Transleerimismeetodid

Varmo VENE
Arvutiteaduse Instituut
Tartu Ülikool

EMAIL: varmo@cs.ut.ee
WWW: http://www.cs.ut.ee/~varmo/TM2002/
LIST: ati.pk@lists.ut.ee

Interpretaator

Eelised: Ei toimu programmiteksti eeltöötlemist

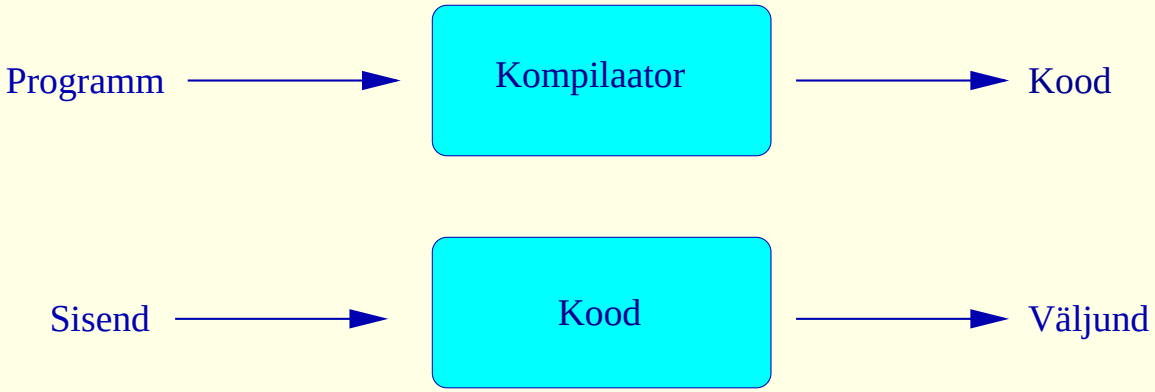
⇒ algkäivituseks kulub vähe aega;

⇒ soodustab interaktiivset programmide kirjutamist.

Puudused: Programmi osasid analüüsitakse täitmisajal korduvalt

⇒ programmide täitmine suhteliselt aeglane.

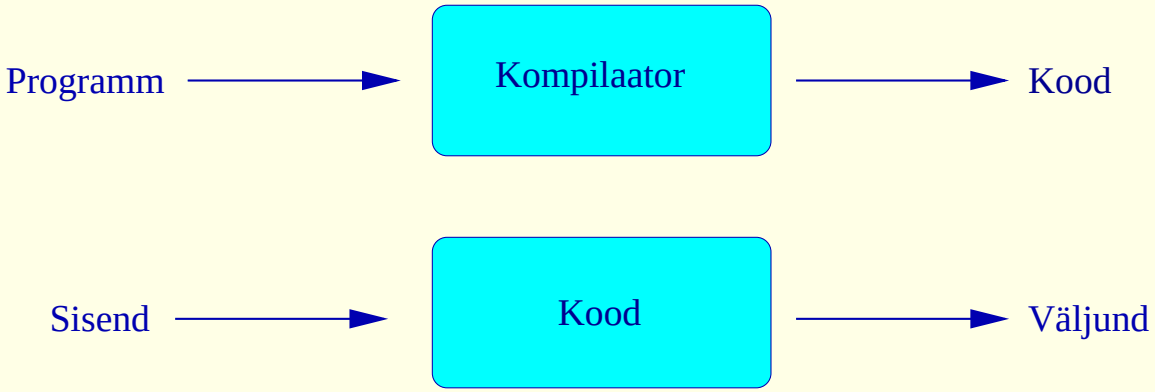
Kompilaator



Kaks faasi:

- Lähteprogramm transleeritakse täidetavaks (masin-)koodiks (kompileerimisaeg)
- Koodi täitmine antud sisendandmetel (täitmisaeg)

Kompilaator



Eelised: Programmi analüüsitakse ainult üks kord

⇒ võimaldab programmide (globaalset) optimeerimist;

⇒ programmide täitmine on kiirem.

