



# 程序设计语言研究概览

熊英飞

北京大学计算机学院软件研究所



给你定个题目：

编程语言研究探秘

然后我明天给你一个裁剪的建议！



编程语言研究一直给人理论严谨、高不可攀的印象。实际上，许多计算机系统领域的重要研究工作的基石都来自编程语言研究领域，本次报告将分享编程语言领域的研究者到底在做什么，以及对计算机系统方面的研究有怎样的启发。

我给你定了一个简介，哈哈。看有没有要修改的地方。我希望能加一些内容，就是这些编程语言里的各种机制，到底是如何影响计算机系统的。



不会，哈哈哈

我主要怕讲太浅了被人打



所以如果讲这个我要把锅甩到你身上



就说主席非要我讲的





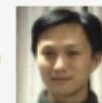
# 编程语言和系统软件不分家

- 一道常用面试题：
  - 请描述一下当编译运行下面这条C语句的时候，编译器运行时操作系统硬件等计算机系统各个层次都做了什么事。假设系统是Linux，编译器GCC。
    - `int* p = (int*)malloc(sizeof(int));`
  - 理解该语句需要充分理解编程语言和计算机系统
    - 关联C的标准库和Linux系统调用
    - 了解高级代码如何和汇编代码对应
    - 了解编译器的内存管理和操作系统的内存管理的交互
    - 了解编译器和链接器如何处理系统调用

# 编程语言社区和系统软件社区现实世界交流不够多



我大概懂你的思路了，就是说编程语言社区搞了很多天马行空的东西，这些东西有可能对现实世界是有用的，但是编程语言这些人都是比较奇怪，对于现实世界不太关心，自己在小圈子里自己玩。



对！其实PL的人有时候只管theory，反正架子搭好了，你们玩去吧。但一个好东西要大家用，其实有时候还要考虑是不是natural、用起来好不好、推广等等。如果能有一些思考就太好了。

# 修仙的编程语言研究人员



Haskell: Avoid success at all cost

Scheme:  
R6RS vs R7RS

Idris:  
不希望用于实际开发

# Rust



现实世界	PL研究人员
2010年，Rust第一次公开	<div data-bbox="1025 558 1449 868"><p>啥玩意，上个世纪的ownership type又拿出来炒</p></div> <div data-bbox="1481 544 1798 865"></div>
2023：微软用Rust重写Windows内核 2024：白宫推荐使用Rust	<div data-bbox="1265 946 1559 1255"></div>

# seL4



现实世界	PL研究人员
<p>2009年，SOSP上发表seL4论文</p>	<p>有啥大惊小怪的， CompCert好几年前就发布了</p> 
<p>2019年，seL4在国防安全、自动驾驶等领域已广泛使用，获SIGOP HoF奖 2020年，美国国防部改进采购政策，以便支持形式化验证</p>	

