下一代编程语言是什么样的？

王迪

2024.06.22

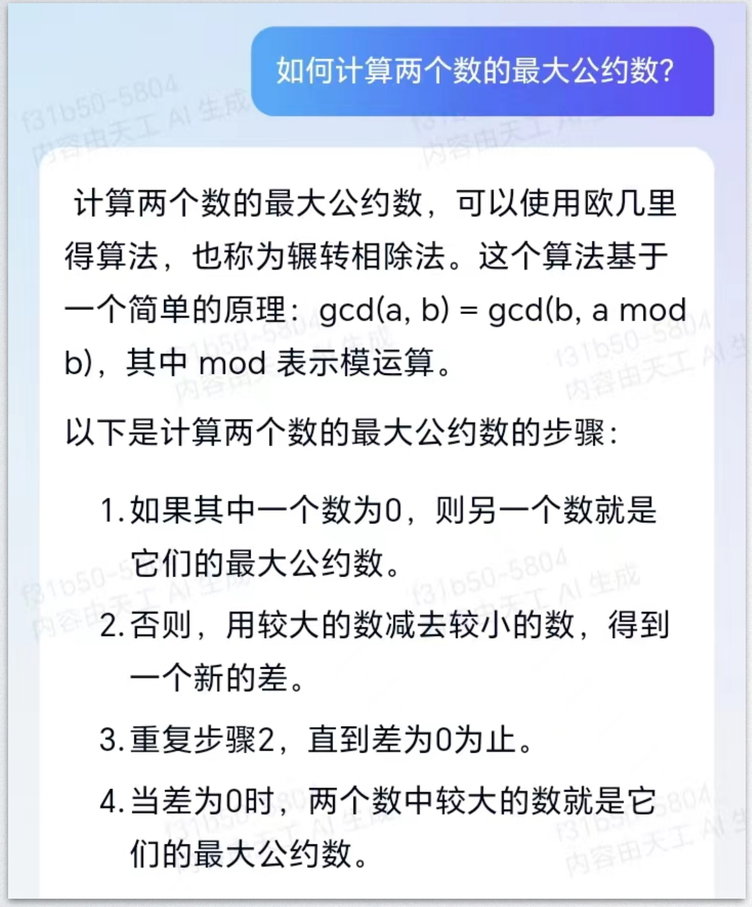
正好在昨天——


下一代编程语言
仓颉开启预览
原生智能化
天生全场景
高性能
强安全
developer.huawei.com/consumer/cn/cangjie/
HDC.2024
华为开发者大会2024

编程语言领域研究些什么？

- 研究怎么造编程语言？
  - 既是，也不是
- 为啥有这么多编程语言？
- 编程语言的本质是什么？

编程语言与计算



编程语言：C

```c
int gcd(int a, int b) {
  while (b != 0) {
    int t = b;
    b = a % b;
    a = t;
  }
  return a;
}
```

