符号学习

-2- 逻辑的边界

(Press ? for help, n and p for next and previous slide)

戴望州

南京大学智能科学与技术学院

2025年-秋季

https://daiwz.net

逻辑

逻辑是什么？

逻辑的作用是**系统性地检验论证的有效性**

1. 什么叫“论证”（arguments）？
2. 什么叫“检验”（evaluation）？
3. 什么叫“有效性”（validity）？
4. 什么叫“系统性地”（systematically）？

“系统性地检验”

在数理逻辑课上，我们认识到：

1. “前提”来自各个领域，其正确性和“逻辑”本身无关
2. 逻辑学家关心的是：“从某些（假定为真的）论据出发，究竟能不能得到最终的结论”

“系统性地检验”

在数理逻辑课上，我们认识到：

1. ~~逻辑学家关心的是：“从某些（假定为真的）论据出发，究竟能不能得到最终的结论”~~
   » 语法：一阶语言 $\mathcal{L}$ 与逻辑公理 Λ
2. ~~“前提”来自各个领域，其正确性和“逻辑”本身无关~~
   » 语义：理论 Γ 与结构 $\mathfrak{A}$

语法（Syntax）

1. 词汇表、符号
   - 逻辑词、非逻辑词
     - 连词、布尔函数、连词的完全组
   - 等词
2. 表达式
   - **合式公式（wff）**
   - 归纳原理（结构归纳）、括号引理、唯一可读性

语义（Semantics）

什么叫做“真”？

- 命题词、素公式、真值指派
- Tarski：“结构”与“解释”

有效性

论证的过程是否**滴水不漏**?

有效性

$$\Gamma \models_{\mathfrak{A}} \varphi[s]$$

有效性

$$\Gamma \models \varphi$$

论证：$\vdash$

定义（*derivation*）：

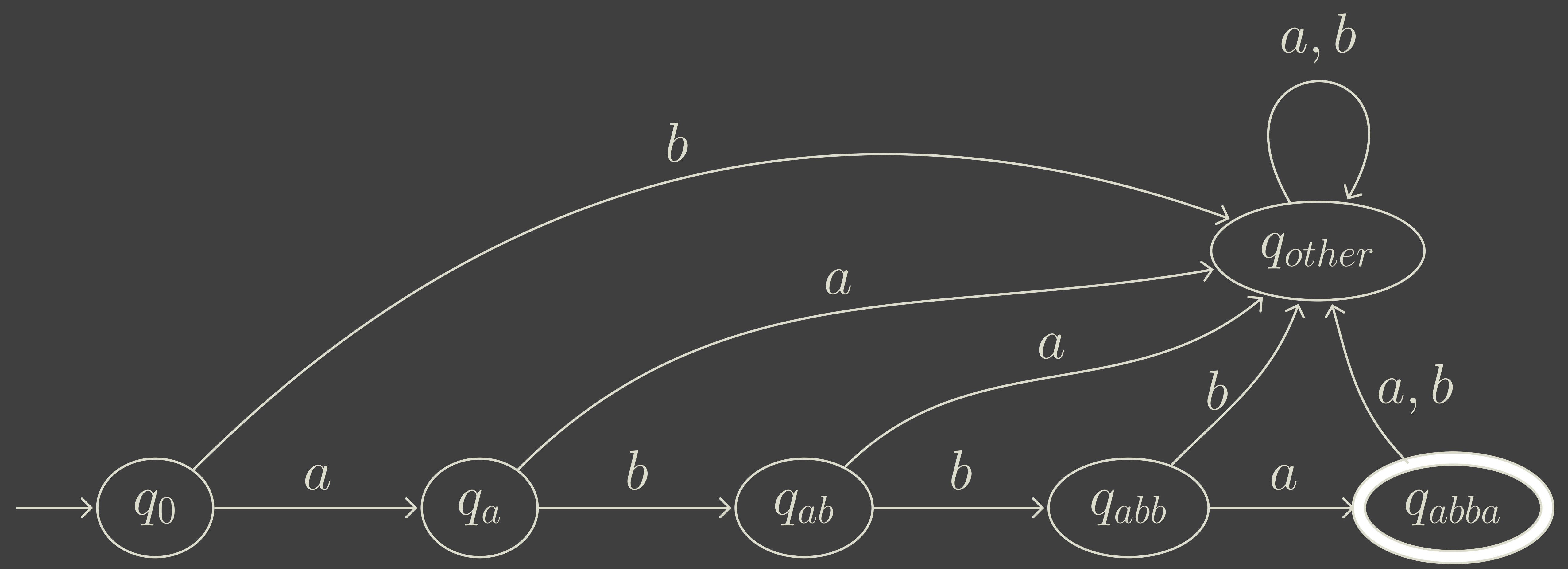
若Γ是一个wff集合，那么Γ中的一个推导是一个wff有穷序列$\langle \alpha_1, \ldots, \alpha_n \rangle$，对任意$i \leq n$有下面三者之一成立：

