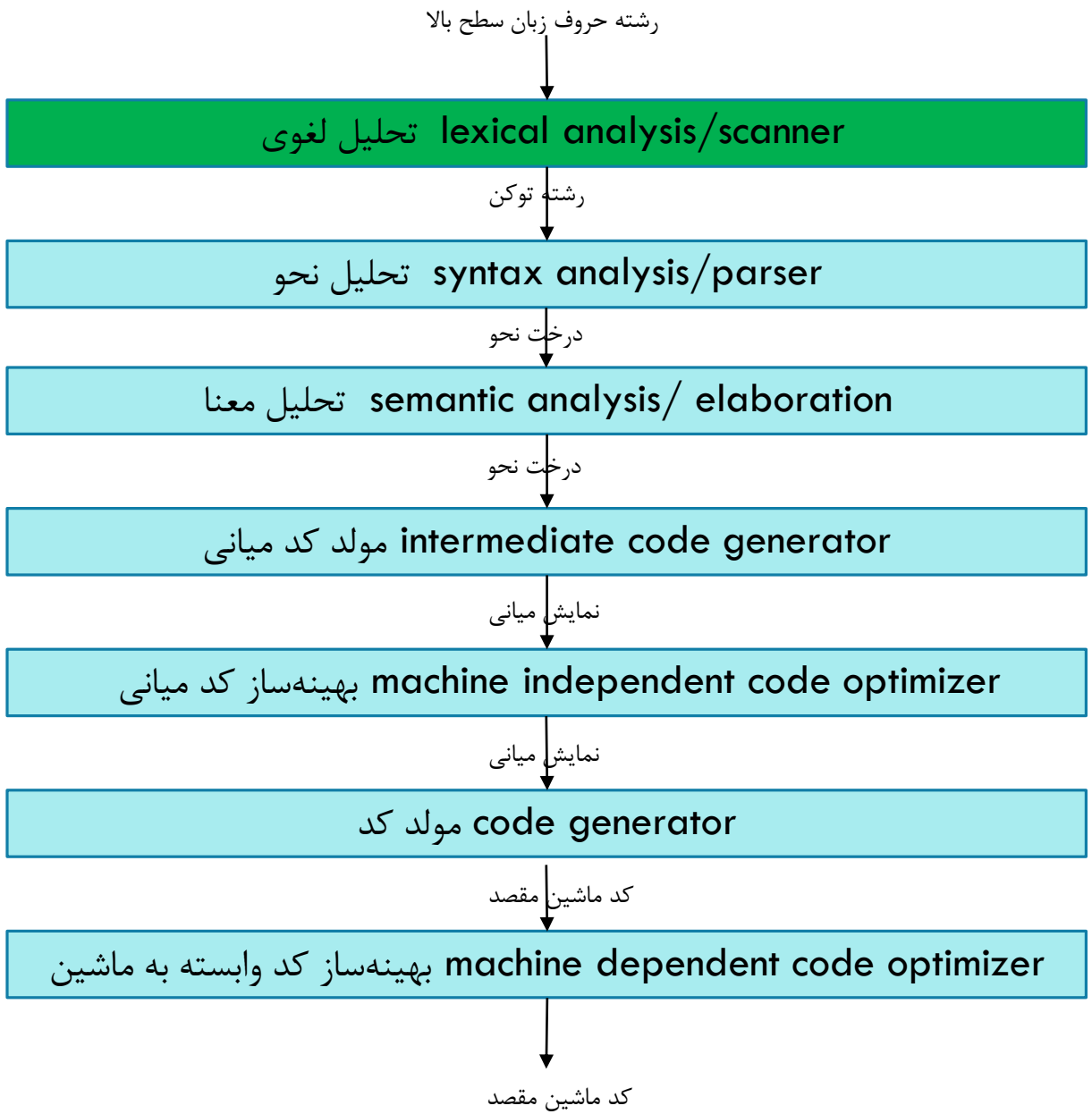
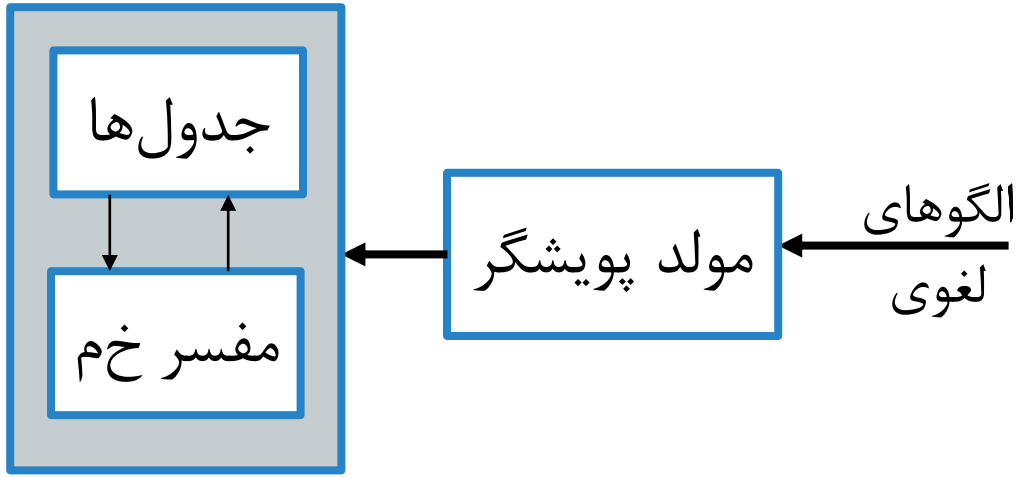
کامپایلر
پویشگر
مختصری درباره پیاده‌سازی -/د/مه

