in which ϕ is true. By our induction hypothesis, we know that all P_j and therefore $P_1 \wedge P_2 \wedge \cdots \wedge P_{k_i}$ have to be true in v as well. The conjunct $P_1 \wedge P_2 \wedge \cdots \wedge P_{k_i} \rightarrow P'$ of ϕ has be to true in v, too, from which we infer that P' has to be true in v.

By mathematical induction, we therefore secured that (1.8) holds no matter how many cycles that while-statement went through.

Finally, we need to make sure that the if-statement above always renders correct replies. First, if $\bot$ is marked, then there has to be some conjunct $P_1 \wedge P_2 \wedge \cdots \wedge P_{k_i} \rightarrow \bot$ of ϕ such that all P_i are marked as well. By (1.8) that conjunct of ϕ evaluates to $\mathtt{T} \rightarrow \mathtt{F} = \mathtt{F}$ whenever ϕ is true. As this is impossible the reply 'unsatisfiable' is correct. Second, if $\bot$ is not marked, we simply assign $\mathtt{T}$ to all marked atoms and $\mathtt{F}$ to all unmarked atoms and use proof by contradiction to show that ϕ has to be true with respect to that valuation.

If ϕ is *not* true under that valuation, it must make one of its principal conjuncts $P_1 \wedge P_2 \wedge \cdots \wedge P_{k_i} \rightarrow P'$ false. By the semantics of implication this can only mean that all P_j are true and P' is false. By the definition of our valuation, we then infer that all P_j are marked, so $P_1 \wedge P_2 \wedge \cdots \wedge P_{k_i} \rightarrow P'$ is a conjunct of ϕ that would have been dealt with in one of the cycles of the while-statement and so P' is marked, too. Since $\bot$ is not marked, P' has to be $\top$ or some atom q. In any event, the conjunct is then true by (1.8), a contradiction □

Note that the proof by contradiction employed in the last proof was not really needed. It just made the argument seem more natural to us. The literature is full of such examples where one uses proof by contradiction more out of psychological than proof-theoretical necessity.

1.6 SAT solvers

The marking algorithm for Horn formulas computes marks as constraints on all valuations that can make a formule true. By (1.8), all marked atoms have to be true for any such valuation. We can extend this idea to general formulas ϕ by computing constraints saying which subformulas of ϕ require a certain truth value for all valuations that make ϕ true:

'All marked subformulas evaluate to their mark value
for all valuations in which ϕ evaluates to $\mathtt{T}$.' (1.9)

In that way, marking atomic formulas generalizes to marking subformulas; and 'true' marks generalize into 'true' and 'false' marks. At the same

time, (1.9) serves as a guide for designing an algorithm and as an invariant for proving its correctness.

1.6.1 A linear solver

We will execute this marking algorithm on the parse tree of formulas, except that we will translate formulas into the adequate fragment

$$\phi ::= p \mid (\neg\phi) \mid (\phi \wedge \phi) \tag{1.10}$$

and then share common subformulas of the resulting parse tree, making the tree into a directed, acyclic graph (DAG). The inductively defined translation

$$\begin{array}{ll} T(p) = p & T(\neg\phi) = \neg T(\phi) \\ T(\phi_1 \wedge \phi_2) = T(\phi_1) \wedge T(\phi_2) & T(\phi_1 \vee \phi_2) = \neg(\neg T(\phi_1) \wedge \neg T(\phi_2)) \\ T(\phi_1 \rightarrow \phi_2) = \neg(T(\phi_1) \wedge \neg T(\phi_2)) & \end{array}$$

transforms formulas generated by (1.3) into formulas generated by (1.10) such that ϕ and $T(\phi)$ are semantically equivalent and have the same propositional atoms. Therefore, ϕ is satisfiable iff $T(\phi)$ is satisfiable; and the set of valuations for which ϕ is true equals the set of valuations for which $T(\phi)$ is true. The latter ensures that the diagnostics of a SAT solver, applied to $T(\phi)$, is meaningful for the original formula ϕ. In the exercises, you are asked to prove these claims.

Example 1.48 For the formula ϕ being $p \wedge \neg(q \vee \neg p)$ we compute $T(\phi) = p \wedge \neg\neg(\neg q \wedge \neg\neg p)$. The parse tree and DAG of $T(\phi)$ are depicted in Figure 1.12.

