Inference in first-order logic

Note Title

Topics today:
1. Substitutions
2. Propositionalization
3. Completeness and decidability of FOL [9.2.1] arithmetic

} based closely on textbook 9.1

not based on textbook, but still required for this course

(1) Substitutions

Notation for substitutions:

- $\{x/A, y/B, z/C\}$

  means replace x with A, y with B, z with C.

- $\text{SUBST}(\{x/A\}, \alpha)$

  means replace x with A in sentence α.

examples:

- $\text{SUBST}(\{x/\text{John}, y/\text{Mother(John)}\}, \text{Knows}(x, y))$

  becomes do as exercise

- $\text{SUBST}(\{x/z, y/z\}, \text{Knows}(x, y))$

  becomes do as exercise

② Propositionalization

Recall that a major motivation for introducing FOL was that PL required too many facts

e.g.

PL: CSMajor(Daviella) ⋮ CSMajor(Chris) } one fact for every student

FOL: $\forall x$ CSMajor(x)

But, we can go back the other way if desired.
i.e. transform from FOL to PL by eliminating quantifies and variables.
This requires application of two obvious rules:

UI (Universal Instantiation)

For any variable v and ground term g and sentence α:

ground term g: a term with no variables

$$\frac{\forall v\ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

Example: Objects are students in this class.
Relation CSMajor(x), function ProjectPartner(x)
Knowledge base is:

$$\forall x \quad CSMajor(x)$$

Then we can use UI to add any or all of the following to the KB:

CSMajor(Omar)
CSMajor(Sam)
CSMajor(ProjectPartner(Omar))
CSMajor(ProjectPartner(ProjectPartner(Omar)))

Existential Instantiation (EI)

for any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base

$$\frac{\exists v \ \alpha}{SUBST(\{v/k\}, \alpha)}$$

Example: Similar scenario to last example,
Objects are students in this class.
Relation CSMajor(x), Relation IsProjectPartner(x, y)

not the same as the function in the earlier example.

But now suppose entire KB is:

CSMajor (Omar)
CSMajor (Justine)
$\exists x$ AstronomyMinor(x) — ✡
$\exists x$ IsProjectPartner (Justine, x) — *

Applying EI to ✡ we can add to the KB:

AstronomyMinor (Someone) — (++)

or even:

AstronomyMinor (Cole) ← Cole is just an arbitrary label here. It might not be the 'real' Cole

but not:

AstronomyMinor (Justine)

but what if Justine were an Astronomy minor? Statement (++) covers that! 'Justine' and 'Someone' can refer to the same object.

Applying EI to (*) we can add to the KB:

IsProjectPartner (Justine, Someone2)

or

IsProjectPartner (Justine, Cole2) ← seems wrong. Why is it OK? See comments above

Remark The above rules can convert any KB from FOL to PL. Then we can apply our PL resolution technique to infer entailment. But there are two problems:

(1) KB could become very big. It's better to do inference within FOL rather than converting to PL. The textbook explains how to do this (e.g. there is an FOL version of resolution in Section 9.5) but we don't study it.

(2) Functions lead to infinitely many possible substitutions
e.g. Father(x), Father(Father(x)), Father(Father(Father(x)))
etc.