محسن هوشمند
دانشکده تکنولوژی اطلاعات و علم رایانه
دانشگاه تحصیلات تکمیلی علوم پایه زنجان


رشته حروف زبان سطح بالا
lexical analysis/scanner تحلیل لغوی
رشته توکن
syntax analysis/parser تحلیل نحو
درخت نحو
semantic analysis/ elaboration تحلیل معنا
درخت نحو
intermediate code generator مولد کد میانی
نمایش میانی
machine independent code optimizer بهینه‌ساز کد میانی
نمایش میانی
code generator مولد کد
کد ماشین مقصد
machine dependent code optimizer بهینه‌ساز کد وابسته به ماشین
کد ماشین مقصد

پویشگر جدول-گرا



نمایش خودکاره در قالب ساختار داده

- جدول انتقال دو بعدی
- هر خانه جدول مشخص‌گر
  - یا انتقال به حالت دیگر
  - یا تحویل تکه
  - یا اعلام خطاء

جدول دیگر

- مشخص‌کننده اینکه در هر حالت تکه کاملی بدست آمده یا خیر
  - به همراه تکه

پویشگر جدول-گرا

مثال $r\ [0\ldots9]^+$

دارای چهاربخش

- آغازین
- حلقه مدل‌ساز رفتار خم
  - خواندن حرف بعدی و تقلید رفتار خم
  - توقف با ورود به حالت خطا
  - دو جدول CharCat و دلتا: جهت نگهداری اطلاعات خم
- حلقه برگشت هنگام گذرکردن از رشته‌های موردپذیرش خم
  - استفاده از پشته جهت برگرداندن پویشگر به آخرین حالت پذیرش
- بخش تفسیر و اعلام نتیجه

```
NextWord()
  state ← s0;
  lexeme ← " ";
  clear stack;
  push(bad);

  while (state≠se) do
    NextChar(char);
    lexeme ← lexeme + char;
    if state ∈ SA
        then clear stack;
    push(state);
    cat ← CharCat[char];
    state ← δ[state,cat];
  end;

  while(state ∉ SA and
        state≠bad) do
    state ← pop();
    truncate lexeme;
    RollBack();
  end;

  if state ∈ SA
    then return Type[state];
    else return invalid;
```

r	0, 1, 2, ..., 9	EOF	Other
Register	*Digit*	*Other*	*Other*

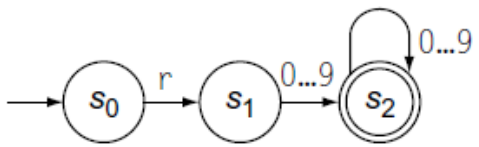
The Classifier Table, *CharCat*

	Register	*Digit*	*Other*
s_0	s_1	s_e	s_e
s_1	s_e	s_2	s_e
s_2	s_e	s_2	s_e
s_e	s_e	s_e	s_e

