Bevezetés

Dr. Horpácsi Dániel
ELTE Informatikai Kar
2023-2024-2

Oktatók

Dr. Horpácsi Dániel
daniel-h@elte.hu
Dr. Kaposi Ambrus
akaposi@inf.elte.hu
Bereczky Péter
berpeti@inf.elte.hu

Számonkérés

- Gyakorlat
- Előadás

Miről fogunk tanulni?

Kapcsolódás más tárgyakhoz (ORSI, NyT, FP, FormMód)

Ezen előadás diasor első változata Dr. Kozma László dékán úr tematikájából és jegyzeteiből kiindulva készült a 2012-2013-as tanévben. Ezúton is köszönöm segítségét és támogatását.

Programozási nyelvek formális definíciói:

Motiváció, alapfogalmak, módszerek

Nyelvek

- A gondolatainkat szavakban, mondatokban fejezzük ki
- A helyesen megalkotott mondatoknak (egyértelmű) jelentése van
- Akik beszélik ugyanazt a nyelvet, kommunikálhatnak egymással

- A lingvisztika a következő kérdésekre próbál választ adni:
  - Hogyan értjük meg a nyelvet, a mondatokat?
  - Pontosan mitől lesz egy mondat “helyes”, érthető?
  - Hogyan lehet precízen meghatározni egy mondat jelentését?
  - Hogyan lehet reprezentálni a jelentésfogalmat?
  - Hogyan viszonyítjuk egymáshoz két mondat jelentését?
  - Lehetséges a megértés folyamatát automatizálni?

Programozási nyelvek

- A program egy speciális kommunikáció ember és gép között, amelynek célja a számítógép vezérlése
- A programokat nem természetes, hanem mesterséges nyelven írjuk:
  - A természetes nyelvek rendkívül komplexek, mondataik sokszor többféleképp értelmezhetőek, így nehéz őket formalizálni, elemezni, illetve automatizálni a megértésüket
  - A géppel történő kommunikáció speciális problémák és algoritmusok közvetítése, amelyhez egyszerűbb, specifikusabb nyelvek is elegendőek
  - A közlés általában imperatív: a program tulajdonképpen utasítások sorozata, amelyeket egyenként végre kell hajtani

Programozási nyelvek definíciója

Nyelvek definícióit általában három fő komponenssel adjuk meg:

Szintaxis

Mely mondatokat tekintjük jól formáltnak?

Azaz, mely szimbólumsorozatoknak van értelme, jelentése?

Szemantika

Mit jelentenek a helyesen formált mondatok?

Azaz, mi a hatásuk, hogyan lehet őket kiértékelni, végrehajtani?

Pragmatika

Hogyan lehet olvasható, konvencionális, hatékony kódot írni?

(Ebben a kurzusban ezzel az aspektussal keveset foglalkozunk.)

A szintaxist, a helyes mondatokat környezetfüggetlen és attribútum grammatikákkal viszonylag könnyen le tudjuk írni, de a dinamikus szemantika formális megadása már nehezebb feladat.

Programozási nyelvek definíciója

Szintaxis? szemantika? pragmatika?

- P4 P4 kérdéses esetek
- Elm
- C++ pragmatika
- K framework

Példa.

