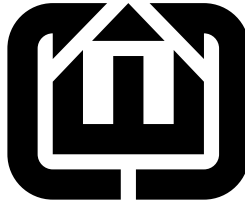


CEC GRC simulator



Stephen A. Edwards, Jia Zeng, Cristian Soviani
Columbia University
sedwards@cs.columbia.edu

Abstract

This simulates much of the GRC format. It was designed primarily for testing and is not terribly efficient.

Contents

1	Visitor Methods	2
1.1	Fork	2
1.2	Terminate	2
1.3	Sync	2
1.4	Enter	4
1.5	Suspend	6
1.6	Actions: Emit, Exit, Assign, Startcounter, Nop	6
1.7	Switch	9
1.8	Test	10
1.9	Expressions	11
1.10	DefineSignal	13
2	The GRCsim Class	14
3	Helper Methods	16
3.1	schedule and schedule_child	16
3.2	intVal	16
3.3	doswitch	17
3.4	dfs	18
3.5	execute max	19
3.6	Execute Vectors	20

3.7	init	22
3.8	Clear Inputs	23
3.9	dotick	24
4	Top-level files	25

1 Visitor Methods

1.1 Fork

Fork schedules all its children.

2a *<method declarations 2a>*≡
 Status visit(Fork &);

2b *<method definitions 2b>*≡
 Status GRCsim::visit(Fork &s)
 {
 if (debug) cerr << cfgmap[&s] << ":fork" << '\n';
 for (vector<GRCNode *>::iterator i = s.successors.begin() ;
 i != s.successors.end() ; i++) schedule(*i,globalcc);
 return Status();
 }

1.2 Terminate

A Terminate node simply schedules its children.

2c *<method declarations 2a>*+≡
 Status visit(Terminate &);

2d *<method definitions 2b>*+≡
 Status GRCsim::visit(Terminate &s)
 {
 if (debug) cerr << cfgmap[&s] << ":term " << s.code << endl;
 schedule_child(&s,globalcc);
 return Status();
 }

1.3 Sync

The Sync node computes the maximum exit level among all its predecessors (these should be Terminate nodes by construction) and schedules its corresponding child.

2e *<method declarations 2a>*+≡
 Status visit(Sync &);

```

3  <method definitions 2b>+≡
    Status GRCSim::visit(Sync &s)
    {
        int MAXTERM = 0;
        int range_bg = 0;    //range begin
        int range_end = 0;  //range end
        int i;

        assert(s.dataPredecessors.size() > 0); // grcdps should have computed them

        //find maximam term lvl
        for ( vector<GRCCNode *>::const_iterator it = s.dataPredecessors.begin() ;
              it != s.dataPredecessors.end() ; it++ ) {
            Terminate *t = dynamic_cast<Terminate*>>(*it);
            assert(t); assert(t->code >= 0);
            if (debug & debugSync)
                if (debug) cerr << cfgmap[&s] << ":sync checking "
                    << cfgmap[t] << ":term " << t->code << '\n';
            if (t->code > MAXTERM)
                MAXTERM = t->code;
        }

        int orall[MAXTERM+1]; //term lvls and correspond ctrl values

        for(i = 0; i <= MAXTERM; i++)
            orall[i] = 0;

        //OR all the term lvl code & control value
        for ( vector<GRCCNode *>::const_iterator it = s.dataPredecessors.begin() ;
              it != s.dataPredecessors.end() ; it++ ) {
            Terminate *t = dynamic_cast<Terminate*>>(*it);

            if (sched[t] != 0){
                if (sched[t] == 1)
                    orall[t->code] = 1;
                else if (orall[t->code] == 0)
                    orall[t->code] = sched[t];
            }
        }

        //get the range of active term lvls
        bool bg = false;
        for (i = MAXTERM; i >= 0; i--){
            if (!bg){
                if (orall[i] != 0){
                    bg = true;
                    range_bg = i;
                }
            }
            if (orall[i] == 1)