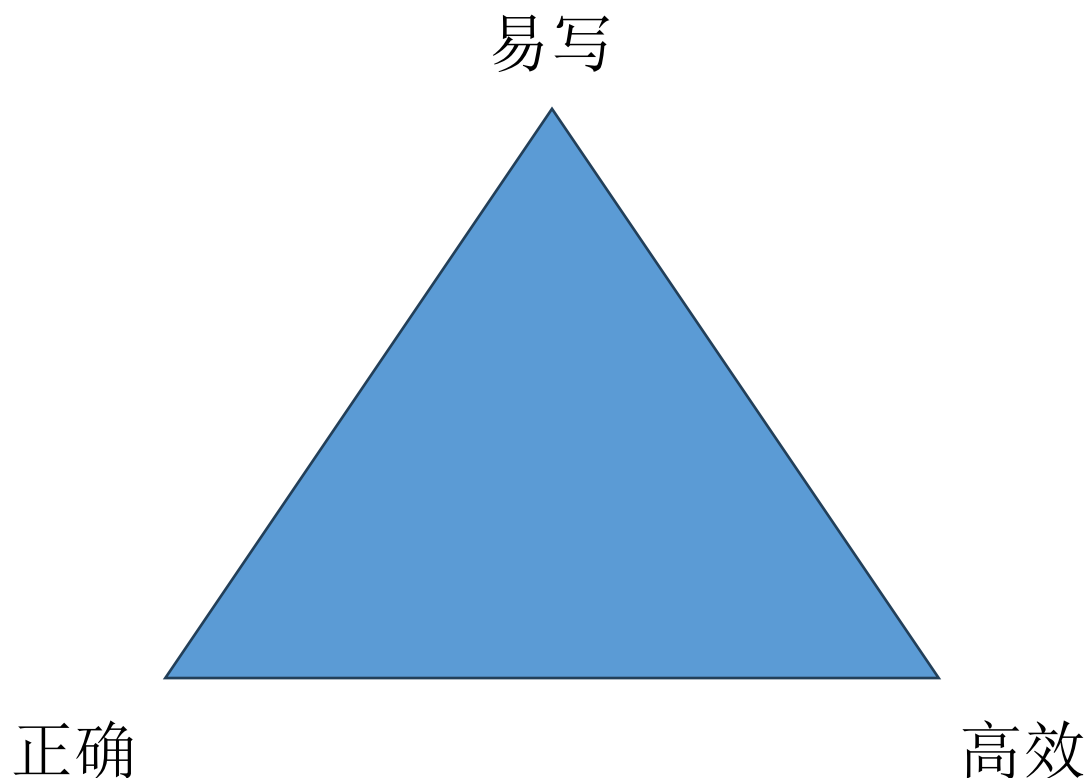




有人拿了修仙界少量技术改变了世界  
务实的ChinaSys社区或许能发掘更多技术?

