2025年西安交通大学
形式语言与编译

前 言

- 在现代计算机系统中，编译器紧随计算机硬件架构的演化，发挥着越来越重要的作用。
- 自1957年的第一个编译器IBM Fortran问世至今，尽管计算机系统已发展到这样的阶段——多样化的处理器架构如此丰富，以至于计算世界如此繁荣，然而处理器架构与高级语言程序之间依然存在着巨大的“鸿沟”，
- 让用高级语言写的程序提供尽可能接近手动调整的机器代码的性能始终是编译器的核心目标。

硬件架构发展至今成果丰富

- 从20世纪60年代由Burroughs公司制造的STAR-100、ILLIAC IV到21世纪20年代的各种微处理器芯片、高端处理器芯片、GPU核等，所有架构方面的成果得以整合和体现，从一方面加剧了“鸿沟”这一事实。
- ILP（流水线、超标量、乱序、SIMD等）
- TLP（多核、SIMT等）
- CPU-GPU等
- 高端并行机等

高级语言发展至今种类繁多

- 另一方面，计算机架构方面任何合理的进步都伴随着编译器技术方面合理的进步。数量众多的高级程序设计语言为编译技术开拓了广阔的发展空间。

- 据统计，程序设计语言已发展到多达数千种。从常见的程序设计语言排行榜可见到100种最为活跃的语言。比如，TIOBE发布的2025年1月编程语言排行榜（https://hellogithub.com/report/tiobe），排名居前的编程语言为：Python，C++，Java，C，C#，JavaScript，Go，SQL，Visual BASIC，Fortran，Delphi/Object Pascal等。

AI编程助手的出现

- 随着可以自动生成代码的AI编程助手的出现，程序开发的生产力得以空前提升。在这样一种程序“爆炸”场景下，更深层次的编译反馈以及更为优化的代码生成方法成为新的编译挑战。在高性能计算、可信计算、网络空间安全等领域，从编译器中获取功能并作为责任移交到程序员的手里，展现出编译知识更为广泛的应用场景。

本课程强调理论性

- 事实上，今天的编译器已经成为一种庞大的不可替代的复杂软件，是工业基础软件高地之一。常见的像GCC和LLVM这些主流编译器，除了有广泛的移植性等优点以外，还担任研发平台角色，并在安全、可信等领域得到广泛应用。

- 在以上所述复杂背景下，面对原理、方法、技术、实现等多方面编译知识，将强调理论性作为本课程的一个重要属性，以帮助学生应对复杂的细节，提升针对共性问题领悟到的思维创新能力。

本课程采取的“化难为易”措施

- 目的是为了缓解编译课程普遍存在的“难学”制约能力的问题。
- 其一，采用文法作为一条确定的、可扩展的主线来组织知识，避免出现常见的编译知识“碎片化”现象造成思维不连续困惑。
- 其二，提供一个同时满足简明性、一致性、完备性要求的主符号系统，作为形式化描述知识的手段，有助于思维的逻辑性、抽象性和一致性，并以此提升学习效率。
- 其三，强调从“为什么”方面来表达知识，有助于学生在学习过程中进行主动思考。

第1章 绪论

- 关于课程名
  - “形式语言与编译”是本校自行设计的课程，对标“编译原理”，强调形式语言理论及其在回答编译原理中诸多为什么之问题的作用，有助于化难为易、启发思维这种学习过程中发生的理想状况之呈现。

- 从三个角度开始了解本章内容：
  - 什么是编译？
  - 为什么学编译？
  - 本课程是什么？

问题1：什么是编译？

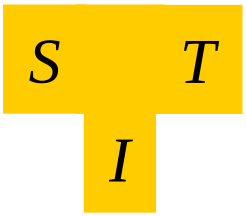
- 是指某个编译程序（语言处理器）：也称为编译器、编译系统，也对应于解释程序（解释器）。
- 是指某类软件框架：前端（多源语言）、后端（多目标机）、中间表示、自动生成工具（词法及语法分析器生成器、代码生成器的生成器）、多优化遍集成等。
- 是指某种程序转换过程（含分析与综合），如5阶段编译过程、编译遍等。
- 是关于一套理论和方法：2型和3型形式语言理论（多语言联合DFA、itemPDA等）、LL(*k*)和LR(*k*)分析法、属性文法与语法制导翻译、运行时环境、代码优化、流分析、寄存器分配等。

编译程序

- 一个编译程序本身是个程序，器功能是将用某个语言写的程序（称为源程序）作为输入，并翻译成为功能上等价的另一个语言程序（称为目标程序）。
- 用来写源程序所用的语言，即源语言，通常是高级语言，如C，FORTRAN等。
- 用来写目标程序所用的语言，即目标语言，通常是低级语言，如汇编或机器语言。
- 随着翻译过程的进行，编译程序也报告错误信息以帮助程序员改错，直到翻译通过为止。

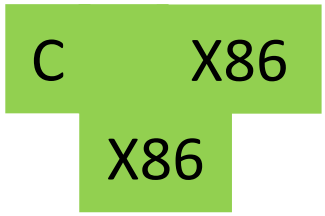
编译器的T型图表示

源程序 ⇨ 编译器 ⇨ 目标程序/错误信息

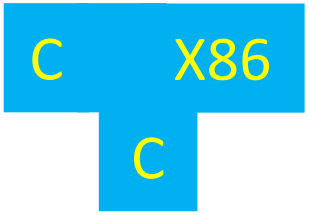


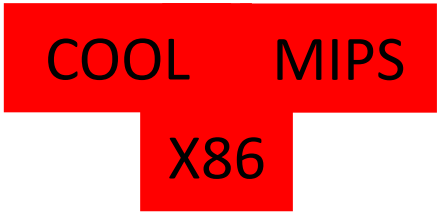
- *S* – 源语言（高级语言）
- *T* – 目标语言（低级语言）
- *I* – 实现语言

用*I*语言写的编译器被用于将*S*语言写的源程序翻译成用*T*语言写的目标程序

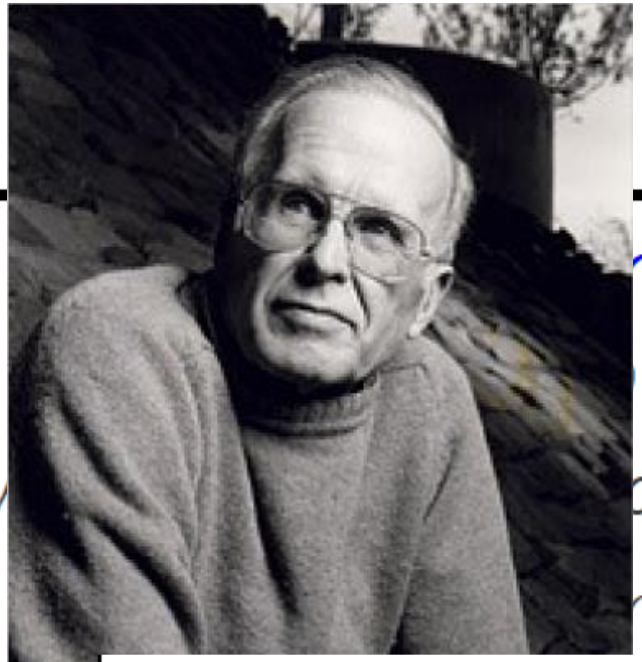
.c 

.o





高级语言



race

r of

d the

r."

John Backus, team lead on FORTRAN.

Image: http://upload.wikimedia.org/wikipedia/commons/thumb/5/55/Grace_Hopper.jpg/300px-Grace_Hopper.jpg

http://www.nytimes.com/2007/03/20/business/20backus.html

分层认识编程语言

ELF/COFF

```
#as globe var
.data
msg:   .ascii "Hello World"

#as main
.text
la $a0 msg
li $v0 4
syscall
```

```
1  #include <stdio.h>
2
3  main() {
4    printf("hello world\n");
5  }
```

- 低级语言（硬件直接执行）：指令、地址、RAM;
- 高级语言（命令式与陈述式等）（名字、值）。
- 高级语言数千种每年还有近百种被发明。
- 对编译器的影响：要求适应新语言、利用新硬件、提升性能

编译器示意

目标语言是高级/低级语言时？
实现语言是高级/低级语言时？
目标机与宿主机相异时？ 交叉编译器

```
1  #include <stdio.h>
2
3  main() {
4    printf("hello world\n");
5  }
```

源程序

错误信息

编译器

源语言

实现语言

目标程序

```
#as globe var
.data
msg:   .ascii "Hello World"

#as main
.text
la $a0 msg
li $v0 4
syscall
```

目标语言

解释器

源程序

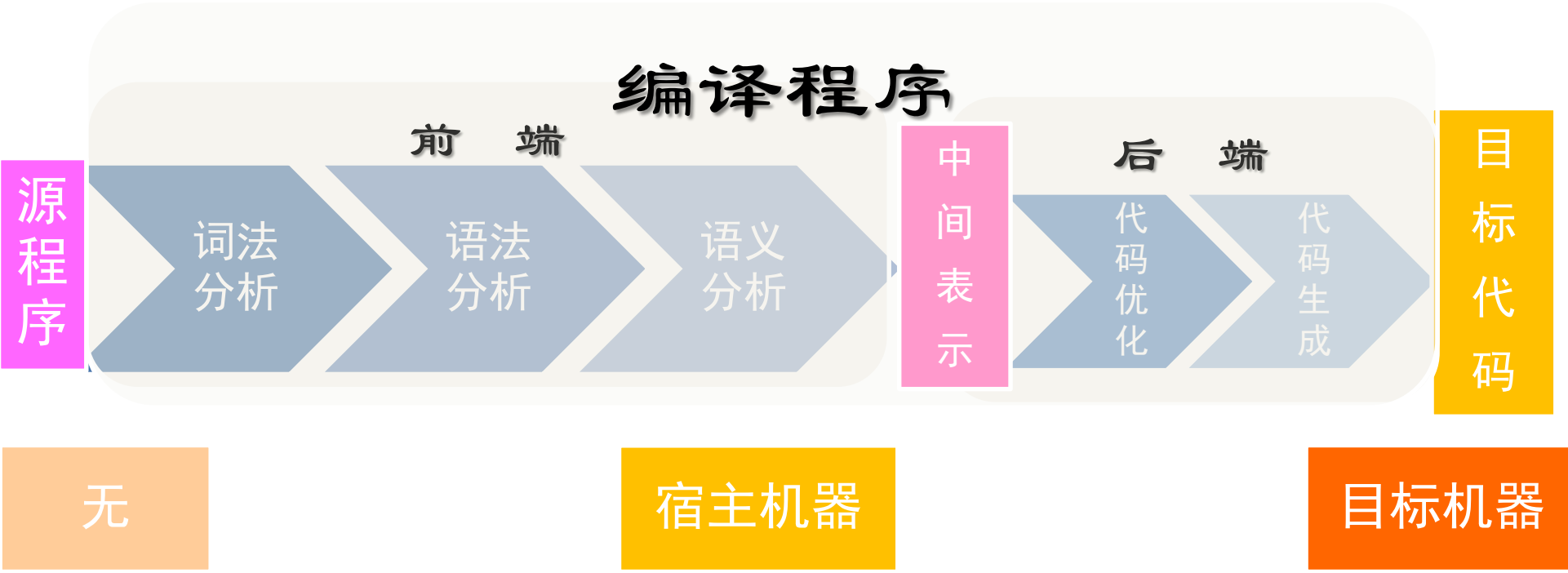
源程序的输入

源程序的输出/错误信息

- **与编译器对比：**
  - 概念上的不同（翻译/直接执行）
  - 基于解释执行的程序可以动态修改自身，而基于编译执行的程序动态性较弱
  - 基于解释方式便于调试
  - 基于解释方式有利于人机交互
  - 基于解释执行的速度较慢
  - 解释器需要保存的信息较多，空间开销大
  - 二者实现技术相似