Puudused: Kompileerimine võtab aega

⇒ kompileerimine tasub ära pikalt töötavate ja tihti kasutatavate programmide korral.

Süntaksjuhitav kompileerimine

- Alates Algol 60st, mis oli esimene formaalselt defineeritud süntaksiga keel, juhindub kompilaatori poolt teostatav transleerimisprotsess lähteprogrammi süntaktilisest struktuurist.
- Programmeerimiskeele definitsioon koosneb:

  leksika: defineerib keeles legaalsete sõnade, *lekseemide*, moodustamise reeglid;

  süntaks: defineerib programmide grammatilise struktuuri;

  semantika: kirjeldab kontekstist sõltuvaid tingimusi (näit. tüüpimisreeglid) ja programmi tähendust.

Kompilaatori struktuur

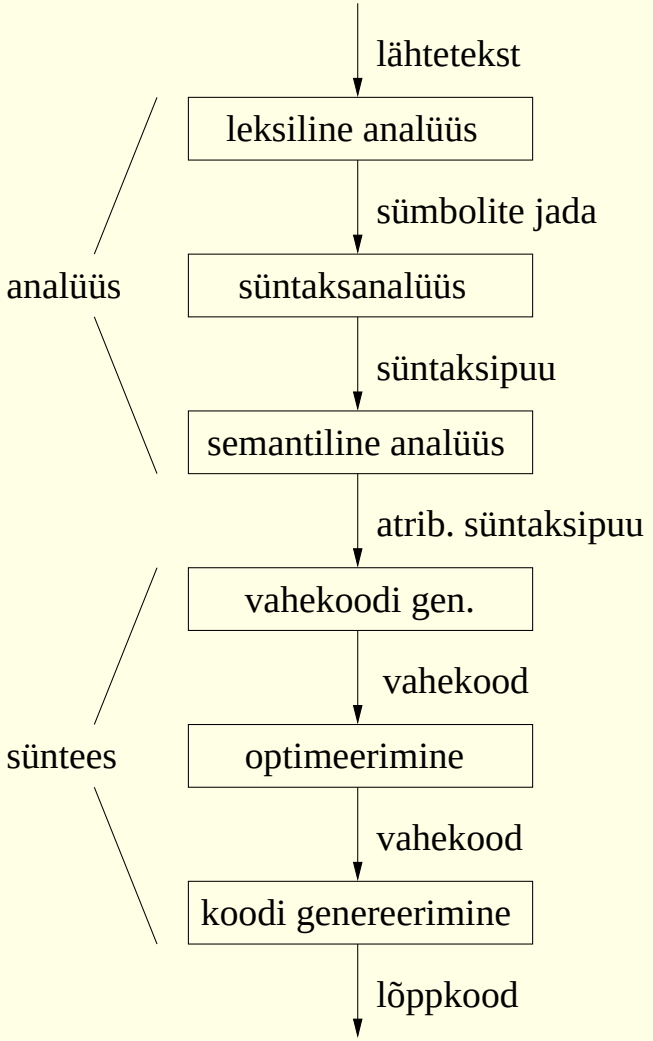
- Kompileerimisprotsess koosneb suures plaanis kahest faasist:

  analüüs: lähteteksti struktuuri äratundmine vastavalt grammatikale ja kontekstuaalsete sõltuvuste kontrollimine;

  süntees: koodi genereerimine ja optimeerimine.

