CLASE 07/10/2020

I.Sintaxi i semantica estatica

3/41

Sintaxi: quina sequencia de caracters constitueixen un programa “legal”. Elements sintactics del llenguatge.

Semantica: que significa (que calcula) un programa legal donat. Importancia:

1.Ajuda al programador a “raonar” sobre el programa.

2. Es necessaria per a implementar correctament el llenguatge (models d’execuccio)

3.Premet desenvolipar tecniques i eines de: Analisi i optimitzacio, depuracio, verificacio, transformacio.

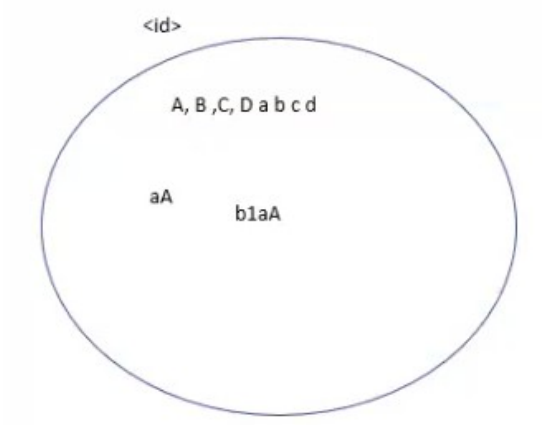
4/41

Notació BNF

Regles de produccio. La barra vertical | es una “ó”

Tenim un grup “letter” lletres ABCDabcd
Els identificadors valids son… una lletrá LETTER ó un identificador+lletra ó identificador+ digit

Se defineixen de manera recursiva



El que esta entre claudators [i] pot estar o no, es opcional.

5/41

Exemples us de gramatiques BNF

6/41

Com un compilador processa un programa font

Le symbol table//Othertables se guarda informacio com el tipus de les variables. En el SyntacticAnalisis pot guardar en la taula que la x es int. En el semanticAnalysis consulta la taula si puc guardar un “5” en una x. Si que pot perque es enter.

ANEM A VEURE CADA PART del processament en les proximes diapos.

7/41

Anàlisi lexic i Anàlisi sintactic

LEXIC: Divideis els caracters del programa en paraules o sintactics primitius (tokens)

SINTACTIC: Reconeix els tokens anteriors i obte una sequencia d’instruccions en forma d’arbre sintactic (diapos 9)

8/41

Exemples 1 i 2 son el analisis lexic. El exemple 1 te caracters a principi sense significat. El exemple 2 agafa els caracters del principi i construeix tokes d’acord a la sintaxi del llenguatge de programacio en el que estem treballant.
Exemple 3 es el sintactic. Comprova la instruccio que han aparegut en els tokens i veiem que es una funcio.

9/41
Arbre sintactic. Es lo que en la diapos 7 possa a la dreta en rosa de “parse tree”.
Representa el codi de la diapos anterior, el exemple 3. Se van desplegant les fulles a mesura que s’avança en el codi.

10/41
Analisi semantic
Semantica estatica: Tenim restriccions que no se poden expressar en el BNF anterior que hem vist, pero si que poden comprovarse **en temps de compilacio**. *Per exemple no declarar una variable i intentar utilitzarla… te diria en compilacio que algo falla.*

Semantica dinamica: Restriccions que nomes es poden comprovar **durant l’execucio del programa**. Pexemple: Detectar una divisio per zero. Programant no ens hem adonat pero en la execucio del programa, se’ns diu. (*La dinamica la veurem en la diapos 14)*

11/41
Fase 2: Comprovacions en el analitzador semantic:
1. Variables declarades abans d’utilitzar-les.
2. Compatibilitat i conversio de tipus (cohersio)
3. Comprovar que els parametres introduits son els que corresponen amb els descrits en el codi.
4. Etc

Després de les comprovacions se produeix un codi intermedi per a la tercera fase.

12/41
Fase 3: Compilació i enllaç: Generacio de codi executable.
-Optimitzar el codi.
-Generacio del codi del programa.
-S’enllaça el codi del programa amb altres programes o llibreries i se genera el codi executable.

