TRUTH(v) = 0, set TRUTH(v) ← 1, S_v ← v, and s ← s + 1. Otherwise, for 1 ≤ j ≤ k, create a hypothesis record h and set CLAUSE(h) ← c, PREV(h) ← LAST(u_j), LAST(u_j) ← h.

C2. [Prepare to assert p.] Terminate the algorithm if s = 0; the desired core now consists of all propositions whose TRUTH has been set to 1. Otherwise set s ← s - 1, p ← S_s, and h ← LAST(p).

C3. [Done with hypotheses?] If h = A, return to C2.

C4. [Validate h.] Set c ← CLAUSE(h) and COUNT(c) ← COUNT(c) - 1. If the new value of COUNT(c) is still nonzero, go to step C6.

C5. [Deduce CONCLUSION(c).] Set p ← CONCLUSION(c). If TRUTH(p) = 0, set TRUTH(p) ← 1, S_p ← p, s ← s + 1.

C6. [Loop on h.] Set h ← PREV(h) and return to C3.

Notice how smoothly the data structures work together, avoiding any need to search for a place to make progress in the calculation. Algorithm C is similar in many respects to Algorithm 2.2.3T (topological sorting), which was the first example of multilinked data structures that we discussed long ago in Chapter 2; in fact, we can regard Algorithm 2.2.3T as the special case of Algorithm C in which every proposition appears on the right-hand side of exactly one clause. (See exercise 47.)

Exercise 48 shows that a slight modification of Algorithm C solves the satisfiability problem for Horn clauses in general. Further discussion can be found in a paper by W. F. Dowling and J. H. Gallier, J. Logic Programming 1 (1984), 267–284.

We turn now to Krom functions and the 2SAT problem. Again there’s a linear-time algorithm; but again, we can probably appreciate it best if we look first at a simplified-but-practical application. Let’s suppose that seven comedians have each agreed to do one-night standup gigs at two of five hotels during a three-day festival, but each of them is available for only two of those days because of other commitments:

Tomlin should do Aladdin and Caesar on days 1 and 2; Unwin should do Bellagio and Excalibur on days 1 and 2; Vegas should do Desert and Excalibur on days 2 and 3; Williams should do Aladdin and Desert on days 1 and 3; Xie should do Caesar and Excalibur on days 1 and 3; Yankovic should do Bellagio and Desert on days 2 and 3; Zany should do Bellagio and Caesar on days 1 and 2.

(37)

Is it possible to schedule them all without conflict? (38)

To solve this problem, we can introduce seven Boolean variables \{t, u, v, w, x, y, z\}, where t (for example) means that Tomlin should do Aladdin on day 1 and Caesar on day 2 while f means that the days booked in those hotels occur in the opposite order. Then we can set up constraints to ensure that no two comedians get booked in hotels on the same day:

\[
\neg(t \land w) \quad [A1] \\
\neg(y \land z) \quad [B2] \\
\neg(t \land z) \quad [C2] \\
\neg(u \land y) \quad [D3] \\
\neg(u \land z) \quad [B1] \\
\neg(f \land z) \quad [C1] \\
\neg(u \land y) \quad [D2] \\
\neg(u \land y) \quad [E1] \\
\neg(f \land y) \quad [B2] \\
\neg(f \land z) \quad [C1] \\
\neg(u \land w) \quad [D3] \\
\neg(u \land y) \quad [E2] \\
\neg(f \land w) \quad [B3] \\
\neg(f \land z) \quad [C1] \\
\neg(u \land y) \quad [D3] \\
\neg(v \land x) \quad [E3]
\]

Each of these constraints is, of course, a Krom clause; we must satisfy

\[
(f \lor w) \land (y \lor z) \land (u \lor y) \land (u \lor z) \land (y \lor z) \land (f \lor z) \land (f \lor y) \land (f \lor x)
\]

(39)

Furthermore, Krom clauses (like Horn clauses) can be written as implications:

\[
t \Rightarrow w, \quad u \Rightarrow f, \quad t \Rightarrow z, \quad y \Rightarrow x, \quad f \Rightarrow z, \quad f \Rightarrow y, \quad x \Rightarrow z, \\
u \Rightarrow f, \quad u \Rightarrow y, \quad z \Rightarrow u, \quad w \Rightarrow f, \quad y \Rightarrow z, \quad u \Rightarrow v, \quad v \Rightarrow u.
\]

(40)

And every such implication also has an alternative, "contrapositive" form:

\[
w \Rightarrow f, \quad z \Rightarrow u, \quad y \Rightarrow u, \quad f \Rightarrow u, \quad y \Rightarrow z, \quad x \Rightarrow f, \\
z \Rightarrow f, \quad g \Rightarrow v, \quad w \Rightarrow v, \quad y \Rightarrow v, \quad y \Rightarrow w, \quad z \Rightarrow u, \quad v \Rightarrow f.
\]

(41)

But oops — alas — there is a vicious cycle,

\[
u \Rightarrow z \Rightarrow g \Rightarrow v \Rightarrow u \Rightarrow z \Rightarrow f \Rightarrow x \Rightarrow u
\]

(42)

This cycle tells us that u and v must both have the same value; so there is no way to accommodate all of the conditions in (37). The festival organizers will have to renegotiate their agreement with at least one of the six comedians \{t, u, v, x, y, z\}, if a viable schedule is to be achieved. (See exercise 53.)

The organizers might, for instance, try to leave v out of the picture temporarily. Then five of the sixteen constraints in (38) would go away and only 22 of the implications from (40) and (41) would remain, leaving the directed graph illustrated in Fig. 6. This digraph does contain cycles, like z ⇒ u ⇒ x ⇒ z and t ⇒ f ⇒ t; but no cycle contains both a variable and its complement. Indeed...