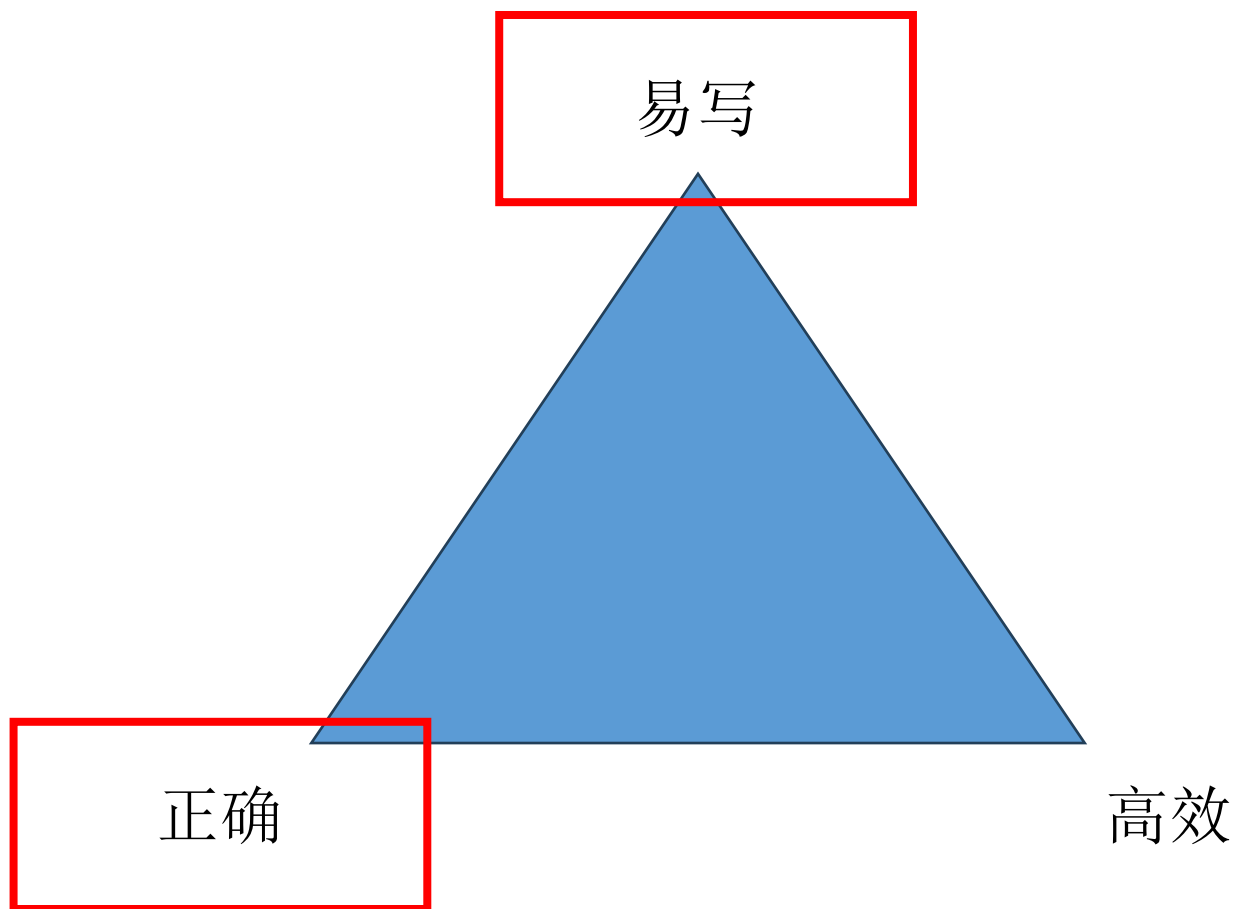


# 程序设计语言的期望属性



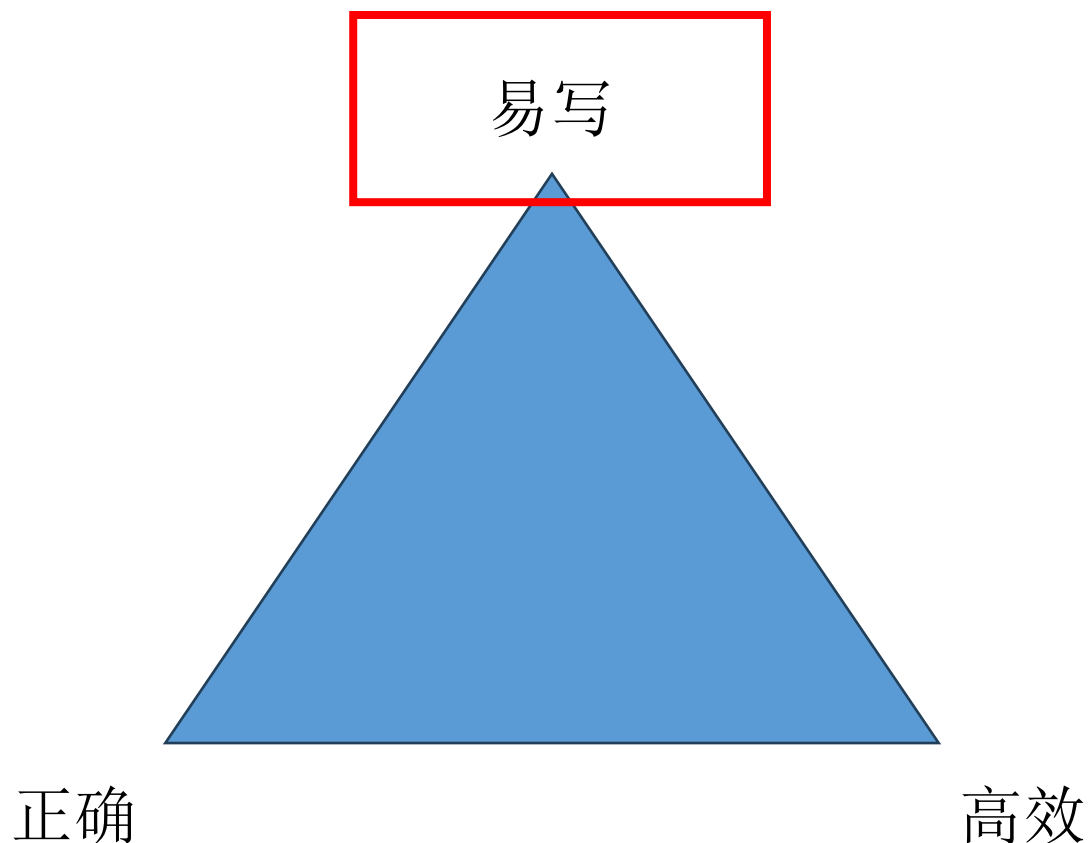


# 程序设计语言的期望属性





# 程序设计语言的期望属性





# 如何让程序容易写？

- 常见命令式语言描述计算机执行的操作
- 能否只描述目标，让计算机自动写程序？
- 如何描述目标？
  - 用自然语言描述？
    - Edsger W.Dijkstra. On the foolishness of “natural language programming”. 1978
  - 数学家：
    - 程序是从输入到输出的函数
    - 用数学公式表达函数
  - 逻辑学家：
    - 程序的目标由其外部属性决定
    - 用逻辑推导式和谓词表达外部属性