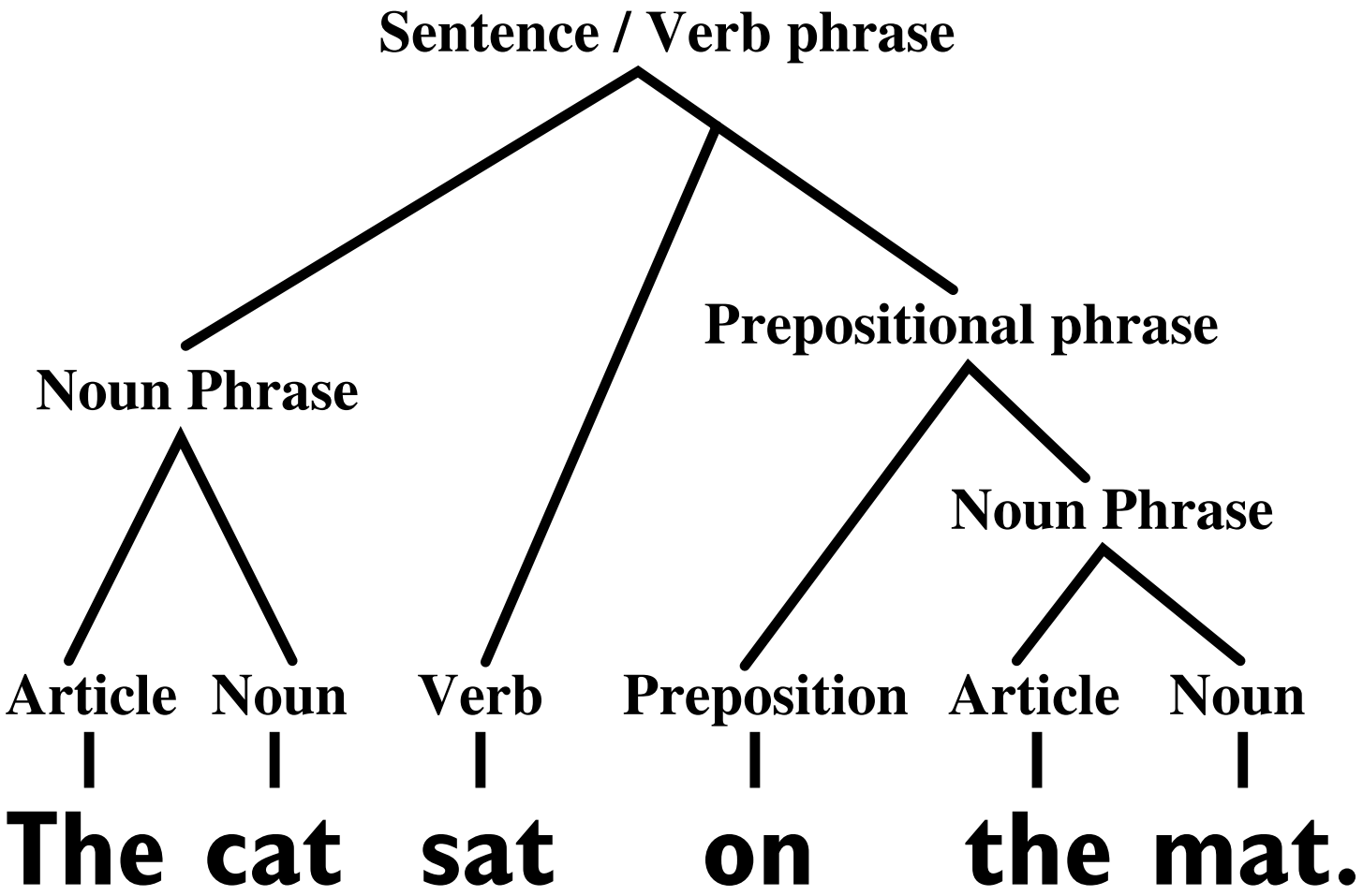
Természetes nyelvekben...

- a szavak betűk sorozatai
- a mondatok pedig szavakból és írásjelekből állnak össze

A nyelv szintaxisa leírja a mondatok megengedett szerkezeteit:

- Szavak lehetnek névelők, főnevek, igék stb.
- Ezekből különböző névszói és igei csoportokat lehet képezni
- A kifejezések megfelelő kombinációja pedig mondatot alkot

Basic constituent structure analysis of a sentence:



Szintaxis kontra szemantika

Ha lehagyunk egy névelőt (vagy egy oda nem illőt használunk), akkor a mondat jó eséllyel helytelen, értelmetlen lesz (szintaxis).

> “There is a a good shop on next corner.”

De az is lehet, hogy egy-két írásjel lehagyása szintaktikusan helyes mondatot eredményez, csak más jelentéssel (szintaxis, szemantika).

> “My sister [,] who lives in London [,] visited me last week.”

> “A királynét megölni nem kell félnetek jó lesz ha mindenki egyetért én nem ellenzem.”

A természetes nyelvi mondatok jelentése sokszor nem egyértelmű:

> “Time flies.”
> “Students hate annoying professors.”