编译过程与编译程序

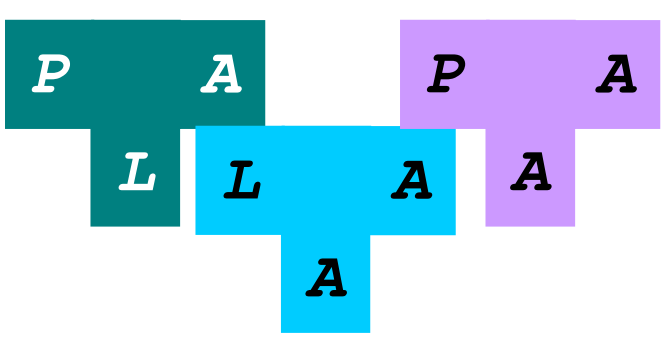
源语言　　实现语言　　目标语言

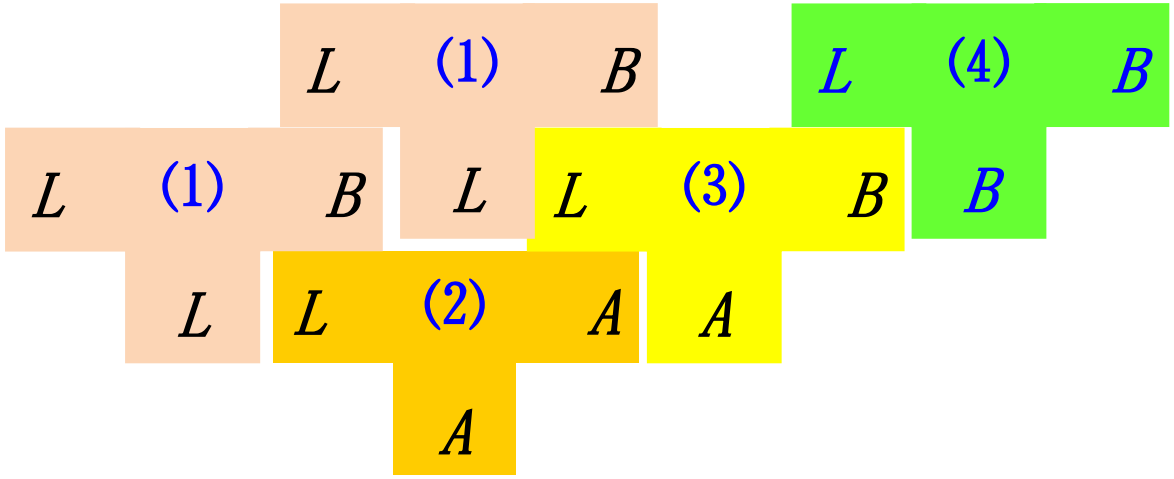


无　　宿主机器　　目标机器

✓ 涉及到三个程序，各程序是用什么语言写的？运行的机器是什么？

用T型图表示移植编译器的过程

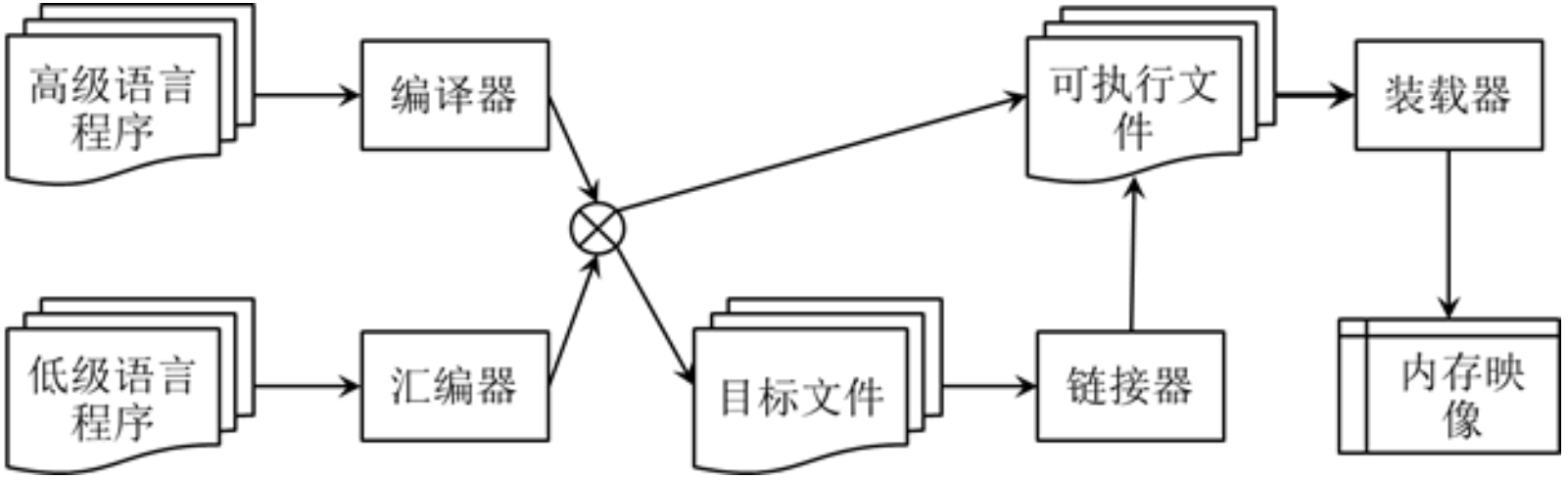




给定用*L*写的*P*语言编译器，使用*L*语言的*A*机器编译器进行编译，得到*A*机器的*P*语言编译器

将编译器(1)利用编译器(2)移植为编译器(4)

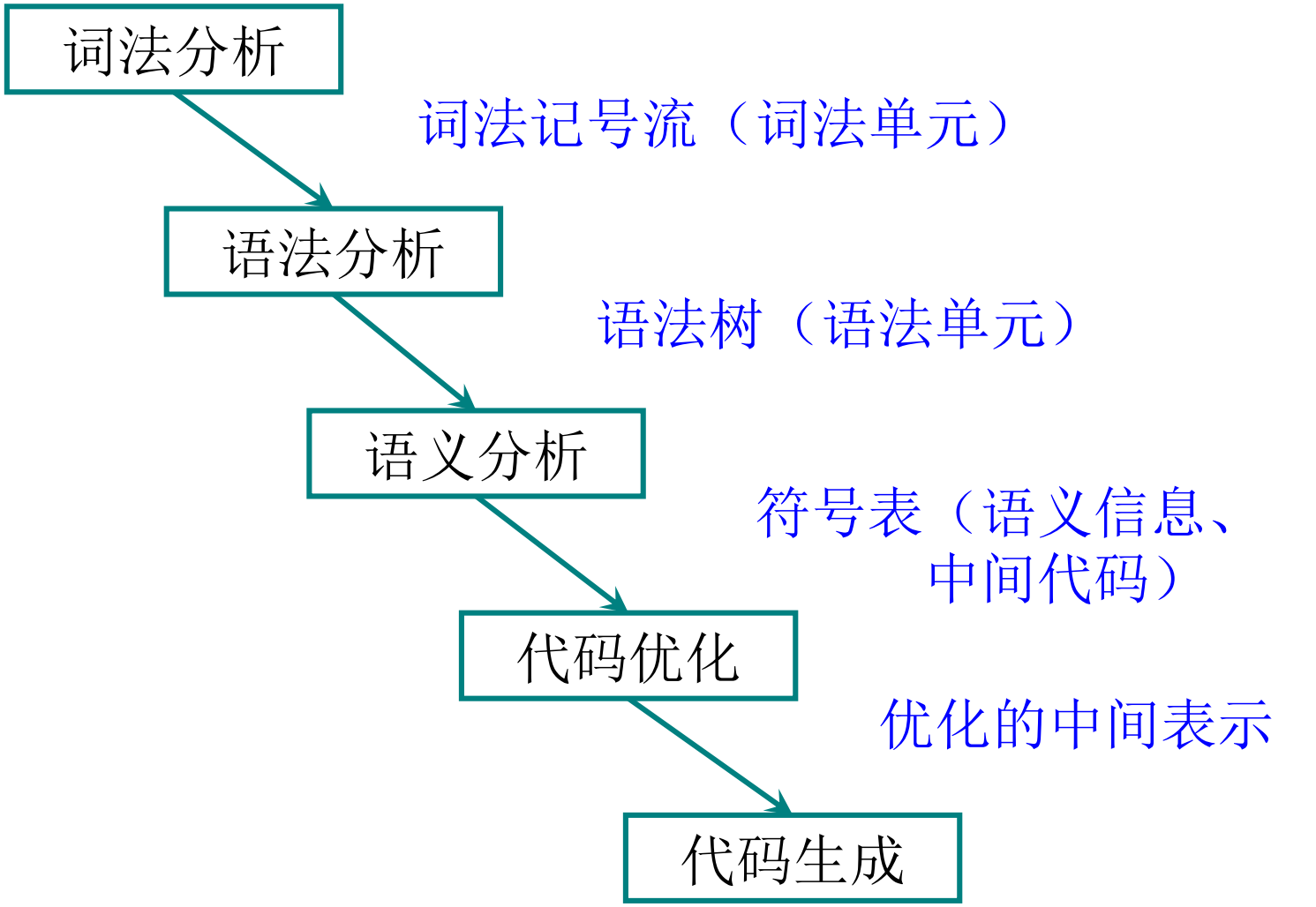
编译工具链示意



- **编译器种类：**
  - 诊断编译程序：侧重于错误信息，帮助开发调试
  - 优化编译程序：侧重于提高代码效率
  - 交叉编译程序：一般为嵌入式系统生成可执行代码
  - 可重定向编译程序：仅定制编译器中跟机器有关部分实现针对新目标机的代码生成

编译过程

源程序（字符流）


词法分析
词法记号流（词法单元）
语法分析
语法树（语法单元）
语义分析
符号表（语义信息、
中间代码）
代码优化
优化的中间表示
代码生成
目标程序（可执行代码）

Step1 词法分析

```
int x;
int fact(int n; int a;){
    if (n==1) return a else return fact (n-1, n*a,)
};
x=123+fact(5,1,);
print x
```

- int fact(int n; int a;){\n\tif (n==1) return a else return fact(n-1, n*a,)\n};

i	n	t		f	a	c	t	(	i	n	t		n	;		i	n	t		a	;	)	{	↲	→	i	f	(	n	=	=	1	)	r	e

词法分析结果：词法记号串

- int fact(int n; int a;){\n\tif (n==1)
- (INT, _) (ID, fact) (LPA, _) (INT, _) (ID, n) (SCO, _) (INT, _) (ID, a) (SCO, _) (RPA, _) (BLA, _) (IF, _) (LPA, _) (ID, n) (ROP, ==) (NUM,1)

