Dagens tema

- Syntaks
  (kapittel 2.1 + Komp. 47, kap. 1 og 2)

Litt om kompilering og interpretering

En **kompilator** oversetter et program til et annet språk, for eksempel maskinspråk.

Et program **interpreteres** når det finnes et annet program som leser det inn og så *simulerer* operasjonene.

Hva trenger vi å vite når vi skal lage en kompilator/interpreter for et språk?

Syntaks og semantikk

En beskrivelse av et (programmerings-) språk består av to hovedkomponenter:

Syntaktiske regler forteller hva slags *form* et lovlig program skal ha.

Semantiske regler sier noe om hva de ulike setningene i språket *betyr*.

I tillegg snakker vi gjerne om pragmatikk. Dette er en uformell del som forteller om språket og *bruken* av det. I denne sammenhengen er eksempler viktige.

Syntaktiske regler

Leksikalske regler som omhandler oppbygningen av de minste meningsbærende enhetene i programmet (navn, tall, nøkkelord, operatorer, . . .)

- Er store/små bokstaver signifikante?
- Hvordan ser tilordningsoperatoren ut?

De "egentlige" syntaksreglene som forteller hvordan program kan bygges opp fra de leksikalske enhetene.

- Alle venstreparenteser må ha en matchende høyreparentes (og motsatt).
- Ikke semikolon før `else`!

Kompilatorer har gjerne en egen "preprosessor" som deler programmet opp i en sekvens av leksikalse enheter og leverer disse videre til resten av kompilatoren.

Semantiske regler

Også de semantiske reglene kan deles i to grupper:

Statiske regler omtaler det som kan sjekkes av kompilatoren *før* selve utførelsen av programmet.

- Alle variable skal være deklarert.
- Samsvar mellom deklarasjon og bruk av en variabel.

Dynamiske regler sier hva som skal skje under utførelsen av et program.

(Ghezzi&Jazayeri) bruker en *operasjonell semantikk*, det vil si at de beskriver oppførselen til en abstrakt prosessor som utfører program skrevet i språket.

Syntaktisk beskrivelse av språk

Litt terminologi

Alfabet – en mengde av (grunn-)symboler

Streng – en sekvens av symboler fra alfabetet

Setning – en tillatt streng

Grammatikk – en matematisk angivelse av setninger ved hjelp av *grunnsymboler*, *metasymboler* og *produksjoner*

Språk – en mengde av setninger

kan beskrives ved en grammatikk eller en automat

Noen eksempler på syntaksbeskrivelse

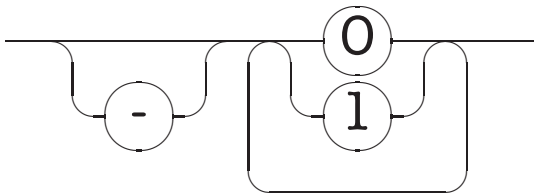
BNF-grammatikker

$\langle tall \rangle \quad \rightarrow$ **-** $\langle siffer \rangle \mid \langle siffer \rangle$
$\langle siffer \rangle \rightarrow$ **0** $\langle sifre \rangle \mid$ **1** $\langle sifre \rangle$
$\langle sifre \rangle \rightarrow$ **0** $\langle sifre \rangle \mid$ **1** $\langle sifre \rangle \mid \varepsilon$

Utvidet BNF

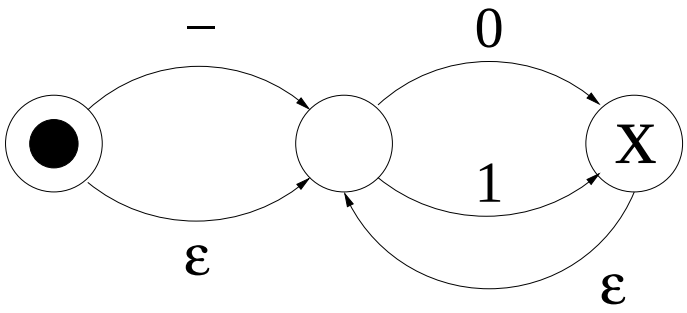
$\langle tall \rangle \rightarrow [\![\textbf{-}]\!]^{?} [\![\textbf{0} \mid \textbf{1}]\!]^{+}$

