Lecture 12 Beyond (and Beneath) FOL

COMP24412: Symbolic AI

Giles Reger

March 2019

Aim and Learning Outcomes

The aim of this lecture is to:

Understand how FOL relates to other knowledge representation formalisms

Learning Outcomes

By the end of this lecture you will be able to:

1. Recall that adding arithmetic to FOL makes it fully undecidable
2. Recall a number of different formalisms and describe the general way in which they are related to FOL (e.g. more or less expressive)

Today

Beyond First-Order Logic

- Adding arithmetic
- Higher-order logic

Beneath First-Order Logic

- Fragments of FOL
- Description Logic
- Modal Logic

Arithmetic is Useful

People modelling in first-order logic usually assume that they have access to arithmetic. However, it does not come for free.

There are different ways of encoding arithmetic in first-order logic but the most general are undecidable.

Often full arithmetic is not required and it is better to encode orderings or counting in some other way.

Different Kinds of Arithmetic

Presburger Arithmetic has symbols $0, succ, +, =$ defined by some axioms

$$\begin{aligned} &0 \neq x + 1 \\ &(x + 1 = y + 1) \rightarrow x = y \\ &(x + 0) = x \\ &x + (y + 1) = (x + y) + 1 \end{aligned}$$

and *induction* e.g. for every formula $\phi[n]$

$$(\phi[0] \wedge \forall x.(\phi[x] \rightarrow \phi[x + 1])) \rightarrow (\forall y.\phi[y])$$

but induction is not finitely axiomitisable - we cannot represent arithmetic in first order logic.

However, by itself Presburg arithmetic is decidable.

Different Kinds of Arithmetic

Peano Arithmetic has symbols $0, succ, +, \times, =, \leq$ defined by some axioms

$$\forall x.(x \neq succ(x))$$
$$\forall x, y.(succ(x) = succ(y) \rightarrow x = y)$$
$$\forall x.(x + 0 = x)$$
$$\forall x, y.(x + succ(y) = succ(x + y))$$
$$\forall x.(x \times 0 = 0)$$
$$\forall x, y.(x \times succ(y) = x + (x \times y))$$
$$\forall x, y.(x \leq y \leftrightarrow \exists z.(x + z = y))$$

and *induction* e.g. for every formula $\phi[n]$

$$(\phi[0] \wedge \forall x.(\phi[x] \rightarrow \phi[succ(x)])) \rightarrow (\forall y.\phi[y])$$

which, again is not finitely axiomitisable.

Peano arithmetic is incomplete and undecidable.

Validity Modulo a Theory

Let $\Sigma_{\mathcal{T}}$ be an interpreted signature e.g. $+, \times, \geq, 1, 2, 3, \ldots$

Let a theory $\mathcal{T}$ over $\Sigma_{\mathcal{T}}$ be a class of interpretations that fix some interpretation for $\Sigma_{\mathcal{T}}$. Often this class is singular e.g. we only interpret $+$ in one way.

An interpretation is consistent with $\mathcal{T}$ if it is consistent on $\Sigma_{\mathcal{T}}$

A formula is consistent modulo $\mathcal{T}$ if it has a model consistent with $\mathcal{T}$. It is valid if it is true in all interpretations consistent with $\mathcal{T}$.

If we saturate then we can build a model, but it is not guaranteed that this model is consistent with $\mathcal{T}$. We cannot use model construction argument.

Reasoning with Arithmetic

Often we just use Integer Arithmetic with constants $0, 1, 2, 3, \ldots$ that interprets $+, -, \times, =\leq$ etc directly on these. When we add division we get an infinite set of models where division by 0 can be interepreted arbitrarily. Clearly, we can do as much as in Peano arithmetic.

Just add some things that we know are true e.g.

$$x + 0 = x$$
$$x + y = y + x$$
$$(x + y) + z = x + (y + z)$$
$$x + 1 > x$$

In Vampire we do other clever reasoning tricks (evaluation of ground things, calling out to ground decision procedures for sub-problems etc).

There are tools called Satisfiability Modulo Theories (SMT) solvers that work differently from Vampire by model building for quantifier-free problems with theories.

