Logics and Statistics for Language Modeling

Carlos Areces

areces@loria.fr
http://www.loria.fr/~areces/ls
INRIA Nancy Grand Est
Nancy, France

2009/2010

Today's Program

Today's Program

- First Order Logics.
  - Models
  - Quantification
  - Infinite Models
  - Undecidability
- Unification
  - Motivation
  - Brief History
  - Preliminaries
  - Unification Algorithm

Some Examples of Models

Some Examples of Models

- Let's consider the first two sentences of last class:
  - There is one triangle and two circles.
  - Each object has a color: either red, blue or green.

Quantification

Quantification

- $M, g \models \exists x.(\varphi)$ if and only if $M, g' \models \varphi$ for some assignment g' identical to g excepts perhaps in $g(x)$.

Quantification

- $M, g \models \exists x.(\varphi)$ if and only if $M, g' \models \varphi$ for some assignment g' identical to g excepts perhaps in $g(x)$.
- $M, g \models \neg\exists x.(\neg\varphi)$ iff

Quantification

- $M, g \models \exists x.(\varphi)$ if and only if $M, g' \models \varphi$ for some assignment g' identical to g excepts perhaps in $g(x)$.
- $M, g \models \neg\exists x.(\neg\varphi)$ iff
  $M, g \not\models \exists x.(\neg\varphi)$ iff

Quantification

- $M, g \models \exists x.(\varphi)$ if and only if $M, g' \models \varphi$ for some assignment g' identical to g excepts perhaps in $g(x)$.
- $M, g \models \neg\exists x.(\neg\varphi)$ iff
  $M, g \not\models \exists x.(\neg\varphi)$ iff
  $M, g' \not\models (\neg\varphi)$ for some assignment g' identical to g except perhaps in $g(x)$ iff

Quantification

- $M, g \models \exists x.(\varphi)$ if and only if $M, g' \models \varphi$ for some assignment g' identical to g excepts perhaps in $g(x)$.
- $M, g \models \neg\exists x.(\neg\varphi)$ iff
  $M, g \not\models \exists x.(\neg\varphi)$ iff
  $M, g' \not\models (\neg\varphi)$ for some assignment g' identical to g except perhaps in $g(x)$ iff
  $M, g' \models \varphi$ for all assignments g' identical to g except perhaps in $g(x)$.

Quantification

- $M, g \models \exists x.(\varphi)$ if and only if $M, g' \models \varphi$ for some assignment g' identical to g excepts perhaps in $g(x)$.
- $M, g \models \neg\exists x.(\neg\varphi)$ iff
  $M, g \not\models \exists x.(\neg\varphi)$ iff
  $M, g' \not\models (\neg\varphi)$ for some assignment g' identical to g except perhaps in $g(x)$ iff
  $M, g' \models \varphi$ for all assignments g' identical to g except perhaps in $g(x)$.
- $M, g \models \forall x.(\varphi)$ if and only if $M, g' \models \varphi$ for all assignment g' identical to g excepts perhaps in $g(x)$.

Some Properties of Quantifiers

Some Properties of Quantifiers

- $\forall x.\forall y.\varphi$ is the same as $\forall y.\forall x.\varphi$

Some Properties of Quantifiers

- $\forall x.\forall y.\varphi$ is the same as $\forall y.\forall x.\varphi$
- $\exists x.\exists y.\varphi$ is the same as $\exists y.\exists x.\varphi$

Some Properties of Quantifiers

- $\forall x.\forall y.\varphi$ is the same as $\forall y.\forall x.\varphi$
- $\exists x.\exists y.\varphi$ is the same as $\exists y.\exists x.\varphi$
- $\exists x.\forall y.\varphi$ is not the same as $\forall y.\exists x.\varphi$

Some Properties of Quantifiers

- $\forall x.\forall y.\varphi$ is the same as $\forall y.\forall x.\varphi$
- $\exists x.\exists y.\varphi$ is the same as $\exists y.\exists x.\varphi$
- $\exists x.\forall y.\varphi$ is not the same as $\forall y.\exists x.\varphi$
- $\forall x.\varphi$ is the same as $\forall y.\varphi[x/y]$ if y does not appear in φ, and similarly for $\exists x.\varphi$ and $\exists y.\varphi[x/y]$.

