Exercicis paramètrics via *web*

por M. Dalmau, S. Martin i J. Saludes

Universitat Politècnica de Catalunya

Correo-e: miquel.dalmau@upc.es, jordi.saludes@upc.es, sebasm@mat.upc.es

1 El contexte

L'any 2000 els professors que estàvem a càrrec de l'assignatura *Mètodes matemàtics 1* (obligatòria de la carrera d'Enginyeria Industrial a l'ETSEIT) vam decidir complementar les exposicions teòriques i les classes de problemes amb sessions pràctiques sobre els temes del curs. L'objectiu d'aquestes sessions és de familiaritzar l'estudiant amb les dificultats de la programació efectiva dels mètodes de càlcul numèric que són l'objecte principal de l'assignatura.

Ara bé: Les experiències prèvies dels autors en aquests tipus de pràctiques aconsellaven portar-les a terme en grups molt reduïts, ja que l'estudiant requereix atenció personalitzada per tal de trobar què falla en el seu programa; però al mateix temps el professor ha de validar els resultats obtinguts per tal d'assignar una nota a cada estudiant.

Tant l'oferta del centre com les possibilitats del departament requerien formar grups de 20 estudiants (pel cap baix) sota la supervisió d'un únic docent. En aquestes condicions, no fóra possible fer més de quatre pràctiques per persona i quadrimestre.

És per això que vam plantejar-nos de automatitzar la tasca de correcció dels resultats: Si construiem un sistema informàtic pels estudiants que pogués supervisar llurs respostes, el personal podria dedicar-se exclusivament a la tasca d'atendre consultes i tot el ritme de la classe s'acceleraria considerablement.

1.1 L'experiència

El sistema que en aquell moment es va desenvolupar s'usa a l'assignatura quadrimestral de *Mètodes matemàtics 1* de la carrera d'Enginyeria Industrial de l'ETSEIT des del curs 2000-01 i s'han completat amb èxit prop de 7.000 exercicis, que comptant una mitjana de 120 alumnes/quadrimestre ens dóna aproximadament 7 exercicis per persona (tenint en compte que als alumnes que repeteixen no refan els exercicis).

Per regla general, una bona part dels alumnes assisteixen regularment a les sessions pràctiques, que es fan a raó de 2 hores cada 15 dies. Un altre grup, més reduit, fa les pràctiques des de casa i sense ajuda dels professors.

2 El sistema actual

Consta de

- Un *motor de càlcul*.
- Una interfície amb l'usuari.
- Una base de dades *SQL* per a recollir l'evolució de cada estudiant.

El *motor de càlcul* calcula totes les expressions matemàtiques que s'han de mostrar a l'alumne i fa les comporvacions. Actualment, aquest motor de càlcul és un "kernel" de *Mathematica* [Math] i la comunicació entre la interfície i el *kernel* es basa en la llibreria *MathLink*.

La interfície d'usuari (que està basada en *web*) envia cadenes de caracters al motor de càlcul que són avaluades com a expressions *Mathematica* i envia missatges d'error a la interfície (també en forma de cadenes de caracters).

Tant la interfície d'usuari com la connexió amb el motor de càlcul i la base de dades s'ha construit amb el llenguatge *Python*, [Python] mitjançant llibreries especialitzades en cada un d'aquests components.

Cada exercici consta d'un directori en un lloc determinat del servidor *web* que contè:

- Un fitxer de codi *Mathematica* de nom `init.m` on es calculen totes les entitats matemàtiques que apareixen a l'exercici. Aquestes entitats es generen aleatòriament amb el generador de números pseudoaleatoris del motor de càlcul, usant com a *llavor* el número de DNI de l'estudiant. Això garanteix que a cada estudiant li correspon un exercici diferent (anomenem això una *instanciació* de l'exercici)
- Un o més fitxers que determinen l'aspecte *web* de l'exercici: Bàsicament un fitxer de nom `index.html` que pot portar associats fitxers d'imatges o vincles a altres documents. Aquest fitxer principal és processat per la comanda `Splice` de *Mathematica* que permet avaluar expressions d'aquest llenguatge incrustades al texte `HTML` mitjançant els delimitadors `(*` i `*)`.

A banda de generar els objectes matemàtics personalitzats, el fitxer `init.m` conté una descripció de les dades que s'esperen a cada una de les caixes de formulari que l'estudiant ha d'omplir:

- El *tipus* de dades que s'espera: Un enter, un real, una llista de reals, un polinomi, etc. Així si especifiquem `poly[x]` vol dir que esperem que l'estudiant escrigui quelcom que es pugui interpretar com un polinomi en la variable x. Si posem `{integer}` volem dir que esperem una llista d'enters, etc.
- Un conjunt de regles de la forma $\langle \textit{condició} \rangle \Longrightarrow \langle \textit{acció} \rangle$ que detecten possibles errors en la resposta de l'estudiant i li donen informació sobre l'error comés. Per exemple hom pot imposar com a condició que el grau d'un polinomi sigui n (Una variable precalculada a `init.m`) i si la condició no es compleix, assenyalar a l'estudiant que el grau del polinomi ha de ser tal.