```

```

        break;
    }
    range_end = i;

    assert(range_bg < (int) s.successors.size());
    assert(orall[range_bg] != 0);

    if (orall[range_bg] == -1){//unknown
        if (debug) cerr << cfgmap[&s] << ":may sync at level ";
        for(i = range_bg; i >= range_end; i--){
            if(orall[i] != 0){
                schedule(s.successors[i],-1);
                if(debug) cerr<< i << ' ';
            }
        }
        if(debug) cerr<<'\n';
    }
    else{
        assert(range_bg == range_end);
        assert(orall[range_bg] == 1);
        schedule(s.successors[range_bg],globalcc);
        if (debug) cerr << cfgmap[&s] << ":sync at level " << range_bg << '\n';
    }

    return Status();
}

```

1.4 Enter

The key operation for the Enter node is walking up the selection tree to find the closest exclusive node whose child this node is.

```

4  <method declarations 2a>+≡
    void setState(STNode *n, bool isInit);

```

```

5  <method definitions 2b>+≡
    void GRCSim::setState(STNode *n, bool isInit)
    {
        STexcl *exclusive = 0;

        for (;;) {
            assert(n);

            STNode *parent = n->parent;

            if (dynamic_cast<STpar*>(parent) != NULL) {
                if (debug) cerr << "parent of node " << stmap[n]
                    << " is a parallel node" << endl;
                return;
            }

            exclusive = dynamic_cast<STexcl*>(parent);
            if (exclusive != NULL) break; // found the exclusive node
            n = parent;
        }

        assert(exclusive != NULL);

        // Locate node n among the children of "parent"
        vector<STNode*>::iterator i = exclusive->children.begin();
        while (*i != n && i != exclusive->children.end()) i++;

        assert(i != exclusive->children.end());

        int childnum = i - exclusive->children.begin();

        assert(childnum >= 0);

        // Check for program termination
        if (exclusive == stroot) {
            STleaf *lf = dynamic_cast<STleaf *>(n);
            if ((lf) && (lf->isfinal())) {
                ISFinal = true;
                if (debug) cerr << "program terminated\n";
            }
        }

        if(isInit){
            state[exclusive] = childnum;
            if (debug) cerr << " state[" << stmap[exclusive] << "] = "
                << childnum << endl;
        }
        else{

```

```

        nextstate[exclusive]=childnum;
        if (debug) cerr << "  nextstate[" << stmap[exclusive] << "] = "
            << childnum << endl;
    }

    //NOTE: Enter does not set state if any input signal is unknown
    //FIXME: it may be wrong and should change to no ctrl flw is unknown in
    //in current state.
}

```

The visitor for the Enter node marks itself as selected using `setState` and schedules its children.

```

6a  <method declarations 2a>+≡
      Status visit(Enter &);

6b  <method definitions 2b>+≡
      Status GRCsim::visit(Enter &s)
      {
        if (debug) cerr << cfgmap[&s] << ":enter " << stmap[s.st];
        setState(s.st,false);
        schedule_child(&s,globalcc);
        return Status();
      }

```

1.5 Suspend

```

6c  <method declarations 2a>+≡
      Status visit(STSSuspend &);

      This just schedules its child.

6d  <method definitions 2b>+≡
      Status GRCsim::visit(STSSuspend &s)
      {//FIXME: anything special?
        if (debug) cerr << cfgmap[&s] << ":suspend " << stmap[s.st]<<"\n";
        schedule_child(&s,globalcc);
        return Status();
      }

```

1.6 Actions: Emit, Exit, Assign, Startcounter, Nop

```

6e  <method declarations 2a>+≡
      Status visit(Action &);
      Status visit(Emit &);
      Status visit(Exit &);
      Status visit(Assign &);
      Status visit(StartCounter &);
      Status visit(Nop &);

```

Action GRC nodes simply invoke their bodies.

```
7a  <method definitions 2b>+≡
    Status GRCsim::visit(Action &s)
    {
        if (debug) cerr << cfgmap[&s];
        s.body->welcome(*this);
        schedule_child(&s,globalcc);
        return Status();
    }
```