意识中的计算 → 程序 → 硬件上的计算

编程语言：驾驭计算的基本工具

计算任务

「计算两个数的最大公约数」

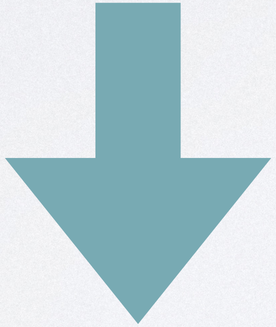
「解一个数独」

「判断一张图片是不是人脸」

「给出一个棋盘状态下的最优决策」

「写一首李白风格的关于火锅的现代诗」

「生成一张黑色柴犬在雪地的图片」



用编程语言为计算任务写程序

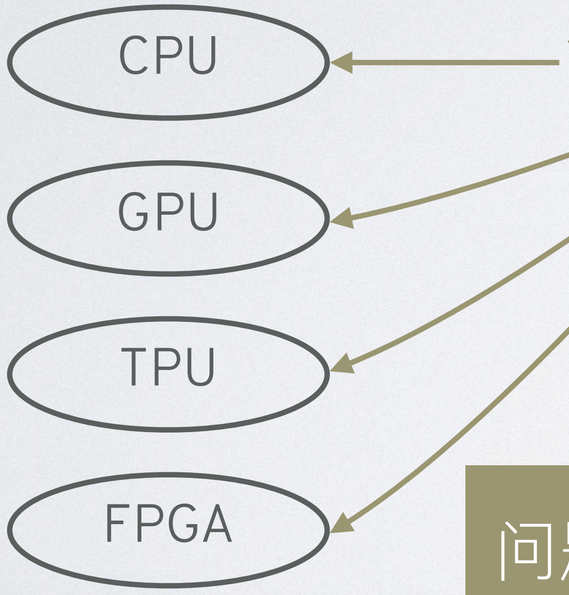
编译器自动把程序转换为计算实现

计算实现

CPU　GPU　TPU　FPGA　……

为啥有这么多编程语言？

机器语言 → 汇编语言 → 高级编程语言 → 下一代编程语言？



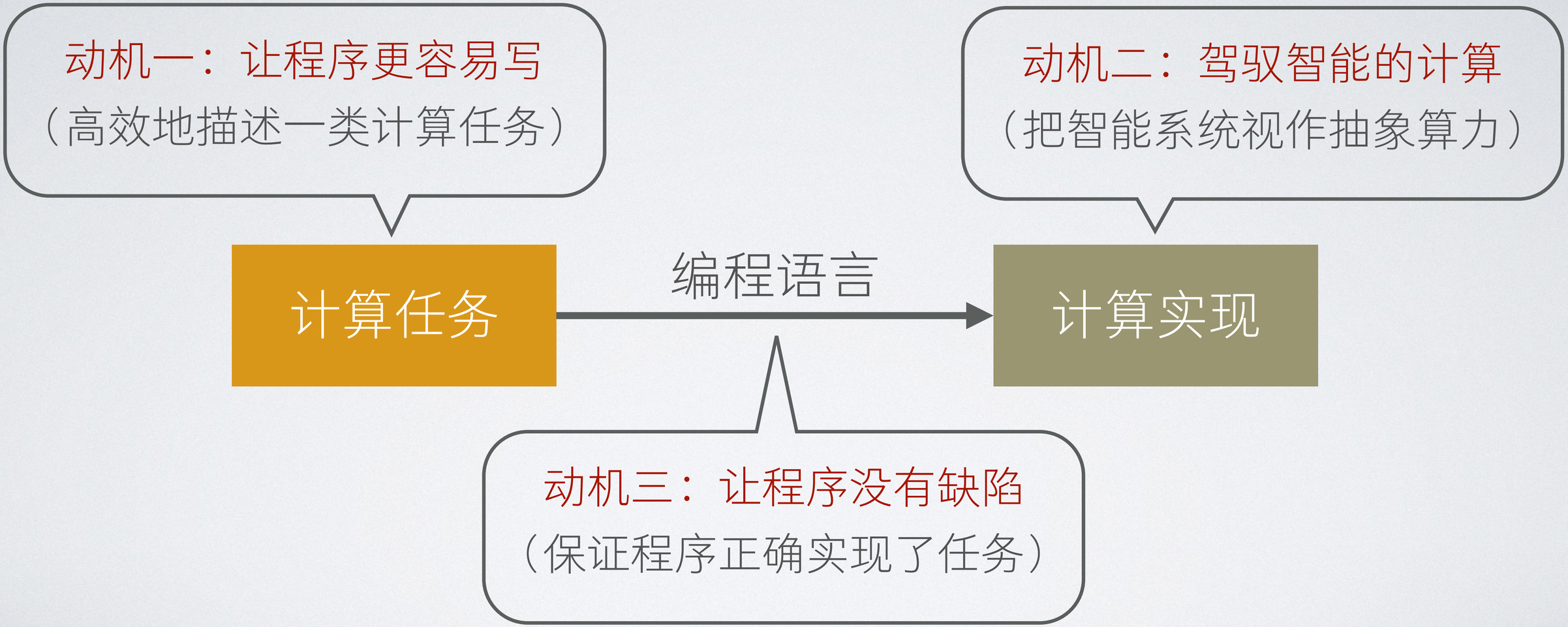
人能够阅读和编写的机器语言

问题：抽象程度低

命令式语言
函数式语言
逻辑式语言
面向对象语言
……

动机：给某一类计算任务设计抽象

设计下一代编程语言的动机



动机一：让程序更容易写

希望我的外婆也能编程

函数式编程

- **动机：**能否只描述目标，让计算机自动完成相应的计算？
- **如何描述目标？**
  - 程序是从输入到输出的函数
- **如何描述函数？**
  - 数学家：通过等式限定一个多对一关系
  - $gcd(a,0) = a$
  - $gcd(a,b) \mid b \neq 0 = gcd(b, a \bmod b)$
- **函数式语言：**采用如上方法描述计算目标

代表语言

Haskell

OCaml

Racket

函数式编程：快速排序

quicksort 是一个从 Int 列表到 Int 列表的函数

```
quicksort :: [Int] -> [Int]

quicksort [] = []

quicksort (p:xs) = (quicksort less) ++ [p] ++ (quicksort greater)
  where
    less    = [ x | x <- xs, x < p ]
    greater = [ x | x <- xs, x >= p ]
```

