

<https://github.com/lazyparser/becoming-a-compiler-engineer>

方舟·编译技术入门与实战

第二课：词法分析

吴伟 (@lazyparser)

2019-12-12

错过直播没关系，所有直播都有回看

- 课程配套代码及幻灯片地址（也是提问的地方）
 - <https://github.com/lazyparser/becoming-a-compiler-engineer>
 - <https://github.com/lazyparser/becoming-a-compiler-engineer-codes>
- 课程视频回看（包含所有直播及录播视频）
 - <https://space.bilibili.com/296494084>
- 课程直播地址（可以弹幕或评论区互动）
 - <https://live.bilibili.com/10339607>

本次课程将介绍

- 第一节课课后作业的复习（包含一些元学习理论知识）
- 词法分析：知识地图、算法、可视化工具（很好玩）
- 几个必要的数学概念的补充（不要怕）
- 前端编程练习用语言PL/0的准备

你会使用到的资源 (课后阅读一定要完成)

- <https://www.clear.rice.edu/comp412/> Keith Cooper 老师
- <https://courses.cs.washington.edu/courses/csep501/08wi/> Hal Perkins 老师
- <https://www.bilibili.com/video/av9946063> 冯博琴老师
- <https://www.bilibili.com/video/av59119931> 王挺老师



COMP 412: Compiler Construction for Undergraduates
Keith Cooper
Michael Burke
Department of Computer Science, Rice University
Houston, Texas, USA
Fall 2019: HRZ 212 on Monday, Wednesday, and Friday at 11:00am



University of Washington
Computer Science & Engineering
CSEP501, Winter 2008 Lecture Slides and Video Archives



上周课后作业的复习

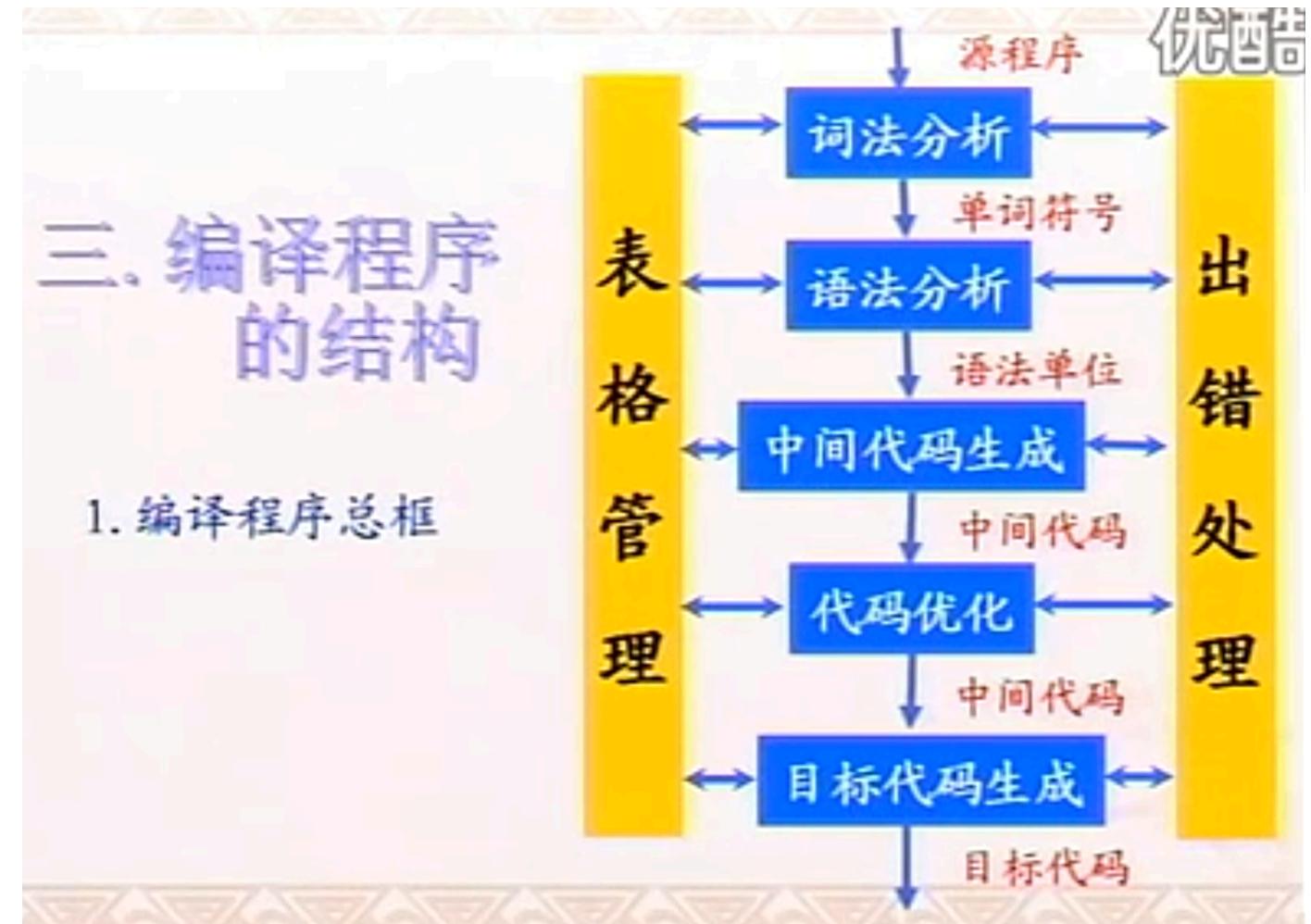
- 引论部分的自学资料（请在下次课程前自学）：

- <https://www.bilibili.com/video/av59119931?p=1>
- <https://www.bilibili.com/video/av59119931?p=2>
- <https://www.bilibili.com/video/av9946063?p=1>
- <https://www.bilibili.com/video/av9946063?p=2>