Emit actions simply set their signal's status to present.

```
7b  <method definitions 2b>+≡
    Status GRCsim::visit(Emit &e)
    {
        assert(e.signal);
        SignalSymbol *ss = e.signal;
        if(ss->type && ss->type->name=="integer"){
            signals_v[ss]=intVal(e.value);
            if(debug) cerr<<" value : "<<signals_v[ss]<<" ";
        }
        if (debug) cerr << ".emit " << ss->name <<" = "<<globalcc<< endl;
        if(signals[ss] != 1)
            signals[ss] = globalcc;

        return Status();
    }
```

Exit actions are similar to Emit.

```
7c  <method definitions 2b>+≡
    Status GRCsim::visit(Exit &e)
    {
        assert(e.trap);
        if(debug) cerr<<"Warning: 3-valued-simulator does NOT support Exit yet!\n";
        SignalSymbol *ss = e.trap;
        if (debug) cerr << ".exit " << ss->name << endl;
        signals[ss] = globalcc; //FIXME: correct?

        return Status();
    }
```

Assign statement set their variable to the value of their expression.

```
8a  <method definitions 2b>+≡
    Status GRCsim::visit(Assign &a)
    {
        assert(a.variable);
        assert(a.value);
        assert(a.variable->type);

        if(debug) cerr<<"Warning: 3-valued-simulator does NOT support Assign yet!\n";

        if (a.variable->type->name != "integer" &&
            a.variable->type->name != "boolean") {
            throw IR::Error("Only integer and boolean variables supported");
        }

        var[a.variable] = intVal(a.value);

        return Status();
    }
```

StartCounter actions reset their counter.

```
8b  <method definitions 2b>+≡
    Status GRCsim::visit(StartCounter &stcnt)
    {
        if (debug) cerr << ":start counter\n";
        if(debug) cerr<<"Warning: 3-valued-simulator does NOT support StartCounter yet!\n";
        assert(stcnt.counter);
        assert(stcnt.count);
        counters[stcnt.counter] = intVal(stcnt.count);
        return Status();
    }
```

NOP nodes do, not surprisingly, nothing.

```
8c  <method definitions 2b>+≡
    Status GRCsim::visit(Nop &n)
    {
        if (debug) cerr << ":nop\n";
        schedule_child(&n,globalcc);
        return Status();
    }
```


1.7 Switch

```
9  <method definitions 2b>+≡
    Status GRCSim::visit(Switch &s)
    {
        assert(s.st);

        //Notice that switch always has one active child - no matter
        // its ctrlflow value is 1 or x, just pass by the ctrl value
        // to this child.

        int act_ch = doswitch(s.st);

        assert(act_ch < (int) s.successors.size());
        GRNode *child = s.successors[act_ch];
        if (debug) cerr << cfgmap[&s] << ":switch " << stmap[s.st]
            << " --" << act_ch << "--> "
            << (child ? cfgmap[child] : -1) << '\n';
        if (child) schedule(child,globalcc);

        return Status();
    }
```

1.8 Test

```

10  <method definitions 2b>+≡
    Status GRCsim::visit(Test &s)
    {
        GRCNode *successor;

        assert(s.predicate);
        //assert(s.successors.size()==2);
        int predvalue = intVal(s.predicate);

        if (predvalue==--1){//unknown

            if (debug) {
                cerr << cfgmap[&s] << ":test -- " << predvalue << " --> ";
                for(vector<GRCNode *>::iterator i = s.successors.begin();
                    i != s.successors.end(); i++){
                    if (*i) cerr << cfgmap[*i];
                    else cerr<<" (none)";
                }
                cerr<<"\n";
            }

            ternary = true;

            for(vector<GRCNode *>::iterator i = s.successors.begin();
                i != s.successors.end(); i++){
                schedule(*i,-1);
            }
        }
        else{
            assert ( predvalue >= 0 && predvalue < (int)s.successors.size() );
            successor = s.successors[predvalue];

            if (debug) {
                cerr << cfgmap[&s] << ":test --" << predvalue << "--> ";
                if (successor) cerr << cfgmap[successor];
                else cerr << "(none)";
                cerr << '\n';
            }

            if (successor) schedule(successor,globalcc);
        }

        return Status();
    }

