§1 SAT-TSEYTIN INTRO 1

1. Intro. Given a graph this program produces the unsatisfiable SAT problem in Tseytin's classic paper about lower bounds for regular resolution. The output is suitable for input to SAT0, SAT1, etc.

```
#include "gb_graph.h"
                                 /* we use the GB_GRAPH data structures */
#include "gb_save.h"
                                /* and input the graph in the usual way */
  char bit[1000];
  int main(int argc, char *argv[])
     register d, k, j;
     register Graph *g;
     register Vertex *u, *v;
     register Arc *a;
     \langle \text{Process the command line } 2 \rangle;
     \langle Generate the clauses 3 \rangle;
2. \langle \text{Process the command line } 2 \rangle \equiv
  if (argc \neq 2) {
    fprintf(stderr, "Usage: \_\%s\_foo.gb\n", argv[0]);
     exit(-1);
  }
  g = restore\_graph(argv[1]);
  if (\neg g) {
    fprintf(stderr, "I_{\square}couldn't_{\square}reconstruct_{\square}graph_{\square}%s! \n", argv[1]);
     exit(-2);
  printf("\"argv[1]);
This code is used in section 1.
```

**3.** There's one variable in the SAT program for each edge of g. So we call that variable u.v, when the edge runs from u to v. For each vertex v we generate  $2^{d-1}$  clauses, where d is the degree of v; each of those clauses involves the d variables adjacent to v. There's one clause for each way to complement an odd number of literals, except that we complement an even number when v is the very first vertex.

```
 \langle \text{ Generate the clauses } 3 \rangle \equiv \\ \text{ for } (v = g \text{-}vertices; \ v < g \text{-}vertices + g \text{-}n; \ v \text{++}) \ \{ \\ \text{ for } (d = -1, a = v \text{-}arcs; \ a; \ a = a \text{-}next) \ d \text{++}; \\ \text{ while } (1) \ \{ \\ \quad \langle \text{ Generate a clause for the current } bit \text{ setting } 4 \rangle; \\ \text{ for } (k = 0; \ bit[k]; \ k \text{++}) \ bit[k] = 0; \\ \text{ if } (k \equiv d) \ \text{ break}; \\ bit[k] = 1; \\ \} \\ \}
```

This code is used in section 1.

2 INTRO SAT-TSEYTIN §4

```
4. ⟨Generate a clause for the current bit setting 4⟩ ≡
for (j = (v > g¬vertices), k = 0, a = v¬arcs; a; a = a¬next, j ⊕= bit[k], k++) {
    printf("□");
    if (k ≡ d) ⟨Adjust the parity of the final literal 5⟩
    else if (bit[k]) printf("~");
    u = a¬tip;
    if (u < v) printf("%s.%s", u¬name, v¬name);
    else printf("%s.%s", v¬name, u¬name);
}
    printf("\n");
This code is used in section 3.</li>
5. ⟨Adjust the parity of the final literal 5⟩ ≡
    {
        if (j) printf("~");
    }

This code is used in section 4.
```

 $\S 6$  Sat-Tseytin index 3

## 6. Index.

a: 1.

Arc: 1.

arcs: 3, 4.

argc: 1, 2.

argv: 1, 2.

bit: 1, 3, 4.

d: 1.

exit: 2.

fprintf: 2.

g: 1.

Graph: 1.

j: 1.

k: 1.

main: 1.

name: 4.

next: 3, 4.

printf: 2, 4, 5.

restore\_graph: 2.

stderr: 2.

tip: 4.

 $v: \underline{1}.$  Vertex: 1. vertices: 3, 4.

u:  $\underline{1}$ .

4 NAMES OF THE SECTIONS SAT-TSEYTIN

```
 \begin{array}{lll} \langle \mbox{ Adjust the parity of the final literal 5} \rangle & \mbox{Used in section 4.} \\ \langle \mbox{ Generate a clause for the current $bit$ setting 4} \rangle & \mbox{Used in section 3.} \\ \langle \mbox{ Generate the clauses 3} \rangle & \mbox{Used in section 1.} \\ \langle \mbox{ Process the command line 2} \rangle & \mbox{ Used in section 1.} \\ \end{array}
```

## SAT-TSEYTIN

	Section	Page
Intro	 1	1
Index	 6	3