Any valuation that makes $p \wedge \neg\neg(\neg q \wedge \neg\neg p)$ true has to assign `T` to the topmost $\wedge$-node in its DAG of Figure 1.12. But that forces the mark `T` on the p-node and the topmost $\neg$-node. In the same manner, we arrive at a complete set of constraints in Figure 1.13, where the time stamps '1:' etc indicate the order in which we applied our intuitive reasoning about these constraints; this order is generally not unique.

The formal set of rules for forcing new constraints from old ones is depicted in Figure 1.14. A small circle indicates any node ($\neg$, $\wedge$ or atom). The force laws for negation, $\neg_{\mathrm{t}}$ and $\neg_{\mathrm{f}}$, indicate that a truth constraint on a $\neg$-node forces its dual value at its sub-node and vice versa. The law $\wedge_{\mathrm{te}}$ propagates a `T` constraint on a $\wedge$-node to its two sub-nodes; dually, $\wedge_{\mathrm{ti}}$ forces a `T` mark on a $\wedge$-node if both its children have that mark. The laws $\wedge_{\mathrm{fl}}$ and $\wedge_{\mathrm{fr}}$ force a `F` constraint on a $\wedge$-node if any of its sub-nodes has a `F` value. The laws $\wedge_{\mathrm{fll}}$

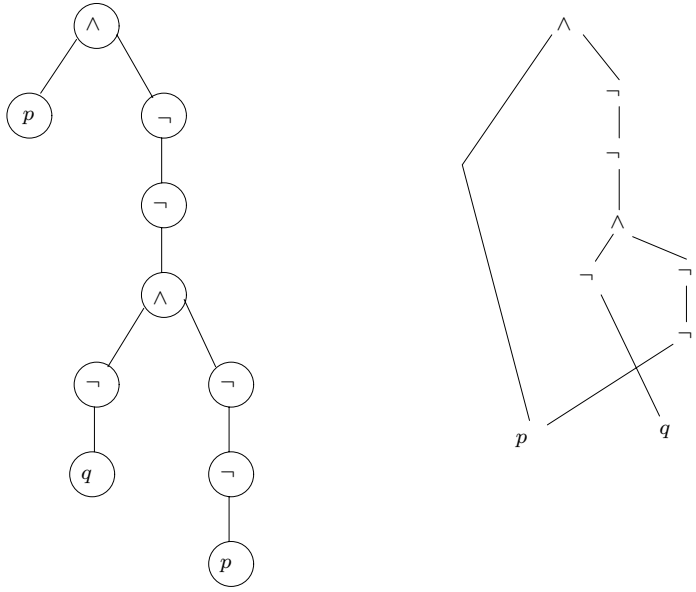

Figure 1.12. Parse tree (left) and directed acyclic graph (right) of the formula from Example 1.48. The p-node is shared on the right.

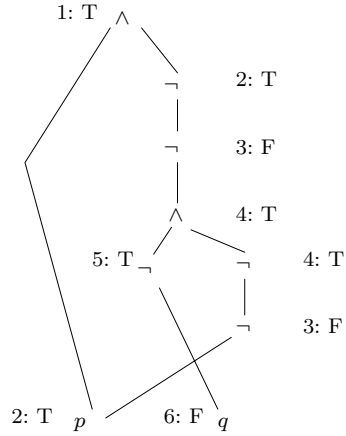

Figure 1.13. A witness to the satisfiability of the formula represented by this DAG.

and $\wedge_{\mathrm{frr}}$ are more complex: if an $\wedge$-node has a `F` constraint and one of its 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 marking does not yet show that this is a witness to the satisfiability of the formula