```

1.9 Expressions

- 11a *<method definitions 2b>+≡*

```
Status GRCSim::visit(Literal &s)
{
    int val;
    if ( sscanf(s.value.c_str(), "%d", &val) != 1 ) assert(0);
    return Status(val);
}
```
- 11b *<method definitions 2b>+≡*

```
Status GRCSim::visit(LoadSignalExpression &lse)
{
    SignalSymbol *ss = lse.signal;
    assert(ss);
    return Status(signals[ss]);
}
```
- 11c *<method definitions 2b>+≡*

```
Status GRCSim::visit(LoadSignalValueExpression &lse)
{
    SignalSymbol *ss = lse.signal;
    assert(ss);
    return Status(signals_v[ss]);
}
```
- 11d *<method definitions 2b>+≡*

```
Status GRCSim::visit(LoadVariableExpression &lve)
{
    VariableSymbol *vs = lve.variable;
    assert(vs);
    if (var.find(vs) == var.end())
        throw IR::Error("reading uninitialized variable " + vs->name);
    return Status(var[vs]);
}
```
- 11e *<method definitions 2b>+≡*

```
Status GRCSim::visit(CheckCounter &chkcnt)
{
    assert(chkcnt.counter);
    assert(chkcnt.predicate);
    if (intVal(chkcnt.predicate))
        counters[chkcnt.counter]--;

    return Status( counters[chkcnt.counter] == 0 );
}
```

```

12a  <method definitions 2b>+≡
      Status GRCsim::visit(BinaryOp &e)
      {
        assert(e.source1);
        assert(e.source2);
        if(debug) cerr<<"Warning: 3-valued-simulator does NOT support BinaryOp yet!\n";
        int val1 = intVal(e.source1);
        int val2 = intVal(e.source2);

        int result =
          (e.op == "and") ? (val1 && val2) :
          (e.op == "or") ? (val1 || val2) :
          (e.op == "+") ? (val1 + val2) :
          (e.op == "-") ? (val1 - val2) :
          (e.op == "*") ? (val1 * val2) :
          (e.op == "/") ? (val1 / val2) :
          (e.op == "mod") ? (val1 % val2) :
          (e.op == "=") ? (val1 == val2) :
          (e.op == "<>") ? (val1 != val2) :
          (e.op == "<") ? (val1 < val2) :
          (e.op == "<=") ? (val1 <= val2) :
          (e.op == ">") ? (val1 > val2) :
          (e.op == ">=") ? (val1 >= val2) :
          0;

        return Status(result);
      }

12b  <method definitions 2b>+≡
      Status GRCsim::visit(UnaryOp &e)
      {
        if(debug) cerr<<"Warning: 3-valued-simulator does NOT support UnaryOp yet!\n";

        int val = intVal(e.source);

        int result =
          (e.op == "not") ? ( !val ) :
          (e.op == "-") ? ( -val ) :
          0;

        return Status(result);
      }

12c  <method definitions 2b>+≡
      Status GRCsim::visit(FunctionCall &)
      {
        throw IR::Error("Function calls are not supported");
      }

```

1.10 DefineSignal

A DefineSignal node resets its signal's presence. This is used to initialize local signals when they enter scope.

```
13a  <method declarations 2a>+≡
      Status visit(DefineSignal &);

13b  <method definitions 2b>+≡
      Status GRCsim::visit(DefineSignal &d)
      {
        assert(d.signal);
        SignalSymbol *ss = d.signal;
        assert(ss);
        if(debug) cerr<<"Warning: 3-valued-simulator does NOT support DefineSignal yet!\n";
        if (debug) cerr << cfgmap[&d] << ":DefineSignal " << ss->name << '\n';
        signals[ss] = 0;
        schedule_child(&d,globalcc);
        return Status();
      }
```

