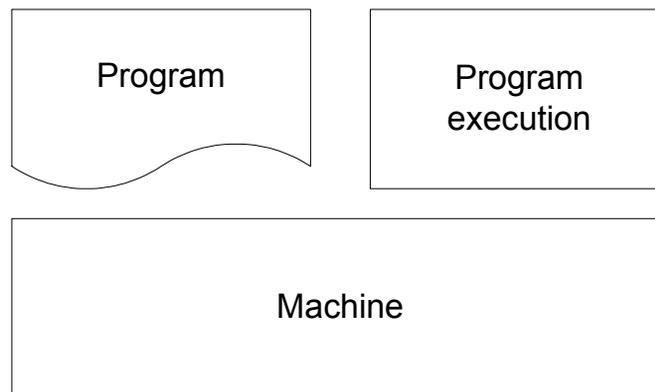
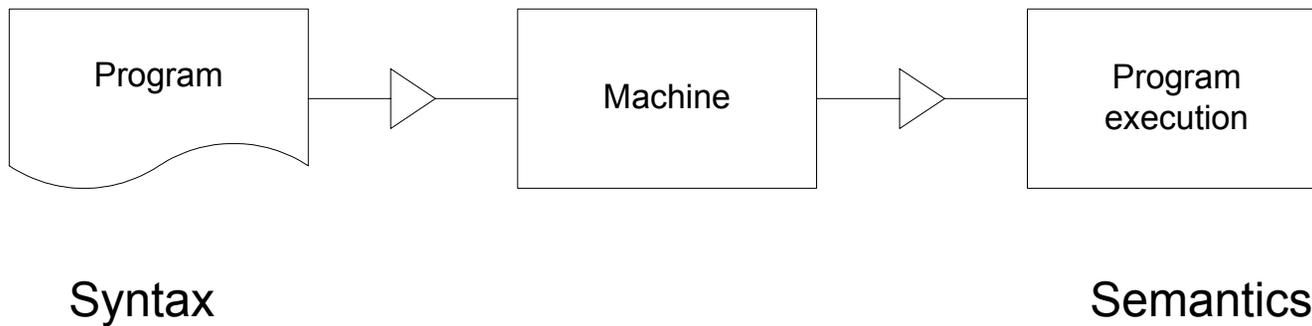




Syntax/semantics

- Program \leftrightarrow program execution
- Paradigms
- Compiling/interpretation
- Syntax
 - Classes of languages
 - Regular languages
 - Context-free languages
 - Scanning/Parsing
- Meta models

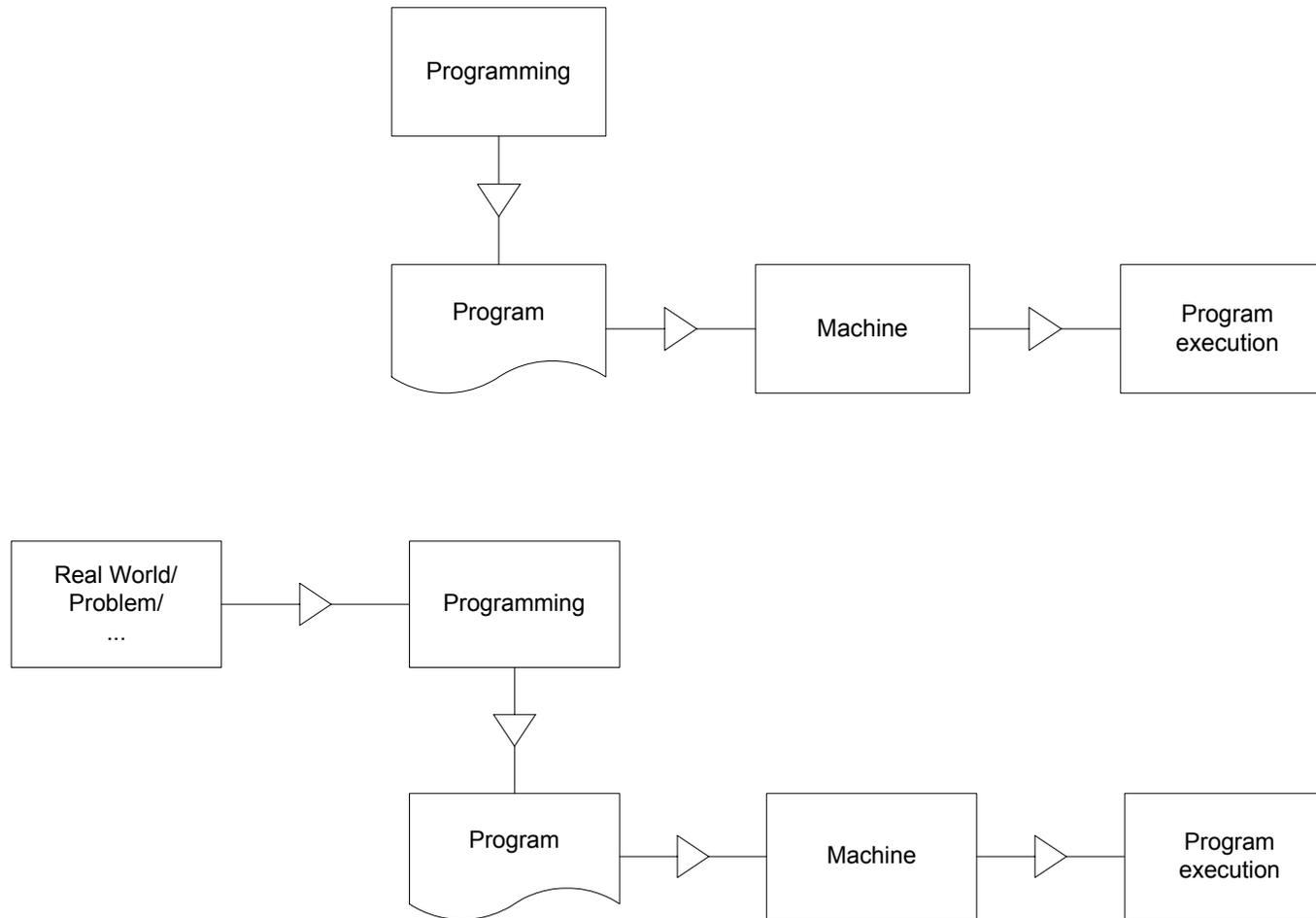
Program \leftrightarrow program execution