1. $\alpha_i \in \Gamma$
2. α_i是一条公理
3. α_i根据推理规则从某些α_j（及α_k）得到，其中$j < i$（且$k < i$）
   - 唯一的规则：若$\beta \to \alpha$和β同时出现在推导的**前段**，那么β也是正确的
   - 推理规则（*rule of inference*）给出了推理步骤正确性的充分条件。它其实是一种“元规则”，即一种模式，并不只是一条规则

计算

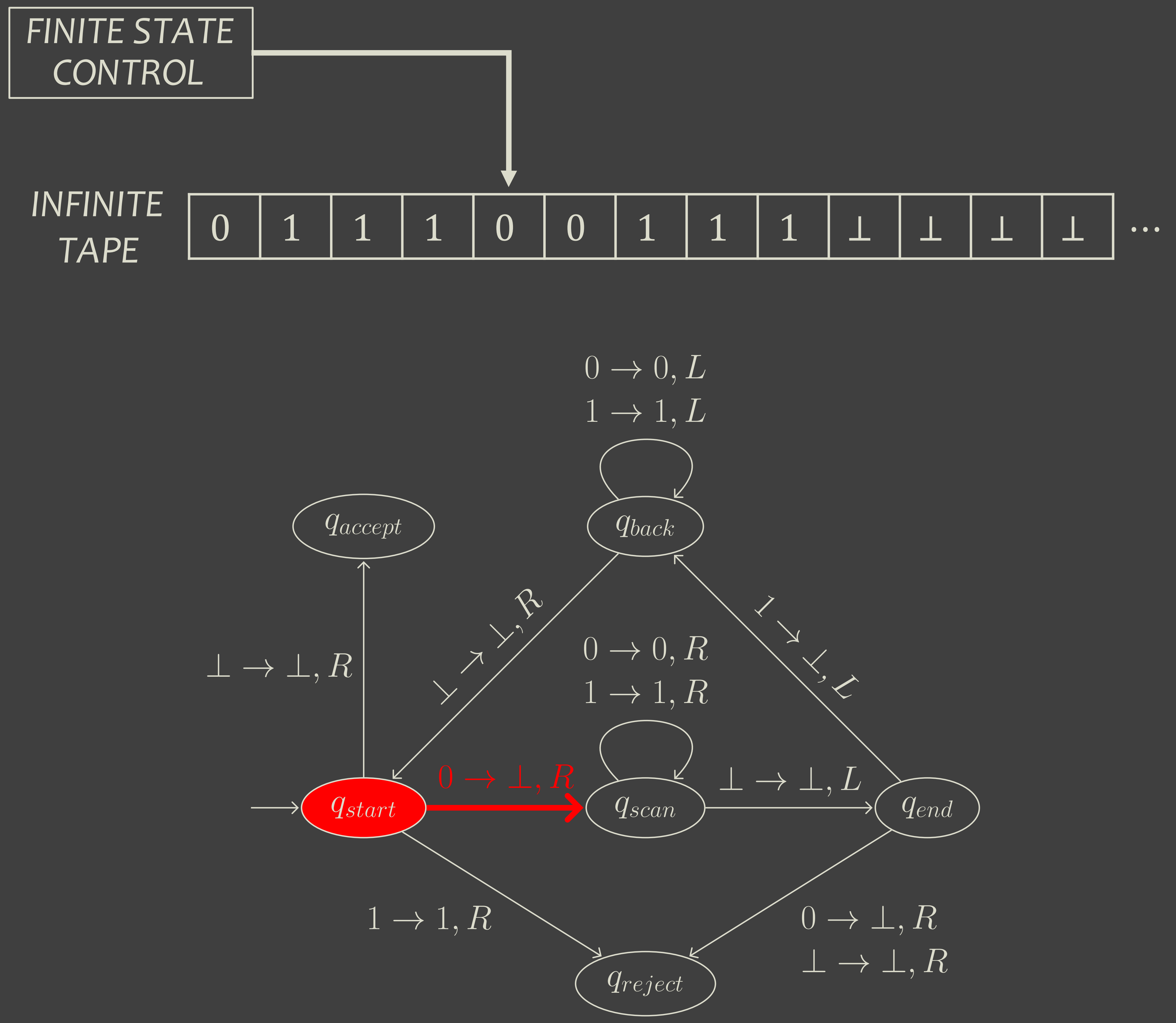
离散状态机

断言：“计算”可以用**离散状态机（discrete state machines）**表示

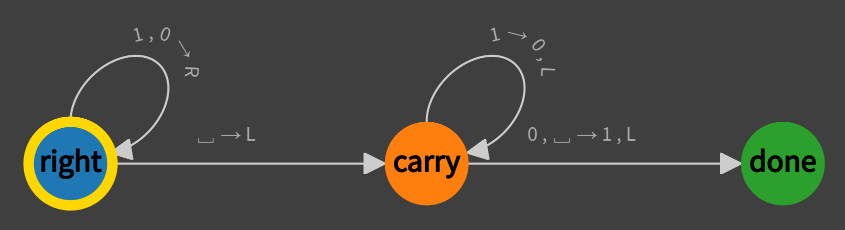


想像你手下有一个足够“愚蠢”的计算员（computer），TA能够可靠地执行任务，但记忆力很差，还很喜欢上厕所。一旦任务进行到一半，上厕所回来以后就会忘记该做什么。因此，我们要给TA提供无限多的草稿纸，并且帮TA把执行任务的规则记下来；同时，TA也能用草稿纸把任务的执行状态记下来，甚至方便另一些“愚蠢”的计算员接替TA干活。

Turing Machine



TM 的例子 (TURINGMACHINE.IO)




```
# Adds 1 to a binary number.
input: '1011'
blank: ' '
start state: right
table:
  # scan to the rightmost digit
  right:
    [1,0]: R
    ' '  : {L: carry}
  # then carry the 1
  carry:
    1      : {write: 0, L}
    [0,' ']: {write: 1, L: done}
  done:


# Exercises:

# • Modify the machine to always halt on the
  leftmost digit
#   (regardless of the number's length).
#   Hint: add a state between carry and done.

# • Make a machine that adds 2 instead of 1.
#   Hint: 2 is '10' in binary, so the last digit
  is unaffected.
#   Alternative hint: chain together two copies of
  the machine from
#   the first exercise (renaming the states of the
  second copy).

# • Make a machine to subtract 1.
#   To simplify things, assume the input is always
  greater than 0.
```

Try it out

Run the machine to see it in action. At any time, you can **step** or **pause** to get a closer look, or **reset** to start over.

There are over a dozen different example machines to explore.

Most of the examples take input. Experiment with

Make your own

Make spinoffs from the examples or your own creations by using *Edit > Duplicate document*. You can also start from scratch with a new blank document.

All it takes to describe a Turing machine is a start state, blank symbol, and transition table.