- 常数：NUM，FLO
- 标识符：用户定义的名字：ID
- 关键字/保留字：系统提供的名字：INT，IF，ELSE, RETURN
- 运算符：AOP，ROP
- 分隔符：LPA，SCO，RPA，BLA，CMA，BRA

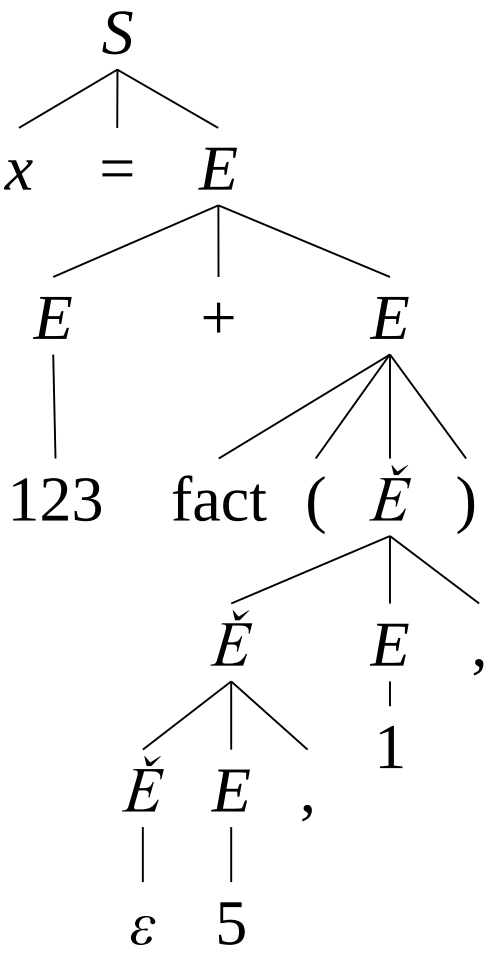
- x=12..3+fa#ct(5; 1);　有词法错误?

正则语言理论

Step2 语法分析

- 依据语言的语法规则，将词法记号序列转化为一些语法单元（短语、句子、程序）及其之间的关系，并确定整个输入串是否构成一个语法上正确的程序。

右图对应于列表1-1的78~98（下页）



上下文无关语言理论

x=123+fact(5,1,)

78 _____*S*　　*code*:[t6 = 123; ③; t10 = t6+t9; x = t10]

79 _____*d*<x>

80 _____=

81 _____*E*　　*place*:t10; *code*:[t6 = 123; ③; t10 = t6+t9]; 登记t10到@table

82 _____*E*　　*place*:t6; *code*:[t6 = 123]; 登记t6到@table

83 _____*i*<123>

84 _____+

85 _____*E* *place*:t9; *code*:[t7=5; t8=1; PAR t8; PAR t7; t9=CALL fact, 2]③; 登记t9到

86 _____*d*<fact>　　@table

87 _____(

88 _____*Ě*　　*place*: (t7 t8); *code*: ([t7 = 5] [t8 = 1])

89 _____*Ě*　　*place*: (t7); *code*: ([t7 = 5])

90 _____*Ě*　　*place*:NIL; *code*:[]

91 _____*ε*

92 _____*E*　　*place*:t7; *code*:[t7 = 5]; 登记t7到@table

93 _____*i*<5>

94 _____,

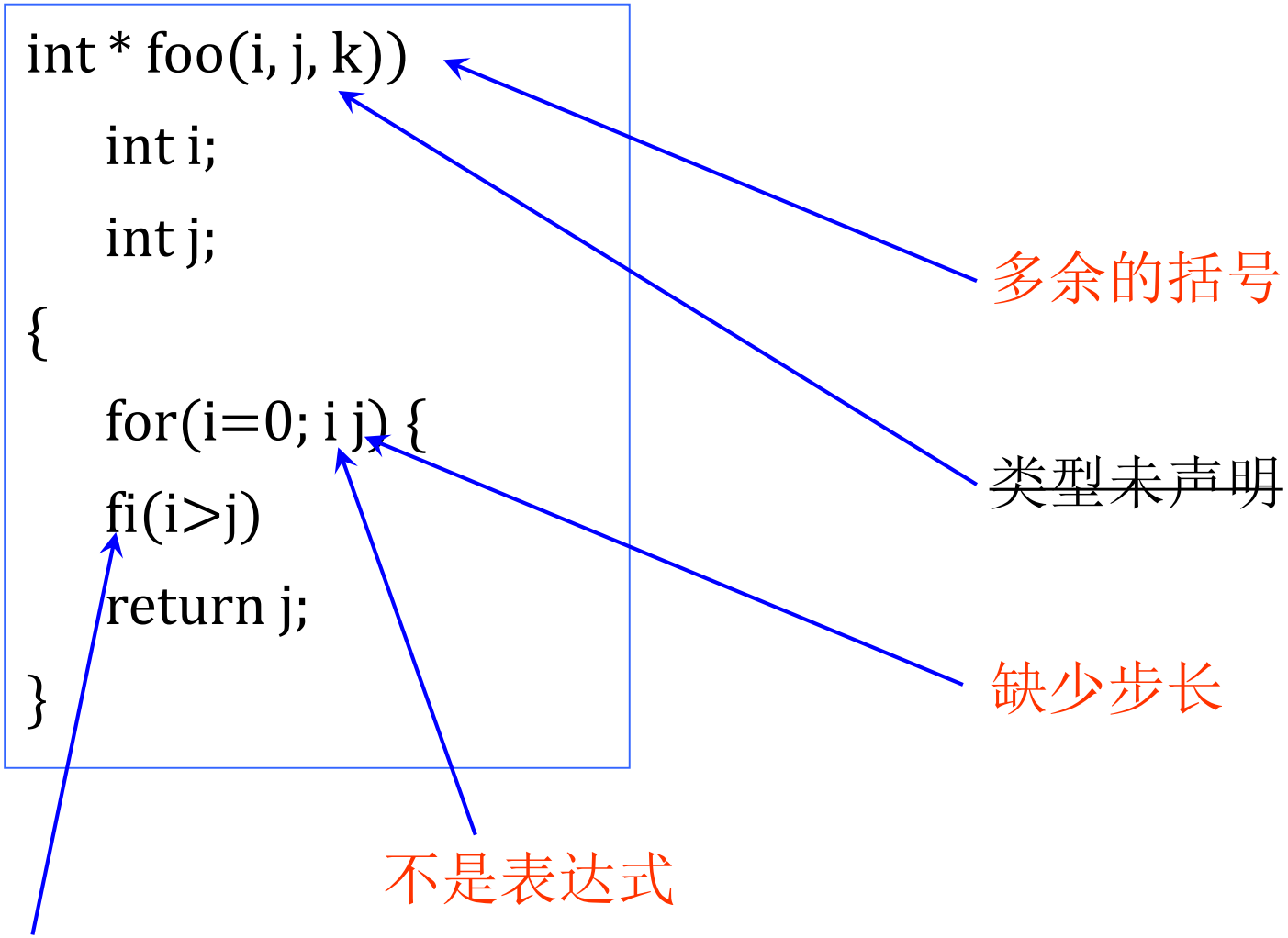
95 _____*E*　　*place*:t8; *code*:[t8 = 1]; 登记t8到@table

96 _____*i*<1>

97 _____,

98 _____)

语法分析发现语法错误（示例）



Step3 语义分析

- 对语法分析所识别出的各个语法单元，知道其语义：
  - 从程序声明中提取语义信息并登记到符号表里；
  - 对程序语句都生成中间代码并以函数为单位进行组织。
- 发现语义错误。

对示例程序进行语义分析的过程
标注在列表1-1各行上，
如PPT片所示（行78~98）。
所得结果为2个符号表，见下页所示。

属性文法、语法制导翻译

fact程序的语义分析结果

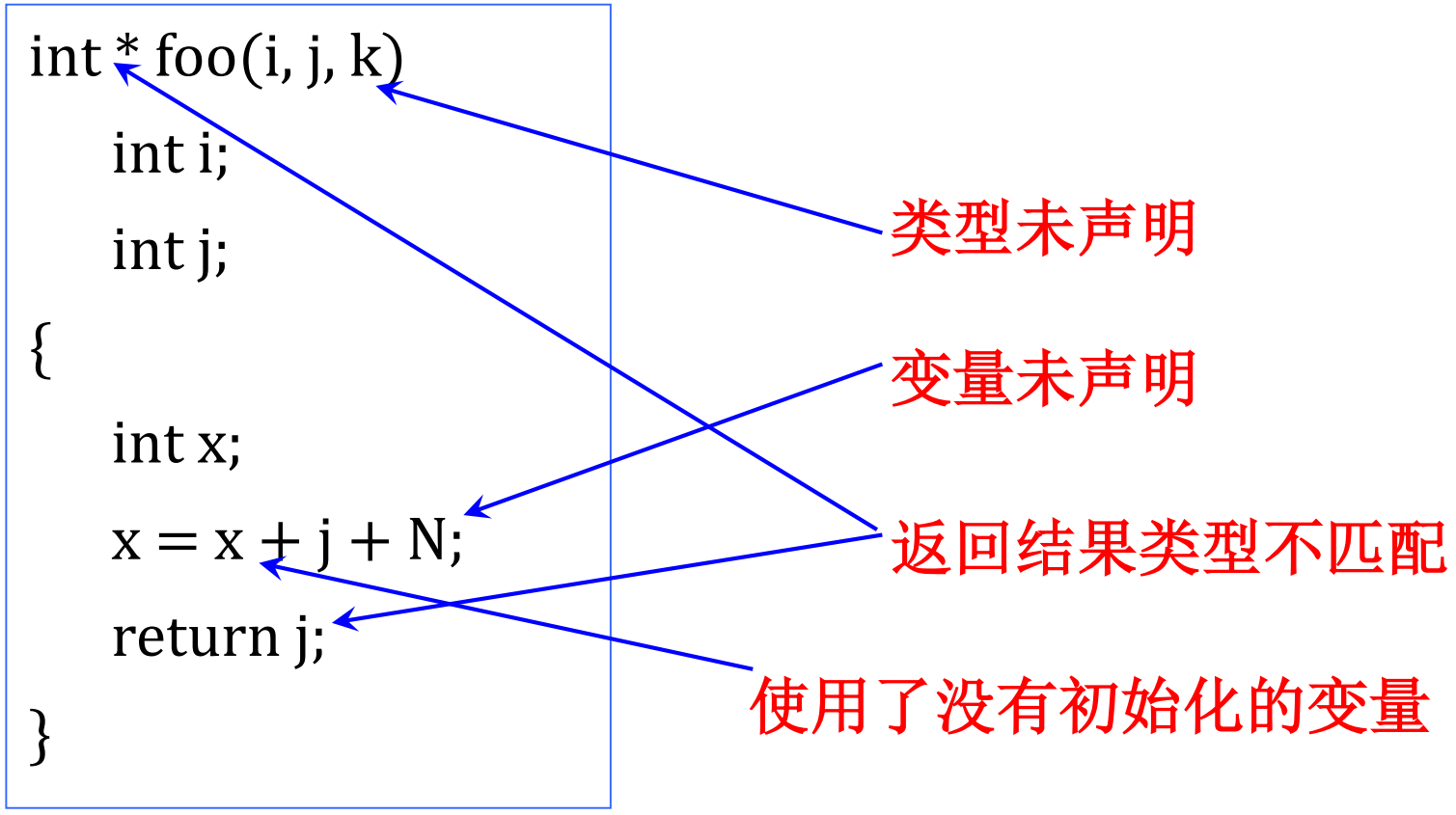
```
int x;
int fact(int n; int a;){
    if (n==1) return a else return fact (n-1, n*a,)};
x=123+fact(5,1,);
print x
```

