Computational Logic

Implementations of Herbrand's Theorem

Damiano Zanardini

UPM European Master in Computational Logic (EMCL)
School of Computer Science
Technical University of Madrid
damiano@fi.upm.es

Academic Year 2009/2010

Introduction

General idea

- we have seen that, in order to prove the unsatisfiability of a set of clause, it is enough to find an unsatisfiable finite set of ground instances of the clauses
- practically thinking, we look for a method to generate ground instances of the clauses and prove their unsatisfiability
- *level-saturation*: generate incrementally sets S_i of ground instances by going through the *levels* $H_0, H_1, .., H_k, ..$ of the Herbrand Universe
- for every set S_i, transform it in order to find a *contradiction*, i.e, to prove that it is unsatisfiable
- if the contradiction cannot be found, then generate new instances and repeat
- this method relies on the *contradiction lemma*

Introduction

Generation

- the technique used for checking $SAT(S)$ is independent of the technique for generating S
- we can suppose that all methods presented in this section generate S in the same way (with level-saturation)

Complexity

- note that deciding $SAT(S)$ is the well-known $\mathcal{NP}$-complete SAT problem

Introduction (side-topic)

Where wa are, where we are going to...

- in the field of *propositional logic* (no variables)
- to decide the satisfiability of a propositional formula is the SAT problem, which is extremely important in computer science
  - verification of circuits and processors
  - timetables, scheduling, calendars, optimization...
  - properties of programs (e.g., termination or the correctness of some operations)
- due to this, there has been so far so much research on algorithms for solving SAT:
  - because it has so many applications $\rightsquigarrow$ `http://www.satlive.org`
  - and also because of $\mathcal{NP}$-completeness

Introduction (side-topic)

Example: the *null* value in Java

in $\longrightarrow$
out $\longrightarrow$

```
a = new MyClass();
c = b;
...
a.f = 1;
c.g = d.m(2);
```

- the goal is to guarantee that a `NullPointerException` will *never* (not in any possible execution) be thrown
- (without explicit controls as `if (x!= null) {...}`)
- this can be formalized by means of *propositional formulæ*, where one or more propositions correspond to every program variable

v_{in}, v_{out} $\qquad v_{pp} = \mathbf{t}$ if we know that v is not *null* at pp

$\qquad\qquad\quad v_{pp} = \mathbf{f}$ if we do not know whether v is *null* or not at pp

Introduction (side-topic)

And if we were in first-order

- logic programming
- automated theorem proving
- rewriting systems
- artificial intelligence
- semantic web (*description logics*)
- verification of criptographic protocols

Therefore

Computational logic proposes automatic tecniques for efficiently solving some of these problems

Back to the introduction

Lemma (contradiction)

A formula F is unsatisfiable iff it is possible to derive a contradiction from it: $[F] \vdash G \wedge \neg G$

Proof.

❶ $[F] \vdash G \wedge \neg G$ iff $\vdash F \rightarrow G \wedge \neg G$ (deduction theorem)

❷ $\vdash F \rightarrow G \wedge \neg G$ iff, for every interpretation, (1) $I(F) = \mathbf{f}$; or (2) $I(F) = \mathbf{t}$ and $I(G \wedge \neg G) = \mathbf{t}$

❸ $I(G \wedge \neg G) = \mathbf{f}$ for every I, so that $\vdash F \rightarrow G \wedge \neg G$ iff $I(F) = \mathbf{f}$ for every I

❹ $\vdash F \rightarrow G \wedge \neg G$ iff F is unsatisfiable (by ❸)

❺ $[F] \vdash G \wedge \neg G$ iff F is unsatisfiable (by ❶ and ❹)

Back to the introduction

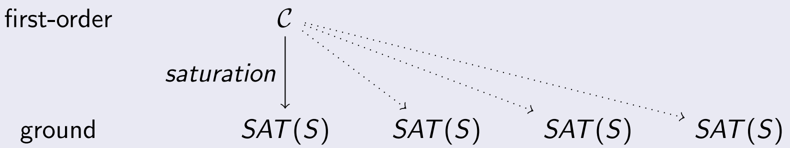
Using level-saturation: for a set of clauses $\mathcal{C}$