2 The GRCsim Class

This is the heart of the simulator. Derived from the `Visitor` class, its methods simulate each type of GRC node.

```

14  <GRCsim class 14>≡
      class GRCsim: public Visitor {

          int debug;
          static const int debugDFS = 1 << 1;
          static const int debugSync = 1 << 2;
          static const int debugVectors = 1 << 3;
          int useold;
          int globalcc; //a hack

          GRCgraph *top;
          EnterGRC *entergrc;
          SymbolTable *sigs;
          STNode *stroot;
          STleaf *boot;
          GRCNode *grcroot;

          Module *module;

          GRCNode::NumMap &cfgmap;
          STNode::NumMap &stmap;

          bool ternary;
          map<STNode*, int > state;
          map<STNode*, int > nextstate;
          // state & next state information for selection tree nodes

          vector<GRCNode*> topo;
          map<GRCNode*, int> sched;
          set<GRCNode*> dfs_notwhite, dfs_black;
          map<SignalSymbol*, int> signals; // Status of each signal
                                          // 0,1,2(unknown)
          map<SignalSymbol*, int> signals_v;
          map<Counter*, int> counters; // Value of each counter
          map<VariableSymbol*, int> var; // Value of each variable

          std::ostream &outf;
          bool ISFinal;
          bool jump;

          public:
          GRCsim(GRCgraph *top, Module *m, GRCNode::NumMap &cm,
                STNode::NumMap &sm, int db, std::ostream &outf)
                : debug(db), top(top), sigs(m->sigs),
                  module(m), cfgmap(cm), stmap(sm), outf(outf) {}

```

```
virtual ~GRCsim() {}

void visit (GRCNode *);

Status visit(EnterGRC &) { return Status(); }
Status visit(ExitGRC &) { return Status(); }
Status visit(Switch &);

Status visit(Test &);
Status visit(LoadSignalExpression &);
Status visit(CheckCounter &);
Status visit(BinaryOp &);
Status visit(UnaryOp &);
Status visit(LoadSignalValueExpression &);
Status visit(FunctionCall &);
Status visit(LoadVariableExpression &);
Status visit(Literal &);

Status visit(STexcl &) { return Status(); }
Status visit(STref &) { return Status(); }
Status visit(STpar &) { return Status(); }
Status visit(STleaf &) { return Status(); }

void dfs(GRCNode *n);
void init();

void schedule_child(GRCNode *,int);
void schedule(GRCNode *,int);
void clear_inputs();
void dotick();
void execute_max(int);
void execute_vectors(std::istream &);
int doswitch(STNode *n);

    <method declarations 2a>
};
```

3 Helper Methods

3.1 schedule and schedule_child

Schedule adds the given node to the “to be executed” map. `schedule_child` schedules a node’s child, if one exists.

```
16a <method definitions 2b>+≡
void GRCsim::schedule(GRCNode *n, int ctrlflw)
{
  assert(ctrlflw != 0);
  if (n){
    if (sched[n] != 1)
      sched[n] = ctrlflw;
  }
}

void GRCsim::schedule_child(GRCNode *n, int ctrlflw)
{
  assert(n);
  assert(n->successors.size() <= 1);
  if ( !n->successors.empty() )
    schedule(n->successors.front(),ctrlflw);
}
```

3.2 intVal

This evaluates what is expected to be an integer or Boolean expression and returns the result.

```
16b <method declarations 2a>+≡
int intVal(ASTNode *n)
{
  assert(n);
  Status s = n->welcome(*this);
  return s.i; // Integer-valued result
}
```


3.3 doswitch

Returns the number of the currently-active child of a Switch node in the selection tree.

```
17  <method definitions 2b>+≡
    int GRCsim::doswitch(STNode *n)
    {
        if ( (dynamic_cast<STexcl*>(n)) == NULL )
            cerr << "error: testing non-excl node\n";
        if ( state.find(n) == state.end() ){
            cerr << "error: testing uninitialized state for node " << stmap[n] << endl;
            assert(0);
        }

        assert(state[n] >= 0);
        assert(state[n] < (int) n->children.size());

        return state[n];
    }
```

3.4 dfs

Order the nodes in the control-flow graph.