The Transition Table, δ

s_0	s_1	s_2	s_e
invalid	*invalid*	*register*	*invalid*

The Token Type Table, *Type*



The Underlying DFA

پویشگر جدول-گرا – ایجاد جداول رده‌بند و انتقال

ایجاد جدول اولیه

- هر نویسه ورودی -> یک ستون
- هر حالت -> یک ردیف

پویشگر جدول-گرا – ایجاد جداول رده‌بند و انتقال

ایجاد جدول اولیه

- هر نویسه ورودی -> یک ستون
- هر حالت -> یک ردیف

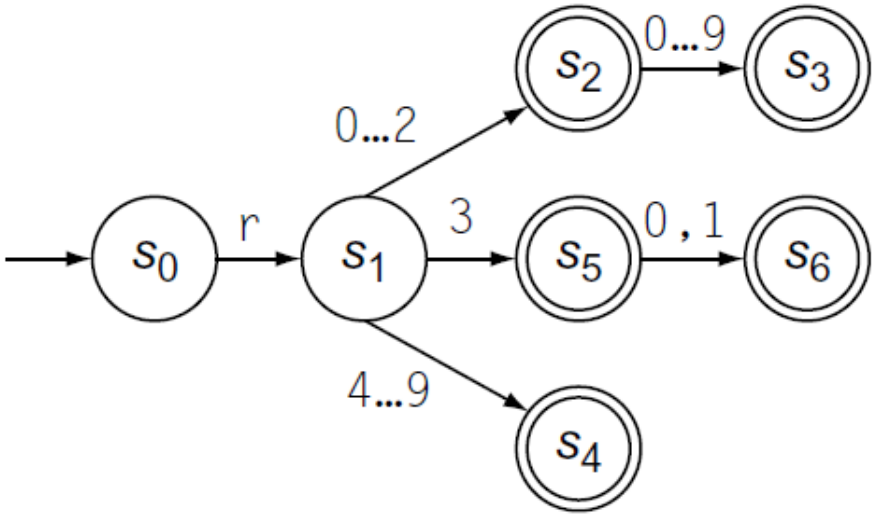
فشرده‌سازی

- ترکیب ستون‌های همسان در یک ستون
- دو ستون همسان
  - دارای مقادیر یکسان انتقال حالت

پویشگر جدول-گرا – ایجاد جداول رده‌بند و انتقال

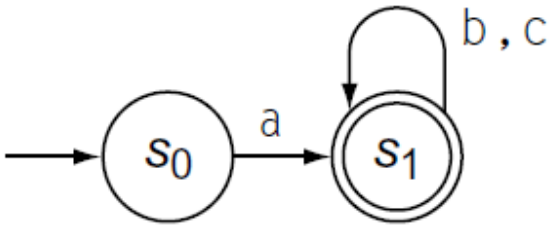
	r	0,1	2	3	4...9	Other
s_0	s_1	s_e	s_e	s_e	s_e	s_e
s_1	s_e	s_2	s_2	s_5	s_4	s_e
s_2	s_e	s_3	s_3	s_3	s_3	s_e
s_3	s_e	s_e	s_e	s_e	s_e	s_e
s_4	s_e	s_e	s_e	s_e	s_e	s_e
s_5	s_e	s_6	s_e	s_e	s_e	s_e
s_6	s_e	s_e	s_e	s_e	s_e	s_e
s_e	s_e	s_e	s_e	s_e	s_e	s_e

$r([0...2] ([0...9] |\epsilon) |[4...9] | (3 (0 | 1 |\epsilon)))$



پویشگر جدول-گرا – ایجاد جداول رده‌بند و انتقال

a(b | c)*



	a	b , c	Other
s_0	s_1	s_e	s_e
s_1	s_e	s_1	s_e

پویشگر کدی

بهبود عملکرد روش جدول‌گرا؟

ۆایته و هیورینگ پیشنهاد استفاده مستقیم از کد به جای جدول

پویشگر کدی

کاهش هزینه جدول‌گرا

- کاهش یکی یا هر دو عمل بنیادی
  - خواندن نویسه
  - یافتن انتقال بعدی
- راه‌حل نمایش ضمنی
  - ساده‌کردن روش دومرحله‌ای و محاسبات جستجوی جدول
  - حذف مراجعه به حافظه
  - کارکرد همانند با جدولی
  - ولی با افزونگی کمتر
- تغییر اساسی در حلقه مرکزی

```
while (state ≠ s_e) do
  NextChar(char);
  cat ← CharCat[char];
  state ← δ[state,cat];
end;
```

پویشگر کدی

برای هر نویسه دو بار بررسی جدول

دستیابی به عضو i–ام CharCat،

$$@CharCat_0 + i \times w$$

دستیابی به جدول دلتا

$$@\delta_0 + (state \times number\ of\ columns\ in\ \delta + cat) \times w$$

پویشگر کد

$r[0...9]+$

```
s_init :  lexeme ← " ";
          clear stack;
          push(bad);
          goto s_0 ;

s_0 :     NextChar(char);
          lexeme ← lexeme + char;
          if state ∈ S_A
              then clear stack;
          push(state);
          if (char = 'r')
              then goto s_1 ;
              else goto s_out ;

s_1 :     NextChar(char);
          lexeme ← lexeme + char;
          if state ∈ S_A
              then clear stack;
          push(state);
          if ('0' ≤ char ≤ '9')
              then goto s_2 ;
              else goto s_out ;

s_2 :     NextChar(char);
          lexeme ← lexeme + char;
          if state ∈ S_A
              then clear stack;
          push(state);
          if '0' ≤ char ≤ '9'
              then goto s_2 ;
              else goto s_out

s_out :   while (state ∉ S_A and
                 state ≠ bad) do
              state ← pop();
              truncate lexeme;
              RollBack();
          end;
          if state ∈ S_A
              then return Type[state];
              else return invalid;
```

```
state := 1                          -- start state
loop
    read cur_char
    case state of
        1 : case cur_char of
                ' ', '\t', '\n' : ...
                'a'...'z' :       ...
                '0'...'9' :       ...
                '>' :             ...
                ...
        2 : case cur_char of
                ...
        ...
        n: case cur_char of
                ...
```