$i = 0$;
$S = \emptyset$;
while $(SAT(S))$
  $H_i =$ the i-th level of $H(\mathcal{C})$
  $X = \{C' \mid C \in \mathcal{C}$ and C' is obtained from C
    by replacing variables with terms in $H_i\}$;
  $S = S \cup X$;
  $i = i + 1$;

Back to the introduction

The general picture

- three methods for checking satisfiability
  - Gilmore
  - Davis-Putnam
  - ground Resolution
- all of them use level-saturation
- the difference is how they decide the satisfiability of ground instances
  - that is, how they try to deduce a contradiction



Gilmore's method (1960)

The techniques

- instances are generated by level-saturation
- a method for verifying $SAT(S)$ is needed
- Gilmore chose one: *multiplication*

Multiplication

- put S in Disjunctive Normal Form ($DNF(S)$)
  - disjunction of conjunctions of literals, ex. $(p \wedge q) \vee r \vee (q \wedge \neg r)$
- search for a contradiction in every conjunction

a: if the contradiction is found *everywhere*, then the set is unsatisfiable

b: if there exists a conjunct which does not contain a contradiction (see lemma Gil-1), then the set is satisfiable

Gilmore's method (1960)

Lemma (Gil-1)

Given a conjunction F of propositions, a contradiction can be derived iff it is a subformula of F

Lemma ($DNF(F)$)

For every (quantifier-free) formula F, $DNF(F)$ exists and is equivalent to F

Gilmore's method (1960)

Theorem

A propositional formula F is unsatisfiable iff DNF(F) contains a contradiction in every conjunct

Proof.

1. F is unsatisfiable iff $DNF(F)$ is (Lemma $DNF(F)$)
2. $DNF(F) = D_1 \vee .. \vee D_n$ is unsatisfiable iff we can derive a contradiction from it (contradiction lemma)
3. $DNF(F)$ is unsatisfiable iff every D_i (conjunction of literals) is
4. $DNF(F)$ is unsatisfiable iff every D_i contains a contradiction (Lemma Gil-1)
5. F is unsatisfiable iff every D_i of $DNF(F)$ contains a contradiction (by ❶ and ❹)

Gilmore's method (1960)

How to compute $DNF(F)$

Not surprisingly, we use the same rules as for $CNF(F)$, but we often conceptually *change* the direction

- connectives
  $\vdash (F \rightarrow G) \leftrightarrow (\neg F \vee G)$
  $\vdash (F \leftrightarrow G) \leftrightarrow (F \rightarrow G) \wedge (G \rightarrow F)$
- De Morgan
  $\vdash \neg(F \wedge G) \leftrightarrow \neg F \vee \neg G$ $\quad$ $\vdash \neg(F \vee G) \leftrightarrow \neg F \wedge \neg G$
- distributivity of $\wedge$ and $\vee$
  $\vdash F \wedge (G \vee H) \leftrightarrow (F \wedge G) \vee (F \wedge H)$
  $\vdash F \vee (G \wedge H) \leftrightarrow (F \vee G) \wedge (F \vee H)$

Gilmore's method (1960)

Example (null pointers)

in $\longrightarrow$ / out $\longrightarrow$

```
a = new MyClass();
c = b;
...
a.f = 1;
c.g = d.m(2);
```

- what the first two lines do: $F = a_{out} \wedge (b_{in} \rightarrow (b_{out} \wedge c_{out})) \wedge (d_{in} \rightarrow d_{out})$
- some specific input information (saying that `b` and `d` are not null at the beginning): $G = b_{in} \wedge d_{in}$
- the correctness condition (that no exceptions are thrown): $H = a_{out} \wedge c_{out} \wedge d_{out}$
- the deductive structure:

$$\{F, G\} \vdash H \qquad \text{(iff} \quad UNSAT(\{F, G, \neg H\}))$$

Gilmore's method (1960)