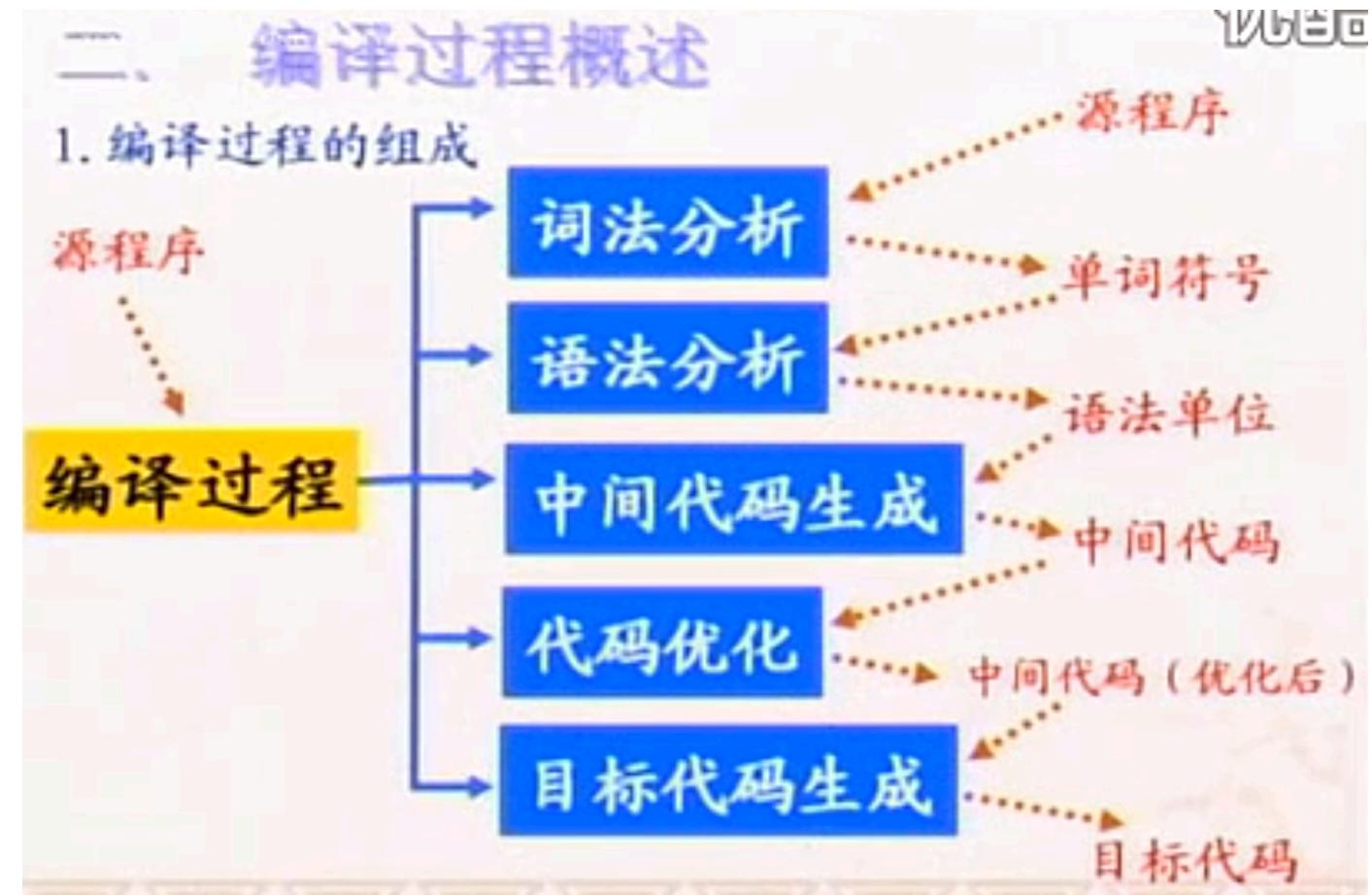
上周的课后阅读，建立了整体概念

- 以冯博琴老师的公开课为例



上周的课后阅读，建立了整体概念

- 以冯博琴老师的公开课为例



上周的课后阅读，建立了整体概念

- 以冯博琴老师的公开课为例：很好的动态演示

2. 词法分析

示例

FOR K := 1 TO 100

M := I + 10 * K

N := J + 10 * K

NEXT K

2. 词法分析

示例

定义符

FOR K := 1 TO 100

标识符

M := I + 10 * K

分界符

N := J + 10 * K

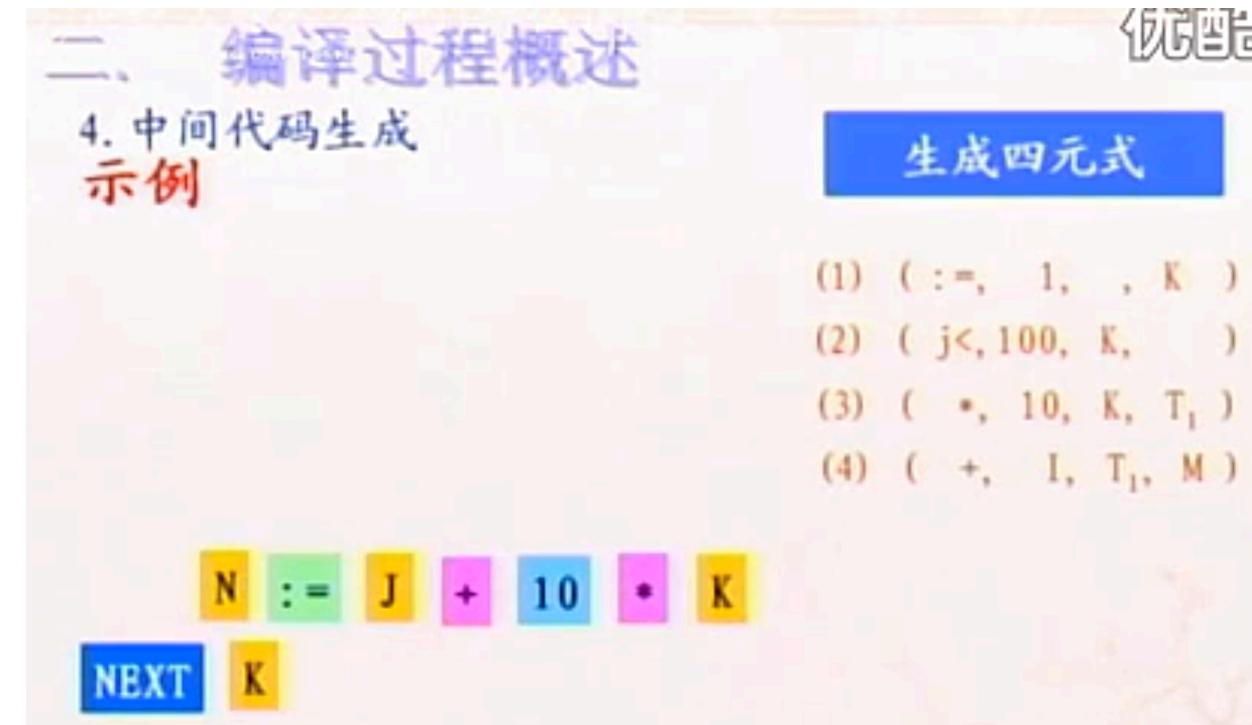
运算符

常数

NEXT K

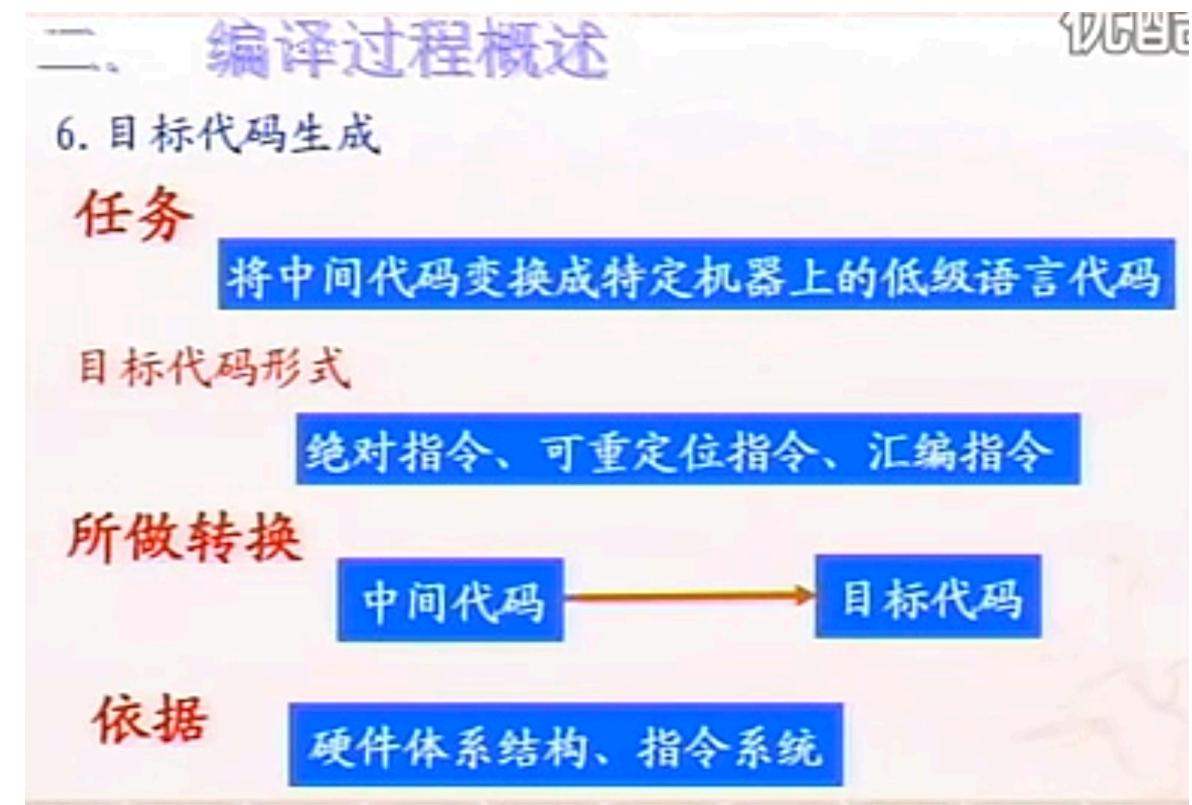
上周的课后阅读，建立了整体概念

- 以冯博琴老师的公开课为例：很好的动态演示



上周的课后阅读，建立了整体概念

- 以冯博琴老师的公开课为例



四个老师，为什么第一课都是总体介绍？

- 不管是国内还是国外的编译老师，都先讲编译器的总体架构
- 进一步的，每个领域的课程系列，第一课都是 big picture

(还没完成上周课后阅读作业的同学请本周完成)

