Logical and Theoretical Foundations of Computer Science

LaTFoCS

Pamela Fleischmann

fpa@informatik.uni-kiel.de

Winter Semester 2019

Kiel University
Dependable Systems Group

SEIT 350 JAHREN GANZ WEIT OBEN
CAU

First-Order Logic: Resolution

Expectations in Predicate Logic

Propositional Logic:

- resolution is sound and complete (formula unsatisfiable iff algo outputs UNSAT)
- decision procedure (terminates always)

Expectations in Predicate Logic

Propositional Logic:

- resolution is sound and complete (formula unsatisfiable iff algo outputs UNSAT)
- decision procedure (terminates always)

Predicate Logic:

- resolution will be shown to be sound and complete

Expectations in Predicate Logic

Propositional Logic:

- resolution is sound and complete (formula unsatisfiable iff algo outputs UNSAT)
- decision procedure (terminates always)

Predicate Logic:

- resolution will be shown to be sound and complete
- but it is not a decision prodecure

2-Stage Resolution

Generalisation of resolution has two stages

- ground resolution: works on ground literals (generalisation of literals)
- unification: works on non-ground literals

Ground . . .

Definition

- ground term: term without variables
- ground atomic (FO-)formula: atomic formula (only 1. from the formula definition) with only ground terms
- ground (FO-)literal: ground atomic formula or its negation
- ground (FO)-formula: quantifier-free formula with only grond atomic formulae

Ground Resolution Rule

Definition

C_1, C_2 ground clauses, ℓ ground literal, $\ell \in C_1$, $\neg\ell \in C_1$

Ground Resolution Rule

Definition

C_1, C_2 ground clauses, ℓ ground literal, $\ell \in C_1$, $\neg\ell \in C_1$

- C_1, C_2 clash on ℓ

Ground Resolution Rule

Definition

C_1, C_2 ground clauses, ℓ ground literal, $\ell \in C_1$, $\neg\ell \in C_1$

- C_1, C_2 clash on ℓ
- C resolvent of C_1, C_2: $C = (C_1 \backslash \{\ell\}) \cup (C_2 \backslash \{\neg\ell\})$

Ground Resolution Rule

Definition

C_1, C_2 ground clauses, ℓ ground literal, $\ell \in C_1$, $\neg\ell \in C_1$

- C_1, C_2 clash on ℓ
- C resolvent of C_1, C_2: $C = (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\neg\ell\})$
- C_1, C_2 parent clause of C

Satisfiability of parents and resolvent

Theorem

A resolvent is satisfiable iff the parents are both satisfiable.

Satisfiability of parents and resolvent

Theorem

A resolvent is satisfiable iff the parents are both satisfiable.

Proof. Not done here (based on Herbrand interpretations)

Satisfiability of parents and resolvent

Theorem

A resolvent is satisfiable iff the parents are both satisfiable.

Proof. Not done here (based on Herbrand interpretations)

Theorem

The ground resolution is sound and complete.

Satisfiability of parents and resolvent

Theorem

A resolvent is satisfiable iff the parents are both satisfiable.

Proof. Not done here (based on Herbrand interpretations)

Theorem

The ground resolution is sound and complete.

Proof. Not done here.

Limits of Ground Resolution

- $|\mathscr{F}| \geq 1$

Limits of Ground Resolution

- $|\mathscr{F}| \geq 1$
- $\rightsquigarrow$ set of ground terms infinite

Limits of Ground Resolution

- $|\mathscr{F}| \geq 1$
- $\rightsquigarrow$ set of ground terms infinite
- $\rightsquigarrow$ ground resolution not a useful refutation procedure

Limits of Ground Resolution

- $|\mathscr{F}| \geq 1$
- $\rightsquigarrow$ set of ground terms infinite
- $\rightsquigarrow$ ground resolution not a useful refutation procedure
- Robinson (1965): substitutions causing clashes
  - substitution maps variables to terms
  - empty substitution does not map anything (sets as point of view)