Example (null pointers)

$$a_{out} \wedge (b_{in} \rightarrow (b_{out} \wedge c_{out})) \wedge (d_{in} \rightarrow d_{out}) \wedge b_{in} \wedge d_{in} \wedge (\neg a_{out} \vee \neg c_{out} \vee \neg d_{out})$$

$$\begin{array}{rcl} DNF(F \wedge G \wedge \neg H) & = & (a_{out} \wedge b_{in} \wedge d_{in} \wedge \neg b_{in} \wedge \neg d_{in} \wedge \neg a_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge \neg b_{in} \wedge \neg d_{in} \wedge \neg c_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge \neg b_{in} \wedge \neg d_{in} \wedge \neg d_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge \neg b_{in} \wedge d_{out} \wedge \neg a_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge \neg b_{in} \wedge d_{out} \wedge \neg c_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge \neg b_{in} \wedge d_{out} \wedge \neg d_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge b_{out} \wedge c_{out} \wedge \neg d_{in} \wedge \neg a_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge b_{out} \wedge c_{out} \wedge \neg d_{in} \wedge \neg c_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge b_{out} \wedge c_{out} \wedge \neg d_{in} \wedge \neg d_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge b_{out} \wedge c_{out} \wedge d_{out} \wedge \neg a_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge b_{out} \wedge c_{out} \wedge d_{out} \wedge \neg c_{out}) \\ & \vee & (a_{out} \wedge b_{in} \wedge d_{in} \wedge b_{out} \wedge c_{out} \wedge d_{out} \wedge \neg d_{out}) \end{array}$$

Gilmore's method (1960)

Another example

$$\begin{array}{rcl} C_1 & = & p(x, f(y)) \vee \neg q(x) \\ C_2 & = & q(a) \\ C_3 & = & \neg p(a, z) \end{array}$$

- $S_0 = \{ \ \ p(a, f(a)) \vee \neg q(a), \ \ \ q(a), \ \ \ \neg p(a, a) \ \ \}$

  $$DNF(S) = (p(a, f(a)) \wedge q(a) \wedge \neg p(a, a)) \vee (\neg q(a) \wedge q(a) \wedge \neg p(a, a))$$

- $S_1 = \left\{ \begin{array}{ccc} p(a, f(a)) & \vee & \neg q(a), \\ p(f(a), f(a)) & \vee & \neg q(f(a)), \\ p(a, f(f(a))) & \vee & \neg q(a), \\ p(f(a), f(f(a))) & \vee & \neg q(f(a)), \\ q(a), & & \\ \neg p(a, a), & & \\ \neg p(a, f(a)) & & \end{array} \right\} \qquad DNF(S) = \ldots$

The method of Davis-Putnam (1960)

General idea

- generate each set S of ground instances by saturation
- simplify S, getting a new set S' by means of four *rules*, until a *contradiction* is detected
- if all possible rules have been applied and no contradiction is detected, then S is satisfiable

The rules

1. tautology rule
2. one-literal rule
3. pure-literal rule
4. splitting rule

The method of Davis-Putnam (1960)

1. Tautology rule

Given a set of ground instances, delete all instances which are *tautologies*

Example

$$\begin{array}{rcl} S & = & \{p,\ q,\ r \vee \neg r\} \\ S' & = & \{p,\ q\} \end{array}$$

clearly, S is satisfiable iff S' is

Lemma (tautology rule)

Since tautologies are always true, eliminating them does not affect satisfiability: the remaining set S' is satisfiable iff S is

The method of Davis-Putnam (1960)

2 One-literal rule

If there is a *unit* instance L in S (i.e., a ground instance which only consists of the literal L), then S' can be obtained iteratively by

- deleting all instances in S which contain L
- deleting $\neg L$ *from* the instances in S which contain $\neg L$

Example

$$\begin{array}{rcl} S & = & \{ \neg p \vee \neg u,\ p \vee q \vee \neg r,\ p \vee \neg q,\ \neg p,\ r,\ u \} \quad \rightsquigarrow \quad [\text{rule on } \neg p] \\ & & \{ q \vee \neg r,\ \neg q,\ r,\ u \} \quad \rightsquigarrow \quad [\text{rule on } \neg q] \\ & & \{ \neg r,\ r,\ u \} \quad \rightsquigarrow \quad [\text{rule on } \neg r] \\ S' & = & \{ \Box,\ u \} \end{array}$$

the *empty clause* $\Box$ (which can be obtained from r or $\neg r$) means that there is a contradiction: S' is unsatisfiable (like S)

