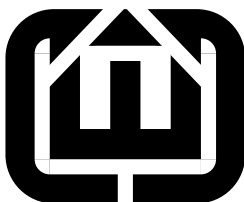


CEC GRC dot format printer



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

Contents

1	Overview	1
2	Node printers	1
2.1	Control-flow graph nodes	1
2.2	Selection Tree Nodes	6
3	Topmost files	9

1 Overview

2 Node printers

2.1 Control-flow graph nodes

The basic operation for a control-flow graph node is to create a new graph node, use the visitor to label it, draw its data dependencies, and draw arcs and recurse on its successors. Since it may be a DAG, this is done as a depth-first search with the `reached` set indicating which nodes have been visited.

```
1  <declarations 1>≡ (9) 3>
    void visit_cfg(GRCNode *);
```

```

2  <definitions 2>≡ (10) 4>
    void GRCDP::visit_cfg(GRCNode *n) {
        assert(n);

        if (reached.find(n) == reached.end()) {
            reached.insert(n);

            // Print a definition for the node
            assert(cfgnum.find(n) != cfgnum.end());

            mynum = cfgnum[n]; // used by most visitors
            o << 'n' << mynum << ' ';
            n->welcome(*this);

            // Draw data dependencies

            for (vector<GRCNode *>::const_iterator k = n->dataPredecessors.begin() ;
                k != n->dataPredecessors.end() ; k++) {
                assert(cfgnum.find(*k) != cfgnum.end());
                o << 'n' << cfgnum[*k] << " -> n" << cfgnum[n];
                if (clean) o << " [color=red]\n";
                else o << " [color=red constraint=false]\n";
            }

            // Draw control dependencies

            for ( vector<GRCNode*>::iterator j = n->successors.begin() ;
                j != n->successors.end() ; j++ ) {
                if (*j) {
                    o << 'n' << cfgnum[n] << " -> n" << cfgnum[*j];
                    if ( n->successors.size() > 1) {
                        if (clean) {
                            if (dynamic_cast<Switch*>(n) != NULL ||
                                dynamic_cast<Sync*>(n) != NULL)
                                o << " [label=\"\" << j - n->successors.begin() << "\"]";
                            else if (dynamic_cast<Test*>(n) != NULL &&
                                j == n->successors.end() - 1)
                                o << " [label=\"P\"]";
                        } else {
                            o << " [label=\"\" << j - n->successors.begin() << "\"]";
                        }
                    }
                    o << '\n';
                } else if (!clean) {
                    o << 'n' << cfgnum[n] << " -> n" << nextnum << "[label=\"\"
                        << j-n->successors.begin() << "\"]" << '\n';
                    o << 'n' << nextnum++
                        << " [shape=octagon style=filled color=black]\n";
                }
            }
        }
    }

```

```
// Visit control successors and predecessors

for ( vector<GRCNode*>::iterator j = n->successors.begin() ;
      j != n->successors.end() ; j++ )
    if (*j) visit_cfg(*j);

for ( vector<GRCNode*>::iterator j = n->predecessors.begin() ;
      j != n->predecessors.end() ; j++ ) visit_cfg(*j);

// Visit data successors and predecessors

for ( vector<GRCNode*>::iterator j = n->dataSuccessors.begin() ;
      j != n->dataSuccessors.end() ; j++ ) visit_cfg(*j);

for ( vector<GRCNode*>::iterator j = n->dataPredecessors.begin() ;
      j != n->dataPredecessors.end() ; j++ ) visit_cfg(*j);
}
}
```

```

3      (declarations 1)+≡
      Status visit(Switch &);
      Status visit(Test &);
      Status visit(Terminate &);
      Status visit(Sync &);
      Status visit(Fork &);
      Status visit(Action &);
      Status visit(Enter &);
      Status visit(STSuspend &);
      Status visit(EnterGRC &);
      Status visit(ExitGRC &);
      Status visit(Nop &);
      Status visit(DefineSignal &);

