# Computer-Aided Verification

## *CS745/ECE745*

**Dr. Borzoo Bonakdarpour**

University of Waterloo

(Fall 2013)

Propositional Logic and SAT Solving

(Some Slides Adapted from Nancy Day's Lectures)

## Agenda

- What is verification?
- What is logic?
- Propositional logic
- SAT Solvers

## Agenda

- *What is verification?*
- What is logic?
- Propositional logic
- SAT Solvers

## What is Verification?

Verification involves checking a *satisfaction* relation, usually in the form of a sequent:

$$\mathcal{M} \models \phi, \text{ where}$$

- $\mathcal{M}$ is a *model*,

## What is Verification?

Verification involves checking a *satisfaction* relation, usually in the form of a sequent:

$$\mathcal{M} \models \phi, \text{ where}$$

- $\mathcal{M}$ is a *model*,
- $\phi$ is a *property* (or specification)

## What is Verification?

Verification involves checking a *satisfaction* relation, usually in the form of a sequent:

$$\mathcal{M} \models \phi, \text{ where}$$

- $\mathcal{M}$ is a *model*,
- $\phi$ is a *property* (or specification)
- $\models$ is a relationship that should hold between $\mathcal{M}$ and $\phi$; i.e., $(\mathcal{M}, \phi) \in \models$

# What is Verification?

Verification involves checking a *satisfaction* relation, usually in the form of a sequent:

$$\mathcal{M} \models \phi, \text{ where}$$

- $\mathcal{M}$ is a *model*,
- $\phi$ is a *property* (or specification)
- $\models$ is a relationship that should hold between $\mathcal{M}$ and $\phi$; i.e., $(\mathcal{M}, \phi) \in \models$

  We say that the model *satisfies* or “has” the property, or that we can conclude the property from the model.

# What is Verification?

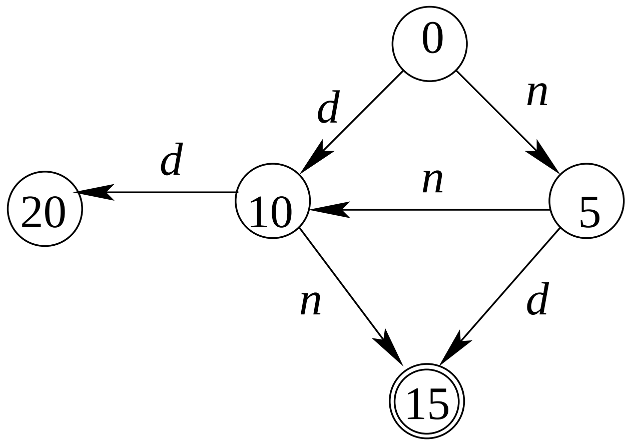
Verification involves:

1. specifying the model/system/implementation
2. specifying the property/specification
3. choosing the satisfaction relation
4. checking the satisfaction relation

These 4 steps are *NOT* independent.

## Example

Consider the operation of a soft drink vending machine which charges 15 cents for a can. The following figure is a *model* $\mathcal{M}$ of such a machine.

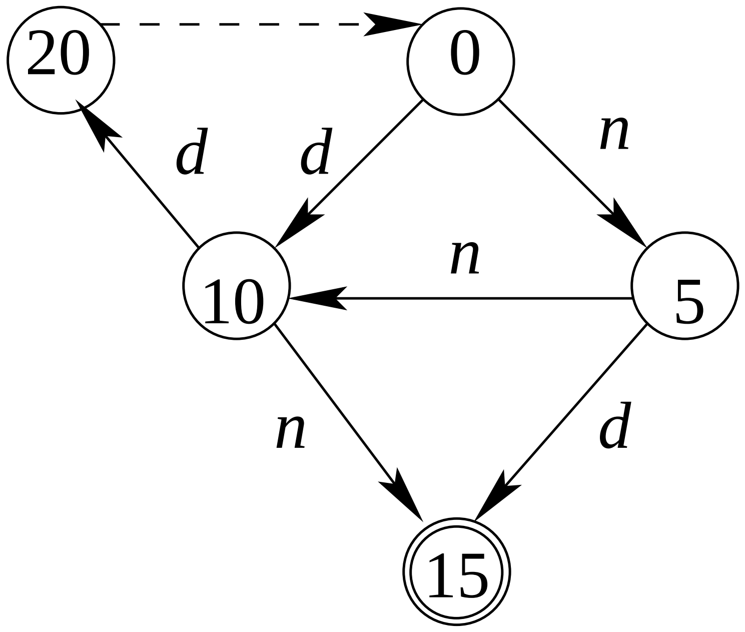


The following regular expression *specifies* the acceptable behavior of the machine: $\phi = n(d + nn)$

Question. $\mathcal{M} \models \phi$?

## Example (cont'd)

What about this model?



## Models and Properties

- The term *model* is used loosely here. It might not be executable, and it might not be a complete description of the system's behavior. The terms *implementation* and *specification* are relative.

## Models and Properties

- The term *model* is used loosely here. It might not be executable, and it might not be a complete description of the system's behavior. The terms *implementation* and *specification* are relative.
- An implementation generally contains more details than a specification. The specification for one level of verification might be the implementation at a higher level of verification.

# Models and Properties

- The term *model* is used loosely here. It might not be executable, and it might not be a complete description of the system’s behavior. The terms *implementation* and *specification* are relative.
- An implementation generally contains more details than a specification. The specification for one level of verification might be the implementation at a higher level of verification.
- In hardware, often the model is a description of the circuit in a hardware description language such as VHDL or Verilog. The real thing is the physical realization of the chip.

# Models and Properties

- The term *model* is used loosely here. It might not be executable, and it might not be a complete description of the system's behavior. The terms *implementation* and *specification* are relative.
- An implementation generally contains more details than a specification. The specification for one level of verification might be the implementation at a higher level of verification.
- In hardware, often the model is a description of the circuit in a hardware description language such as VHDL or Verilog. The real thing is the physical realization of the chip.
- Sometimes the model is actually a specification and the property is an attribute such as completeness or consistency.

## Logic and Verification

- Different modelling languages and logics give us different ways of expressing $\mathcal{M}$ and $\phi$ and defining membership of the pair $(\mathcal{M}, \phi)$ to $\models$.

# Logic and Verification

- Different modelling languages and logics give us different ways of expressing $\mathcal{M}$ and $\phi$ and defining membership of the pair $(\mathcal{M}, \phi)$ to $\models$.
- Hopefully, the calculation of the satisfaction relation is *compositional* in either the property or the model. This decomposes the verification task.

# Logic and Verification

- Different modelling languages and logics give us different ways of expressing $\mathcal{M}$ and $\phi$ and defining membership of the pair $(\mathcal{M}, \phi)$ to $\models$.

