LSS 2018: Computability and Incompleteness

3. Logic and (In)Computability

Michael Norrish

Michael.Norrish@data61.csiro.au

Outline

Last Time…

Results in Computability:

- Turing Machines and Recursive Functions coincide.
  - Alternatively, each can simulate the other
- It's impossible to tell if a computation will finish (Halting Problem)
- It's impossible to determine if a machine/function has any particular extensional property (Rice's Theorem)
- Recursive Enumerability is the limit of computability…
  - but it's possible to say things about degrees of hardness beyond that.

Logic and Computability

Logic is not just a tool for human use.

Automating logical reasoning is a very productive activity:

- software verification
- hardware verification
- mechanised mathematics

But automation happens on computers, and perhaps computer logic is necessarily limited. . .

Outline

First Order Logic

Syntax:

- Term variables: $x, y, z, \ldots$
- Function Symbols (make terms from other terms): $f, g, h, \ldots$
- Predicate Symbols (make formulas of terms): $P, Q, R, \ldots$
  - Equality (fixed binary predicate): $t_1 = t_2$
- Propositional Connectives (make formulas of formulas): $\wedge, \neg, \ldots$
- Quantifiers (over/binding term variables): $\forall x, \exists y, \ldots$

Function and predicate symbols have arities.
The arity of a symbol is the number of arguments it can take.
Allowing arities of zero is OK.

First Order Logic

Semantics:

- A closed formula is interpreted with respect to an interpretation that
  - specifies a domain $\mathcal{D}$
  - maps function symbols of arity n into functions $\mathcal{D}^n \rightarrow \mathcal{D}$
  - maps predicate symbols of arity m into predicates $\mathcal{D}^m \rightarrow \mathbb{B}$
- Truth value of the closed formula is given recursively over its structure.

  For example:
  - $\mathcal{I}(\phi \wedge \psi)$ is true iff $\mathcal{I}(\phi)$ is true and $\mathcal{I}(\psi)$ is true.
  - $\mathcal{I}(\forall x.\phi)$ is true if $\mathcal{I}(\phi)$ is true of all elements $d \in \mathcal{D}$. (Sloppy alert!)

Write $\models \Phi$ if Φ is true in all interpretations ("valid").

First Order Logic

FOL has Proof Systems:

- Axiomatic ("Hilbert") System
- Natural Deduction
- Sequent Calculus
- ...

A proof system defines derivability/provability relation $\vdash$

Soundness and completeness (proved in another course!):

if $\vdash \Phi$ then $\models \Phi$ if $\models \Phi$ then $\vdash \Phi$

Derivable is Enumerable

Easily seen from axiomatic systems:

- Can enumerate all possible formulas.
- Can enumerate all possible instantiations of the axiom schemes.
- Can enumerate all possible applications of inference rules to theorems.

Thus, There is a Semi-Decision Procedure for Validity

I wish to determine if Φ is valid.

1. Run my favourite theorem-enumerator.
2. Wait for Φ to appear . . .
3. If it does, say "Yes!"

The "Yes!" result means $\vdash \Phi$, and soundness means $\models \Phi$.

Conversely, if Φ is valid, then $\models \Phi$ and completeness means $\vdash \Phi$, and my enumerator *will* get to Φ eventually.

Not a Decision Procedure!

I want to decide validity of Φ.

How about:

1. Run my favourite theorem-enumerator.
2. If Φ appears, say “Yes!”
3. If $\neg\Phi$ appears, say “No!”

Why doesn’t this work?

Not a Decision Procedure! (continued)

[Enumerating theorems, and waiting for Φ or $\neg\Phi \cdots$]

Example: $\forall x.\, R(x,x)$ is not valid.

- R might be interpreted by something that is not reflexive

But: $\neg(\forall x.\, R(x,x))$ (equivalent to $\exists x.\, \neg R(x,x)$) is not valid either.

- R might be interpreted by something that *is* reflexive.

In general, the mistake was to imagine that $\not\models \Phi$ implied $\models \neg\Phi$.

(Validity ($\models$) has a hidden universal over interpretations inside!)

Validity in First Order Logic is Not Decidable

Alonzo Church
(1903–1995)

First shown by Church.