El sistema associa a cada *tipus* de dades les funcions següents:

`in` Converteix la cadena de caracters que ha introduit l'estudiant a la caixa de formulari en una expressió *Mathematica* del tipus corresponent. Si no es pot interpretar com un element d'aquest tipus, dóna un missatge d'error a l'estudiant (Error de *sintaxi*).

`equal` Avalua si dues expressions d'aquest tipus són iguals. Això permet simplificar la sintaxi a l'hora de comprovar, per exemple, que dues llistes `x` i `y` d'enters (és a dir de tipus `{integer}`) són iguals: Només cal posar `x==y` i el sistema ja entén que ha de comprovar la igualtat element per element.
Això també permet modificar el comportament per defecte del motor de càlcul en alguns tipus. Per exemple, al tipus `real`, hom no pot confiar en l'operació d'igualtat de *Mathematica*, ja que no té en compte els errors d'aproximació que inevitablement apareixen al càlcul numèric. En lloc d'això, podem definir una funció `equal` per al tipus `real` que calculi l'error relatiu entre els dos elements comparats i digui que són iguals si aquest és prou petit.

`test` Un cop passat `in`, comprova condicions addicionals per admetre l'expressió com pertanyent al tipus.

`name` Retorna un parell de cadenes de caracters amb el nom del *tipus* (en singular i plural) que es poden usar al confegir automàticament missatges d'error pels estudiants.

`out` Converteix una expressió *Mathematica* en una cadena de caracters en la notació que l'estudiant coneix. Juntament amb la funció `in`, ens permet deslliurar-nos de la sintaxi del motor de càlcul i presentar o acceptar expressions en el llenguatge habitual dels estudiants.

`random` Genera aleatòriament un element del tipus donat.

`form` Construeix una cadena de caracters que mostri una caixa de formulari apta per a rebre una expressió del tipus donat. Així per exemple, hom pot desplegar caixes de més d'una línia per a contenir llistes, mentre es mostren caixes petites per a contenir números.

Entre els tipus que usem a *Mètodes matemàtics 1* hi comptem: Llistes, reals, enters, racionals en expressió binària periòdica, intervals de números reals i polinomis.

3 Desenvolupament futur

Part de les idees que s'exposaran en aquest apartat estan recollides a [SMD].

Cal parlar de les tres menes d'usuari que tenen a veure amb un sistema d'exercicis com aquest:

L'estudiant: que és la persona al qual s'adrça el sistema.

L'autor: que és la persona que escriu l'exercici. Normalment un professor de l'assignatura.

El desenvolupador: que és qui crea o adapta el sistema informàtic que gestiona el conjunt d'exercicis.

Cada un d'aquests usuaris té unes capacitats, interessos i actituds diferents. En la situació actual, els rols d'autor i de desenvolupador es confonen. És per això que és difícil que una experiència com la que estem tractant pugui ser exportada a altres entorns: Tota cosa que no sigui una còpia literal d'un sistema, necessitarà adaptacions que exigiran un ampli coneixement del sistema a la persona encarregada de fer l'adaptació.

En la majoria dels casos, el gruix de l'adaptació a un altre entorn educatiu consisteix en la creació de nous continguts, és a dir, és feina de l'autor. Sembla evident, que no es pot demanar als professors d'una assignatura que, a banda dels coneixements propis, coneguin a fons el sistema informàtic i el llenguatge del motor de càlcul que es pretén implantar.

És per això que pensem que caldria desenvolupar un **llenguatge de molt alt nivell** que permetés d'expressar els passos de resolució i les comprovacions que cal fer per resoldre un exercici, d'una manera senzilla i independent del motor de càlcul subjacent.

3.1 Requeriments de llenguatge

A la vida real, hom tracta els *problemes* per via mecànica o humana. Tant en un com en altre cas, podriem dir que un problema és caracteritza per:

- Dades: Objectes matemàtics $a_1 \in T_1, ..., a_{\mathrm{m}} \in T_{\mathrm{m}}$ que pertanyen a certs dominis T_{i} (*tipus*) i que identifiquen el problema.
- Incògnites: Objectes matemàtics $x_1 \in T_1^0, ..., x_{\mathrm{n}} \in T_{\mathrm{n}}^0$, la determinació dels quals constitueix la resolució de l'exercici.
- Condicions: Són predicats $c_{\mathrm{i}}(a_1, ..., a_{\mathrm{m}}; x_1, ..., x_{\mathrm{n}})$ (funcions a valors booleans) que lliguen les dades i les incògnites. La satisfacció de totes les condicions associades a un problema és el que demostra que els valors de les incògnites que s'han triat corresponen a una solució veritable del problema.