Example

Tips

Editor keyboard shortcuts

Ctrl - S	**Load machine** Save changes and load the machine.
Ctrl - /	**Toggle comment**

λ-演算 (Lambda Calculus)

阿隆佐·邱奇 (Alonzo Church, 1930s) 提出的一种**极简计算模型**，仅基于函数抽象与应用

- 语法:
  - 变量：$x, y, z, \ldots$
  - 函数抽象：$\lambda x.\, M$
  - 函数应用：$(M\ N)$
- 一切计算都用“函数调用”来表达 → 没有数字、条件、循环，只是函数
- 例子：$(\lambda x.\, x + 1)\ 5 \to 5 + 1 \to 6$

λ-演算 (Lambda Calculus)

以 Church 数表示的自然数为例

- 定义:
  - $0 \equiv \lambda f.\, \lambda x.\, x$
  - $1 \equiv \lambda f.\, \lambda x.\, f(x)$
  - $2 \equiv \lambda f.\, \lambda x.\, f(f(x))$
- 加法:

$$\mathrm{ADD} \equiv \lambda m.\, \lambda n.\, \lambda f.\, \lambda x.\ m\ f\ (n\ f\ x)$$

例子：$1+2$

1. 代入定义:

$$\text{ADD } 1\ 2 \to (\lambda m.\,\lambda n.\,\lambda f.\,\lambda x.\ m\ f\ (n\ f\ x))\ 1\ 2$$

2. 代入$m=1, n=2$得到

$$(\lambda f.\,\lambda x.\ 1\ f\ (2\ f\ x))$$

3. 考虑$2 \equiv \lambda f.\,\lambda x.\ f\ (f\ x)$，得到

$$\lambda f.\,\lambda x.\ 1\ (f\ (f\ x))$$

4. 代入$1 \equiv \lambda f.\,\lambda x.\,f\ x$的定义，得到

$$\lambda f.\,\lambda x.\ f\ (f\ (f\ x))$$

这正是**Church 数 3**

乘法

- Church 数 n = "对函数 f 迭代 n 次"
- 加法 = 在同一个抽象值（ x ）上"把一个抽象函数迭代次数加起来"
- 乘法 = "迭代的次数再迭代"

所以乘法可以写成:

$$\text{MUL} \equiv \lambda m.\, \lambda n.\, \lambda f.\, \lambda x.\, m\ (n\ f)\ x$$

解释:

- $(n\ f) \equiv (\lambda g \lambda x.\ \ g \cdots g)\ [f = g] \equiv \lambda x.\ \ f \cdots f$，是一个函数：表示"对输入执行 f 共 n 次";
- $m\ (n\ f)\ x$ 意味着把这个"迭代 n 次的函数"在 x 上应用 m 次;

例子：$2 * 3$

$$
\begin{aligned}
& \mathrm{MUL}\ 2\ 3 \\
\to\ & \lambda f.\, \lambda x.\ \ 2\ (3\ f)\ x \\
\to\ & \lambda f.\, \lambda x.\ \ (3\ f)\ ((3\ f)\ x) \\
\to\ & \lambda f.\, \lambda x.\ \ (f\ f\ f\ (f\ f\ f\ x))
\end{aligned}
$$

递归？

在 λ 演算里，所有函数都是匿名的，没有“直接调用自己”的方式，但是递归需要函数能“看到自己”。

- 于是我们需要一个“魔法工具”：给定一个函数生成器 F（它期望接收自身作为参数）
- 让它能自己调用自己，从而得到一个递归函数。
- 这个魔法工具就是**不动点组合子**

Church 发明的一个经典版本叫做 Y 算子：

$$Y \equiv \lambda f.\,(\lambda x.\,f(xx))(\lambda x.\,f(xx))$$

它的性质是

$$Y\ F \to F\ (Y\ F)$$

递归?