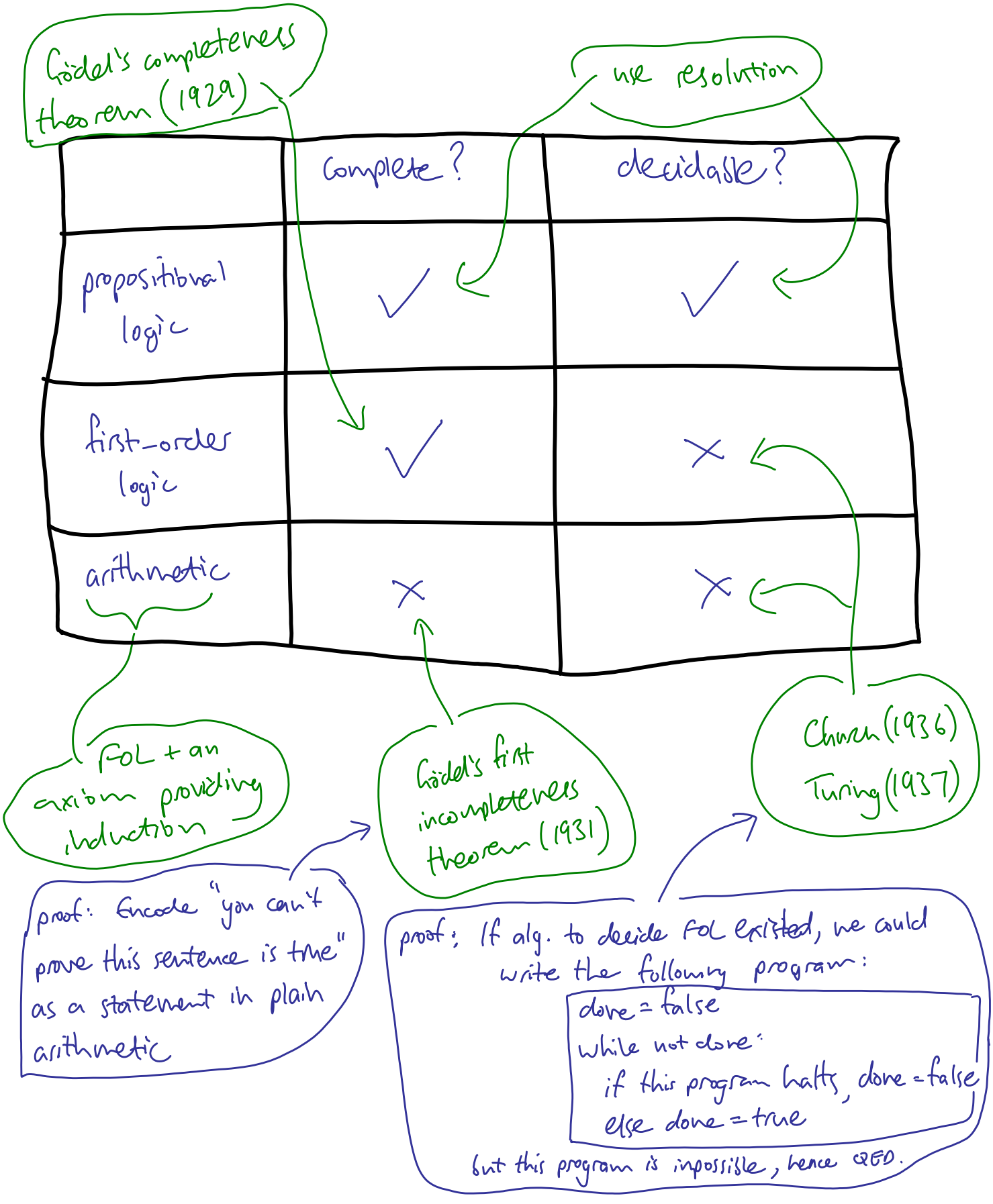
In practice this means that
- any entailed sentence can be proved
- we cannot, in general, prove that a sentence is not entailed.

Quoting from textbook p325:

"The question of entailment for first-order logic is semidecidable — that is algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence."

③ Summary of completeness and decidability for PL, FOL, arithmetic

- See next page for summary
- definition of <u>complete</u>: every valid statement has a proof
- definition of <u>decidable</u>: there exists an algorithm that decides whether a given statement is valid


Gödel's completeness theorem (1929)
use resolution
complete?
decidable?
propositional logic
first-order logic
arithmetic
FOL + an axiom providing induction
Gödel's first incompleteness theorem (1931)
Church (1936)
Turing (1937)
proof: Encode "you can't prove this sentence is true" as a statement in plain arithmetic
proof: If alg. to decide FOL existed, we could write the following program:
done = false
while not done:
if this program halts, done = false
else done = true
but this program is impossible, hence QED.