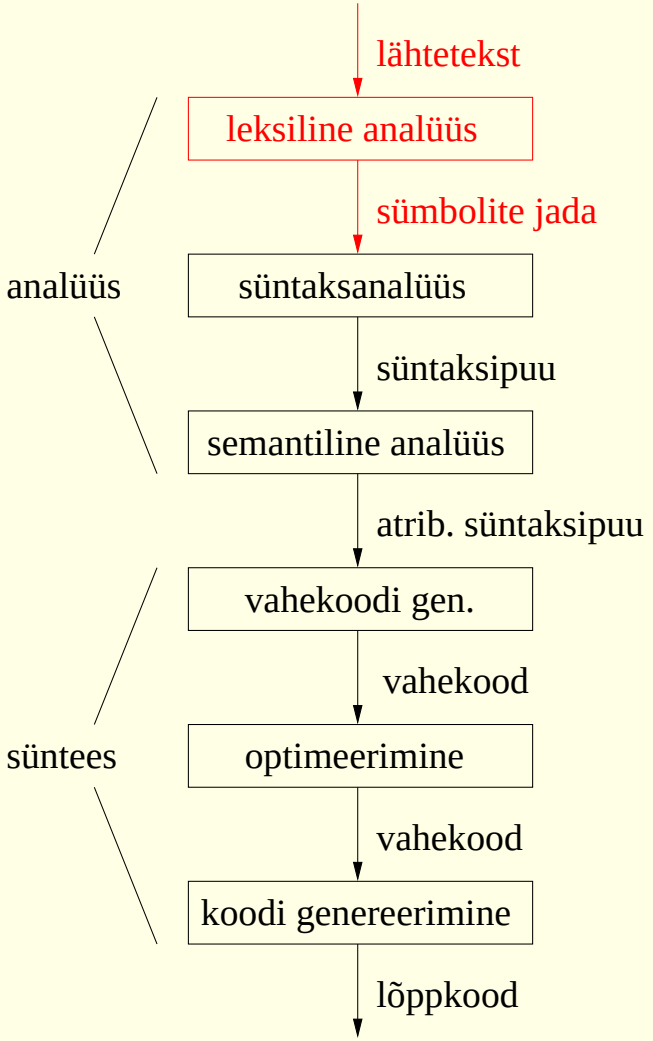
- Need omakorda jagunevad mitmeteks alamfaasideks.

- NB! Erinevates faasides teostatavad toimingud võivad toimuda paralleelselt.



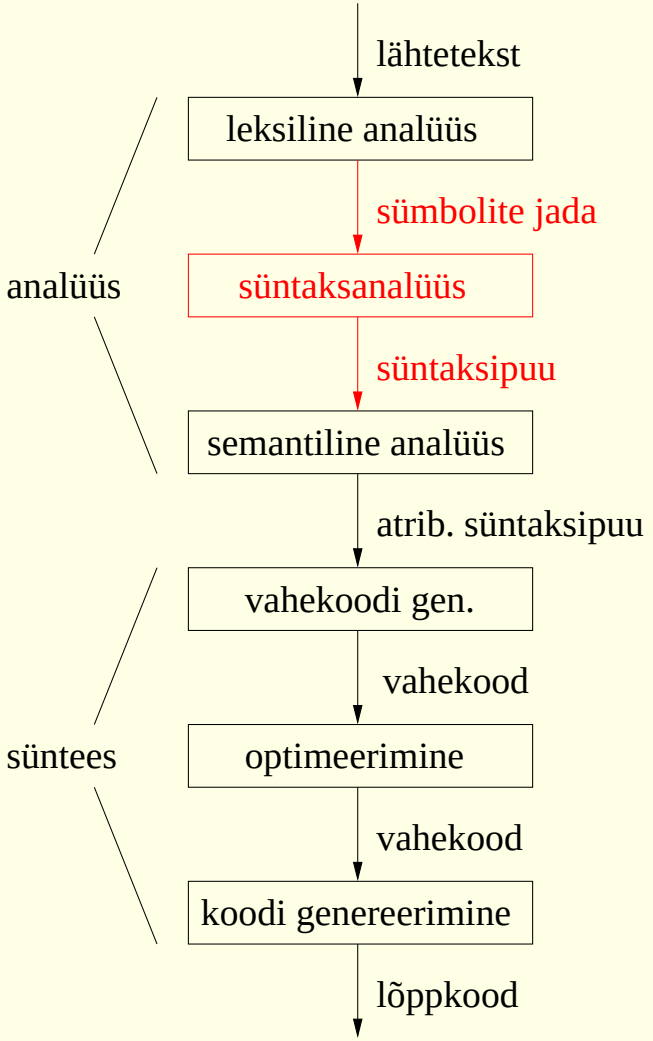
Kompilaatori struktuur

- *Leksiline analüüs* kontrollib programmi sõnade (literaalsümbolite) vastavust leksilistele reeglitele, eemaldab tühisümbolid ja kommentaarid ning teisendab programmi sümbolite (*tokens*) jadaks.
- Leksilist analüüsi kutsutakse *skaneerimiseks* ning vastavat analüsaatorit nimetatakse *skanneriks*.