声明2个函数、1个全局变量x、2个形参变量

最外层函数的2个语句、fact函数的if语句

```
@table: (
    outer: NULL width: 28 argc: 0 arglist: NIL rtype: INT level: 0
    code: [t6=123; t7=5; t8=1; PAR t7; PAR t8; t9=CALL fact, 2; t10=t6+t9; x=t10; PRINT x]
    entry: (name: x type: INT offset:4)
    entry: (name: fact type: FUNC offset: 12 mytab: fact@table)
    entry: (name: t6 type: INT offset: 16) entry: (name: t7 type: INT offset: 20)
    entry: (name: t8 type: INT offset: 24) entry: (name: t9 type: INT offset: 28))
fact@table: (
    outer: @table width: 30 argc: 2 arglist: (n a) rtype: INT level: 1
    code: [t1=1; IF n<=t1 THEN l1 ELSE l2; LABLE l1; RETURN a; GOTO l3; LABEL l2;
    t2=1; t3=n-t2; t4=n*a; PAR t3; PAR t4; t5=CALL fact, 2; RETURN t5; LABEL l3]
    entry: (name: n type: INT offset: 4) entry: (name: a type: INT offset: 8)
    entry: (name: t1 type: INT offset: 12) entry: (name: t2 type: INT offset: 16)
    entry: (name: t3 type: INT offset: 20) entry: (name: t4 type: INT offset: 24)
    entry: (name: t5 type: INT offset: 30))
```

语义分析发现语义错误（示例）



Step4 优化

int fact(int n; int a;){if (n==1) return a else return fact(n-1, n*a,)}

- 消除尾递归

[t1=1; IF n=t1 THEN l1 ELSE l2; LABLE l1; RETURN a; GOTO l3; LABEL l2; t2=1; t3=n-t2; t4=n*a; PAR t3; PAR t4; t5=CALL fact, 2; RETURN t5; LABEL l3]

[LABEL l0; t1=1; IF n=t1 THEN l1 ELSE l2; LABLE l1; RETURN a; GOTO l3; LABEL l2; t2=1; t3=n-t2; t4=n*a; ~~PAR t3; PAR t4; t5=CALL fact, 2; RETURN t5~~ n=t3; a=t4; GOTO l0; LABEL l3]

Step5 可执行代码

- 运行时环境
- 构建函数的可执行代码
  - 序言、尾声
  - PAR-CALL转换为调用指令序列callseq
  - RETURN转换为返回指令序列retseq
  - 将局部名字的定义与引用代码替换为基于栈帧的存储访问代码

可执行代码示例

fact@label:

[sp = sp−4; M[sp] = $ra; sp = sp−fact@width; #序言

n = M[fp+8]; a = M[fp+12]; #参数传递

LABEL l0; t1 = 1; slt t0, t1, n; beq t0, R0, l2; #指令模板；替换t1为M[fp−16]

LABEL l1; $v0 = a; GOTO l4; #置返回值并转尾声

LABEL l2; t2 = 1; t3 = n−t2; t4 = n*a; n = t3; a = t4; GOTO l0; #已优化尾递归为循环

LABEL l3;

LABEL l4; t0 = M[fp−4]; sp = fp; JR t0] #尾声

其他编译器话题

- 错误处理程序
- 符号表管理
- 编译遍
- 一遍与多遍编译
- 前端、后端、中间端
- 构造工具：LEX、YACC等
- 应用

一遍与多遍编译

- 是对源程序或编译中间结果从头到尾扫描一次，并做有关的加工处理，生成新的中间结果或目标程序。

- 注意各遍间的中间结果（内部形式、外部形式）

One-pass compiler: Type of software compiler that passes through the source code only once. One-pass compilers are faster, but may not generate an as efficient program. In addition, one-pass compilers cannot compile all types of source codes.
http://www.computerhope.com/jargon/o/onepassc.htm

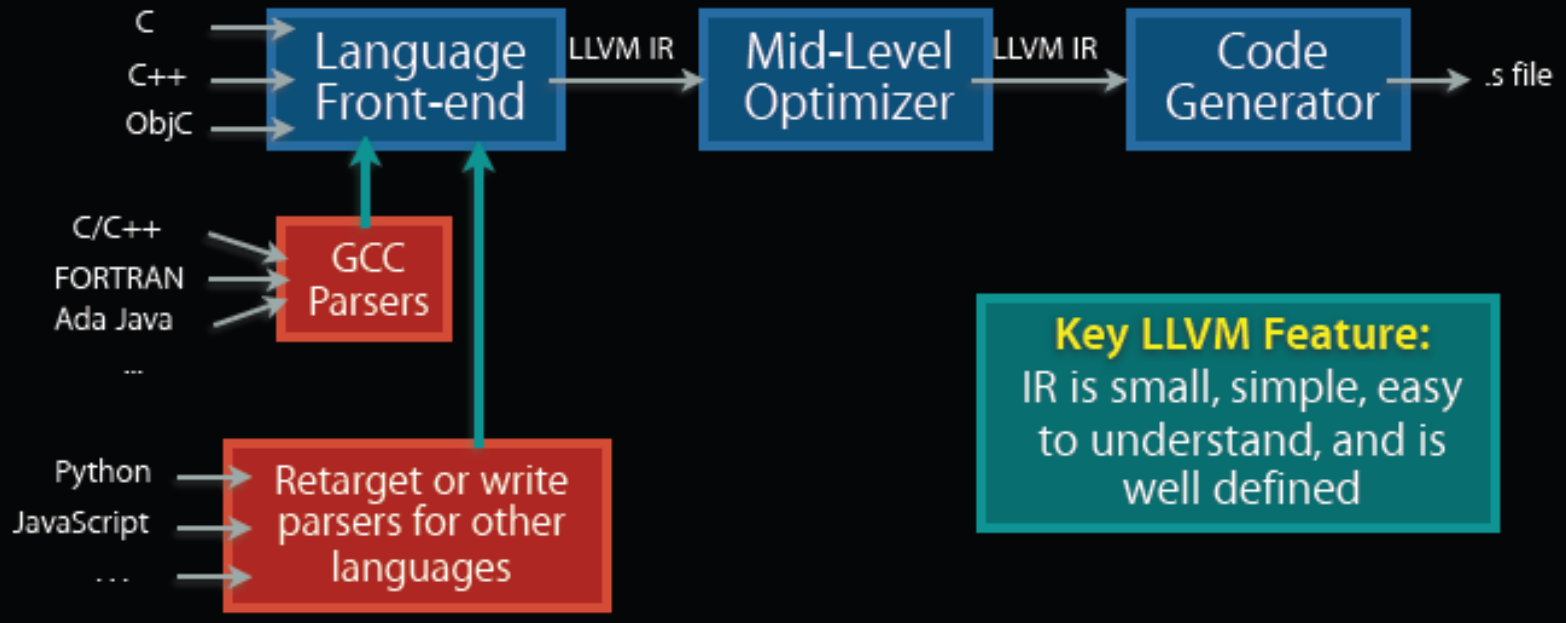
Example of a Simple Static Compiler

- Standard compiler organization, which uses LLVM as midlevel IR:
  - Language specific front-end lowers code to LLVM IR
  - Language/target independent optimizers improve code
  - Code generator converts LLVM code to target (e.g. IA64) code

Many compilers (e.g. GCC) follow this model.

Front-end options for this compiler

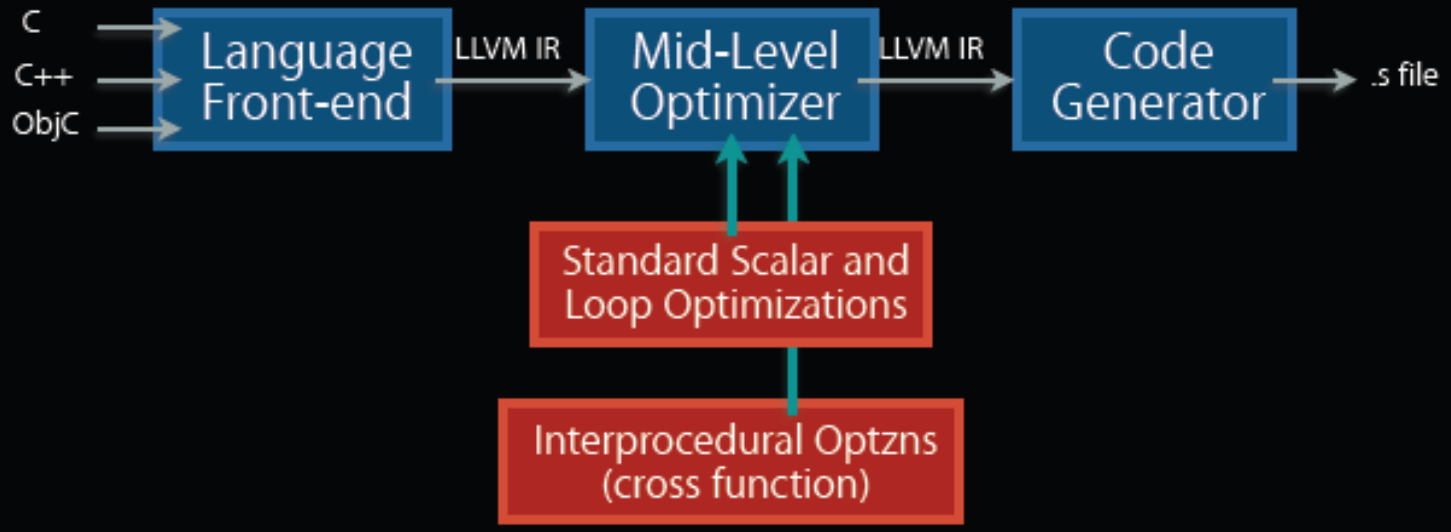
- Front-ends are truly separate from optimizer & codegen
  - Can use front-end AST's that are tailored to the source language
  - Optimizer & Codegen improvements benefit all front-ends
  - Front-ends generate debug info and include it in the IR



llvm-gcc currently uses the GCC 4.0.1 parsers

Optimizer options for this compiler

- Optimizer is solely concerned with semantics of LLVM IR
  - Optimizer & Codegen only know LLVM, not all source languages
  - LLVM includes IP framework and aggressive IP optimizations
  - LLVM uses a modern and light-weight (fast) SSA-based optimizer



