Logic and Theoretical Foundation of Computer Science

LaTFoCS

Pamela Fleischmann

fpa@informatik.uni-kiel.de

Winter Semester 2019

Kiel University

Dependable Systems Group

Resolution

Recap

Definition

- A formula $\varphi \in \Phi$ is in Conjunctive Normal Form (CNF) iff φ is a conjuction of disjunctions of literals.
- A clause is a disjunction of literals.
- A unit clause has exactly one literal.
- The empty clause $\Box$ is the clause without literals.

Recap

Definition

- A formula $\varphi \in \Phi$ is in Conjunctive Normal Form (CNF) iff φ is a conjuction of disjunctions of literals.
- A clause is a disjunction of literals.
- A unit clause has exactly one literal.
- The empty clause $\square$ is the clause without literals.

Every formula in propositional logic can be transformed into an equivalent formula in CNF.

Switching Point of View

- Clauses have only $\vee \rightsquigarrow$ consider it to be a set of literals

Switching Point of View

- Clauses have only $\vee \rightsquigarrow$ consider it to be a set of literals
- Formula in CNF have only $\wedge \rightsquigarrow$ consider it to be a set of clauses

Switching Point of View

- Clauses have only $\vee \rightsquigarrow$ consider it to be a set of literals
- Formula in CNF have only $\wedge \rightsquigarrow$ consider it to be a set of clauses

Definition

The Clausal Form of a formula $\varphi \in \Phi$ is the set $C_\varphi = \{C_1, \ldots, C_m\}$ such that $C_1, \ldots, C_m$ are exactly φ's clauses.

Switching Point of View

- Clauses have only $\vee \rightsquigarrow$ consider it to be a set of literals
- Formula in CNF have only $\wedge \rightsquigarrow$ consider it to be a set of clauses

Definition

The Clausal Form of a formula $\varphi \in \Phi$ is the set $C_\varphi = \{C_1, \ldots, C_m\}$ such that $C_1, \ldots, C_m$ are exactly φ's clauses.

What are the advantages of the the clausal form?

Dealing with Triviality

Definition

A clause C is called trivial iff there exists $p \in A$ with $p, \neg p \in C$.

Dealing with Triviality

Definition

A clause C is called trivial iff there exists $p \in A$ with $p, \neg p \in C$.

Lemma

If S clausal form, $C \in S$ trivial then $S \backslash C \equiv S$.

Dealing with Triviality

Definition

A clause C is called trivial iff there exists $p \in A$ with $p, \neg p \in C$.

Lemma

If S clausal form, $C \in S$ trivial then $S \setminus C \equiv S$.

Proof. Etudes ;-)

Dealing with Triviality

Definition

A clause C is called trivial iff there exists $p \in A$ with $p, \neg p \in C$.

Lemma

If S clausal form, $C \in S$ trivial then $S \backslash C \equiv S$.

Proof. Etudes ;-)

We assume that formula are trivial-clause-free!

Hard to believe but notwithstanding true

Lemma

$\square$ *is unsatisfiable and* $\emptyset$ *is valid.*

Hard to believe but notwithstanding true

Lemma

$\square$ *is unsatisfiable and* $\emptyset$ *is valid.*

Proof.

Hard to believe but notwithstanding true

Lemma

$\square$ *is unsatisfiable and* $\emptyset$ *is valid.*

Proof.

- satisfiable: there exists valuation such that formula true

Hard to believe but notwithstanding true

Lemma

$\square$ *is unsatisfiable and* $\emptyset$ *is valid.*

Proof.

- satisfiable: there exists valuation such that formula true
- valuation: mapping from atoms to truth values

Hard to believe but notwithstanding true

Lemma

$\square$ is unsatisfiable and $\emptyset$ is valid.

Proof.

- satisfiable: there exists valuation such that formula true
- valuation: mapping from atoms to truth values
- $\square$ has no literals $\rightsquigarrow$ no literals that are true

Hard to believe but notwithstanding true

Lemma

$\square$ is unsatisfiable and $\emptyset$ is valid.