پویشگر دستی

استفاده اغلب کامپایلرها از پویشگر دستی

کامپایلرهای تجاری

کاهش افزونگی استفاده از خ‌م‌م

بافر دنباله ورودی

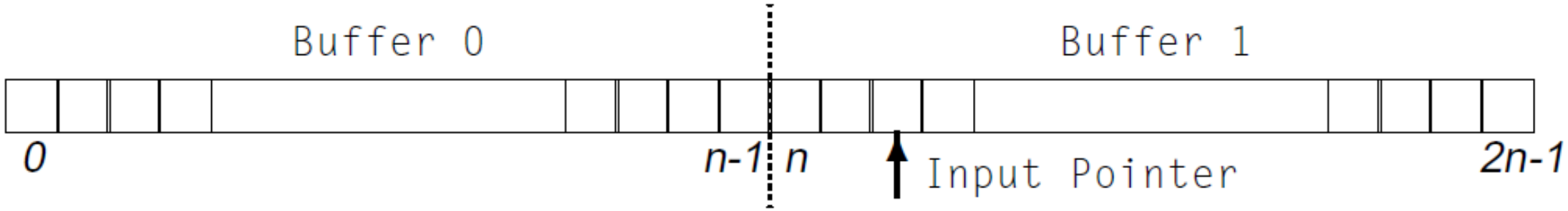
ورودی/خروجی

- خواندن نویسه‌به‌نویسه از ورودی پرهزینه
- سرعت کمتر
- نیاز به خواندن یک یا چند نویسه بعدی جهت اطمینان از انتخاب لغت درست
  - مثال- شناسه، یا روابط چون =>
- ورودی/خروجی بافرشده
- دارای هزینه:
  - افزونگی بزرگ ثابت
  - هزینه به ازای هر نویسه
- بافر بزرگتر، هزینه کمتر

عمل بازگشت به عقب

- ساده‌تر شدن عمل با بافر و اشاره‌گر
- کاهیدن مقدار اشاره‌گر

جفت‌بافر



```
Char ← Buffer[Input];
Input ← (Input+1) mod 2n;

if (Input mod n = 0)
    then begin;
       fill Buffer[Input:Input+n-1];
       Fence ← (Input+n) mod 2n;
   end;

return Char;
```

Implementing *NextChar*

```
Input ← 0;
Fence ← 0;
fill Buffer[0:n];
```

Initialization

```
if (Input = Fence)
   then signal roll back error;
Input ← (Input-1) mod 2n;
```

Implementing *RollBack*

خطای لغوی

۱- سختی صدور خطای لغت‌کاو بدون کمک دیگر مولفه‌ها

- خطای کد مبدا
- مثال- **fi** را شناسه تعیین می‌کند تا بعد

```
fi ( a == f(x)) ...
```

۲- مواقع ناتوانی لغت‌کاو از ادامه کار به دلیل نیافتن هیچ الگوئی

- وجه بیم و اضطرار!
  - زدودن هراس
  - ساده‌ترین راه‌حل و استراتژی
  - حذف تمامی نویسه‌ها بعدی
- حذف یک نویسه از ورودی باقی‌مانده
- درج نویسه مفقود به ورودی باقی‌مانده
- جانشینی نویسه‌ای با نویسه دیگر
- تعویض مکان دو نویسه همسایه

تحلیل

جدایی تحلیل لغوی از تحلیل نحوی

- کارایی
- امروزه پیدایش فنون تجزیه کارا موجب تزلزل در این نظر
- جداسازی متناسب بین مسائل واژوی و نحوی

تاثیر آنچنان ندارد

نگسمان و همکاران

- روش‌های سریعتر کردن کد دستی

جونز ۱۹۸۸

- کد مستقیم ولی نه به شکل کد اسپاگتی عرضه شده
- جریان کنترل بهتر

بروؤر مقایسه ۱۲ پیاده‌سازی پویشگر

منابع

[ببر سبز]

[اژدرها]

[کوپر]

[فیشر]

[نیسان]