学习的方法：建立自己的知识地图

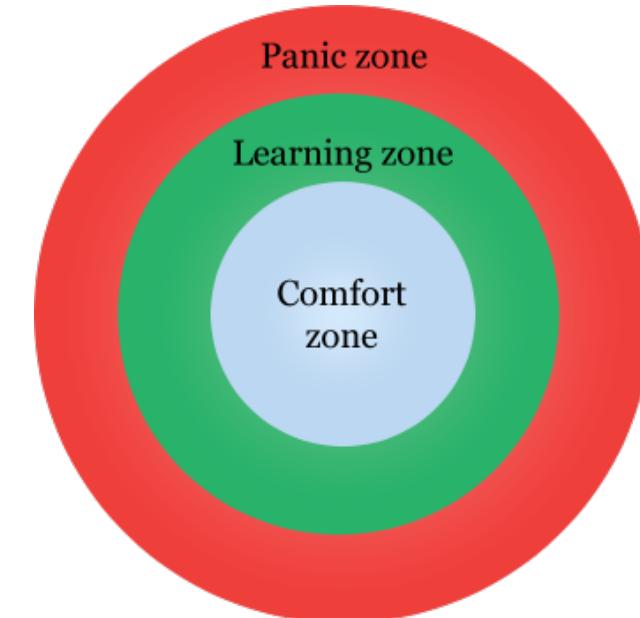
- 这不是某位授课老师的发明创意；这是教育和认知科学的成果
- 进入新的领域，最有效率的方法之一是快速建立知识地图
- 在熟悉的领域内工作，可以看成不断扩展地图边界的过程

对于成年人而言，掌握学习方法论尤为重要

- 本课程的学习者，大多已经成年，有着各类条件约束
 - 有限的时间、有限的注意力、有限的体力
 - 多线任务，有时候不太合乎情理的项目交付 Deadline
- 学会如何快速学习是我作为新人培训人员希望新人达到的目標
 - 学习的方法论能够让我们在尽可能短时间内高质量地获得知识能力

知识地图的作用

- 增加面对未知情况做决策时的自信，缩小风险的概率区间
- 有计划的自我激励，避免从学习区进入恐慌区（Panic Zone）
- 认清自己的工作是工程还是研究
 - SSR中的R一般是在人类知识地图边界处



https://commons.wikimedia.org/wiki/File:Panic_Learning_and_Comfort_Zones.png

看英文资料时候的注意事项（不要怕/逃）

- 重要的心理门槛：念出单词正确的发音（价值百万的培训点:P）
- 术语直接随手查词典（从下周开始基本上都是英文资料了）
 - 大量在线免费词典，例如 <https://dict.youdao.com>
- 人名不会念很正常
 - 发音不一定是英语，要问；初次见面一定要问对方
- 公司组内或公开报告前确保查询过所有出现单词
 - 否则现场时候心态很可能因为单词发音不确定紧张到崩

本次课程将介绍

- ~~第一节课课后作业的复习（包含一些元学习理论知识）~~
- 词法分析：知识地图、算法、可视化工具（很好玩）
- 几个必要的数学概念的补充（不要怕）
- 前端编程练习用语言PL/0的准备

词法分析：自学 Cooper 教授课件的 1 – 8

- <https://www.clear.rice.edu/comp412/Lectures/>

Slides and/or Notes

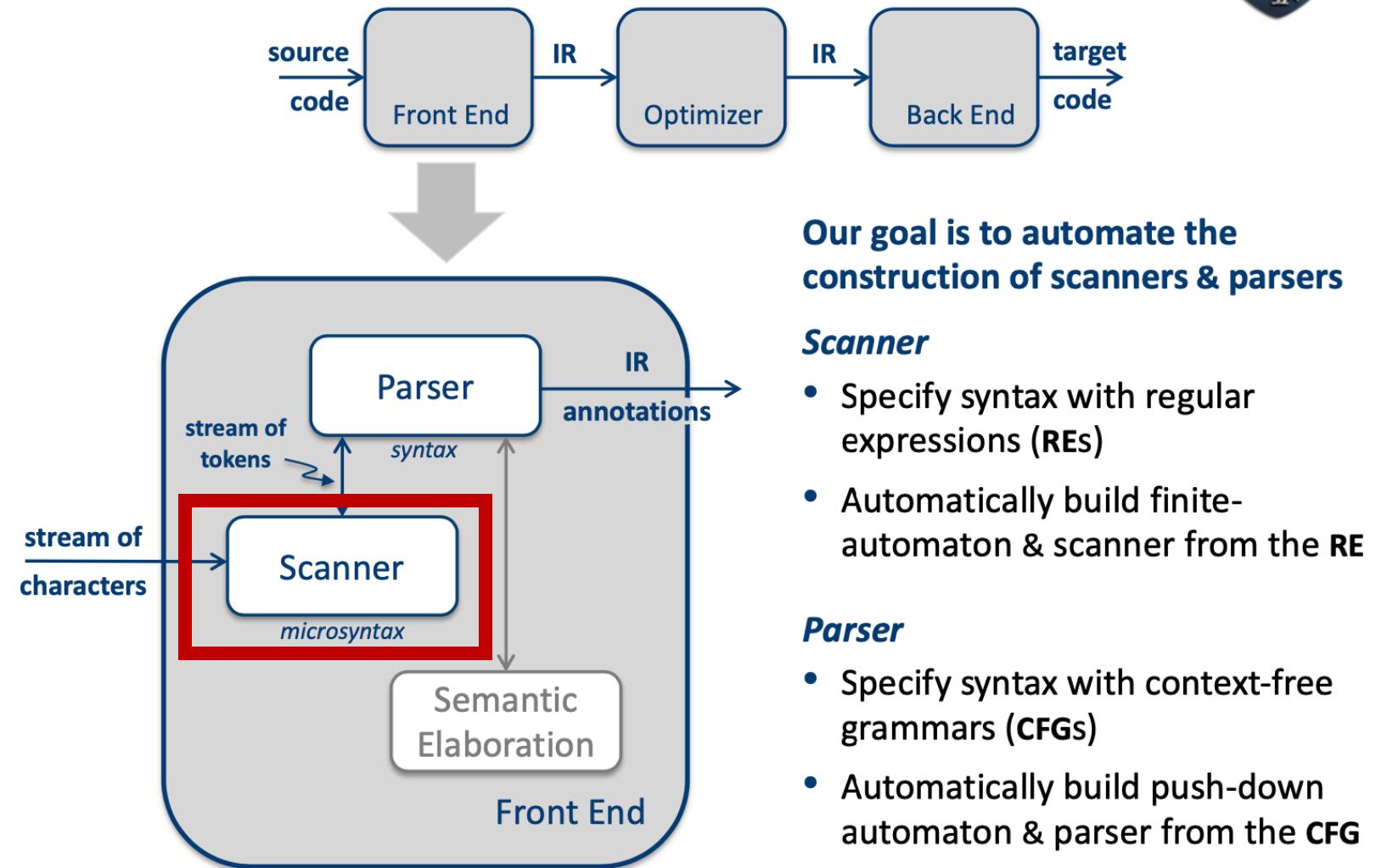
1. [Introduction and First Content](#) [Blog post on Cloudflare outage](#) (see lecture notes)
2. [Building a Scanner](#), from a lab 1 perspective
3. [Building a Simple Parser](#), from a lab 1 perspective (corrected)
4. [Lexical Analysis, I](#) Chapter 2 in EaC2e
5. [Lexical Analysis, II](#)
Ken Thompson, Regular Expression Search Algorithm, CACM, June 1968 [DOI](#)
Michael Rabin and Dana Scott, "Finite Automata and Their Decision Problems," IBM Journal of Research and Development, 3(2), April 1959. [DOI](#)
6. [Lexical Analysis, III](#) DFA Minimization
[Excerpt from EaC3e](#) on Hopcroft's algorithm skip this part in EaC2e
7. [Lexical Analysis, IV](#) Building a Scanner from a DFA
8. [Another way to approach Kleene's construction](#) (Regular Expression from NFA)
9. [Parsing, I](#) Chapter 3 in EaC2e