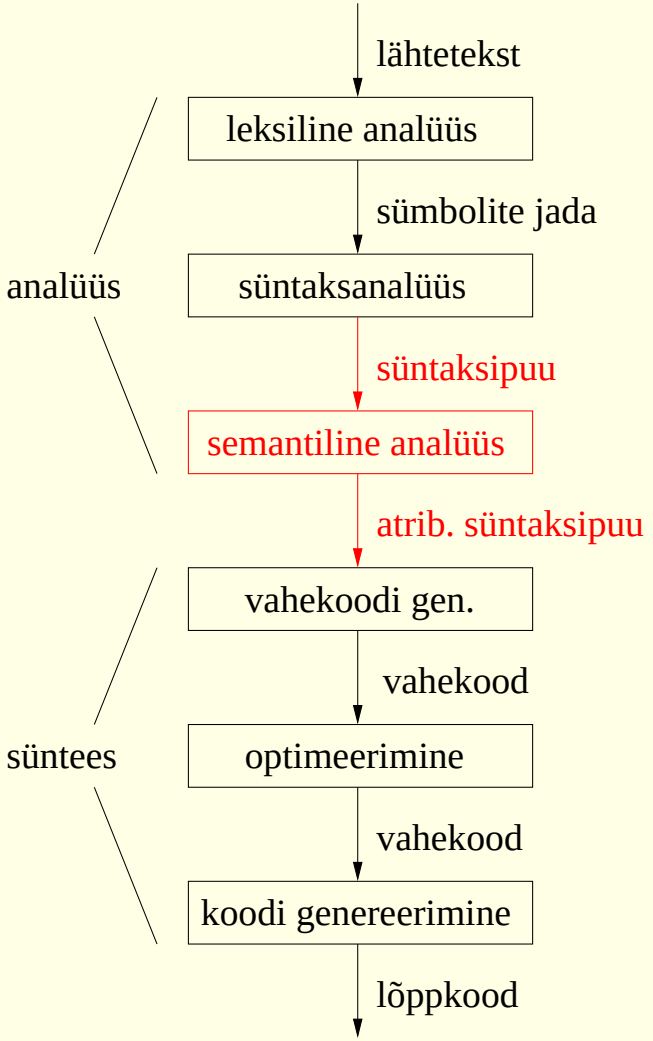
Kompilaatori struktuur

- *Süntaksanalüüs* kontrollib programmi struktuuri vastavust keele grammatikale ning väljastab programmi esitava (abstraktse) süntaksipuu.
- Süntaksanaüüsi kutsutakse *parsimiseks* ning vastavat analüsaatorit nimetatakse *parseriks*.