- Turing's PhD supervisor.
- Inventor of the λ-calculus.
- Author of Church's Thesis.

Valid sentences are recursively enumerable.

Proof that valid sentences are not recursive is by reduction to the Halting Problem.

Reduction to the Halting Problem

[As with Rice's Theorem.]

Proof by contradiction.

Assume we can solve our problem.

Show that this results in us being able to solve the Halting Problem too.

Conclude that we can't solve the original problem.

Reduction to the Halting Problem for FOL Validity

Assume we can decide $\models \Phi$ for all Φ

- We are given machine M with input n.
- We will determine if it halts or not.

Trick is to encode "do you halt?" in first order logic.

Encoding Turing Machine Computation in FOL

[Multiple approaches possible. This one is from B&J.]

Have two function symbols:

- $\mathbf{0}$ — function symbol of arity zero (stands for 0)
- s — function symbol of arity one (stands for successor)

Tape is indexed with integers (infinite in both directions).

Have one binary predicate per machine-state (Q_i),
and one binary predicate per tape-symbol (S_j),
and binary predicate $<$.

- $Q_i(t, p)$ — at time t, machine is in state i and position p on the tape
- $S_i(t, p)$ — at time t, tape holds symbol i at position p
- $i < j$ — i is less than j

Encoding the Initial State

[Assume initial machine state is 0 *and initial input is* n*.]*
At time $t = 0$, machine is initially in state 0 at tape position 0:

$$Q_0(\mathbf{0}, \mathbf{0})$$

At time $t = 0$, tape positions $0 \ldots n - 1$ are filled with symbol 1:

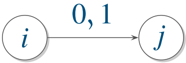
$$\bigwedge_{i \in 0 \ldots n-1} S_1(\mathbf{0}, s^i(\mathbf{0}))$$

All other tape positions are filled with symbol 0:

$$\forall p. \left(\bigwedge_{q \in 0 \ldots n-1} p \neq q \right) \Rightarrow S_0(\mathbf{0}, p)$$

Encoding the Turing Machine

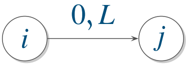
$$
\begin{aligned}
\forall t\,p\,q.\ & Q_i(t,p) \wedge S_0(t,p) \ \Rightarrow \\
& \quad Q_j(s(t),s(p)) \wedge (S_0(t,q) \Rightarrow S_0(s(t),q)) \wedge \\
& \quad (S_1(t,q) \Rightarrow S_1(s(t),q))
\end{aligned}
$$



$$
\begin{aligned}
\forall t\,p\,q.\ & Q_i(t,p) \wedge S_0(t,p) \ \Rightarrow \\
& \quad Q_j(s(t),p) \wedge S_1(s(t),p) \wedge \\
& \quad (q \neq p \wedge S_0(t,q) \Rightarrow S_0(s(t),q)) \wedge \\
& \quad (q \neq p \wedge S_1(t,q) \Rightarrow S_1(s(t),q))
\end{aligned}
$$

Encoding the Machine: Moving Left and Integers

With just a successor function, how do we talk about going leftwards, even unto negative positions?



$$\begin{aligned}\forall t\,p\,q.\ & Q_i(t,s(p)) \wedge S_0(t,s(p)) \;\Rightarrow \\ & Q_j(s(t),p) \wedge (S_0(t,q) \Rightarrow S_0(s(t),q)) \wedge \\ & (S_1(t,q) \Rightarrow S_1(s(t),q))\end{aligned}$$

Still need to assert that every number is the unique successor of another:

$$\forall n.\ \exists m.\ (n = s(m)) \wedge \forall p.\ (n = s(p)) \Rightarrow (m = p)$$

And properties of $<$:

$$\forall x\,y\,z.\ (x < y \wedge y < z \Rightarrow x < z) \wedge \neg(x < x) \wedge x < s(x)$$

Encoding the Question

Have machine description, and some super-minimalist arithmetic. Call all this Δ.

Add H:

$$\bigvee_{(i,j)\in\mathcal{H}} \exists t\, p.(Q_i(t,p) \wedge S_j(t,p))$$

where $\mathcal{H}$ is set of state-symbol pairs where machine specifies no action.

