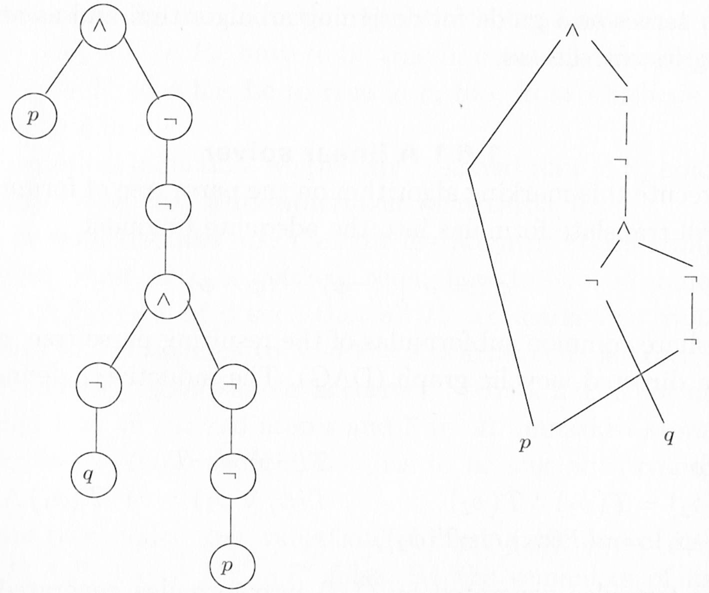


**Figure 1.12.** Parse tree (left) and directed acyclic graph (right) of the formula from Example 1.48. The $p$-node is shared on the right.

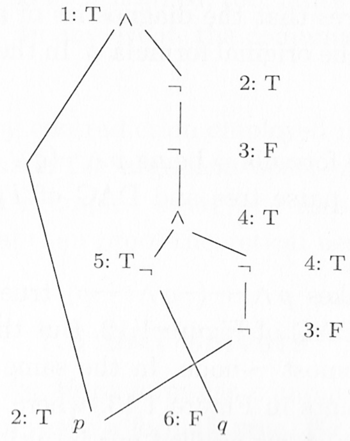
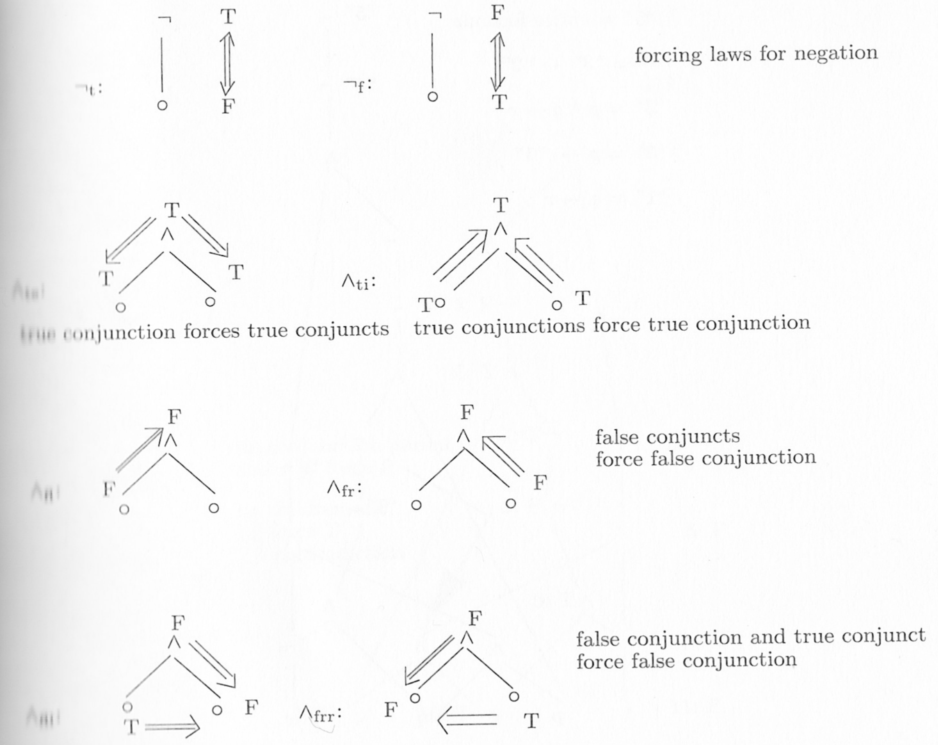