- Hopefully, the calculation of the satisfaction relation is *compositional* in either the property or the model. This decomposes the verification task.

- The model and property both describes sets of *behaviors*.

## Logic and Verification

- Different modelling languages and logics give us different ways of expressing $\mathcal{M}$ and $\phi$ and defining membership of the pair $(\mathcal{M}, \phi)$ to $\models$.

- Hopefully, the calculation of the satisfaction relation is *compositional* in either the property or the model. This decomposes the verification task.

- The model and property both describes sets of *behaviors*.

- The satisfaction relation is a relation between the set of behaviors of the model and the set of behaviors of the property.

# Agenda

- What is verification?
- *What is logic?*
- Propositional logic
- SAT Solvers

## What is Logic?

According to Kelly:

In general, *logic* is about reasoning. It is about the *validity* of arguments, *consistency* among statements (. . . ) and matters of *truth* and *falsehood*.

In a formal sense logic is concerned only with the *form of arguments* and the principles of *valid inferencing* .

## What is Logic?

According to Webster’s, logic is:

the science of *correct reasoning*, valid induction or deduction. Symbolic logic is a modern type of formal logic using special mathematical symbols for propositions, quantifiers, and relationships among propositions and concerned with the elucidation of permissible operations upon such symbols.

## What is Logic?

According to the Free On-Line Dictionary of Computing:

A branch of philosophy and mathematics that deals with the formal principles, methods and criteria of validity of inference, reasoning and knowledge.

Logic is concerned with what is true and how we can know whether something is true. This involves the formalization of logical arguments and proofs in terms of symbols representing propositions and logical connectives. The meanings of these logical connectives are expressed by a set of rules which are assumed to be self-evident .

In symbolic logic, arguments and proofs are made in terms of symbols representing propositions and logical connectives. The meanings of these begin with a set of rules or primitives which are assumed to be self-evident. Fortunately, even from vague primitives, functions can be defined with precise meaning.

## Elements of a Logic

A logic consists of:

1. syntax
2. semantics
3. proof procedure(s) (also called proof theory)

# Syntax and Semantics

- syntax:
  - define "well-formed formula"
- semantics:
  - define " $\models$ " ("satisfies")
    $\mathcal{M} \models \phi$ (satisfaction relation)
  - define $\phi_1, \phi_2, \phi_3 \models \psi$ ("entails", or semantic entailment) means:
    from the *premises* $\phi_1, \phi_2, \phi_3,$ we may conclude $\psi$, where $\phi_1, \phi_2, \phi_3,$ and are all well-formed formulae in the logic.

# Proof Procedure

- proof procedure(s):
  - define “ $\vdash$ ” (pronounced “proves”)
  - a proof procedure is a way to calculate $\phi_1, \phi_2, \phi_3, \cdots \vdash \psi$ (also called a sequent). By “calculation”, we mean that there is a procedure for determining if $((\phi_1, \phi_2, \phi_3, \cdots), \psi) \in \vdash$
  - there may be multiple proof procedures, which we will indicate by subscripting $\vdash$, e.g., the *sequent calculus* proof procedure for propositional logic will be $\vdash_{\mathsf{SQ}}$.
  - for some logics, there is not a proof procedure that always terminates for any sequent.

## Proof Procedures

Proof procedures are algorithms that perform mechanical manipulations on strings of symbols. A proof procedure does not make use of the meanings of sentences, it only manipulates them as formal strings of symbols.

There may be multiple ways to prove a sequent in a particular proof procedure.

## Soundness and Completeness

The semantics and the proof procedures ($\models$ and $\vdash$) are related in the concepts of *soundness* and *completeness*.

Definition. A proof procedure is *sound* if $\phi_1, \phi_2, \phi_3 \vdash \psi$ then $\phi_1, \phi_2, \phi_3 \models \psi$.
A proof procedure is sound if it proves only tautologies.

Definition. A proof procedure is *complete* if $\phi_1, \phi_2, \phi_3 \models \psi$ then $\phi_1, \phi_2, \phi_3 \vdash \psi$.
A proof procedure is complete if it proves every tautology.

Note that in the literature, there is not consistent use of the symbols $\models$ and $\vdash$.

## Consistency

Definition. A proof procedure is *consistent* if it is not possible to prove both $A$ and $\neg A$, i.e.,

$$\text{not both } \vdash A \text{ and } \vdash \neg A.$$

## Agenda

- What is verification?
- What is logic?
- *Propositional logic*
- SAT Solvers

## Propositional Logic: Syntax

Its syntax consists of:

- Two constant symbols: **true** and **false**
- Proposition letters
- Propositional connectives
- Brackets

# Propositional Connectives

Definition. The propositional (logical) connectives are:

| Symbol | Informal Meaning |
|---|---|
| $\neg$ | negation (not) |
| $\wedge$ | conjunction (and, both) |
| $\vee$ | disjunction (or, at least one of) |
| $\Rightarrow$ | implication (implies, logical consequence, conditional, if . . . then ) |
| $\Leftrightarrow$ | equivalent (biconditional, if and only if) |

Others may use different symbols for these operations.

# Terminology

For an implication $p \Rightarrow q$:

- $p$ is the *premise* or *antecedent* or *hypothesis*
- $q$ is the *consequent* or *conclusion*

$\neg b \Rightarrow \neg a$ is called the *contrapositive* of $a \Rightarrow b$.

The set of connectives $\{\wedge, \neg\}$ are complete in the sense that all the other connectives can be defined using them, e.g., $a \vee b = \neg(\neg a \wedge \neg b)$. Other subsets of the binary connectives are also complete in the same sense.

## Propositions

Definition. *Proposition letters* represent declarative sentences, i.e., sentences that are true or false. Sentences matching proposition letters are atomic (non-decomposable), meaning they don't contain any of the propositional connectives.

Here are some examples:

- It is raining outside.
- The sum of 2 and 5 equals 3.
- The value of program variable `a` is 42.

Sentences that are interrogative (questions), or imperative (commands) are not propositions.

## Using Symbols

Because in logic, we are only concerned with the structure of the argument and which structures of arguments are valid, we “encode” the sentences in symbols to create a more compact and clearer representation of the argument. We call these propositional symbols or proposition letters.

DO NOT use $T$, $F$, $t$, or $f$ in any font as symbols representing sentences!

## Well-formed formulas

Definition. The *well-formed formulae* of propositional logic are those obtained as follows:

$$\varphi ::= \mathbf{true} \mid \mathbf{false} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2$$

where $p$ is an atomic proposition.

Note that this is an inductive definition, meaning the set is defined by basis elements, and rules to construct elements from elements in the set. Thus, the best strategy to prove properties of propositional formulas is using *structural induction*.