$f(a, b) = a + b$   
 $f(a+1, b+1) = a + b - 2$

`sort(In, Out) :-`  
    `perm(In, Out),`  
    `sorted(Out).`



# 如何让程序容易写？

- 如何从目标得到程序？
  - 由于有大量问题都不可判定，无法保证能从目标得到程序
- 解决方案1：限定目标在一个可计算的子集
  - 函数式语言
  - 逻辑式语言
- 解决方案2：尝试生成程序，没成功就放弃
  - 程序合成



# 函数式语言举例

- 一行写插入排序

```
insert = foldr insert []
```

- 5行写快速排序

```
quicksort [] = []
```

```
quicksort (p:xs) = (quicksort lesser) ++ [p] ++ (quicksort greater)
```

```
  where
```

```
    lesser = filter (< p) xs
```

```
    greater = filter (>= p) xs
```

- 代表语言

- Lisp, Scheme, Racket
- ML, Ocaml
- Haskell



# 现代函数式语言特性

- 无副作用：和数学函数对应，避免副作用带来的Bug
- 高阶函数：代码更容易复用，也更容易并行化
- 代数数据类型：更简洁（相对OO的继承）和安全（相对C的Struct和Union）地表示各种数据结构
- 结构化类型系统：比传统语言的Nominal Type System更强大
- 函数式数据结构：高效Immutable数据结构实现





# 函数式数据结构

```
print(a[2]); // 5
```

```
a[2] = 1;
```

```
print(a[2]); // 1
```

```
print(a[2]); //5
```

```
b = update(a, 2, 1);
```

```
print(a[2]); //5
```

```
print(b[2]); //1
```



# 函数式数据结构：复杂度

Scala标准库	更新	随机访问
Vector	Effective Constant	Effective Constant

Scala标准库	添加	删除	查找	找最小
HashSet	Effective Constant	Effective Constant	Effective Constant	Linear
TreeSet	Log	Log	Log	Log



# 逻辑式编程语言举例

- 一行写排序

`sort(In, Out) :- perm(In, Out), sorted(Out).`

- `perm`和`sorted`也可以类似定义

`sorted([]).`

`sorted([_]).`

`sorted([X,Y|Rest]) :- X =< Y, sorted([Y|Rest]).`

`perm([],[]).`

`perm(List, [H|Perm]) :- delete(H, List, Rest), perm(Rest, Perm).`

- 代表语言： Prolog, Erlang, Curry



# 程序合成

- 输入程序的语法空间、要满足的逻辑规约
- 输出符合语法、满足规约的程序
- 输出允许失败
- 例：
  - 语法：常见程序设计语言语法
  - 规约： $\text{Perm}(\text{sort}(l), l) \wedge \text{Sorted}(\text{sort}(l))$
  - 期望答案：高效排序算法
- 广泛用于编译器优化、算法设计、硬件设计等领域
  - superopt, souper：编译时自动搜索更高效优化
  - Sufu（北大）：输入穷举程序，输出高效算法
  - Enlightenment-1（计算所）：程序合成设计的CPU



# 领域特定语言

- 特定领域的任务未必适合用通用方法描述
  - 数据库查询语言：SQL
  - 可微分语言：TensorFlow/PyTorch
  - 概率编程语言：Stan, PyMC
  - 大型语言模型提示工程编程语言：LMQL
  - 智能合约编程语言：Solidity
  - GPU描述语言：Triton
  - 网络设备编程语言：P4, NetKat