输入为空列表时，输出也是空列表

输入为非空列表，第一个元素是 p，其余元素是 xs

++ 表示连接两个列表

筛选 xs 中满足条件的元素

逻辑式编程

- **动机**：能否只描述目标，让计算机自动完成相应的计算？
- **如何描述目标？**
  - 给出问题的描述，让计算机自动解答
- **如何描述问题？**
  - 逻辑学家：应该用逻辑来描述
  - $max(a, b, c) := c \geq a \wedge c \geq b \wedge (c = a \vee c = b)$
  - $sudoku(puz, sol) := valid(sol) \wedge consistent(puz, sol)$
- **逻辑式语言**：使用逻辑来描述问题，然后让计算机自动求解

代表语言

Prolog

Erlang

Curry

逻辑式编程：数独求解

```
sudoku :: [Int] -> Bool
sudoku m =
  let s11, s12, s13, ..., s19 free in
  let s21, s22, s23, ..., s29 free in
  ...
  m =:= [s11, s12, s13, ..., s19,
         s21, s22, s23, ..., s29,
         ...
         s91, s92, s93, ..., s99] &&
  domain m 1 9 &&
  allDifferent [s11, s12, s13, ..., s19] &&
  allDifferent [s11, s21, s31, ..., s91] &&
  allDifferent [s11, s12, s13, ..., s31, s32, s33] &&
  ...
```