## Well-formed formulas

Brackets around the outermost formula are usually omitted using the following rules of *precedence*: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

$\Rightarrow$ is right *associative*, meaning $p \Rightarrow q \Rightarrow r$ is $p \Rightarrow (q \Rightarrow r)$.

## Semantics

Semantics means “meaning”. Semantics relate two worlds. Semantics provide an interpretation (mapping) of expressions in one world in terms of values in another world. Semantics are often a *function* from expressions in one world to expressions in another world.

The semantics (i.e., the mapping) is often called a *model* or an *interpretation*. We write $\mathcal{M} \models \phi$ to mean the model satisfies the formula. In propositional logic, models are called Boolean valuations.

Proof procedures transform the syntax of a logic in ways that respect the semantics.

## Semantics of Propositional Logic

We have described the syntax for propositional logic, which is the domain of the semantic function.

*Classical logic* is two-valued. The two possible truth values are **T**, and **F**, which are two distinct values.

The range of the semantic function for propositional logic is the set of truth values:

$$\boxed{\mathbf{Tr} = \{\mathbf{T}, \mathbf{F}\}}$$

Note that these truth values are distinct from the syntax elements **true**, and **false**.

# Semantics of Propositional Logic

Truth Values: **Tr** = {**T**, **F**}

There are functions on these truth values that correspond to the meaning of the propositional connectives. We overload the operators "$\wedge$", "$\vee$", etc. to be both part of the syntax of propositional logic, and operations on the sets of truth values in our model for propositional logic.

$\neg : \mathbf{Tr} \rightarrow \mathbf{Tr}$
$\wedge : (\mathbf{Tr} \times \mathbf{Tr}) \rightarrow \mathbf{Tr}$
etc.

Truth tables are used to describe the functions of operations on these truth values.

## Truth Tables

| $p$ | $\neg p$ |
|---|---|
| **T** | **F** |
| **F** | **T** |

| $p$ | $q$ | $p \wedge q$ | $p \vee q$ | $p \Rightarrow q$ | $p \Leftrightarrow q$ |
|---|---|---|---|---|---|
| **T** | **T** | **T** | **T** | **T** | **T** |
| **T** | **F** | **F** | **T** | **F** | **F** |
| **F** | **T** | **F** | **T** | **T** | **F** |
| **F** | **F** | **F** | **F** | **T** | **T** |

# Boolean Valuations

Definition. A *Boolean valuation* is a mapping $v$ from the set of propositional formulas to the set $\mathbf{Tr}$ meeting the conditions:

- $v(\mathbf{true}) = \mathbf{T}, v(\mathbf{false}) = \mathbf{F}$
- $v(\neg p) = \neg(v(p))$
- for all the connectives: $v(p \circ q) = v(p) \circ v(q)$

Note that $\neg(v(p))$ and $v(p) \circ v(q)$ are given by the truth tables on the previous slide.

# Satisfiability

Definition. A formula a is *satisfiable* if there is a Boolean valuation $v$ such that $v(a) = \mathbf{T}$.

We sometimes say that the formula “has a satisfying assignment” to mean that it is satisfiable.

We are mostly interested in the propositional formulas that map to T in all the possible Boolean valuations (i.e., in all model).

# Tautologies

Definition. A propositional formula a is a *tautology* (also called *valid* or a *theorem*) if $v(a) = \mathbf{T}$ for every Boolean valuation $v$.

i.e., , a tautology is a formula that is true for all possible truth values of the propositional letters used in the formula. The last column of the truth table for a tautology contains all $\mathbf{T}$.

Note that a formula $a$ is a tautology iff $\neg a$ is not satisfiable.

# Semantic Entailment

$$\phi_1, \phi_2, \phi_3 \models \psi$$

means that for all $v$, if $v(\phi_1) = \mathbf{T}$ and $v(\phi_2) = \mathbf{T}$ and $v(\phi_3) = \mathbf{T}$, then $v(\psi) = \mathbf{T}$, which is equivalent to saying

$$(\phi_1 \wedge \phi_2 \wedge \phi_3) \Rightarrow \psi$$

is a tautology, i.e.,

$$\boxed{(\phi_1, \phi_2, \phi_3 \models \psi) \quad \equiv \quad (\phi_1 \wedge \phi_2 \wedge \phi_3) \Rightarrow \psi}$$

# Models and Entailment

In propositional (and predicate) logic, $\models$ is overloaded and has two meanings:

- $\mathcal{M} \models \phi$ relates a model to a formula, saying that $\mathcal{M}$ satisfies the formula. This is called a *satisfaction relation*.
- $\psi \models \phi$ relates two formulas, saying that forall $v$ (i.e., for all possible models), if $v(\psi) = \mathbf{T}$, then $v(\phi) = \mathbf{T}$. This is called *semantic entailment*.

These two uses can be distinguished by their context.

## Contradiction

Definition. A *contradiction* is a formula that is false for all possible truth values of the propositional symbols used in the formula. The last column of the truth table for a tautology contains all $\mathbf{F}$.

## Decidability

A logic is *decidable* if there is an algorithm to determine if any formula of the logic is a tautology (is a theorem, is valid).

Propositional logic is decidable because we can always construct the truth table for the formula.

Question: What is the *complexity* of deciding propositional logic?

## Proof Procedures

We can always determine if a formula is a tautology by using truth tables to determine the value of the formula for every possible combination of values for its proposition letters, but this would be very tedious since the size of the truth table grows *exponentially*.

Proof procedures for propositional logic are alternate means to determine tautologies. As long as the proof procedure is sound, we can use the proof procedure in place of truth tables to determine tautologies.

## Proof Styles

A *proof procedure* is a set of rules we use to transform premises and conclusions into new premises and conclusions.

A *goal* is a formula that we want to prove is a tautology. It has premises and conclusions.

A *proof* is a sequence of proof rules that when chained together relate the premise of the goal to the conclusion of the goal.

# Sequent Calculus

Definition. A *sequent* is a pair $(\Gamma, \Delta)$ of finite sets of formulae.

We will write a sequent as $\Gamma \hookrightarrow \Delta$ and drop the set brackets around the sets of formulae. $X$ will represent a single formula, where as $\Gamma$ is a set of formulae.

(More commonly, $\hookrightarrow$ is written as $\rightarrow$.)

The $\hookrightarrow$ is like implication. A sequent asserts: if all the formulae on the left of the arrow are true, then at least one of the formulae on the right are true.

## Meaning of a Sequent

We can extend Boolean valuations to describe the meaning of a sequent.

$v(\Gamma \hookrightarrow \Delta) = \mathbf{T}$

iff $v(X) = \mathbf{F}$ for some $X$ in $\Gamma$

or $v(X) = \mathbf{T}$ for some $X$ in $\Delta$.