词法分析

- 复习：在编译器整体架构中的位置

Implementation Strategies



COMP 412, Fall 2019

<https://www.clear.rice.edu/comp412/Lectures/>

1



记住这一页

- 记住了，理论部分就全部都不会忘了

The Plan for Scanner Construction

$\text{RE} \rightarrow \text{NFA}$ (*Thompson's construction*)

- Build a *nondeterministic finite automaton* (**NFA**) for each term in the **RE**
- Combine them in patterns that model the operators

$\text{NFA} \rightarrow \text{DFA}$ (*Subset construction*)

- Build a **DFA** that simulates the **NFA**

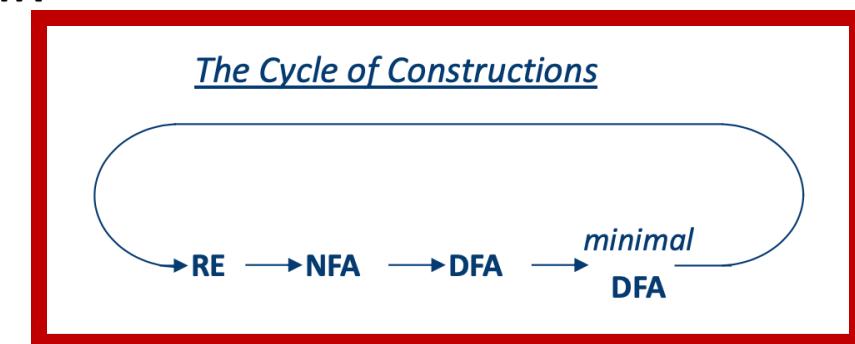
$\text{DFA} \rightarrow \text{Minimal DFA}$

- Brzozowski's algorithm
- Hopcroft's algorithm

$\text{DFA} \rightarrow \text{RE}$

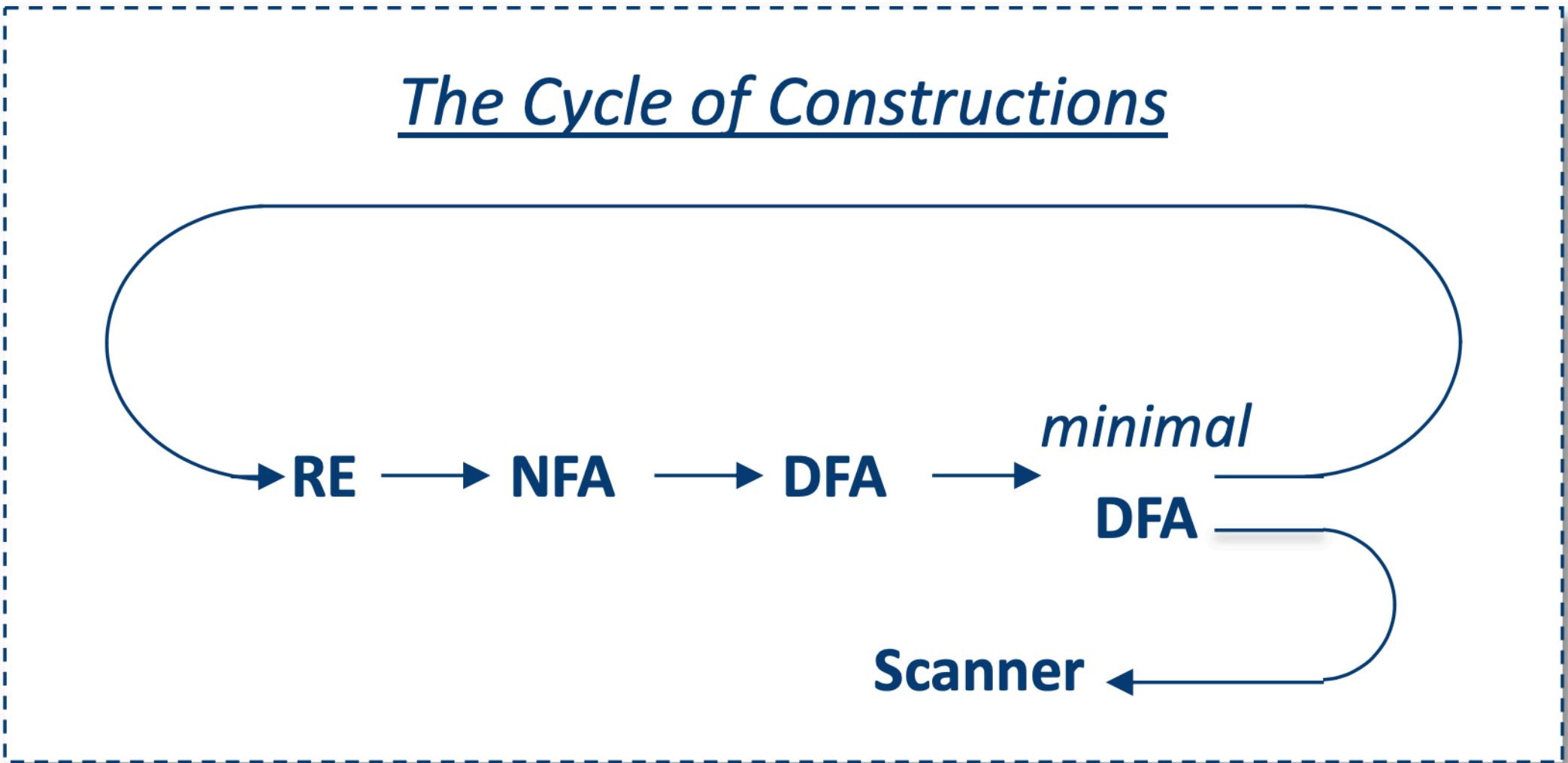
- All pairs, all paths problem
- Union together paths from s_0 to a final state

The Cycle of Constructions



Taken together, these constructions prove that DFAs, NFAs and REs are equivalent.

