Logo Programming Language

Miguel Moreno Gómez (Grup 11)

GRAU-LP — QT 2017/2018

1 Part 1:

1.1 Què és *Logo*?

Logo és un llenguatge de programació dissenyat per Wally Feurzeig, Seymour Paper i Cynthia Solomon l'any 1967 com un projecte del laboratori d'intel·ligència artificial del MIT. Va ser dissenyat com un dialecte de Lisp amb un enfoc educatiu i per a ús domèstic, amb una filosofia "ni terra ni sostre", donant èmfasi a la facilitat d'apendre i a la vegada a la capacitat creativa dels programes que es poden crear amb aquest llenguatge.

Etimològicament, el nom prové de la paraula grega *logos*, que vol dir "paraules", i va ser escollit per diferenciar-lo de llenguatges basats en conceptes més matemàtics i/o abstractes.

1.2 Quins són els objectius i aplicacions de *Logo*?

L'objectiu principal de *Logo* és ajudar a que els nens aprenguin a programar amb un llenguatge de programació complet, i que també tinguin consicència del què fan. És a dir, que mentre dissenyen el codi tenen una idea clara de què és el què s'executarà.

Es pot veure com el predecessor espiritual a eines com ara Scratch o LEGO Mindstorm[1], les quals s'empren actualment per ensenyar a programar a nivell d'educació primària o secundària.

A més d'aplicacions educatives, té aplicacions gràfiques i/o matemàtiques, com mostrar *plots* de funcions, composició de formes geomètriques i fractals, i tractament de llistes. Per acabar, cal remarcar la capacitat artísitca amb la qual es poden arribar a fer programes amb sortides bastant agradabes a la vista.

[1]Tot i que en aquest últim cas LEGO tenia el seu propi intèrpret de Logo, simplement anomenat LEGO *Logo*.

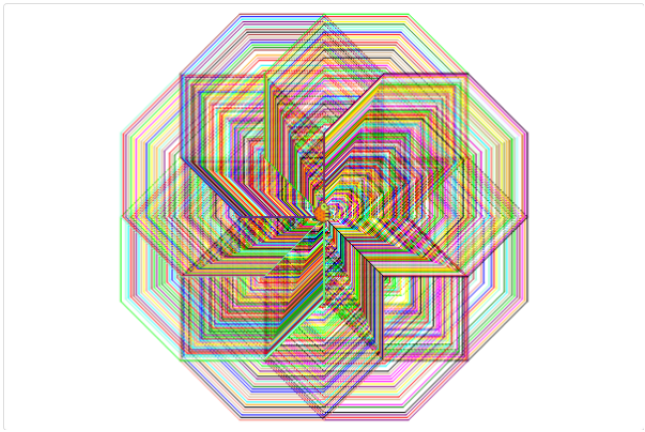

Figure 1: Exemple d'execució d'un programa fet amb *Logo*

1.3 Característiques de *Logo*

1.3.1 Paradigmes de programació

Logo és un llenguatge multi-paradigma; ara explicaré breument els detalls sobre cada paradigma que implementa:

Funcional *Logo* admet funcions d'ordre superior amb una sintàxis similar que la de Haskell per a `map`, `filter` i `reduce` (com a foldr1):

- `map f List`: per exemple, `print map "sum [1 2 3]` = 6
- `filter f List` per exemple, `print filter "numberp [Roundabout by Yes lasts 8 min 37 secs]` retorna `8 37`[2]
- `reduce f List`: similar al `foldr1` de Haskell. Per exemple, `print reduce "difference [8 2 1]` retorna el número `7`
- `cascade x F S`: permet composar funcions F sobre S tantes vegades com sigui x. Per exemple, `print cascade 2 [? * 5] 3` retorna `75` (és a dir, $3x^2$ amb $x = 5$), i `print cascade 3 [? + 7] 11` retorna `32` (és a dir, $3x + 11$ amb $x = 7$).
- `power x y`: Calcula x^y. Per exemple, `print power 2 8` retorna `256`

[2]Un apunt a comentar és que les cadenes de caràcters a *Logo* es representen amb llistes, com ara `[Llista de mida quatre]`, o bé com a paraules delimitades per espais començant amb unes soles cometes dobles, com ara `"word`

- `reverse L`: Inverteix la llista L. Per exemple, `print reverse [hot not mans]` retorna la llista `[mans not hot]`

Noteu que `f` pot ser tant una paraula representant una funció predefinida com una llista que inclogui funcions o funcions anònimes:

Les funcions anònimes de *Logo* representen les lambdes com a interrogants, seguides d'un número si n'hi ha més d'un paràmetre a la funció (si només n'hi ha un, és vàlid posar tant `?` com `?1`).