$$
\begin{aligned}
& Y\ F \\
\to\ & (\lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x)))\ F \\
\to\ & (\lambda x.\ F\ (x\ x))\ (\lambda x.\ F\ (x\ x)) \\
\to\ & F\ [(\lambda x.\ F\ (x\ x))\ (\lambda x.\ F\ (x\ x))] \\
\equiv\ & F\ (Y\ F)
\end{aligned}
$$

注意:

- $Y\ F$并没有“自己调用自己”
- 而是通过展开，变成了$F\ (Y\ F)$，相当于F通过展开拿到了递归的定义。

阶乘的递归定义

递归生成函数:

$$
\begin{aligned}
F = \lambda f.\, \lambda n.&\\
&\text{IF } (\text{ISZERO } n)\\
&\quad 1\\
&\quad (\text{MUL } n\ (f\ (\text{PRED } n)))
\end{aligned}
$$

阶乘（递归函数）:

$$\text{FACT} = Y\ F$$

例子：FACT3

$$
\begin{aligned}
& \text{FACT } 3 \\
\to\ & (Y\ F)\ 3 \\
\to\ & F\ (Y\ F)\ 3 \\
\to\ & \text{IF (ISZERO 3) 1 (MUL 3 } ((Y\ F) \text{ (PRED 3)))} \\
\to\ & \text{IF (ISZERO 3) 1 (MUL 3 (FACT (PRED 3)))} \\
\to\ & \text{(MUL 3 (FACT 2))}
\end{aligned}
$$

λ-演算与图灵机

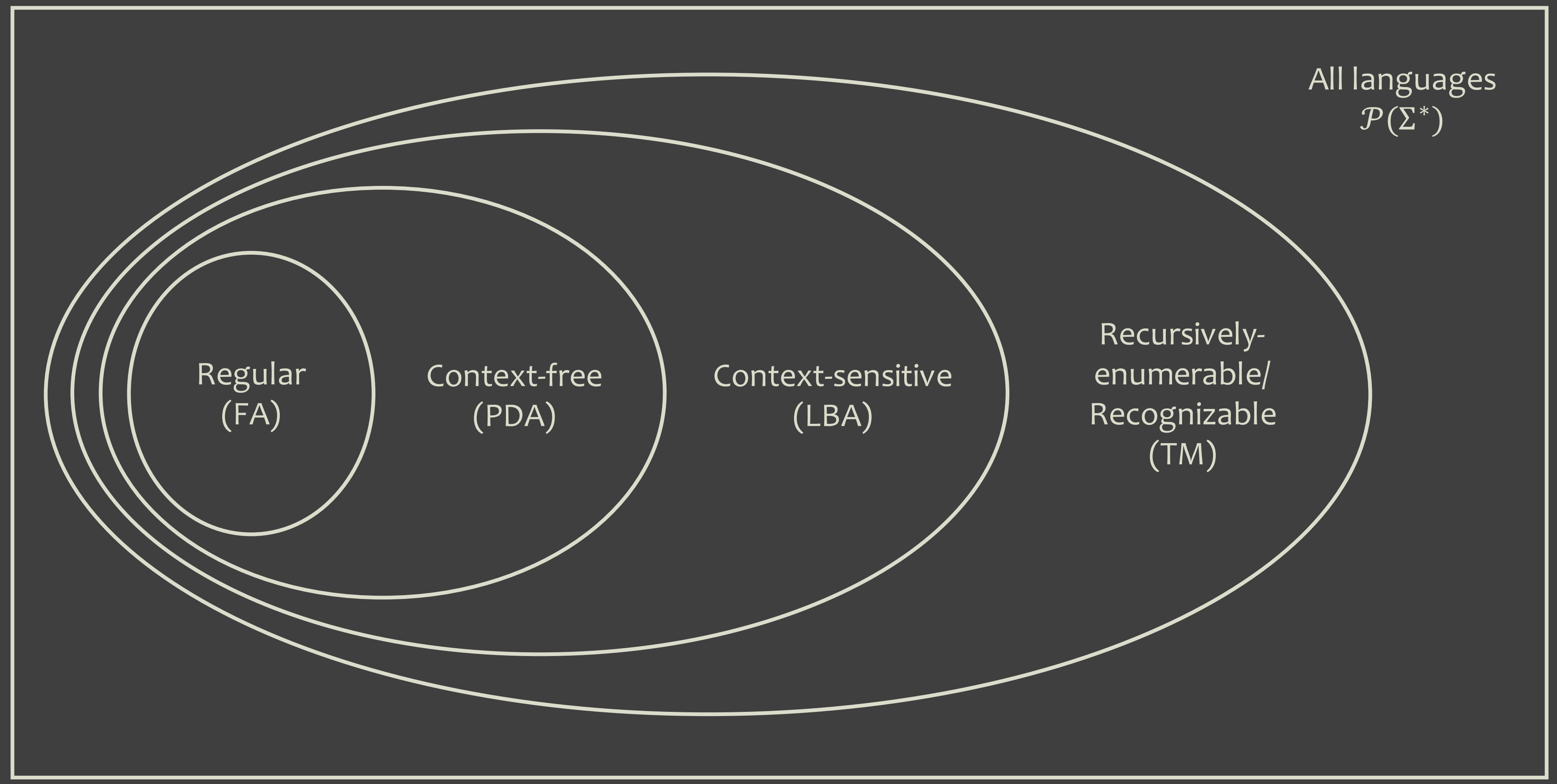
- λ-演算 与 图灵机**等价**，即它们都能表达所有**可计算函数**
- 直观比喻：
  - 图灵机：偏重“过程、状态、符号操作”
  - λ-演算：偏重“函数、代换、表达式简化”