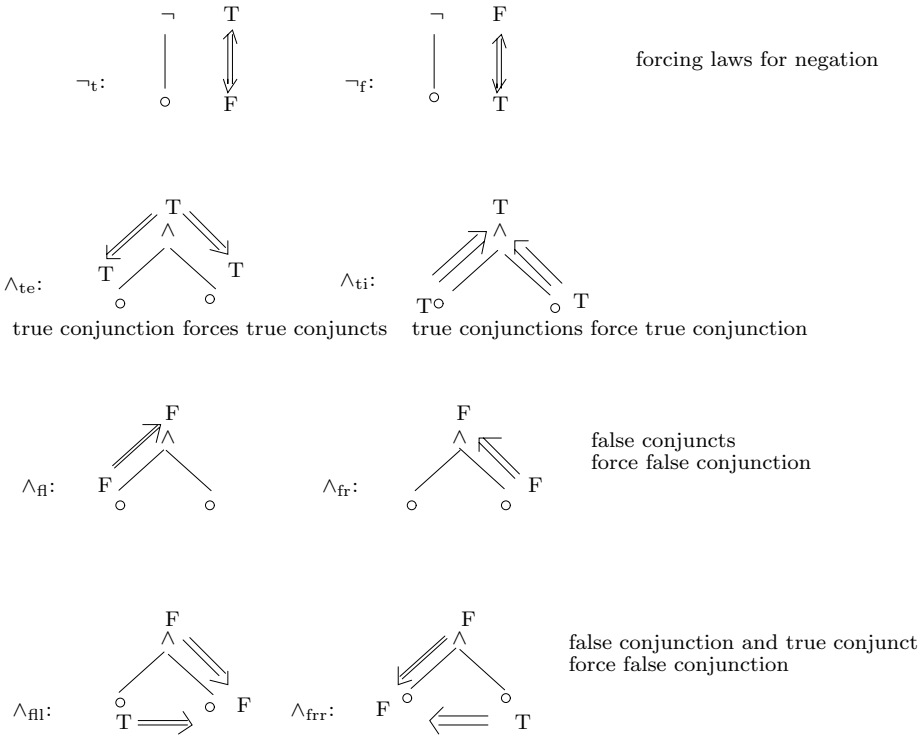

Figure 1.14. Rules for flow of constraints in a formula's DAG. Small circles indicate arbitrary nodes ($\neg$, $\wedge$ or atom). Note that the rules $\wedge_{\mathrm{fll}}$, $\wedge_{\mathrm{frr}}$ and $\wedge_{\mathrm{ti}}$ require that the source constraints of both $\Longrightarrow$ are present.

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 $T(\phi)$ whose DAG equals that of this figure are not satisfiable. In particular, all

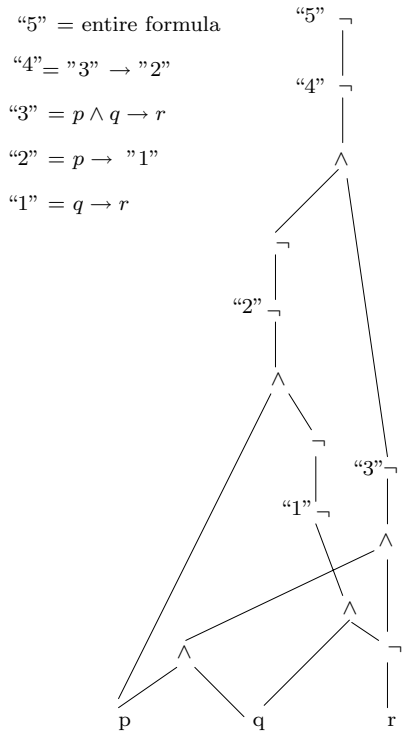

Figure 1.15. The DAG for the translation of $\neg((p \wedge q \to r) \to p \to q \to r)$. Labels "1" etc indicate which nodes represent what subformulas.

such ϕ 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 ϕ – 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 that this occurs for formulas of the form $\neg(\phi_1 \wedge \phi_2)$.

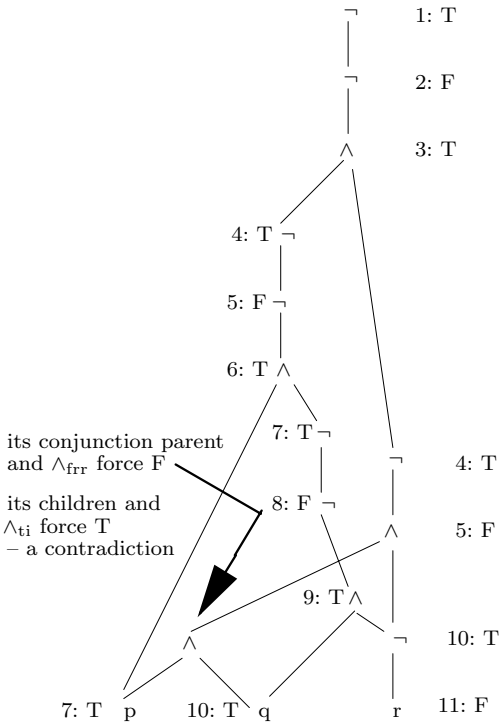

Figure 1.16. The forcing rules, applied to the DAG of Figure 1.15, detect contradictory constraints at the indicated node – implying that the initial constraint '1:T' cannot be realized. Thus, formulas represented by this DAG are not satisfiable.