Quantifying over Functions/Predicates/Sets

There are some things we cannot say in first-order logic, for example the induction schema we saw earlier where we want to quantify over all predicates in the language.

Something we cannot express in first-order logic is reachability (the reason why is quite complex) but we can in higher-order logic

$$\forall P(\forall x, y, z \left(\begin{array}{l} P(x,x) \wedge \\ (P(x,y) \wedge P(z,y) \rightarrow P(x,z)) \wedge \\ (R(x,y) \rightarrow P(x,y)) \end{array} \right) \rightarrow P(u,v))$$

In program specification/verification we want to reason over objects representing programs and requirements (predicates on states).

Higher-order Logic

In first-order logic we have variables representing individuals and we can quantify over them.

In second-order logic we have variables representing sets of individuals (or functions on individuals) and we can quantify over them.

In third-order logic we have variables representing sets of sets of individuals...

We call second-order logic and above higher-order logic

λ-calculus

This is a calculus of functions. The standard building-block is the anonymous function $\lambda x.E[x]$ which can be read as a function that takes a value for x and evaluates E with x replaced by that value (similar to functions we're familiar with).

We can then have functions that take other functions as arguments and can return functions.

Functions can then be applied to each other e.g.

$$(\lambda x.\lambda y.xy)(\lambda x.x) \rightarrow_\beta (\lambda y.(\lambda x.x)y) \rightarrow_\eta \lambda x.x$$

This is a big and interesting topic but we don't have any more space here.

It is a useful term language for higher-order logic as λ-terms witness function variables

Reasoning in Higher-order Logic

Higher-order logic is clearly undecidable

Standard methods for reasoning in it are either...

Interactive e.g. using a proof assistant to allow a human to make reasoning steps (and suggesting possibly good reasoning steps)

or

Approximate by a translation to first-order logic that preserves inconsistency (this is what we do in Vampire)

Less Expressive/More Efficient

Find things that are decidable but still usefully expressive:

- Propositional logic (QBF, PLFD)
- Well-behaved First-order fragments
- (Some) Description Logics
- (Some) Modal Logics

Some Decidable Fragments of FOL

Monadic Fragment
Every predicate has arity at most 1 and there are no function symbols.

Two-variable Fragment
The formula can be written using at most 2 variables.

Guarded Fragment
If the formula is built using $\neg$ and $\wedge$, or is of the form $\exists \overline{x}.(G[\overline{y}] \wedge \phi[\overline{z}])$ such that G is an atom and $\overline{z} \subseteq \overline{y}$. Intuitively all usage of variables are *guarded* by a something positive.

Prenex Fragments
If a function-free formula is in prenex normal form and can be written as $\exists^* \forall^*.F$ it is in the BernaysSchönfinkel fragment.

The following logics often target these fragments to ensure decidability.

Description Logic

A family of logics that are usually decidable. They are used for describing ontologies. The terminology is different from what we're used to.

In description logic we separate facts in the $\mathcal{A}$-Box and rules in the $\mathcal{T}$-Box

Individuals belong to Concepts and may be related by Roles.

Concepts are sets of elements (unary predicates)

Roles relate two individuals (binary relations/predicates)

Complex concepts are logical combinations of concepts/roles

Facts assert individuals belong to concepts or roles

Rules capture relationships between complex concepts

Description Logic: Concepts

The most basic description logic

(In $\mathcal{ALC}$) Complex concepts:

- $A \sqcap B$: things that are A and B
- $A \sqcup B$: things that are A or B
- $\neg A$: things that are not A
- $\exists r.C$ things that are related by r to things that are C
- $\forall r.C$ things where all r related things are C

Examples:

Somebody that has a human child	$\exists$hasChild.Human
Somebody that only drinks beer	$\forall$drinks.Beer
Somebody that is either French or knows somebody who is	French $\sqcup \exists$knows.French