Com a exemple, considerem el problema de trobar el màxim comú divisor de dos enters positius a i b. Les dades són a i b que pertanyen al domini dels naturals. La incògnita és un d que també pertany al mateix domini i les condicions associades són les següents:

i. $c_1(a, d) = \{d \text{ divideix } a\}$;

ii. $c_2(b, d) = \{d \text{ divideix } b\}$;

iii. $c_3(a, b, d) = \{d \text{ és el més gran que compleix a la vegada } c_1 \text{ i } c_2\}$.

Existeixen llenguatges de programació (els anomenats *llenguatges lògics*) que permeten expressar tots els tres components esmentats d'un problema, de cara a la cerca automàtica de solucions. Proposem basar-nos en llenguatges d'aquest estil per a expressar un sistema de guia que permeti a l'estudiant humà cercar la solució del problema. Les diferències amb un sistema de solució automàtica són evidents: Un estudiant necessita que l'exercici li sigui presentat en una notació que ell conegui i cal permetre en tot moment que l'estudiant porti la iniciativa.

En general el procés de resolució d'un problema consta de:

1. Presentació de l'exercici en forma adient (Això inclou l'elecció de la notació per a cada tipus de dades i la creació de caixes de formulari per tal que l'estudiant pugui introduir els valors de les incògnites).

2. L'estudiant pot optar entre:
   a) Donar valor directament a les incògnites. En aquest cas, el sistema comprova les condicions i si n'hi alguna que falla, ho indica a l'estudiant amb un missatge entenedor.
   b) Demanar ajuda al sistema. En aquest segon cas, el sistema descomposa l'objectiu principal en sub-problemes. Hi ha dues possibilitats:
      - Es mostren estratègies alternatives: Qualsevol d'elles condueix a la solució. L'estudiant n'ha de triar una.
      - L'objectiu principal es pot obtenir resolent seqüencialment uns sub-problemes. En aquest cas, el sistema ha de mostrar el pla general d'atac i anar conduint l'estudiant a cada pas.

   En els dos casos, la situació reverteix a l'exposició d'un altre problema: Reprenem l'etapa 1 pel nou problema.

3.1.1 Concurrència

Com es veu el sistema ha de respondre a esdeveniments generats per l'estudiant (demanar ajuda, especificar el valor d'una incògnita). A efectes docents, cal destacar que s'ha d'assenyalar que una condició ha fallat en el precís moment en que es té prou informació per fer-ho: Moltes vegades no cal esperar a que totes les incògnites estiguin determinades per a decidir que la solució és incorrecta. Per exemple, si demanem a l'estudiant de trobar un quadrat màgic, en el moment en què una fila no suma el que caldria, ja podem avisar l'estudiant que no cal que tiri endavant: És molt més informatiu procedir així que no pas esperar a que l'estudiant hagi omplert totes les caselles per a donar la llista de totes les condicions que no es verifiquen.

Idealment, el llenguatge hauria de generar un *thread* per a cada condició a comprovar. Cada un d'aquests *threads* quedaria en suspens fins que totes les incògnites que intervenen en la condició fossin determinades, en aquest moment es comprovaria la condició i es generaria el missatge d'error a l'estudiant si aquesta fallés. Altres *threads* estarien a l'aguait de altres accions de l'usuari (prèmer botons, etc).

Aquestes consideracions fan que el llenguatge hagi de ser de tipus lògic i *concurrent*

3.1.2 Sistema de tipus

Una peça clau en el desenvolupament futur del sistema és garantir que la creació de nous exercicis parametritzats sigui senzilla per l'autor. En particular demanariem que l'autor no li calgués de conèixer el llenguatge del *motor de càlcul* subjacent del sistema i que bona part de la feina de conversió ja estigui feta en el propi sistema. Per això cal reforçar el sistema de *tipus* fent-los paramètrics i permetent la sobrecàrrega d'operadors.

Com a primera tasca el tipus d'una variable ha de saber convertir un objecte d'aquest tipus de la notació de l'usuari a la forma interna i a la recíproca (Essencialment la feina que feien fins ara les funcions `in` i `out` del l'apartat 2). A la taula següent hem posat alguns exemples de tipus de dades amb la forma interna i la notació de l'usuari. En el primer cas ens referim a números racionals en forma binària: el signe '`%`' a la notació `0.1%01` l'usem per denotar el període (es tracta doncs del número 0.101010...).

