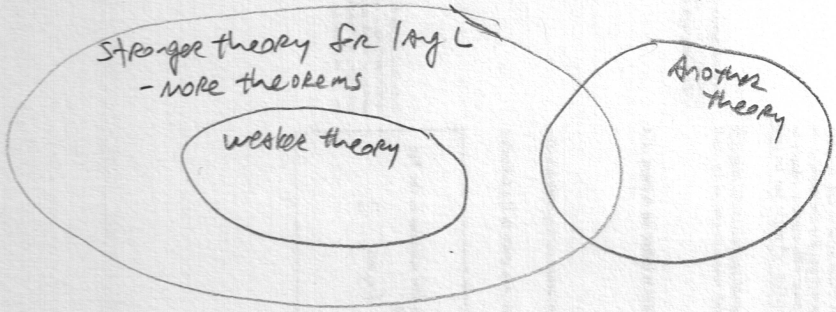
Classical FOL, with or without functors or equality

Language: choice of nonlogical symbols (sent constants/predicates, term constants/functors)

lang of arithmetic ( —— / perhaps $<$, $Z$ / $S$ / $+$, $\bullet$, perhaps $\uparrow$ )

---




Any model of the stronger theory will make true all its theorems & so too all theorems of the weaker theory.

So M is a model of stronger $\rightarrow$ M is a model of weaker

but: $\phi$ provable in stronger $\leftarrow$ $\phi$ provable in weaker

Theory
- set of sentences closed under $\vdash$ / $\models$
- may/may not be consistent

sentence $\phi$ decided by a theory $T$
- either $T \vdash \phi$ or $T \vdash \sim\phi$
- either $\phi \in T$ or $\sim\phi \in T$

Theory $T$ is complete
- it decides every sentence in its language
- Also has to be consistent? i.e. exactly one of $\phi, \sim\phi$ is a theorem?

---

weak arithmetics

stronger theory ↑

"medium" arithmetics (Robinson, Shoenfield, Burgess)

[FO] Peano arithmetic

[other FO axiomatizations of arith neither stronger nor weaker than Peano]

---

Theory of a model M = all sentences satisfied by/true on that model.

Std model of lang of arithmetic — its theory { "true arith" / "complete arith" / natural arithmetic

$\infty$ many sentences (but so too other ariths)

strongest FO arithmetic
- will turn out not to be (finitely or even decidably) axiomatizable
- is complete

## Want formal theories to be

1. consistent
2. Axiomatizable – finitely or at least decidably (easy to axiomatize inconsistent theories)
3. Complete – theory decides all sentences of its language
   MAX consistent
   exhaustive
4. decidable what its theorems are/what's provable from it
   (even though FOL not in general decidable, ⇏ all FO theories are)
   (Axiom + complete $\xrightarrow{CT}$ decidable)

---

Some theories (eg the weak arith) – can achieve all
w/ smaller langs

Middly + Arith – not achievable (only 1+2)
Natural Arith (1+3+4, not 2)

What is it about these theories that makes that impossible?

---

proof systems = syntactic manipulation

syntactic properties/transformations can be mechanical/effective

[in fact only need limited resources of "prim recursive" funcs

strings
syntactic properties
syntactic transformations

⟷

$\mathbb{N}$ and
operations on $\mathbb{N}$s
Arithmetic

Arithmetic strong enough to $\boxed{\text{"capture/repr/define"}}$ enough syntax to prove even things abt what's provable in that very theory

"Arithmetization of syntax"

list of $\phi$s $\rightarrow$ list of #s $\rightarrow$ single #

$\phi \rightarrow$ #

$\forall x_1\ Z \neq Sx_1 \supset Z \neq SZ$

strict about ( )s
no $\neq$
no infix relations
can avoid need for ( ) by using Polish notation

$(1+3)\cdot 2$ infix
$\cdot + 1 3\ 2$ Polish
$1 3 + 2 \cdot$ Reverse polish

$(\forall \boxed{x_1} \sim Z = S \boxed{x_1} \supset \sim Z = SZ)$

list of #s $\rightarrow$ single #
eg. by $2^{1st} \cdot 3^{-} \cdot 5^{-} \cdot 7^{-} \cdot$ etc

---

$\bar{n}$ or $S^n Z$ (# into std term for that # in formal theory)

$\#\phi$ or $\ulcorner \phi \urcorner$ (translate that expression from formal language into a #)

$\overline{\#\phi}$ or $S^{\ulcorner \phi \urcorner} Z$ (... and then back into the formal language's std term for that #)

general models of computability / effective / calculable

- lambda calculi (various type systems... untyped)
- formal automata (DFAs/NFAs, PDAs, ... Turing machines)
- theory of "Recursive functions" (Kleene, μ-recursive, general recursive)

Base functions

$K_0(x) = 0$ $\quad K_0^{2\text{Adic}}(x_1, x_2) = 0 \quad K_0^{3\text{Adic}}(x_1, x_2, x_3) = 0$

$K_3^{1\text{Adic}}(x) = 3$

$\text{Succ}(x) = x+1$

$Id_j^{n\text{-Adic}}(x_1, \dots x_n) = x_j$ $\quad Id = Id_1^{1\text{-Adic}}(x_1) = x_1$

$Id_1^{2\text{Adic}}(x_1, x_2) = x_1 \quad Id_2^{2\text{Adic}}(x_1, x_2) = x_2$

Combining functions

general composition $\boxed{g^{n\text{-Adic}} \circ (f_1^{k\text{-Adic}}, \dots f_n^{k\text{-Adic}})}$ familiar composition $g \circ f(x) = g(f(x))$

maps $(x_1 \dots x_k) \mapsto g(f_1(x_1 \dots x_k), \dots, f_n(x_1 \dots x_k))$

general recursion

$\text{Rec}\left[\text{base}^{k\text{-Adic}}, \text{step}^{(k+2)\text{Adic}}\right]$

familiar recursion $f(n)$ $\begin{cases} \text{when } n = 0 \text{ is } \underline{\text{base}} \\ \text{when } n \text{ is } j+1 \text{ is } \underline{j+1 \dots f(j) \dots} \end{cases}$

maps $(0, x_1 \dots x_k) \mapsto \text{base}(x_1 \dots x_k)$

maps $(j+1, x_1 \dots x_k) \mapsto \text{step}(j+1, \text{Rec}[..](j, x_1 \dots x_k), x_1 \dots x_k)$

primitively recursive functions

can define

bounded min $\left[f^{(k+1)\text{Adic}}, \text{bound}\right]$

maps $(x_1 \dots x_k) \mapsto$ $\begin{cases} \text{least } y < \text{bound where } f(y, x_1 \dots x_k) = 0 \\ \text{else bound} \end{cases}$

(5)

Unbounded MINIMIZATION ($\mu$) $[f^{(k+1)}]$

MAPS $(x_1 \ldots x_k) \mapsto$

$$\begin{cases} \text{least } y \text{ where } f(y, x_1, \ldots x_k) = 0 \\ \quad \text{and for all } y' < y: f(y', x_1, \ldots x_k) \text{ defined (so } > 0) \\ \text{else (there's no } y \text{ where } f(y, x_1, \ldots x_k) = 0, \\ \quad \text{or at least none before } f(-, x_1, \ldots x_k) \text{ is undefined)} \\ \text{is undefined} \end{cases}$$

[Kleene Normal Form Theorem]

primitively recursive functions - can be total (always defined)

$\downarrow \subsetneq$

recursive functions $\Big\langle$ total $\downarrow \subsetneq$ partial