Proof.

- satisfiable: there exists valuation such that formula true
- valuation: mapping from atoms to truth values
- $\square$ has no literals $\rightsquigarrow$ no literals that are true
- $\square$ unsatisfiable

Hard to believe but notwithstanding true

Lemma

$\square$ is unsatisfiable and $\emptyset$ is valid.

Proof.

- satisfiable: there exists valuation such that formula true
- valuation: mapping from atoms to truth values
- $\square$ has no literals $\leadsto$ no literals that are true
- $\square$ unsatisfiable
- validity: true under all valuations

Hard to believe but notwithstanding true

Lemma

$\square$ is unsatisfiable and $\emptyset$ is valid.

Proof.

- satisfiable: there exists valuation such that formula true
- valuation: mapping from atoms to truth values
- $\square$ has no literals $\rightsquigarrow$ no literals that are true
- $\square$ unsatisfiable
- validity: true under all valuations
- $\emptyset$ has no clauses which can be false $\rightsquigarrow$ valid $\square$

Resolution

- Refutation procedure for the unsatisfiability problem

Resolution

- Refutation procedure for the unsatisfiability problem

Resolution Rule

C_1, C_2 clauses with $\ell \in C_1$ and $\neg\ell \in C_2$

- C_1, C_2 clashing clauses

Resolution

- Refutation procedure for the unsatisfiability problem

Resolution Rule

C_1, C_2 clauses with $\ell \in C_1$ and $\neg\ell \in C_2$

- C_1, C_2 clashing clauses
- C resolvent of C_1, C_2 w.r.t. ℓ: $C = (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\neg\ell\})$

Resolution

- Refutation procedure for the unsatisfiability problem

Resolution Rule

C_1, C_2 clauses with $\ell \in C_1$ and $\neg\ell \in C_2$

- C_1, C_2 clashing clauses
- C resolvent of C_1, C_2 w.r.t. ℓ: $C = (C_1 \backslash \{\ell\}) \cup (C_2 \backslash \{\neg\ell\})$
- C_1, C_2 parent clauses of C

Resolution

- Refutation procedure for the unsatisfiability problem

Resolution Rule

C_1, C_2 clauses with $\ell \in C_1$ and $\neg\ell \in C_2$

- C_1, C_2 clashing clauses
- C resolvent of C_1, C_2 w.r.t. ℓ: $C = (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\neg\ell\})$
- C_1, C_2 parent clauses of C
- resolution is performed if C_1 and C_2 are substituted by C

Greed implies Triviality

Lemma

The resolvant of two clauses sharing more than one literal is trivial.

Greed implies Triviality

Lemma

The resolvant of two clauses sharing more than one literal is trivial.

Proof.

- C_1, C_2 clauses, ℓ_1, ℓ_2 literals with $\ell_1, \ell_2 \in C_1$, $\neg\ell_1, \neg\ell_2 \in C_2$

Greed implies Triviality

Lemma

The resolvant of two clauses sharing more than one literal is trivial.

Proof.

- C_1, C_2 clauses, ℓ_1, ℓ_2 literals with $\ell_1, \ell_2 \in C_1, \neg\ell_1, \neg\ell_2 \in C_2$
- resolvent w.r.t. ℓ_1:

$$\begin{aligned} C &= (C_1 \backslash \{\ell_1\}) \cup (C_2 \backslash \{\neg\ell_1\}) \\ &= (C_1 \backslash \{\ell_1, \ell_2\} \cup \{\ell_2\}) \cup (C_2 \backslash \{\neg\ell_1, \neg\ell_2\} \cup \{\neg\ell_2\}) \\ &= (C_1 \backslash \{\ell_1, \ell_2\}) \cup (C_2 \backslash \{\neg\ell_1, \neg\ell_2\}) \cup \{\ell_2, \neg\ell_2\} \qquad \square \end{aligned}$$

Greed implies Triviality

Lemma

The resolvant of two clauses sharing more than one literal is trivial.

Proof.