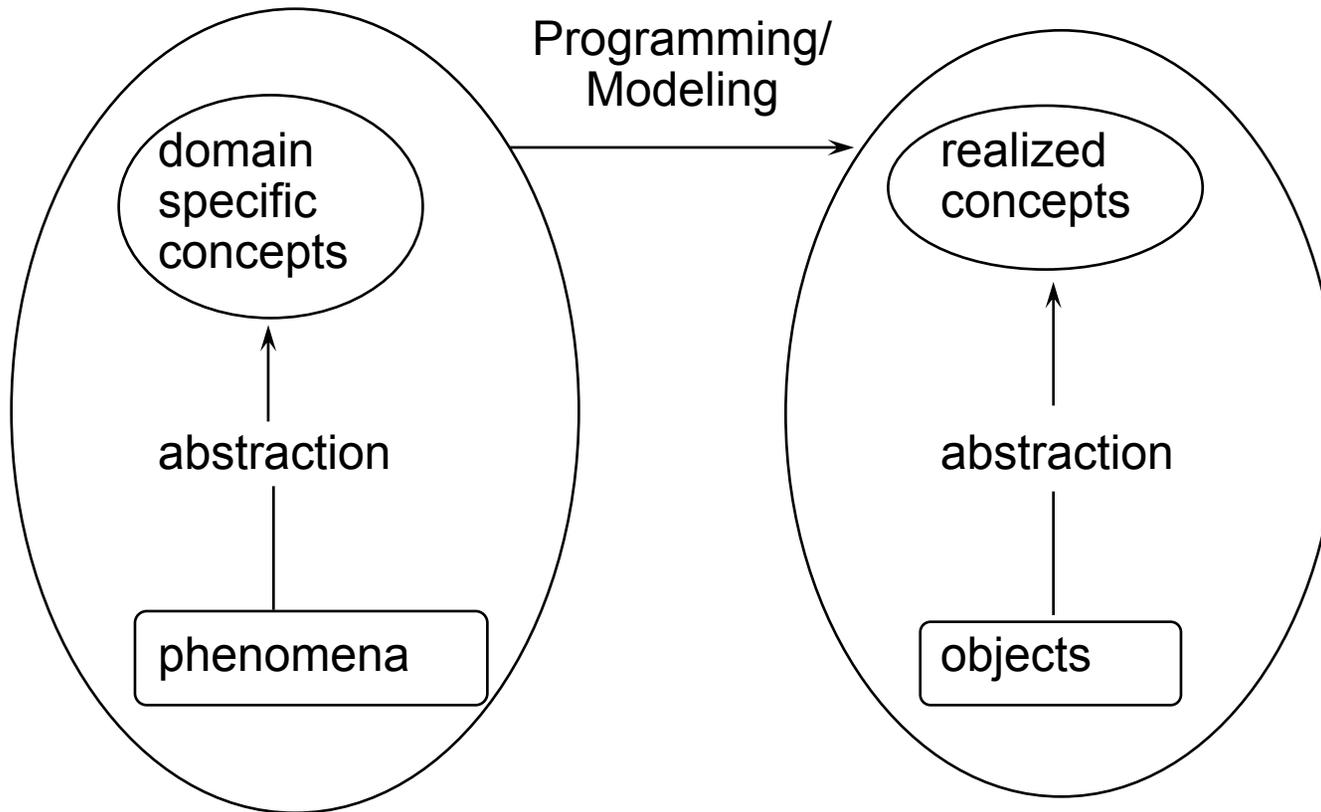
Syntaks <> Semantikk

- En beskrivelse av et (programmerings-) språk består av to hovedkomponenter:
 - *Syntaktiske* regler: hva slags *form* et lovlig program skal ha.
 - *Semantiske* regler: hva de ulike setningene i språket *betyr*.
 - *Statisk* semantikk: regler omtaler det som kan sjekkes av kompilatoren *før* selve utførelsen av programmet. F.eks.:
 - Alle variable skal være deklarerert.
 - Samsvar mellom deklarasjon og bruk av en variabel (typesjekk).
 - *Dynamisk* semantikk: regler sier hva som skal skje *under* utførelsen av et program, f.eks i form av en operasjonell semantikk, det vil si en som beskriver oppførselen til en abstrakt prosessor som utfører program skrevet i språket.

Programming/Modeling I



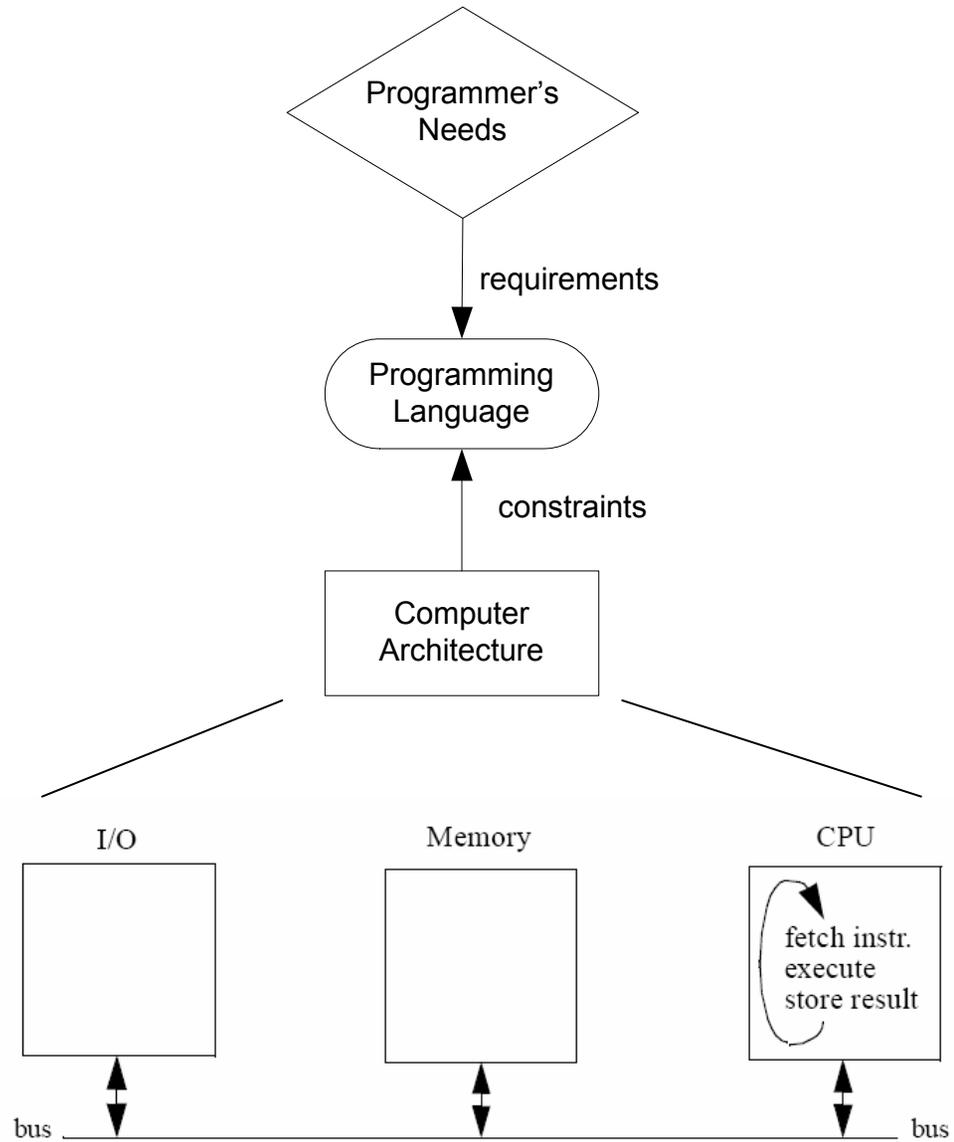
Programming/Modeling II – ex. object orientation