```

```

4  <definitions 2>+≡ (10) <2 7a>
    Status GRCDP::visit(Switch &s) {
        if (clean) {
            o << "[label=\"s\" << stnum[s.st]
              << "\"" shape=diamond peripheries=2]\n";
            o << "{ rank=same n\" << mynum << " n\" << stnum[s.st] << " }\n";
        } else {
            o << "[label=\"\" << mynum << " switch ";
            o << stnum[s.st]
              << "\"" shape=diamond color=pink style=filled]\n";
            drawSTlink(&s,s.st);
        }
        return Status();
    }

    Status GRCDP::visit(Test &s) {
        o << "[label=\"\";
        if (!clean) o << mynum << " test ";
        s.predicate->welcome(ep);
        o << "\"" shape=diamond]\n";
        return Status();
    }

    Status GRCDP::visit(STSuspend &s){
        o << "[label=\"\";
        if (!clean) o << mynum << " Suspend ";
        o << stnum[s.st]
          << "\"" shape=egg]\n";
        return Status();
    }

    Status GRCDP::visit(Terminate &s) {
        if (clean) {
            o << "[label=\"\" << s.code
              << "\"" shape=octagon]\n";
        } else {
            o << "[label=\"\" << mynum << ' ' << s.index << '@'
              << s.code
              << "\"" shape=octagon color=red style=filled "
                "fontcolor=white fontname=\"Times-Bold\"]\n";
        }
        return Status();
    }

    Status GRCDP::visit(Sync &s) {
        o << "[label=\"\";
        if (!clean) o << mynum << " sync\" << " \" << stnum[s.st];
        o << "\"" shape=invtriangle]\n";

        // Set all the predecessors (should be Terminates) at the same level

```

```

    o << "{ rank=same; ";
    for ( vector<GRNode*>::iterator i = s.predecessors.begin() ;
          i != s.predecessors.end() ; i++ )
        o << 'n' << cfname[*i] << " ";
    o << "}\n";
    return Status();
}

Status GRCDP::visit(Fork &s) {
    o << "[label=\"";
    if (!clean) o << mynum << " fork";
    o << "\" shape=triangle]\n";
    return Status();
}

Status GRCDP::visit(Action &s) {
    o << "[label=\"";
    if (!clean) o << mynum << " action ";
    s.body->welcome(ep);
    o << '\n';
    if (dynamic_cast<Emit*>(s.body))
        o << " shape=house orientation=270]\n";
    else
        o << " shape=box]\n";
    return Status();
}

Status GRCDP::visit(Enter &s) {
    if (clean) {
        // Calculate the child number

        STNode *n = s.st;
        STNode *parent = NULL;
        STexcl *exclusive = NULL;
        for (;;) {
            parent = n->parent;
            exclusive = dynamic_cast<STexcl*>(parent);
            if ( exclusive != NULL ) break;
            n = parent;
        }
        vector<STNode*>::iterator i = exclusive->children.begin();
        while (*i != n && i != exclusive->children.end()) i++;
        int childnum = i - exclusive->children.begin();

        o << "[label=\"s" << stnum[parent] << '=' << childnum << "\" shape=box]\n";
    } else {
        o << "[label=\"\" << mynum << " enter " << stnum[s.st]
          << "\" shape=house color=palegreen1 style=filled]\n";
    }
}

```

```

    return Status();
}

Status GRCDP::visit(EnterGRC &s){
    o << "[label=\"";
    if (!clean) o << mynum << " EnterGRC";
    o << "\"]\n";
    return Status();
}

Status GRCDP::visit(ExitGRC &s){
    o << "[label=\"";
    if (!clean) o << mynum << " ExitGRC";
    o << "\"]\n";
    return Status();
}

Status GRCDP::visit(Nop &s){
    o << "[label=\"";
    if (!clean) o << mynum << " ";
    if (s.isflowin()) o << "*"; else
        if (s.isshortcut()) o << "#";
        else o << "\n" << s.code;
    o << "\" shape=circle]\n";
    return Status();
}

Status GRCDP::visit(DefineSignal &s){
    o << "[label=\"";
    if (!clean) o << mynum << " DefS\\n";
    o << s.signal->name
        << "\" shape=house orientation=90]\n";
    return Status();
}