Expressions and Instances

Definition

- E expression: term, literal, clause, or set of clauses
- $\beta(E)$ instance of E: replace simultaneously all occurrences of x_i by $\beta(x_i)$ for substitution β and variables x_i
- composition of substitutions $\beta_1 \circ \beta_2$ for all $x \in \mathrm{dom}(\beta_1) \cup \mathrm{dom}(\beta_2)$

$$(\beta_1 \circ \beta_2)(x) = \begin{cases} \beta_2(\beta_1(x)) & \text{if } x \in \mathrm{dom}(\beta_1) \wedge x \neq \beta_2(\beta_1(x)) \\ \beta_2(x) & \text{if } x \notin \mathrm{dom}(\beta_1) \wedge x \in \mathrm{dom}(\beta_2) \\ \text{undef} & \text{otherwise.} \end{cases}$$

Properties of Instances

Lemma

E substitution, $\beta_1, \beta_2, \beta_3$ *substitutions*

- $(\beta_1 \circ \beta_2)(E) = \beta_2(\beta_1(E))$
- $\beta_1 \circ (\beta_2 \circ \beta_3) = (\beta_1 \circ \beta_2) \circ \beta_3$

Properties of Instances

Lemma

E substitution, $\beta_1, \beta_2, \beta_3$ *substitutions*

- $(\beta_1 \circ \beta_2)(E) = \beta_2(\beta_1(E))$
- $\beta_1 \circ (\beta_2 \circ \beta_3) = (\beta_1 \circ \beta_2) \circ \beta_3$

Proof. Etudes.

Making things clashable

$$P(f(x), g(y)) \quad \text{not clashing with} \quad \neg P(f(f(a)), g(z))$$

Making things clashable

$$P(f(x), g(y)) \quad \text{not clashing with} \quad \neg P(f(f(a)), g(z))$$

Can we get it clashing?

Making things clashable

$$P(f(x), g(y)) \quad \text{not clashing with} \quad \neg P(f(f(a)), g(z))$$

Can we get it clashing? Consider the substitution $\sigma(x) = f(a)$, $\sigma(y) = g(b)$, and $\sigma(z) = g(b)$.

Making things clashable

$$P(f(x), g(y)) \quad \text{not clashing with} \quad \neg P(f(f(a)), g(z))$$

Can we get it clashing? Consider the substitution $\sigma(x) = f(a)$, $\sigma(y) = g(b)$, and $\sigma(z) = g(b)$.

- $\sigma(P(f(x), g(y))) = P(f(f(a)), g(g(b)))$

Making things clashable

$$P(f(x), g(y)) \quad \text{not clashing with} \quad \neg P(f(f(a)), g(z))$$

Can we get it clashing? Consider the substitution $\sigma(x) = f(a)$, $\sigma(y) = g(b)$, and $\sigma(z) = g(b)$.

- $\sigma(P(f(x), g(y))) = P(f(f(a)), g(g(b)))$
- $\sigma(P(f(f(a)), g(z))) = P(f(f(a)), g(g(b)))$

Making things clashable

$$P(f(x), g(y)) \quad \text{not clashing with} \quad \neg P(f(f(a)), g(z))$$

Can we get it clashing? Consider the substitution $\sigma(x) = f(a)$, $\sigma(y) = g(b)$, and $\sigma(z) = g(b)$.

- $\sigma(P(f(x), g(y))) = P(f(f(a)), g(g(b)))$
- $\sigma(P(f(f(a)), g(z))) = P(f(f(a)), g(g(b)))$
- $\rightsquigarrow$ they clash

Unification

Definition

Let $\varphi_1, \ldots, \varphi_n$ be predicate logic formulae.

- A substitution σ is a unifier for $\varphi_1, \ldots, \varphi_n$ if $\sigma(\varphi_i) = \sigma(\varphi_j)$ for all $i, j \in [n]$.

Unification

Definition

Let $\varphi_1, \ldots, \varphi_n$ be predicate logic formulae.