Church–Turing Thesis

断言：所有“能行可计算”的函数都可以由**λ-演算**这些等价模型来计算

- 任何“机械化”计算过程（有效的、逐步的、有限的）都可以被图灵机模拟
- 不关心物理实现（算盘、晶体管、DNA 计算机…）。只要是“算法”，都不会超越图灵机
- 这是一个**哲学性断言**，而非数学定理
  - 没法严格证明，但至今无人找到反例

可计算性



等价模型: General Recursive Functions (Gödel), Lambda Calculus (Church), 算盘机 (胡海星；宋方敏), *etc.*

- 形式语言层次中的“最强”模型 = 图灵机
- 所有有效算法能计算的函数，都在图灵机的能力范围内
- 反之，若某个函数不能由图灵机计算 → 它不可计算

HALTING PROBLEM

问题：是否存在一个图灵机 H，它可以接受输入 $\langle M, w \rangle$ 并判定：

- 若图灵机 M 在输入 w 上会停机，则输出“YES”
- 若不会停机，则输出“NO”

- $H(M, w)$：一个“万能判停机 TM”
- 图灵 1936 证明：这样的 H 不存在

Halting Problem — 证明思想

1. 假设存在这样的判停机器 H
   - $H(M, w) \to$ YES / NO
2. 构造一个新的图灵机 $D(M)$，令:
   - 输入为个机器编码 M
   - 调用 $H(M, M)$
     - 若 H 说“YES”，那么 D 进入无限循环
     - 若 H 说“NO”，那么 D 停机退出

Halting Problem — 自指矛盾

- 考虑D的输入为自身：$D(D)$
- 如果 $H(D,D)=YES$，说明$D(D)$会停机
  - 但D的定义要求它进入无限循环 → 矛盾
- 如果$H(D,D)=NO$，说明$D(D)$不停机
  - 但D的定义要求它停机 → 矛盾
- 因此，H不可能存在，即停机问题是不可判定的

可判定与可计算

定义（能行可计算，*effectively computable*）：

一个函数 f（在某个定义域 D 上的全函数）是能行可判定的当且仅当存在一个算法，对任意 $o \in D$ 都可以在有穷步数内计算出 $f(o)$

- 对于一个定义在 D 上的性质，我们可定义一个特征函数（*characteristic function*）c_P，当 o 满足性质 P 时有 $c_P(o) = 1$，否则 $c_P(o) = 0$