Natural Language/
Domain languages

Programming language/
Modeling Language

Computer Architecture and Languages

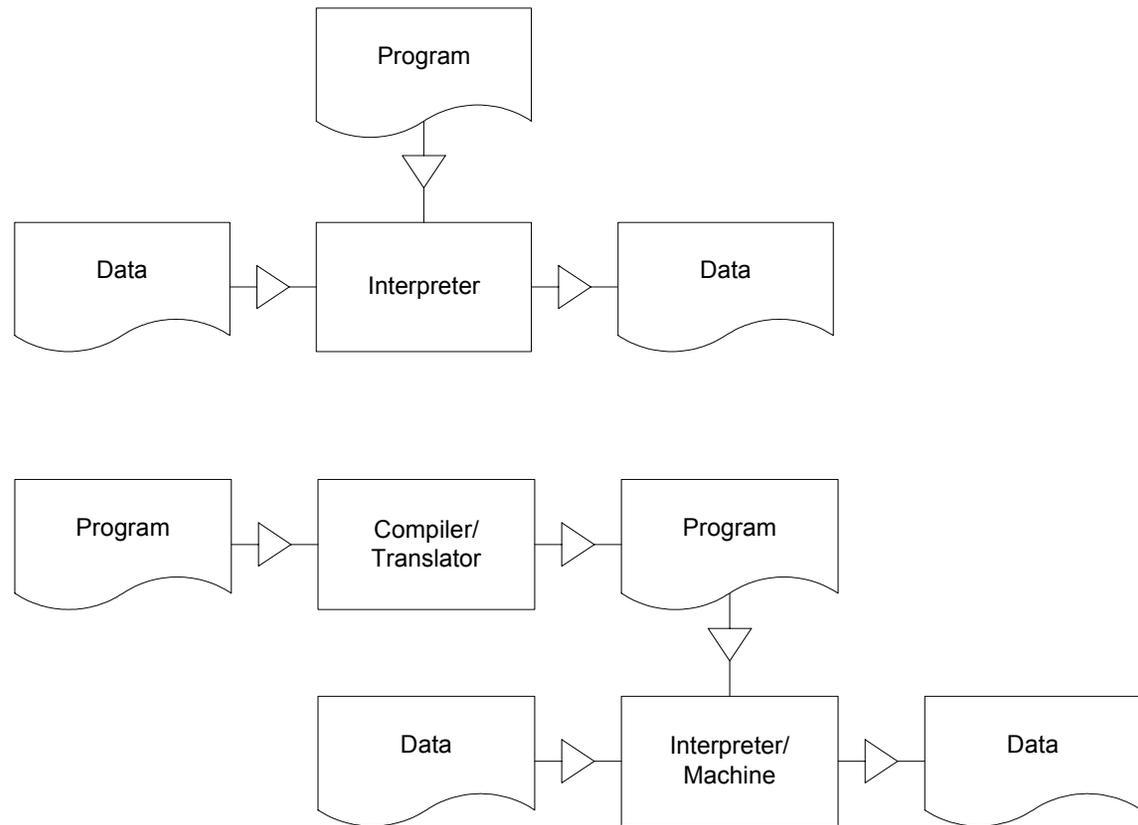


Paradigmes/perspectives

- Procedural/imperative Programming
 - A program execution is regarded as a sequence of operations manipulating a set of registers (programmable calculator)
- Functional Programming
 - A program is regarded as a mathematical function describing a relation between in-data and out-data
- Constraint-Oriented/Declarative (Logic) Programming
 - A program is regarded as a set of equations describing relations between in-data and out-data
- Object-Oriented Programming
 - A program execution is regarded as a physical model simulating a real or imaginary part of the world

Compiling/interpretation

- An interpreter reads a program and simulates its operations.
- A compiler/translator translates a program to another language, typically a machine language or to a language for a virtual machine.

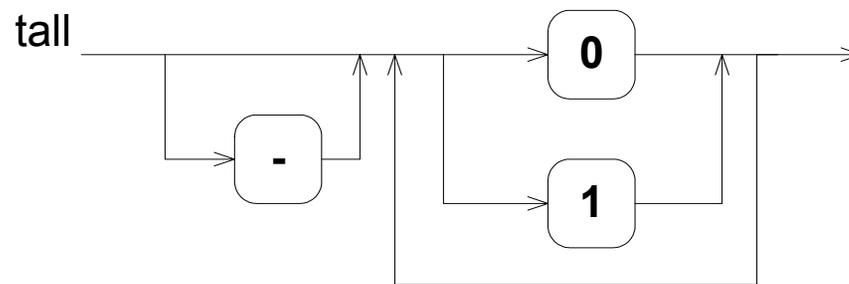


Hvordan beskrive syntaks – I

- BNF-grammatikk

<tal> → - <siffer>
<tal> → <siffer>
<siffer> → 0 <sifre>
<siffer> → 1 <sifre>
<sifre> → 0 <sifre>
<sifre> → 1 <sifre>
<sifre> → ϵ

- Syntaksdiagram ('jernbannediagram')



Utvidet BNF (Extended BNF)

- I utvidet BNF kan vi bruke følgende metasymboler i høyresiden:

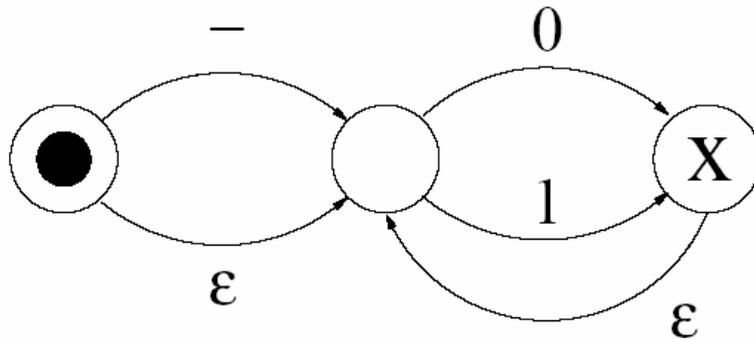
	skiller alternativer	
?	angir at noe kan forekomme	0 eller 1 gang
*	angir at noe kan forekomme	0 eller flere ganger
+	angir at noe kan forekomme	1 eller flere ganger
{...}	grupperer symboler	

<tall>	→	- <siffer> <siffer>
<siffer>	→	0 <sifre> 1 <sifre>
<sifre>	→	0 <sifre> 1 <sifre> ε

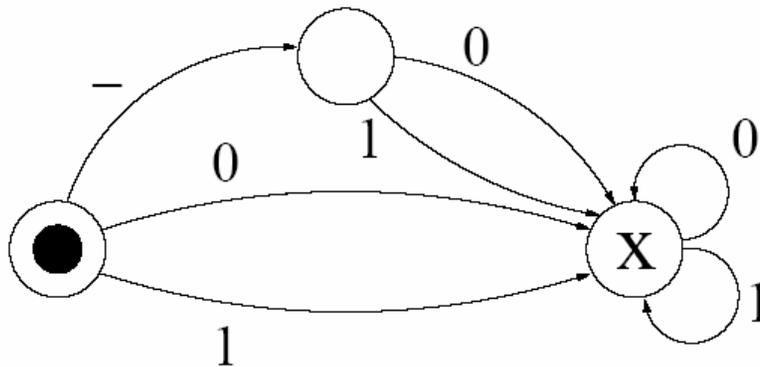
<tall>	→	{-}?{0 1}+
--------	---	------------