Per exemple, `show filter [?1 < 5] [0 2 4 6 8]` retorna la llista `[0 2 4]`.[3]

Imperatiu/Procedural *Logo* permet dissenyar i implementar procediments (o subrutines) per ajudar a fer programes més senzills de llegir. A més, té instruccions de condicional (`if .. ifelse .. else`) i de salt en forma de bucle (`repeat n [ .. ]`). Per exemple, aquest programa dibuixa una versió recursiva del triangle de Sierpiński:

```
TO Sierpinski :n
    ifelse (:n >= 5) [
    repeat 3 [
        fd :n      rt 120
        Sierpinski (:n / 2)
       ]
    ] [stop] ; else
END
cs  rt 30   Sierpinski 200
```

[3]La diferència entre `print` i `show` és que `print` mostra el contingut de les llistes, i `show` mostra les llistes amb els claudators que la limiten.

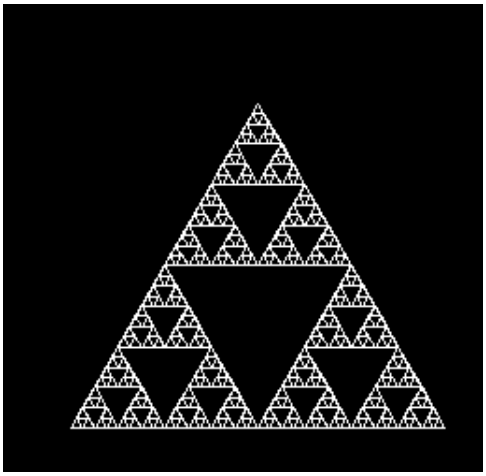

Figure 2: Resultat de la funció anterior, feta amb turtle graphics

Reflectiu *Logo* permet modificar el comportament dels programes en temps real, amb les següents instruccions:

- `text "name`: retorna el contingut del procediment de nom `name` en forma de llista, donant primer els paràmetres a passar i llavors el seu contingut. Si n'hi han comentaris, els omet. Els mètodes primitius (o prèviament definides), però, només retornaràn que `name is a primitive`
- `define "name List`: Donat un procediment en forma de llista ben formada, `List` i un nom per al procediment `name`, defineix un procediment de nom `name` que executará les instruccions contingudes a `List`. La part reflectiva de *Logo* és que `name` pot ser el mateix que el d'una funció prèviament definida, i això es pot fer en temps real, **amb una excepció:** les funcions ja prèviament definides, que es consideren com a primitives.

Un exemple de programació reflectiva en *Logo* és:

```
TO foo :n
    if :n = :n [ ; being 200% sure
               output :n ; return it
         ]
END

show foo 7   ; foo --> id :n
show text "foo
define "foo [[n] [if :n = :n [output :n+1]]]
show foo 7   ; foo --> succ :n
show text "foo
```

L'execució d'aquest bloc retorna:

```
> 7
> [[n] [if :n = :n [output :n]]]
> [[n] [if :n = :n [output :n+1]]]
> 8
```

Per acabar, una característica interessant de *Logo* és que hi han abreviatures per algunes de les instruccions del llenguatge, que serveixen per escriure codi més ràpidament sacrificant una mica de llegibilitat. Es veuràn, sobretot, a l'apartat de Turtle rendering.

1.3.2 És *Logo* compilat o interpretat?

Logo es tracta d'un llenguatge interpretat, tret de dues implementacions, les quals són compilades: Lhogho i Liogo. Admet càrrega de fitxers des del mateix intèrpret però no la seva compilació amb la instrucció `load "filename.lgo` i desar-los amb `save "filename.lgo`, on `filename.lgo` és un fitxer amb codi en *Logo*.

Hi han diversos intèrprets de *Logo* (més de 300!), i alguns d'ells són:

- **UCBLogo/Berkeley Logo**: L'intèrpret *de facto* del llenguatge (*Logo* no en té, d'oficial), dissenyat per la Universitat de California, Berkeley i publicat per primer cop l'any 1992. Ara bé, la seva última *release* va ser al 2009.
- **MSWLogo**: intèrpret amb capacitat per a múltiples cursors, 3D. Ha evolucionat en **FMSLogo** que, tot i ser software lliure, només ha sigut publicar per a Windows.
- **Atari Logo i Commodore Logo**: versions de *Logo* dissenyades per a ser programades des de consoles Atari i computadores Commodore.
- **MicroWorlds (vanilla, EX, Jr)**: una versio de *Logo* dissenyada pel propi creador, Seymour Papert. A diferència d'altres intèrprets, aquest és propietari.