Some Properties of Quantifiers

- $\forall x.\forall y.\varphi$ is the same as $\forall y.\forall x.\varphi$
- $\exists x.\exists y.\varphi$ is the same as $\exists y.\exists x.\varphi$
- $\exists x.\forall y.\varphi$ is not the same as $\forall y.\exists x.\varphi$
- $\forall x.\varphi$ is the same as $\forall y.\varphi[x/y]$ if y does not appear in φ, and similarly for $\exists x.\varphi$ and $\exists y.\varphi[x/y]$.
- $\varphi \wedge Qx.\psi$ is the same as $Qx.(\varphi \wedge \psi)$ if x does not appear in φ ($Q \in \{\forall, \exists\}$).

Some Properties of Quantifiers

- $\forall x.\forall y.\varphi$ is the same as $\forall y.\forall x.\varphi$
- $\exists x.\exists y.\varphi$ is the same as $\exists y.\exists x.\varphi$
- $\exists x.\forall y.\varphi$ is not the same as $\forall y.\exists x.\varphi$
- $\forall x.\varphi$ is the same as $\forall y.\varphi[x/y]$ if y does not appear in φ, and similarly for $\exists x.\varphi$ and $\exists y.\varphi[x/y]$.
- $\varphi \wedge Qx.\psi$ is the same as $Qx.(\varphi \wedge \psi)$ if x does not appear in φ ($Q \in \{\forall, \exists\}$).
- $\neg\exists x.\varphi$ is equivalent to $\forall x.\neg\varphi$ and $\neg\forall y.\varphi$ is equivalent to $\exists x.\neg\varphi$.

What can we express?

What can we express?

- Properties on the Natural Numbers

$$\forall x.(nat(x) \rightarrow (x + 0 = x)).$$

$$\forall x.(nat(x) \rightarrow 0 * x = 0).$$

$$\forall x.\forall y.(nat(x) \wedge nat(y) \rightarrow x + succ(y) = succ(x + y)).$$

$$\forall x.\forall y.(nat(x) \wedge nat(y) \rightarrow x * y = y * x).$$

What can we express?

- Properties on the Natural Numbers

$$\forall x.(nat(x) \rightarrow (x + 0 = x)).$$

$$\forall x.(nat(x) \rightarrow 0 * x = 0).$$

$$\forall x.\forall y.(nat(x) \wedge nat(y) \rightarrow x + succ(y) = succ(x + y)).$$

$$\forall x.\forall y.(nat(x) \wedge nat(y) \rightarrow x * y = y * x).$$

- The Zermelo-Fraenkel axioms for Set Theory can be stated in FO.

$$\forall x.\forall y.((x = y) \leftrightarrow (x \subseteq y \wedge y \subseteq x)$$

What can we express?

- Properties on the Natural Numbers

$$\forall x.(nat(x) \rightarrow (x + 0 = x)).$$

$$\forall x.(nat(x) \rightarrow 0 * x = 0).$$

$$\forall x.\forall y.(nat(x) \land nat(y) \rightarrow x + succ(y) = succ(x + y)).$$

$$\forall x.\forall y.(nat(x) \land nat(y) \rightarrow x * y = y * x).$$

- The Zermelo-Fraenkel axioms for Set Theory can be stated in FO.

$$\forall x.\forall y.((x = y) \leftrightarrow (x \subseteq y \land y \subseteq x)$$

- Infinite models.

What can we express?