FIXME: Not all constructive programs can be scheduled statically; some will need data-dependent dynamic scheduling.

```

18  <method definitions 2b>+≡
    void GRCsim::dfs(GRCNode *n){
        vector<GRCNode *>::iterator i;

        if (!n) return;

        if ( debug & debugDFS ) cerr << "visiting " << cfgmap[n] << "...";
        if ( dfs_notwhite.count(n) > 0 ) {
            if (debug & debugDFS ) cerr << "visited before\n";
            if (dfs_black.count(n) == 0) {
                cerr << "GRCsim: cycle detected\n";
                exit(100);
            }
            return;
        }
        dfs_notwhite.insert(n);

        if (jump) {
            if (debug & debugDFS ) cerr << "JUMP!\n";
            jump = false;
        }

        if ( debug & debugDFS ) cerr << "visiting\n";

        if ( debug & debugDFS ) cerr << "dfs children\n";
        for (i = n->successors.begin(); i != n->successors.end(); i++)
            dfs(*i);

        if (n->dataSuccessors.size()>0){
            if ( debug & debugDFS ) cerr << "dfs datadps children\n";
            for (i = n->dataSuccessors.begin();
                i != n->dataSuccessors.end(); i++){
                jump = true;
                dfs(*i);
                jump = true;
            }
        }

        assert(n);
        topo.push_back(n);
        dfs_black.insert(n);
        if ( debug & debugDFS ) cerr << "push back " << cfgmap[n] << "\n";
    }

```

3.5 execute max

A simulation dispatch method. Initialize the simulation and run the program for either the given number or ticks or until the program terminates, whichever is sooner.

```
19  <method definitions 2b>+≡
    void GRCsim::execute_max(int maxticks)
    {
        int ntick;

        useold=0;

        init();
        ntick=0;
        ISFinal = false;

        do {
            if (debug) cerr << "##### TICK " << ntick << endl;
            char buf[5];
            sprintf(buf, "%4d ", ntick);
            outf << buf;
            ntick++;
            clear_inputs();
            dotick();
        } while ((!ISFinal) && (maxticks < 0 || (ntick < maxticks)));
    }
```

3.6 Execute Vectors

A simulation dispatch method. Initialize the simulation and run the program, taking input signals presence/absence information from a test vector file.

```

20  <method definitions 2b>+≡
    void GRCsim::execute_vectors(std::istream &vf)
    {
        string line;

        // Enumerate the inputs to be read from the vector file

        vector<SignalSymbol*> inputs;

        for ( SymbolTable::const_iterator isym = sigs->begin() ;
              isym != sigs->end() ; isym++ ) {
            SignalSymbol *ss = dynamic_cast<SignalSymbol*>>(*isym);
            assert(ss);
            if ( ss->name != "tick" && (ss->kind == SignalSymbol::Input ||
                                         ss->kind == SignalSymbol::Inputoutput ) ) {
                if ( debug & debugVectors ) std::cout << ss->name << '\n';
                inputs.push_back(ss);
            }
        }

        init();

        int ntick = 0;
        ISFinal = false;
        do {
            getline(vf, line);
            if (vf.fail()) break;

            if ( line.size() < inputs.size() ) {
                cerr << "Not enough inputs ( " << line.size() << '<' << inputs.size()
                     << " ) in test vector file\n"
                     << "Got \"" << line << "\"\n";
                exit(-2);
            }

            clear_inputs();
            string::const_iterator j = line.begin();
            for ( vector<SignalSymbol*>::const_iterator i = inputs.begin() ;
                  i != inputs.end() ; j++ )
                if ((*j) != ' ') {
                    if ((*j) == '1')
                        signals[*i] = 1;
                    else if ((*j) == '0')
                        signals[*i] = 0;
                    else{
                        signals[*i] = -1; //unknown
                    }
                }
        } while (!ISFinal);
    }