The method of Davis-Putnam (1960)

Lemma (one-literal rule)

$S = \{L,\ (L \vee F_1), .., \ (L \vee F_n),\ (\neg L \vee G_1), .., \ (\neg L \vee G_m),\ H_1, .., \ H_p\}$ is unsatisfiable iff $S' = \{G_1, .., \ G_m,\ H_1, .., \ H_p\}$ is

- *provided neither L nor $\neg L$ occur in any H_k*

Proof ($\longrightarrow$).

1. S is unsatisfiable
2. suppose $\{G_1, .., \ G_m,\ H_1, .., \ H_p\}$ is not: then, there exists an interpretation $\mathcal{I}$ which makes all G_j and H_k true
3. if $\mathcal{I}$ also verifies L (it is always possible to find such $\mathcal{I}$), then it verifies all $L \vee F_i$, so that it satisfies the original set
4. contradiction ❷, ❸: $\{G_1, .., \ G_m,\ H_1, .., \ H_p\}$ is unsatisfiable

The method of Davis-Putnam (1960)

Lemma (one-literal rule)

$S = \{L,\ (L \vee F_1), .., \ (L \vee F_n),\ (\neg L \vee G_1), .., \ (\neg L \vee G_m),\ H_1, .., \ H_p\}$ is unsatisfiable iff $S' = \{G_1, .., \ G_m,\ H_1, .., \ H_p\}$ is

- *provided neither L nor $\neg L$ occur in any H_k*

Proof ($\longleftarrow$).

❶ $\{G_1, .., \ G_m,\ H_1, .., \ H_p\}$ is unsatisfiable

❷ suppose S is not: then, there exists an interpretation $\mathcal{I}$ which makes L and all $L \vee F_i$, $\neg L \vee G_j$ and H_k true

❸ $\mathcal{I}$ makes $\neg L$ false, then, since it makes $\neg L \vee G_j$ true, it must make G_j true

❹ $\mathcal{I}$ satisfies $\{G_1, .., \ G_m,\ H_1, .., \ H_p\}$ (by ❸)

❺ contradiction ❷, ❹: S is unsatisfiable

The method of Davis-Putnam (1960)

3 Pure-literal rule

If S contains a *pure* literal L, then S' can be obtained by deleting all instances which contain L

- a literal is pure if it only occurs with one sign (positive or negative)

Example

p is pure is S

$$\begin{array}{rclcl} S & = & \{\ p \vee q,\ p \vee \neg q,\ r \vee q,\ r \vee \neg q\ \} & \leadsto & [\text{rule on } p] \\ & & \{\ r \vee q,\ r \vee \neg q\ \} & \leadsto & [\text{rule on } r] \\ S' & = & \{\ \} & = & \emptyset \end{array}$$

S' is satisfiable (like S)

The method of Davis-Putnam (1960)

Lemma (pure-literal rule)

$S = \{L \vee F_1, .., \; L \vee F_n, .., \; G_1, .., \; G_m\}$ *is unsatisfiable iff* $\{G_1, .., \; G_m\}$ *is*

- *provided* L *is pure and does not appear in any* F_j *or* G_k

Proof ($\longrightarrow$).

1. S is unsatisfiable
2. suppose $\{G_1, .., \; G_m\}$ is not: then, there exists $\mathcal{I}$ which makes all G_j true
3. $\mathcal{I}$ can be found which makes L true: therefore, it satisfies all instances $L \vee F_j$, and therefore S
4. contradiction ❷, ❸: $\{G_1, .., \; G_m\}$ is unsatisfiable

Proof ($\longleftarrow$).

easy because $\{G_1, .., \; G_m\}$ is a subset of the clauses of S

The method of Davis-Putnam (1960)

4 Splitting rule

If S takes the form $\{(L \vee F_1), .., (L \vee F_n), (\neg L \vee G_1), .., (\neg L \vee G_m), H_1, .., H_p\}$, then two sets S' and S'' can be obtained as