- Properties on the Natural Numbers

$$\forall x.(nat(x) \rightarrow (x + 0 = x)).$$

$$\forall x.(nat(x) \rightarrow 0 * x = 0).$$

$$\forall x.\forall y.(nat(x) \wedge nat(y) \rightarrow x + succ(y) = succ(x + y)).$$

$$\forall x.\forall y.(nat(x) \wedge nat(y) \rightarrow x * y = y * x).$$

- The Zermelo-Fraenkel axioms for Set Theory can be stated in FO.

$$\forall x.\forall y.((x = y) \leftrightarrow (x \subseteq y \wedge y \subseteq x)$$

- Infinite models.
- An important part of natural language can be formalized in FO.

Undecidability of FOL

Undecidability of FOL

How can we prove that problem X is undecidable?

Undecidability of FOL

How can we prove that problem X is undecidable?
One way is

- Ask somebody, more intelligent than us, to prove that some problem Y is undecidable

Undecidability of FOL

How can we prove that problem X is undecidable?
One way is

- Ask somebody, more intelligent than us, to prove that some problem Y is undecidable
- Prove that if X would be decidable then Y would be decidable, giving a codification of Y into X.

Undecidability of FOL

How can we prove that problem X is undecidable?
One way is

- Ask somebody, more intelligent than us, to prove that some problem Y is undecidable
- Prove that if X would be decidable then Y would be decidable, giving a codification of Y into X.

The halting problem of Turing machines is the standard example of an undecidable problem. The behaviour of a Turing machine, and the predicate that says that a given turing machine stops on all inputs can be expressed in FO.

Deciding the undecidable!

Deciding the undecidable!

- We just said that the problem of checking a first-order formula for satisfiability was undecidable.

Deciding the undecidable!

- We just said that the problem of checking a first-order formula for satisfiability was undecidable.
- Still, can we use a computer somehow to check if a formula is satisfiable?

Deciding the undecidable!

- We just said that the problem of checking a first-order formula for satisfiability was undecidable.
- Still, can we use a computer somehow to check if a formula is satisfiable?
  - YES!

Deciding the undecidable!

- We just said that the problem of checking a first-order formula for satisfiability was undecidable.
- Still, can we use a computer somehow to check if a formula is satisfiable?
  - YES!
  - Undecidable means that we cannot solve the problem for all first-order formulas, but we can solve it for some.

Deciding the undecidable!

- We just said that the problem of checking a first-order formula for satisfiability was undecidable.
- Still, can we use a computer somehow to check if a formula is satisfiable?
  - YES!
  - Undecidable means that we cannot solve the problem for all first-order formulas, but we can solve it for some.
  - Whenever we do get an answer SAT/UNSAT, this is useful information.
- We will learn that we can use resolution to decide whether a formula is satisfiable. But first we need to know what unification is.

What is Unification?

What is Unification?

- Goal: Identify two symbolic expressions.
- Method: Replace certain subexpressions (variables) by other expressions.

What is Unification?

- Goal: Identify two symbolic expressions.
- Method: Replace certain subexpressions (variables) by other expressions.

Example

- Goal: Identify $f(x, a)$ and $f(b, y)$.
- Method: Replace the variable x by b, and the variable y by a. Both initial expressions become $f(b, a)$.
- The substitution $\{x \mapsto b, y \mapsto a\}$ unifies the terms $f(x, a)$ and $f(b, y)$.

What is Unification?

- Goal: Identify two symbolic expressions.
- Method: Replace certain subexpressions (variables) by other expressions.

Example

- Goal: Identify $f(x, a)$ and $f(b, y)$.
- Method: Replace the variable x by b, and the variable y by a. Both initial expressions become $f(b, a)$.
- The substitution $\{x \mapsto b, y \mapsto a\}$ unifies the terms $f(x, a)$ and $f(b, y)$.
- Of course, one should know what expressions are variables, and what are not.

What is Unification Good For?

What is Unification Good For?

- To make an inference step in theorem proving.

What is Unification Good For?

- To make an inference step in theorem proving.
- To perform an inference in logic programming.

What is Unification Good For?

- To make an inference step in theorem proving.
- To perform an inference in logic programming.
- To make a rewriting step in term rewriting.