Szintaxis informálisan

Definiáljunk egy egyszerű mesterséges nyelvet! Lexikális szinten vannak benne változók (kisbetűk), literálok (számok) és operátorok (*, =, ;). A lehetséges mondatokat a következőképp specifikáljuk:

- A mondatok utasítások, amelyeket pontosvessző (;) zár
- Az utasítások egy változónévből, egy egyenlőségjelből (=), és egy kifejezésből állnak
- A változónevek és a literálok helyes kifejezések
- Összetett kifejezést szorzással (*) alkothatunk

Szintaxis formálisan

A (programozási) nyelvek szintaxisát általában formálisan definiálják (többnyire környezetfüggetlen grammatikákkal, BNF formában).

```
<statement>  ::= <variable> '=' <expression> ';'
<expression> ::= <variable> | <literal>
               | <expression> '*' <expression>
```

A szintaktikusan helyes mondatok nyelvtanilag helyesek, de ez nem garantálja, hogy "olvashatóak", és hogy van értelmük, jelentésük.

Szintaxis formálisan

```
<statement>  ::= <variable> '=' <expression> ';'
<expression> ::= <variable> | <literal>
               | <expression> '*' <expression>
```

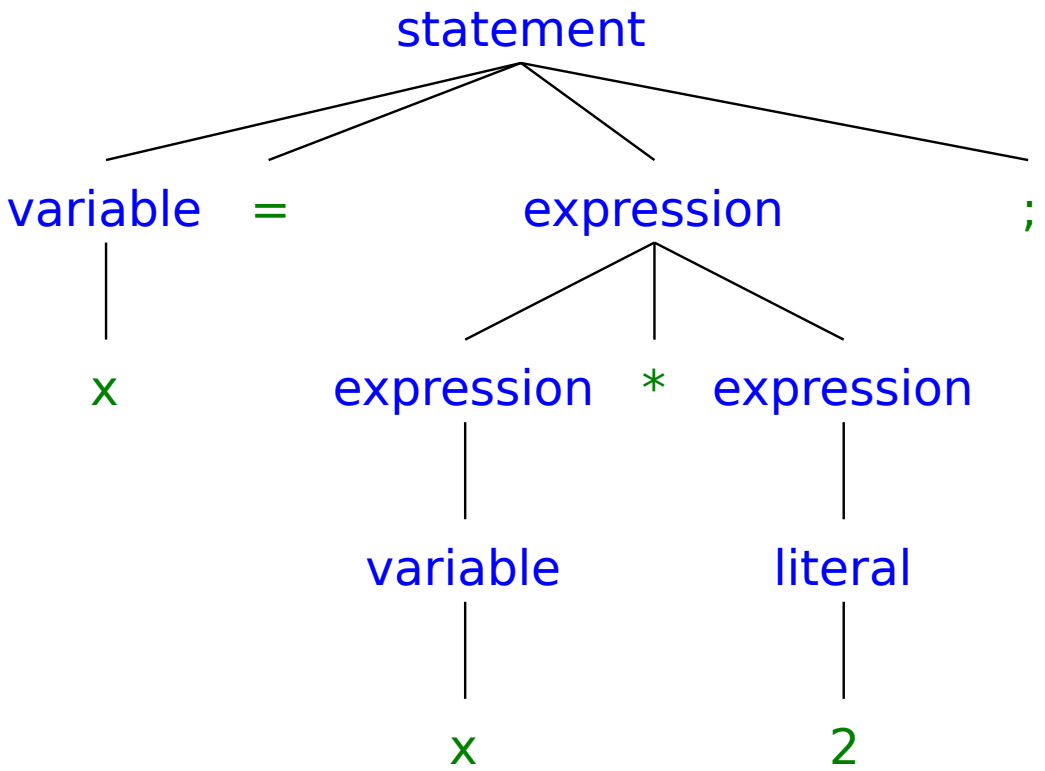
Konkrét helyett absztrakt szintaxis leírás:

$$
\begin{aligned}
\mathit{assign} &: \mathit{name} \to \mathit{expr} \to \mathit{statement} \\
\mathit{lit} &: \mathit{int} \to \mathit{expr} \\
\mathit{var} &: \mathit{name} \to \mathit{expr} \\
\mathit{mul} &: \mathit{expr} \to \mathit{expr} \to \mathit{expr}
\end{aligned}
$$

Például: $\mathit{assign}(\text{"}x\text{"}, \mathit{mul}(\mathit{var}(\text{"}x\text{"}), \mathit{lit}(2)))$

```
<statement>  ::= <variable> '=' <expression> ';'
<expression> ::= <variable> | <literal>
               | <expression> '*' <expression>
```

Az “x = x * 2;” mondat helyes a fenti grammatika szerint.



