May 19, 2018 at 02:30

1. Intro. This program reads in a gate graph (in the standard Stanford GraphBase format as defined in GB_GATES) and converts it a similar graph in which every vertex corresponds to a “wire” in a physical network for the original circuit. Informally, the wires correspond to the parts of a circuit that can be “stuck” or not. The main difference is that each wire is the input to at most one subsequent gate, except for “fanout branch” wires; the latter are inputs to exactly two subsequent vertices called “clones.”

A clone vertex v has $v\to typ$ = `'F'`. (`F` stands for fanout; the letter `C` was already taken, to stand for a constant gate.) It has one outgoing arc, which points to its parent. The two clones with the same parent are consecutive. All clones descended from a normal gate appear immediately following that gate. (Thus the n inputs to the circuit, which formerly were the first n gates, are now interspersed with their clones.)

Every wire has a “top” and a “bottom.” The top is either external (namely one of the n inputs), or the output of a previous ordinary gate, or the implicit junction for two clones. The bottom is either external (namely one of the m outputs), or an input to a subsequent ordinary gate, or an input to an implicit junction.

Thus the number of vertices is n, plus the number A of ordinary gates, plus $2B$ for the clones, where B is the number of implicit junctions. This counts the wires by the number of tops of wires.

We can also count them by the number of bottoms. Then we conclude that the number of vertices is m, plus $2A$ for the inputs to ordinary gates (if we assume that they all have exactly two inputs), plus B. Equating these two counts gives $m + A = n + B$, although the relationship will be different if ordinary gates aren’t all binary. (The general formula is $m + \Sigma A - A = n + B$, where ΣA is the sum of in-degrees of all ordinary gates. Notice that $A + n = g\to n$ and $\Sigma A = g\to m$ in terms of the input graph; hence we’re adding $2B = 2(m + g\to m - g\to n)$ clone vertices to that graph.)

No two outputs can share the same wire, nor can a wire be associated with more than one input port to a gate or a junction.

This program also converts a sequential circuit to a combinational circuit, by changing all latches ($v\to typ$ = `'L'`) to inputs ($v\to typ$ = `'I'`) , $\wedge adding them to the list of outputs . Names of the input \wedge output files must be given on the comma$
line . include "gb$_g$raph.h"include"gb$_g$ates.h"include"gb$_s$ave.h"$FILE*in_file, *out_file; charbuf[1000]; main(intargc, cha$
$argv[])registerinti, j, k, n, r, s; registerVertex * u, *v, *w; registerArc * a, *b, *aa; registerGraph * g, *gg;$

```
⟨ Process the command line 2 ⟩ +≡
  ;
  ⟨ Compute out-degrees and prepend all latches to the list of outputs 3 ⟩;
  ⟨ Create the new graph, gg 4 ⟩;
  strcpy(gg→util_types, g→util_types);
  s = save_graph(gg, argv[2]);
  if (s) fprintf(stderr, "Output␣written␣to␣%s␣(anomalies␣%x!)\n", argv[2], s);
  else fprintf(stderr, "Output␣written␣to␣%s\n", argv[2]);
  }
```

2.

```
⟨ Process the command line 2 ⟩ ≡
  if (argc ≠ 3) {
    fprintf(stderr, "Usage:␣%s␣gates-in.gb␣wires-out.gb\n", argv[0]);
    exit(−1);
  }
  g = restore_graph(argv[1]);
  if (¬g) {
    fprintf(stderr, "I␣can't␣restore␣the␣graph␣`%s'!\n", argv[1]);
    exit(−2);
  }
```

This code is used in section 1.

3.
```
#define deg u.I    /* utility field of a vertex */
⟨ Compute out-degrees and prepend all latches to the list of outputs 3 ⟩ ≡
  for (k = 0, v = g→vertices; v < g→vertices + g→n; v++) {
    v→deg = 0;
    if (v→typ ≡ 'L') {
      v→typ = 'I';
      a = gb_virgin_arc( );
      a→next = g→outs;
      a→tip = v→alt;
      g→outs = a;
    }
    else if (v→typ ≠ 'I') {
      for (a = v→arcs; a; a = a→next) k++, a→tip→deg++;
    }
  }
  for (a = g→outs; a; a = a→next) k++, a→tip→deg++;
```

This code is used in section 1.

4.
```
⟨ Create the new graph, gg 4 ⟩ ≡
  gg = gb_new_graph(2 * k − g→n);
  if (¬gg) {
    fprintf(stderr, "I␣couldn't␣create␣a␣new␣graph!\n");
    exit(−99);
  }
  make_compound_id(gg, "", g, "|wires");
  for (u = g→vertices, v = gg→vertices; u < g→vertices + g→n; u++) {
    v→name = gb_save_string(u→name);
    v→typ = u→typ;
    u→alt = v;
    for (a = u→arcs; a; a = a→next) {
      w = a→tip→alt;
      a→tip→alt++;
      gb_new_arc(v, w, a→len);
    }
    k = (u→deg) − 1, v++;
    for (j = 0; j < k; j++) {
      sprintf(buf, "%s#%d", u→name, 2 * j + 1);
      v→name = gb_save_string(buf);
      v→typ = 'F';
      gb_new_arc(v, u→alt, 0);
      v++;
      sprintf(buf, "%s#%d", u→name, 2 * j + 2);
      v→name = gb_save_string(buf);
      v→typ = 'F';
      gb_new_arc(v, u→alt, 0);
      v++;
      u→alt++;
    }
  }
  ⟨ Create gg→outs 5 ⟩;
```

This code is used in section 1.

5. First we reverse (and destroy) the output list of g. Then we translate it into the corresponding vertices of gg, reversing it again in the process.

```
⟨ Create gg→outs 5 ⟩ ≡
  for (b = Λ, a = g→outs; a; a = aa) {
    aa = a→next;
    a→next = b;
    b = a;
  }
  for ( ; b; b = b→next) {
    a = gb_virgin_arc( );
    w = b→tip;
    a→tip = w→alt;
    w→alt++;
    a→next = gg→outs;
    gg→outs = a;
  }
```

This code is used in section 4.

6. Index.

aa: 1, 5.
adding: 1.
alt: 3, 4, 5.
Arc: 1.
arcs: 3, 4.
argc: 1, 2.
argv: 1, 2.
be: 1.
buf: 1, 4.
command: 1.
deg: 3, 4.
exit: 2, 4.
files: 1.
fprintf: 1, 2, 4.
gb_new_arc: 4.
gb_new_graph: 4.
gb_save_string: 4.
gb_virgin_arc: 3, 5.
gg: 1, 4, 5.
given: 1.
Graph: 1.
in_file: 1.
input: 1.
len: 4.
list: 1.
main: 1.
make_compound_id: 4.
must: 1.
name: 4.
Names: 1.
next: 3, 4, 5.
of: 1.
on: 1.
out_file: 1.
output: 1.
outputs: 1.
outs: 3, 5.
restore_graph: 2.
save_graph: 1.
sprintf: 4.
stderr: 1, 2, 4.
strcpy: 1.
the: 1.
them: 1.
tip: 3, 4, 5.
to: 1.
typ: 1, 3, 4.
util_types: 1.
Vertex: 1.
vertices: 3, 4.

⟨ Compute out-degrees and prepend all latches to the list of outputs 3 ⟩ Used in section 1.
⟨ Create the new graph, *gg* 4 ⟩ Used in section 1.
⟨ Create *gg*→*outs* 5 ⟩ Used in section 4.
⟨ Process the command line 2 ⟩ Used in section 1.

GATES-TO-WIRES

	Section	Page
Intro	1	1
Index	6	4