编译技术的应用

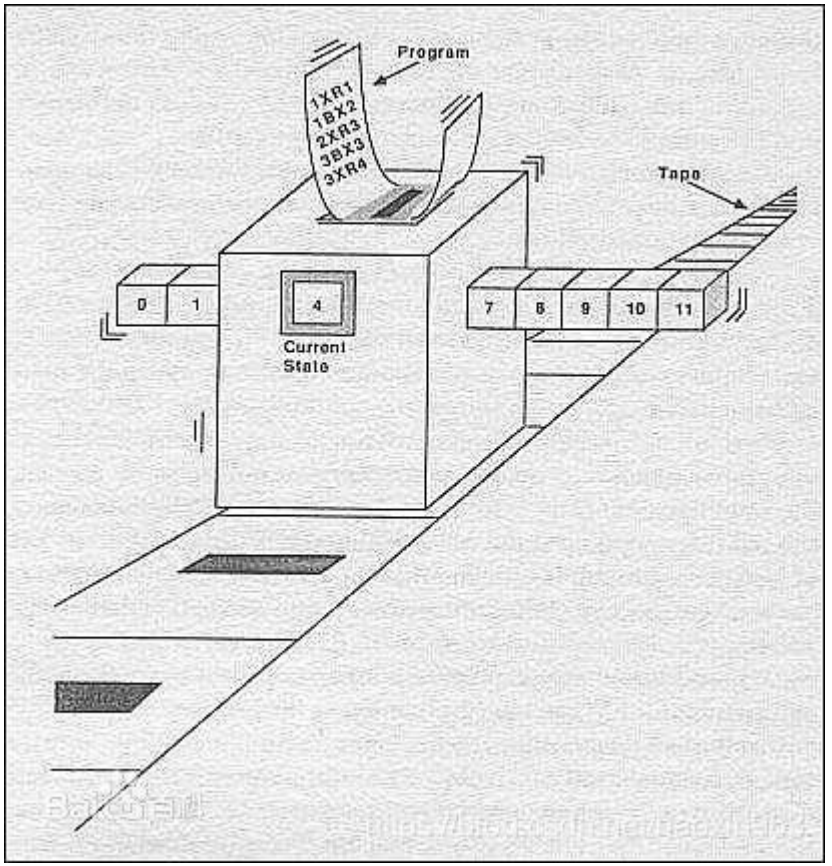
- 高级程序设计语言的实现
  - 语言特征与编译优化
  - 动态编译
- 针对计算机体系结构的优化
  - 并行性
  - 内存层次结构
- 新计算机体系结构的设计
  - RISC、专用体系结构
- 程序翻译
  - 二进制翻译、硬件合成、数据查询解释器、编译-模拟
- 软件生产率工具
  - 类型检查、边界检查、内存管理工具

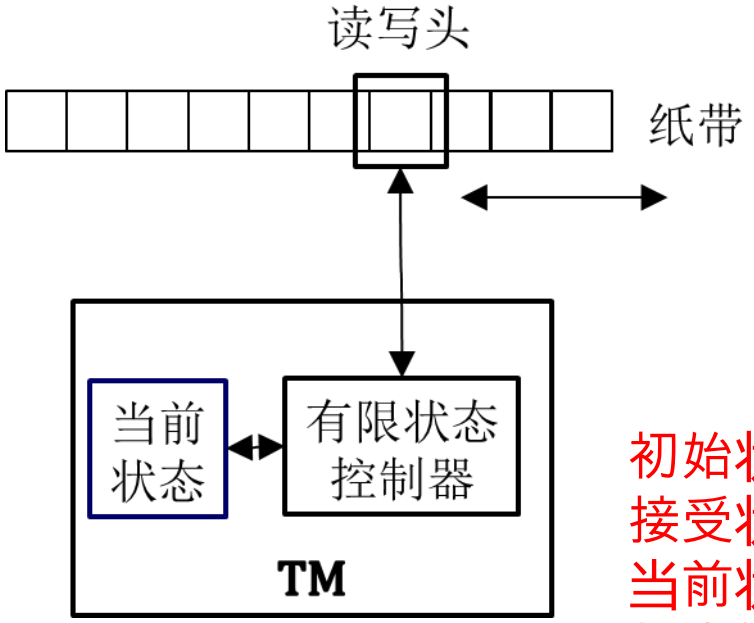
问题2：为什么学编译？

- 为了掌握高级语言翻译为机器语言的原理和方法？
- 为了会编写编译器？
- 为了对程序有更深的理解？

从图灵机认识抽象计算模型

图灵机是由英国数学家A.M.图灵1936年提出的一种抽象计算模型，即将人们使用纸笔进行数学运算的过程进行抽象，由一个虚拟的机器替代人们进行数学运算。





初始状态；
接受状态；
当前状态；
状态转换规则集。

图灵机例子

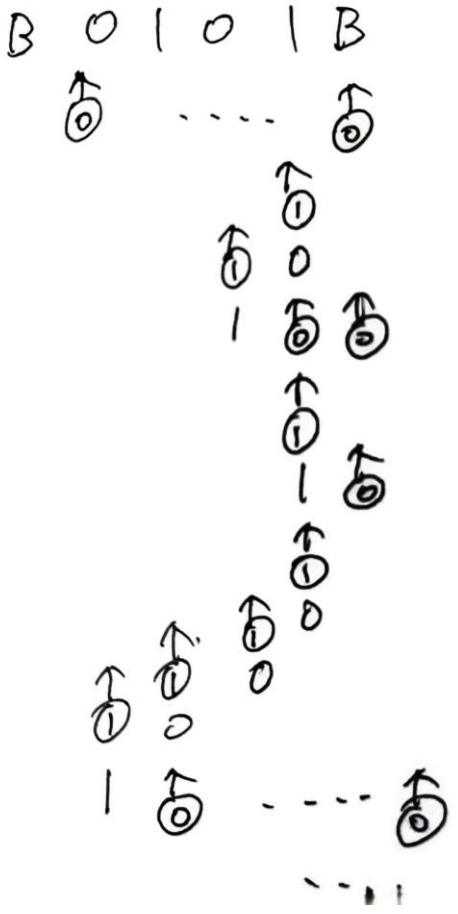
- (State Number, Symbol Read) -> (Next State Number, Symbol To Write) Next Cell

- Symbol Read：(0, 1, 空格)
- (1,0) -> (0,1) Left
  - 若机器处于状态 1且单元内容为0，则修改为1、状态转为0、向左移动一个单元
- (1,1) -> (1,0) Left
  - 在状态1若单元内为1则改为0、停留状态1并向左移动一格
- (1,B) -> (0,1) Right
  - 若单元内为空白改为1、状态转为0并右移动一格
- (1,B) -> (0,1) Halt
  - 停机指令，若当前单元为空白则改为1状态转0并停机

计数器图灵机示例

```
(0, 1) -> (0, 1) Right
(0, 0) -> (0, 0) Right
(0, B) -> (1, B) Left

(1, 0) -> (0, 1) Right
(1, 1) -> (1, 0) Left
(1, B) -> (0, 1) Right
```



无符号数加1图灵机：第4、第6条规则中的Right改为Halt，其它不变。（接受状态隐含停机）

通用图灵机

通用图灵机是图灵机，它能模拟任意图灵机。

具体地说，如果把任意一个图灵机的指令集用图灵于1936–1937年自己提出的一种规范方式编码并预存在纸带上，那么通用图灵机就能够根据纸带上已有的信息，在纸带的空白处模拟那台图灵机的运作，输出那台图灵机应该输出的结果。

通用图灵机被认为是存储程序计算机（冯诺依曼结构1946年）的起源。

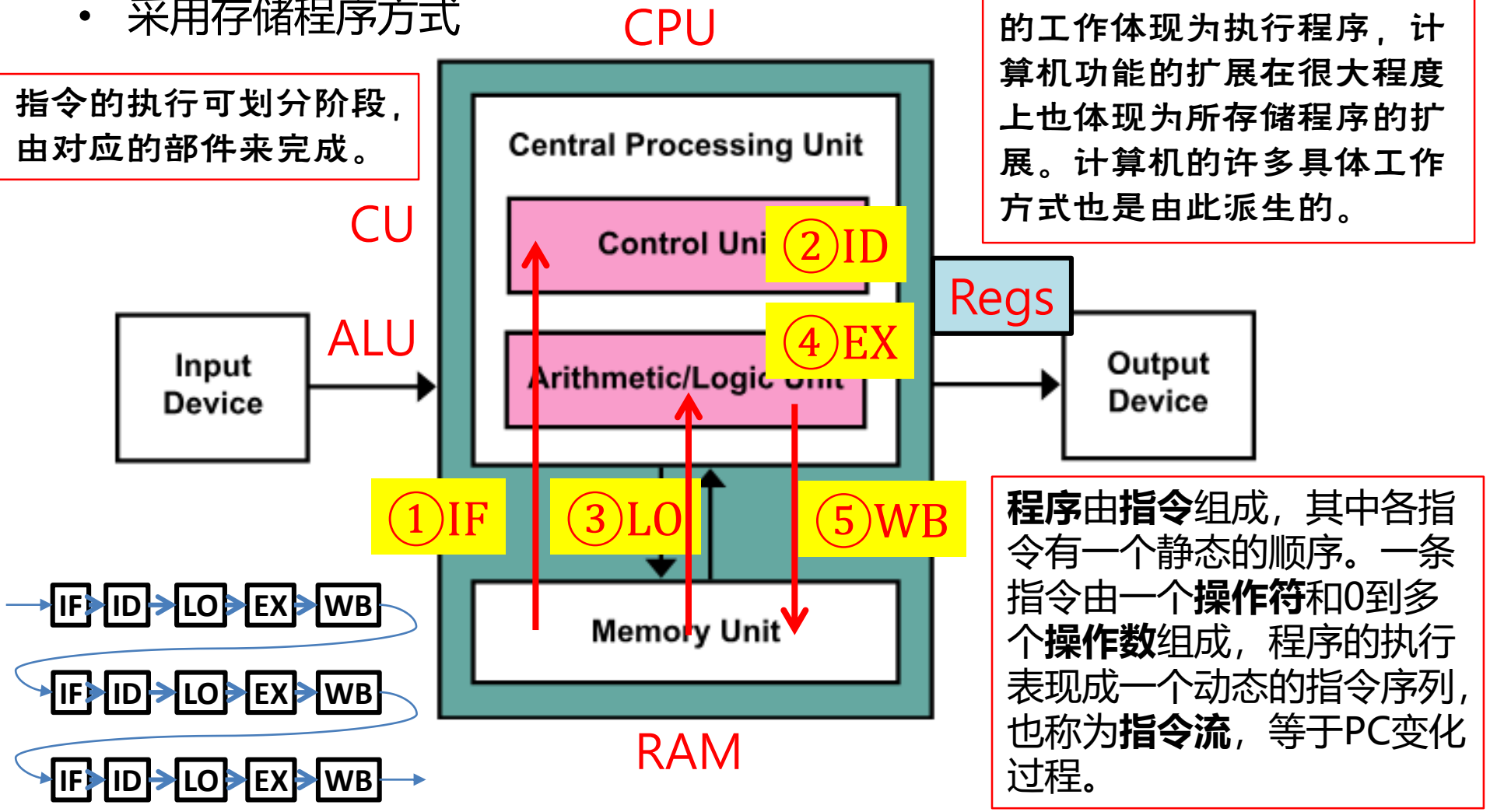
高效通用图灵机定理：存在通用图灵机 U ，使得对于任意的 $\alpha,x\in\{0,1\}^*$ ，均有 $U(\alpha,x)=M_\alpha(x)$ 。而且，如果 $M_\alpha(x)$ 的时间函数是T，则 $U(\alpha,x)$ 的时间函数是 $O(|\alpha|^c T\log T)$ ，其中c是一个与 α,x 无关的常数。