Szintaxis formálisan

```
<statement>  ::= <variable> = <expression> ;
<expression> ::= <variable> | <literal>
               | <expression> * <expression>
```

Szerkezetileg helyes mondatoknak azonban nem feltétlenül van értelme, illetve többértelműek is lehetnek.

Az “`x = y * 2;`” egy szerkezetileg helyes mondat. Vajon mit jelent, ha y még nem kapott értéket? Az x változó nullázódik vagy hibát kellene jelezni? Elérkeztünk a szintaxis és a szemantika határára.

Érdekesség: különböző stílusú nyelvekben nagyon eltérően nézhetnek ki nagyon hasonló jelentésű mondatok.

```
while(x == 1)
{
  f();
}
```

Vagy pedig:

```
while x? = 1 do
  call f
done
```

Különbözőképp néznek ki, de vajon ugyanazt jelentik?

```
if(x == 1) f();
else       g();
```

vö.

```
if(x == 1) {
    f();
} else {
    g();
}
```

vö.

```
(x == 1) ? f() : g();
```

Továbbá: konkrét és absztrakt szintaxis!

Informális szemantika

A programozási nyelvek megszületésével szinte csak informális leírások voltak a nyelvekről. A szemantika minden esetben angolul, természetes nyelven volt megadva.

> *Execution of the statement*
>
> `if <cond> then <stmts> fi`
>
> *checks the condition at first, and if it evaluates to* `true`*, then the statements are executed one after the other.*

- A természetes nyelvi leírások nem precízek, nem tekinthetők formális definíciónak (ellenben a matematikával és matematikai logikával), következésképp félreértésekhez vezettek.
- Másrészt tény, hogy formális szemantikadefiníciót készíteni egy nyelvhez hosszú és bonyolult feladat, továbbá a megértése is nehezebb egy olvasmányos leírásnál.

Informális kontra formális

Az informális leírásokat a fordítóprogramok és a programozók is többféleképp értelmezték...

- Ugyanahhoz a programozási nyelvhez készült fordítóprogramok különböző tárgykódokat eredményeztek egyazon program esetében, így a lefordított programok másképp működtek
- A programozók nem tudtak igazán jó és hatékony kódokat írni, mert ugyan nagyjából értették, mi mit csinál, de nem tudták, pontosan mi folyik a háttérben, milyen optimalizációkat hajt végre a rendszer

De ami nekünk, ELTE-seknek még fontosabb: ahhoz, hogy érvelhessünk programekvivalenciáról vagy programhelyességről, elengedhetetlen a formális szemantikadefiníció.

A fordítóprogram/interpreter definiálja a szemantikát?

Mi az *f* függvény visszatérési értéke a következő C nyelvű programban?

```
int f(void) {
    int x = 0;
    return (x = 2) + (x = 1);
}
```

A fordítóprogram/interpreter definiálja a szemantikát?

Mi az *f* függvény visszatérési értéke a következő C nyelvű programban?

```
int f(void) {
    int x = 0;
    return (x = 2) + (x = 1);
}
```

- GCC (4.8.4): **2**
- CLANG (3.4): **3**

A fordítóprogram/interpreter definiálja a szemantikát?

Mi az *f* függvény visszatérési értéke a következő C nyelvű programban?

```
int f(void) {
    int x = 0;
    return (x = 2) + (x = 1);
}
```

- GCC (4.8.4): **2**
- CLANG (3.4): **3**

...a jelentés sokszor szándékosan nem definiált vagy többértelmű.

Informális kontra formális

Mi értelme tehát a szemantikát formálisan tárgyalni?

- Alapvető dokumentációt ad a programozási nyelvnek
- Feltárja a félreértelmezhető elemeket a nyelvben, hogy végül minden fejlesztő, fordító és értelmező pontosan ugyanúgy értse a programot
- Precíz jelentésfogalmat alkot, amelynek fontos következményei:
  - Vizsgálhatjuk programok tulajdonságait
  - Vizsgálhatjuk programok ekvivalenciáját
  - Vizsgálhatjuk programok helyességét

Formális szemantika — igények

A nyelv különböző használóinak különböző formális definíciók, megfogalmazások lehetnek hasznosak. Kinek mi érdekes a nyelv szemantikájából, min van a hangsúly?