# 元编程

- 领域特定语言这么多，开发领域特定语言也成了巨大负担
- 能否让领域特定语言的编写更加容易？
- 元编程语言： Racket, Rosette



# 元编程例子

- SAT求解器

```
#lang rosette

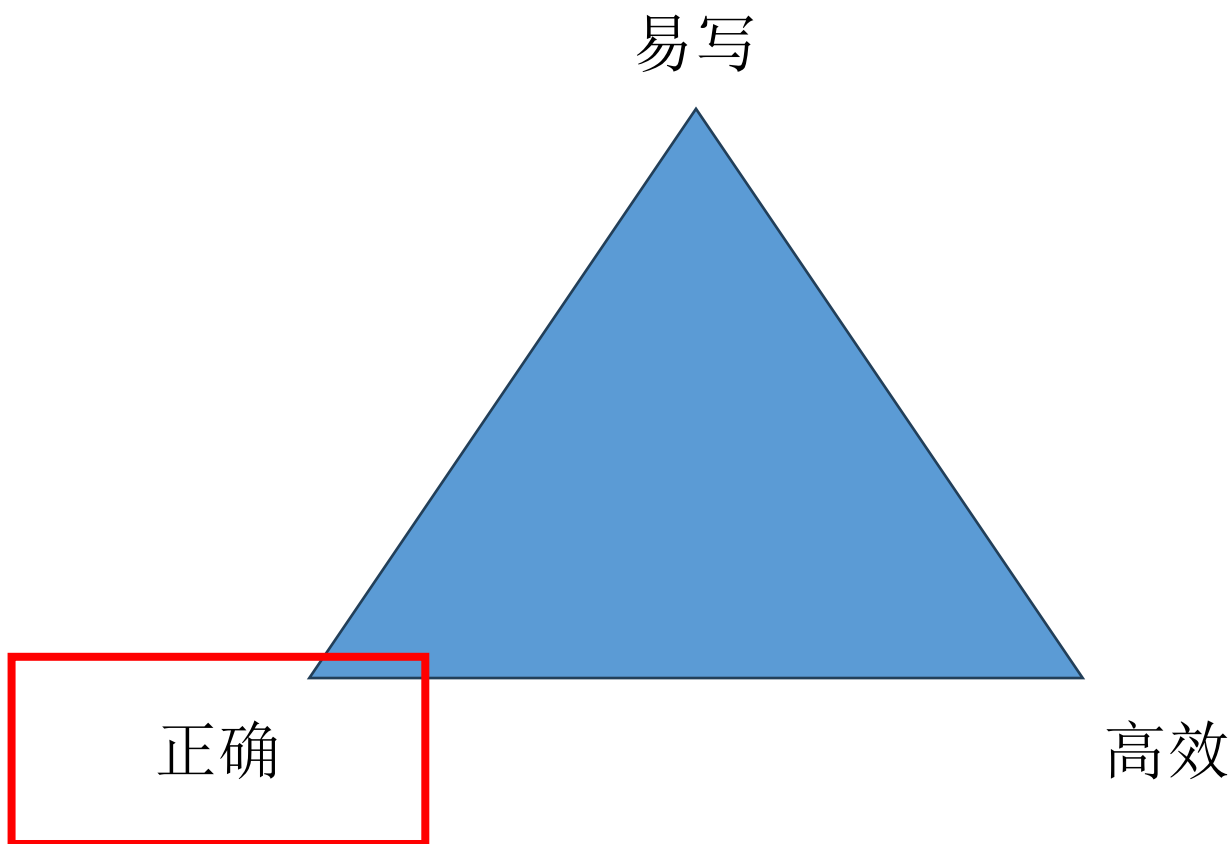
(define (interpret formula)
  (match formula
    [`(^ ,expr ...) (apply && (map interpret expr))]
    [`(v ,expr ...) (apply || (map interpret expr))]
    [`(¬ ,expr)      (! (interpret expr))]
    [lit             (constant lit boolean?)]))

; This implements a SAT solver.
(define (SAT formula)
  (solve (assert (interpret formula))))

(SAT `(^ r o (v s e (¬ t)) t (¬ e)))
```



# 程序设计语言的期望属性







# 如何知道程序是正确的？



程序员

给我写一个排序


写好了，看：  
quicksort (x:xs) =  
quicksort [a | a <- xs, a <= x] ++ [x]  
++ quicksort [a | a <- xs, a > x]