```

```
        ternary = true; //FIXME: may be wrong
    }
    i++;
}

if (debug) cerr << "##### TICK " << ntick << " TV :"<<line<< endl;
char buf[5];
sprintf(buf, "%4d ", ntick);
outf << buf;
ntick++;

dotick();
} while (!ISFinal);
}
```

3.7 init

Initialize the simulation. Gather special nodes in the graph and run DFS to order them.

```

22  <method definitions 2b>+≡
    void GRCsim::init()
    {
        entergrc = dynamic_cast<EnterGRC*>(top->control_flow_graph);
        if (entergrc)
            grcroot = dynamic_cast<GRCNode*>(entergrc->successors.back());
        else
            grcroot = dynamic_cast<GRCNode*>(top->control_flow_graph);
        assert (grcroot);
        stroot = dynamic_cast<STexcl *>(top->selection_tree);
        assert(stroot);
        boot = dynamic_cast<STleaf* >(stroot->children.back());
        assert(boot);

        globalcc = 1;
        setState(boot,true);
        ternary = false;

        if ( debug & debugDFS ) cerr << "will dfs\n";
        jump = false;
        dfs(grcroot);
        if (debug & debugDFS ) cerr << "init ok\n";

        if (debug & debugDFS ){
            cerr << "-----topo-----:\n";
            for (vector<GRCNode*>::iterator i = topo.end()-1; i >= topo.begin(); i--)
                cerr << cfgmap[*i] << ' ';
            cerr << "\n";
        }

        // Initialize constants

        assert(module->constants);
        for ( SymbolTable::const_iterator i = module->constants->begin();
            i != module->constants->end() ; i++ ) {
            ConstantSymbol *cs = dynamic_cast<ConstantSymbol*>(*i);
            if (cs && cs->initializer) {
                assert(cs->type);
                if (cs->type->name != "integer" &&
                    cs->type->name != "boolean")
                    throw IR::Error("only integer and boolean constants are supported");
                var[cs] = intVal(cs->initializer);
            }
        }
    }
}

```

3.8 Clear Inputs

Reset all the signals; set the tick signal to be present.

```
23  <method definitions 2b>+≡
    void GRCsim::clear_inputs()
    {
      for( SymbolTable::const_iterator isym = sigs->begin() ; isym != sigs->end() ;
          isym++ ) {

        SignalSymbol *ss = dynamic_cast<SignalSymbol*>>(*isym);
        if (ss) {
          signals[ss] = (ss->name == "tick");
          if(ss->type && ss->type->name=="integer") signals_v[ss]=0;
        }
      }
    }
}
```

3.9 dotick

Simulate the program for a single cycle.

```

24  <method definitions 2b>+≡
    void GRCsim::dotick()
    {
        int tsz = topo.size();

        for (int i = 0 ; i < tsz ; i++) sched[topo[i]] = 0;

        sched[ topo[tsz-1] ] = 1;

        for ( int i = tsz - 1 ; i >= 0 ; i-- ) {
            assert(topo[i]);
            globalcc = 0;
            if ( sched[topo[i]] != 0 ){
                globalcc = sched[topo[i]];
                topo[i]->welcome(*this);
            }
        }

        for ( SymbolTable::const_iterator isym = sigs->begin() ;
              isym != sigs->end() ; isym++ ) {
            SignalSymbol *ss = dynamic_cast<SignalSymbol*>(*isym);
            assert(ss);
            if (debug) cerr << ss->name << " : " << signals[ss] << "\n";
            if ( ss->kind == SignalSymbol::Output ){
                outf << ss->name << "=" << signals[ss] << " ";
                if(ss->type && ss->type->name=="integer") outf<< "("<<signals_v[ss]<<") ";
            }
            else if ( ss->kind == SignalSymbol::Inputoutput )
                outf << ss->name << "_IO_0=" << (signals[ss] ? '1' : '0') << " ";
        }
        outf << "\n";

        if(!ternary){ //FIXME: other effects, like counter, also should
            // be taken back if ternary = true
            for(map<STNode *,int>::iterator it = nextstate.begin();
                it != nextstate.end(); it++){
                state[(*it).first] = (*it).second;
            }
            //nextstate.swap(state);
            nextstate.clear();
        }
        ternary = false;
    }
}