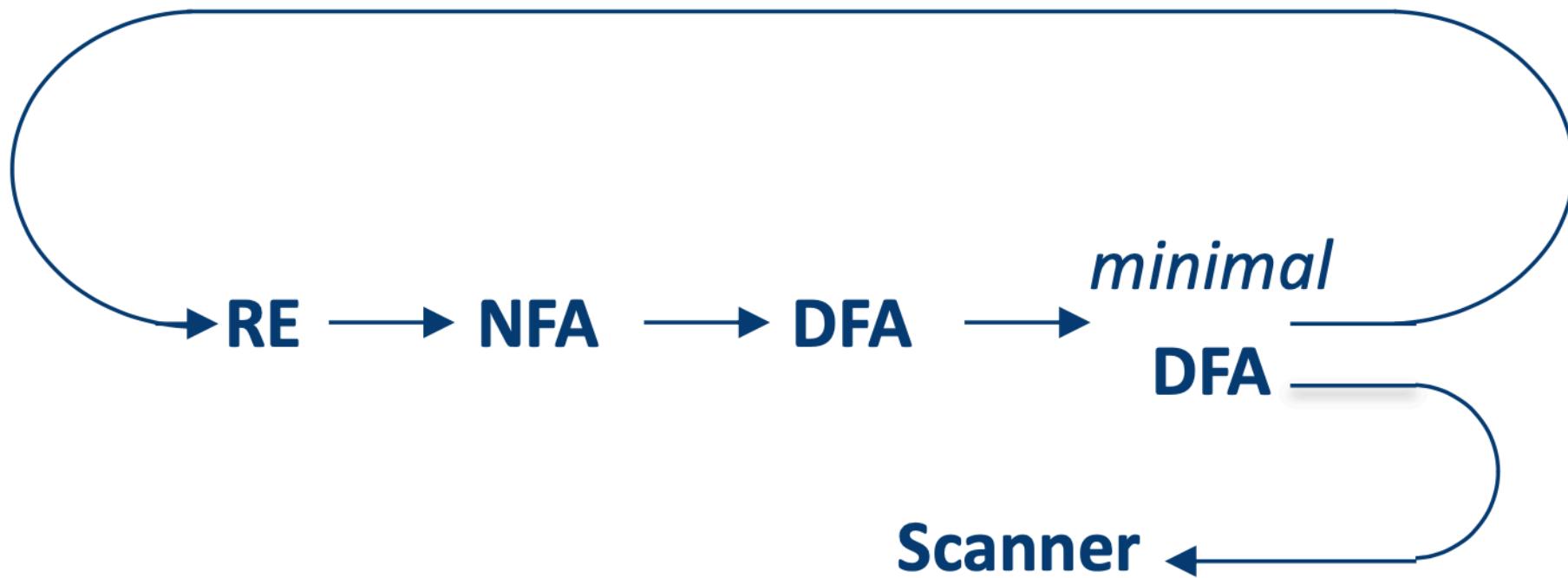
有了这张图，就不会忘记了



<https://www.clear.rice.edu/comp412/Lectures/>

提问：有 minNFA 么？为什么没画？😊

The Cycle of Constructions



<https://www.clear.rice.edu/comp412/Lectures/>



记住这张图

- 作业：查如何发音
 - nondeterministic
 - finite
 - automaton
 - paths
 - final
 - Brzozowski
 - Hopcroft
 - union

The Plan for Scanner Construction

RE → NFA (Thompson's construction)

- Build a nondeterministic finite automaton (NFA) for each term in the RE
- Combine them in patterns that model the operators

NFA → DFA (Subset construction)

- Build a **DFA** that simulates the **NFA**

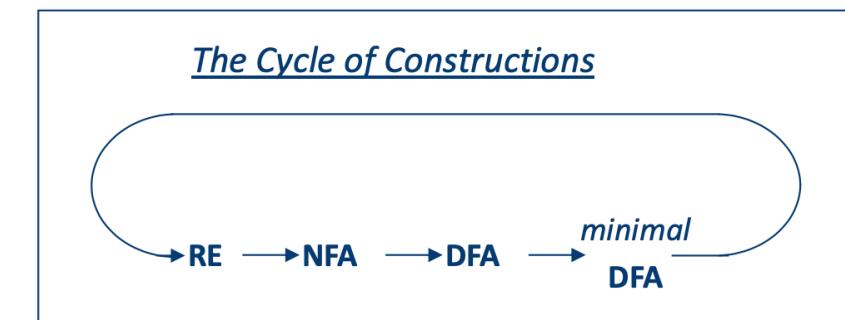
DFA → Minimal DFA

- Brzozowski's algorithm
- Hopcroft's algorithm

DFA → RE

- All pairs paths problem
- Union together paths from s_0 to final state

The Cycle of Constructions



Taken together, these constructions prove that DFAs, NFAs and REs are equivalent.

COMP 412, Fall 2019

2

STOP!

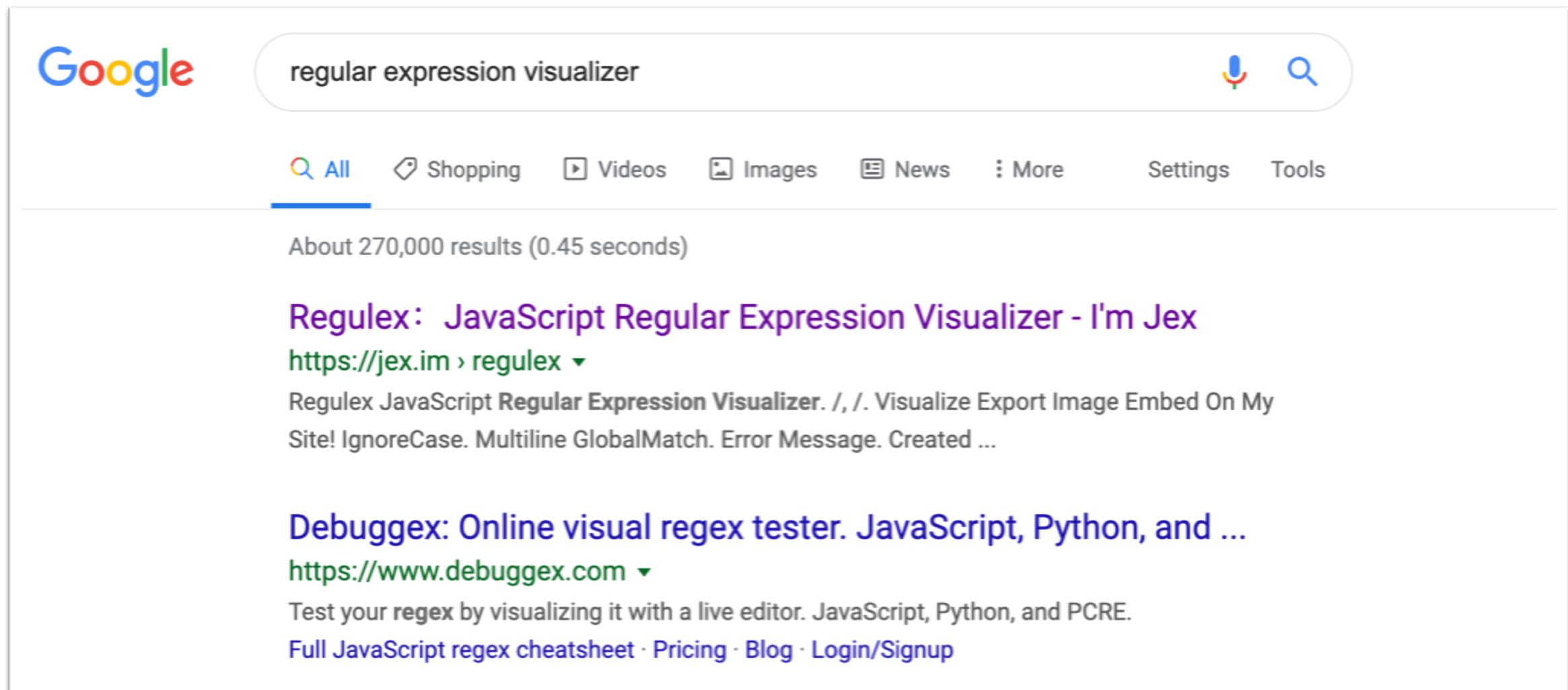
词法分析几个算法的掌握程度

- 要求掌握正则表达式的概念和写法
 - 不同的语言/工具可能有不同的表示法，需要注意细节
- 能够手工构造 RE to NFA、NFA to DFA
 - 能够理解 DFA to min DFA 的算法思想