冯·诺依曼结构

- 采用二进制形式表示数据和指令
- 采用存储程序方式

意味着事先编制程序，事先将程序(包含指令和数据)存入内存，运行时自动地、连续地从存储器中依次取出指令且执行。这是计算机能高速自动运行的基础。计算机的工作体现为执行程序，计算机功能的扩展在很大程度上也体现为所存储程序的扩展。计算机的许多具体工作方式也是由此派生的。

指令的执行可划分阶段，由对应的部件来完成。



程序由**指令**组成，其中各指令有一个静态的顺序。一条指令由一个**操作符**和0到多个**操作数**组成，程序的执行表现成一个动态的指令序列，也称为**指令流**，等于PC变化过程。

编程语言用于构建计算系统

- 计算系统的基础是图灵机和冯诺依曼体系结构；
  - 图灵机►通用图灵机►冯诺依曼结构
- 编程语言是专门设计用来表达计算过程的语言，分层适应于计算系统的构建；
- 编译程序的作用就是降低计算系统构建的复杂度。

问题3：本课程是什么？

- 可参看思源学堂课程介绍
  - 课程名称由来
  - 课程简介
  - 课程目标
  - 毕业要求
  - 知识点

各章安排

- 第1~3、5、7章介绍II、III型形式语言理论，描绘统一视觉下的语言识别器、判定性质及等价性质。
- 第4、6、8章介绍词法、语法分析，描绘语言识别器的可分解、可组合、可转换原理。
- 第9、10章包含属性文法、符号表、中间语言和SLR(1)引导的语义分析方法，描绘形式与内涵一致的、可符号化的语义分析原理。
- 第11章介绍运行时环境和函数的可执行代码，描绘最基本的可执行代码框架及构建方法。

课程目标

- 课程目标1：
  - 掌握形式语言理论的基本概念和基本方法，包括正则语言、有穷自动机、正则表达式、上下文无关语言、上下文无关文法、下推自动机等，能综合应用于形式语言的判定和分析。特征：偏理论的知识点，应用于程序设计语言工程
- 课程目标2：
  - 能够运用形式语言识别器原理、语法制导的语义分析原理、基于属性文法的中间代码生成原理、函数调用栈原理等，识别、表达文法歧义性、过程非确定性、运行时有效性等相关的编译问题，获得有用结论。特征：应用所列原理识别表达程序设计语言工程中的所列问题
- 课程目标3：
  - 能设计词法分析框架、语法分析框架和语义分析框架，能构建函数的运行时环境、设计函数的可执行代码框架等。特征：针对文法能进行所列框架的部分或全部设计，产生正确结果

教学安排

- 综合成绩评定
  - 平时成绩 30%，包括作业、课堂表现、到课情况
  - 期末考试 70%
- 授课方式
  - 线下：见课表安排
  - 思源学堂：师生对接处，对接课程目标、课件、课程资料等信息
  - 雨课堂（待定）：发布作业、公告等，同学提交作业
  - 微信群：常规联系处，通知、公告、反馈、集体答疑等
  - 微信私信：具体问题讨论、答疑等
  - 另外有任何联系需要都可发邮箱zhaoy@xjtu.edu.cn

关于课程的作业

- 完成老师即时布置的作业，预计总共7次左右。
- 注意，讲授内容和作业中含MIPS指令系统与ARM a32指令系统，同学只须二选一完成相关作业即可。

教材及参考书

- 赵银亮 形式语言与编译，西安交通大学，2025年，ISBN 978-7-5693-2653-6

- John E. Hopcroft等. Introduction to Automata Theory, Languages, and Computation (3rd Ed.中文版，孙家骕等译)，机械工业出版社，2008。
- Torben Ægidius Mogensen. Introduction to Compiler Design (2nd edition). Springer International Publishing AG, 2017.
- 龙书： Alfred V. Aho,et.al.，赵建华等译. 编译原理（第二版），机械工业出版社，2009。

先修课程

- 为了最好地修读本课程：
  - 读者应当学习过关于“离散数学”、“数据结构”和“程序设计”课程；
  - 熟悉集合论与代数系统，熟悉常见数据结构如序列、表、树、图等，熟悉C语言并具有一定的程序设计经验；
  - 熟悉数学归纳法和逻辑推理等定理证明技巧；
  - 若对于Pascal、Fortran和Lisp语言有初步了解会更有利于学习。
- 以上的都不是必要条件。

1.5 形式语言概论

- 什么是形式语言？
- 形式文法
- 有穷自动机

什么是形式语言？

- 自然语言：人类交流所用语言
- 人工语言：人类有目的地构建一些communication systems和语言，用于非常专门的任务。
  - 公元前三百年Aristotle的命题逻辑
  - 十一世纪圭多·达·阿列佐发明音乐记谱法（四线谱）
  - 人机交互语言：计算的、做文档的、搜索DB或Web的、控制机器人的等
  - 物-物接口：如PDF
- 任意被设计成的语言都是人工的，有其定义。但不是所有人工语言都是形式化的。
  - 比如Java是形式化的，而世界语是由人设计的但不是形式化的。

什么是形式语言？

- 任意被设计成的语言都是人工的，有定义。但不是所有人工语言都是形式化的，比如Java是形式化的，而世界语是由人设计的但不是形式化的。
- 基因（DNA）语言是科学家发明的，用于自然生物系统，有人考虑用形式语言的方法来描述。Searls DB(2002) The language of genes. Nature 420:211-217

- **对于形式语言，其句子的语法和语义必须是精确定义的，是基于算法的定义。**
  - 换句话说，应该能够由一台计算机来检查它的所有句子都是语法上正确的，并且能得出它们的语义含义
  - 限定范围：最基本的形式语言理论只精确描述句子的词法和语法，而对句子的语义通过等价转换来刻画，转换为另一语言的句子。

什么是形式语言？

- 以重写规则 $\alpha \rightarrow \beta$ 的形式表示形式语言，其中 α 与 β 均为符号串。
- 从一个初始的符号串开始，按照不同的顺序，不断应用不同的重写规则，可以得到不同的新符号串。
- 那些不可重写的新符号串全集作为该语言。
- 说这些重写规则定义了该形式语言。

形式文法

- 这类形式文法描述形式语言的基本想法是，从一个特殊的初始符号出发，不断地应用一些产生式规则，从而生成一个符号串的集合。
- 产生式规则指定了某些符号组合如何被另外一些符号组合替换。

$S \rightarrow aSb$

$S \rightarrow ba$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aababb$

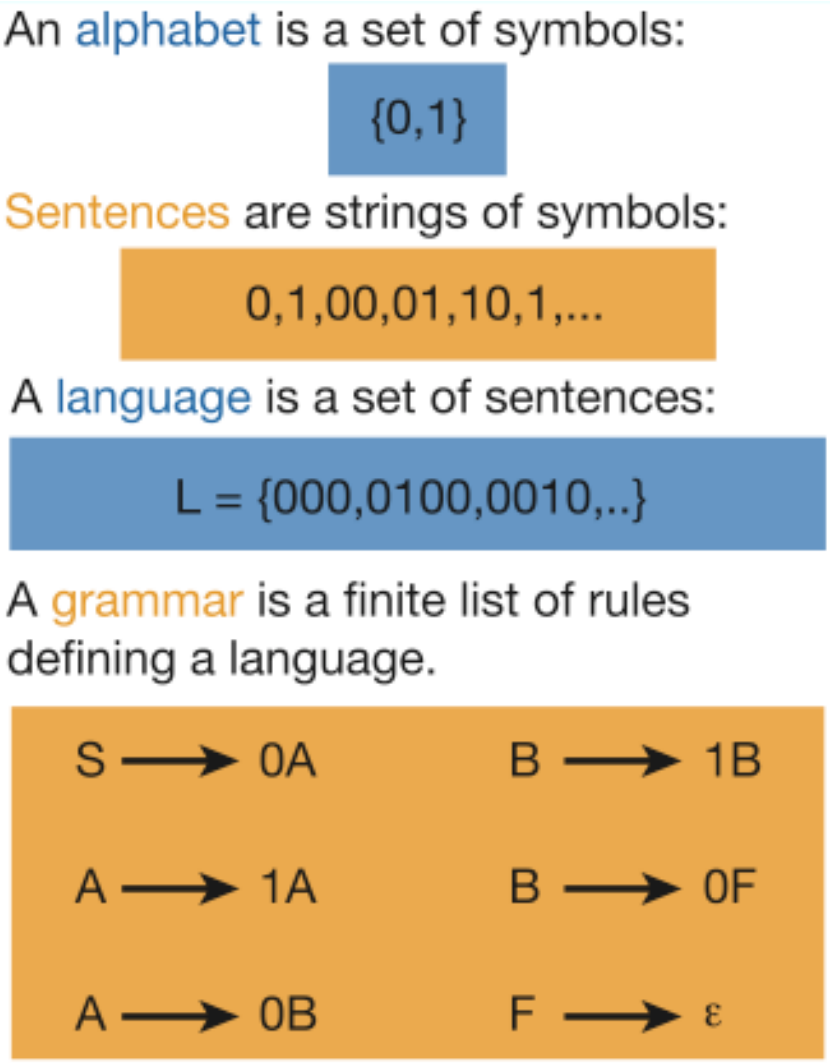
$ba, abab, aababb, aaababbb, \ldots$

形式语言与形式文法

- 在计算机科学中，形式语言是字母表上，一些有限长符号串的集合。
- 形式文法是描述这种语言的一种途径。

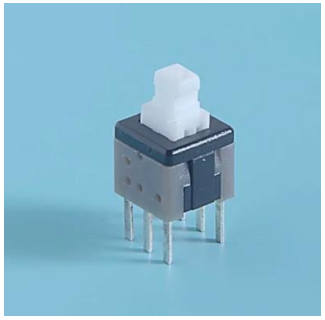
- Languages: “语言是句子的集合，句子是有穷长度的且由某个有穷字母表中的符号构成”
- Grammars: “文法可以被看做是一个装置，它恰好枚举一个语言的句子

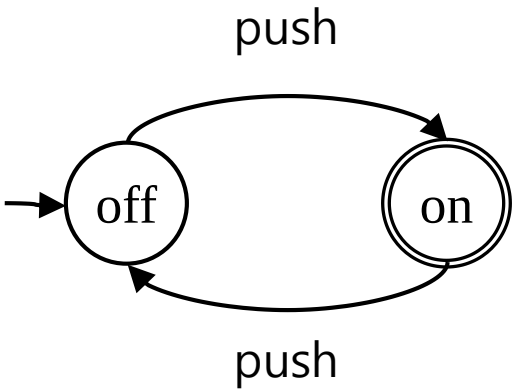
- *N. Chomsky, Information and Control, Vol 2, 1959*