II.Semantica dinamica

14/41

Semantica dinàmica:

El compilador no pot detectar tots els errors possibles. Per exemple:

1. Errors que se manifesten durant la execucio. *Com dividir per 0, o ixir-se'n d'un array.*

2,Moltes propietats interessants d'un programa **no son decidibles**. (Decidible significa que existeis un algorisme que pas a pas i en un temps finit ens diu si algo se cumpleix o no). Per exemple:

-La terminacio: No podem saber si un programa ha acabat o no per a qualsevol possible execució. *Pot caure en un bucle infinit per exemple i no acabar*

-Si dos programes son equivalents semanticament, es a dir… Si fan la mateixa funció encara que estiguen escrits de manera diferent.

-Si dos descripcions BNF genenren el mateix llenguate.

-Etc

***Semidecidible es que si acaba genial, pero si no acaba…. Deixarlo mes temps i no podem dir res.*

Aquestos problemes citats en la diapos, els podrem solucionar (no tots) amb la semantica dinamica. La següent dispos dira de que esta composada la semantica dinamica.

15/41

Estils de definicio semantica

Mirarem la operacional i la Axiomatica en classe.

Operacional: **Punt 3 del tema, diapos 16.** Es la mes intuitiva, ve a ser com definir un interpret per al llenguatge

Axiomatica **Punt 4 del tema, diapos 30.** La axiomatica es diferent a la operacional, se sol utilitzar en temes de verificacions formals

Declarativa: Denotacional, algebraica, teoria de models, punt fix *(Esta no la veurem)*

III.Semantica operacional

16/41

Semantica operacional:

Consisteix a definir una maquina (abstracta) M i expressa el significat de cada construccio del llenguatge en termes de les accions a realitzar per la maquina per a executar aquesta instruccio. *La idea es donar un interpret per al llenguatge de programacio i dir per a cada instruccio del llenguatge, que fer en l'execució.*

17/ 41

Classe 7/10/2020 Mirar ultims 15-20 minuts de la classe per entendreho.

Necessitem un estat "s" que prendra variables del programa i ens dira quin valor tenen. S: X→ D

$$s : \mathcal{X} \to D$$

*Els estats venen definits en **parells variable-valor**. En pla… per a una variable X1 tenim un valor V1, Per a una variable X2 tenim un valor X2…*

$$s = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}.$$

Les configuracions de la maquina son un parell. <i,s> on la i es una instruccio (el codi que volem executar, siga una unica ordre o tot el codi. Se pot veure de diverses maneres) y la s un estat (estat actual de la maquina o l'execucio que estiguem considerant)

18/41

Per a formalitzar la execució del programa utilitzem una relació de transició i el que diu es…. Donada una configuracio (que representa el estat actual de la maquina) com avançar en l'execució del programa

Donat un estat inicial, si se compleix la premisa, avancem al estat següent. →

$$\frac{premissa}{\langle i, s\rangle \to \langle i', s'\rangle}$$

Tambe utilitzem altres relacions per a descriure:
-La **avaluacio de expressions** aritmetiques. ----> $(\langle exp, s\rangle \Rightarrow n).$

-La **obtenció directa** d'un estat final----------> $(\langle i, s\rangle \Downarrow s')$

Se definixen amb regles similars a la fraccio groga, blava, verda.

CLASE 08/10/2020
19/41

Llenguatge SIMP.

Sintaxi

$$a ::= C \mid V \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2$$

← 1, Defineix **expresions aritmetiques**. Amb constants numeriques (C) i variables (V), la suma/resta/multiplicacio aritmètica

$$b ::= \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \vee b_2$$

← 2, Defineix **expresions booleanes** true o false amb igualtat, desigualtat, negacio o unió.