Syntaksdiagram (jernbanediagram)

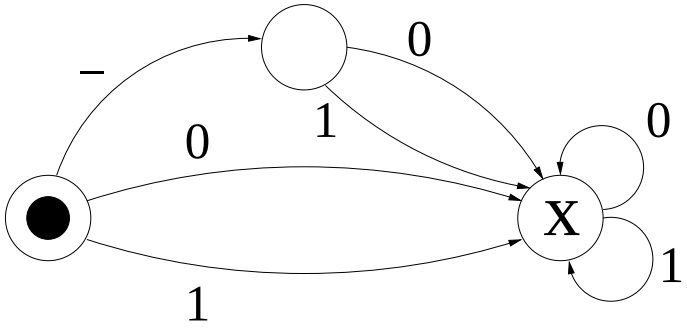


Noen eksempler på syntaksbeskrivelse

Ikke-deterministiske automater



Deterministiske automater



BNF-grammatikker og syntaksdiagram

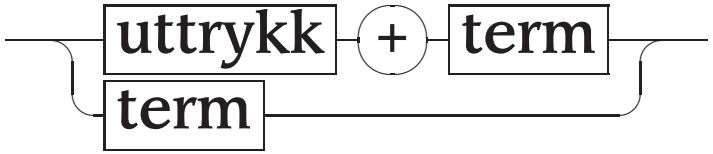
BNF-grammatikk:

$\langle uttrykk \rangle \rightarrow \langle uttrykk \rangle$ **+** $\langle term \rangle \mid \langle term \rangle$

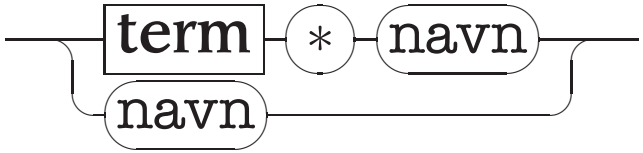
$\langle term \rangle \quad \rightarrow \langle term \rangle * \textbf{navn} \mid \textbf{navn}$

Syntaksdiagram:

uttrykk



term



Begge disse er bygget opp av:

Grunnsymboler **(terminalsymboler)** er de symbolene som forekommer i selve programteksten: +, ∗, **navn**

Metasymboler **(ikke-terminaler)** med egne definisjoner: ⟨*uttrykk*⟩, ⟨*term*⟩

Produksjoner er definisjoner av metasymboler:
⟨*uttrykk*⟩ → ⟨*uttrykk*⟩ **+** ⟨*term*⟩ | ⟨*term*⟩
(Merk: Dette er egentlig *to* produksjoner.)

Takket være metasymbolene kan vi:

- forenkle definisjonene
- spare kode ved gjentagelse
- lage rekursive definisjoner

Rekursive produksjoner er produksjoner der metasymbolet på venstresiden kan gå igjen én eller flere ganger på høyresiden:

$\langle uttrykk \rangle \rightarrow \langle uttrykk \rangle$ **+** $\langle term \rangle \mid \langle term \rangle$

Dette er et eksempel på venstrerekursjon, siden metasymbolet kommer igjen helt til venstre i høyresiden. Tilsvarende kan vi ha høyrerekursjon.

Vi kan også ha produksjoner med tom høyreside:

$\langle sifre \rangle \rightarrow$ **0** $\langle sifre \rangle \mid$ **1** $\langle sifre \rangle \mid \varepsilon$

Avledning av setninger

De mulige setningene i et språk definert av en BNF-grammatikk er nøyaktig de som fremkommer ved å benytte følgende prosedyre:

1. Start med startsymbolet.
2. For hvert metasymbol, erstatt dette med et av alternativene på høyresiden i den tilhørende definisjonen.
3. Gjenta forrige punkt til vi står igjen med bare grunnsymboler.

Dette kalles for en *avledning* fra startsymbolet til en ferdig setning, og kan representeres som et syntakstre.

En slik avledning kan også angis som en sekvens av halvutviklede setningsformer, der vi begynner med startsymbolet og i hvert steg bare anvender én produksjon.