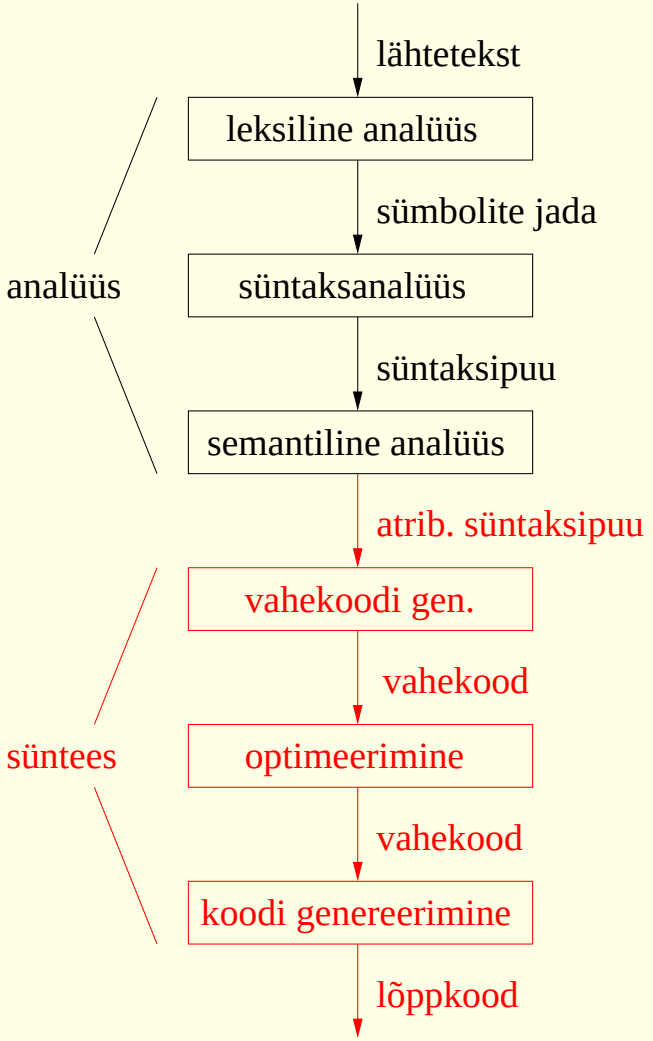
Kompilaatori struktuur

- *Semantiline analüüs* kontrollib programmi kontekstuaalsete sõltuvuste korrektsust, leiab vastavuse defineerivate ja kasutusesinemiste vahel, leiab esinemiste tüübid ja kontrollib nende vastavust reegilitele . . .
- Süntaksipuu dekoreeritakse tüübi ja muu kontekstist sõltuva infoga.