**Figure 1.13.** A witness to the satisfiability of the formula represented by this DAG.

and $\wedge_{\mathrm{frr}}$ are more complex: if an $\wedge$-node has a F constraint and one of it sub-nodes has a T constraint, then the *other* sub-node obtains a F-constraint Please check that all constraints depicted in Figure 1.13 are derivable from these rules. The fact that each node in a DAG obtained a forced markin



¬t:
¬f:
forcing laws for negation
∧ti:
true conjunction forces true conjuncts
true conjunctions force true conjunction
∧fr:
false conjuncts force false conjunction
∧frr:
false conjunction and true conjunct force false conjunction

**Figure 1.14.** Rules for flow of constraints in a formula's DAG. Small circles indicate arbitrary nodes ($\neg$, $\wedge$ or atom). Note that the rules $\wedge_{\mathrm{fll}}$, $\wedge_{\mathrm{frr}}$ and $\wedge_{\mathrm{ti}}$ require that the source constraints of both $\Longrightarrow$ are present.

represented by this DAG. A post-processing phase takes the marks for all atoms and re-computes marks of all other nodes in a bottom-up manner, as done in Section 1.4 on parse trees. Only if the resulting marks match the ones we computed have we found a witness. Please verify that this is the case in Figure 1.13.

We can apply SAT solvers to checking whether sequents are valid. For example, the sequent $p \wedge q \to r \vdash p \to q \to r$ is valid iff $(p \wedge q \to r) \to p \to q \to r$ is a theorem (why?) iff $\phi = \neg((p \wedge q \to r) \to p \to q \to r)$ is *not* satisfiable. The DAG of $T(\phi)$ is depicted in Figure 1.15. The annotations "1" etc indicate which nodes represent which sub-formulas. Notice that such DAGs may be constructed by applying the translation clauses for $T$ to sub-formulas in a bottom-up manner – sharing equal subgraphs were applicable.

The findings of our SAT solver can be seen in Figure 1.16. The solver concludes that the indicated node requires the marks T *and* F for (1.9) to be met. Such contradictory constraints therefore imply that all formulas $T(\phi)$

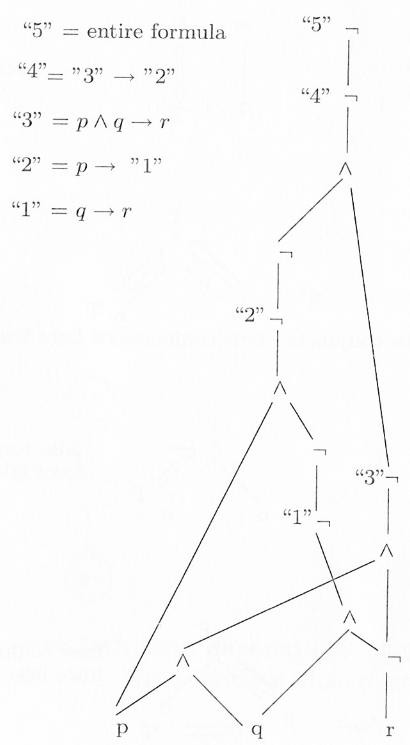


**Figure 1.15.** The DAG for the translation of $\neg((p \wedge q \rightarrow r) \rightarrow p \rightarrow q \rightarrow r)$. Labels "1" etc indicate which nodes represent what subformulas.

such $\phi$ are unsatisfiable. This SAT solver has a linear running time in the size of the DAG for $T(\phi)$. Since that size is a linear function of the length of $\phi$ – the translation $T$ causes only a linear blow-up – our SAT solver has a linear running time in the length of the formula. This linearity came with a price: our linear solver fails for all formulas of the form $\neg(\phi_1 \wedge \phi_2)$.