词法分析几个算法的掌握程度

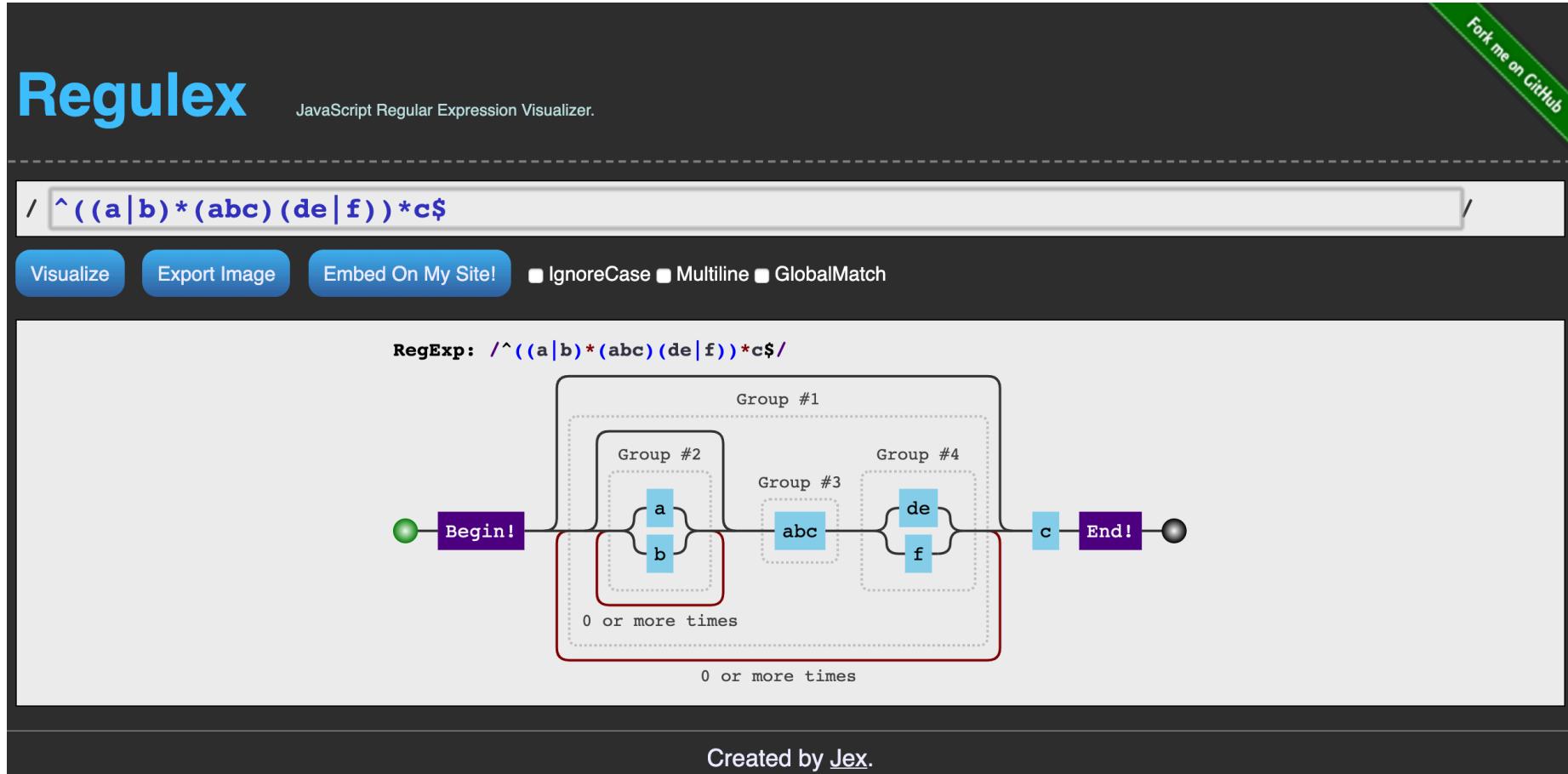
- 要求掌握正则表达式的概念和写法
 - 不同的语言/工具可能有不同的表示法，需要注意细节
- 能够手工构造 RE to NFA、NFA to DFA
 - 能够理解 DFA to min DFA 的算法思想
- 其它算法实现细节及练习，以后工作中用到的时候再回来做
 - 知识地图：你知道算法在这里，有开源实现，知道自己能重新实现

使用工具：正确性验证、可视化



A screenshot of a Google search results page. The search query "regular expression visualizer" is entered in the search bar. Below the search bar, the "All" filter is selected, along with Shopping, Videos, Images, News, More, Settings, and Tools. The search results indicate approximately 270,000 results found in 0.45 seconds. The top result is a link to "Regulex: JavaScript Regular Expression Visualizer - I'm Jex" with the URL <https://jex.im>. The description for this result mentions "Regulex JavaScript Regular Expression Visualizer. /, /. Visualize Export Image Embed On My Site! IgnoreCase. Multiline GlobalMatch. Error Message. Created ...". The second result is a link to "Debuggex: Online visual regex tester. JavaScript, Python, and ..." with the URL <https://www.debuggex.com>. The description for this result mentions "Test your regex by visualizing it with a live editor. JavaScript, Python, and PCRE. Full JavaScript regex cheatsheet · Pricing · Blog · Login/Signup".

<https://jex.im/regulex>



<https://cyberzhg.github.io/toolbox/regex2nfa>

Toolbox Home Repository Issues build passing issues 4 open pull requests 1 open contributors 1 Star 220 Follow 843

Input: $(alb)^*abb(alb)^*$

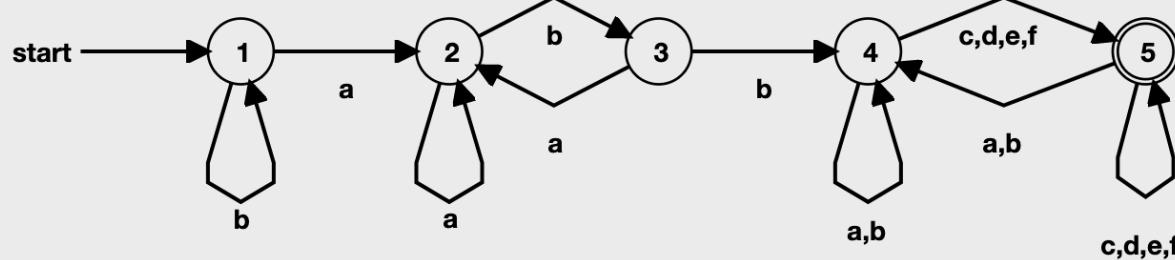
DFA: <https://cyberzhg.github.io/toolbox/nfa2dfa?regex=KGF8YikqYWJkGFG8Yikq>

CONVERT

```
graph LR; start((start)) -- "" --> 0((0)); 0 -- "ε" --> 1((1)); 1 -- "ε" --> 0; 1 -- "ε" --> 2((2)); 1 -- "ε" --> 4((4)); 2 -- "a" --> 3((3)); 4 -- "b" --> 5((5)); 6((6)) -- "ε" --> 7((7)); 7 -- "a" --> 8((8)); 7 -- "b" --> 10((10)); 10 -- "ε" --> 11((11)); 10 -- "ε" --> 12((12)); 10 -- "ε" --> 13((13)); 10 -- "a" --> 13; 10 -- "b" --> 15((15)); 11 -- "ε" --> 12; 11 -- "ε" --> 14((14)); 11 -- "ε" --> 16((16)); 16 -- "ε" --> 17((17))
```

https://cyberzhg.github.io/toolbox/min_dfa

DFA STATE	Min-DFA STATE	TYPE	a	a,b	b	c,d,e,f
{A,C}	1		2		1	
{B}	2		2		3	
{D}	3		2		4	
{E,F,G,L,M,N,S}	4			4		5
{H,I,J,K,O,P,Q,R}	5	accept		4		5



<https://cyberzhg.github.io/toolbox/nfa2dfa>

NFA STATE	DFA STATE	TYPE	a	b
{0,1,2,4,7}	A		B	C
{1,2,3,4,6,7,8}	B		B	D
{1,2,4,5,6,7}	C		B	C
{1,2,4,5,6,7,9}	D		B	E
{1,2,4,5,6,7,10,11,12,14,17}	E	accept	F	G
{1,2,3,4,6,7,8,11,12,13,14,16,17}	F	accept	F	H
{1,2,4,5,6,7,11,12,14,15,16,17}	G	accept	F	G
{1,2,4,5,6,7,9,11,12,14,15,16,17}	H	accept	F	I
{1,2,4,5,6,7,10,11,12,14,15,16,17}	I	accept	F	G

The diagram illustrates the conversion of an NFA to a DFA. At the top, an NFA state transition table maps sets of NFA states to DFA states. Below the table, the NFA is shown as a graph with states A through I. State A is the start state, indicated by an incoming arrow labeled 'start'. Transitions are labeled with symbols 'a' or 'b' and may include self-loops or multiple transitions between states. The DFA at the bottom shows states A through I, where each state corresponds to a specific NFA state set from the table. State E is the accept state, indicated by a double circle. Transitions in the DFA also follow the same labeling scheme as the NFA.