Hvordan beskrive syntaks – II

- Ikke-deterministiske automater

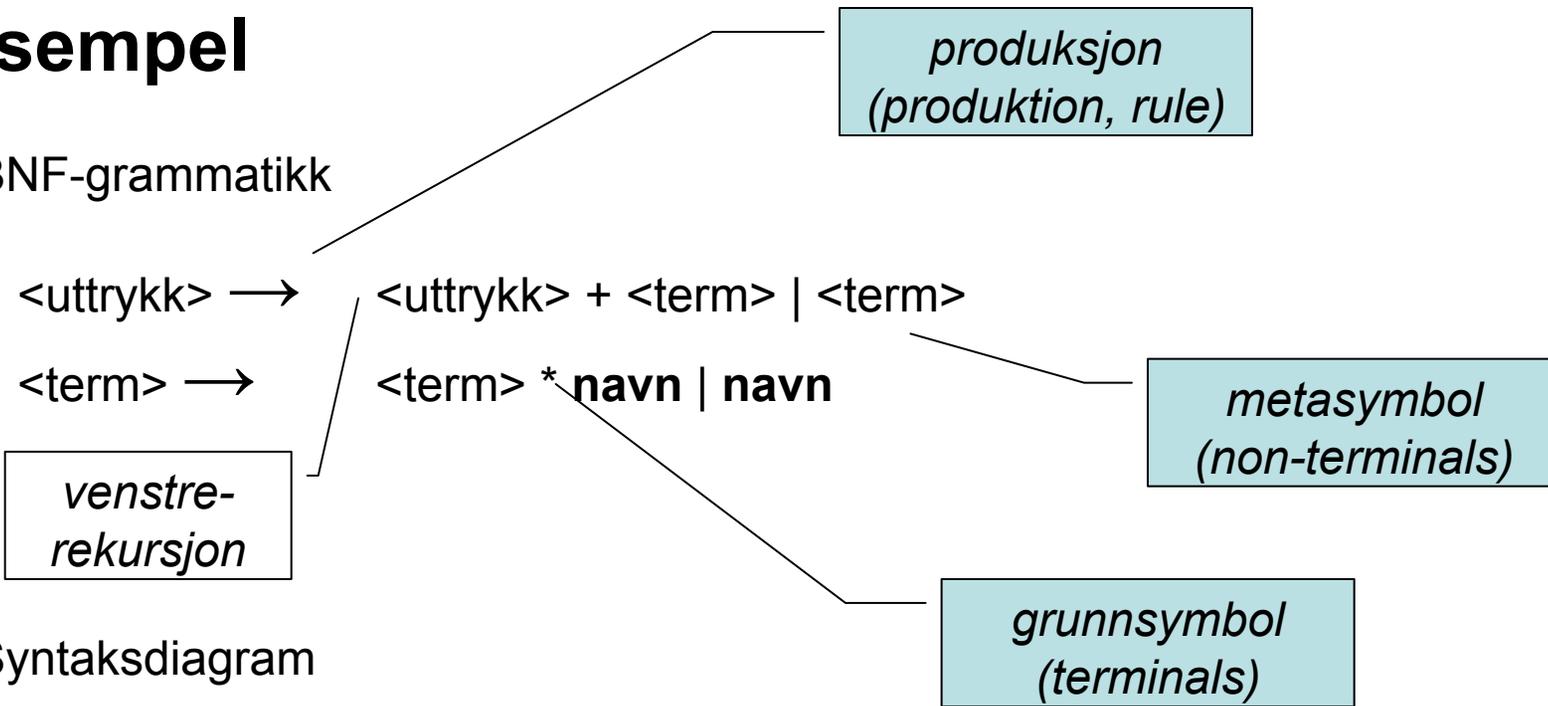


- Deterministiske automater

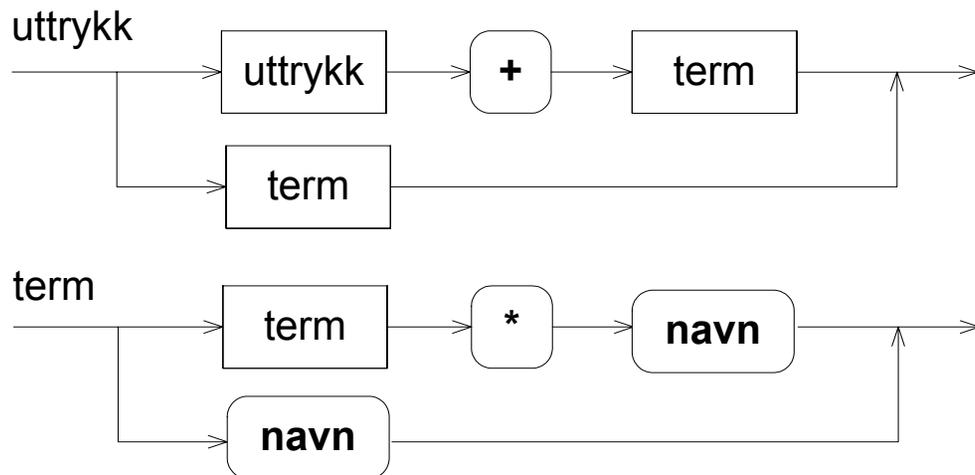


Eksempel

- BNF-grammatikk

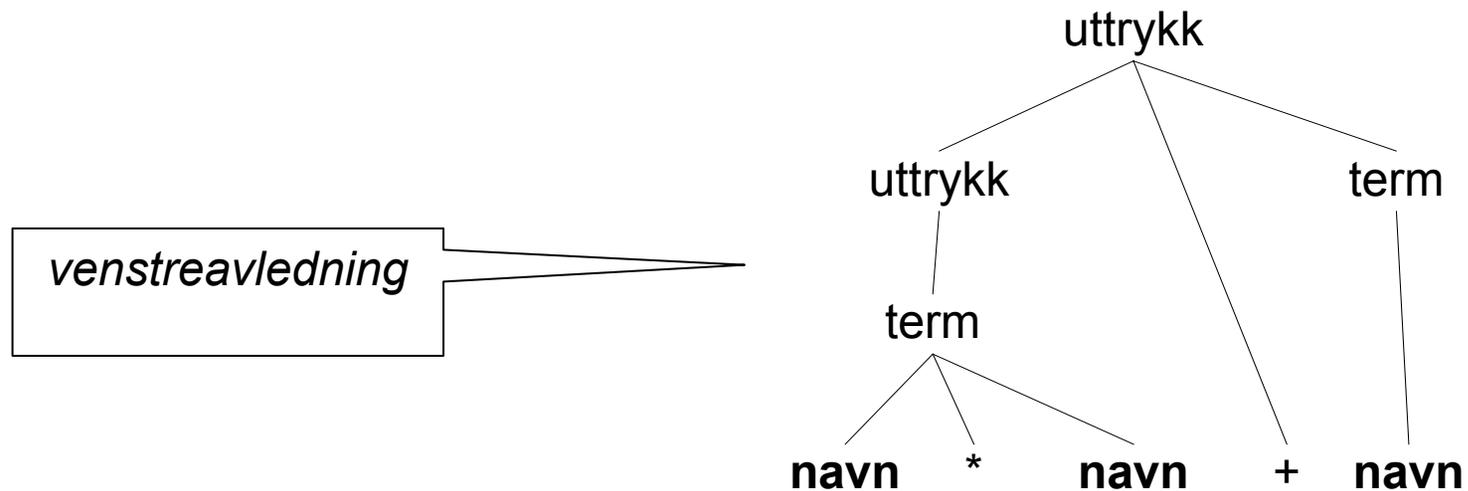


- Syntaksdiagram



Avledning av setninger (Derivation of sentences)

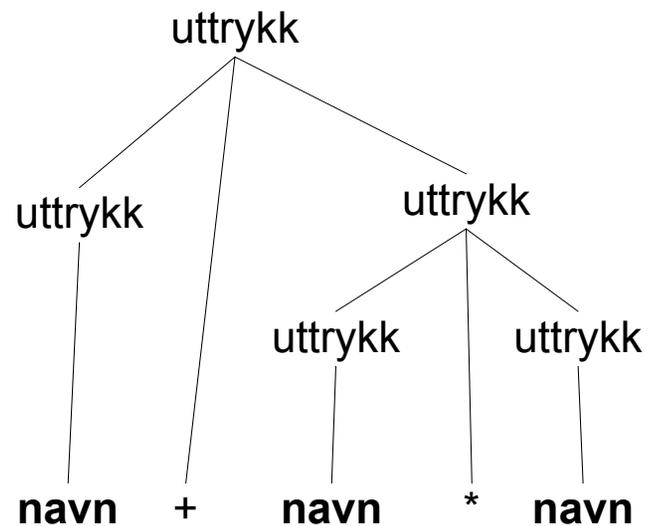
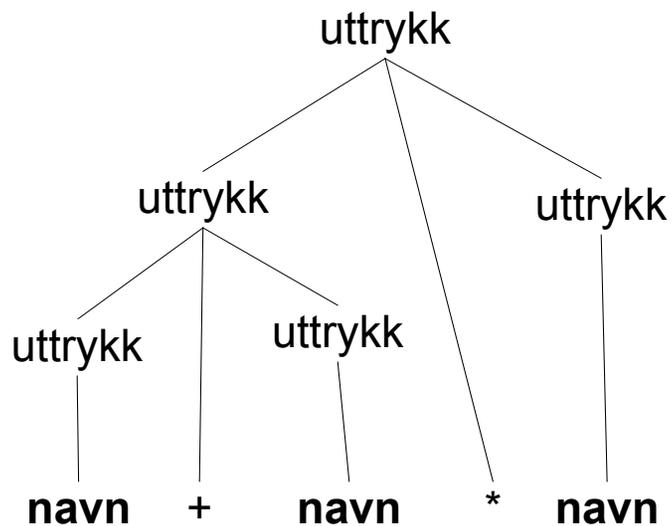
- De mulige setningene i et språk definert av en BNF-grammatikk er nøyaktig de som fremkommer ved å benytte følgende prosedyre:
 1. Start med startsymbolet.
 2. For hvert metasymbol, erstatt dette med et av alternativene på høyresiden i den tilhørende definisjonen.
 3. Gjenta forrige punkt til vi står igjen med bare grunnsymboler.
- Dette kalles for en *avledning* (*derivation*) fra startsymbolet til en ferdig setning, og kan representeres som et *syntakstre* (*syntax tree*)



Entydige/flertydige grammatikker

- Dersom enhver setning i språket kan avledes ved ett og bare ett syntakstre, er grammatikken *entydig* (*unambiguous*), ellers er den *flertydig* (*ambiguous*).