「逻辑式」：描述问题本身，而非描述解法

数独的第一行数字两两不同

数独的第一列数字两两不同

数独的第一个 3x3 方块中数字两两不同

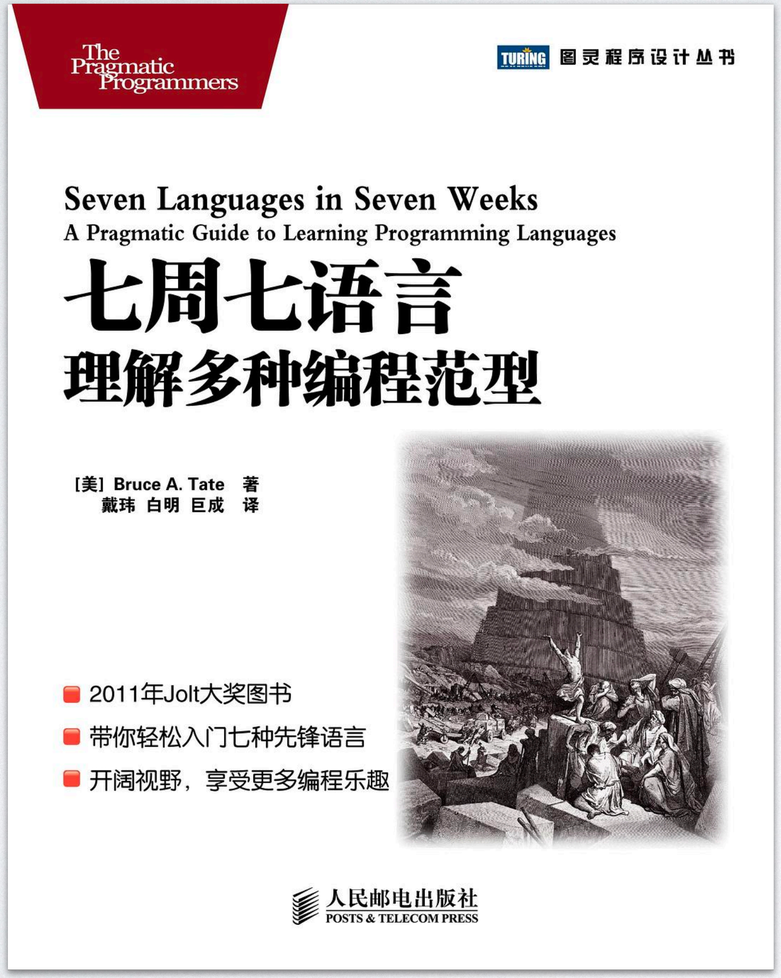
逻辑式编程：数独求解

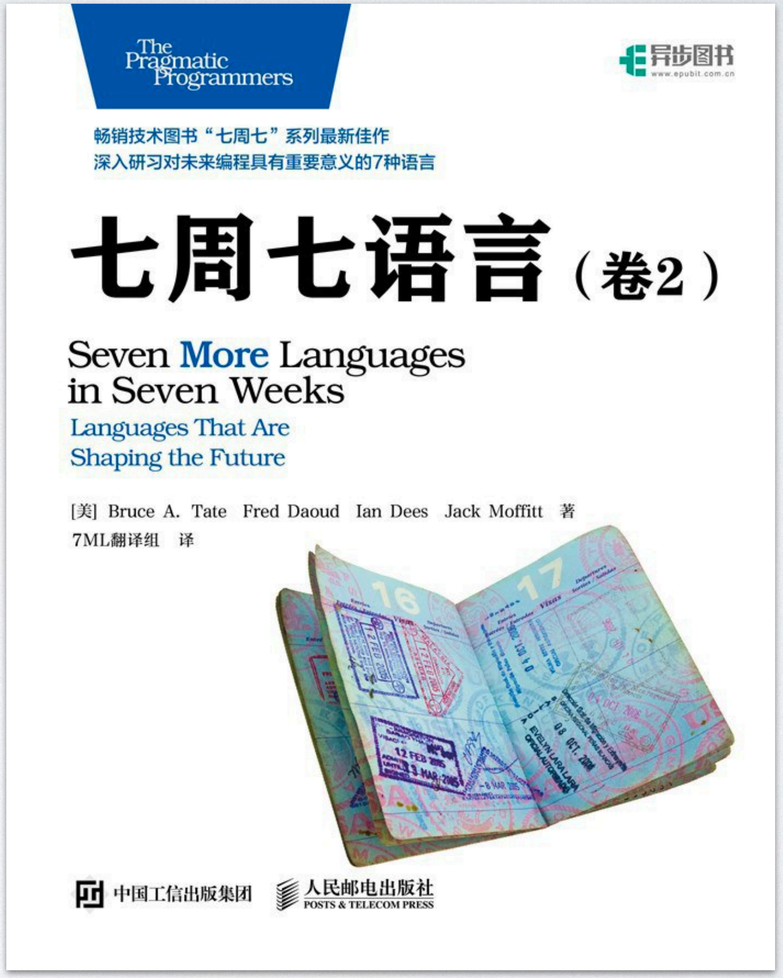
```
sol :: [Int]
sol | sudoku puz = puz
  where
    puz = [_, _, 5, 8, _, _, 1, _, _,
           _, _, 7, _, 1, 6, _, _, 3,
           _, 6, _, 4, _, _, _, _, _,
           5, _, _, _, _, _, _, _, 4,
           _, 2, _, _, 7, 1, 5, _, _,
           _, _, _, _, 8, _, _, _, _,
           _, 7, _, _, 5, 2, 4, _, _,
           _, _, 3, _, _, _, _, 9, _,
           _, _, _, 6, _, _, _, _, _]
```

自动求解

```
> sol
[3,9,5,8,2,7,1,4,6,
 2,4,7,9,1,6,8,5,3,
 8,6,1,4,3,5,9,2,7,
 5,1,8,2,6,9,3,7,4,
 6,2,4,3,7,1,5,8,9,
 7,3,9,5,8,4,2,6,1,
 9,7,6,1,5,2,4,3,8,
 1,5,3,7,4,8,6,9,2,
 4,8,2,6,9,3,7,1,5]
```

两本闲书


The Pragmatic Programmers
TURING 图灵程序设计丛书
Seven Languages in Seven Weeks
A Pragmatic Guide to Learning Programming Languages
七周七语言
理解多种编程范型
[美] Bruce A. Tate 著
戴珏 白明 巨成 译
2011年Jolt大奖图书
带你轻松入门七种先锋语言
开阔视野，享受更多编程乐趣
人民邮电出版社
POSTS & TELECOM PRESS


The Pragmatic Programmers
异步图书
www.epubit.com.cn
畅销技术图书“七周七”系列最新佳作
深入研习对未来编程具有重要意义的7种语言
七周七语言（卷2）
Seven More Languages in Seven Weeks
Languages That Are Shaping the Future
[美] Bruce A. Tate Fred Daoud Ian Dees Jack Moffitt 著
7ML翻译组 译
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