When there is nothing on the LHS or RHS of the arrow, we assume it is the empty set of formulae.

This means $v(\hookrightarrow) = \mathbf{F}$, and $v(\hookrightarrow X) = v(X)$.

# Axioms

$$X \hookrightarrow X \text{ (Id)}$$

$$\mathbf{false} \hookrightarrow$$

$$\hookrightarrow \mathbf{true}$$

## Sequent Schemata

The rules of the sequent calculus are written in the form:

$$\frac{S_1}{S_2}$$

If a formula matches the schema $S_1$, then it can be replaced by one matching $S_2$.

Stated another way: if you know $S_1$ is true, then $S_2$ is also true.

# Sequent Calculus Rules

*Structural Rule (Thinning)*

If $\Gamma_1 \subseteq \Gamma_2$ and $\Delta_1 \subseteq \Delta_2$ then:

$$\frac{\Gamma_1 \hookrightarrow \Delta_1}{\Gamma_2 \hookrightarrow \Delta_2}$$

Thinning is like *precondition strengthening* and *postcondition weakening* for those familiar with that terminology.

Adding formulae on the RHS of the sequent is adding them to a disjunction, so this is weakening the RHS formulae.

Adding formulae on the LHS of the sequent is adding them to a conjunction, so this is strengthening the LHS formulae.

# Sequent Calculus Rules

*Negation Rules*

$$\frac{\Gamma \hookrightarrow \Delta, X}{\Gamma, \neg X \hookrightarrow \Delta} \qquad \frac{\Gamma, X \hookrightarrow \Delta}{\Gamma \hookrightarrow \Delta, \neg X}$$

*Conjunction Rules*

$$\frac{\Gamma, X, Y \hookrightarrow \Delta}{\Gamma, X \wedge Y \hookrightarrow \Delta} \qquad \frac{\Gamma_1 \hookrightarrow \Delta_1, X \qquad \Gamma_2 \hookrightarrow \Delta_2, Y}{\Gamma_1, \Gamma_2 \hookrightarrow \Delta_1, \Delta_2, X \wedge Y}$$

*Disjunction Rules*

$$\frac{\Gamma \hookrightarrow \Delta, X, Y}{\Gamma \hookrightarrow \Delta, X \vee Y} \qquad \frac{\Gamma_1, X \hookrightarrow \Delta_1 \qquad \Gamma_2, Y \hookrightarrow \Delta_2}{\Gamma_1, \Gamma_2, X \vee Y \hookrightarrow \Delta_1, \Delta_2}$$

# Sequent Calculus Rules

*Implication Rules*

#1

$$\frac{\Gamma, X \hookrightarrow \Delta, Y}{\Gamma \hookrightarrow \Delta, X \Rightarrow Y}$$

#2

$$\frac{\Gamma_1 \hookrightarrow \Delta_1, X \qquad \Gamma_2, Y \hookrightarrow \Delta_2}{\Gamma_1, \Gamma_2, X \Rightarrow Y \hookrightarrow \Delta_1, \Delta_2}$$

The right-hand rule is similar to *modus ponens*. The idea is that if $X$ can be derived, then from $X \Rightarrow Y$ , $Y$ can be derived, and therefore whatever can be derived from $Y$ can be derived. If $\Gamma_2$, and $\Delta_1$ are empty:

$$\frac{\Gamma_1 \hookrightarrow X \qquad Y \hookrightarrow \Delta_2}{\Gamma_1, X \Rightarrow Y \hookrightarrow \Delta_2}$$

## Cut Rule

This a derived rule:

$$\frac{\Gamma_1 \hookrightarrow \Delta_1, X \qquad \Gamma_2, X \hookrightarrow \Delta_2}{\Gamma_1, \Gamma_2 \hookrightarrow \Delta_1, \Delta_2}$$

Question: How is it derived?

# Proofs in the Sequent Calculus

Definition. A proof is a *tree* labelled with sequents (generally written with the root at the bottom), such that: if node $N$ is labelled with $\Gamma \hookrightarrow \Delta$, then if $N$ is a leaf node, $\Gamma \hookrightarrow \Delta$ must be an axiom; if $N$ has children, their labels must be the premises from which $\Gamma \hookrightarrow \Delta$ follows by one of the rules. The label on the root node is the sequent that is proved.

Definition. A formula $X$ is a theorem of the sequent calculus if the sequent $\hookrightarrow X$ has a proof, i.e.,

$$\vdash_{\mathsf{SQ}} X$$

The sequent calculus for propositional logic is both sound and complete.

## Example

Show $\hookrightarrow A \Rightarrow (B \Rightarrow A)$

| | | |
|---|---|---|
| 1. | $A \hookrightarrow A$ | (id) |
| 2. | $A, B \hookrightarrow A$ | (Thinning) |
| 3. | $A \hookrightarrow B \Rightarrow A$ | (Implication #1) |
| 4. | $\hookrightarrow A \Rightarrow (B \Rightarrow A)$ | (Implication #1) |

## Agenda

- What is verification?
- What is logic?
- Propositional logic
- *SAT Solvers*

## SAT Solving

A *SAT solver* is an algorithm that determines whether or not a propositional formula is satisfiable.

A naive (and costly) algorithm is forming the truth table of a formula.

More efficient algorithms can be designed based on the structure of formulas.

## SAT Solving - Horn Formulas

A *Horn formula* is a formula $\varphi$ if it can be generated as follows:

$$\begin{aligned} P &::= \bot \mid \top \mid p \\ A &::= P \mid P \wedge A \\ C &::= A \Rightarrow P \\ H &::= C \mid C \wedge H \end{aligned}$$

We call each instance of $C$ a *Horn clause*. I.e., horn formulas are conjunctions of Horn clauses.

## SAT Solving - Horn Formulas

Example of Horn formulas:

$$(p \wedge q \wedge s \Rightarrow p) \wedge (q \wedge r \Rightarrow p) \wedge (p \wedge s \Rightarrow s)$$
$$(p \wedge q \wedge s \Rightarrow \bot) \wedge (q \wedge r \Rightarrow p) \wedge (\top \Rightarrow s)$$

These formulas are not Horn formulas:

$$(p \wedge q \wedge s \Rightarrow \neg p) \wedge (q \wedge r \Rightarrow p) \wedge (p \wedge s \Rightarrow s)$$
$$(p \wedge q \wedge s \Rightarrow \bot) \wedge (\neg q \wedge r \Rightarrow p) \wedge (\top \Rightarrow s)$$

## SAT Solving - Horn Formulas

**Algorithm:** For a Horn formulas $\varphi$, maintain a list of occurrences of type $P$ in $\varphi$ and proceed as follows:

1. Mark $\top$ if it occurs in that list.
2. If there is a conjunct $P_1 \wedge P_2 \wedge \cdots P_{k_i} \Rightarrow P'$ of $\varphi$ such that all $P_j$ $(1 \leq j \leq k_i)$ are marked, then mark $P'$ and go to 2.
3. If $\bot$ is marked, print out '*unsatisfiable*' and stop.
4. Print out '*satisfiable*'.