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

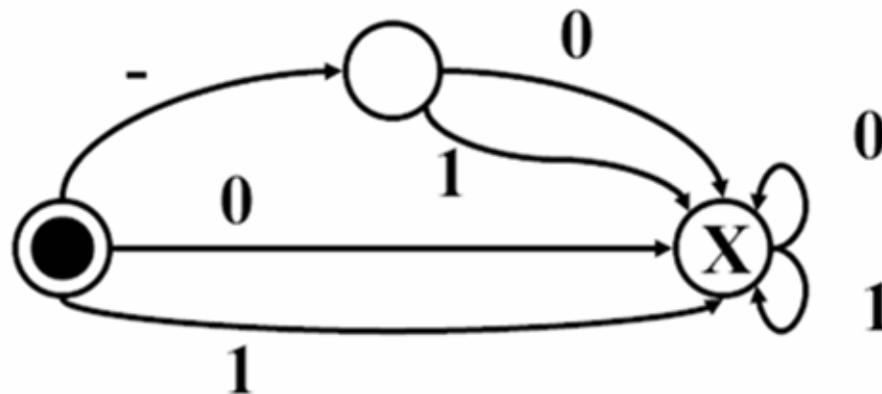


Språktyper og syntaksanalyseteknikker

- Regulære språk (type 3-språk)
 - En BNF-grammatikk med ett metasymbol på venstresiden og kun grunnsymboler på høyresiden, eventuelt med et metasymbol til slutt.
 - Kan analyseres med ikke-deterministiske og deterministiske automater
- Kontekstfrie språk (type 2-språk)
 - En BNF-grammatikk med bare ett metasymbol på venstresiden
 - Nesten alle programmeringsspråk er av denne typen
 - Kan analyseres med parsere (syntakstolkere)
- 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.

Bruk av deterministisk automat

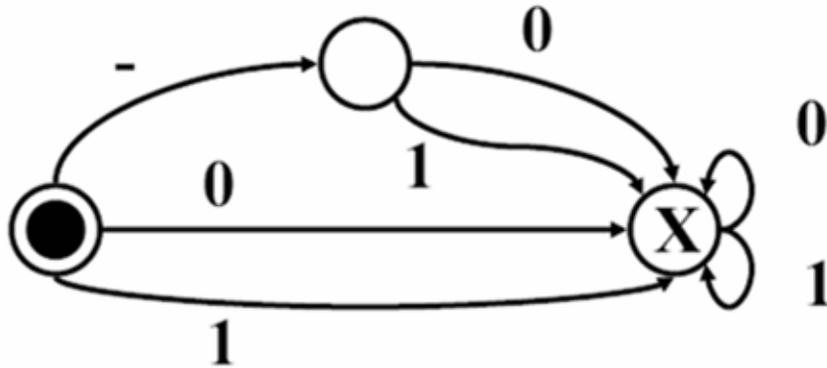
- En deterministisk automat brukes typisk for å sjekke om en gitt streng er med i det aktuelle regulære språket eller ikke:



Eksempel: **- 0 1 0**

Bruk av deterministiske automater

- Hva hvis strengen ikke er med i språket?



Eksempel: **- 0 - 1**

Hvordan lage en deterministisk automat?

- En deterministisk automat (D-automat) er lett å bruke, men ikke nødvendigvis så intuitiv å forstå eller lage.
- Fra et regulært uttrykk (evt. via et jernbandediagram) er det imidlertid ganske lett å lage en *ikke-deterministisk* automat (ID-automat).

Kan ha:

- Tomme kanter (såkalte ϵ -kanter).
- Flere kanter fra samme node med samme symbol.

- Deretter kan vi bruke en standard-algoritme for å gjøre om ID-automaten til en D-automat.

Example

$\langle \text{tall} \rangle \rightarrow 0 \langle \text{FP} \rangle \mid 1 \langle \text{IFP} \rangle$
 $\langle \text{IFP} \rangle \rightarrow 1 \langle \text{IFP} \rangle \mid 0 \langle \text{IFP} \rangle \mid \langle \text{FP} \rangle$
 $\langle \text{FP} \rangle \rightarrow \varepsilon \mid . \langle \text{EP} \rangle$
 $\langle \text{EP} \rangle \rightarrow 0 \mid 1 \mid 0 \langle \text{EP} \rangle \mid 1 \langle \text{EP} \rangle$

$\langle \text{tall} \rangle \rightarrow \{ 0 \mid 1 \{ 0 \mid 1 \}^* \} \{ . \{ 0 \mid 1 \}^+ \} ?$

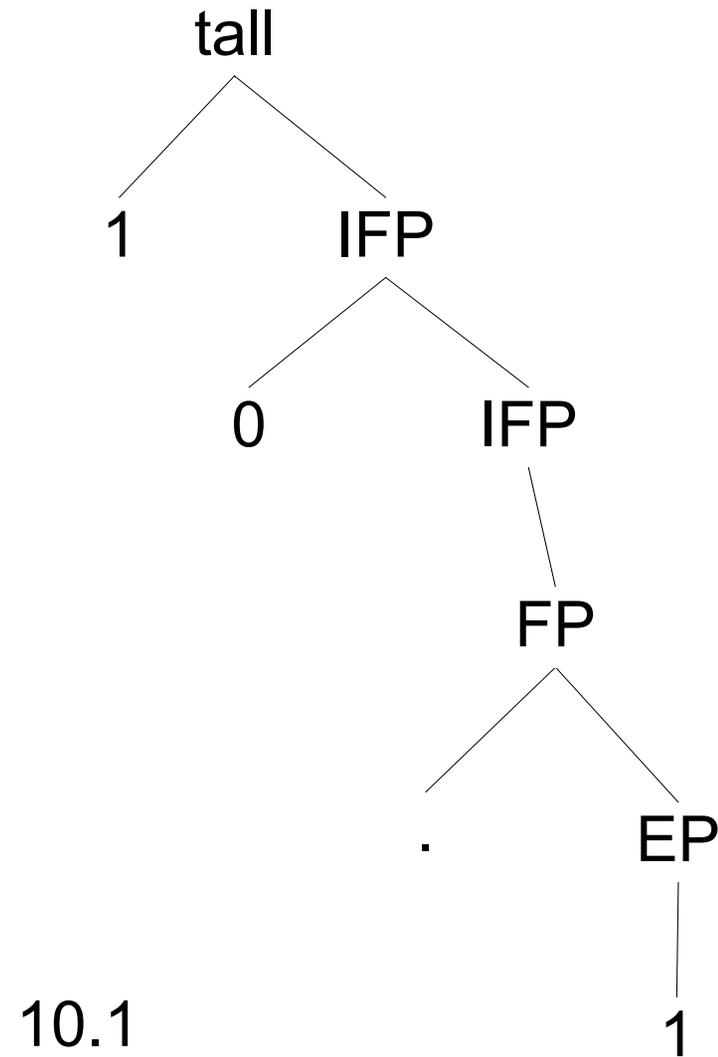
Allowed words are

0 1 101 0.10 100.1010 10.1

However, not allowed with leading 0 or
"decimalpoint" without preceding or following ciffers, so
the following is not allowed:

001 10. .01

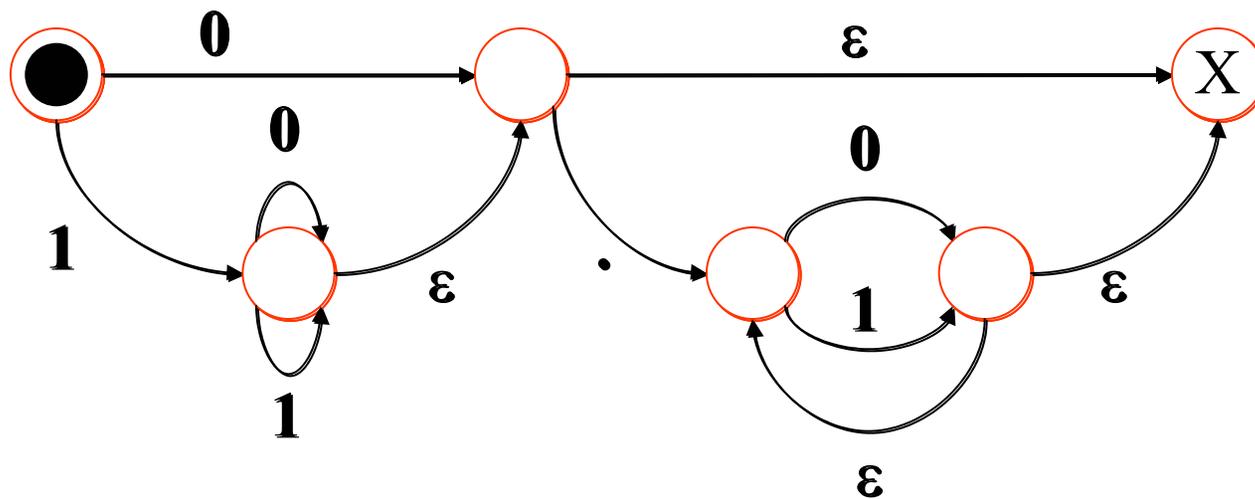
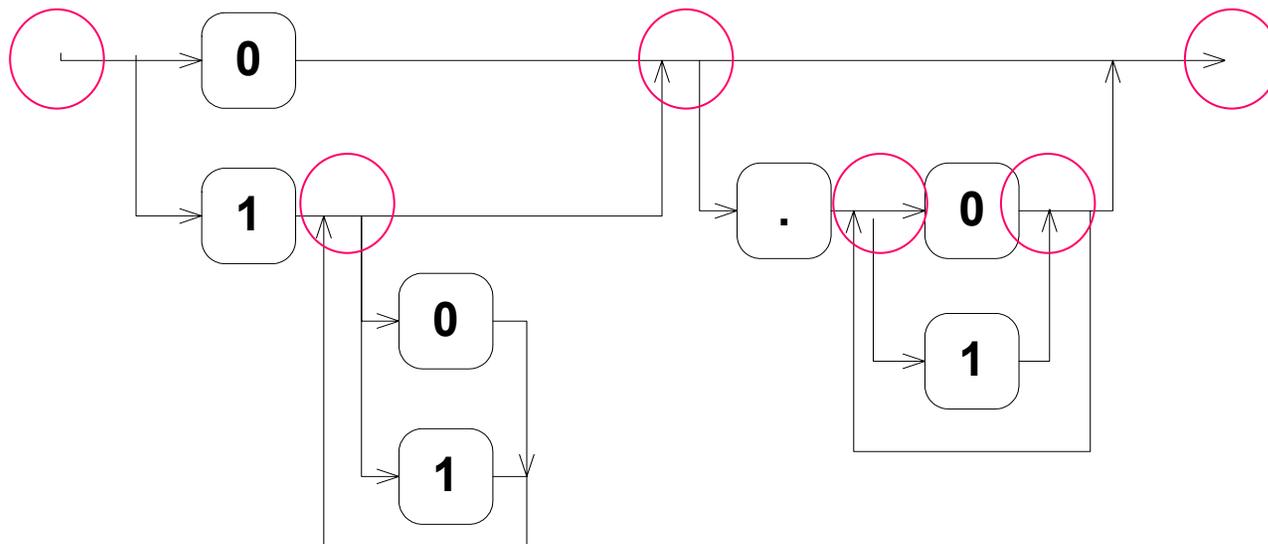
Parsetree



Fra jernbandediagram til ID-automat

1. Hver "pens" blir til en node i ID-automaten.
2. Linjene (med eventuelle symboler) blir merkede kanter mellom nodene.
Noen kanter får et tomt symbol (ε).
3. Merk nodene hvor man skal begynne og slutte.

Eksempel



As a table

	1	2	3	4	5	error
0	2	error	3	5	5	
1	3	error	3	5	5	
.	error	4	4	error	error	
end		ok			ok	

Sjekk algoritme

Gitt en slik tabell **t**, finnes det en enkel sjekk algoritme:

```
tilstand := 1;
while <flere tegn igjen> do begin
    c := <neste tegn>;
    tilstand := t(tilstand,c);
end while;
if ok(tilstand) then <match funnet>
else <ingen match>;
```

Oppsummering: Hvordan lage raske sjekkprogrammer:

- Sett opp en ikke-deterministisk automat for det regulære uttrykket**
- Lag en deterministisk automat ut fra den ikke-deterministiske automaten.**
- Sett opp tilstandstabellen t**
- Bruk søkeløkken over til å sjekke input mot uttrykket.**

Enkel forklaring på kompilering/interpretering

Ved å legge inn programsetninger for å generere kode eller kalle på rutiner får vi henholdsvis en kompilator eller en interpreter

```
tilstand := 1;
while <flere tegn igjen> do begin
    c := <neste tegn>;
    tilstand := t(tilstand,c);
    // generering av kode (kompilering) eller
    // kall på rutiner (interpretering)
end while;
if ok(tilstand) then <match funnet>
else <ingen match>;
```

Syntaksanalyse

- Skanning
- Parsering
 - "top-down"
 - "bottom-up"
- LL(1)-parsering
 - Recursive descent

- Parser:
 - Et program som analyserer en tekst i henhold til en syntaks.
 - Alle kompilatorer og interpretere inneholder en parser.
- Skanner:
 - En «preprocessor» til en parser.

Skanner

- Det er vanlig at en parser har en "preprocessor", kalt en *skanner/leksikalsk analysator*.



- Skanneren setter sammen tegn til symboler (ofte kalt *leksikalske enheter* eller *tokens*)
- Eksempel:



- Et alternativt syn at skanneren konverterer fra tastaturets alfabet til språkets alfabet
- Skanneren kan konstrueres som en deterministisk automat

Parsering

- Å sjekke at en setning (eller et program) er syntaktisk riktig, det vil si å konstruere det tilhørende syntakstreet.
- Generelt ønsker vi å kunne konstruere treet ved å lese setningen *en* gang, fra venstre mot høyre.
- Eksempelgrammatikk

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{uttrykk} \rangle + \langle \text{term} \rangle$

