

Horn Clauses in Propositional Logic

Notions of complexity:

In computer science, the efficiency of algorithms is a topic of paramount practical importance.

- The best known algorithm for determining the satisfiability of a set of wff's in propositional logic has worst-case time complexity $\Theta(2^n)$, with n the size of the formula. (The size is length of the string representing the formula.)
- Resolution, Sequent inference, and semantic tableaux, among others, have time complexity $\Theta(2^n)$ in the worst case.
- While testing for satisfiability in DNF, or for validity in CNF, may be performed efficiently, translation of an arbitrary wff to either of these forms has time complexity $\Theta(2^n)$ in the worst case. Furthermore, the new formula may even grow exponentially in size relative to the original one.
- The situation is summarized in the table below.

	CNF	DNF	General
Satisfiability	$\Theta(2^n)$	$\Theta(n)$	$\Theta(2^n)$
Tautology	$\Theta(n)$	$\Theta(2^n)$	$\Theta(2^n)$
Conversion of an arbitrary wff to ..	$\Theta(2^n)$	$\Theta(2^n)$	–

Q: Has it been proven that no better algorithms exist?

A: No.

Q: Is it likely that someone will discover more efficient algorithms sometime soon?

A: No.

The problem of testing satisfiability of an arbitrary wff in propositional logic belongs to class of problems termed NP-complete, denoted NPC. (NP = nondeterministic polynomial.)

This class contains hundreds, if not thousands, of important problems

- Scheduling problems
 - Resource-allocation problems
 - Search problems
 - Computational-geometry problems
-
- If one of these problems has a solution which is better than $\Theta(2^n)$ in the worst case, then they all do.
 - Many researchers have been working on these problems for many years, without finding such algorithms.
 - Problem which are NPC (or which are at least that difficult – the so-called NP-hard problems) are often termed *computationally intractable*.

Thus, the above table has the true characterization as follows:

	CNF	DNF	General
Satisfiability	NPC	$\Theta(n)$	NPC
Tautology	$\Theta(n)$	NPC	NPC
Conversion of an arbitrary wff to ..	$\Theta(2^n)$	$\Theta(2^n)$	–

Note: The conversion of an arbitrary wff to CNF and DNF will cause the formula to grow in size exponentially, in the worst case. No theory is ever going to change this, hence the $\Theta(2^n)$ entries in the above table.

Q: Given that logical inference is such an important issue in computer science, which other options are available?

A: NPC is a *worst case* characterization. One can look for important classes of formulas which admit more efficient solutions.

1. One can look for algorithms which perform well (statistically) over certain distributions of formulas. Often, the formulas which result in worst-case performance are “oddball,” and are unlikely to occur in practical applications.
2. When addressing a specific application involving satisfiability or logical inference, one can look for algorithms which perform well on the formulas which arise in that context. Such classes are often defined by other structures (*e.g.*, graphs) arising from the application.
3. One can look for mathematically interesting, yet practical, classes of formulas which admit tractable inference.

In these slides, approach 3 will be followed, exploring the class of formulas known as Horn clauses.

Notions of deduction:

The type of deduction which we have looked at so far falls into the following general category:

Given:

- (a) A set Φ of formulas; and
- (b) A goal formula φ .

Determine:

whether $\Phi \models \varphi$ holds.

This may be performed by testing whether

- $\Phi \cup \{\neg\varphi\}$ is unsatisfiable, or
- $\forall \{\neg\psi \mid \psi \in \Phi \cup \{\neg\varphi\}\}$ is a tautology.

The idea is the same in either case:

- We are given the candidate conclusion, as well as the hypotheses, and we conduct a test with a yes/no result.

We might consider the following more comprehensive deduction problem:

Given:

A set Φ of formulas;

Determine

The set of all formulas φ for which $\Phi \models \varphi$ holds.

This is extremely ambitious. However, there is a very important class of formulas for which this set φ may be obtained.