```

2.2 Selection Tree Nodes

The basic operation for a selection tree node is to print the node and its label using the visitor, then recurse on its children. This is a simple recursive walk because the selection tree is a tree.

```

6  <declarations 1>+≡ (9) <3 7b>
    void visit_st(STNode *);

```

7a $\langle \text{definitions } 2 \rangle + \equiv$ (10) $\triangleleft 4 \ 8 \triangleright$

```
void GRCDP::visit_st(STNode *n) {
    assert(n);

    mynum = stnum[n];
    o << 'n' << mynum << ' ';
    n->welcome(*this);

    // Visit children

    for ( vector<STNode*>::const_iterator i = n->children.begin() ;
          i != n->children.end() ; i++ )
        if (*i) {
            visit_st(*i);
            o << 'n' << stnum[n] << " -> n" << stnum[*i];
            if (!clean || dynamic_cast<STexcl*>(n) != NULL)
                o << " [label=\"\" << (i - n->children.begin()) << "\"]";
            o << '\n';
        } else {
            o << 'n' << stnum[n] << " -> n" << nextnum << "[label=\"\"
              << i - n->children.begin() << "\"]" << '\n';
            o << 'n' << nextnum++;
            if (clean) o << " [shape=point]\n";
            else o << " [shape=octagon style=filled color=black]\n";
        }
    }
}
```

7b $\langle \text{declarations } 1 \rangle + \equiv$ (9) $\triangleleft 6$

```
Status visit(STexcl &);
Status visit(STref &);
Status visit(STpar &);
Status visit(STleaf &);
```

8 $\langle definitions\ 2 \rangle + \equiv$ (10) $\triangleleft 7a$

```

Status GRCDP::visit(STexcl &s) {
    if (clean) {
        o << "[label=\"s\" << mynum << \"\" shape=diamond peripheries=2]\n";
    } else {
        o << "[label=\"\" << mynum << \"\" shape=diamond color=pink style=filled]\n";
    }
    return Status();
}

Status GRCDP::visit(STref &s) {
    if (clean) {
        o << "[shape=box label=\"\"]\n";
    } else {
        o << "[label=\"\" << mynum << \" \";
        if(s.isabort()) o << "A";
        if(s.issuspend()) o << "S";
        o << "\" ]\n";
    }
    return Status();
}

Status GRCDP::visit(STpar &s) {
    if (clean) {
        o << "[label=\"\" shape=triangle]\n";
    } else {
        o << "[label=\"\" << mynum << \"\" shape=triangle]\n";
    }
    return Status();
}

Status GRCDP::visit(STleaf &s) {
    if (clean) {
        o << "[label=\"\"";
        if(s.isfinal()) o << "*";
        o << "\" shape=box]\n";
    } else {
        o << "[label=\"\" << mynum << \" \";
        if(s.isfinal()) o << "*";
        o << "\" shape=box]\n";
    }
    return Status();
}

```


3 Topmost files

```

9  <GRCPrinter.hpp 9>≡
    #ifndef _GRC_PRINTER_HPP
    #   define _GRC_PRINTER_HPP
    #   include "AST.hpp"
    #   include "EsterelPrinter.hpp"
    #   include <iostream>
    #   include <map>
    #   include <set>

    namespace GRCDot {
        using namespace AST;
        using std::map;
        using std::set;

        typedef map<GRCNode *, int> CFGmap;
        typedef map<STNode *, int> STmap;
        void GRCDot(std::ostream &, GRCgraph *, Module *, bool, bool);
        int GRCDot(std::ostream &o, GRCgraph *g, Module *m, bool drawstlink,
                   bool clean, CFGmap &cfgmap, STmap &stmap, int mxnode);

        class GRCDP : public Visitor {
            std::ostream &o;
            CFGmap &cfgnum; // Node numbers for control-flow graph
            STmap &stnum; // Node numbers for selection tree
            set<GRCNode *> reached; // Used during DFS of CFG

            int nextnum;
            int mynum;

            EsterelPrinter ep;
        public:
            GRCDP(std::ostream &oo, CFGmap &cm, STmap &sm, int nextnum) :
                o(oo), cfgnum(cm), stnum(sm), nextnum(nextnum), ep(oo),
                drawstlink(false), clean(false) {}

            virtual ~GRCDP() {}

            // Output style flags

            bool drawstlink;
            bool clean;

            <declarations 1>

            void drawSTlink(GRCNode *, STNode *);
        };
    };
#endif