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{term} \rangle$

$\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \text{navn}$

$\langle \text{term} \rangle \rightarrow \text{navn}$

Ut fra denne grammatikken skal vi se på parsering av setningen:

navn * navn + navn

Top-down parsing

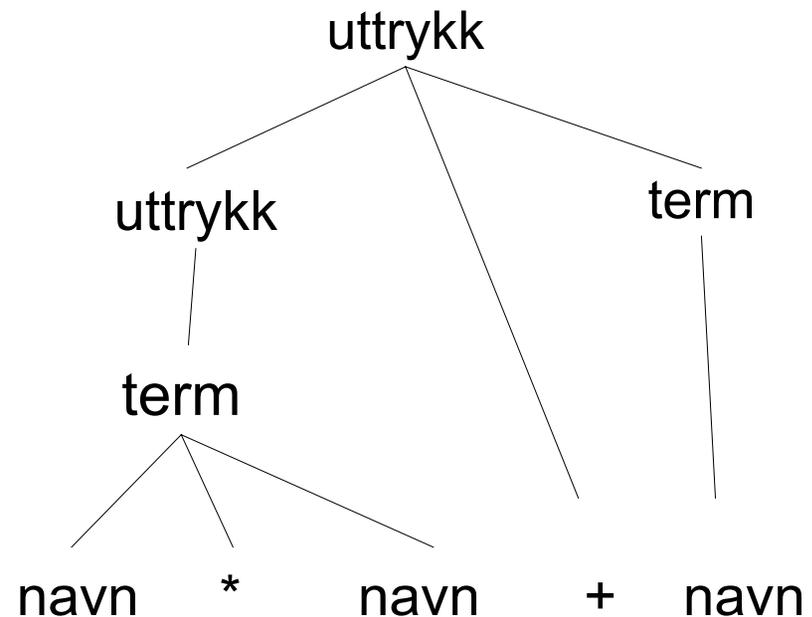
Her konstrueres treet ovenfra og ned, det vil si at vi starter med startsymbolet i roten, og så forsøker å avlede den aktuelle setningen ut fra dette:

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{uttrykk} \rangle + \langle \text{term} \rangle$

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{term} \rangle$

$\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \text{navn}$

$\langle \text{term} \rangle \rightarrow \text{navn}$



Bottom-up parsing

Her konstruerer vi treet nedenfra og opp. Vi starter da med å finne noe i setningen som tilsvarer høyresiden i en produksjon, og *reduserer* denne delsetningen til det tilhørende metasymbolet.

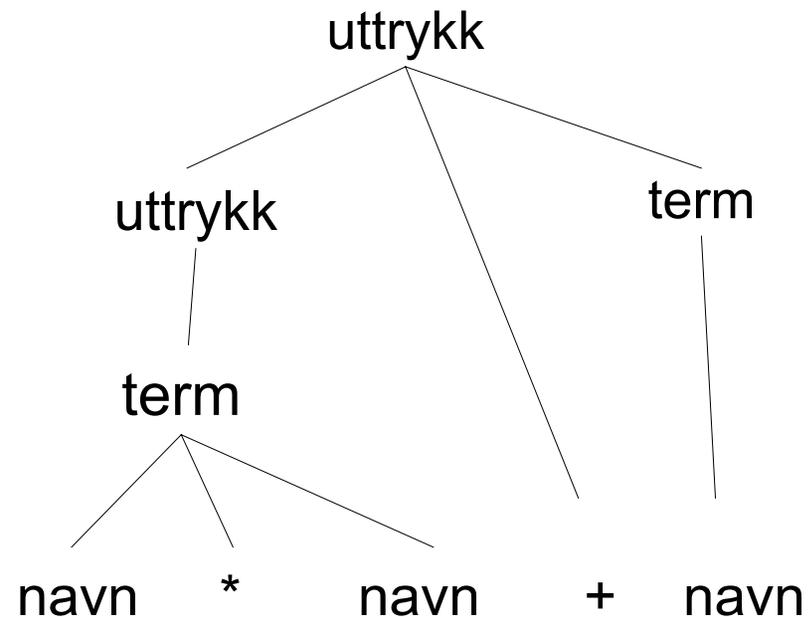
Målet er da å redusere seg tilbake til startsymbolet:

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{uttrykk} \rangle + \langle \text{term} \rangle$

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{term} \rangle$

$\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \text{navn}$

$\langle \text{term} \rangle \rightarrow \text{navn}$



LL(1)-parsering

- LL(1)-parsering er en top-down strategi der vi foretar en *venstrevledning* fra startsymbolet.
- Recursive descent
 - Til hvert metasymbol svarer en metode.
 - Metoden tar seg av det ene metasymbolet, men kan kalle andre metoder.
 - For hvert *grunnsymbol* i høyresiden: Sjekk at symbolet i skanneren er lik dette grunnsymbolet.
 - For hvert *metasymbol* i høyresiden: Kall metasymbolets metode.
 - Når metoden kalles, skal skanneren inneholde første symbol i den aktuelle produksjonen (for syntaktisk korrekt setning).
 - Når metoden er ferdig, skal skanneren inneholde første symbol etter den leste teksten.

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{uttrykk} \rangle + \langle \text{term} \rangle$
 $\langle \text{uttrykk} \rangle \rightarrow \langle \text{term} \rangle$
 $\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \text{navn}$
 $\langle \text{term} \rangle \rightarrow \text{navn}$

```
static void uttrykk() {  
    uttrykk();  
    readSymbol('+');  
    term();  
}
```

LL(1)-grammatikker

- Vi kan ikke lage en "recursive descent"-parser for alle grammatikker, de må oppfylle følgende krav:
 - Grammatikken må være kontekstfri, dvs. kun ha ett metasymbol på venstre side.
 - Under parseringen må vi vite hvilket alternativ vi skal velge; dvs. at mengden av startsymboler (den *utvidede startmengden*) for hvert alternativ må være disjunkte.
- En LL(1)-grammatikk er en grammatikk som oppfyller dette, og slik at vi hele tiden kan se på "neste" symbol (men ikke mer) for å bestemme hvilket alternativ som skal velges.

Hva hvis en grammatikk ikke er LL(1)?

To standard-teknikker for å skrive om en grammatikk til å bli LL(1).

- Fjerning av venstrerekursjon
- Ikke-disjunkte startmengder

Fjerning av venstre-rekursjon I

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{uttrykk} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle$

- Strategi:

1. Skriv om til utvidet BNF:

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{term} \rangle \{ + \langle \text{term} \rangle \}^*$

2. Før tilbake til klassisk BNF, men nå med høyrerekursjon (og eventuelt ekstra metasymboler):

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{term} \rangle \langle \text{xterm} \rangle$
 $\langle \text{xterm} \rangle \rightarrow + \langle \text{term} \rangle \langle \text{xterm} \rangle \mid \epsilon$

```
static void uttrykk() {  
    term();  
    xterm();  
}
```

```
static void xterm() {  
    readSymbol('+');  
    term();  
    xterm();  
}
```

Fjerning av venstre-rekursjon II

$\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \text{navn} \mid \text{navn}$

- Strategi:

1. Skriv om til utvidet BNF:

$\langle \text{term} \rangle \rightarrow \text{navn} \{ * \text{navn} \}^*$

2. Før tilbake til klassisk BNF, men nå med høyrerekursjon (og eventuelt ekstra metasymboler):

$\langle \text{term} \rangle \rightarrow \text{navn} \langle \text{xnavn} \rangle$
 $\langle \text{xnavn} \rangle \rightarrow * \text{navn} \langle \text{xnavn} \rangle \mid \varepsilon$

```
static void term() {  
    read(navn);  
    xnavn();  
}
```

```
static void xnavn() {  
    readSymbol('*');  
    read(navn);  
    xnavn();  
}
```

Entydig?