- A substitution σ is a unifier for $\varphi_1, \ldots, \varphi_n$ if $\sigma(\varphi_i) = \sigma(\varphi_j)$ for all $i, j \in [n]$.
- A unifier σ is called most general unifier (mgu) such that for every unifier σ' there exists a substitution σ'' with $\sigma' = \sigma \circ \sigma''$.

Example

$\sigma(x) = f(a)$, $\sigma(y) = g(b)$, and $\sigma(z) = g(b)$ is a unifier for $P(f(x), g(y))$ and $\neg P(f(f(a)), g(z))$ but it is not mgu:

- $\mu(x) = f(a), \mu(z) = \mu(y) = b$

Example

$\sigma(x) = f(a)$, $\sigma(y) = g(b)$, and $\sigma(z) = g(b)$ is a unifier for $P(f(x), g(y))$ and $\neg P(f(f(a)), g(z))$ but it is not mgu:

- $\mu(x) = f(a), \mu(z) = \mu(y) = b$
- $\nu(y) = g(g(b)) = \nu(z)$

Example

$\sigma(x) = f(a)$, $\sigma(y) = g(b)$, and $\sigma(z) = g(b)$ is a unifier for $P(f(x), g(y))$ and $\neg P(f(f(a)), g(z))$ but it is not mgu:

- $\mu(x) = f(a), \mu(z) = \mu(y) = b$
- $\nu(y) = g(g(b)) = \nu(z)$
- $\sigma = \mu \circ \nu$

Preparations for a Unification Algorithm

Definition

A set of term equations is in solved form iff

- all equations are of the form $x_i = t_i$ for variables x_i and terms t_i
- each variable x_i that appears on the left-hand side of an equation does not appear elsewhere in the set

Preparations for a Unification Algorithm

Definition

A set of term equations is in solved form iff

- all equations are of the form $x_i = t_i$ for variables x_i and terms t_i
- each variable x_i that appears on the left-hand side of an equation does not appear elsewhere in the set

A set of equation in solved form defines a substitution.

Unification Algorithm

Input: Set of term equations

While a rule is applicable

1. Transform $t = x$ into $x = t$.
2. Delete $x = x$.
3. For each $t = t'$: if the left-most symbols are not identical, stop with *not unifiable*

   For each $t = t'$ with $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$
   - delete $t = t'$
   - insert $t_i = t'_i$ for all $i \in [k]$
4. For $x = t$ with other occurrences of x in the set:
   - If x occurs in t and differs from t, stop with *not unifiable*
   - Otherwise replace each x by t.

Remarks on the Unification Algorithm

- What is the complexity?

Remarks on the Unification Algorithm

- What is the complexity?
- In the worst case exponential!

Remarks on the Unification Algorithm

- What is the complexity?
- In the worst case exponential!
- Consider the set of equations $\{x_i = f(x_{i-1}, x_{i-1})\}$ ...

Remarks on the Unification Algorithm

- What is the complexity?
- In the worst case exponential!
- Consider the set of equations $\{x_i = f(x_{i-1}, x_{i-1})\} \ldots$
- Notice that the algorithm is non-deterministic!

Remarks on the Unification Algorithm

- What is the complexity?
- In the worst case exponential!
- Consider the set of equations $\{x_i = f(x_{i-1}, x_{i-1})\} \ldots$
- Notice that the algorithm is non-deterministic!
- In practice: omit occurs-check and choose heuristic

Correctness and Soundness of the Unification Algorithm

Theorem

1. *The algorithm terminates with the set of equations in solved form or it reports* not unifiable.

Correctness and Soundness of the Unification Algorithm

Theorem

1. *The algorithm terminates with the set of equations in solved form or it reports* not unifiable.
2. *If the algorithm reports* not unifiable, *there is no unifier for the set of equations.*

Correctness and Soundness of the Unification Algorithm

Theorem

1. *The algorithm terminates with the set of equations in solved form or it reports* not unifiable.
2. *If the algorithm reports* not unifiable, *there is no unifier for the set of equations.*
3. *If the algorithm terminates successfully, the mgu is given by the equations* $x_i = t_i$.