## SAT Solving - Horn Formulas

This technique is called *unit propagation*. I.e., we first find the necessary assignments that are propagated in the rest of clauses.

## SAT Solving - Horn Formulas

**Example.** Consider formula

$$\varphi = (p \Rightarrow q) \wedge (p \wedge q \Rightarrow r) \wedge (\top \Rightarrow p) \wedge (r \Rightarrow \bot)$$

- Step 1: marked = $\{\top\}$
- Step 2 marked = $\{\top, p\}$ (3rd conjunct)
- Step 2 marked = $\{\top, p, q\}$ (1st conjunct)
- Step 2 marks $\{\top, p, q, r\}$ in the 2nd conjunct
- Step 2 marks $\{\top, p, q, r, \bot\}$ in the 4th conjunct
- Step 3 declared unsatifiability of $\varphi$

## SAT Solving - A Linear Solver

We now generalize the marking algorithm on parse tree of formulas by translating them into the following fragment:

$$\varphi ::= p \mid (\neg\varphi) \mid (\varphi_1 \wedge \varphi_2)$$

and then share common subformulas of the resulting parse tree, making the tree into a DAG:

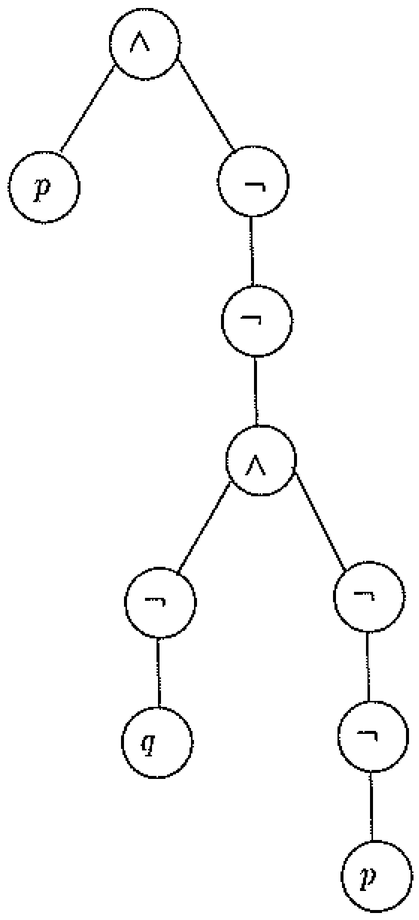
$$T(p) = p \qquad T(\varphi_1 \wedge \varphi_2) = T(\varphi_1) \wedge T(\varphi_2)$$

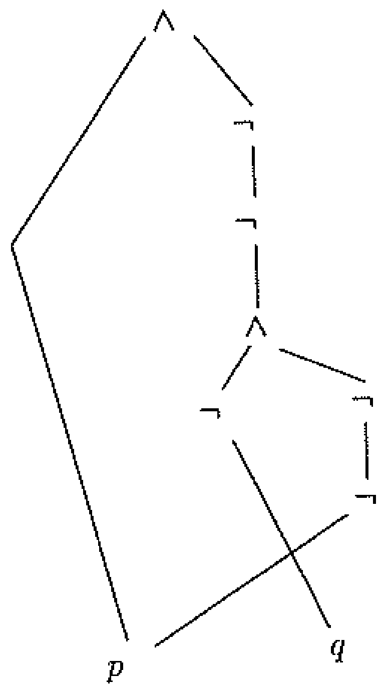
$$T(\varphi) = \neg T(\varphi) \quad T(\varphi_1 \vee \varphi_2) = \neg(\neg T(\varphi_1) \wedge \neg T(\varphi_2))$$

$$T(\varphi_1 \Rightarrow \varphi_2) = \neg(T(\varphi_1) \wedge \neg T(\varphi_2))$$

## SAT Solving - A Linear Solver

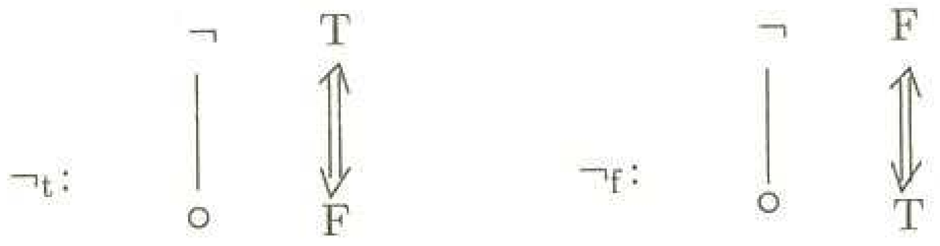
**Example.** For formula $\varphi = p \wedge \neg(q \vee \neg p)$, the parse tree and DAG of $T(\varphi)$ are the following:



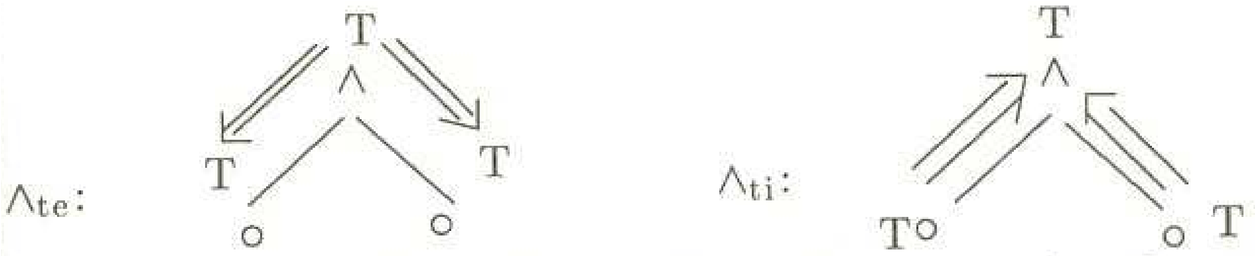


# SAT Solving - A Linear Solver

Unit propagation rules for the formulas DAGs:



forcing laws for negation

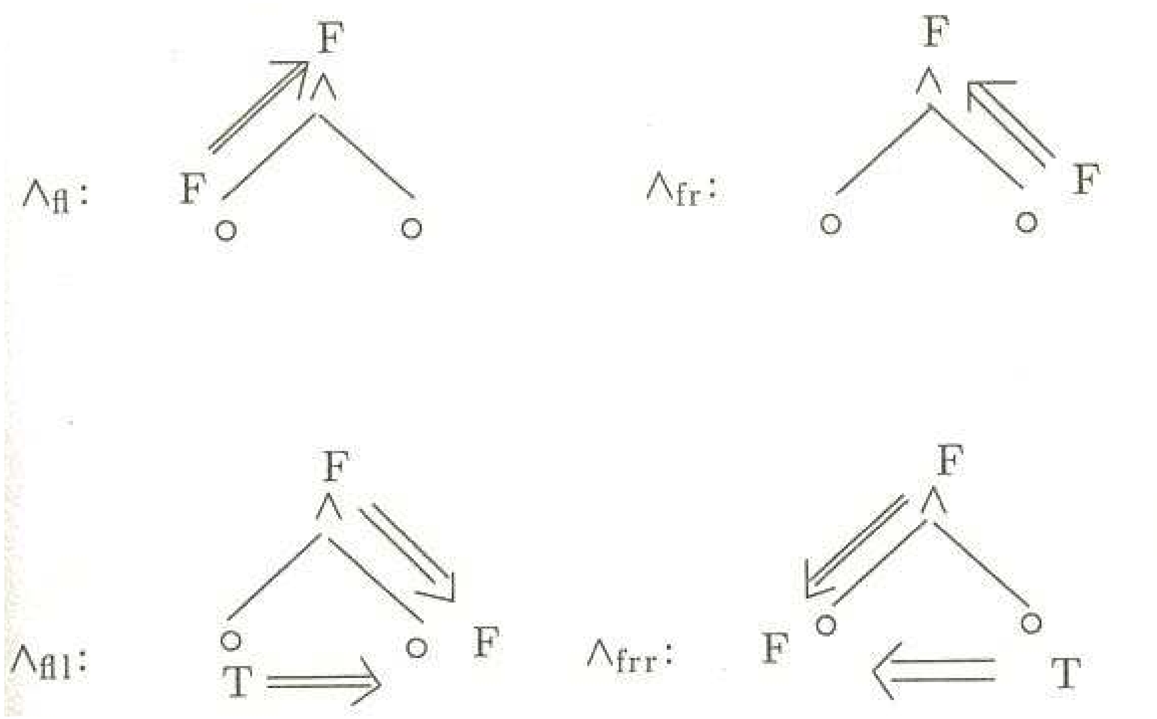


true conjunction forces true conjuncts

true conjunctions force true conjunction

# SAT Solving - A Linear Solver

Unit propagation rules for the formulas DAGs:



false conjuncts
force false conjunction

false conjunction and true conjunct
force false conjunction

# SAT Solving - A Linear Solver

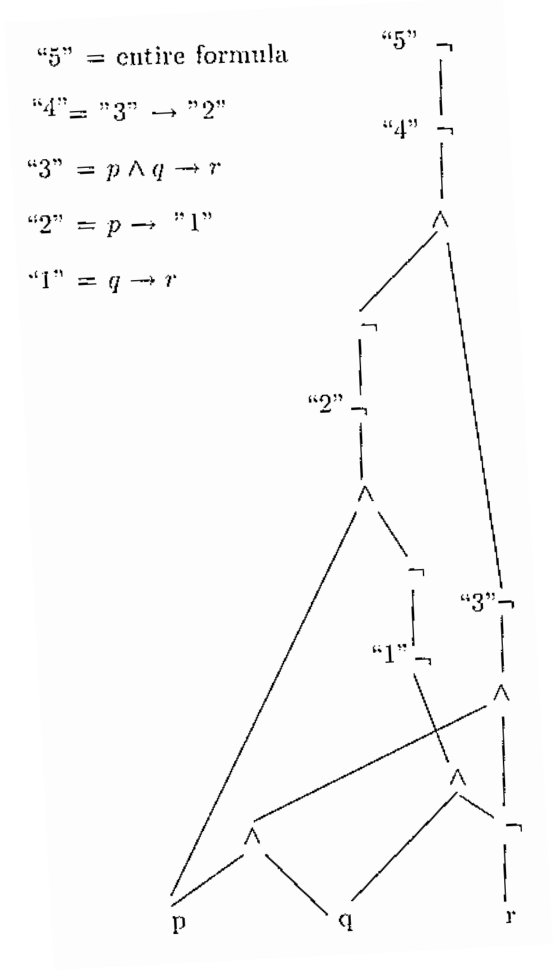
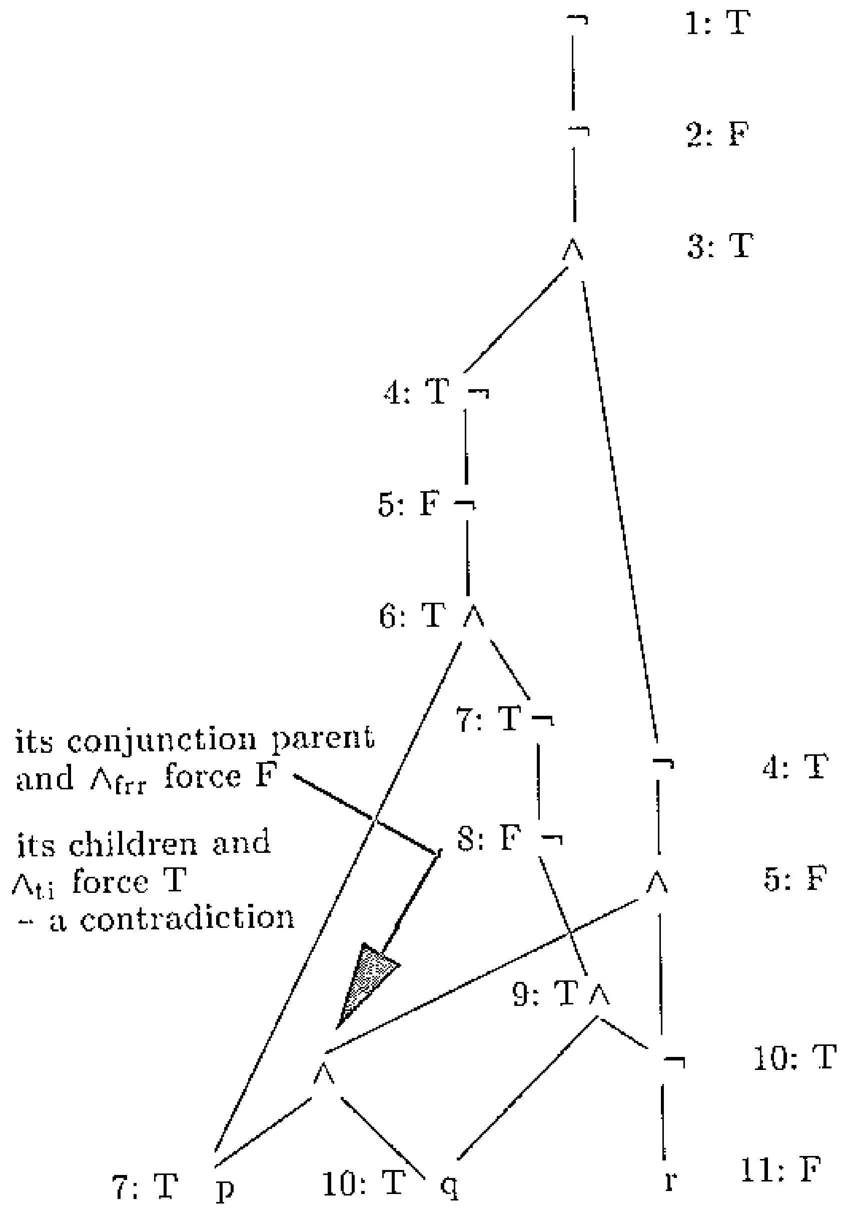