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 depicts the DAG for $T(\phi)$, where ϕ is as in (1.11); and reveals

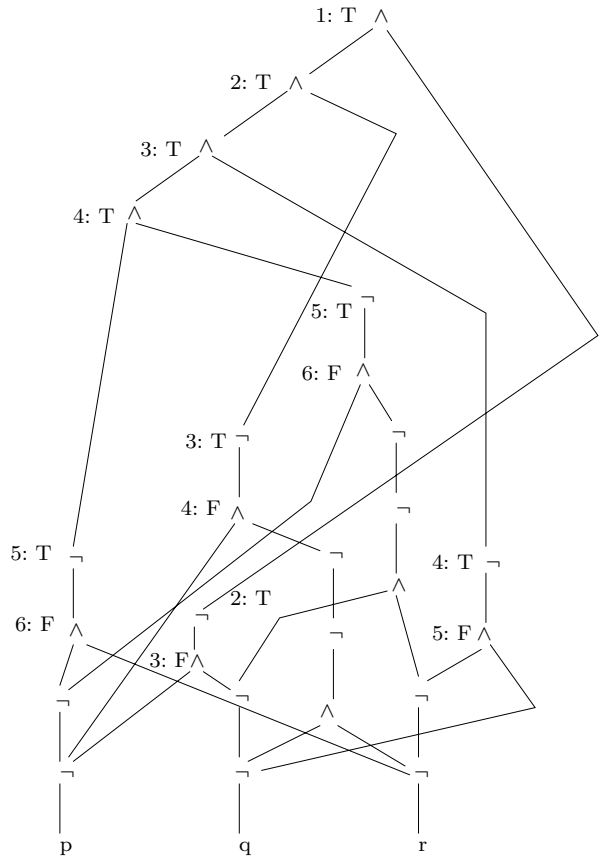

Figure 1.17. The DAG for the translation of the formula in (1.11). It has a $\wedge$-spine of length 4 as it is a conjunction of five clauses. Its linear analysis gets stuck: all forced constraints are consistent with each other but several nodes, including all atoms, are unconstrained.

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 1.17, the `F` mark only to n.