- C_1, C_2 clauses, ℓ_1, ℓ_2 literals with $\ell_1, \ell_2 \in C_1, \neg\ell_1, \neg\ell_2 \in C_2$
- resolvent w.r.t. ℓ_1:

$$\begin{aligned}
C &= (C_1 \backslash \{\ell_1\}) \cup (C_2 \backslash \{\neg\ell_1\}) \\
&= (C_1 \backslash \{\ell_1, \ell_2\} \cup \{\ell_2\}) \cup (C_2 \backslash \{\neg\ell_1, \neg\ell_2\} \cup \{\neg\ell_2\}) \\
&= (C_1 \backslash \{\ell_1, \ell_2\}) \cup (C_2 \backslash \{\neg\ell_1, \neg\ell_2\}) \cup \{\ell_2, \neg\ell_2\} \quad \square
\end{aligned}$$

Delete clauses sharing more than one literal straight away.

We are on the correct path

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

We are on the correct path

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

We are on the correct path

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ

We are on the correct path

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- for "$\longrightarrow$": C satisfiable by valuation β

We are on the correct path

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- for "$\longrightarrow$": C satisfiable by valuation β
- there exists literal $\ell' \in C$ with $\beta(\ell') = \mathtt{true}$

We are on the correct path

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- for "$\longrightarrow$": C satisfiable by valuation β
- there exists literal $\ell' \in C$ with $\beta(\ell') = \mathtt{true}$
- $\ell' \in C \leadsto \ell' \in C_1$ or $\ell' \in C_2 \leadsto$ one clause is satisfied

We are on the correct path

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- for "$\longrightarrow$": C satisfiable by valuation β
- there exists literal $\ell' \in C$ with $\beta(\ell') = \mathtt{true}$
- $\ell' \in C \rightsquigarrow \ell' \in C_1$ or $\ell' \in C_2 \rightsquigarrow$ one clause is satisfied
- β undefined on ℓ (since ℓ not in C)

We are on the correct path

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- for "$\longrightarrow$": C satisfiable by valuation β
- there exists literal $\ell' \in C$ with $\beta(\ell') = \texttt{true}$
- $\ell' \in C \rightsquigarrow \ell' \in C_1$ or $\ell' \in C_2 \rightsquigarrow$ one clause is satisfied
- β undefined on ℓ (since ℓ not in C)
- choose $\beta(\ell)$ such that the other clause is satisfied

Cont.

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

Cont.

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ

Cont.

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ

Cont.

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- "$\rightarrow$": $\surd$

Cont.

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- "$\rightarrow$": $\surd$
- "$\leftarrow$" β valuation satisfying C_1 and C_2

Cont.

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- "$\rightarrow$": $\surd$
- "$\leftarrow$" β valuation satisfying C_1 and C_2
- w.l.o.g. $\beta(\ell) = \texttt{true}$, $\ell \in C_1$

Cont.

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- "$\longrightarrow$": $\surd$
- "$\longleftarrow$" β valuation satisfying C_1 and C_2
- w.l.o.g. $\beta(\ell) = \texttt{true}$, $\ell \in C_1$
- C_1 satisfied and C_2 not satisfied through ℓ

Cont.

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- "$\rightarrow$": $\surd$
- "$\leftarrow$" β valuation satisfying C_1 and C_2
- w.l.o.g. $\beta(\ell) = \texttt{true}$, $\ell \in C_1$
- C_1 satisfied and C_2 not satisfied through ℓ
- exists $\ell' \in C_2$ with $\beta(\ell') = \texttt{true}$ and $\ell' \neq \ell$

Cont.

Theorem

A resolvant is satisfiable iff the parent clauses are both satisfiable.

Proof.