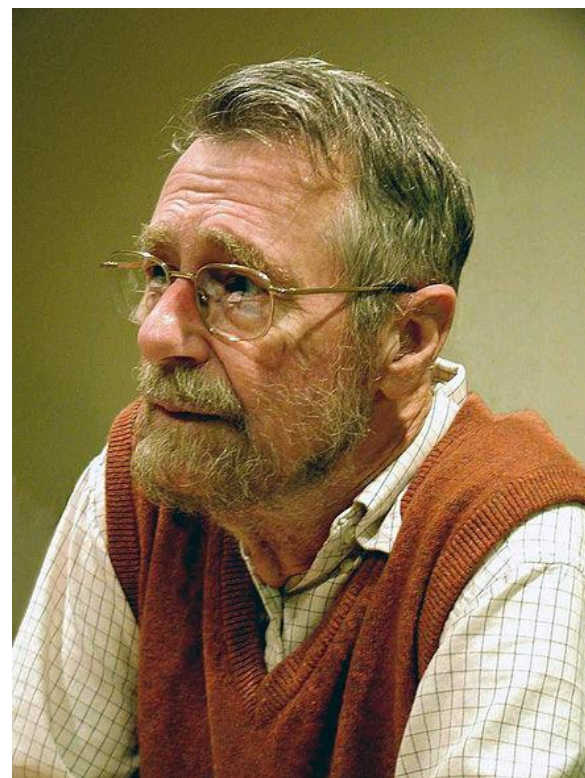
老板

写对了吗？

# 如何知道程序是正确的？

-  程序员：我测试了！

Testing shows the presence,  
not the absence of bugs.



Edsger W. Dijkstra



# 能否证明程序正确性？

- 本质上需要证明程序是满足规约的：
  - $\text{Perm}(\text{sort}(l), l) \wedge \text{Sorted}(\text{sort}(l))$
  - 假设：规约很简单，不会写错
    - 就算写错，也不会刚好和实现错得一样
- 如何证明？
  - 形式语法、形式语义：将程序转换成数学对象，使得可以用数学和逻辑学方法论证正确性
  - 霍尔逻辑：用于证明程序正确性的逻辑
  - 分离逻辑：用于证明指针正确性的逻辑
  - Incorrectness Logic：用于证明程序不正确性的逻辑



# 霍尔逻辑证明举例

(1)  $\{\{ \text{True} \}\}$

$\rightarrow$

(2)  $\{\{ n \times 0 + m = m \}\}$

$X := m;$

(3)  $\{\{ n \times 0 + X = m \}\}$

$Y := 0;$

(4)  $\{\{ n \times Y + X = m \}\}$

while  $n \leq X$  do

(5)  $\{\{ n \times Y + X = m \wedge n \leq X \}\}$

$\rightarrow$

(6)  $\{\{ n \times (Y + 1) + (X - n) = m \}\}$

$X := X - n;$

(7)  $\{\{ n \times (Y + 1) + X = m \}\}$

$Y := Y + 1$

(8)  $\{\{ n \times Y + X = m \}\}$

end

(9)  $\{\{ n \times Y + X = m \wedge \neg(n \leq X) \}\}$

$\rightarrow$

(10)  $\{\{ n \times Y + X = m \wedge X < n \}\}$



# 论证程序的正确性

- 老板：我们公司的1000万行程序都需要证明正确性
- 困难1：人力时间成本
  - seL4：写代码用了2.2人年，写证明用了20人年
- 困难2：怎么知道证明写对了
  - <https://www.win.tue.nl/~gwoegi/P-versus-NP.htm>
    - 至少有62篇论文证明了 $P = NP$ ，50篇论文证明了 $P \neq NP$