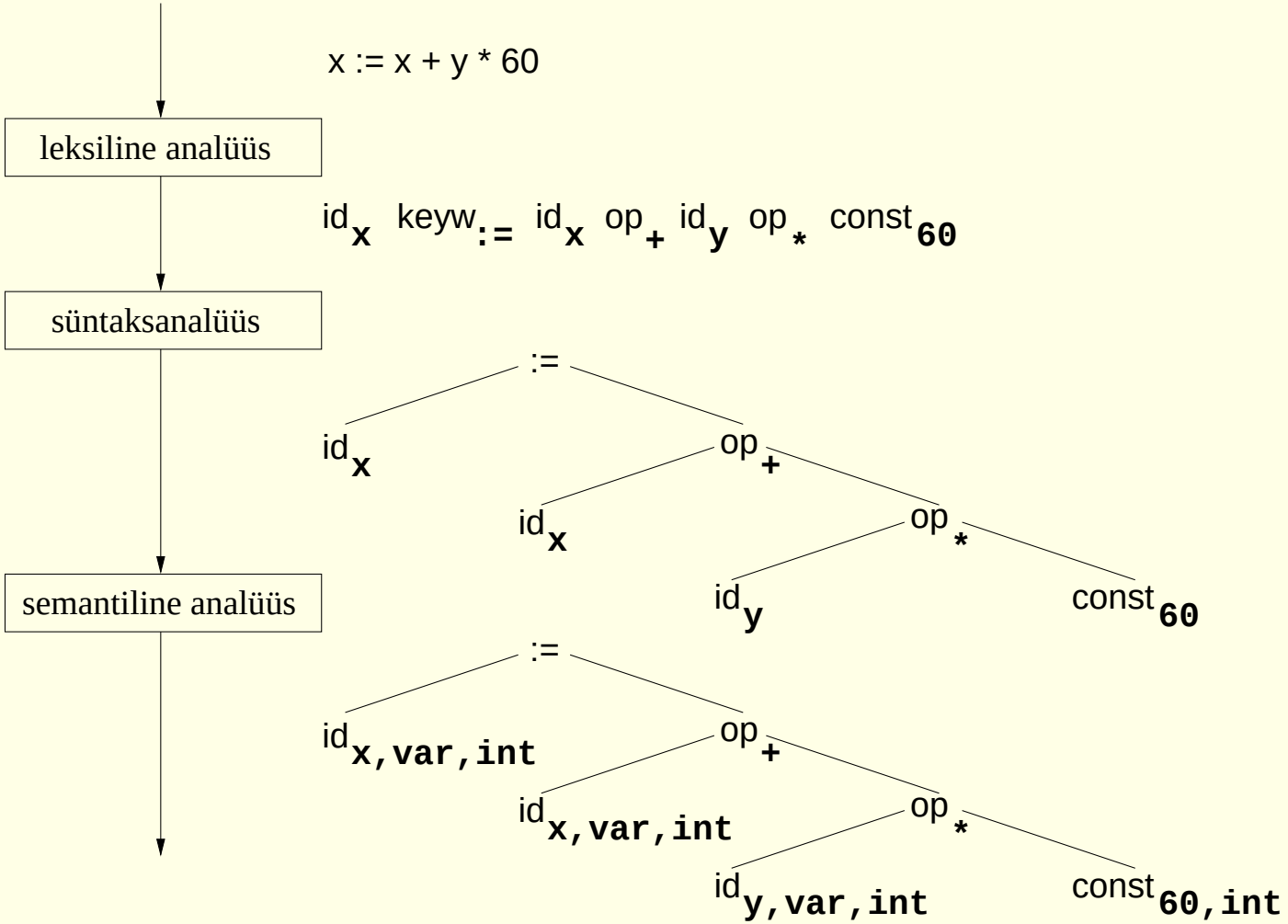


Kompilaatori struktuur

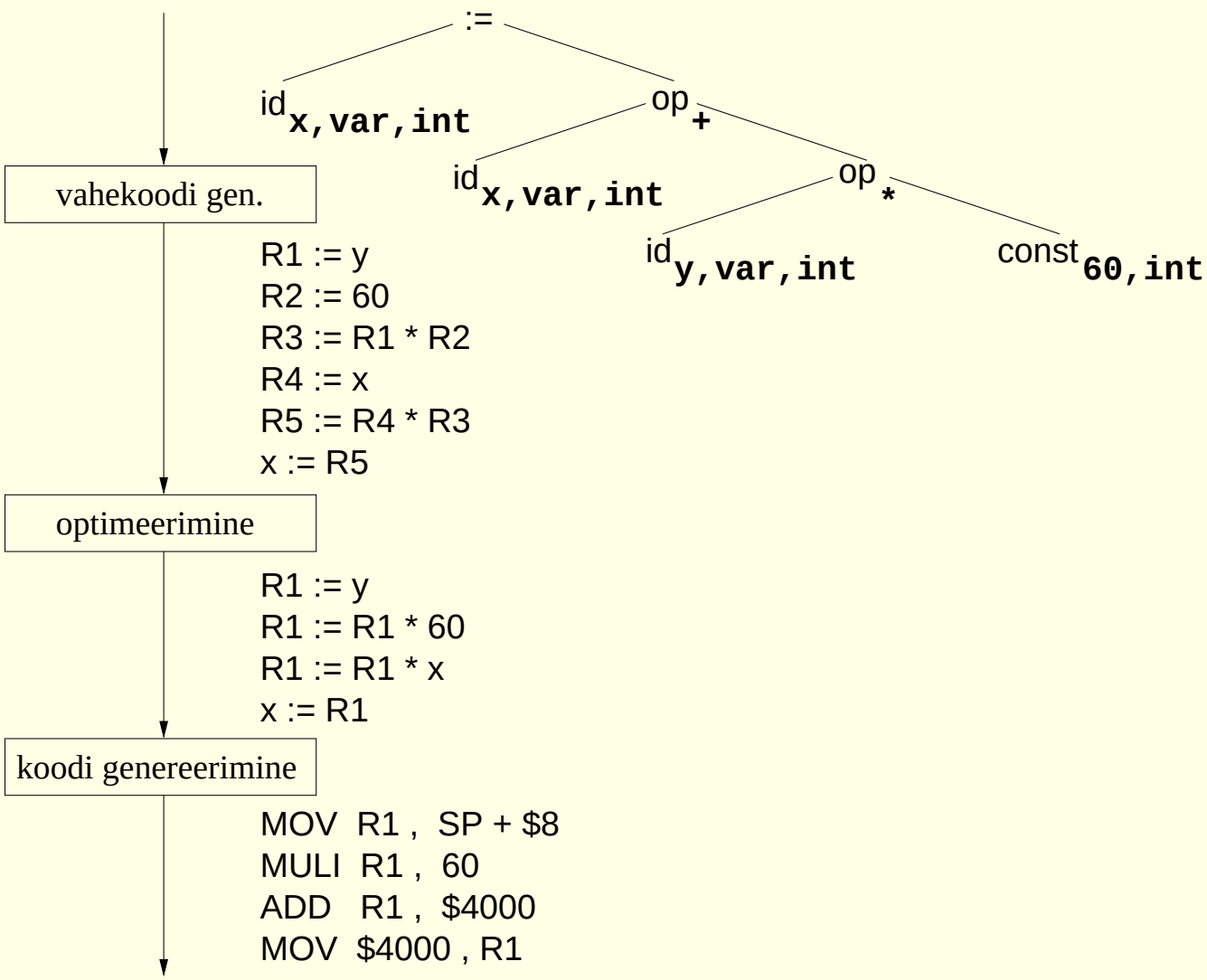
- Sünteesifaasis toimub koodi genereerimine ja optimeerimine.
- Eesmärgiks on riistvara ressursside võimalikult efektiivne ärakasutamine.
- Koodi genereerimine võib toimuda otse peale analüüsi, aga tavaliselt kasutatakse arhitektuurist vähem sõltuvate tegevuste teostamiseks vahekoodi.
- Koodi genereermisel toimub muuhulgas keelekonstruktsioonide asendamine semantiliselt ekvivalentsete instruktsioonide jadadega ning registrite määramine.