<https://cyberzhg.github.io/toolbox>

Toolbox Home Repository Issues build passing issues 4 open pull requests 1 open contributors 1 Star 220 Follow 843

Introduction

Convert simple regular expressions to minimum deterministic finite automaton.
(Regex => NFA => DFA => Min-DFA)

Supported grammars

- $r = (s)$
- $r = st$
- $r = slt$
- $r = s^*$
- $r = s^+$
- $r = s^?$
- $r = \epsilon$

(Copy this character to input if needed)

Examples

- $(alb)^*$
- $(a^*lb^*)^*$
- $((\epsilon la)b^*)^*$
- $(alb)^*abb(alb)^*$

Input: `(alb)^*abb((alb)^*(clidlef))+`

CONVERT

<https://regexr.com/>

The screenshot shows the Regexr interface. The top navigation bar includes a title 'Untitled Pattern', a 'Save' button, and a 'New' button. On the right, there are links for 'by gskinner', 'GitHub', and 'Sign In'. The left sidebar contains a 'Menu' section with 'Pattern Settings', 'My Patterns', 'Cheatsheet', 'RegEx Reference', 'Community Patterns', and 'Help'. Below this, a descriptive text block states: 'RegExr is an online tool to **learn, build, & test** Regular Expressions (RegEx / RegExp).'. A bulleted list details features: 'Supports **JavaScript & PHP/PCRE** RegEx.', 'Results update in **real-time** as you type.', 'Roll over a match or expression for details.', 'Save & share expressions with others.', 'Use **Tools** to explore your results.', 'Full **RegEx Reference** with help & examples.', and 'Undo & Redo with cmd-Z / Y in editors.'.

The main workspace is titled 'Expression' and contains the regular expression pattern: `/[+|-]?\d+/g`. This pattern is highlighted with a red box. Below the expression, the 'Text' area displays the input code:

```
const max = 100;
var arg, ret;

procedure isprime;
var i;
begin
  ret := 1;
```

This text is also highlighted with a red box. To the right of the text area, it says '8 matches (0.1ms)'. The 'Tools' section at the bottom provides a tooltip: 'Roll-over elements below to highlight in the Expression above. Click to open in Reference.' A yellow callout box at the bottom defines the character set: '[Character set. Match any character in the set.]'.

[http://ivanzuzak.info/noam/webapps/fsm simulator/](http://ivanzuzak.info/noam/webapps/fsm_simulator/)

FSM simulator

Visually simulate your DFAs, NFAs and ϵ -NFAs one input symbol at a time!

① Create automaton

[Input regex](#) [Input automaton](#)

Enter a regular expression into the input field below or click **Generate random regex** to have the app generate a simple regex randomly for you. Next, click **Create automaton** to create a FSM for the defined regex and display its transition graph.

A valid regex consists of alphanumeric characters representing the set of input symbols (e.g. **a**, **B**, **9**), the **\$** character representing the empty string, the choice operator **+**, the Kleene operator *****, and parentheses **(** and **)**. An example of a valid regex is:
(a+B)*(c9+\$)+\$.

[Generate random regex](#)



(ab)*abb(ab)*

② Simulate automaton

Enter a sequence of input symbols into the input field below or click **Random string**, **Acceptable string** and **Unacceptable string** to have the app generate random acceptable and unacceptable sequences for you.

Click **Read next** to have the FSM consume the next input symbol in the sequence and **Read all** to consume all remaining input symbols. Click **Step backward** to go back one symbol and **Reset** to reset the FSM and go back to the beginning of the input sequence.

The input field highlights the input symbol that will be read next.

[Random string](#) [Acceptable string](#) [Unacceptable string](#)

ababababababbab

[Start](#) [Reset](#) [Step backward](#) [Read next](#) [Read all](#)

[http://ivanzuzak.info/noam/webapps/fsm simulator/](http://ivanzuzak.info/noam/webapps/fsm_simulator/)

(ab)*abb(ab)*

Start Reset Step backward Read next Read all

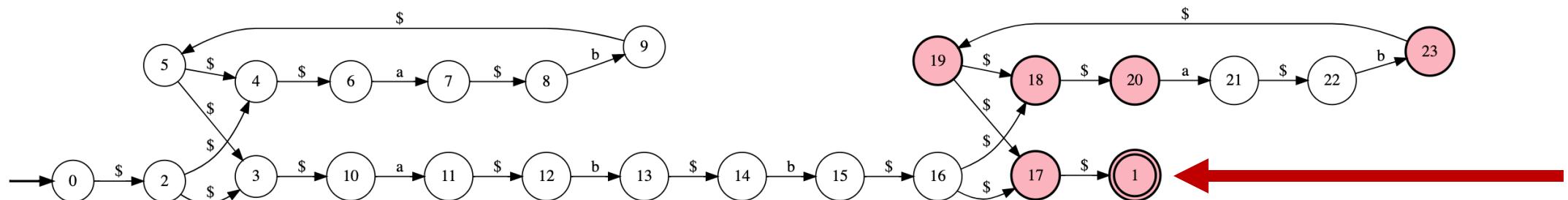
Select generated FSM type:

eNFA

Create automaton

③ Transition graph

The FSM being simulated is displayed in the form of a transition graph. The nodes representing the current states of the FSM are colored in ■.



课后作业（词法分析部分）

- 学习 Cooper 的课件（幻灯片）
 - 不明白的地方在我们课程的repo/issues里面提问，我会回答
 - 看完之后阅读/观看另外三位老师的课程，查缺补漏
- 玩一玩推荐的这几个可视化演示的小站点
 - 多数是GitHub开源的，直接看到算法的源代码
 - 如果寻找到好玩的可视化的站点欢迎开issues推荐

本次课程将介绍

- ~~第一节课课后作业的复习（包含一些元学习理论知识）~~
- ~~词法分析：知识地图、算法、可视化工具（很好玩）~~
- 几个必要的数学概念的补充（不要怕）
- 前端编程练习用语言PL/0的准备

几个必要的数学概念的补充 (不要怕)

- 我们这门课程里面说的数学，包含哪些东西？
- 问题的定义和解空间的确定
- 解的存在性
- 解的构造算法、找到最优解的算法
- 数学解法通过计算机程序实现的问题

(现在不理解不影响学习；后续遇到具体的例子会再解释)