Proof

- Rule 1. to 3. can only be used finitely many times without using 4.

Proof

- Rule 1. to 3. can only be used finitely many times without using 4.
- m distinct variables in the set

Proof

- Rule 1. to 3. can only be used finitely many times without using 4.
- m distinct variables in the set
- Rule 4. is at most m times aplicable

Proof

- Rule 1. to 3. can only be used finitely many times without using 4.
- m distinct variables in the set
- Rule 4. is at most m times aplicable
- $\rightsquigarrow$ termination

Proof

- Rule 1. to 3. can only be used finitely many times without using 4.
- m distinct variables in the set
- Rule 4. is at most m times aplicable
- $\rightsquigarrow$ termination
- if the algorithm terminates with *not unifiable*, either the left-most symbols are not identical or x has to be t and something different

Proof

- Rule 1. to 3. can only be used finitely many times without using 4.
- m distinct variables in the set
- Rule 4. is at most m times aplicable
- $\leadsto$ termination
- if the algorithm terminates with *not unifiable*, either the left-most symbols are not identical or x has to be t and something different
- $\leadsto$ there cannot be a unifier

Proof

- Rule 1. to 3. can only be used finitely many times without using 4.
- m distinct variables in the set
- Rule 4. is at most m times aplicable
- $\leadsto$ termination
- if the algorithm terminates with *not unifiable*, either the left-most symbols are not identical or x has to be t and something different
- $\leadsto$ there cannot be a unifier
- otherwise the substitution is given by the equations $x_i = t_i$

Proof that the unifier is most general

- a transformation is called *equivalence transformation* if it preserves the set of unifiers

Proof that the unifier is most general

- a transformation is called *equivalence transformation* if it preserves the set of unifiers
- Rule 1. and 2. are equivalence transformations

Proof that the unifier is most general

- a transformation is called *equivalence transformation* if it preserves the set of unifiers
- Rule 1. and 2. are equivalence transformations
- for proving that Rule 3. is an equivalence transformation consider $t = f(t_1, \dots, t_k)$ and $t' = f(t'_1, \dots, t'_k)$

Proof that the unifier is most general

- a transformation is called *equivalence transformation* if it preserves the set of unifiers
- Rule 1. and 2. are equivalence transformations
- for proving that Rule 3. is an equivalence transformation consider $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$
- If $\sigma(t) = \sigma(t')$ we have $\sigma(t_i) = \sigma(t'_i)$ - inductive argument

Proof that the unifier is most general

- a transformation is called *equivalence transformation* if it preserves the set of unifiers
- Rule 1. and 2. are equivalence transformations
- for proving that Rule 3. is an equivalence transformation consider $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$
- If $\sigma(t) = \sigma(t')$ we have $\sigma(t_i) = \sigma(t'_i)$ - inductive argument
- If $t_i = t'_i$ then $\sigma(t_i) = \sigma(t'_i)$ and thus $\sigma(t) = \sigma(t')$

Proof that the unifier is most general

- a transformation is called *equivalence transformation* if it preserves the set of unifiers
- Rule 1. and 2. are equivalence transformations
- for proving that Rule 3. is an equivalence transformation consider $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$
- If $\sigma(t) = \sigma(t')$ we have $\sigma(t_i) = \sigma(t'_i)$ - inductive argument
- If $t_i = t'_i$ then $\sigma(t_i) = \sigma(t'_i)$ and thus $\sigma(t) = \sigma(t')$
- $\rightsquigarrow$ Rule 3. is equivalence transformation

Proof Cont.

- for proving that Rule 4. is an equivalence transformation consider $t_1 = t_2$ is tranformed by 4. into $u_1 = u_2$ on $x = t$

Proof Cont.