## 1.6.2 A cubic solver

When we applied our linear SAT solver, we saw two possible outcomes: we either detected contradictory constraints, meaning that no formula represented by the DAG is satisfiable (e.g. Fig. 1.16); or we managed to force consistent constraints on all nodes, in which case all formulas represented by this DAG are satisfiable with those constraints as a witness (e.g. Fig. 1.13). Unfortunately, there is a third possibility: all forced constraints are consistent with each other, but not all nodes are constrained! We already remarked

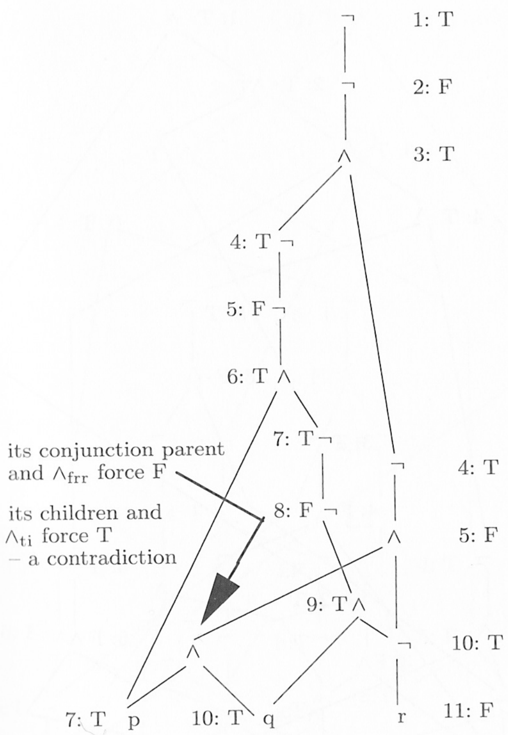

¬ 1: T
¬ 2: F
∧ 3: T
4: T ¬
5: F ¬
6: T ∧
7: T ¬
8: F ¬
¬ 4: T
∧ 5: F
9: T ∧
¬ 10: T
its conjunction parent and $\wedge_{\mathrm{frr}}$ force F
its children and $\wedge_{\mathrm{ti}}$ force T – a contradiction
∧
7: T p
10: T q
r 11: F

**Figure 1.16.** The forcing rules, applied to the DAG of Figure 1.15, detect contradictory constraints at the indicated node – implying that the initial constraint '1:T' cannot be realized. Thus, formulas represented by this DAG are not satisfiable.

Recall that checking validity of formulas in CNF is very easy. We already hinted at the fact that checking satisfiability of formulas in CNF is hard. To illustrate, consider the formula

$$((p \vee (q \vee r)) \wedge ((p \vee \neg q) \wedge ((q \vee \neg r) \wedge ((r \vee \neg p) \wedge (\neg p \vee (\neg q \vee \neg r)))))) \tag{1.11}$$

in CNF – based on Example 4.2, page 77, in [Pap94]. Intuitively, this formula should not be satisfiable. The first and last clause in (1.11) 'say' that at least one of $p$, $q$, and $r$ are false and true (respectively). The remaining three clauses, in their conjunction, 'say' that $p$, $q$, and $r$ all have the same truth value. This cannot be satisfiable, and a good SAT solver should discover this without any user intervention. Unfortunately, our linear SAT solver can neither detect inconsistent constraints nor compute constraints for all nodes.

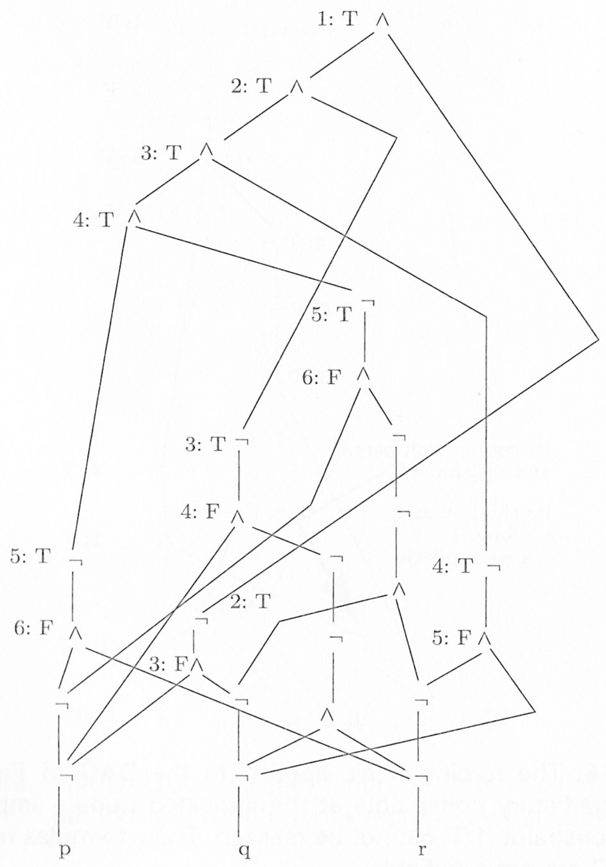


**Figure 1.17.** The DAG for the translation of the formula in (1.11). It has a $\wedge$-spine of length 4 as it is a conjunction of five clauses. Its linear analysis gets stuck: all forced constraints are consistent with each other but several nodes, including all atoms, are unconstrained.