# 针对困难1：能不能让计算机自动判断程序的正确性？

- 能否让计算机自动证明程序正确性或不正确性？

## 否定三联



哥德尔

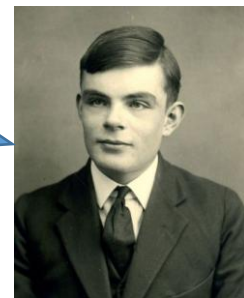
“总是有些定理不存在证明的。”  
——哥德尔不完备定理，1931年

“对于停机这个性质，无论什么算法，总是有程序没法自动证的。”——停机问题，1936年



莱斯

“世界上绝大多数程序性质都跟停机一样没法自动证。”——莱斯定理，1953年

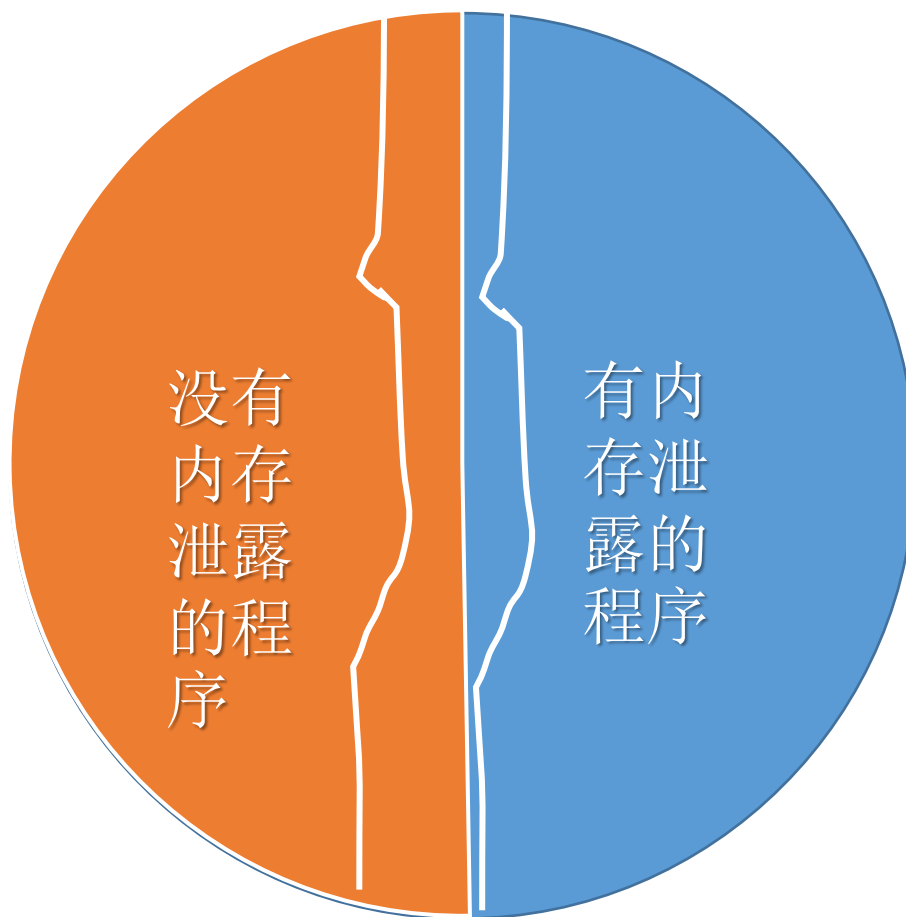


图灵



# 妥协：给出近似的答案

上近似  
/may analysis  
用于证明程序没错



下近似  
/must analysis  
用于证明程序有错



# 自动给出近似答案的技术

- 上近似技术
  - 抽象解释：将分析在容易计算的抽象域进行
  - 类型系统：不容易计算的地方再让用户标记一下
- 下近似技术
  - 随机/模糊测试：只检查部分输入
  - 有界模型检查：只检查特定范围内的输入
  - 符号执行：只检查部分执行路径





# 抽象解释举例

- 给定四则运算表达式，如
  - $a+b*c$
- 如果输入都为正数，结果也一定是正数吗？



# 抽象解释举例

- 正 = {所有的正数}
- 零 = {0}
- 负 = {所有的负数}
- 乘法运算规则：
  - 正 \* 正 = 正
  - 正 \* 负 = 负
  - 正 \* 零 = 零
  - 负 \* 正 = 负
  - 负 \* 负 = 正
  - 负 \* 零 = 零
  - 零 \* 正 = 零
  - 零 \* 负 = 零
  - 零 \* 零 = 零
- 加法运算规则：正 + 正 = 正
- 对任意抽象输入包括的任意具体输入，其对应具体输出包括在抽象输出中



# 抽象解释举例

- 正 = {所有的正数}
- 零 = {0}
- 负 = {所有的负数}

$$a+b*c$$
$$\text{正} + \text{正} * \text{正} = \text{正}$$

- 乘法运算规则:

- |             |             |             |
|-------------|-------------|-------------|
| • 正 * 正 = 正 | • 负 * 正 = 负 | • 零 * 正 = 零 |
| • 正 * 负 = 负 | • 负 * 负 = 正 | • 零 * 负 = 零 |
| • 正 * 零 = 零 | • 负 * 零 = 零 | • 零 * 零 = 零 |