3. Intruccions:

-Com skip (que no fa res i s'utilitza com a acabador de codi)
-V:=a Un asignador
-i_1 i_2 La seqüencioa de dos instruccions
-if b then i1 else i2
-while b do i.

$$i ::= skip \mid V := a_1 . \mid i; i_1 \mid \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2 \mid \texttt{while } b \texttt{ do } i$$

Estos tres punts d'enrere estan incomplets, hi han moltes mes, pero expliquen eixes en clase per simplificar. Faltaria divisio, disjunció etc

En les proximes diapos s'explica com crear les expressions aritmetiques (21/41) i les booleanes (22/41)

20/41

Llenguatge SIMP: Avaluació d'espressions

(La clase del 14/10/2020 te un video explicant un cas practic des de la diapos 20 a la diapos 25)

Donada una expressio (exp) i un estat (s), torna un valor "de debò" n. Tin en compter que se gasta la fletxa doble per a aço i aixo va tornarnos un **valor numèric** o un **valor booleà**

$$\langle exp, s \rangle \Rightarrow n$$ a

21/41

Llenguatge SIMP: Avaluació d'espressions aritmètiques

Constants numeriques:

Una variable i qualsevol estat, torna el valor de la variable, dona igual que estat que tinguem. Aci no tenim una premisa. *Si volem mirar la constant 8, tornem la constant 8, no se mira estat ni comprovar res.*

$$\langle \mathtt{n}, s \rangle \Rightarrow n$$

Variables:

Se fa de manera intuitiva… Mirar quant val eixa variable x en $\langle x, s \rangle \Rightarrow s(x)$ el estat actual s i eixe es el valor que torna, S(x)

Adición (suma) (la resta i el producte es similar)

Se mira recursivament

Per a evaluar una suma a1 + a2 de premisa tenim que cal evaluar el primer operando a1 per a un cert valor n1 i el segon operand a2 per aun valor n2 i tornar la suma de n1 i n2. *Observar que en la premisa tenim dos casos simples de constants numeriques (lo primer que hem vist en esta diapos)*

$$\frac{\langle a_1, s \rangle \Rightarrow n_1 \qquad \langle a_2, s \rangle \Rightarrow n_2}{\langle a_1 + a_2, s \rangle \Rightarrow n}$$

22/41

Llenguatge SIMP: Avaluació d'espressions aritmètiques

Constants booleans:

El valor que te, es el que torna. Si es false, torna false, si es true, torna true. Aixo seria el cas base, lo mes simple. Igual que feiem amb les constants numeriques-→

$$\langle \mathtt{false}, s \rangle \Rightarrow false \mid \langle \mathtt{true}, s \rangle \Rightarrow true$$

Igualtat:

Igual que feiem en la suma de la diapos anterior, evaluar el valor de cada operand i una vegada els tenim, veure si son iguals. Si son iguals torna true, si son diferents torna false . El de dalt son iguals i torna true i el de baix son diferents i torna false.

$$\frac{\langle a_1, s \rangle \Rightarrow n_1 \qquad \langle a_2, s \rangle \Rightarrow n_2}{\langle a_1 = a_2, s \rangle \Rightarrow true}$$

$$\frac{\langle a_1, s \rangle \Rightarrow n_1 \qquad \langle a_2, s \rangle \Rightarrow n_2}{\langle a_1 = a_2, s \rangle \Rightarrow false}$$

Menor o igual (Aplicable a mayor o igual)

De nou s'avaluen els valors de cada operand per separat i despres se fa la comparacio de la desigualtat. Si se compleix torna true i si no se compleix torna false. En la imatge de la dreta la de dalt se compleix i la de baix no.

$$\frac{\langle a_1, s \rangle \Rightarrow n_1 \qquad \langle a_2, s \rangle \Rightarrow n_2}{\langle a_1 \leq a_2, s \rangle \Rightarrow true}$$

$$\frac{\langle a_1, s \rangle \Rightarrow n_1 \qquad \langle a_2, s \rangle \Rightarrow n_2}{\langle a_1 \leq a_2, s \rangle \Rightarrow false}$$

Negacio:

Canvia el valor de la regla definida. Si la regla s'avalua a true, torna false. Si la regla s'avalua a false, torna true.

$$\frac{\langle b, s \rangle \Rightarrow true}{\langle \neg b, s \rangle \Rightarrow false} \mid \frac{\langle b, s \rangle \Rightarrow false}{\langle \neg b, s \rangle \Rightarrow true}$$

Disjuncio:

Evaluar el valor dels operands per separat n1 i n2. Si algun dels dos o els dos es true, aleshores torna true.