Basic notions:

Working within the class of Horn formulas, both of the following desiderata may be realized:

- Tractable inference ($\Theta(n)$ with appropriate algorithms).
- The ability to compute all consequences of a set of clauses.

Definition: A literal is *positive* if it is a proposition name. A literal is *negative* if it is the complement of a proposition name.

Definition: A *Horn clause* is a clause which contains at most one positive literal. The general format of such a clause is thus as follows:

$$\neg A_1 \vee \neg A_2 \vee \dots \vee \neg A_n \vee B$$

This may be rewritten as an implication:

$$(A_1 \wedge A_2 \wedge \dots \wedge A_n) \rightarrow B$$

A *Horn formula* is a conjunction of Horn clauses.

There are various special cases of Horn clauses.

1. If there are no negative literals, the clause consists of a single positive literal, which is called a *fact*.

Examples from the blocks world:

On[P1,B1]
On[B2,B1]

2. If there are both positive and negative literals, the clause is called a *rule*.

$$(A_1 \wedge A_2 \wedge \dots \wedge A_n) \rightarrow B$$

Example from the blocks world:

On[P1,B1] \wedge On[B1,B2] \rightarrow On_table[P2]

3. If there are only negative literals, the clause is called a *compound negation*. It takes the following form:

$$(A_1 \wedge A_2 \wedge \dots \wedge A_n) \rightarrow \perp$$

Examples from the blocks world.

On[P1,B1] \wedge On[P1,B2] $\rightarrow \perp$
On[P1,B1] \wedge On[P2,B1] $\rightarrow \perp$
On[P1,B1] \wedge On_table[P1] $\rightarrow \perp$
On[P1,P1] $\rightarrow \perp$

4. The empty clause \perp is also a Horn clause.

Rule-based systems:

Facts and rules can together be used to deduce new information. This format has been widely used in so-called *rule-based expert systems*, which were popular from the mid-1970's to the mid 1980's. Here are a few examples of rules from such systems:

MYCIN: MYCIN is an expert system which was designed to assist physicians in the diagnosis of and prescription of treatment for infectious blood disease.

Example of a rule from MYCIN:

If:

1. The site of the culture is blood.
2. The stain of the organism is gramneg.
3. The morphology of the organism is rod.
4. The aerobicity of the organism is anerobic.
5. The portal of entry of the organism is GI.

Then:

- There is strongly suggestive evidence (0.9) that the identity of the organism is bacteroides.

MYCIN is the "granddaddy" of all rule-based expert systems, but was never used in a practical clinical setting because of its *brittleness*.

R1: R1 is a rule-based expert system which was designed by Digital Equipment Corporation (DEC) in the late 1970's for configuration of VAX computer systems.

Examples of a rule from R1:

If:

1. The most current active component is distributing massbus devices.
2. There is a single-port disk drive that has not been assigned to a massbus.
3. There are no unassigned single-port disk drives
4. The number of devices that each massbus should support is known.
5. There is a massbus that has been assigned at least one disk drive and that should support additional drives.
6. The type of cable needed to connect the disk drive to the previous device on the massbus is known.

Then:

- Assign the disk drive to the massbus.

Unlike MYCIN, R1 was a commercial success, partly because the domain was sufficiently restricted.

R1 also incorporated the RETE matching algorithm, which provided a very efficient method for matching rules to applicable facts.

General notions of rule-based systems:

- Nowadays, the rule-based approach is regarded as too limited, by itself, for most intelligent applications.
- The expert systems of today make use of many other technologies, such as case-based reasoning and intelligent agents.
- Nonetheless, rule-based components are still an important component of the systems of today.
- (First-order) Horn-clause inference also forms the basis of programming languages such as Prolog.

A toy example of an expert system:

An automobile-problems system.