可判定与可计算

定义（能行可判定，*effectively decidable*）：

一个性质 P（在某个论域 D 上定义）是能行可判定的当且仅当存在一个算法，在有穷步数内可判断任意 $o \in D$ 满足或不满足性质 P

- 命题逻辑中的重言式是能行可判定的
  - 枚举真值表
- wff中的主连词的位置是能行可判定的
  - 对括号个数进行计数
- 可计算/可判定性的“算法”运行在一个“抽象的计算机”中，我们不考虑它的物理运算速度或者内存大小，唯一重要的是有穷步能输出

计算与逻辑

```prolog
% Binary summation
start_binary_add :-
    string_chars("1011+11001", TapeChars),
    run(right, 0, TapeChars).

start_binary_add(S) :-
    string_chars(S, TapeChars),
    run(right, 0, TapeChars).

% UTM
run(done, _, Tape) :-
    format("*** Machine has halted. Final tape: ~w~n", [Tape]),
    !.
run(State, Pos, Tape) :-
    read_sym(Tape, Pos, Sym),
    transition(State, Sym, NewState, WriteSym, MoveDir),
    write_sym(Tape, Pos, WriteSym, TempTape, PosAfterWrite),
    move_pos(PosAfterWrite, MoveDir, NextPos),
    format("State=~w, Read=~w, Write=~w, Move=~w, NewState=~w, Tape=~w~n",
           [State, Sym, WriteSym, MoveDir, NewState, TempTape]),
    run(NewState, NextPos, TempTape)
```

符号

“语言和大脑共同进化”

自然语言和经典逻辑

实质蕴涵（material implication, $\to$）的问题：

- 事实条件句按经典逻辑全部为真，但直观上显然有真有假
  - 例如反事实条件句没有逆否规则：从 $A \to B$ 推不出 $\neg B \to \neg A$：
  - “如果希特勒当年没死，他也活不到现在”。这句话直观为真，但其逆否“如果希特勒活到现在，那他当年就死了”显然为假
- 非确定性推理也不符合经典的推理规则
  - 即使你有很高的概率接受 $A \vee B$，你也不一定有很高的概率接受 $\neg A \to B$
  - 你非常相信现在是12点，因此你也非常相信现在是12点或6点，但你不会相信“如果现在不是12点那现在就是6点”
- 大多数自然语言中的条件句都不能通过实质蕴含来形式化，否则
  - 当我们否定一个条件句，我们就能得到该条件句的前件：$\neg(A \to B)$ 可以推出 A
  - 但：“并非（如果我是一个好的逻辑学家，我就是一个好的文学家）”是真的，因此“我是一个好的逻辑学家”

符号的边界

“西边的欧钢有老板
生儿维特根斯坦
他言说马户驴又鸟鸡
到底那马户是驴还是驴是又鸟鸡
那驴是鸡那个鸡是驴那鸡是驴那个驴是鸡
那马户又鸟
是我们人类根本的问题”
——刀郎《罗刹海市》

符号的边界

1. **语言游戏（Language Game）**：语言的多样性和情境性
   - 语言不是抽象规则的系统，而是一种在特定情境下的活动。每种语言的使用，都像是在玩一个游戏，而不同游戏有不同的规则和目的
     - 问题和背景不同，就会产生不同的“语言游戏”：法庭上 - 讲证据、辩论；数学课堂上 - 定义、证明；……
   - 语言的意义来自于使用（use），而不是严格的定义
2. **家族相似性（Family Resemblance）**：概念之间关系的方式
   - 概念之间并不是通过一组严格定义的共同特征来划分，而是像家族成员一样，有一些重叠的、交织的相似性
   - 没有必要为所有事物找到一个严格的“本质定义”
   - 就像家族成员：有的有相似的鼻子，有的有相似的眼睛，但并没有一个所有成员都有的特征

所以，我们为什么还要讨论《符号学习》？