有穷自动机理论

- 在理论计算机科学中，自动机理论是对抽象机器和它们能解决的问题的研究。
- 自动机是有限状态机器，通过状态之间的转移来消耗掉输入串，以判定是否为自动机所接受。
- 设想有穷自动机运行过程中从左到右逐一符号地读输入串，根据输入符号进行状态转移，一旦输入串被读完，自动机就“停止”了。
- 根据自动机停止时的状态是不是“接受”状态，就得到自动机是否接受这个符号串的结论。
- 自动机接受的符号串的全集被称为“这个自动机接受的语言”。

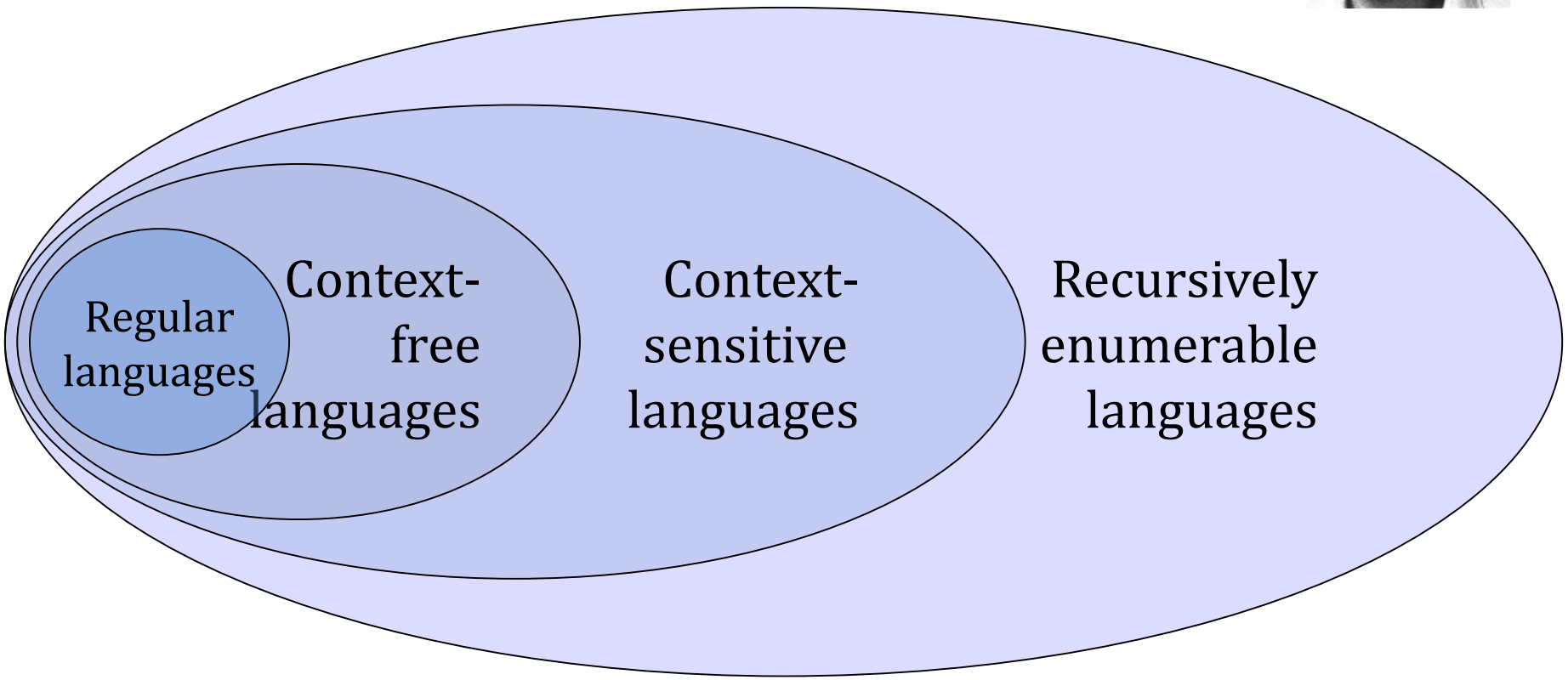




可见，有穷自动机、形式文法都能用于定义形式语言

Chomsky 体系

计算机科学中刻画形式文法表达能力的一个分类谱系，是由诺姆·乔姆斯基于1956年提出的。



DFA	Pushdown	Liner bounded	Turing machine
III型文法	II 型文法	I 型文法	0型文法
$A\to\varepsilon\|a\|cB$	$A\to\gamma$	$\alpha A\beta\to\alpha\gamma\beta$	无限制文法

一些类型的“抽象机器”

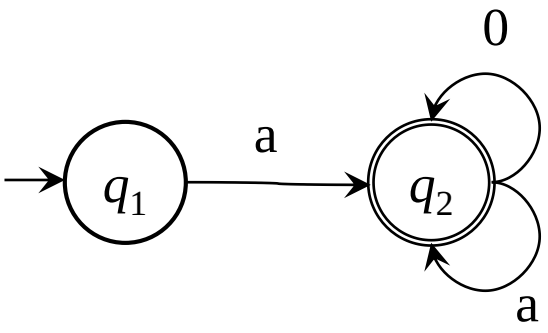
有穷自动机	有少量存储的装置 用于建模非常简单的事情	识别复杂度：线性
下推自动机	有无限量存储并以受限方式访问的装置 可用于语法分析	识别复杂度：多项式
图灵机	有无限量存储的装置 这是实际的计算机	
时间界限自动机	存储器无限大，但运行时间有界 这是以合理速度运行的计算机	a type of Turing machine with a tape of finite length, bounded by the length of the input string

历史回顾

1930s	• 图灵研究图灵机 • 可判定性 • 停机问题
1940-1950s	• “有穷自动机”机器 • Noam Chomsky提出形式语言的“乔姆斯基体系”
1969	• Cook 提出了“NP难”问题
1970-	• 现代计算机科学: 编译器, 可计算性&复杂性理论 得到发展

有穷自动机的应用

- 数字电路设计与性能检查
- 单词的模型：词法分析器



区汝就：有限自动机在数字逻辑电路设计中的应用，2020

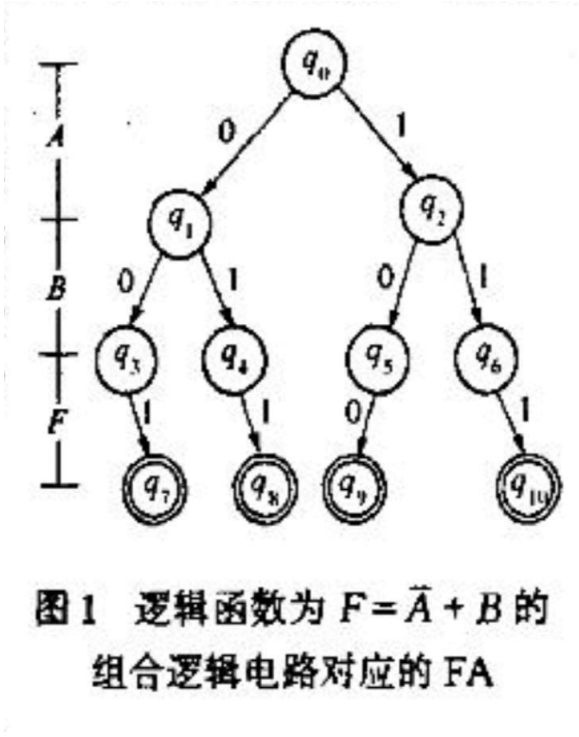


图1 逻辑函数为 $F=\bar{A}+B$ 的组合逻辑电路对应的 FA

有穷自动机的应用

- 单词、短语或其它模式：文本搜索、模式匹配

匹配Email地址：\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*

匹配网址URL：[a-zA-z]+://[^\s]*

匹配帐号是否合法(字母开头，允许5-16字节，允许字母数字下划线)：^[a-zA-Z][a-zA-Z0-9_]{4,15}$
评注：_表单验证时很实用

匹配国内电话号码：\d{3}-\d{8}|\d{4}-\d{7}
评注：匹配形式如 0511-4405222 或 021-87888822

匹配腾讯QQ号：[1-9][0-9]{4,}
评注：腾讯QQ号从10000开始

匹配中国邮政编码：[1-9]\d{5}(?!\d)
评注：中国邮政编码为6位数字

匹配ip地址：\d+\.\d+\.\d+\.\d+
评注：提取ip地址时有用

MS Word中通配符

还有DFA用于实现敏感词过滤等等。。。

有穷自动机的应用

- 对于系统中的各种约定建立模型：协议验证

  基于DFA的协议解析方法
  基于DFA的高速协议分类引擎研究

- 一些物理系统、管理系统的模型：模拟仿真

  口香糖球售货机模型
  简单购物模型
  ……

自动机、形式文法、形式语言的等价性

自动机	形式文法	形式语言
DFA	正则文法	正则语言
PDA	CFG	CFL
TM	0型文法	递归可枚举语言

- 自动机便于执行
- 形式文法便于推导、推理、生成
- 形式语言便于枚举
- 都称为语言识别器
- 本课程讨论范围为表的前两行所示

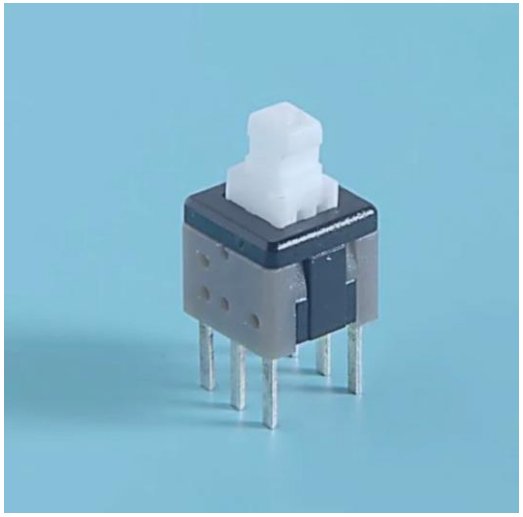
关于可计算性、复杂性理论

- **图灵机的运行时间：**
  对于某输入，图灵机运行的步数。
  - 时间有穷：从开始运行到停机所经过的步数。
  - 时间无穷：不停机。

- 可计算性：即能否为图灵机所判定（可判定性）
- 计算复杂性：对问题按难度分类，寻找对难问题的转化求解方法。

两相开关的DFA建模

建模对象：两相开关，是一个物理系统。

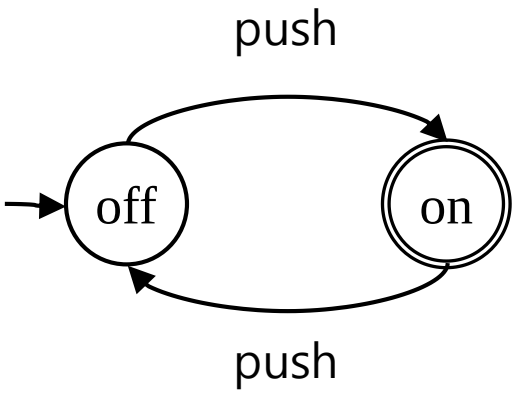


- 有“开”和“关”两个状态；
- 外界按压（push）表示在系统边界上发生的交互行为；
- 开始时开关的状态为“开”；
- 一个push导致状态“开”、“关”切换一次；
- 一到多个push之后开关的状态是“开”，则达到接通之目的。