- C_1, C_2 clauses, C resolvent of C_1, C_2 sharing the literal ℓ
- "$\rightarrow$": $\surd$
- "$\leftarrow$" β valuation satisfying C_1 and C_2
- w.l.o.g. $\beta(\ell) = \texttt{true}$, $\ell \in C_1$
- C_1 satisfied and C_2 not satisfied through ℓ
- exists $\ell' \in C_2$ with $\beta(\ell') = \texttt{true}$ and $\ell' \neq \ell$
- $\ell' \in C$ by definition $\square$

Algorithm

Input: clausal form S

$S_0 := S$, flag $=$ `true`
while flag **do**
  Choose clashing $C_1, C_2 \in S$
  Determine resolvent C of C_1, C_2
  $S_{i+1} := S_i \backslash \{C_1, C_2\} \cup \{C\}$
  if $C = \square$ or $S_{i+1} = \emptyset$ **then**
    flag $=$ `false`
  end if
end while
if $C = \square$ **then**
  return UNSAT
end if
if $S_n = \emptyset$ **then**
  return SAT

Resolution as Trees

Definition

Let S be a clausal form with conjuncts $C_1, \ldots, C_m$ for $m \in \mathbb{N}$. Define the resolution tree T of S by

- T is a binary tree
- $C_1, \ldots, C_m$ are T's leaves
- C is the parent node of C_i and C_j iff C is the resolvent of C_i and C_j

Refutation by Resolution

Definition

Derivation of $\square$ from clausal form S is a refutation by resolution of S.

Soundness Resolution

Theorem

If the clauses representing the leaves of the resolution tree are satisfiable then so is the root.

Soundness Resolution

Theorem

If the clauses representing the leaves of the resolution tree are satisfiable then so is the root.

Proof. follows directly with Theorem about satisfiability of resolvent □

Soundness Resolution

Theorem

If the clauses representing the leaves of the resolution tree are satisfiable then so is the root.

Proof. follows directly with Theorem about satisfiability of resolvent □

Corollary (Soundness)

If there is a refutation by resolution for a clausal form S then S is unsatisfiable.

Theorem

If the clauses representing the leaves of the resolution tree are satisfiable then so is the root.

Proof. follows directly with Theorem about satisfiability of resolvent □

Corollary (Soundness)

If there is a refutation by resolution for a clausal form S then S is unsatisfiable.

Notice: the other direction of the Theorem is not true!

Completeness of Resolution

- harder to prove

Completeness of Resolution

- harder to prove
- to prove: if S is unsatisfiably then the procedure halts eventually with $\square$

Completeness of Resolution

- harder to prove
- to prove: if S is unsatisfiably then the procedure halts eventually with $\square$
- termination done: finite set of clauses, no pair of clauses is taken twice

Completeness of Resolution

- harder to prove
- to prove: if S is unsatisfiably then the procedure halts eventually with $\square$
- termination done: finite set of clauses, no pair of clauses is taken twice
- for proving that $\square$ is the output we need semantic trees

Semantic Trees

Definition

S clausal form with atoms $A_S = \{p_1, \ldots, p_n\}$ for $n \in \mathbb{N}$

T_S is semantic tree (tableau) for S iff

- T_S complete binary tree of depth n
- for a node in depth i: left child is p_{i+1}, right child is $\neg p_{i+1}$

Semantic Trees

Definition

S clausal form with atoms $A_S = \{p_1, \ldots, p_n\}$ for $n \in \mathbb{N}$

T_S is semantic tree (tableau) for S iff

- T_S complete binary tree of depth n
- for a node in depth i: left child is p_{i+1}, right child is $\neg p_{i+1}$

natural evalutation w.r.t. branch b: assign `true` if the literal of an edge is positiv and `false` otherwise

Semantic Trees Cont.

Definition

Given a branch b of T and a clausal form S: evaluate S by the evaluation w.r.t b

- b is closed iff S is evaluated to `false`
- b is open iff S is evaluated to `true`
- T is closed (open) iff all branches are closed (open)

Relation: closed and unsatisfiable

Theorem

T_S is closed iff S is unsatisfiable

Relation: closed and unsatisfiable

Theorem

T_S is closed iff S is unsatisfiable

Proof.

- T_S closed, β evaluation of S

Relation: closed and unsatisfiable

Theorem

T_S is closed iff S is unsatisfiable

Proof.

