

# LL-Parser: Fortgeschrittene Techniken

---

Carsten Gips (HSBI)

Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

# LL-Parser mit Backtracking

```
wuppie();           // Vorwärtsdeklaration  
wuppie() { ...}     // Definition
```

```
func : fdef | fdecl ;  
fdef : head '{' body '}' ;  
fdecl: head ';' ;  
head : ... ;
```

# LL-Parser mit Backtracking

```
wuppie();           // Vorwärtsdeklaration  
wuppie() { ... }    // Definition
```

```
func : fdef | fdecl ;  
fdef : head '{' body '}' ;  
fdecl: head ';' ;  
head : ... ;
```

```
def func():  
    if speculate(fdef): fdef()      # Spekchiere auf "fdef"  
    elif speculate(fdecl): fdecl()  # Spekchiere auf "fdecl"  
    else: raise Exception()
```

## Details: Speklatives Matchen

```
def speculate(fn):  
    success = True  
  
    mark()                # markiere aktuelle Position  
  
    try:    fn()           # probiere Regel fn()  
    catch: success = False  
  
    clear()               # Rollback  
  
    return success
```

Quelle: Eigener Code basierend auf einer Idee nach (Parr 2010, p. 60)

# Spekulatives Matchen: Hilfsmethoden I/II

```
class Parser:
    Lexer lexer
    markers = []      # Integer-Stack: speichere Tokenpositionen
    lookahead = []    # Puffer (1 Token vorbefüllt via Konstruktor)
    start = 0         # aktuelle Tokenposition im lookahead-Puffer

    def mark():
        markers.push(start)

    def clear():
        start = markers.pop()
```

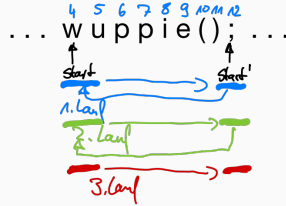
Quelle: Eigener Code basierend auf einer Idee nach (Parr 2010, pp. 61/62)

# Spekulatives Matchen: Hilfsmethoden II/II

```
def consume():  
    ++start  
    if start == lookahead.count() and markers.isEmpty():  
        start = 0; lookahead.clear()  
    sync(1)  
  
def lookahead(i):  
    sync(i)  
    return lookahead.get(start+i-1)  
  
def sync(i):  
    n = start + i - lookahead.count()  
    while (n > 0):  
        lookahead.add(lexer.nextToken()); --n
```

Quelle: Eigener Code basierend auf einer Idee nach (Parr 2010, pp. 61/62)

# Verbesserung Backtracking: Packrat Parser (Memoizing)



```
def fun():  
    if spec(def): def()  
    if spec(decl): decl()
```

```
def def():  
    head()  
    match("#;")  
    ...
```

```
def decl():  
    head()  
    match("#;")
```

`head_memo[4] = 12`

# Skizze: Idee des Packrat-Parsing

```
head_memo = {}

def head():
    if head_memo.get(start) == -1:
        raise Exception()           # kein Match
    if head_memo.get(start) >= 0:
        start = head_memo[start]; return True    # Vorspulen
    else:
        failed = False; start_ = start
        try: ...                     # rufe die ursprüngliche head()-Regel auf
        catch(e): failed = True; raise e
        finally: head_memo[start_] = (failed ? -1 : start)
```

Quelle: Eigener Code basierend auf einer Idee nach (Parr 2010, pp. 65/66)



Problem in Java: `enum` ab Java5 Schlüsselwort

```
prog : (enumDecl | stat)+ ;  
stat : ... ;  
  
enumDecl : ENUM id '{' id (',' id)* '}' ;
```

# Semantische Prädikate

Problem in Java: `enum` ab Java5 Schlüsselwort

```
prog : (enumDecl | stat)+ ;  
stat : ... ;  
  
enumDecl : ENUM id '{' id (',' id)* '}' ;
```

```
def prog():  
    if lookahead(1) == ENUM and java5: enumDecl()  
    else: stat()
```

# Semantische Prädikate in ANTLR

```
@parser::members {public static boolean java5;}
```

```
prog : ({java5}? enumDecl | stat)+ ;
```

```
stat : ... ;
```

```
enumDecl : ENUM id '{' id (',' id)* '}' ;
```

```
ENUM : 'enum' {java5}? ;
```

```
ID : [a-zA-Z]+ ;
```

- LL(1) und LL(k): Erweiterungen
  - Dynamischer Lookahead: BT-Parser mit Packrat-Ergänzung
  - Semantische Prädikate zum Abschalten von Alternativen



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.