En venstreavledning er en slik sekvens der man hele tiden utvikler det symbolet som står lengst til venstre. Tilsvarende kan vi snakke om en høyreavledning.

Flertydige grammatikker

Dersom enhver setning i språket kan avledes ved ett og bare ett syntakstre, er grammatikken entydig, ellers er den flertydig.

Eksempel:

$$\langle uttrykk \rangle \rightarrow \textbf{navn} \mid \langle uttrykk \rangle \ \textbf{+} \ \langle uttrykk \rangle \mid \langle uttrykk \rangle \ \textbf{*} \ \langle uttrykk \rangle$$

navn + navn * navn

Utvidet BNF

I utvidet BNF kan vi bruke følgende metasymboler i høyresiden:

$\mid$	skiller alternativer (også *innenfor* en produksjon)
?	angir at noe kan forekomme 0 eller 1 gang
*	0 eller flere ganger
+	1 eller flere ganger
$[\![\ldots]\!]$	grupperer symboler

Eksempel: Flyt-tall i Java

(hentet fra *The Java Language Specification*)

$$
\begin{aligned}
\langle \textit{FloatingPointLiteral} \rangle &\rightarrow \langle \textit{Digits} \rangle \ \texttt{.} \ \langle \textit{Digits} \rangle^{?} \ \langle \textit{ExponentPart} \rangle^{?} \ \langle \textit{FloatTypeSuffix} \rangle^{?} \\
\langle \textit{FloatingPointLiteral} \rangle &\rightarrow \texttt{.} \ \langle \textit{Digits} \rangle \ \langle \textit{ExponentPart} \rangle^{?} \ \langle \textit{FloatTypeSuffix} \rangle^{?} \\
\langle \textit{FloatingPointLiteral} \rangle &\rightarrow \langle \textit{Digits} \rangle \ \langle \textit{ExponentPart} \rangle \ \langle \textit{FloatTypeSuffix} \rangle^{?} \\
\langle \textit{FloatingPointLiteral} \rangle &\rightarrow \langle \textit{Digits} \rangle \ \langle \textit{ExponentPart} \rangle^{?} \ \langle \textit{FloatTypeSuffix} \rangle \\
\langle \textit{Digits} \rangle &\rightarrow \langle \textit{Digit} \rangle^{+} \\
\langle \textit{Digit} \rangle &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\langle \textit{ExponentPart} \rangle &\rightarrow [\![\texttt{e} \mid \texttt{E}]\!] \ [\![\texttt{+} \mid \texttt{-}]\!]^{?} \ \langle \textit{Digits} \rangle \\
\langle \textit{FloatTypeSuffix} \rangle &\rightarrow \texttt{f} \mid \texttt{F} \mid \texttt{d} \mid \texttt{D}
\end{aligned}
$$

Oppgave:

Hvilke av disse tallene er lovlige flyt-tall i Java?

3
3.14
6.28e–18
.1
22.D
e3
1.2e+2.0

Ulike typer språk

Det er vanlig å dele språk som kan beskrives med en BNF-grammatikk inn i følgende grupper:

- **Type 3**-språk («regulære språk») har ett metasymbol på venstresiden og kun grunnsymboler på høyresiden, eventuelt med et metasymbol til sist.

  ⟨*binærtall*⟩ → **0** |
  0 ⟨*binærtall*⟩ |
  1 |
  1 ⟨*binærtall*⟩

- **Type 2**-språk («kontekst-frie») har bare ett metasymbol på venstresiden.

  Omtrent alle programmeringsspråk benytter en kontekst-fri grammatikk til å definere språkets syntaks.

  — Det gir en klar men lettlest definisjon av syntaksen.

  — Det er en basis for å skrive en syntakstolker («parser»).

- **Type 1**-språk («kontekst-sensitive») krever at høyresiden er minst like lang som venstresiden.

  Dette gjør det mulig å sjekke navnebindinger og finne typefeil. Ble brukt til Algol-68 men lite siden.

- **Type 0**-språk har ingen restriksjoner.

  Disse har bare teoretisk interesse.