Abbrev.	Meaning
EGG	Engine is getting gas.
ETO	Engine turns over.
ETON	Engine does not turn over.
LW	The lights work.
LWN	The lights do not work
FT	Fuel in the tank.
FC	Fuel in the carburetor.
TL	Temperature is very low.
TLN	Temperature is not very low.
MW	Motor warmer in operation.
MWN	Motor warmer not in operation.
PBAT	Problem with the battery.
PSTM	Problem with the starter motor.
PIGN	Problem with the ignition.
PTMP	Problem with low temperature.

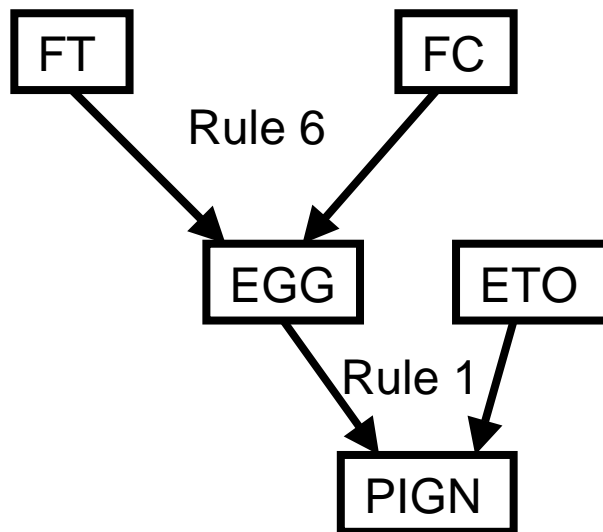
Rule clauses:

1. $EGG \wedge ETO \rightarrow PIGN$
2. $ETON \wedge LWN \wedge TL \wedge MWN \rightarrow PTMP$
3. $ETON \wedge LWN \wedge MW \rightarrow PIGN$
4. $ETON \wedge LWN \wedge TLN \rightarrow PIGN$
5. $ETON \wedge LW \rightarrow PSTM$
6. $FT \wedge FC \rightarrow EGG$

Suppose that the following facts are given:

FT, FC, TL, MW, ETO

The system may then diagnose the problem as an ignition problem, as illustrated by the following graph.



Note that:

- The system reasons by "firing" rules based upon known facts.
- These firings generate new facts.
- The system can provide an explanation of its reasoning process by noting how the rules fired.

The role of compound negations:

Compound negations cannot be used to deduce new facts. However, they may be used to detect inconsistent states.

Example: The statements MW and MWN are mutually exclusive. The full representation of this fact is recaptured by the following sentence.

$$(MW \leftrightarrow \neg MWN)$$

This is equivalent to

$$(MW \rightarrow \neg MWN) \wedge (\neg MWN \rightarrow MW)$$

which is equivalent to the two clauses:

$$(\neg MW \vee \neg MWN) \wedge (MWN \vee MW)$$

The first clause is equivalent to

$$(MW \wedge MWN) \rightarrow \perp$$

The second clause is not Horn, and illustrates the fundamental limitation of the Horn representation: positive disjunction is not representable.

Nonetheless, the knowledge that MW and MWN cannot occur simultaneously is sufficient for this and many other systems. The compound negation may be used as check for inconsistency in the knowledge base.

The Closed-World Assumption:

Often, information is presented with the intent that statements which are not explicitly identified to be true must be false.

Example: The list of students for this class contains 41 names. The presumption is that students whose names are not on this list are not enrolled in the course.

In knowledge representation, this idea is termed the *semantic closed-world assumption* or *semantic CWA* for short. In words it is formulated as follows:

Semantic CWA: The only propositions which are taken to be true are those for which can be established that they are true from the given information. All others are taken to be false.

From a more formal point of view, the semantic CWA proposes that the following meta-rule be followed:

Formal Semantic CWA: Let Φ be a set of wff's, and let A be a single proposition name (a positive literal). Then,

$$\text{If } \Phi \not\models A \text{ then } \Phi \models \neg A.$$

Clarification:

Let Φ be a set of wff's, and let A be a single proposition name (a positive literal). Regarding the question,

“Does Φ entail A ?”

there are three possibilities:

(a) $\Phi \models A$

- A is true in every model of Φ .