我们这门课程里面说的数学，是个大杂烩

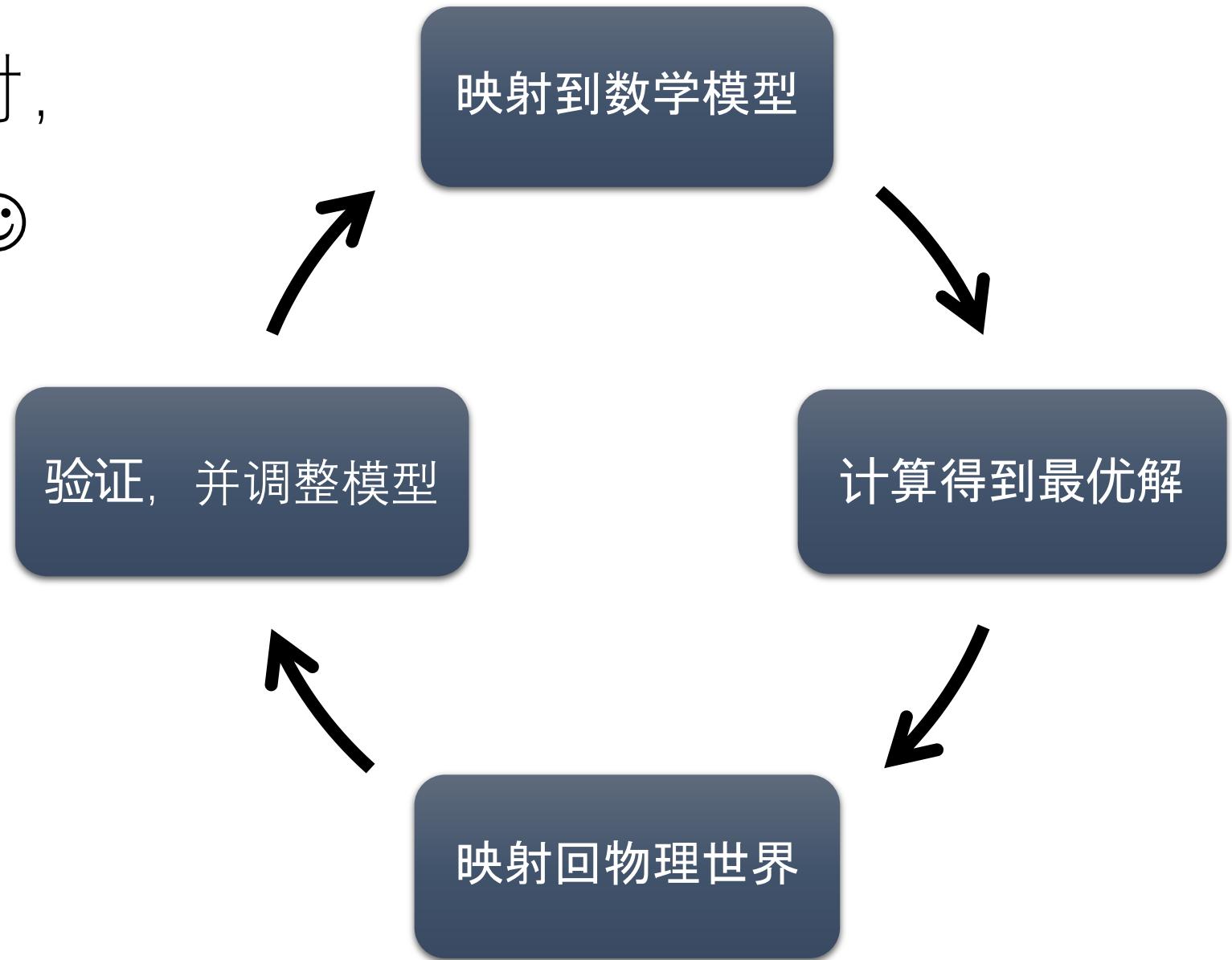
- 命题逻辑：如果p成立可以推出q成立， q推导r， 那么 $p \rightarrow r$
- 数学建模：建立一个模型然后在模型中推导
- 形式语言与自动机理论、计算理论
- 整数规划、最优化方法等
- 其它我也尚未理解的数学领域（部分优化使用到了）

(这一小节听不懂没关系，是我没讲清楚还没有找到更简单的方式，后续我会继续再遇到的时候再次详细说明)

物理世界 → 数学世界 → 物理世界

- 物理世界的属性和关系进行抽象，得到一个概念上的模型
 - 例如CPU抽象成编译器中微架构的表示，编程不同部件之间的图
- 在模型上，通过一些计算法则和约束条件，推演得到结论
 - 例如CPU被抽象之后，可以调度和排序出在模型中速度最快的代码序列
- 将模型中的结论映射回物理世界，验证是否满足要求
 - 不断修改模型也是预期中的工作，不可能一下子就完美

数学模型有可能不对，
尤其是你自己写的 😊
需要不断调整



解的空间和解的存在性

- 具体问题抽象成模型之后，所有可能的答案构成了一个集合/空间
 - 例：二维平面上的所有点
 - 例：所有的正则表达式
 - 例：一个编译器所有的参数组合
- 给出一个目标，是否存在一个答案（解）能够满足这个目标？
 - 是否有一个点，即落于线段A、又落于线段B上？
 - 是否存在一个正则表达式，能够识别字符串集合A，并拒绝字符串集合B？

找到解的算法、寻找最优的解、验证最优解

- 解决了解的存在性的问题之后，接下来就是如何找到这个（些）解
 - 穷举法如果能work，问题都好说
 - 构造法是另一种常见算法
 - 其它方法就比较难了：各种各样的搜索法

找到解的算法、寻找最优的解、验证最优解

- 最优解
 - 给定一些约束条件，一个目标，在可行解中找目标最大化的解
 - 数学性质好的解空间可以构造、搜索、逼近；不好的主要是搜索或近似解
 - 贪婪法、动态规划、剪枝查找、模拟退火、启发式算法.....
- 连续空间和离散空间
 - 数学性质（连续性、可微性）不好的离散空间比较难搞
 - 编译器中后端算法中存在大量的离散空间模型，后续会逐步说明

找到解的算法、寻找最优的解、**验证最优解**

- 验证最优解

- 一般而言，验证一个解是最优的，比找到这个最优解要容易（不会更难）
- 在真实的工程场景中，验证最优解有时候也不容易
- 一般性的，是有一个「差不多可以当作最优解接受」的范围

数学算法变成计算机代码

- 数字的有效范围和精度范围问题，至今仍是一大类安全隐患
- 算法的时间复杂度（后续每个后端算法都会有说明）
- 算法的空间复杂度（后续每个后端算法都会有说明）
- 数学模型跟真实世界之间的映射关系的精确程度：大量的取舍

一些可以当作知识点记忆的内容

- 编译器中大量算法处理的是离散空间的最优解问题
- 大量的问题是NP的；也就是说没有多项式时间的解法
- 因此，编译器中大量使用启发式算法寻找近似解
 - 启发式（Heuristic）就是「专家的猜测」☺

本次课程将介绍

- ~~第一节课课后作业的复习（包含一些元学习理论知识）~~
- ~~词法分析：知识地图、算法、可视化工具（很好玩）~~
- ~~几个必要的数学概念的补充（不要怕）~~
- 前端编程练习用语言PL/0的准备

PL/0 是什么？一个用于编译器教学的语言

- <https://en.wikipedia.org/wiki/PL/0>
 - 这里有完整的语法和历史介绍，阅读这一个网页就可以进行下一步练习了
 - 里面提供了一些例子，可以用于前一小节RE的测试中
- 理论上图灵完备
 - 实际上受限于可编写的程序的规模（跟C++之父为什么要发明C++同样原因）

为什么我们要用 PL/0

- 我们的课程包含了前端的实现练习，需要一个语言规范
 - 可选范围：PL/0、Tiger、C、C++、Java、Python、Ruby
- 同时，前端并不是我们本次课程重点，需要一个简单的语言
 - 可选范围：PL/0、Tiger、~~C、C++、Java、Python、Ruby~~
- 我们是自学+指导的慕课SSR教学模式，最好用有大量的资料
 - 可选范围：PL/0、~~Tiger~~（有防作弊需要）、~~C、C++、Java、Python、Ruby~~

在GitHub上搜索 PL0 : 大量学习资料 😊

The screenshot shows the GitHub search interface with the query 'pl0' entered in the search bar. The results page displays 575 repository results. On the left, there is a sidebar with navigation links for Repositories (575), Code, Commits (27K), Issues (798), Packages, Marketplace, Topics (1), Wikis (100), and Users (134). Below this is a Languages section showing counts for C (157), C++ (110), Java (53), Python (46), and JavaScript (20).

575 repository results Sort: Best match ▾

oriontransfer/PL0-Language-Tools ● Python ★ 58
An implementation of PL/0 including a tree-based interpreter and virtual machine.
Updated on Sep 19, 2017

cwalk/PL0-Compiler ● C ★ 80
Compiler written for PL0 programming Language.
Written in C, for COP3402 class from UCF.
c compiler
Updated on Apr 1, 2016

gdut-yy/PL0 ● C++ ★ 20
GDUT 编译原理课程的课内实验和课程设计（含报告）
Updated on Nov 15, 2018

我们会怎么用 PL/0 来帮助学习

- 根据语言规范，用 flex/bison 实现词法分析和语法分析
- 翻译到 LLVM IR，用这个过程学习 LLVM IR 的操作
- 掌握后，你可以在工作中自己创造新的语言，DSL或通用都可以

课后练习 (PL/0部分)

- 阅读 PL/0 的语言规范，尝试自己手写几个例子
- 用之前小节提到的可视化工具尝试用RE实现词法分析的匹配部分
- 从GitHub或者gitee上找一些参考的例子，看一看
- 预习 flex 这个工具的语法及使用，跑个例子

方舟·编译技术入门与实战

记得完成本次课程的作业 ☺

<https://github.com/lazyparser/becoming-a-compiler-engineer>