Programmi vaheesitused

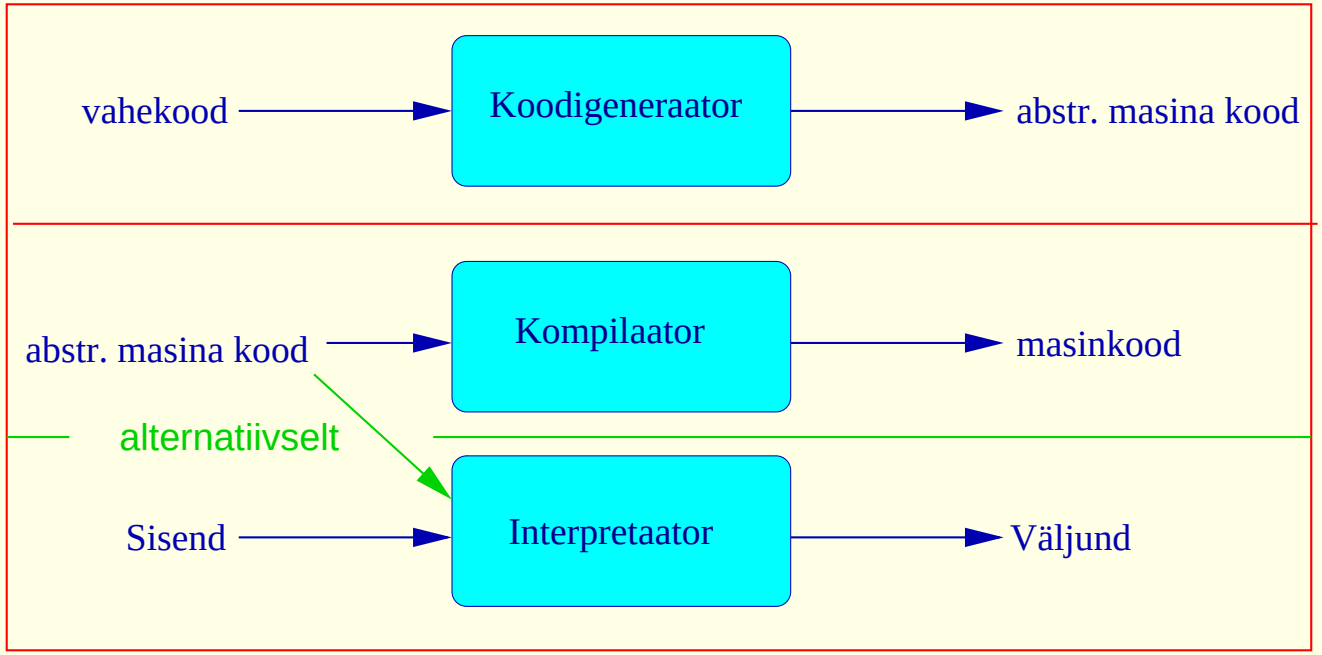


Programmi vaheesitused



Abstraktsed masinad

- Koodi genereerimine koosneb sageli kahest eraldiseisvast osast:
  - programmi transleerimine abstraktse masina koodiks;
  - sellest konkreetse koodi genereerimine või ka otse interpreteerimine.

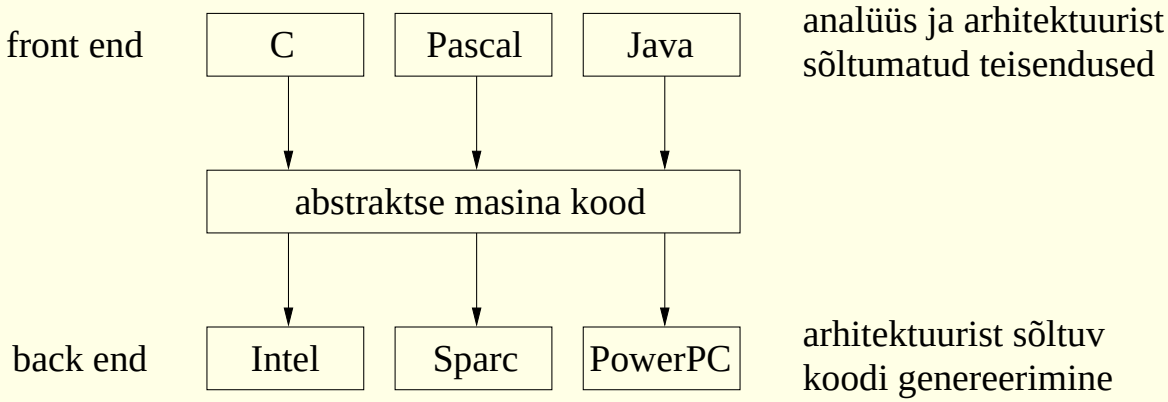


Abstraktsed masinad

- *Abstraktne masin* on idealiseeritud arhitektuuriga masin, mille koodi on lihtne genereerida ja mida on ka lihtne realiseerida erineva arhitektuuriga tegelikel masinatel.
- Keelekonstruktsioonide transleerimine on eraldatud konkreetse riistvara spetsiifilistest omadustest.
- Lihtsustab kompilaatori teisaldamist uue arhitektuuriga masinatele.
- Samuti lihtsustab kompilaatori realiseerimist uute keelte jaoks.

Abstraktsed masinad

- Teoreetiliselt, kui on *m* keelt ja *n* masinat, siis $n \times m$ kompilaatori jaoks on vaja *m front end'i* ja *n back end'i*.



- Praktikas töötab see ainult siis, kui keeled ja masinate arhitektuurid on piisavalt lähedased.

Abstraktsed masinad

- Näiteid abstraktsetest masinatest:

Pascal	$\rightarrow$	P-machine
Java	$\rightarrow$	JVM (”Java Virtual Machine”)
Haskell	$\rightarrow$	STGM (”Spineless-Tagless G-Machine”)
Prolog	$\rightarrow$	WAM (”Warren Abstract Machine”)