$\langle \text{uttrykk} \rangle \rightarrow \langle \text{term} \rangle \langle \text{xterm} \rangle \quad (1)$

$\langle \text{xterm} \rangle \rightarrow + \langle \text{term} \rangle \langle \text{xterm} \rangle \mid \varepsilon \quad (2)$

$\langle \text{term} \rangle \rightarrow \mathbf{navn} \langle \text{xnavn} \rangle \quad (3)$

$\langle \text{xnavn} \rangle \rightarrow * \mathbf{navn} \langle \text{xnavn} \rangle \mid \varepsilon \quad (4)$

$\langle \text{uttrykk} \rangle$

$\rightarrow \langle \text{term} \rangle \langle \text{xterm} \rangle \quad (1)$

$\rightarrow \mathbf{navn} \langle \text{xnavn} \rangle \langle \text{xterm} \rangle \quad (3)$

$\rightarrow \mathbf{navn} * \mathbf{navn} \langle \text{xnavn} \rangle \langle \text{xterm} \rangle \quad (4.1)$

$\rightarrow \mathbf{navn} * \mathbf{navn} \langle \text{xterm} \rangle \quad (4.2)$

$\rightarrow \mathbf{navn} * \mathbf{navn} + \langle \text{term} \rangle \langle \text{xterm} \rangle \quad (2)$

$\rightarrow \mathbf{navn} * \mathbf{navn} + \mathbf{navn} \langle \text{xnavn} \rangle \langle \text{xterm} \rangle \quad (3)$

$\rightarrow \mathbf{navn} * \mathbf{navn} + \mathbf{navn} \langle \text{xterm} \rangle \quad (4.2)$

$\rightarrow \mathbf{navn} * \mathbf{navn} + \mathbf{navn} \quad (2.1)$

Ikke-disjunkte startmengder

Ofte vil to alternative startsider kunne begynne på samme måte (ha overlappende utvidede startmengder), men forskjellig avslutning, som f.eks.:

$$\langle \text{setning} \rangle \rightarrow \langle \text{uttrykk} \rangle + \langle \text{term} \rangle \mid \langle \text{uttrykk} \rangle * \langle \text{term} \rangle$$

Vi kan her innføre et nytt metasymbol som skal stå for den varierende delen:

$$\begin{aligned} \langle \text{setning} \rangle &\rightarrow \langle \text{uttrykk} \rangle \langle \text{xsetning} \rangle \\ \langle \text{xsetning} \rangle &\rightarrow + \langle \text{term} \rangle \mid * \langle \text{term} \rangle \end{aligned}$$

Example

<program> → <stmtList>

<stmtList> → <stmt> +

<stmt> → <input> | <output> | <assignment>

<input> → ? <variable>

<output> → ! <variable>

<assignment> → <variable> = <variable> <operator> <operand>

<operator> → + | -

<operand> → <variable> | <number>

<variable> → v <ciffer>

<ciffer> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<number> → <ciffer> +

```
static void assignment() {  
    variable();  
    readSymbol('=');  
    variable();  
    operator();  
    operand();  
}
```

<program> → <stmtList>
<stmtList> → <stmt> +
<stmt> → **<input>** | **<output>** | **<assignment>**
<input> → ? <variable>
<output> → ! <variable>
<assignment> → <variable> = <variable> <operator> <operand>
<operator> → + | -
<operand> → <variable> | <number>
<variable> → v <ciffer>
<ciffer> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<number> → <ciffer> +

```
static void stmt() {  
    if (checkSymbol('v')) {  
        assignment();  
    } else if (checkSymbol('?')) {  
        input();  
    } else if (checkSymbol('!')) {  
        output();  
    }  
}
```

Beregning av utvidede startmengder

- For å beregne de utvidede startmengdene må vi først ha beregnet tre andre mengder:
 - *Startmengden* til et metasymbol:
Mengden av de grunnsymbolene som kan stå først i en (del-) setning avledet fra dette metasymbolet.
 - *Etterfølgermengden* til et metasymbol:
Mengden av de grunnsymboler som kan stå etter metasymbolet på et tidspunkt i avledningen.
 - *Meta-til-tom-mengden*:
Mengden av de metasymboler som kan videreavledes til den tomme setning.

Meta-til-tom-mengden

- Metasymboler som kan produsere den tomme setningen kompliserer ting litt, så vi beregner meta-til-tom-mengden først:
 1. Initialisering:
La meta-til-tom-mengden bestå av de metasymboler som har en tom høyreside.
 2. Dersom det finnes et metasymbol med en høyreside som bare består av metasymboler som er med i meta-til-tom-mengden, skal dette også inkluderes i mengden.
 3. Gjenta inntil meta-til-tom-mengden ikke øker mer.

Startmengder

- For enkelthets skyld sier vi at grunnsymboler har seg selv som startmengde.
- Initialisering: La startmengden for hvert metasymbol være de grunnsymbolene som står først i en høyreside for dette metasymbolet.
 - Se på en høyreside for et metasymbol M . For hvert (grunn- eller meta-) symbol x i høyresiden som enten står først, eller bare har metasymboler fra meta-til-tom-mengden foran, øk startmengden til M med startmengden til x .
 - Gjenta inntil ingen av startmengdene kan økes mer.

Etterfølgermengder

1. Initialisering:

For hvert metasymbol, la etterfølgermengden være unionen av startmengdene til de (grunn- eller meta-) symbolene som i en eller annen høyreside står enten

- umiddelbart bak dette metasymbolet, eller
- bak slik at de mellomliggende symboler er metasymboler i meta-til-tom mengden.

2. Hvis metasymbolet M i en høyreside til metasymbolet N enten står bakerst, eller det bak M bare står metasymboler i meta-til-tom-mengden: øk etterfølgermengden til M med etterfølgermengden til N.

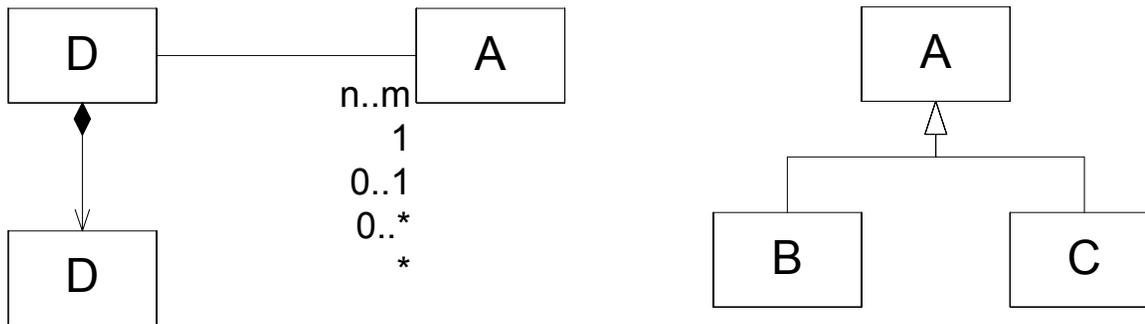
3. Gjenta til ingen av etterfølgermengdene kan økes mer.

Utvidede startmengder

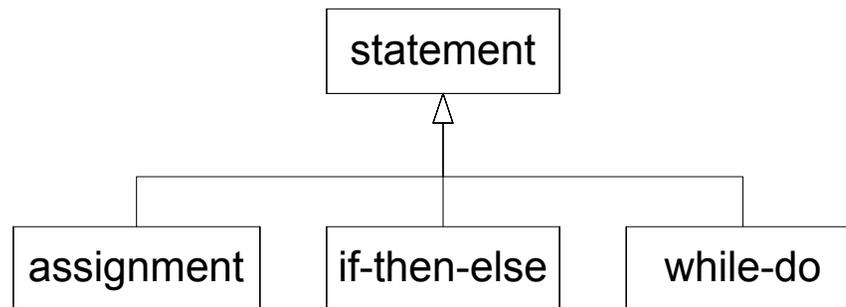
- For en gitt produksjon består denne mengden av alle symbolene i startmengden til de symbolene i høyresiden som enten står først eller bare har metasymboler i meta-til-tom-mengden foran seg.
- Hvis høyresiden er tom eller bare består av meta-til-tom-mengden, inngår også symbolene i etterfølgermengden til produksjonens venstreside.

Meta models

- Alternative to grammars and syntax trees
- Object model representing the program (*not* the execution)



`<statement> → <assignment> | <if-then-else> | <while-do>`



Why meta models?

- Inspired by abstract syntax trees in terms of object structures, interchange formats between tools
- Not all modeling/programming tools are parser-based (e.g. wizards)
- Growing interest in domain specific languages, often with a mixture of text and graphics