Si els dos foren falsos, torna fals.

$$\frac{\langle a_1,s\rangle \Rightarrow n_1 \qquad \langle a_2,s\rangle \Rightarrow n_2}{\langle a_1 \vee a_2,s\rangle \Rightarrow true}$$

$$\frac{\langle a_1,s\rangle \Rightarrow false \qquad \langle a_2,s\rangle \Rightarrow false}{\langle a_1 \vee a_2,s\rangle \Rightarrow false}$$

Les regles vistes abans, servixen per a les dos clases de semantica. La small step i la big step, pero la small i la big son independents entre elles. O escollim una o escollim l'altra, pero no les dos.

III.1Semantica operacional: Small Step

SMALL STEP

23/41

La idea es… donat una execucio d'un programa anar pas a pas fins arribar al resultat final.

Partim d'un programa P i un estat inicial s_i on no hi ha cap variable asignada. L'bjectiu es **contruir una traça** que denote la configuració d'eixe programa. La primera configuracio de la traça es igual al programa P inicial i al estat inicial (s_i) i la idea es **avançar aplicant regles** fins que no pugam aplicar-ne mes… Que son dos situacions:

$$\langle P, s_I \rangle = \langle P_1, s_1 \rangle \rightarrow \langle P_2, s_2 \rangle \rightarrow \cdots \rightarrow \langle P_n, s_n \rangle$$

Situació 1 (ACABA.): Que P_n siga un skip… **Aixo es bo** perque significa que el programa no te res mes per executar i ha acabat correctament. I ademes tenim un S_n que es el estat final y conte el resultat d'executar aqueix programa.

Situació 2 (NO ACABA): Que no puguem arribar a un punt on P_n siga un skip, **això no es bo** perque significa que **sempre podem aplicar una regla** i aixo implica que hem arribat a una situacio de bucle infinit.

24/41

Semàntica del Small Step

Seqüencia

Tenim dos seqüencies, la de l'esquerra ia la de la dreta.

ESQUERRA:

Si te un skip; instruccions, estat, pues com lo inicial esta acabat perque te el skip, pues ens torna les instruccions restants amb el estat que tingen.

$$\frac{}{\langle skip;i,s\rangle \rightarrow \langle i,s\rangle}$$

DRETA:

Si la primera instruccio no es skip (groc), s'evalua recursivament i diu que comences evaluant i_1 (verd) i una vegada ho hajes evaluat (taronja) actualitza lo inicial (blau).

$$\frac{\langle i_1,s\rangle \rightarrow \langle i_1',s'\rangle}{\langle i_1;i_2,s\rangle \rightarrow \langle i_1';i_2,s'\rangle}$$

La seqüencia requereix sempre un punt i coma. Si no tinguera punt i coma, probablement siga *asignacio***.**

Al final de la classe de 8/10/2020 s'explica. Els ultims 10-15 minuts.

Asignació:

Evaluar el que val la expressio a la dreta del ":=" (<a,s> en l'exemple) i després actualitzar el valor de la variable amb eixe valor. La variable es el estat s.

$$\frac{\langle a,s\rangle \Rightarrow n}{\langle x := a,s\rangle \rightarrow \langle skip, s[x \mapsto n]\rangle}$$

Torna un skip perque la asignacio ja s'ha fet pero com a estat, el nou valor de la X. Si X no valia res, ara val n i si X valia una altra cosa, ens dona igual el que valguera perque ara val n.

CLASE 14/10/2020

25/41

Condicional:

$$\frac{\langle b,s\rangle \Rightarrow true}{\langle \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2, s\rangle \rightarrow \langle i_1, s\rangle} \qquad \frac{\langle b,s\rangle \Rightarrow false}{\langle \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2, s\rangle \rightarrow \langle i_2, s\rangle}$$

Evaluar la condició del if, la qual pot ixirnos que es vertadera o falsa.
-Si es vertadera, s'avaluen les condicions del **then** i_1.
-Si es falsa, s'avaluen les condicions del **else** i_2 ; sempre per a un estat s siga i1 o i2.