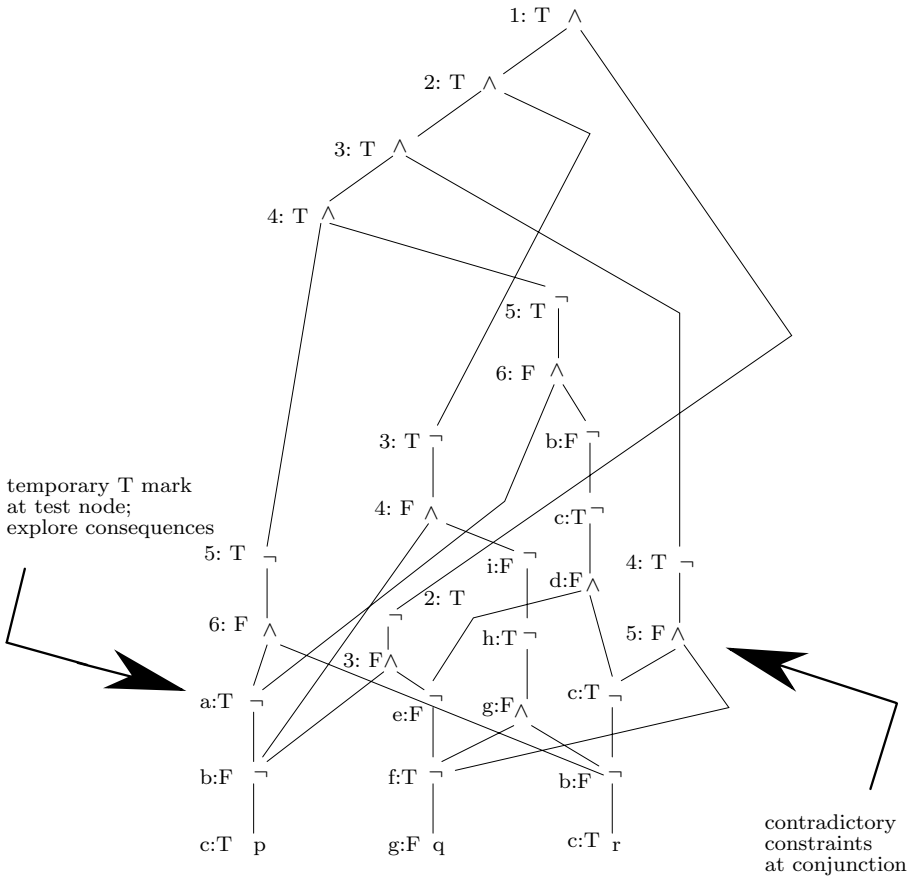

Figure 1.18. Marking an unmarked node with T and exploring what new constraints would follow from this. The analysis shows that this test marking causes contradictory constraints. We use lowercase letters 'a:' etc to denote temporary marks.

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 represented by that DAG are satisfiable.

Example 1.49 We revisit our stuck analysis of Figure 1.17. We test a ¬-node and explore the consequences of setting that ¬-node's mark to `T`; Figure 1.18 shows the result of that analysis. Dually, Figure 1.19 tests the consequences of setting that ¬-node's mark to `F`. Since both runs reveal a contradiction, the algorithm terminates, ruling that the formula in (1.11) is not satisfiable.

In the exercises, you are asked to show that the specification of our cubic SAT solver is sound. Its running time is indeed cubic in the size of the DAG (and the length of original formula). One factor stems from the linear SAT solver used in each test run. A second factor is introduced since each unmarked node has to be tested. The third factor is needed since each new permanent mark causes *all* unmarked nodes to be tested again.

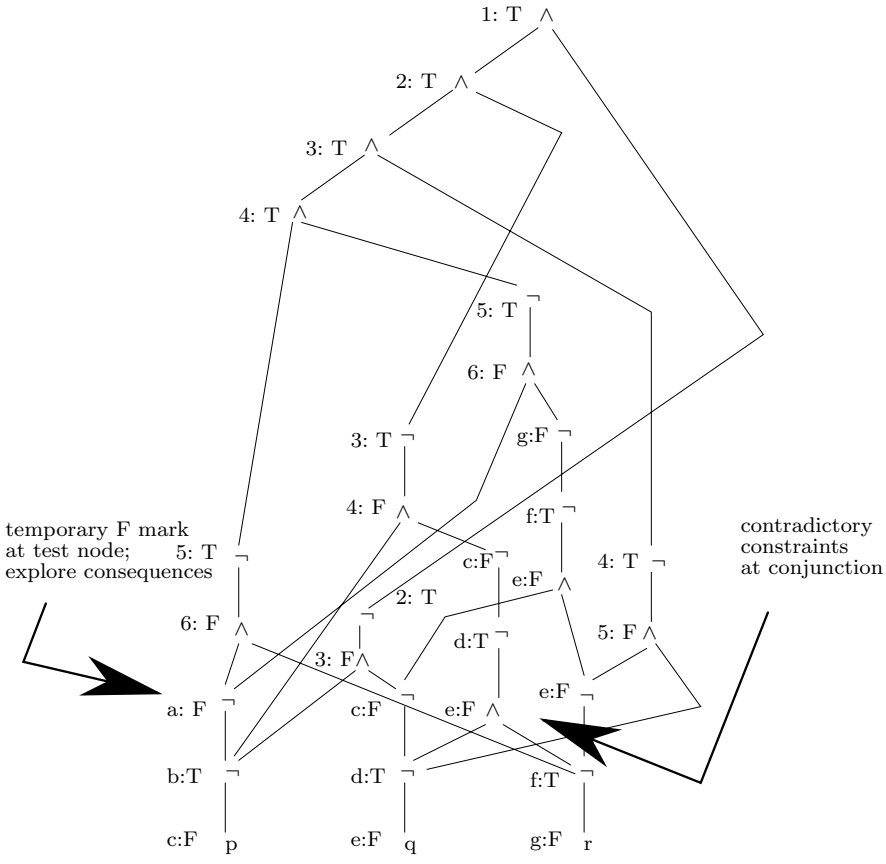

Figure 1.19. Marking the same unmarked node with `F` and exploring what new constraints would follow from this. The analysis shows that this test marking also causes contradictory constraints.