(b) $\Phi \models \neg A$

- A is false in every model of Φ .

(c) Neither $\Phi \models A$ nor $\Phi \models \neg A$ holds.

- A is true in some models of Φ and false in others.

- The semantic CWA attempts to exclude condition (c), by forcing it to be equivalent to condition (b).

Question: When does this work just fine, and when does it cause serious problems?

Short answer: Modulo some reasonable adjustments, It works just fine when Φ is a set of Horn clauses, and it fails otherwise.

This issue will now be examined in some detail.

Definition: Let Φ be any set of propositional formulas over a propositional logic $L = (P, C, A)$. Define the set of *atomic consequences* of Φ to be the following set:

$$\text{AtCons}(\Phi) = \{ A \in P \mid \Phi \models A \}.$$

The set Φ has the *least model property* if the interpretation

$$v_{\perp}^{\Phi}(A) : \begin{cases} 1 & \text{if } A \in \text{AtCons}(\Phi). \\ 0 & \text{if } A \notin \text{AtCons}(\Phi). \end{cases}$$

is a model of Φ .

Not every set of wff's has the least model property. For example, let $\Phi = \{A \vee B\}$. Then

$$\text{AtCons}(\Phi) = \emptyset$$

Yet the associated function

$$v_{\perp}^{\Phi}(A) : A \mapsto 0, B \mapsto 0$$

is not a model of Φ , since at least one of A or B must be true. However, the following remarkable fact is true.

Theorem: Let Φ be a satisfiable set of Horn clauses. Then Φ has the least model property. \square

Before looking at how to compute the least model, we take an alternate view at its characterization, which justifies the terminology.

Definition: Let $v_1, v_2 \in \text{Interp}(L)$. Define the relation \preceq on $\text{Interp}(L)$ by

$$v_1 \preceq v_2 \text{ iff for each } A \in P, v_1(A) \leq v_2(A).$$

In other words, whenever v_1 is true, so too is v_2 .

Observation: The relation \preceq forms a partial order on $\text{Interp}(L)$.

Proposition: Let Φ be a set of clauses with the least model property. Then

$$v_{\perp}^{\Phi} = \inf_{\preceq} \{v \mid v \in \text{Mod}(\Phi)\}. \quad \square$$

Here \inf_{\preceq} means the least element under the ordering \preceq .

Thus, the least model of Φ , when it exists, is the model whose true propositions are precisely those which are true in all models of Φ .

Constructing the least model of a set of Horn clauses:

The construction proceeds by starting with the facts, and then repeatedly *firing* the rules whose left-hand sides match existing facts. The final set of facts so obtained defines the least model. The compound negations do not contribute to this process, other than as a check of satisfiability.

Example: Let $P = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$, and let $\Phi = \{ A_1, A_2, (A_1 \wedge A_2) \rightarrow A_3, A_3 \rightarrow A_4, A_3 \rightarrow A_5, (A_5 \wedge A_6) \rightarrow A_7 \}$.

Then the least model v_{\perp} is defined by

$$v_{\perp}(A_i) = \begin{cases} 1 & \text{if } i \leq 5. \\ 0 & \text{if } i > 5. \end{cases}$$

If the constraint $(A_4 \wedge A_6) \rightarrow \perp$ is added to Φ , the least model does not change. A compound negative does not change the least model.

However, if the constraint $(A_4 \wedge A_5) \rightarrow \perp$ is added to Φ , the entire set becomes inconsistent.