```

```

10  <GRCPrinter.cpp 10>≡
    #include "GRCPrinter.hpp"
    #include <cassert>

    namespace GRCDot {

        <definitions 2>

        void drawDot(std::ostream &o, GRCgraph *g, Module *m, bool drawstlink,
                     bool clean, CFGmap &cfgmap, STmap &stmap, int mxnode)
        {
            GRCDP visitor(o, cfgmap, stmap, mxnode+1);
            visitor.drawstlink = drawstlink;
            visitor.clean = clean;

            o << "digraph " << m->symbol->name << " {" << std::endl;
            o << "size=\"7.5,10\"\\n";

            visitor.visit_st(g->selection_tree);
            visitor.visit_cfg(g->control_flow_graph);

            o << "}" << std::endl;
        }

        int GRCDot(std::ostream &o, GRCgraph *g, Module *m, bool drawstlink,
                   bool clean, CFGmap &cfgmap, STmap &stmap, int mxnode)
        {
            assert(g);
            assert(m);
            assert(m->symbol);
            mxnode = g->enumerate(cfgmap, stmap, mxnode);
            drawDot(o, g, m, drawstlink, clean, cfgmap, stmap, mxnode);
            return mxnode;
        }

        void GRCDot(std::ostream &o, GRCgraph *g, Module *m, bool drawstlink,
                   bool clean)
        {
            assert(g);
            assert(m);
            assert(m->symbol);

            CFGmap cfgmap;
            STmap stmap;

            int mxnode = g->enumerate(cfgmap, stmap);
            drawDot(o, g, m, drawstlink, clean, cfgmap, stmap, mxnode);
        }
    }

```

```
void GRCDP::drawSTlink(GRCNode *g, STNode *s)
{
    o << "{ rank=same; n" << cfignum[g] << "; n" << stnum[s] << " }\n";
    if (!drawstlink) return;

    assert( stnum.find(s) != stnum.end() );

    o << 'n' << cfignum[g] << " -> n" << stnum[s];
    o << "[color=blue constraint=false]";
    o << '\n';
}
}
```

```

12  <cec-grcdot.cpp 12>≡
    #include "IR.hpp"
    #include "AST.hpp"
    #include "GRCPrinter.hpp"
    #include <iostream>
    #include <stdlib.h>

    struct UsageError {};

    int main(int argc, char *argv[])
    {
        try {

            bool clean = false;
            bool stlink = false;

            --argc; ++argv;

            while (argc > 0 && argv[0][0] == '-') {
                switch (argv[0][1]) {
                    case 'c': clean = true; break;
                    case 'l': stlink = true; break;
                    default:
                        std::cerr << "unrecognized option \"" << argv[0] << "\"\n";
                        /* FALLTHROUGH */
                    case 'h':
                        throw UsageError();
                }
                --argc; ++argv;
            }

            if (argc > 0) throw UsageError();

            IR::XMLListstream 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");

            for ( std::vector<AST::Module*>::iterator i = mods->modules.begin() ;
                  i != mods->modules.end() ; i++ ) {
                assert(*i);

                AST::GRCgraph *g = dynamic_cast<AST::GRCgraph*>((*i)->body);
                if (!g) throw IR::Error("Module is not in GRC format");

                GRCDot::GRCDot(std::cout, g, *i, stlink, clean);
            }
        } catch (IR::Error &e) {

```

```
std::cerr << e.s << std::endl;
exit(-1);
} catch (UsageError &) {
std::cerr <<
    "Usage: cec-grcdot [-c] [-s] [-h]\n"
    "-c   Print a cleaner version of the graph\n"
    "-l   Draw links to the ST graph\n"
    "-h   Print this usage message\n"
    ;
return 1;

}
return 0;
}
```