Bucle while:

$$\frac{\langle b,s\rangle \Rightarrow false}{\langle \texttt{while } b \texttt{ do } i, s\rangle \rightarrow \langle skip, s\rangle} \qquad \frac{\langle b,s\rangle \Rightarrow true}{\langle \texttt{while } b \texttt{ do } i, s\rangle \rightarrow \langle i; \texttt{while } b \texttt{ do } i, s\rangle}$$

Evaluar la condició del while , la qual pot ixirnos que es vertadera o falsa.
-Si es falsa es que el bucle ha acabat i torna skip i un estat.
-Si es vertader, torna una nova configuracio en la que se te primer el cos del bucle while i despres del punt i coma una copia de la configuració original. *Amb aixo conseguim que se faça una iteracio i que despres el estat comprove si seguim executant o ja eixim.*

VIDEO EXEMPLE DE COM SE FA EL SMALL STEP PEL PROFE DE FLIP:
https://media.upv.es/player/?id=bd868100-9f9d-11e7-adbd-33a63bcab066

Esta clase conte una explicacio de les regles de transicio, miraho perque es super interessant de nou. I explica tot lo vist des de la diapos 20/41. El llenguatge SIMP

III.2Semantica operacional: Big Step

26/41

BIG STEP

En el big step se mira… Donat un programa P amb un estat inisial S_i, fer una transicio directa a un estat final S_f .

$$\langle P, s\rangle \Downarrow s'$$

El que fa es relacionar una configuració amb un estat

27/41

Semàntica del Big Step

Instruccio buida:

Si tenim un skip es que no podem fer res mes i torna l'estat final.

$$\frac{}{\langle skip, s\rangle \Downarrow s}$$

Seqüencia:

Evaluar completament la primera intruccio de la seqüencia que torna un estat s_1 i per altra banda evaluar la segona instruccio de la seqüencia que torna un estat s'. L'estat que torna (S') es l'estat final associat a tota la seqüencia

$$\frac{\langle i_1, s\rangle \Downarrow s_1 \qquad \langle i_2, s_1\rangle \Downarrow s'}{\langle i_1; i_2, s\rangle \Downarrow s'}$$

IMPORTANT. La segona instruccio s'avalua amb l'estat que torna la primera instruccio.

Assignació:

Pareguda a la del small step. Donada una expressio, s'avalua l'estat d'aquesta . La diferencia es que aci no se torna una configuracio amb skip com passava en el smallStep, aci nomes retorna l'estat.

$$\frac{\langle a, s\rangle \Rightarrow n}{\langle x := a, s\rangle \Downarrow s[x \mapsto n]}$$

Condicional:

Evaluar la condició a true o false i a diferencia del SmStep que tornava el then o el else, ací se possa el then o el else **en la premisa** segons si es true o false fins arribar a un estat final.

$$\frac{\langle b,s\rangle \Rightarrow true \qquad \langle i_1,s\rangle \Downarrow s'}{\langle \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2, s\rangle \Downarrow s'} \qquad \frac{\langle b,s\rangle \Rightarrow false \qquad \langle i_2,s\rangle \Downarrow s'}{\langle \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2, s\rangle \Downarrow s'}$$

Bucle While:

Similar de nou pero passa com el condicional… Si es false, torna el estat i punto, no fa el bucle. Si es true, en lloc de tornar les instruccions junt al bucle complet, passa com el condicional del big step, **que el coloca en la premisa** avalua el bucle a un nou estat s' i executa el bucle de nou amb el nou estat s' fins que arribe a un false i la s'''' que tinguem sera l'estat final.

$$\frac{\langle b,s\rangle \Rightarrow false}{\langle \texttt{while } b \texttt{ do } i, s\rangle \Downarrow s} \qquad \frac{\langle b,s\rangle \Rightarrow true \qquad \langle i,s\rangle \Downarrow s' \qquad \langle \texttt{while } b \texttt{ do } i, s'\rangle \Downarrow s''}{\langle \texttt{while } b \texttt{ do } i, s\rangle \Downarrow s''}$$

CLASE 16/10/2020

29/41 Mes o menys sobre 1 hora de video de la clase, comença la teoria.