Figure 1: $\neg((p \wedge q \Rightarrow r) \Rightarrow p \Rightarrow q \Rightarrow r)$

# SAT Solving - A Linear Solver



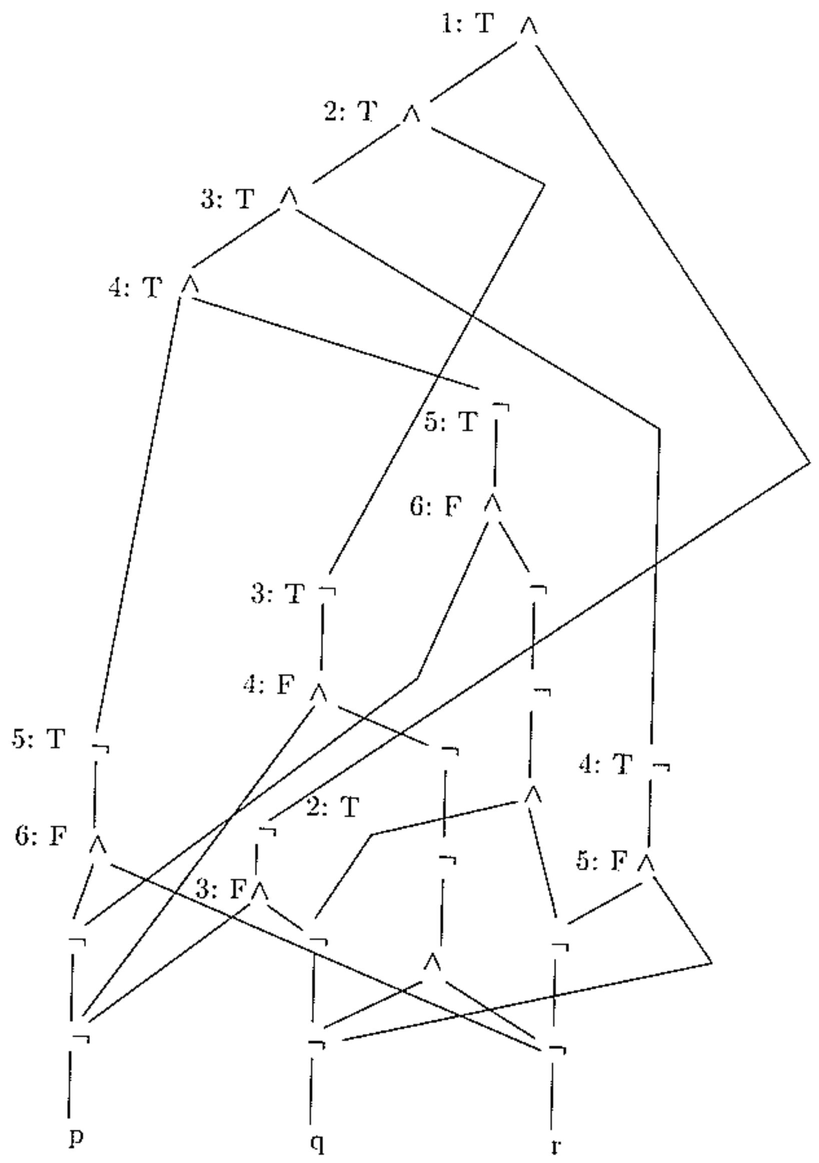
## SAT Solving - A Cubic Solver

The linear solver may get stuck on some formulas. For instance, the following formula:

$$(p \vee q \vee r) \wedge (p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (\neg p \vee \neg q \vee \neg r)$$

a SAT solving algorithm should reach a contradiction quickly (why?), but the linear algorithm gets stuck (see the next slide).

# SAT Solving - A Cubic Solver



## SAT Solving - A Cubic Solver

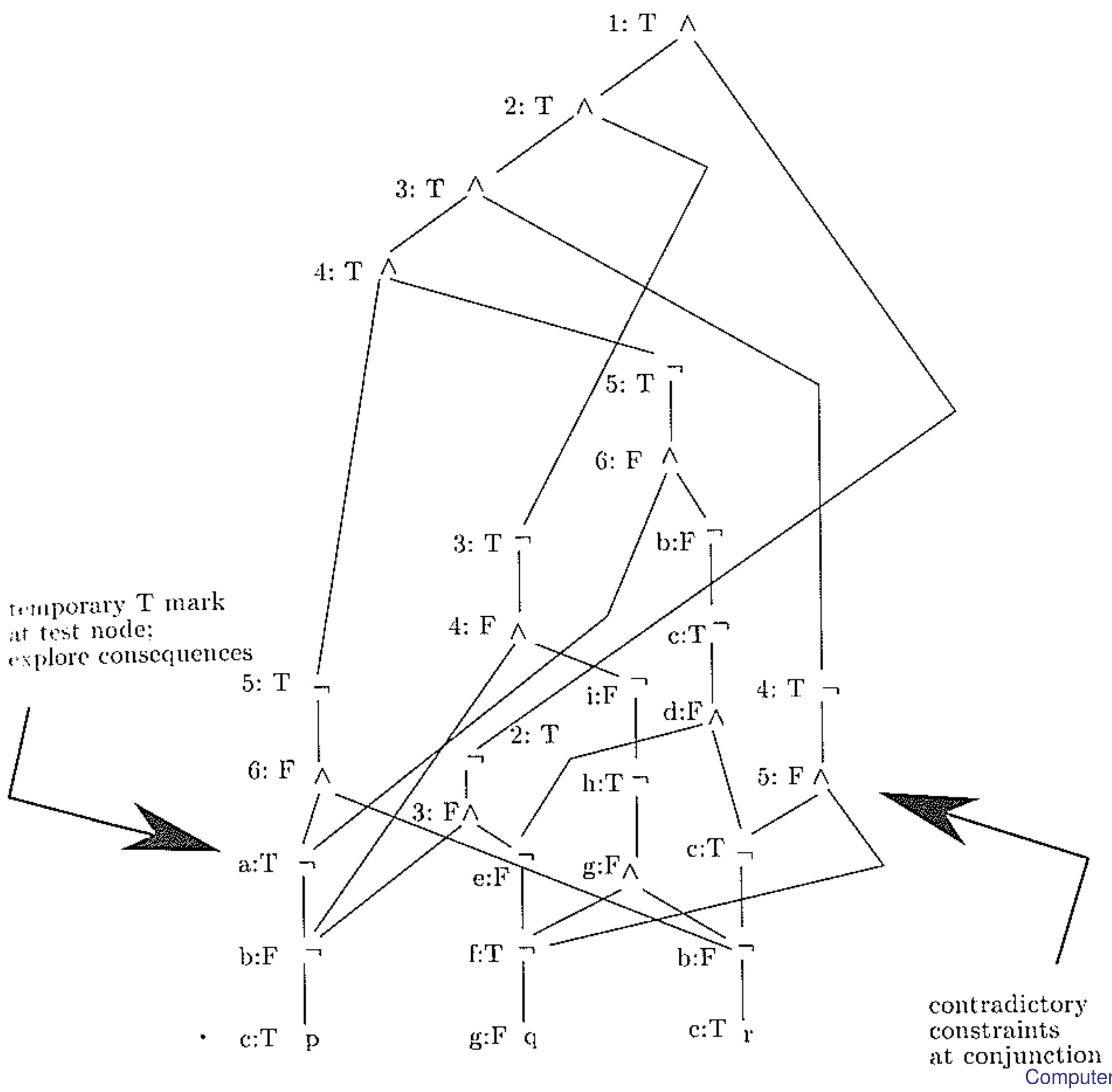
The cubic solver works as follows:

1. Pick an umakred node $n$, where the linear solver got stuck:
2. determine which *temporary* marks are forced by adding mark $T$ only to $n$
3. determine which *temporary* marks are forced by adding mark $F$ only to $n$
4. If steps 2 and 3 find contradictory constraints, the algorithm reports unsatisfiability and stops.

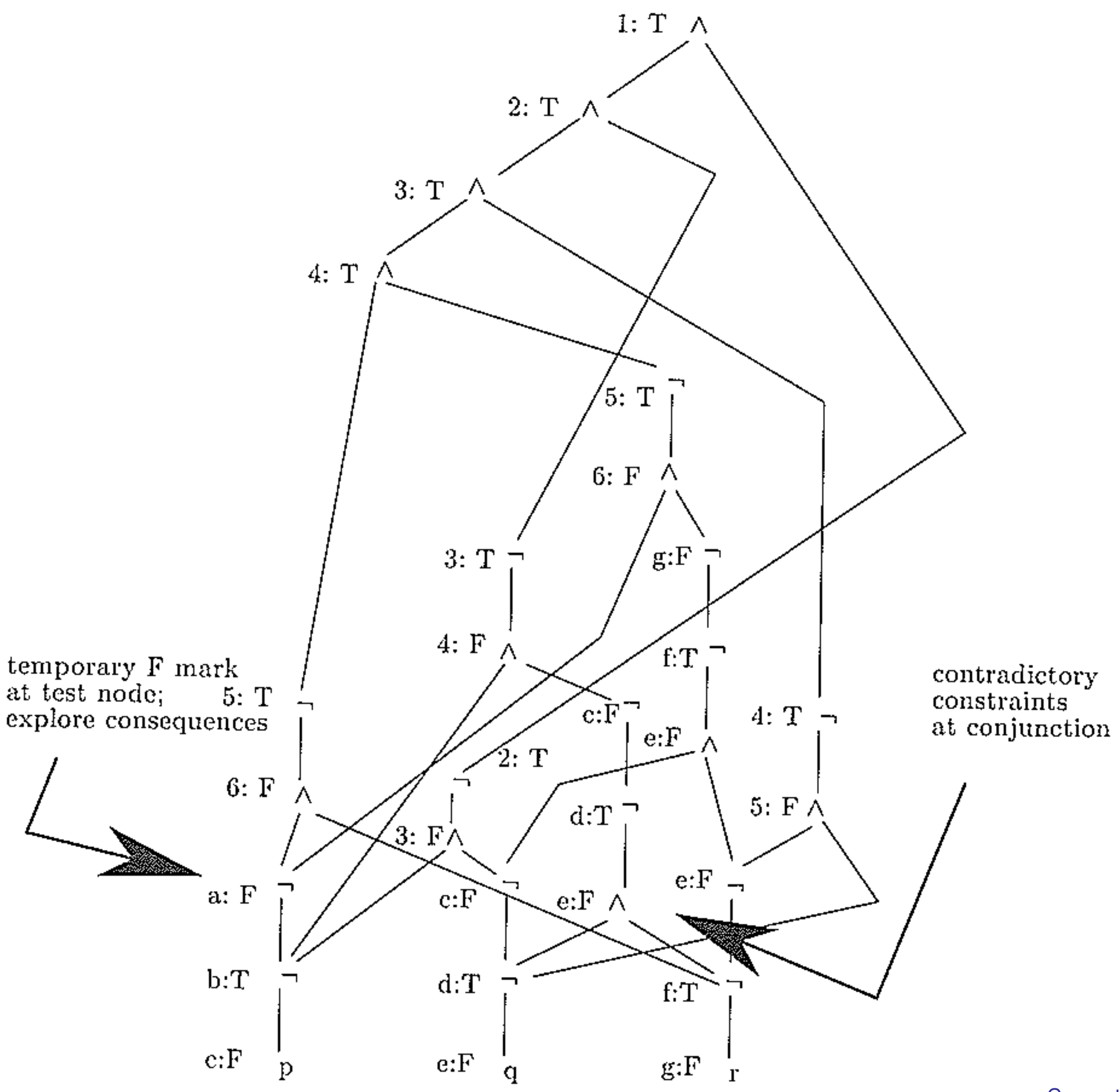
# SAT Solving - A Cubic Solver

5. Nodes that received the same mark in both runs, receive a *permanent* mark
6. If there exists an unmarked node, then go to step 1.
7. We continue this until either
   - find contradictory permanenet markes
   - complete witness to satisfiability, or
   - we have tested all currently unmarked nodes in this manner without detecting any shared marks (no solution).

# SAT Solving - A Cubic Solver



# SAT Solving - A Cubic Solver



## SAT Solving - A Cubic Solver

The cubic complexity in the size of the DAG is due to:

- one factor of the linear solver for 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.