that our SAT solver got stuck: no inconsistent constraints were found and not all nodes obtained constraints; in particular, no atom received a mark! So how can we improve this analysis? Well, we can mimic the role of LEM to improve the precision of our SAT solver. For the DAG with marks as in Figure 1.17, pick any node $n$ that is not yet marked. Then *test* node $n$ by making two independent computations:

1. determine which *temporary* marks are forced by adding to the marks in Figure 1.17 the T mark only to $n$; and
2. determine which *temporary* marks are forced by adding, again to the marks in

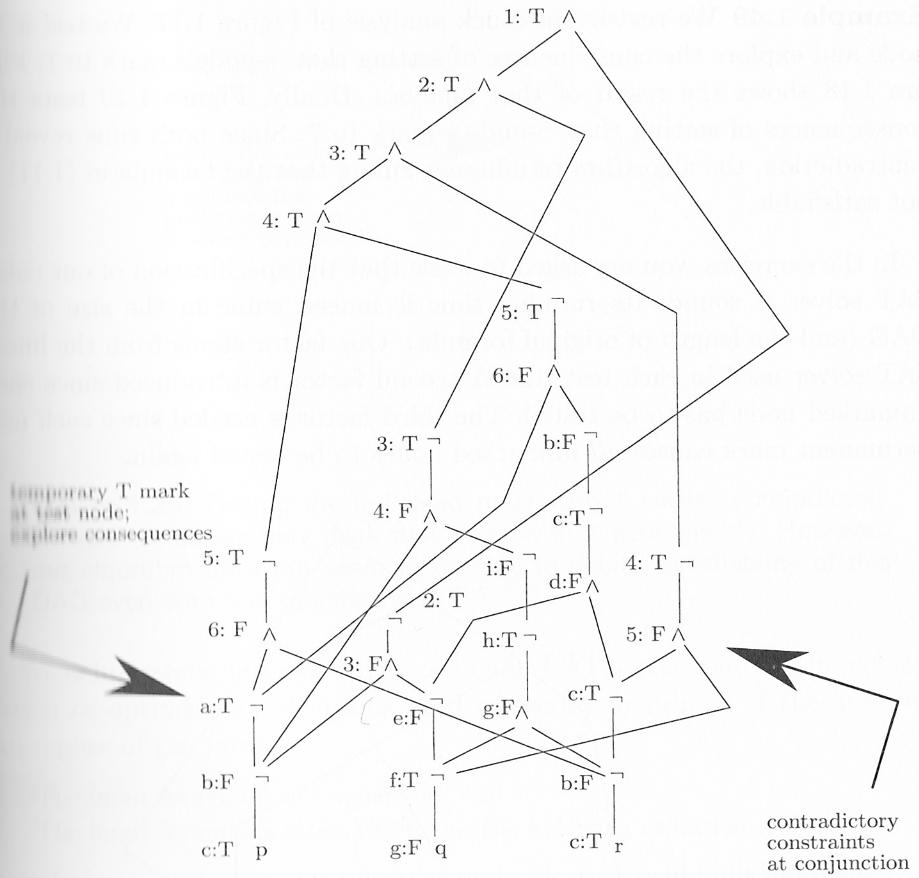


**Figure 1.18.** Marking an unmarked node with T and exploring what new constraints would follow from this. The analysis shows that this test marking causes contradictory constraints. We use lowercase letters 'a' etc to denote temporary marks.

If both runs find contradictory constraints, the algorithm stops and reports that $T(\phi)$ is unsatisfiable. Otherwise, all nodes that received the same mark in both of these runs receive that very mark as a *permanent* one; that is, we update the mark state of Figure 1.17 with all such shared marks.

We test any further unmarked nodes in the same manner until we either find contradictory *permanent* marks, a complete witness to satisfiability (all nodes have consistent marks), or we have tested *all* currently unmarked nodes in this manner without detecting any shared marks. Only in the latter case does the analysis terminate without knowing whether the formulas