Halting Question: $\Delta \Rightarrow H$

Implication 1

If $\models \Delta \Rightarrow H$, then it is true for all interpretations of the symbols Q_i, S_j, $<$, $\mathbf{0}$ and s.

In particular, it is true for the "machine interpretation" we have been using/assuming.

So the given Turing Machine does halt when given the specified input.

Implication 2

This is harder.

If the Turing Machine does halt, we need to show that $\models \Delta \Rightarrow H$.

- *i.e.*, the statement is true in all interpretations.

Thanks to completeness, suffices to show $\vdash \Delta \Rightarrow H$

If the Turing Machine halts, it does so in some number of steps, n.

Will prove our result by induction on step-count.

Descriptions of Moments of Time

A description of time t is

- a ground formula describing the machine and state tape at time t
- machine state captured by $Q_i(t,p)$
- tape state captured by
  - $\bigwedge_{p \in P} S_i(t,p)$ (for various S_i); and
  - $\forall q.\ q \notin P \Rightarrow S_0(t,q)$

Set P will be finite set of positions touched by machine so far (including p).

Negative Tape Positions: P may include negative numbers.
If we want to write, for example, $Q_i(t,-n)$, we do it by writing:

$$(\exists m.\ \mathbf{0} = s^n(m) \wedge Q_i(t,m))$$

Description of the End of the Run

If the machine halts in n steps at position p on the tape, in state q and looking at symbol i, then a correct description will include:

$$Q_q(n, p) \wedge S_i(n, p)$$

And this will imply one of the disjuncts of H.

The Induction

For all times $t \leq n$, machine description Δ implies a correct description of time t.

- Implication must be in all possible interpretations.

Proof is by induction on t.

Base case is that we have a correct description of initial state (at $t = 0$).

- Trivially true as Δ includes it by construction.

Induction's Step-Case

Inductive Hypothesis: Have a correct description of time t. (Alternatively, Δ implies that correct description.)

- $t + 1 \leq n$, so machine has not halted at time t.

Need to show that Δ implies a correct description of the machine at time $t + 1$.

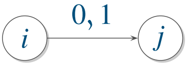
The machine at time t is in state q, at position p and with tape state captured by

- $\bigwedge_{p \in P} S_i(t, p)$ (for various S_i); and
- $\forall q.\ q \notin P \Rightarrow S_0(t, q)$

Step-Case: Write Action

Have: $Q_i(t, p) \wedge S_0(t, p)$ in description of time t.

Have this in Δ:



$$\begin{aligned}\forall t\, p\, q.\;\; & Q_i(t,p) \wedge S_0(t,p) \;\Rightarrow \\ & \quad Q_j(s(t),p) \wedge S_1(s(t),p) \wedge \\ & \quad (q \neq p \wedge S_0(t,q) \Rightarrow S_0(s(t),q)) \wedge \\ & \quad (q \neq p \wedge S_1(t,q) \Rightarrow S_1(s(t),q))\end{aligned}$$

First two conjuncts of conclusion give us correct description of machine-state and symbol at machine-position

Rest of required description is of rest of tape.

Step-Case: Write Action (continued)

Have: $Q_i(t,p) \wedge S_0(t,p)$ in description of time t.

Also have: $S_j(t,p')$, for various j and p'.

From Δ, have:
$$\begin{array}{ll}\forall q. & (q \neq p \wedge S_0(t,q) \Rightarrow S_0(s(t),q)) \wedge \\ & (q \neq p \wedge S_1(t,q) \Rightarrow S_1(s(t),q))\end{array}$$

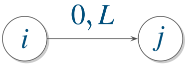
Imagine (for example) $p = 2$, $q = -3$, and $S_1(t,q)$.

- Handling of negative numbers means we actually have
  $\exists q'.\ \mathbf{0} = s(s(s(q'))) \wedge S_1(t,q')$
- Want: $\exists q'.\ \mathbf{0} = s(s(s(q'))) \wedge S_1(s(t),q')$
- Suffices: $\mathbf{0} = s(s(s(q')))) \Rightarrow q' \neq s(s(\mathbf{0}))$
- Follows from properties of $<$.