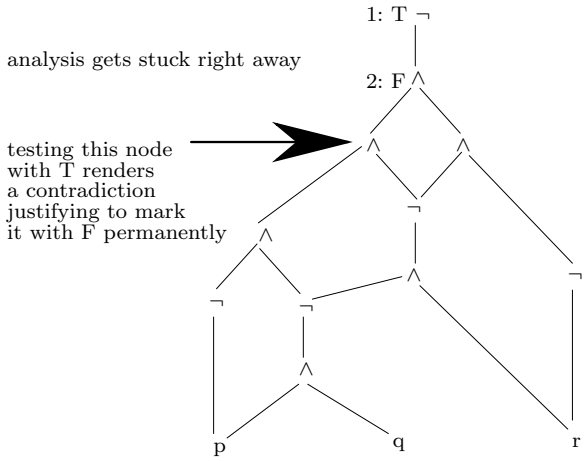

Figure 1.20. Testing the indicated node with T causes contradictory constraints, so we may mark that node with F permanently. However, our algorithm does not seem to be able to decide satisfiability of this DAG even with that optimization.

We deliberately under-specified our cubic SAT solver, but any implementation or optimization decisions need to secure soundness of the analysis. All replies of the form

1. 'The input formula is not satisfiable' and
2. 'The input formula is satisfiable under the following valuation ...'

have to be correct. The third form of reply 'Sorry, I could not figure this one out.' is correct by definition. :-) We briefly discuss two sound modifications to the algorithm that introduce some overhead, but may cause the algorithm to decide many more instances. Consider the state of a DAG right after we have explored consequences of a temporary mark on a test node.

1. If that state – permanent plus temporary markings – contains contradictory constraints, we can erase all temporary marks and mark the test node permanently with the dual mark of its test. That is, if marking node n with v resulted in a contradiction, it will get a permanent mark $\overline{v}$, where $\overline{\mathtt{T}} = \mathtt{F}$ and $\overline{\mathtt{F}} = \mathtt{T}$; otherwise
2. if that state managed to mark *all* nodes with consistent constraints, we report these markings as a witness of satisfiability and terminate the algorithm.

If none of these cases apply, we proceed as specified: promote shared marks of the two test runs to permanent ones, if applicable.

Example 1.50 To see how one of these optimizations may make a difference, consider the DAG in Figure 1.20. If we test the indicated node with

T, contradictory constraints arise. Since any witness of satisfiability has to assign some value to that node, we infer that it cannot be T. Thus, we may permanently assign mark F to that node. For this DAG, such an optimization does not seem to help. No test of an unmarked node detects a shared mark or a shared contradiction. Our cubic SAT solver fails for this DAG.

1.7 Exercises

Exercises 1.1

1. Use $\neg$, $\rightarrow$, $\wedge$ and $\vee$ to express the following declarative sentences in propositional logic; in each case state what your respective propositional atoms p, q, etc. mean:
   * (a) If the sun shines today, then it won't shine tomorrow.
   (b) Robert was jealous of Yvonne, or he was not in a good mood.
   (c) If the barometer falls, then either it will rain or it will snow.
   * (d) If a request occurs, then either it will eventually be acknowledged, or the requesting process won't ever be able to make progress.
   (e) Cancer will not be cured unless its cause is determined and a new drug for cancer is found.
   (f) If interest rates go up, share prices go down.
   (g) If Smith has installed central heating, then he has sold his car or he has not paid his mortgage.
   * (h) Today it will rain or shine, but not both.
   * (i) If Dick met Jane yesterday, they had a cup of coffee together, or they took a walk in the park.
   (j) No shoes, no shirt, no service.
   (k) My sister wants a black and white cat.
2. The formulas of propositional logic below implicitly assume the binding priorities of the logical connectives put forward in Convention 1.3. Make sure that you fully understand those conventions by reinserting as many brackets as possible. For example, given $p \wedge q \rightarrow r$, change it to $(p \wedge q) \rightarrow r$ since $\wedge$ binds more tightly than $\rightarrow$.
   * (a) $\neg p \wedge q \rightarrow r$
   (b) $(p \rightarrow q) \wedge \neg(r \vee p \rightarrow q)$
   * (c) $(p \rightarrow q) \rightarrow (r \rightarrow s \vee t)$
   (d) $p \vee (\neg q \rightarrow p \wedge r)$
   * (e) $p \vee q \rightarrow \neg p \wedge r$
   (f) $p \vee p \rightarrow \neg q$
   * (g) Why is the expression $p \vee q \wedge r$ problematic?

Exercises 1.2

1. Prove the validity of the following sequents:
   (a) $(p \wedge q) \wedge r, s \wedge t \vdash q \wedge s$