- T_S closed, β evaluation of S
- β equates to some branch b

Relation: closed and unsatisfiable

Theorem

T_S is closed iff S is unsatisfiable

Proof.

- T_S closed, β evaluation of S
- β equates to some branch b
- T_S closed $\rightsquigarrow$ b closed $\rightsquigarrow$ $\beta(S) = \texttt{false}$ $\surd$

Relation: closed and unsatisfiable

Theorem

T_S is closed iff S is unsatisfiable

Proof.

- T_S closed, β evaluation of S
- β equates to some branch b
- T_S closed $\rightsquigarrow$ b closed $\rightsquigarrow$ $\beta(S) = \texttt{false}$ $\surd$
- S unsatisfiable $\rightsquigarrow$ all valuation lead to `false`

Relation: closed and unsatisfiable

Theorem

T_S is closed iff S is unsatisfiable

Proof.

- T_S closed, β evaluation of S
- β equates to some branch b
- T_S closed $\rightsquigarrow$ b closed $\rightsquigarrow$ $\beta(S) = \texttt{false}$ $\surd$
- S unsatisfiable $\rightsquigarrow$ all valuation lead to `false`
- branches equates the valuations $\rightsquigarrow$ all branches closed $\rightsquigarrow$ T closed

$\square$

Failure Nodes

Definition

A node n in the semantic tree T_S of a clausal form S is a failure node if the partial evaluation from the root to n falsifies S and n is closest to the root in this branch.

Failure Nodes

Definition

A node n in the semantic tree T_S of a clausal form S is a failure node if the partial evaluation from the root to n falsifies S and n is closest to the root in this branch.

- In a closed semantic tree each branch has a failure node.

Failure Nodes

Definition

A node n in the semantic tree T_S of a clausal form S is a failure node if the partial evaluation from the root to n falsifies S and n is closest to the root in this branch.

- In a closed semantic tree each branch has a failure node.
- The clauses falsified by a failure node are called associated to this node.

Failure Nodes and the Formula

Lemma

A clause C associated with a failure node n is a subset of the complements of the literals appearing on the branch from the root to n.

Failure Nodes and the Formula

Lemma

A clause C associated with a failure node n is a subset of the complements of the literals appearing on the branch from the root to n.

Proof.

- $\ell_1, \ldots, \ell_k$ literals of C for $k \in \mathbb{N}$

Failure Nodes and the Formula

Lemma

A clause C associated with a failure node n is a subset of the complements of the literals appearing on the branch from the root to n.

Proof.

- $\ell_1, \ldots, \ell_k$ literals of C for $k \in \mathbb{N}$
- $E = \{e_1, \ldots, e_m\}$ literals on the edges from the root to n

Failure Nodes and the Formula

Lemma

A clause C associated with a failure node n is a subset of the complements of the literals appearing on the branch from the root to n.

Proof.

- $\ell_1, \ldots, \ell_k$ literals of C for $k \in \mathbb{N}$
- $E = \{e_1, \ldots, e_m\}$ literals on the edges from the root to n
- n failure node for $C \rightsquigarrow$ all ℓ_i valuates to `false`

Failure Nodes and the Formula

Lemma

A clause C associated with a failure node n is a subset of the complements of the literals appearing on the branch from the root to n.

Proof.

- $\ell_1, \ldots, \ell_k$ literals of C for $k \in \mathbb{N}$
- $E = \{e_1, \ldots, e_m\}$ literals on the edges from the root to n
- n failure node for $C \rightsquigarrow$ all ℓ_i valuates to `false`
- $\rightsquigarrow$ for all ℓ_i exists e_j with $\ell_i = \neg e_j$ $\square$

Inference Nodes

Definition

A node n is an inference node iff its children are failure nodes.

Existence of Inference Nodes

Lemma

T_S clsoed semantic tree for clausal form S. If T_S has at least two failure nodes then T has at least one inference node.

Existence of Inference Nodes

Lemma

T_S clsoed semantic tree for clausal form S. If T_S has at least two failure nodes then T has at least one inference node.