```


4 Top-level files

```
25a  <GRCSim.hpp 25a>≡
      #ifndef _GRCSIM_HPP
      #  define _GRCSIM_HPP

      #  include "IR.hpp"
      #  include "AST.hpp"
      #  include <iostream>
      #  include <stdlib.h>
      #  include <stdio.h>
      #  include <set>
      #  include <map>

      namespace AST {

      using std::set;
      using std::map;

      <GRCSim class 14>

      }
      #endif
```

```
25b  <GRCSim.cpp 25b>≡
      #include "GRCSim.hpp"

      using namespace std;

      namespace AST {
        <method definitions 2b>
      }
```

```

26  <cec-grcsim.cpp 26>≡
    #include "GRCsim.hpp"
    #include <fstream>

    using namespace std;

    struct UsageError {};

    int main(int argc, char *argv[])
    {
        try {
            int debug = 0;
            int maxticks = 0;
            string testvectorfile;

            ++argv, --argc;

            while (argc > 0 && argv[0][0] == '-') {
                switch (argv[0][1]) {
                    case 'd':
                        if (argc == 1) throw UsageError();
                        ++argv, --argc;
                        debug = atoi(argv[0]);
                        break;
                    case 'c':
                        if (argc == 1) throw UsageError();
                        ++argv, --argc;
                        maxticks = atoi(argv[0]);
                        break;
                    case 't':
                        if (argc == 1) throw UsageError();
                        ++argv, --argc;
                        testvectorfile = argv[0];
                        break;
                    default:
                        cerr << "Unrecognized option " << argv[0] << endl;
                        throw UsageError();
                }
                ++argv, --argc;
            }

            if ( argc != 0 ) throw UsageError();

            IR::XMListream r(std::cin);
            IR::Node *n;
            r >> n;

            AST::Modules *mods = dynamic_cast<AST::Modules*>(n);
            if (!mods) throw IR::Error("Root node is not a Modules object");

```

```

AST::Module *module = mods->modules.front();
assert(module);

AST::GRCgraph *gn = dynamic_cast<AST::GRCgraph*>(module->body);
assert(gn);

AST::GRCNode::NumMap cfgmap;
AST::STNode::NumMap stmap;

gn->enumerate(cfgmap, stmap);

AST::GRCsim simulator(gn, module, cfgmap, stmap, debug, std::cout);
if ( maxticks > 0 )
    simulator.execute_max(maxticks);
else if (!testvectorfile.empty()) {
    std::ifstream tvf(testvectorfile.c_str());
    if (tvf.bad()) {
        cerr << "Error opening test vector file " << testvectorfile << '\n';
        exit(-2);
    }
    simulator.execute_vectors(tvf);
    tvf.close();
} else throw UsageError();

} catch (IR::Error &e) {
    cerr << e.s << endl;
    exit(-1);
} catch (UsageError &) {
    cerr << "Usage: cec-grcsim [-d <level>] -c <cycles> | -t <vectorfile>\n"
        << "-d <level>  Enable debugging at the given level\n"
        << "-c <cycles>  Simulate with no inputs for this many cycles maximum\n"
        << "-t <vectorfile> Simulate taking inputs from the given vector file\n";
    exit(-1);
}

return 0;
}

```