What is Unification Good For?

- To make an inference step in theorem proving.
- To perform an inference in logic programming.
- To make a rewriting step in term rewriting.
- To extract a part from structured or semistructured data (e.g. from an XML document).

What is Unification Good For?

- To make an inference step in theorem proving.
- To perform an inference in logic programming.
- To make a rewriting step in term rewriting.
- To extract a part from structured or semistructured data (e.g. from an XML document).
- For type inference in programming languages.

What is Unification Good For?

- To make an inference step in theorem proving.
- To perform an inference in logic programming.
- To make a rewriting step in term rewriting.
- To extract a part from structured or semistructured data (e.g. from an XML document).
- For type inference in programming languages.
- For matching in pattern-based languages.

What is Unification Good For?

- To make an inference step in theorem proving.
- To perform an inference in logic programming.
- To make a rewriting step in term rewriting.
- To extract a part from structured or semistructured data (e.g. from an XML document).
- For type inference in programming languages.
- For matching in pattern-based languages.
- For various formalisms in computational linguistics.

Conventions and Notation

Conventions and Notation

- x, y, z denote variables.
- a, b, c denote constants.
- f , g, h denote function.
- s, t, r denote arbitrary terms.

Conventions and Notation

- x, y, z denote variables.
- a, b, c denote constants.
- f , g, h denote function.
- s, t, r denote arbitrary terms.

Examples:

- $f(x, g(x, a), y)$ is a term, where f is ternary, g is binary, a is constant.

Substitutions

Substitutions

Substitution

- A mapping from variables to terms, where all but finitely many variables are mapped to themselves.

Substitutions

Substitution

- A mapping from variables to terms, where all but finitely many variables are mapped to themselves.

Example

A substitution is represented as a set of bindings:

- $\{x \mapsto f(a, b), y \mapsto z\}$.
- $\{x \mapsto f(x, y), y \mapsto f(x, y)\}$.

All variables except x and y are mapped to themselves by these substitutions.

Substitution Application

Substitution Application

Applying a substitution σ to a term t:

$$t\sigma = \begin{cases} \sigma(x) & \text{if } t = x \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, ..., t_n) \end{cases}$$

Substitution Application

Applying a substitution σ to a term t:

$$t\sigma = \begin{cases} \sigma(x) & \text{if } t = x \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Example

- $\sigma = \{x \mapsto f(x, y), y \mapsto g(a)\}$.
- $t = f(x, g(f(x, f(y, z))))$.
- $t\sigma = f(f(x, y), g(f(f(x, y), f(g(a), z))))$.

Substitution Application

Applying a substitution σ to a term t:

$$t\sigma = \begin{cases} \sigma(x) & \text{if } t = x \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, ..., t_n) \end{cases}$$

Example

- $\sigma = \{x \mapsto f(x, y), y \mapsto g(a)\}$.
- $t = f(x, g(f(x, f(y, z))))$.
- $t\sigma = f(f(x, y), g(f(f(x, y), f(g(a), z))))$.

A substitution σ is a unifier of the terms s and t if $s\sigma = t\sigma$.

Unifier, Most General Unifier

Unifier, Most General Unifier

Unification Problem: $f(x, z) \doteq^? f(y, g(a))$.

Unifier, Most General Unifier

Unification Problem: $f(x,z) \doteq^? f(y, g(a))$.

- Some of the unifiers:

$$\{x \mapsto y, z \mapsto g(a)\}$$
$$\{y \mapsto x, z \mapsto g(a)\}$$
$$\{x \mapsto a, y \mapsto a, z \mapsto g(a)\}$$
$$\{x \mapsto g(a), y \mapsto g(a), z \mapsto g(a)\}$$
$$\{x \mapsto f(x,y), y \mapsto f(x,y), z \mapsto g(a)\}$$
$$\ldots$$

Unifier, Most General Unifier

Unification Problem: $f(x,z) \doteq^? f(y,g(a))$.