Step-Case: Head Movement

Have: $Q_i(t,p) \wedge S_0(t,p)$ in description of time t.

Have this in Δ:



$$\begin{array}{l} \forall t\,p\,q.\;\; Q_i(t,s(p)) \wedge S_0(t,s(p)) \;\Rightarrow \\ \qquad Q_j(s(t),p) \wedge (S_0(t,q) \Rightarrow S_0(s(t),q)) \wedge \\ \qquad (S_1(t,q) \Rightarrow S_1(s(t),q)) \end{array}$$

By first arithmetic assumption, the actual p is the successor of some p_0; will instantiate p in movement assumption above with p_0.

Contents of tape are easy, except perhaps for case when move has take machine into hitherto unvisited part of tape.

Visiting New Parts of Tape

Description at time t says

$$\forall q.\ q \neq p_1 \wedge \cdots \wedge q \neq p_m \Rightarrow S_0(t, q)$$

(where p_i values are visited positions to date).

Want to establish $S_0(s(t), p_0)$ for new, concrete p_0 value.

Movement assumption has $\forall q.\ S_0(t, q) \Rightarrow S_0(s(t), q)$.

So just have to establish $p_0 \neq p_1 \wedge \cdots \wedge p_0 \neq p_m$

As before, arithmetic assumptions will get us there.

Outline

Enumerability Inverted

Earlier claimed that *"derivable is enumerable"*, and *"can enumerate all possible formulas"*.

If there is an onto function $\mathbb{N} \rightarrow \alpha$, then there must be an injective function $\alpha \rightarrow \mathbb{N}$.

So, we can convert formulas into natural numbers.

- In fact there are infinitely many ways of doing this.

Manipulating Everything Numerically

The point of turning formulas into numbers is to allow numbers to "stand" for formulas.

- A system that only knows about numbers can still then have "Formula Manipulating Power"

But "manipulation" means doing stuff to formulas, not just having them hang around.

Manipulation means (for example):

- building new formulas from old ones
- doing instantiation of variables
- determining the type of a formula
- pulling formulas apart

Arithmetisation

Choose our Gödel Numbering so as to make it possible (i.e., computable!) to define formula manipulations as arithmetic functions.

We thereby provide an arithmetisation of syntax.

Computable functions will be able to manipulate more than just numbers.

- even though all they're doing is manipulating numbers
- (compare: modern computers as bit-twiddlers)

Outline

A Change of Scene

A first order logic with

- a fixed "non-logical language" ($\mathbf{0}$, $+$, $\cdot$, s)
- a fixed intended interpretation (arithmetic)
- a fixed set of simple axioms

With arithmetic, the really interesting incompleteness results arise.

The logic $\mathcal{Q}$ is minimalist: the interesting incompleteness results about it will apply to all stronger logics too.

$\mathcal{Q}$'s Axioms

Seven Axioms:

- $\forall x\, y.\ s(x) = s(y) \Rightarrow x = y$
- $\forall x.\ \mathbf{0} \neq s(x)$
- $\forall x.\ x \neq \mathbf{0} \Rightarrow \exists y.\ x = s(y)$
- $\forall x.\ x + \mathbf{0} = x$
- $\forall x\, y.\ x + s(y) = s(x + y)$
- $\forall x.\ x \cdot \mathbf{0} = \mathbf{0}$
- $\forall x\, y.\ x \cdot s(y) = (x \cdot y) + x$

Interpretation: arithmetic over the natural numbers.

As axioms are true in the given interpretation, so too are all of their consequences (by soundness of FOL).

What $\mathcal{Q}$ Is Not

Strong: can't even prove that addition is commutative.

Peano Arithmetic: PA includes the axiom (scheme) for natural number induction.

- Induction allows the proof of all sorts of nice properties

Summary

First Order Logic:

- Sound & complete, with computable rules of inference.
- Thus: recursively enumerable (semi-decidable).
- Expressive enough to capture behaviour of a Turing Machine.
- Thus: undecidable.

Gödel Numbering:

- Can convert formulas into numbers
- Can (computably) perform formula operations within arithmetic

The Logic $\mathcal{Q}$

- A basis for incompleteness results to come.