为啥有这么多编程语言？

动机：让程序更容易写

（高效地描述一类计算任务）

计算任务

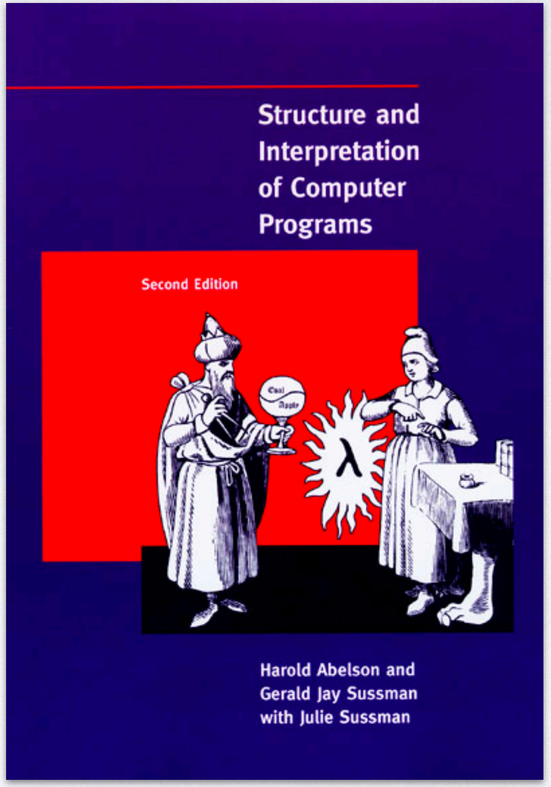
正在进行的工作：

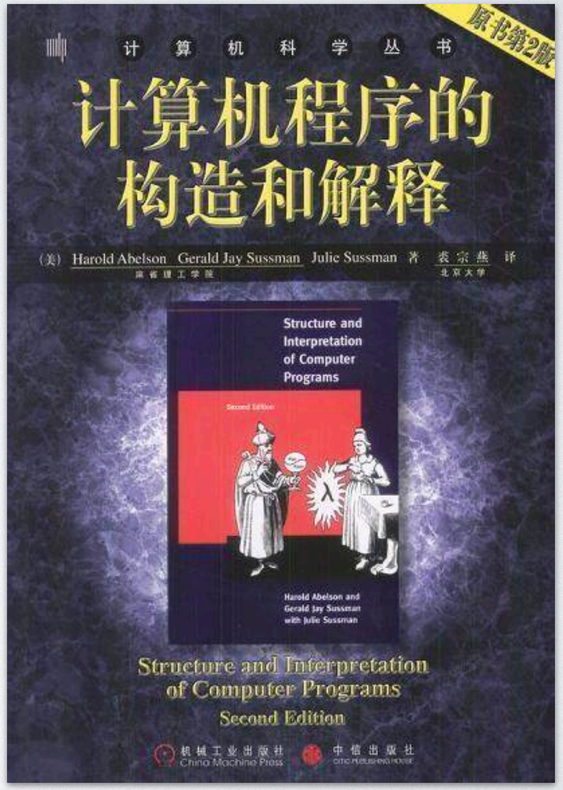
DSL 开发和互操作平台

- 领域特定语言
  - Domain-Specific Language，DSL
- 针对一类计算任务的专用语言，提供**领域特定的抽象**，使得编程更简单
  - Excel 的公式语言
  - 数据库查询语言（SQL）
  - 网页标记语言（HTML）
  - ……

设计编程语言，描述领域问题

「事实上，我们几乎可以把任何程序看做是某个语言的求值器……处理大规模计算机系统的技术，与构造新的编程语言的技术有紧密的联系，而计算机科学本身不过（也不更少）就是关于如何构造适当的描述语言的学科。」