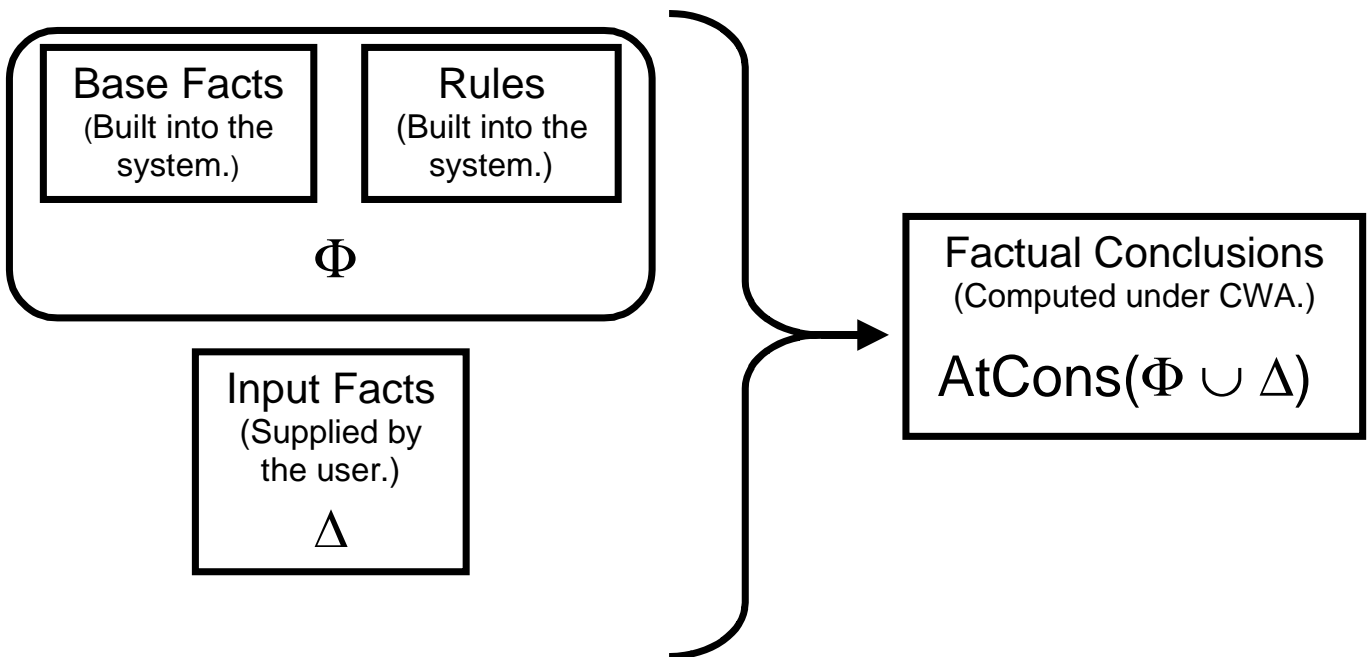
Effect of compound negation: A compound negation does not change the minimal model; it can only cause the entire set of clauses to become inconsistent.

Genericity:

One might propose that the property of having a least model is a characterization of Horn formulas. However, this is not the case, as the following example illustrates.

Example: The formula $A_1 \rightarrow (A_2 \vee A_3)$ has the least model in which all three propositions are false, yet it is clearly not a Horn formula.

A better way to think of the situation of constructing the least model is as follows:



- There are facts and rules built into the base system (Φ), which do not change.
- There are external facts supplied by the user (Δ), which may vary from instance to instance of use.
- The goal is to compute the least model of this combination $v_{\perp}^{\Phi \cup \Delta}$, defined by $\text{AtCons}(\Phi \cup \Delta)$.

- Under this setup, the rules define a least model for any set of facts, and not just certain ones. This is formalized as follows.

Definition: A set Φ of clauses *admits generic assignments* if for every set Δ of positive literals, if $\Phi \cup \Delta$ is satisfiable, then it has a least model.