Ara bé, compte amb **NetLogo**, **StarLogo**, **LEGO Logo** i **LibreLogo**, que tot i portar el nom, no són *Logo per se*, sino o bé extensions del llenguatge, o entorns que han agafat *Logo* com a inspiració.

1.3.3 Sistema de tipus

El sistema de tipus de *Logo* té comprovació dinàmica, ja que tracta les dades en temps d'execució, i és *type safe* ja que els errors de tipus es tracten temps d'execució. Per exemple, l'execució de `show map "sum [1 2 3]` imprimirà el `6`

per consola, però executar `map "sum {1 2 3}` i detectarà un error que imprimirà com a `bfs doesn't like 1 2 3 as input in map1`, ja que la funció`sum` admet llistes com a paràmetres però no arrays.

Al igual que Lisp, *Logo* té tipat fort, cosa que no ens permet barrejar elements de tipus diferents. Un exemple és la instrucció `print sum 1 "5`, que retorna el mateix que `print sum "1 5`, `print sum "1 "5` o `print sum 1 5`, és a dir, `6`, però `print sum 1 [5]` o `print sum 1 {5}` dóna error perquè està barrejant paraules (expressades com a números) i llistes/arrays.

1.4 Turtle rendering

La característica principal de *Logo*, de la qual és pioner, és la capacitat de generar dibuixos mitjançant un cursor. Es diu *turtle* rendering com a analogia de què el cursor té el mateix comportament que una tortuga (o un tanc): es pot moure endavant i endarrere i girar sobre sí mateix. El cap de la tortuga representa la direcció actual. Algunes implementacions de *Logo* admeten múltiples cursors però, per simplicitat, asumirem que en fem servir un, tal i com es fa servir al intèrpret UCBlogo.

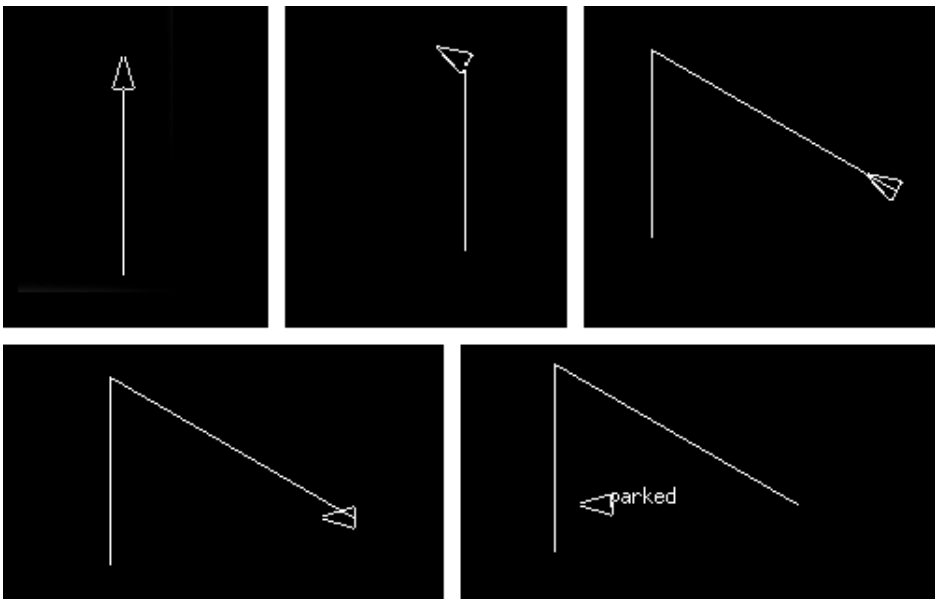

Figure 3: Exemple de turtle rendering amb *Logo* amb les instruccions següents: `fd 100`, `lt 30`, `bk 150`, `rt 330`, `pu`, `fd 100`, `label "parked`

El sistema de coordenades de *Logo* és cartesià, és a dir, donat un punt representat com a (X, Y), el centre de la pantalla és el punt (0, 0), anant a la dreta

incrementarà la coordenada X, i anant endavant incrementarà la Y; les operacions contràries donaràn resultats inversos.

Cal remarcar que *Logo* no és exclusivament un llenguatge dissenyat per generar imatges amb turtle rendering: aquesta és simplement una de les seves característiques.