- $S' = \{F_1, .., F_n, .., H_1, .., H_p\}$
- $S'' = \{G_1, .., G_m, .., H_1, .., H_p\}$

☞ this rule can be applied on every S, but before we have to try with *one-literal* or *pure-literal*

Example

$$
\begin{aligned}
S &= \{\ p \vee \neg q,\ \neg p \vee q,\ q \vee \neg r,\ \neg q \vee \neg r\ \} \\
S' &= \{\ \neg q,\ q \vee \neg r,\ \neg q \vee \neg r\ \} \\
S'' &= \{\ q,\ q \vee \neg r,\ \neg q \vee \neg r\ \}
\end{aligned}
$$

The method of Davis-Putnam (1960)

Lemma (splitting rule)

$S = \{(L \vee F_1), .., (L \vee F_n), (\neg L \vee G_1), .., (\neg L \vee G_m), H_1, .., H_p\}$ is unsatisfiable iff both $S' = \{F_1, .., F_n, .., H_1, .., H_p\}$ and $S'' = \{G_1, .., G_m, .., H_1, .., H_p\}$ are

- *provided neither L nor $\neg L$ appear in any F_i, G_j or H_k*

Proof ($\rightarrow$).

❶ S is unsatisfiable

❷ suppose at least one between S' and S'' is not: therefore, there exists $\mathcal{I}$ which make all H_k true, and either all F_i or all G_j

❸ if $\mathcal{I}$ makes all F_i true, then it makes all $L \vee F_i$ true. $\mathcal{I}$ can be taken which makes L false, so that it makes all $\neg L \vee G_j$ (and S) true

❹ dual reasoning, in the case $\mathcal{I}$ makes all G_j true

❺ in both cases, contradiction (❷, ❸ or ❷, ❹): both S' and S'' are unsatisfiable

The method of Davis-Putnam (1960)

Lemma (splitting rule)

$S = \{(L \vee F_1), .., (L \vee F_n), (\neg L \vee G_1), .., (\neg L \vee G_m), H_1, .., H_p\}$ is unsatisfiable iff both $S' = \{F_1, .., F_n, .., H_1, .., H_p\}$ and $S'' = \{G_1, .., G_m, .., H_1, .., H_p\}$ are

- *provided neither L nor $\neg L$ appear in any F_i, G_j or H_k*

Proof ($\leftarrow$).

❶ both S' and S'' are unsatisfiable

❷ suppose S is not: therefore, there exists $\mathcal{I}$ which makes all $L \vee F_i$, $\neg L \vee G_j$ and H_k true

❸ if $\mathcal{I}$ makes L true, then it makes $\neg L$ false: since it makes $\neg L \vee G_j$ true, it must make G_j true, so that it satisfies S''

❹ dual: if $\mathcal{I}$ makes L false, then it satisfies S'

❺ in both cases, contradiction (❷, ❸ or ❷, ❹): S is unsatisfiable

The method of Davis-Putnam (1960)

Procedure DP: given S, transform it as follows (YES = satisfiable)

while ($S \neq \emptyset$)
  if (*tautology rule* can be applied) **apply** *tautology rule*
  else
    while (*one-literal rule* can be applied) **apply** *one-literal rule*
    if (S contains literals L and $\neg L$) **return** NO
    if ($S = \emptyset$) **return** YES
    while (*pure-literal rule* can be applied) **apply** *pure-literal rule*
    if (S contains literals L and $\neg L$) **return** NO
    if ($S = \emptyset$) **return** YES
    apply *splitting rule*, **apply** DP to both S' and S''
    if (the result is NO for both S' and S'') **return** NO
    else return YES
return YES

The Resolution method of Robinson

Our inspiration

In the following part of this section, and the next one, we will (sometimes literally) refer to a couple of papers by John Alan Robinson:

- [R63] Theorem-Proving on the Computer. Journal of the ACM, April 1963, 163-174.
- [R65] A Machine-Oriented Logic Based on the Resolution Principle. Journal of the ACM, January 1965, 23-41.