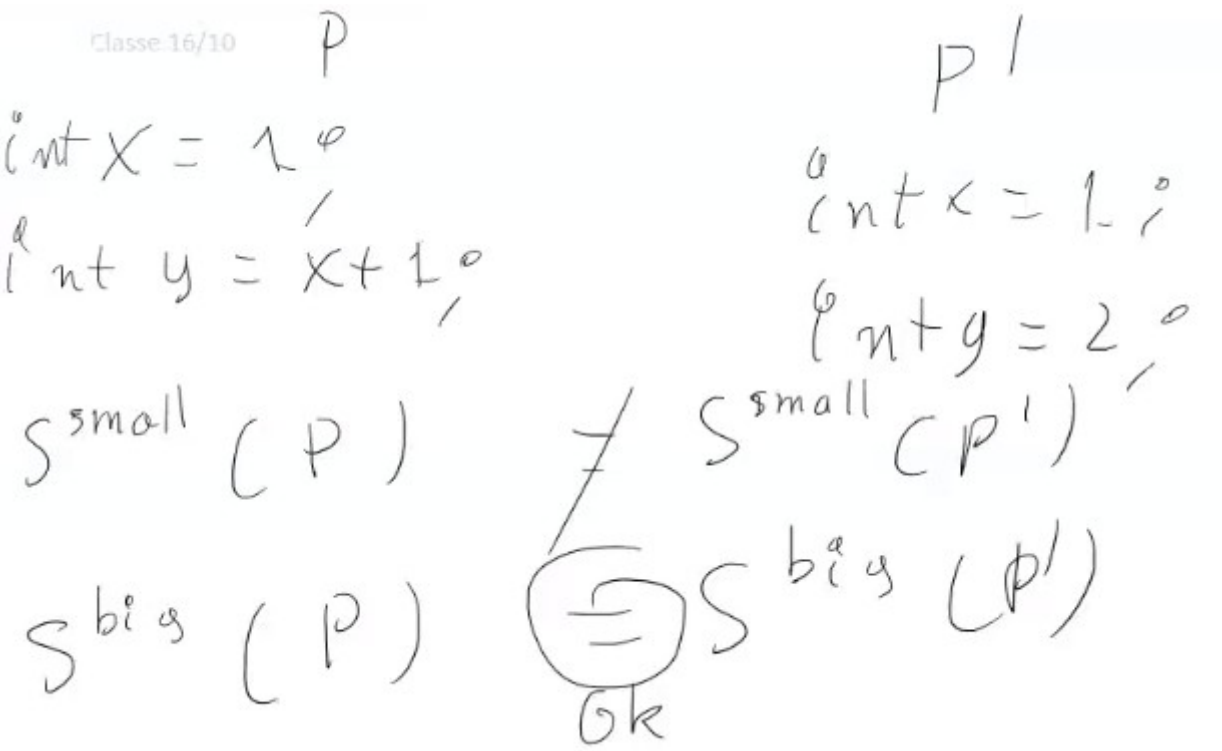
Semantica d'un programa:

S(P) ← Semantica d'un programa

$S^{small}(P)$ Dos programes tenen la mateixa semantica en small si son dos strings identics

$$\langle P, s_I \rangle = \langle P_1, s_1 \rangle \rightarrow \langle P_2, s_2 \rangle \rightarrow \cdots \rightarrow \langle P_n, s_n \rangle = \langle skip, s_F \rangle$$

$S^{big}(P)$ Dos programes son el mateix si donen el mateix estat final, aci no importa la execucio.



IV.Semantica Axiomàtica

30/41

Aquesta es diferent de la operacional que s'utilitzava per a generar interprets per exemple… ***La xiomatica s'utilitza majoritariament per a tasques de verificacio.*** *Per a comprovar la correccio d'un programa. Partim de un* ***Hoare triple*** *que te tres elements, una {P} una S i una {Q} on...*

{P} :Precondicio que han de complir els programes S. Una formula logica.
S :Llenguate programacio, un fragment, una instruccio…
{Q} :Post condicio de nou una formula logica.