建模结果：一个DFA

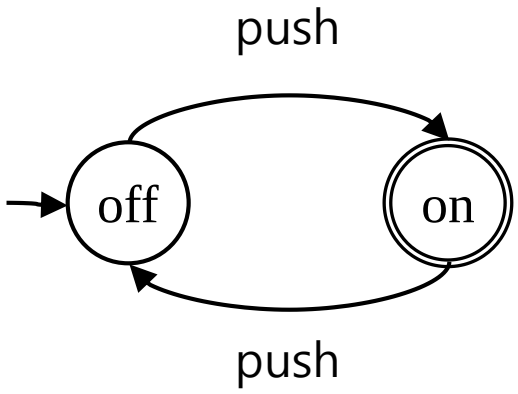
两相开关DFA

- 建模结果



- 两个状态on、off
- 初始状态off
- 接受状态on
- 状态转移弧
  (off, push) -> on
  (on, push) -> off
- 转移弧上的标记push
- 输入符号push

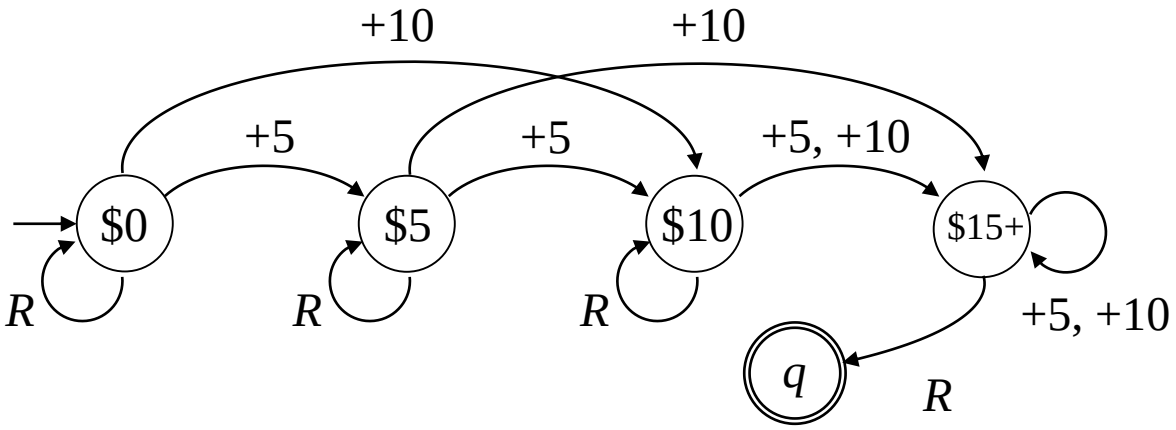
两相开关DFA运行过程



- ✓ 给定输入串，由单个符号push组成的符号串；
- ✓ 从初始状态开始，并将输入串第一个符号作为当前输入符号；
- ✓ 对于当前输入（push表示）发生状态转移；
- ✓ 当输入串消耗完自动机到达on则接受；
- ✓ 否则不接受；
- ✓ 本模型中接受表示物理系统接通，否则是断开的。

口香糖球机

- 机器接受\$5 和\$10 硬币
- 一个口香糖值\$15
- 动作: +5, +10, 释放R



1.6 本课程中心概念

- 中心概念：符号，字母表，符号串，语言，问题
- 主文法与主符号系统

符号、字母表

- 符号是什么？指代事物的手段

> a letter, figure, or other character or mark or a combination of letters or the like used to designate something.

- 符号的来源与它的指代物
- 符号作为名字
- 计算系统与它的那些有穷个的符号
- 形式化原则

符号、字母表

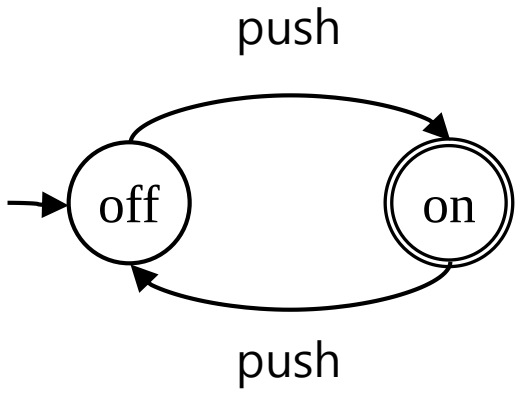
- 字母表是感兴趣的符号的非空集合有穷集合
- 感兴趣？认知域、论域、计算系统

- 二进制数字集合，$\Sigma_{\mathrm{B}}=\{0,1\}$。
- 十进制数字集合，$\Sigma_{\mathrm{D}}=\{0,1,2,3,4,5,6,7,8,9\}$。
- 所有小写拉丁字母集合，$\Sigma_a=\{a,b,\ldots,z\}$。
- 所有ASCII字符的集合。
- 空白字符的集合，$S=\{\backslash t,\backslash n,\backslash r,\backslash 0,' '\}$。
- 字母加特殊符号$构成的集合，$\Sigma_{a\$}=\Sigma_a\cup\{\$\}$。

符号串

- 符号串是构建在字母表上的最为基础的结构。
- 一个符号串是由字母表中符号组成的有穷长度的一个序列。
  - $abfbz$ 是Σ_{a}上的一个串，$\Sigma_a = \{a, b, c, d, ..., z\}$
  - 90221是Σ_{D}上的一个串，$\Sigma_{\mathrm{D}} = \{0, 1, ..., 9\}$
  - $ac\$bc$是$\Sigma_{a\$}$上的一个串，$\Sigma_{a\$} = \{a, b, ..., z, \$\}$
- 符号串的元素、元素的前驱、元素的后继、元素的索引
- w为一个符号串，$|w|$为w的长度，长度为0的符号串ε
- 函数$\pi(w, i)$为w的i前缀，$\pi(w)$为所有前缀组成的集合
- 函数$\tau(w, i)$为w的i后缀，$\tau(w)$为所有后缀组成的集合
- Σ上的所有符号串组成一个集合记为Σ^*

例子（注意到与形式化原则的冲突）



- 状态：on, off
- 初始状态：on
- 接受状态：off
- 转移弧：
  off状态射出的push弧射入on状态；
  on状态射出的push弧射入off状态；
- 转移弧上的标记：push
- 输入符号：push
- 字母表：{push}
- 输入：字母表上符号串

语言

- 长度为0的符号串是空串，习惯上采用ε表示空串。
- 字母表Σ上的符号串全集是Σ^*

- Σ上的语言是Σ上符号串的集合
- 显然，Σ上的语言是Σ^*的一个子集

L_1是Σ_a上的含有子串“to”的所有串

stop, to, toe are in L_1

e, oyster are not in L_1

$L_1 = \{x \in \Sigma_a^* \mid x$含有子串“to”$\}$

语言的例子

$L_2 = \{x \in \Sigma_D^* \mid x \text{ 被7整除}\}$

$= \{7, 14, 21, \ldots\}$

$L_3 = \{s\$s \mid s \in \Sigma_a^*\}$

$ab\$ab$ in L_3

$ab\$ba$ not in L_3

$a\$\$a\$$ not in L_3因为$a\$$不属于Σ_a^*

什么是问题？

- 要求解的问题都是判定性的问题

给定词s, 该词包含“to”子串吗?
给定整数n, 它能被7整除吗?
给定符号串s 和 t, 它们相同吗?

假定字母表是Σ_1，问题即 $s \in L_1$?

假定字母表是Σ_2，问题即 $n \in L_2$?

问题即$|s|==|t|$且 s、\$与t依次连接起来是否属于L_3？

问题即 $s\$t \in \{x\$x \mid x \in \Sigma^*, \$ \notin \Sigma\} = L_3$？

判定性问题的答案为是或否。

任何其它问题如查找、统计等都可以等价转化为判定性问题。

关于形式化证明

- 形式化证明很有用，在后续课程中用起来
  - 演绎证明
  - 归纳证明：数学归纳法
  - 反证法
- 鸽巢原理
- 直接运用先修课的知识

数学归纳法

Example

Prove that $1^2 + 2^2 + 3^2 + \ldots + n^2$ is $\frac{n(n+1)(2n+1)}{6}$.

- **Basis (n=1):** $\frac{1(1+1)(2\cdot 1+1)}{6} = \frac{2\cdot 3}{6} = 1 = 1^2$.
- **Induction:** Let $\mathcal{P}(n) := 1^2 + \ldots + n^2$ is $\frac{n(n+1)(2n+1)}{6}$.

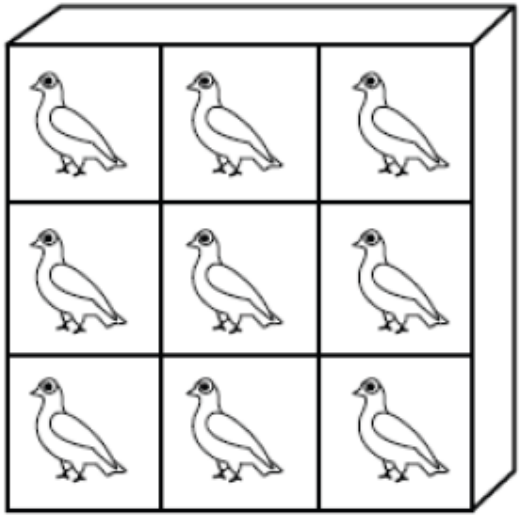
  $\mathcal{P}(n+1) = 1^2 + \ldots + n^2 + (n+1)^2 = \frac{n(n+1)(2n+1)}{6} + (n+1)^2$

  $= (n+1)\left\{\frac{n(2n+1)+6n+6}{6}\right\} = (n+1)\left\{\frac{2n^2+7n+6}{6}\right\}$

  $= (n+1)\left\{\frac{2n^2+4n+3n+6}{6}\right\} = \frac{(n+1)(n+2)(2n+3)}{6}$
- Since $\mathcal{P}(1)$ is true and $\mathcal{P}(n) \Rightarrow \mathcal{P}(n+1)$, we conclude that $\mathcal{P}(n)$ is true for all $n \geq 1$.

鸽巢原理

如果有n+1个鸽子和n个鸽巢，那么某个鸽巢必须至少住进两只鸽子。



如果有kn+1个鸽子和n个鸽巢，那么某个鸽巢必须至少住进k+1只鸽子，k>=1.

Pigeonhole Principle

If there are $n+1$ pigeons and n pigeonholes, then some pigeonhole must contain at least two pigeons.

其它预备知识

- 图论
  - 有向图、无向图
  - 结点的度、入度、出度
  - 路径、路径长度、环
- 集合论
  - 集合运算及性质
  - 关系、映射、函数
  - 划分、覆盖
- 数据结构
  - 线性表、树、杂凑表、链表
  - 操作、算法及其复杂度

小结与作业

- 知识点：编译相关术语，编译过程，图灵机模型，有穷自动机构成及原理，形式文法概念，符号、字母表、符号串、语言、问题等概念。

- 1.7习题：1.3、1.5(2)、1.7(2)
- 阅读：教材第一章