Example: Note that $\Phi = \{A_1 \rightarrow (A_2 \vee A_3)\}$ does not admit generic assignments, since if we define $\Delta = \{A_1\}$, the resulting set $\Phi \cup \Delta = \{A_1 \rightarrow (A_2 \vee A_3), A_1\}$ does not have a least model.

Lemma: Let φ be a satisfiable clause. Then $\{\varphi\}$ admits generic assignments iff φ is Horn.

Proof: If φ is a Horn clause, then $\{\varphi\} \cup \Delta$ is a set of Horn clauses for any set Δ of atoms. Thus, if satisfiable, $\{\varphi\} \cup \Delta$ has a least model, and so $\{\varphi\}$ admits generic assignments.

Conversely, suppose that φ is not Horn. Then it must have the form

$$(p_1 \vee p_2 \vee \dots \vee p_m \vee \neg q_1 \vee \neg q_2 \vee \dots \vee \neg q_n)$$

It must be that $m \geq 2$, but n may be 0. Now define

$$\Delta = \{q_1, q_2, \dots, q_n\}.$$

It is easy to see that $\{\varphi\} \cup \Delta$ has no generic assignment. \square

This result applies not only to a single Horn clause, but to any set of such clauses:

Theorem: Let Φ be a set of wff's. Then Φ admits generic assignments iff it is equivalent to a set of Horn clauses.

Proof: The proof is based upon the above lemma, but requires some that some rather picky details be heeded. The complete proof is thus not presented in these slides. See the reference by Makowsky noted at the end of these slide for the details. \square

Key conclusion: To be able to take

- (a) a set Δ of input facts, and
- (b) a set Φ of constraints,

and deduce a least or canonical possible world from these, it is necessary and sufficient that Φ be a set of Horn clauses.

Important: Here we are in effect deducing a single state or possible world, and not a formula which defines a set of possible worlds.

We now turn to the question of how to implement this sort of computation efficiently.

Resolution and Horn clauses:

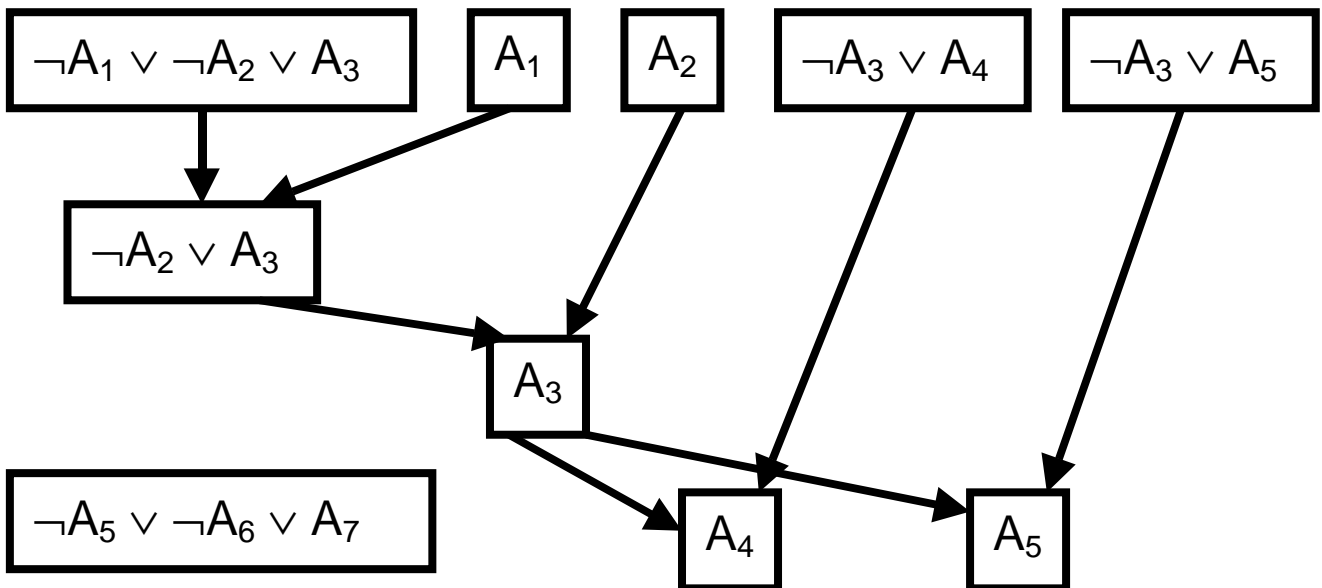
Recall that for general sets of clauses, the best algorithms have complexity $\Theta(2^n)$, where n is the size of the input. We will now establish that the situation is much better in the case of Horn clauses.