动机二：驾驭智能的计算

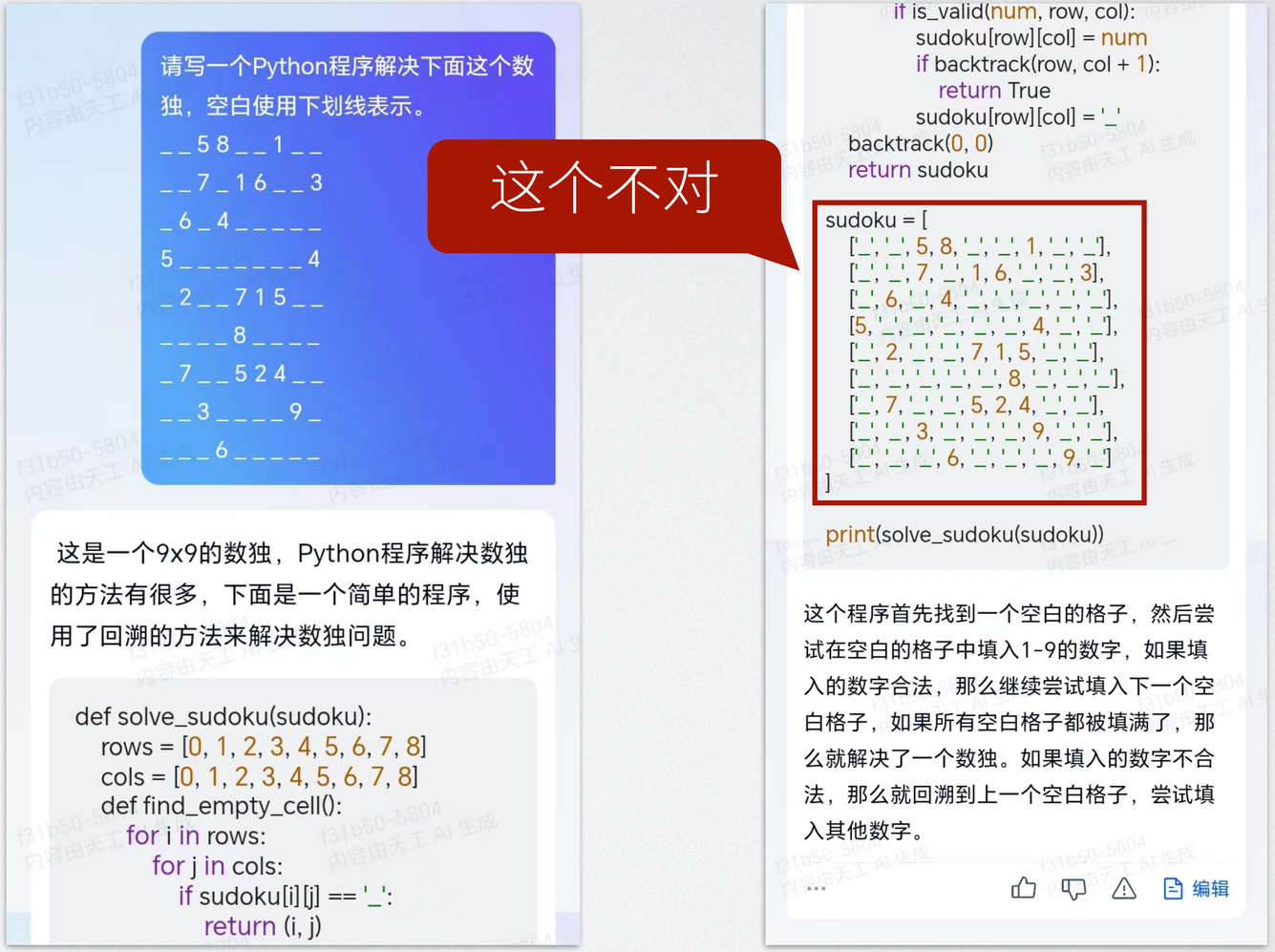
希望编程更加自然、抽象

回顾数独求解问题

- **逻辑式编程**：给描述，算答案
- **大语言模型**：给描述，算答案
- 如何改进一下？

回顾数独求解问题

- **程序合成：** 给描述，生成计算答案的程序
- **大语言模型：** 给描述，生成计算答案的程序
- 但是程序本身是对的！



~~热点问题：怎么让 AI 模型更好地生成程序？~~

为什么用编程语言作为媒介能更好地解决一些问题？

AI 模型本身作为一种计算，它有什么优缺点？

如何更好地驾驭这种智能计算？

生成式计算

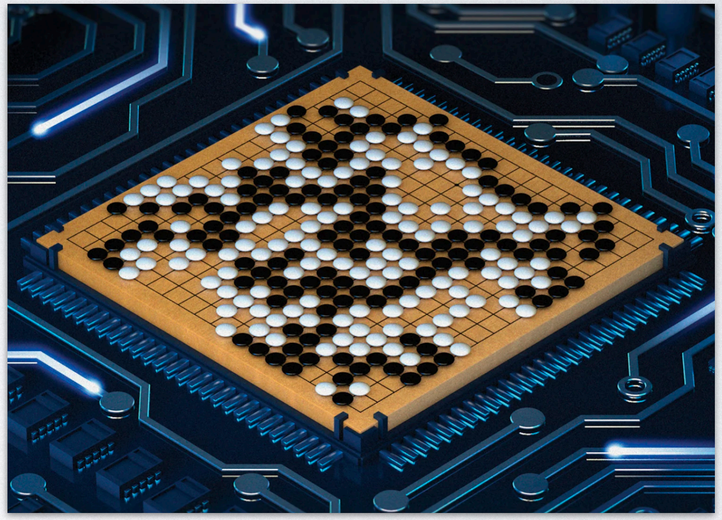
文字

火锅之歌

红炉燃烈火，汤底煮食材。
翻滚三江水，汇聚四海味。
辣中带鲜香，麻中透醇美。
邀友共一宴，举杯话桑梓。

图片

策略



数据驱动的、带有不确定性的、难以用逻辑刻画的、生成式的

计算

概率：生成式计算的基石

输入：提示或上下文 → 大语言模型 → **输出：**下一个词的概率

数据驱动
从数据中学习概率分布

不确定性
通过概率来刻画

生成式
从概率分布中采样

领域特定语言

SCENIC：驾驶场景描述语言

概率分布

```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) *
           Range(10, 20) deg
Car left of spot by 0.5,
facing badAngle relative to roadDirection
```