- for proving that Rule 4. is an equivalence transformation consider $t_1 = t_2$ is tranformed by 4. into $u_1 = u_2$ on $x = t$
- after the application $x = t$ is still in the set

Proof Cont.

- for proving that Rule 4. is an equivalence transformation consider $t_1 = t_2$ is tranformed by 4. into $u_1 = u_2$ on $x = t$
- after the application $x = t$ is still in the set
- $\rightsquigarrow$ any unifier σ fulfils $\sigma(x) = \sigma(t)$

Proof Cont.

- for proving that Rule 4. is an equivalence transformation consider $t_1 = t_2$ is tranformed by 4. into $u_1 = u_2$ on $x = t$
- after the application $x = t$ is still in the set
- $\rightsquigarrow$ any unifier σ fulfils $\sigma(x) = \sigma(t)$
- $\rightsquigarrow$ $\sigma(u_i) = \sigma(t_i[t/x]) = t_i[t/\sigma(x)] = \sigma(t_i)$

Proof Cont.

- for proving that Rule 4. is an equivalence transformation consider $t_1 = t_2$ is tranformed by 4. into $u_1 = u_2$ on $x = t$
- after the application $x = t$ is still in the set
- $\rightsquigarrow$ any unifier σ fulfils $\sigma(x) = \sigma(t)$
- $\rightsquigarrow \sigma(u_i) = \sigma(t_i[t/x]) = t_i[t/\sigma(x)] = \sigma(t_i)$
- σ unifier for $t_1 = t_2 \rightsquigarrow \sigma(u_1) = \sigma(t_1) = \sigma(t_2) = \sigma(u_2)$

Proof Cont.

- for proving that Rule 4. is an equivalence transformation consider $t_1 = t_2$ is tranformed by 4. into $u_1 = u_2$ on $x = t$
- after the application $x = t$ is still in the set
- $\leadsto$ any unifier σ fulfils $\sigma(x) = \sigma(t)$
- $\leadsto$ $\sigma(u_i) = \sigma(t_i[t/x]) = t_i[t/\sigma(x)] = \sigma(t_i)$
- σ unifier for $t_1 = t_2 \leadsto \sigma(u_1) = \sigma(t_1) = \sigma(t_2) = \sigma(u_2)$
- $\leadsto$ σ unifier for $u_1 = u_2$

Proof Cont.

- for proving that Rule 4. is an equivalence transformation consider $t_1 = t_2$ is tranformed by 4. into $u_1 = u_2$ on $x = t$
- after the application $x = t$ is still in the set
- $\rightsquigarrow$ any unifier σ fulfils $\sigma(x) = \sigma(t)$
- $\rightsquigarrow$ $\sigma(u_i) = \sigma(t_i[t/x]) = t_i[t/\sigma(x)] = \sigma(t_i)$
- σ unifier for $t_1 = t_2 \rightsquigarrow \sigma(u_1) = \sigma(t_1) = \sigma(t_2) = \sigma(u_2)$
- $\rightsquigarrow$ σ unifier for $u_1 = u_2$
- $\rightsquigarrow$ Rule 4. is equivalence transformation

Proof Cont.

- for proving that Rule 4. is an equivalence transformation consider $t_1 = t_2$ is tranformed by 4. into $u_1 = u_2$ on $x = t$
- after the application $x = t$ is still in the set
- $\rightsquigarrow$ any unifier σ fulfils $\sigma(x) = \sigma(t)$
- $\rightsquigarrow$ $\sigma(u_i) = \sigma(t_i[t/x]) = t_i[t/\sigma(x)] = \sigma(t_i)$
- σ unifier for $t_1 = t_2 \rightsquigarrow \sigma(u_1) = \sigma(t_1) = \sigma(t_2) = \sigma(u_2)$
- $\rightsquigarrow$ σ unifier for $u_1 = u_2$
- $\rightsquigarrow$ Rule 4. is equivalence transformation
- all unifiers are preserved and the output is the most general one $\square$