Proof.

- n_1 failure node and sibling n_2 as well $\rightsquigarrow \surd$

Existence of Inference Nodes

Lemma

T_S clsoed semantic tree for clausal form S. If T_S has at least two failure nodes then T has at least one inference node.

Proof.

- n_1 failure node and sibling n_2 as well $\rightsquigarrow \surd$
- n_1 failure node and sibling n_2 not $\rightsquigarrow$ no ancestor can be a failure node (minimality of n_1)

Existence of Inference Nodes

Lemma

T_S clsoed semantic tree for clausal form S. If T_S has at least two failure nodes then T has at least one inference node.

Proof.

- n_1 failure node and sibling n_2 as well $\rightsquigarrow \surd$
- n_1 failure node and sibling n_2 not $\rightsquigarrow$ no ancestor can be a failure node (minimality of n_1)
- T_S closed $\rightsquigarrow$ all branches containing n_2 have a failure node

Existence of Inference Nodes

Lemma

T_S clsoed semantic tree for clausal form S. If T_S has at least two failure nodes then T has at least one inference node.

Proof.

- n_1 failure node and sibling n_2 as well $\rightsquigarrow$ $\surd$
- n_1 failure node and sibling n_2 not $\rightsquigarrow$ no ancestor can be a failure node (minimality of n_1)
- T_S closed $\rightsquigarrow$ all branches containing n_2 have a failure node
- this failure node needs to be below n_2

Existence of Inference Nodes

Lemma

T_S clsoed semantic tree for clausal form S. If T_S has at least two failure nodes then T has at least one inference node.

Proof.

- n_1 failure node and sibling n_2 as well $\rightsquigarrow \surd$
- n_1 failure node and sibling n_2 not $\rightsquigarrow$ no ancestor can be a failure node (minimality of n_1)
- T_S closed $\rightsquigarrow$ all branches containing n_2 have a failure node
- this failure node needs to be below n_2
- no inference node there $\rightsquigarrow$ T_S infinite. □

Inference Nodes and Resolvent

Lemma

T_S closed semantic tree for clausal form S; n inference node with failure nodes n_1, n_2; C_1, C_2 associated with n_1, n_2:
C_1, C_2 clash and partial valuation to n falsifies the resolvant

Inference Nodes and Resolvent

Lemma

T_S closed semantic tree for clausal form S; n inference node with failure nodes n_1, n_2; C_1, C_2 associated with n_1, n_2: C_1, C_2 clash and partial valuation to n falsifies the resolvant

Proof.

- C_1, C_2 are not falisfied by any ancestor of n

Inference Nodes and Resolvent

Lemma

T_S closed semantic tree for clausal form S; n inference node with failure nodes n_1, n_2; C_1, C_2 associated with n_1, n_2:
C_1, C_2 clash and partial valuation to n falsifies the resolvant

Proof.

- C_1, C_2 are not falisified by any ancestor of n
- C_1, C_2 are subsets of the complements of the branches to n_1, n_2 resp.

Inference Nodes and Resolvent

Lemma

T_S closed semantic tree for clausal form S; n inference node with failure nodes n_1, n_2; C_1, C_2 associated with n_1, n_2: C_1, C_2 clash and partial valuation to n falsifies the resolvant

Proof.

- C_1, C_2 are not falisfied by any ancestor of n
- C_1, C_2 are subsets of the complements of the branches to n_1, n_2 resp.
- branches to n_1, n_2 identical except of literal $\leadsto \ell \in C_1$, $\neg\ell \in C_2$

Inference Nodes and Resolvent

Lemma

T_S closed semantic tree for clausal form S; n inference node with failure nodes n_1, n_2; C_1, C_2 associated with n_1, n_2: C_1, C_2 clash and partial valuation to n falsifies the resolvant

Proof.