Partint d'un estat P que se cumpleix, aplicarlo al programa S, te porta a un estat Q que se compleix tambe.

Per exemple, definir una funcio que torne el maxim d'un array. La precondició podria ser que el array estiga ordenat de major a menor, tornaria el primer element i la postcondicio es que efectivament es el maxim element del array. Si el array estiguera mal ordenat la funció aniria mal i la postcondicio no se compliria.

31/41
21/10/2020

Semàntica Axiomàtica: **Transformador de predicats de Dijkstra**

El transformador de predicats associa a cada tipus s'isntruccio "i" i postcondicio Q una **precondicio mes feble** pmd(i,Q). El pmd pren una instruccio o un conjunt d'intruccions i una postcondicio Q i ens torna la precondició, es a dir… va al revés. Podria tornarnos infinites precondicons pero el que torna aquest algorisme es la **precondició més feble** es a dir la més general.

Per exemple (I): Si X pot ser igual a 1 a 5 a 65… Podria tornar x>0

Amb aqueixa pmd, podrem comprovar la correcció del programa, es a dir la terna {P} S Q} es correcta. Se fa amb **dos pasos**, no olvidar cap dels dos.

1. Calcular P' = pmd(S,Q) ← Una vegada tenim aço, comprovar el 2.
2. Comprovar que P => P' ← Comprovar que P implica P'

Seguin exemple(I) anterior, haviem calculat que P': x>0 i la X mes menuda era 1. Molt be perque x=1 implica que x>0. Aixo es la comprovació.

32/41

Semàntica Axiomàtica: **Operacions Transformador de predicats pmd**

En el full 1 del tema dos a ma tinc com se fa la **assignacio** i la **seqüencia.**
Els bucles no els mirem perque son mes complicats. Mirarem...

Assignació:

Si tenim una asignacio, torna la mateixa formula de la postcondicio Q pero reemplaçant totes les ocurrencies de la variable "x" per "a que es el que estem assignant.

$$pmd(x{:=}a, Q) = Q[x \mapsto a]$$

Seqüencia:

Procedeix d'una manera seqüencial, es a dir, pas a pas. Si volem calcular la pmd de dos sequencies, primer se calcula la pmd de la ultima (i2) i amb aqueixa precondicio, serveis de postcondicio per la sequencia i1. Si en t inguerem mes de dos, sempre anem de la mes interna cap a fora.

$$pmd(i_1;i_2, Q) = pmd(i_1, pmd(i_2, Q))$$

Condicional:

Aci primer se calcula la pmd dels dos possibles camins, tant la del i1 com la del i2, perque no sabem per quin cami anirem. Després se junta tot en una formula lògica…

Eixa formula diu que se cumpleix la condicio "b" i el pmd(i1,Q) O no se cumpleix la condicio "b" i el pmd(i2,Q)

33/41

Exemple de calcul amb pmd

https://media.upv.es/player/?id=fb61d5a0-a502-11e7-94d5-ad1acb9875fc

```
Comprobar que es correcto

  {x=0 ∧ y=1 ∧ z=2} t:=x; x:=y; y:=t {x=1 ∧ y=0}

Es correcto !

-----------------------------------------------

1) Calcular la precondición más débil:

{P} = {x=0 ∧ y=1 ∧ z=2}
      {P1}
        t := x;
      {P2}
        x := y;
      {P3}
        y := t
{Q} = {x=1 ∧ y=0}

P3 = pmd(y := t, x=1 ∧ y=0) = x=1 ∧ t=0
P2 = pmd(x := y, x=1 ∧ t=0) = y=1 ∧ t=0
P1 = pmd(t := x, y=1 ∧ t=0) = y=1 ∧ x=0

2) Comprobar que x=0 ∧ y=1 ∧ z=2 => y=1 ∧ x=0 ? Sí !
```

34/41
Hoare

V.Propietats Semàntiques: DE SMALL STEP i BIG STEP

35/41

Equivalencia de programes

La idea es utilittzar la semantica operacional que hem vist, la de small step i big step per a raonar i veure certes propietats dels programes. En la diapos 36 mirem per exemple si volem saber quin es mes eficient o si fan el mateix… Segons el que vulguem saber, apliquem un Step o l'altre.