Hi han diverses operacions relacionades amb el turtle rendering. En parèntesis, les abreviacions de les funcions, si n'hi han:

- **Operacions de control:** serveixen per controlar la direcció i el comportament del cursor.
  - `forward, back :num (fd, bk :num)`: mou el cursor endavant i endarrere tants píxels com `:num`
  - `left, right :deg (lt, rt :deg)`: mou el cursor en un angle de `:deg` en la direcció assignada. Evidentment, `:deg` té rang entre 0 i 360
  - `setpos [X Y]`: posa el cursor en les coordenades (X, Y) de la pantalla. Compte, si el cursor està en mode `pd`, dibuixarà en pantalla! Les instruccions `setx X` i `sety Y` fan el mateix per a cada coordenada.
  - `hideturtle (ht), showturtle (st)`: amaga i mostra el cursor visualment, respectivament. Això vol dir que, si s'amaga el cursor amb `pendown` activat i es mou, seguirà dibuixant!

  Algunes interpretacions de *Logo* permeten modificar el comportament de dibuix del cursor:

- **Operacions de color:** serveixen per definir com es dibuixa en pantalla.
  - `setpencolor [R G B] (setpc [R G B])`: fixa el color del cursor amb els colors fixats en format RGB.
  - `penup (pu)`: aixeca el cursor: tot moviment que es faci mourà el cursor, però no dibuixarà res.
  - `pendown (pd)`: posa el cursor en mode dibuix. Per defecte *Logo* comença dibuixant així.
  - `penerase (pe)`: posa el cursor en mode dibuix, però ara el cursor *esborrarà* tot allò que es trobi pel camí.
  - `label "writeme`: escriurà en pantalla el text `writeme` en la posició del cursor.
  - `clearscreen (cs)`: esborra tot el què s'ha dibuixat des del principi i retorna el cursor a la posicio (0, 0).
  - `clean`: el mateix que `cs`, pero sense moure el cursor.

Degut al fet de no haver-hi una interpretació estàndar, és possible que altres intèrprets de *Logo* tinguin noms diferents pels mateixos procediments, o que uns tinguin funcions que altres no tenen per defecte.

2 Part 2: Referències bibliogràfiques

2.1 Introducció

Sota aquesta secció es troba la bibliografia. Descric els comentaris més adients d'aquesta:

- La informació s'ha contrastat entre tots els llocs per comprovar la seva veracitat. El mateix s'aplica amb tots els codis d'exemple d'aquest document, els quals s'han provat en diversos intèrprets i funcionen; en alguns casos, però, s'han fet lleugers canvis de sintàxis per garantir la compatibilitat entre els intèrprets.

- A excepció de la primera imatge, que ha sigut agafada d'un codi fet a turtleacademy.com, la resta d'imatges han sigut generades amb UCBlogo.

- De totes les fonts d'informació, la sèrie de llibres d'en Brian Harvey han resultat, en la meva opinió els més fiables per diverses raons: la seva experiència tant acadèmica com en el desenvolupament de UCBlogo i l'extensa i clara documentació sobre aquest intèrpret.

- Per contrast, la menys fiable és la Viquipèdia pel simple fet de què tothom la pot editar, però la he fet servir com a referència en respecte els temes on es pot ampliar el coneixement. La del MIT, només la he fet servir per als primers apartats, ja que no hi trobava més informació que considerès destacable o necessària.

References

[Har97a] Brian Harvey. *Computer Science Logo Style Volume 1: Symbolic Computing*. MIT, 1997. ISBN: 9780262581486. URL: `https://people.eecs.berkeley.edu/~bh/v1-toc2.html`.

[Har97b] Brian Harvey. *Computer Science Logo Style Volume 2: Advanced Techniques*. MIT, 1997. ISBN: 9780262581493. URL: `https://people.eecs.berkeley.edu/~bh/v2-toc2.html`.

[Pap99] Seymour Papert. *Logo Philosophy and Implementation*. MIT, 1999. ISBN: 2-89371-494-3. URL: https://people.eecs.berkeley.edu/~bh/v2-toc2.html.

[Boy14] Pavel Boytchev. *Logo Tree Project*. http://elica.net/download/papers/LogoTreeProject.pdf. Accessed: 2017-12-30. 2014.

[Aut] Various Authors. *Logo (programming language)*. https://en.wikipedia.org/wiki/Logo_(programming_language). Accessed: 2017-12-30.

[Bel] Joshua Bell. *Online Logo Interpreter*. http://www.calormen.com/jslogo. Accessed: 2017-12-30.

[Fou] The LOGO Foundation. *Logo History*. http://el.media.mit.edu/logo-foundation/what_is_logo/history.html. Accessed: 2017-12-30.

[HLM] HLModTech. *Color Tunnel-supercharged*. https://turtleacademy.com/view/programs/54dd6778f458598e3e3c9869/en. Accessed: 2017-12-30.