- C_1, C_2 are not falisfied by any ancestor of n
- C_1, C_2 are subsets of the complements of the branches to n_1, n_2 resp.
- branches to n_1, n_2 identical except of literal $\leadsto \ell \in C_1$, $\neg\ell \in C_2$
- resolvent of C_1, C_2 does not have $\ell, \neg\ell$

Inference Nodes and Resolvent

Lemma

T_S closed semantic tree for clausal form S; n inference node with failure nodes n_1, n_2; C_1, C_2 associated with n_1, n_2:
C_1, C_2 clash and partial valuation to n falsifies the resolvant

Proof.

- C_1, C_2 are not falisfied by any ancestor of n
- C_1, C_2 are subsets of the complements of the branches to n_1, n_2 resp.
- branches to n_1, n_2 identical except of literal $\leadsto \ell \in C_1$, $\neg\ell \in C_2$
- resolvent of C_1, C_2 does not have $\ell, \neg\ell$
- branch to resolvent falsifies it $\square$

Inference Nodes and Resolvent

Lemma

n inference node of n_1, n_2, C_1, C_2 associated clauses, C resolvent of C_1, C_2:

$S \cup \{C\}$ has failure node which is either n or ancestor of n

Inference Nodes and Resolvent

Lemma

n inference node of n_1, n_2, C_1, C_2 associated clauses, C resolvent of C_1, C_2:
$S \cup \{C\}$ has failure node which is either n or ancestor of n

Proof. Etudes.

Completeness of Resolution

Theorem

If the clausal form S is unsatisfiable then the procedure halts with $\square$.

Completeness of Resolution

Theorem

If the clausal form S is unsatisfiable then the procedure halts with $\Box$.

Proof.

- S unsatisfiable $\rightsquigarrow$ T_S closed semantic tree

Completeness of Resolution

Theorem

If the clausal form S is unsatisfiable then the procedure halts with $\Box$.

Proof.

- S unsatisfiable $\rightsquigarrow$ T_S closed semantic tree
- if S contains $\Box$ $\surd$

Completeness of Resolution

Theorem

If the clausal form S is unsatisfiable then the procedure halts with $\square$.

Proof.

- S unsatisfiable $\rightsquigarrow$ T_S closed semantic tree
- if S contains $\square$ $\surd$
- if S does not contain $\square$ $\rightsquigarrow$ there exist two failure nodes (not proven here)

Completeness of Resolution

Theorem

If the clausal form S is unsatisfiable then the procedure halts with $\square$.

Proof.

- S unsatisfiable $\rightsquigarrow$ T_S closed semantic tree
- if S contains $\square$ $\surd$
- if S does not contain $\square$ $\rightsquigarrow$ there exist two failure nodes (not proven here)
- there exists inference node

Theorem

If the clausal form S is unsatisfiable then the procedure halts with $\Box$.

Proof.

- S unsatisfiable $\rightsquigarrow$ T_S closed semantic tree
- if S contains $\Box$ $\surd$
- if S does not contain $\Box$ $\rightsquigarrow$ there exist two failure nodes (not proven here)
- there exists inference node
- applying resolution: inference node is failure node, two failure nodes deleted

Theorem

If the clausal form S is unsatisfiable then the procedure halts with $\square$.

Proof.

- S unsatisfiable $\rightsquigarrow$ T_S closed semantic tree
- if S contains $\square$ $\surd$
- if S does not contain $\square$ $\rightsquigarrow$ there exist two failure nodes (not proven here)
- there exists inference node
- applying resolution: inference node is failure node, two failure nodes deleted
- successively decereasing failure nodes $\rightsquigarrow$ reaching root

Completeness of Resolution

Theorem

If the clausal form S is unsatisfiable then the procedure halts with $\square$.

Proof.

- S unsatisfiable $\rightsquigarrow$ T_S closed semantic tree
- if S contains $\square$ $\surd$
- if S does not contain $\square$ $\rightsquigarrow$ there exist two failure nodes (not proven here)
- there exists inference node
- applying resolution: inference node is failure node, two failure nodes deleted
- successively decereasing failure nodes $\rightsquigarrow$ reaching root
- empty clause needs to be associated with root $\square$