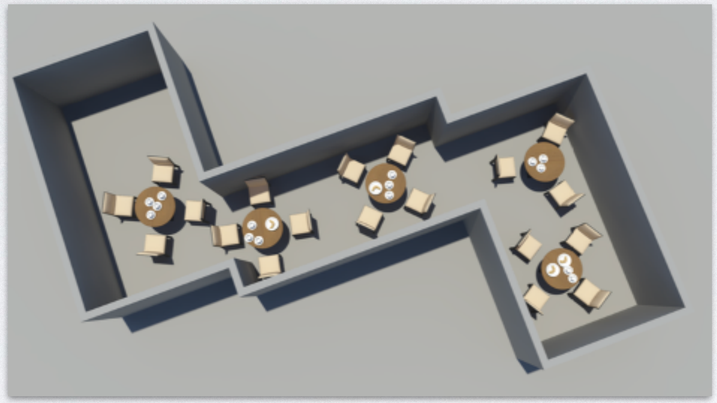
描述场景需要满足的性质

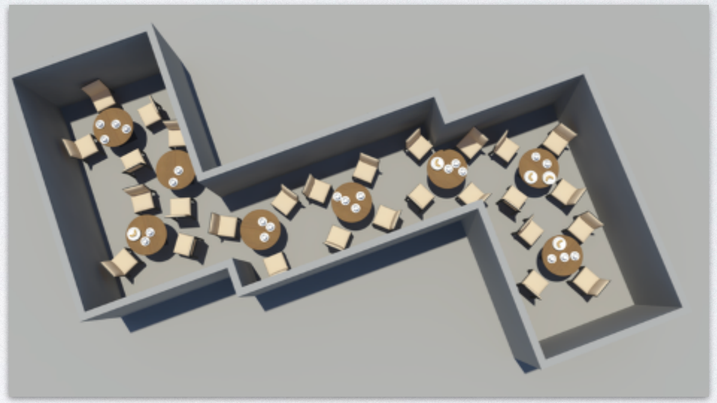
概率编程：以程序的方式组合不同的概率分布，并对生成的输出进行约束

D. J. Fremont el al. 2019. SCENIC: A Language for Scenario Specification and Scene Generation. In *PLDI'19*.

概率编程：协调逻辑约束与生成式计算

- 生成建筑设计图，要求满足安全规定





- 生成法律文本，要求符合相关规范
- 生成考试题目，要求不能超纲

正在进行的工作：

生成式、逻辑式的概率编程语言

动机三：让程序没有缺陷

希望软件都没有 bug

如何知道程序是对的？

```c
int f(int x, int y) {
  int r = 1;
  while (y > 1) {
    if (y % 2 == 1) {
      r = x * r;
    }
    x = x * x;
    y = y / 2;
  }
  return r * x;
}
```

- 左边的 C 程序做了什么事情？
- 这段程序有 bug 吗？
- 如何通过编程语言来提升程序的可信度？

编程语言能提供什么？

- 1969 年
- 无类型系统
- 性能差，不安全

- 1972 年
- 简单类型系统
- 性能好，基本安全

- 2015 年
- 复杂类型系统
- 性能不错，更加安全

「没有免费的午餐」：程序员需要对程序进行更多的设计和分析

前条件

```
int f(int x, int y) {
  int r = 1;
  while (y > 1) {
    if (y % 2 == 1) {
      r = x * r;
    }
    x = x * x;
    y = y / 2;
  }
  return r * x;
}
```

```
int f(int x, int y)
//@require y >= 0;
{
  int r = 1;
  while (y > 1) {
    if (y % 2 == 1) {
      r = x * r;
    }
    x = x * x;
    y = y / 2;
  }
  return r * x;
}
```