The Resolution method of Robinson

General idea

Obtaining new instances by deduction from the original set $\mathcal{C}$, such that $\mathcal{C}$ is found to be unsatisfiable whenever both a literal and its negation are deduced

Ground resolution rule

Given two instances $L \vee C_1$ and $\neg L \vee C_2$, where L is a literal, it is possible to deduce a new instance $C_1 \vee C_2$ which is called the *resolvent*

(*Vintage* version of the rule)

- if C and D are two ground clauses, and $L \subseteq C$, $M \subseteq D$ are two singletons (unit sets) whose respective members form a complementary pair, then the ground clause $(C \setminus L) \cup (D \setminus M)$ is called a ground resolvent of C and D [R65]
- if S is any set of ground clauses, then the ground resolution of S, denoted by $\mathcal{R}(S)$, is the set of ground clauses consisting of the members of S together with all ground resolvents of all pairs of members of S [R65]

The Resolution method of Robinson

Unsatisfiability

By applying the rule, it is possible to derive a contradiction when the set is unsatisfiable: such contradiction comes from applying resolution to L and $\neg L$, which generates the *empty clause* $\Box$

Why ground resolution

- as a specific method for testing a finite set of ground clauses for satisfiability, the method of Davis-Putnam would be hard to improve on from the point of view of efficiency [R65]
- now we give another method, far less efficient than theirs, which plays only a theoretical role in our develpment, ... [R65]

☞ on the other hand, the reason for showing ground resolution is its extension to general resolution

Remark: Idempotence

In order to get a contradition *whenever* the set is unsatisfiable, it is necessary to consider *idempotence* $L \vee L \leftrightarrow L$

Extended resolution

Given two instances $L \vee .. \vee L \vee C_1$ and $\neg L \vee .. \vee \neg L \vee C_2$, it is possible to deduce a resolvent $C_1 \vee C_2$

- Applying this extended rule is called a *resolution step over L with resolvent* $C_1 \vee C_2$

The Resolution method of Robinson

Advantages

The deduction system only consists of one rule

- it is interesting that (as far as the author is aware) no other complete system of first-order logic has consisted of just one inference principle [R65]

Method: given a set S of ground instances

$X = S$
repeat
generate by resolution steps all possible resolvents from the elements of X:
let $R(X)$ be the set of resolvents
if $(\square \in R(X))$ **then STOP**: $UNSAT(S)$
if $(R(X) \sqsubseteq X)$ **then STOP**:
all resolvents have already been generated, so that $SAT(S)$
$X = R(X) \cup X$

The Resolution method of Robinson

Lemma (Res-1)

Let m be a node of the semantic tree of a set S, and m' and m'' be its direct successors, both failure nodes. The clauses S' and S'' which become false in m' and m'' have a resolvent which is false in m

Proof.

1. m' and m'' are at a level n in the tree, corresponding to the atom A_n; A_n is taken to be true in m' and false in m''
2. $I(m)$ is the partial interpretation in m: $I(m') = I(m) \cup \{A_n\}$ and $I(m'') = I(m) \cup \{\neg A_n\}$
3. S' and S'' take the form, resp., $\neg A_n \vee S'_n$ and $A_n \vee S''_n$, where neither between $\neg A_n$ and A_n appear in S'_n or S''_n
4. $I(m)$ makes both S'_n and S''_n false, since it is not affected by A_n (by ❸)
5. $S'_n \vee S''_n$, which is a resolvent of S' and S'', is false in m (by ❹)

The Resolution method of Robinson

Lemma (Res-2)

Let A be a closed semantic tree where the level of failure nodes is $\leq n$. If m' is a failure node at level n, then its brother m'' is also a failure node

Proof.

1. since the tree is closed, the path through m'' contains a failure node
2. the failure node cannot be after m'', since the maximum level of failure nodes is n, which is the level of m''
3. since m' is a failure node, its predecessors cannot be failure nodes
4. the predecessors of m'' are the same as those of m', so that, by ❸, they cannot be failure nodes
5. by ❶, ❷ and ❹, m'' must be a failure node

The Resolution method of Robinson

Lemma (Res-3)

Let S be an unsatisfiable set of instances which has a closed semantic tree of level n. Then, there exists a set R of resolvents from S such that the semantic tree of $S \cup R$ is closed and has level $n - 1$

Proof.