- Fordítóprogramtervezők:
  “Hogyan kell végrehajtani a programot?”
- Nyelvtervezők és helyességbizonyítók:
  “Mi a program lefutásának hatása?”
- Programozók:
  “Hogyan írjak programot, mik a főbb jellemzők?”

Formális szemantika — szintek

- Statikus szemantika
  - A szintaxis azon része, amelyet nem lehet környezetfüggetlen grammatikával megadni
  - Gyakran hívjuk környezetfüggő szintaxisnak
  - Ez is arra ad megkötéseket, hogy mi számít helyes mondatnak

  - Tipikusan: névkötések és típushelyesség
  - Formális eszközök: ATG, típuslevezetés

- Dinamikus szemantika
  - Hogyan kell végrehajtani a programot
  - Mi lesz a végrehajtás eredménye

  - Például: az “x + y” kifejezés esetén kiértékeljük az “x”, majd az “y” kifejezést, végül összeadjuk a részeredményeket

Formális szemantika — fajták, megközelítések

Formális szemantika: a programok jelentésének szigorú, precíz, egyértelmű, matematikai definíciója.

Sokféle megközelítés létezik. Az alapvetőek:

- (Attribúrum grammatikák)
- Operációs (műveleti) szemantika
- Denotációs (leíró) szemantika
- Axiomatikus szemantika

És ezeknek variációi...

- Reduction semantics
- Action semantics
- Categorical semantics
- Game semantics

$$\langle assignment \rangle.code = "mov \ldots"$$

Általában átírási (translation) szemantikát definiálunk vele

- Sokszor a statikus szemantika leírására használatos
- Fordítóprogram generátorok szokásos bemenete
- Egyetlen grammatikából előállítható a szintaktikus és szemantikus elemző, továbbá a kódgenerátor (egy teljes fordítóprogram)

Kényelmesen megadható vele a környezetfüggetlen és környezetfüggő szintaxis is. Használják a következőkre:

- Statikus szemantikus elemzés
- Nyelvspecifikus programozási környezetek
- Teszteset-előállítás

(A denotációs szemantika speciális eseteként is kezelhető.)

Operációs (műveleti) szemantika

$$\langle S, s \rangle \Rightarrow \langle S', s' \rangle$$

Fordítóprogramok és értelmezők alapja lehet, de bizonyos esetekben programtulajdonságok belátására is szokás alkalmazni

- Definiálja, hogyan hajtódik végre a program lépésről lépésre
- Induktívan definiált átmenetrendszerrel adja meg a jelentést
- Lehet jobban vagy kevésbé részletes (strukturális, természetes) (“Small-step” vagy “big-step”)
- Címkézett rendszerek esetén trace szemantika

$[\![S]\!]s$

Nyelvtervezés, helyességbizonyítás, továbbá végrehajtjató szemantika készítése során használatos

- A programokat és azok részeit matematikai objektumokra képezi
- Az operációs leírásnál absztraktabb megfogalmazás
- Általában kompozicionálisan adjuk meg a nyelvi elemek jelentését
- Folytatásos (continuation) stílusban is használjuk

$$\{P\}S\{Q\}$$

Főleg helyességbizonyításhoz

- Szemantika = programok tulajdonságainak levezetése (előfeltétel + utófeltétel, Hoare-hármasok)
- Az állapotfogalomtól független megfogalmazási forma
- Végrehajtást nem definiál, csak tulajdonságokat vezet le
- A denotációs és operációs szemantikánál is absztraktabb

Összegzés

- Informális kontra formális
- Szintaxis kontra szemantika
- Statikus és dinamikus szemantika
- Operációs, denotációs, axiomatikus

Ajánlott irodalom

- Hanne Riis Nielson and Flemming Nielson. 1992. *Semantics with Applications: A Formal Introduction*. John Wiley & Sons, Inc.
- Kenneth Slonneger and Barry L. Kurtz. 1994. *Formal Syntax and Semantics of Programming Languages*. Addison Wesley Longman.
- Glynn Winskel. 1994. *The Formal Semantics of Programming Languages – An Introduction*, Foundations of Computing Series, MIT Press, Cambridge, MA.
- John C. Reynolds. 1998. *Theories of Programming Languages*. Cambridge University Press.