelizeLoop_private_th1 = (("k" "j" "i")) />

<trace _decl1="" />
<eval EraseTraceHandle[repl=(_decl1 Nest1)](Nest1, inputCode); />
<define TRACE_DECL _decl1 />
<define TRACE_INCL inputCode/>
<define TRACE_TARGET inputCode />

<define TRACE_VARS (Nest2_ParallelizeLoop_private_th1) />

<eval EraseTraceHandle[repl=Nest4_grp2](Nest4,inputCode);
 EraseTraceHandle[repl=Nest2_grp3](Nest2,inputCode);
 BlockLoops[factor=tp3](Nest2_grp3[Nest.body],Nest2_grp3);
 ParallelizeLoop[threads=tp1;private=Nest2_private_th1](Nest2_grp3);
 TraceNestedLoops(Nest2_grp2,Nest2_grp3[Nest.body]);
 BlockLoops[factor=tp2;trace_ivars=Nest2_private_th1](Nest4_grp2[Nest.body],Nest2_grp2);
  />

<output to=outputFile syntax="Cfront.code" from=inputCode />
```

2. Using POET, write a translator that automatically transforms the following code

```
/*@;BEGIN(Nest1=Nest)@*/for (t = 0; t < timesteps; t++) {
  /*@;BEGIN(Nest2=Nest)@*/for (k = 1; k < nz - 1; k++) {
    /*@;BEGIN(Nest3=Nest)@*/for (j = 1; j < ny - 1; j++) {
      /*@;BEGIN(Nest4=Nest)@*/for (i = 1; i < nx - 1; i++) {
        Anext[Index3D (nx, ny, i, j, k)] = A0[Index3D (nx, ny, i, j, k + 1)] +
          A0[Index3D (nx, ny, i, j, k - 1)] + A0[Index3D (nx, ny, i, j + 1, k)] +
          A0[Index3D (nx, ny, i, j - 1, k)] + A0[Index3D (nx, ny, i + 1, j, k)] +
          A0[Index3D (nx, ny, i - 1, j, k)] -  A0[Index3D (nx, ny, i, j, k)] *fac ;
      }
    }
  }
  temp_ptr = A0;
  A0 = Anext;
  Anext = temp_ptr;
}
```

to the following equivalent code.

```
/*@;BEGIN(Nest1=Nest)@*/for (t = 0; t < timesteps; t++) {
  /*@;BEGIN(Nest2=Nest)@*/for (k = 1; k < nz - 1; k++) {
    /*@;BEGIN(Nest3=Nest)@*/for (j = 1; j < ny - 1; j++) {
```

```
      /*@;BEGIN(Nest4=Nest)@*/for (i = 1; i < nx - 1; i++) {
        if (t%2 == 0) { l0 = A0; lnext = Anext; }
        else {lnext = A0; l0 = Anext; }
        lnext[Index3D (nx, ny, i, j, k)] = l0[Index3D (nx, ny, i, j, k + 1)] +
          l0[Index3D (nx, ny, i, j, k - 1)] + l0[Index3D (nx, ny, i, j + 1, k)] +
          l0[Index3D (nx, ny, i, j - 1, k)] + l0[Index3D (nx, ny, i + 1, j, k)] +
          l0[Index3D (nx, ny, i - 1, j, k)] -  l0[Index3D (nx, ny, i, j, k)] *fac ;
      }
    }
  }
}
```

What is the criteria of your transformation? Can it be applied to other similar situations?