Rules and Reasoning

Rules are of the form $C \sqsubseteq D$ e.g. everything that is a C is also a D

$C \equiv D$ is the same as $C \sqsubseteq D$ and $D \sqsubseteq C$

e.g. $\text{Father} \equiv \text{Man} \sqcap \exists \text{hasChild.Human}$

An ontology is a set of facts and rules ($\mathcal{A}$-box and $\mathcal{T}$-box)

The semantics are defined in terms of interpretations (should be familiar)

Standard reasoning problems include

- Is an ontology consistent
- Is an individual in a concept (entailment)
- Is one concept subsumed by another (entailment)

Embedding in FOL

Introduce translation function t_x that maps into FOL formula with free x

Concepts map directly to FOL predicts, $t_x(A) = A(x)$

Complex Concepts map to logical combinations e.g.

$$t_x(\exists r.C) = \exists y.r(x, y) \wedge t_y(C)$$

The resulting FOL formulas are in the two-variable fragment and the guarded fragment.

This is for the simplest description logic $\mathcal{ALC}$. There are lots of more complicated description logics with extra features where the translation is less straightforward.

Propositional Modal Logic

It would be nice to be able to not only talk about what is true but when it is true (when in a general sense)

Modal logic allows us to do this. In English a *modal* qualifies a statement.

In modal logic we typically have two modal operators $\Diamond$ and $\Box$

Traditionally $\Diamond P$ means *Possibly* P whereas $\Box P$ means *Necessarily* P

For example,

$$\neg\Diamond win \rightarrow \Box\neg win$$
$$\Box(rain \wedge wind) \rightarrow \Box rain$$
$$(\Diamond rain \wedge \Diamond wind) \rightarrow \Diamond(rain \vee wind)$$

Semantics and Flavours

The semantics of modal logic is given by something called a Kripke structure, which is really just a graph. We have a relation R between worlds where different propositions are true in each world. $\Box$ then means *in all worlds adjacent by* R and $\Diamond$ means *in some worlds adjacent by* r.

We get different kinds of modal logic depending on how we control r, or equivalently which axioms about $\Box$ and $\Diamond$ we assume.

For example, reflexivity of R or $\Box A \rightarrow A$, and transitivity of R or $\Box A \rightarrow \Box\Box A$ gives us temporal logic where $\Box$ means all futures and $\Diamond$ means some

Other popular flavours in AI (particularly agent-based reasoning) are epistemic logic where modalities correspond to knowledge and doxastic logic where they correspond to belief.

Embedding in FOL

We use the adjacency relation R and a predicate $holds(x, y)$ that is true if x is true in world y

We can then encoding the meaning of modal formulas e.g.

$$holds(\Box p, u) \leftrightarrow \forall v.(R(u, v) \rightarrow holds(p, v))$$

The satisfiability/validity of a modal formula is the existential/universal closure of the resulting translation

We also need to add the axioms e.g. reflexivity and transitivity of R

We actually have to do quite a bit of extra work to get things into a decidable fragment, but we can

Decidability does not necessarily mean more efficient

A decision procedure is good because it will terminate in finite time, this does not mean that time is short.

Many fragments/logics also have strong complexity bounds on specific reasoning problems, which can help. But these are upper bounds.

In practice, it might be that a less efficient method, or an incomplete one, solves particular instances of problems faster.

This is generally our experience with Vampire.

Finally, Knowledge Engineering?

I have told you about lots of different formalisms and how to reason in some of them.

You have had to model different domains in some of these formalisms but there has been no general method for this.

At a high level *knowledge engineering* consists of the following steps:

1. Identify what the knowledge will be used for
2. Find the knowledge (where is it/who knows it)
3. Decide on a vocabulary e.g. names for predicates/functions etc
4. Decide on formalism to use
5. Encode general/domain knowledge
6. Encode the specific problem
7. Use the knowledge base (pose queries)
8. Debug the knowledge base

Summary of Part 1

Datalog: syntax, model-based semantics, matching, forward chaining

Prolog: more syntax, unification, backward chaining

First-order logic: syntax, model-based semantics, clausal representation/translation, resolution and paramodulation, completeness argument, orderings and ordered resolution, given clause algorithm

Relation to other logics (today)