- Some of the unifiers:

$$\{x \mapsto y, z \mapsto g(a)\}$$
$$\{y \mapsto x, z \mapsto g(a)\}$$
$$\{x \mapsto a, y \mapsto a, z \mapsto g(a)\}$$
$$\{x \mapsto g(a), y \mapsto g(a), z \mapsto g(a)\}$$
$$\{x \mapsto f(x,y), y \mapsto f(x,y), z \mapsto g(a)\}$$
$$\ldots$$

Most General Unifiers (mgu):

$$\{x \mapsto y, z \mapsto g(a)\}, \{y \mapsto x, z \mapsto g(a)\}.$$

- mgu is unique up to a variable renaming.

Unification Algorithm

Unification Algorithm

Goal: Design algorithm that for a given problem $s \doteq^? t$

- returns an mgu of s and t if they are unifiable,
- reports failure otherwise.

Unification Algorithm

Goal: Design algorithm that for a given problem $s \doteq^? t$

- returns an mgu of s and t if they are unifiable,
- reports failure otherwise.

Naive Algorithm: Write down two terms and set markers at the beginning of the terms. Then:

Unification Algorithm

Goal: Design algorithm that for a given problem $s \doteq^? t$

- returns an mgu of s and t if they are unifiable,
- reports failure otherwise.

Naive Algorithm: Write down two terms and set markers at the beginning of the terms. Then:

1. Move markers simultaneously, one symbol at a time, until both move off the end of the term (success), or until they point to two different symbols;

Unification Algorithm

Goal: Design algorithm that for a given problem $s \doteq^? t$

- returns an mgu of s and t if they are unifiable,
- reports failure otherwise.

Naive Algorithm: Write down two terms and set markers at the beginning of the terms. Then:

1. Move markers simultaneously, one symbol at a time, until both move off the end of the term (success), or until they point to two different symbols;
2. If the two symbols are both non-variables, then fail; otherwise, one is a variable (call it x) and the other one is the first symbol of a subterm (call it t):

Unification Algorithm

Goal: Design algorithm that for a given problem $s \doteq^? t$

- returns an mgu of s and t if they are unifiable,
- reports failure otherwise.

Naive Algorithm: Write down two terms and set markers at the beginning of the terms. Then:

1. Move markers simultaneously, one symbol at a time, until both move off the end of the term (success), or until they point to two different symbols;
2. If the two symbols are both non-variables, then fail; otherwise, one is a variable (call it x) and the other one is the first symbol of a subterm (call it t):
   2.1 If x occurs in t, then fail;

Unification Algorithm

Goal: Design algorithm that for a given problem $s \doteq^{?} t$

- returns an mgu of s and t if they are unifiable,
- reports failure otherwise.

Naive Algorithm: Write down two terms and set markers at the beginning of the terms. Then:

1. Move markers simultaneously, one symbol at a time, until both move off the end of the term (success), or until they point to two different symbols;
2. If the two symbols are both non-variables, then fail; otherwise, one is a variable (call it x) and the other one is the first symbol of a subterm (call it t):
   2.1 If x occurs in t, then fail;
   2.2 Else, replace x everywhere by t (including in the solution), print "$x \mapsto t$" as a partial solution. Go to 1.

Naive Algorithm

Naive Algorithm

- Finds disagreements in the two terms to be unified.
- Attempts to repair the disagreements by binding variables to terms.
- Fails when function symbols clash, or when an attempt is made to unify a variable with a term containing that variable.

Example

$f(x, g(a), g(z))$

$f(g(y), g(y), g(g(x)))$

Naive Algorithm

- Finds disagreements in the two terms to be unified.
- Attempts to repair the disagreements by binding variables to terms.
- Fails when function symbols clash, or when an attempt is made to unify a variable with a term containing that variable.

Example

$f(x, g(a), g(z))$

$f(g(y), g(y), g(g(x)))$

We can also unify formulas, we just consider them as if they were terms.