- 约束输入
- 动态检查
- **安全性**：调用者确保前条件满足

后条件

```
int f(int x, int y)
//@require y >= 0;
{
  int r = 1;
  while (y > 1) {
    if (y % 2 == 1) {
      r = x * r;
    }
    x = x * x;
    y = y / 2;
  }
  return r * x;
}
```

```
int f(int x, int y)
//@require y >= 0;
//@ensure \result
        == POW(x, y);
{
  int r=1, b=x, e=y;
  while (e > 1) {
    if (e % 2 == 1) {
      r = b * r;
    }
    b = b * b;
    e = e / 2;
  }
  return r * b;
}
```

- 约束输出

POW 是什么？

- **正确性**：被调用者确保后条件满足

规约函数

```
int f(int x, int y)
//@require y >= 0;
//@ensure \result
        == POW(x, y);
{
  int r=1, b=x, e=y;
  while (e > 1) {
    if (e % 2 == 1) {
      r = b * r;
    }
    b = b * b;
    e = e / 2;
  }
  return r * b;
}
```

```
int POW(int x, int y)
//@require y >= 0;
{
  if (y == 0) return 1;
  return POW(x, y-1)*x;
}
```

- 在前/后条件中使用
- 描述程序应该做什么
- 没有副作用

- POW 没有 f 高效
- 规约函数通常是函数式的/逻辑式的，通常更容易写对

循环不变式

```
int f(int x, int y)
//@require y >= 0;
//@ensure \result == POW(x, y);
{
  int r = 1, b = x, e = y;
  while (e > 1)
  //@loop_invariant POW(b, e) * r == POW(x,y);
  {
    if (e % 2 == 1) {
      r = b * r;
    }
    b = b * b;
    e = e / 2;
  }
  return r * b;
}
```

e>=0?

- 循环不变式也是一种基于程序点信息的分析
- 在循环过程中（包括退出时）总是成立

过程式分析：

「e 一开始等于本来就非负的 y，然后每次循环都除 2，所以总是非负的」

转换为点到点分析：

把分析表达为程序点上的信息（比如循环不变式）

程序断言

```
int f(int x, int y)
//@require y >= 0;
//@ensure \result == POW(x, y);
{
  int r = 1, b = x, e = y;
  while (e > 1)
  //@loop_invariant e >= 0;
  //@loop_invariant POW(b,e) * r == POW(x,y);
  {
    if (e % 2 == 1) {
      r = b * r;
    }
    b = b * b;
    e = e / 2;
  }
  //@assert e == 0;
  return r * b
}
```

断言适合用来给出分析过程的中间结果，或者描述预期程序点应该满足的性质

- **点到点分析**
- 通过程序点上的信息组合出想要的性质
- 前后条件、循环不变式、程序断言
- 分析具有局部性

此时可知 r==POW(x,y)，所以返回处有 bug，应该直接返回 r

编程语言能提供什么？

```
int f(int x, int y)
//@require y >= 0;
//@ensure \result == POW(x, y);
{
  int r = 1, b = x, e = y;
  while (e > 1)
  //@loop_invariant e >= 0;
  //@loop_invariant POW(b,e) * r == POW(x,y);
  {
    if (e % 2 == 1) {
      r = b * r;
    }
    b = b * b;
    e = e / 2;
  }
  //@assert e == 0;
  return r;
}
```

- **「没有免费的午餐」**：程序员需要自己进行点到点分析并标注前后条件、循环不变式以及程序断言
- **自动化**：尽可能减少程序员需要写的东西
- **核心**：让编译器来检查程序员提供的标注都是对的

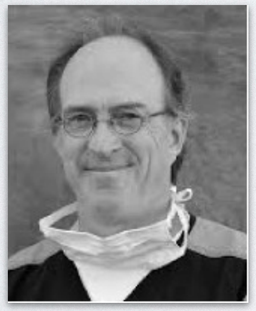
性质的证明没有银弹

哥德尔

「总是有些定理是不能证明或证否的。」
—— 哥德尔不完备定理，1931 年

「对于停机这个性质，无论什么算法，总是有程序是没法证明其停机与否的。」—— 停机问题，1936 年

图灵



莱斯

「世界上绝大多数程序性质都跟停机问题一样没法自动证明或证否。」—— 莱斯定理，1953 年

免责声明：以上说法为了方便理解均进行了不严谨的演绎，它们也不是这些科学家们的原话。

证明要人写，但是自动检查证明

- 交互式定理证明语言
  - 支持定义程序、命题和证明
  - 自动检查证明是否确实证明了命题
    - 并随时提示程序员还没有完成证明的部分

代表语言

Coq

Lean

Isabelle

Agda

- **正在进行的工作：**支持交互式定理证明的类 C 型编程语言

下一代编程语言是什么样的？