1. every failure node at level n has a brother which is also a failure node (Lemma Res-2)
2. every pair of failure nodes has a resolvent r which is false in their predecessor at level $n - 1$ (Lemma Res-1)
3. let $R = \{r \mid r$ is the resolvent of two failure nodes at level $n\}$
4. $S \cup R$ has a closed tree of level $n - 1$ (by ❸)

The Resolution method of Robinson

Theorem (Res)

A set S of ground instances is unsatisfiable iff it is possible to derive $\square$ from it by resolution

Proof ($\rightarrow$).

If S is unsatisfiable, then its semantic tree is closed and finite (if pruned at failure nodes). Let n be the maximum level of failure nodes:

- $n = 1$: there are two failure nodes, corresponding to the atom A_1, where A_1 and $\neg A_1$ become false, respectively. The resolvent is $\square$
- $n > 1$: there exists a set R of resolvents from S such that the semantic tree of $S' = R \cup S$ is closed and has level $n - 1$ (Lemma Res-3)
  - by induction, $\square$ can be derived from S'
  - however, since S' was derived from S by resolution, $\square$ can be derived from S

The Resolution method of Robinson

Theorem (Res)

A set S of ground instances is unsatisfiable iff it is possible to derive $\square$ from it by resolution

Proof ($\longleftarrow$).

1. $S \vdash \square$ by resolution (where $\square$ comes as a resolvent of some L and $\neg L$)
2. $S \models \square$ by ❶ and validity of resolution
3. $\square$ is false in every interpretation
4. S is false in every interpretation (by ❸ and logical consequence)
5. S is unsatisfiable (by ❹)

The Resolution method of Robinson

General method

- generate all possible sets of ground instances
- for every set, apply ground resolution
- the first step is very inefficient
  - the major combinatorial obstacle to efficiency for level-saturation procedures is the enormous rate of growth of the finite sets H_i and HB_i as i increases [R65]

The Resolution method of Robinson

Example from [R63]

arises from seeking to prove the existence of a right identity element in any algebra closed under a binary associative operation having left and right solutions x and y for all equations $x{\cdot}a = b$ and $a{\cdot}y = b$ whose coefficient a and b are in the algebra

$$
\begin{array}{rl}
\mathcal{C} = \{ & \neg p(x,y,u) \vee \neg p(y,z,v) \vee \neg p(x,v,w) \vee p(u,z,w), \\
& \neg p(x,y,u) \vee \neg p(y,z,v) \vee \neg p(u,z,w) \vee p(x,v,w), \\
& p(g(x,y),x,y), \\
& p(x,h(x,y),y), \\
& p(x,y,f(x,y)), \\
& \neg p(k(x),x,k(x)) \quad \}
\end{array}
$$

The Resolution method of Robinson

Example from [R63]

- to prove unsatisfiability, only four ground terms (the *proof set*) are needed:

$$T = \{\ a,\ h(a,a),\ k(h(a,a)),\ g(a,k(h(a,a)))\ \}$$

- however, in order to get T we need to generate a big (19765) number of terms

The Resolution method of Robinson

Example from [R63]

- moreover, only a negligible part of instances of $\mathcal{C}$ over T is needed to get an unsatisfiable S

$$\{ \quad p(a,h(a,a),a),$$
$$\neg p(k(h(a,a)),h(a,a),k(h(a,a))),$$
$$p(g(a,k(h(a,a))),a,k(h(a,a))),$$
$$\neg p(g(a,k(h(a,a))),a,k(h(a,a))) \vee \neg p(a,h(a,a),a) \vee$$
$$\vee \neg p(g(a,k(h(a,a))),a,k(h(a,a))) \vee p(k(h(a,a)),h(a,a),k(h(a,a))) \quad \}$$

The Resolution method of Robinson

Robinson's idea for efficiency

To postpone the substitution of a variable by a term of the Herbrand universe to when it is needed by some resolution step

- work on clauses with variables
- every resolvent (with variables) represents the set of ground instances which would have been obtained by resolution on ground instances