Dos programes P i P' son equivalents respecte a una descripcio Semantica (*1) si el que torna es el mateix: S(P) =S(P') ----> $\mathcal{S}(P) = \mathcal{S}(P')$

(*1): La descripcio semantica se denota aixi: ------> $P \equiv_{\mathcal{S}} P'.$

Exemple de la diapos:

Es veritat o mentida, **depenen de si considerem Small Step o BigStep.**

-En BigStep es veritat perque com el Bstep torna l'estat final, en P tornaria 2 i P'=2 tambe alehores diria que es equivalent.

-En SmallStep no serien equivalents perque el Sstep torna la traça i si fem la traça tant de P com de P', no son iguals, aleshores no son equivalents. En P, primer val 1 la x i despres val 2 la x mentre que en P' un unic pas on se diu que la x=2. Com no es igual, no son equivalents.

36/41
Propiedades semanticas: Example

P Fa una operacio i ho asigna mentre que P' el que fa es un bucle amb iteracions.

-Podriem raonar quin dels dos programes es mes eficient… Per a aixo mirariem la Sstep i contariem el nombre de pasos. En P veiem que es una unica asignacio mentre que en P' te iteracions, te mes instruccions en la traça aleshores es menys eficient.

-Podriem plantejarnos tambe si… ¿Fan el mateix? Pues amb un BigStep ho sabriem, analitzant l'estat final

VII.Implementació dels llenguatges de programacio

37/41

Llenguatges compilats: Te dos fases

1. Un programa font se tradueix mitjançant un compilador a un programa objecte (programa a mes baix nivell).
2. Auqest programa objecte es el que s'executa passant-li unes entrades i tornant-nos uns valor d'eixida.

Llenguatges Interpretats: te una unica fase.

Aci un programa interpret, pren com a entrades un programa font i altres entrades, s'executa i genera una eixida.

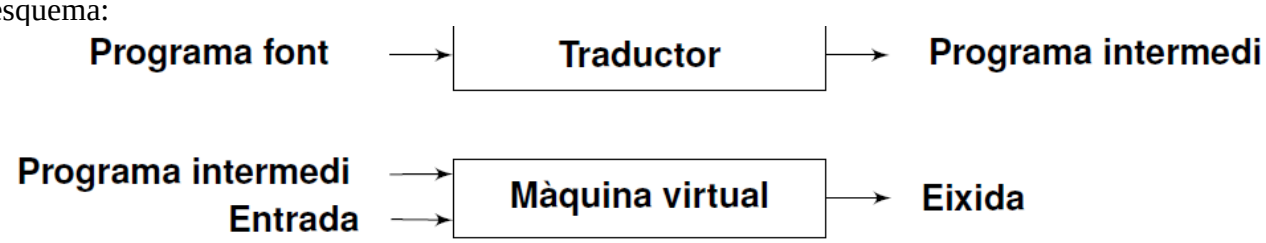
El de compilacio sol ser mes rapid d'executar mentre que el interpretat se l'enten millor pero es mes lent. Els dos son bons, tenen pros i contres. El interpret es millor per a la fase de desarrollo mnetre que el compilat millor per a la fase d'explotació, una vegada anem a implementarlo ja.

38/41

Traducció vs interpretació (I)

No existeix ni compilacio pura ni interpretacio pura.

En el mon real trobarem llenguatges mixtos de compilats e interpretats. Segueix el següent esquema:



39/41

Traducció vs interpretació (II)

Exemples de llenguatges Compilats i interpretats.

Compilats: C, C++, Fortarn, Ada

Interpretats: LISP, ML Smalltalk, Perl,Postscript

Mixtes: Pascal, Prolog, **Java**

40/41

Java Virtual Machine (JVM)

Dona igual la plataforma en la que estem que podem utilitzar la JVM per a generar els .class