Example: Let $P = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$, and let $\Phi = \{A_1, A_2, (A_1 \wedge A_2) \rightarrow A_3, A_3 \rightarrow A_4, A_3 \rightarrow A_5, (A_5 \wedge A_6) \rightarrow A_7\}$.

The set Φ may be rewritten as

$\{A_1, A_2, \neg A_1 \vee \neg A_2 \vee A_3, \neg A_3 \vee A_4, \neg A_3 \vee A_5, \neg A_5 \vee \neg A_6 \vee A_7\}$.

Here is a resolution on these clauses:



Note the following:

- The positive unit clauses which are the result define exactly the least model of the set of clauses. This is not a refutation, but a direct proof. *Thus, we are computing the least model directly. We do not need to conjecture at a conclusion and then test for its satisfaction, as with ordinary deduction.*
- Every resolution is a *positive unit resolution*; that is, a resolution in which one clause is a *positive unit clause* (*i.e., a proposition letter*).
- At each resolution, the input clause which is not a unit clause is a logical consequence of the result of the resolution. (Thus, this input clause may be deleted upon completion of the resolution operation.)
- Following this deletion, the size of the database (the sum of the lengths of the remaining clauses) is one less than it was before the operation.)
- Thus, if n is the size of the database, then at most n positive unit resolutions may be performed on it.

A closer look at positive unit resolution:

Recall that in unit resolution, at least one of the resolvents must be a literal. A *unit refutation* is a resolution refutation of a set of clauses such that every resolution operation is a unit resolution.

Further, a unit refutation is *positive* if the unit clause is a positive literal; *i.e.* a proposition name.

In general, unit resolution is not complete. That is, there are sets of clauses which are unsatisfiable, but which do not admit unit refutations. This cannot happen for sets of Horn clauses.

Theorem: Let Φ be an unsatisfiable set of Horn clauses. Then there is a positive unit refutation of Φ . \square

Unit resolution has some other key properties:

Observation: Let φ be any clause, and let ℓ be a literal which is resolvable with φ . Let $R[\varphi, \ell]$ denote the resolvent of φ and ℓ . Then $R[\varphi, \ell] \models \varphi$. \square

Example: Let $\varphi = \neg A_1 \vee \neg A_2 \vee A_3$, and let $\ell = A_1$. Then $R[\varphi, \ell] = \neg A_2 \vee A_3$.

This leads to an important simplification within the context of unit resolution.

Simplification: When performing the unit resolution of φ and ℓ , and the clause $R[\varphi, \ell]$ is added to the database of clauses, the clause φ may be removed.

Example: In the above, instead of just adding $\neg A_2 \vee A_3$ to the set of clauses, we may replace $\neg A_1 \vee \neg A_2 \vee A_3$ with it.

Define the *size* of a clause to be the number of literals it contains. Define the size of a set of clauses to be the sum of the sizes of its elements.

Example: The size of $\{ A_1, A_2, \neg A_1 \vee \neg A_2 \vee A_3, \neg A_3 \vee A_4, \neg A_3 \vee A_5, \neg A_5 \vee \neg A_6 \vee A_7 \}$ is 12.

