

Syntaxanalyse mit Rekursivem



Abstieg

Satz: a e g c f d b

$G(S):$ $S = aAb \mid cAd.$
 $A = eB \mid f.$
 $B = gS.$

next() --> a erkenne **S**
 erkenne **a** oder **c** (*a erkannt, wähle erste Alternative*)

next() --> e erkenne **A**
 erkenne **e** oder **f** (*e erkannt, wähle 1. Alt.*)

next() --> g erkenne **B**
 erkenne **g** (*g erkannt*)

next() --> c erkenne **S**
 erkenne **a** oder **c** (*c erkannt, wähle 2. Alt.*)

next() --> f erkenne **A**
 erkenne **e** oder **f** (*f erkannt, wähle 2. Alt.*)

next() --> d (*A erkannt*)
 erkenne **d** (*d erkannt*)

next() --> b (*S erkannt*)
 (*B erkannt*)
 (*A erkannt*)
 erkenne **b** (*b erkannt*)
 (*S erkannt*)

Parser: wichtige Felder &



Methoden

```
private Token t;           // last recognized token
private Token la;        // look ahead token
private int sym;         // kind of look ahead token
public Scanner scanner; // Reference to Scanner

private void scan () {
    t = la; la = scanner.next(); sym = la.kind;
}

private void check (int expected) {
    if (sym == expected) {
        scan();
    } else {
        error("TOKEN_EXPECTED", Token.kindToString(expected));
    }
}

public void error (String msgKey, Object... msgParams) {
    scanner.errors.error(la.line, la.col, msgKey, msgParams);
    // panic mode
    throw new Error(Messages.getString("PANIC_MODE"));
}
```

Panic Mode



- beim ersten gefundenen Fehler wird Analyse abgebrochen
- Abbruch nicht mit **System.exit(0);** , weil
 - dadurch die VM beendet wird
 - das beim Testen mit JUnit zum sofortigen Abbruch führt, d.h.
 - es werden keine weiteren Test ausgeführt
 - es wird kein Ergebnis angezeigt bzw. das GUI wird sofort beendet
- Bessere Lösung:
throw new Error(Messages.getString("PANIC_MODE"));
 - Error wird von den Testfällen gefangen und ausgewertet

Bsp 1: S = a B c.

SEQUENZ

```
private void S () {  
    check(a);  
    B();  
    check(c);  
}
```

Bsp 2: $S = a \mid Bc \mid$

d.

ALTERNATIVEN

$\text{first}(B) = \{ e, f \}$

```
private void S () {
    switch (sym) {
        case a:
            scan(); break;
        case e: case f:
            // Erkennung von e und f in B!
            B(); check(c); break;
        case d:
            scan(); break;
        default:
            error(...);
    }
}
```

Bsp 3: $S = (a \mid B)$

C.

SEQUENZ mit ALTERNATIVE

$\text{first}(B) = \{ e, f \}$

```
private void S () {  
    switch (sym) {  
        case a:  
            scan(); break;  
        case e: case f:  
  
            B(); break;  
        default:  
            error(...);  
    }  
    check(c);  
}
```

```
ORDER:  
if (sym == a) {  
    scan();  
} else if (sym == e ||  
           sym == f) {  
    B();  
} else {  
    error(...);  
}  
check(c);
```

Bsp 4: $S = [a \mid B] c.$

SEQUENZ mit OPTIONALER ALTERNATIVE

$\text{first}(B) = \{ e, f \}$

```
private void S () {  
    switch (sym) {  
        case a:  
            scan(); break;  
        case e: case f:  
  
            B(); break;  
    }  
    // kein error  
    check(c);  
}
```

```
ORDER:  
if (sym == a) {  
    scan();  
} else if (sym == e ||  
           sym == f) {  
    B();  
}  
// kein error  
check(c);
```

Bsp 5: $S = \{ a \mid B \}$

C. (1)

SEQUENZ mit ITERATION

$\text{first}(B) = \{ e, f \}$

```
private void S () {  
    while (sym == a || sym == e || sym == f) {  
        if (sym == a) {  
            scan();  
        } else {  
            B();  
        } // kein error  
    }  
    check(c);  
}
```


Bsp 5: $S = \{ a \mid B \}$

C. (2)

SEQUENZ mit ITERATION

$\text{first}(B) = \{ e, f \}$

```
private void S () {  
    while (sym != c) {  
        switch (sym) {  
            case a:  
                scan(); break;  
            case e: case f:  
                B(); break;  
            default: // default Zweig hier nötig!  
                error(...);  
        }  
    }  
    scan();  
}
```

Bsp 5: $S = \{ a \mid B \}$

C. (3)

SEQUENZ mit ITERATION

$\text{first}(B) = \{ e, f \}$

```
private void S () {  
    while (true) {  
        if (sym == a) {  
            scan();  
        } else if (sym == e || sym == f) {  
            B();  
        } else {  
            break; // Schleife verlassen  
        }  
    }  
    check(c);  
}
```

Bsp 6: $S = B \{ a B \}$.

(1)

SEQUENZ mit ITERATION

$\text{first}(B) = \{ e, f \}$

```
private void S () {  
    B();  
    while (sym == a) {  
        scan();  
        B();  
    }  
}
```

Bsp 6: $S = B \{ a B \}$.

(2)

SEQUENZ mit ITERATION

$\text{first}(B) = \{ e, f \}$

```
private void S () {  
    while (true) {  
        B();  
        if (sym == a) {  
            scan();  
        } else {  
            break;  
        }  
    }  
}
```

Bsp 7: $S = a \{ B \} C.$

$\text{first}(B) = \{ e, f \}$

$\text{first}(C) = ?$

```
private void S () {  
    check(a);  
    while (sym == e || sym == f) {  
        B();  
    }  
    C();  
}
```

UE 3: Syntaxanalyse (*Parser*)



- Keine neuen Angabe- und Test-Klassen
- Abgabe
 - siehe Abgabeanleitung auf Homepage!
 - elektronisch bis Mi, 22.11.2006, 18:00
 - alle zum Ausführen benötigten Dateien
 - auf Papier bis Mi, 22.11.2006, 18:00
 - nur Parser.java