- 加法运算规则: 正 + 正 = 正

- 对任意抽象输入包括的任意具体输入, 其对应具体输出包括在抽象输出中



# 类型系统

- 采用抽象解释分析程序
- 采用特定语法标记已知信息和检查项，确保没有特定类型的错误
- `string={所有的字符串}`
- `int={所有的整数}`
- 运算规则：
  - `string+string=string, int+int=int`
- `string a, b;`  
`factorial(a+b); //类型错误`



# 常见现代类型系统

类型系统	作用
Ownership Type	所有权管理
Refinement Type	细化类型范围
Effect Type	控制副作用
Dependent Type	根据输入细化返回类型
Linear Type	资源一次性使用管理
Session Type	确保通信协议的消息顺序和种类匹配
Gradual Type	允许动态语言逐步添加类型
Union/Insect Type	类型的灵活组合



# Dependent Type

- 长度固定的列表类型 (Coq)

```
Inductive list : nat -> Type :=  
  | nil : list 0  
  | cons : forall n, nat -> list n -> list (S n).
```



# Refinement Type

- 限制输入为正 (LiquidHaskell)

`square :: x:{Int | x >= 0} -> Int`



# Effect Type

- 描述函数副作用 (Koka)

`add : (int, int) -> total int` //没有副作用

`add : (int, int) -> IO int` //会读写输入输出

`add : (int, int) -> int state` //会修改变量





# Session Type

- 描述通信的顺序和数据类型（Rust Session库）

```
type PingPong = Send<String, Recv<String, Eps>>;
```



# 上近似的副作用

- 类型系统会禁止部分正确程序的编写
  - `int a = 1;`  
`int b = &a;`  
`int* c = b;`  
`return *c;`
  - 部分高效算法已经无法在Rust写出
- 因此，现代类型系统也只能避免很小一部分错误类别

# 针对困难2：能否自动检查证明的正确性？



- 能，并且能套用类型检查算法

Haskell Brooks Curry



“‘命题-证明’和‘类型-值’之间存在对应关系。”  
——Curry-Howard Correspondence, 1934-1969



# 交互式定理证明语言

- 交互式定理证明语言
  - 支持定义程序、命题和证明
  - 自动检查证明是否证明命题
    - 并随时提示程序员还没有完成证明的部分
- 常见语言：Isabella、Coq、Lean



# 交互式定理证明例子

```
Inductive day : Type :=  
| monday  
| tuesday  
| wednesday  
| thursday  
| friday  
| saturday  
| sunday.
```

```
Definition next_weekday (d:day) : day :=  
match d with  
| monday      => tuesday  
| tuesday     => wednesday  
| wednesday   => thursday  
| thursday    => friday  
| friday      => monday  
| saturday    => monday  
| sunday      => monday  
end.
```

```
Example test_next_weekday:  
  (next_weekday (next_weekday saturday)) = tuesday.
```

```
Proof. simpl. reflexivity. Qed.
```



# 交互式程序证明语言

- 交互式定理证明语言不是为验证程序设计的
  - 首先要在系统中建模程序的语法和语义
  - 然后调用相应定理证明
  - 非常繁琐复杂
- 直接将证明系统和高级编程语言结合起来
  - 绑定自动证明工具
    - Dafny, Frama-C, VST-A (上交)
  - 支持在高级语言中手写证明
    - C\* (北大), F\*



# Dafny程序举例

```
function fib(n: nat): nat
{
  if n == 0 then 0
  else if n == 1 then 1
  else fib(n - 1) + fib(n - 2)
}
```

```
method ComputeFib(n: nat) returns (b: nat)
  ensures b == fib(n)
{
  var i := 1;
  var a := 0;
  b := 1;
  while i < n
    invariant 0 < i <= n
    invariant a == fib(i - 1)
    invariant b == fib(i)
  {
    a, b := b, a + b;
    i := i + 1;
  }
}
```



# 结论

- 程序设计语言研究历史悠久，流派众多
- 虽然研究者大多修仙，但成果可能是有用的
- 从挖矿是可行的研究思路
- 希望今天的报告能提供一个矿区概览
- 非常感谢！