Observation: In the unit resolution of Horn clauses, if the above simplification is employed, the size of the database decreases by one at each step. Thus, the total number of resolutions is bounded by the size of the input set of clauses.

Finally, we have that positive unit resolution computes exactly what we need:

Theorem: Let Φ be a set of Horn clauses. Let $\text{PRes}(\Phi)$ be the closure of Φ under the operation of unit resolution. Then the set of all atoms in $\text{PRes}(\Phi)$ is precisely $\text{AtCons}(\Phi)$. In other words, positive unit resolution may be used to compute the least model of Φ directly. \square

An efficient algorithm for Horn clause inference:

While the number of steps in a positive unit resolution of Horn clauses is bounded by the length of the input, it is not quite true that the algorithm to implement this strategy is necessarily linear in the size of the input.

To ensure the linearity, it is necessary to be rather careful about how data structures are chosen.

On the following slide, an algorithm which will run in linear time under certain circumstances is shown.

- The algorithm supports “extended” Horn clauses of the form

$$A_1 \wedge A_2 \wedge \dots \wedge A_m \rightarrow B_1 \wedge B_2 \wedge \dots \wedge B_n$$

This clause is equivalent to the n clauses:

$$\begin{aligned} A_1 \wedge A_2 \wedge \dots \wedge A_m &\rightarrow B_1 \\ A_1 \wedge A_2 \wedge \dots \wedge A_m &\rightarrow B_2 \\ &\vdots \\ A_1 \wedge A_2 \wedge \dots \wedge A_m &\rightarrow B_n \end{aligned}$$

Such extended clauses increase the efficiency, since only one rule “firing” can trigger is needed to change the values of B_1 through B_n .

The algorithm will run in linear time if:

- Data structures indexed by proposition names may be accessed in constant time. (This is possible if the proposition names are number in a range (e.g., 1..n), so that array lookup is the access operation.
- If propositions are accessed by name, then a symbol table is necessary, and the algorithm will run in time $\Theta(n \cdot \log(n))$.

Details will be covered in class.

```

type clause = record
  antecedent_count: natural_number;
  {Initialized to the number of distinct antecedents.}
  consequent: list_of(ref atom)
  {Empty list implies right-hand side = false.}
end; {clause}
type atom = record
  name: atom_name;
  clause_list: list_of(ref clause);
  truth_value: Boolean {Initialized to false.}
end; {atom}

var queue: clause_queue;
    clauses: list_of(clause);
    consistent: Boolean;

procedure initialize();
  var c: clause;
  begin
    consistent := true;
    for each c in clauses with c.antecedent_count = 0 do
      enqueue(clause_queue,c)
    end foreach;
  end; {initialize}

procedure main();
  var c,v: ref clause;
      x: atom;
  begin
    initialize();
    while ((not empty(clause_queue)) and consistent) do
      dequeue(clause_queue,c);
      if empty(c.consequent)
        then consistent := false
      else
        for each x in c.consequent
          if x.truth_value = false
            then
              x.truth_value = true;
              for each v in x.clause_list do
                report_antecedent_count_decrement(v);
              end foreach;
            end if;
          end foreach;
        end if;
      end while;
    end; {main}

procedure report_antecedent_count_decrement(v: ref clause);
  begin
    v.antecedent_count := v.antecedent_count - 1;
    if v.antecedent_count = 0 then
      enqueue(clause_queue,v)
    end if
  end; {report_antecedent_count_decrement}

```

For more information:

For more ideas on the notion of genericity and the importance of Horn clauses in computer science, consult:

Makowsky, J. A., "Why Horn Formulas Matter in Computer Science: Initial Structures and Generic Examples," *Journal of Computer and System Sciences*, **34**, pp. 266-292, 1987.

For more information on fast inference for Horn clauses, consult:

Dowling, William. F., and Gallier, Jean A., "Linear-time satisfiability of propositional Horn formulae," *Journal of Logic Programming*, **3**, pp. 267-284, 1984.