Tipus	Forma interna	Notació de l'usuari
`radix 2 rational`	`rational(2,3)`	**`0.1%01`**
`interval of reals`	`ival(3.5,4.7)`	**`3.5..4.7`**
		`1`
`list of integers`	`[1 2 5]`	**`2`**
		`5`

Tabla 1. Alguns tipus paramètrics amb representacions internes i externes.

Una altra tasca seria inferir el domini on una expressió binària que impliqui dos operands de tipus diferents, tingui sentit. Així, per exemple, la expressió `X+Y`, on `X` és `rational` i `Y` és `integer` s'ha d'interpretar en el domini dels racionals (que conté els dominis originals de `X` i `Y`). Això es podria fer definint *inclusions* $i: T_1 \longrightarrow T_2$ entre dominis que permetin interpretar un element $x \in T_1$ com si fos a T_2.

3.2 Alguns experiments

Fins al moment present s'han fet algunes experiències usant com a llenguatge base *Oz/Mozart* [Oz]: un llenguatge concurrent multiparadigma que permet descriure tot el que hem anat esmentant. És encara massa aviat per a fixar una sintaxi determinada al llenguatge que estem buscant, però una bona aproximació seria quelcom que es pogués compilar a codi *Oz/Mozart* com es veurà als exemples de l'annex.

4 Referències

[Oz] The Mozart programming system. `http://www.mozart-oz.org`

[Python] Python programming language. `http://www.python.org`

[Math] *Mathematica*. `http://www.wolfram.com`

[SMD] J. Saludes, S. Martín, M. Dalmau.
Parametric Exercises with Zope server. A *Internet Accessible Mathematical Computation*. IAMC2002 (Lille). `http://www.symbolicnet.org/conferences/iamc2002`

5 Annex: Exemples

La sintaxi d'aquests exemples segueix de prop la de *Oz/Mozart*. Per entendre aquests exemples tingueu en compte que `proc` introdueix un nou procediment. Entre claus es llisten el nom del procediment i els seus arguments. Alguns d'aquests poden estar indeterminats (En particular els valors retornats pel procediment són indeterminats a l'iniciar la crida del procediment). L'entorn `unless` inicia un nou *thread* que queda en suspens fins que les variables que intervenen en la condició no estan determinades. Si, en aquest moment, la condició és falsa es continua l'execució; si la condició és certa l'execució acaba.

5.1 Polinomi interpolador

```
proc {Interpola P X Y}
  unless eval(P,X)=Y
    say 'La interpolació és incorrecta'
    fail
  end
end
```

Observeu en aquest primer exemple l'ús de l'operador '=' per a comparar directament dues llistes de nombres reals.

Un altre exemple que estén l'anterior

```
proc {Interpola P X Y}
  % P és el polinomi interpolador dels valors (Xi,Yi)
  let N be degree(P)
  unless length(X)=N+1
    say 'El polinomi ha de ser de grau $N$'
```

```
    fail
  end
  proc {Partial X Y}
    on case
      let X1|Xs be X
      let Y1|Ys be Y
      unless eval(P,X1)=Y1
        say 'El polinomi no interpola al punt $X1$, hauria de donar
$Y1$'
        fail
      end
      {Partial Xs Ys}
    end
  end
  {Partial X Y}
end
```

Se suposa que tenim el tipus `polinomi` definit, i que (a part de les funcions esmentades més amunt) hi ha `eval(P,X)` que pren un polinomi i, o bé un real, o bé una llista de reals i en retorna, en el primer cas, un real i en l'altre cas una llista de reals.

5.2 El màxim comú divisor

En aquest exemple mostrem com podem deixar triar a l'estudiant quina estratègia vol seguir per fer el càlcul del *màxim comú divisor* de dos números. Suposant un sistema que generi els valors de `A` i `B`, cridariem el procediment següent on volem obtenir el resultat a la indeterminada `D`. La comanda `on help` inicia un nou *thread* que espera a que l'estudiant demani ajuda

```
proc {BuscaMCD A B D}
  say "Trobeu el màxim comú divisor de `A` i `B`"
  say "poseu el resultat a `D`"

  on help
   make E1 a button with action {BuscaPerEuclidi A B D} and
        name 'Euclidi'
   make E2 a button with action {BuscaPerDescomposicio A B D} and
        name 'Descomposant'
   say "Teniu dues estratègies: `E1` i `E2`. Trieu la que preferiu"
  end

  unless A mod D = 0
    say "El mcd ha de dividir `A`."
    fail
  end

  unless B mcd D = 0
   say "El mcd ha de dividir `B`."
   fail
  end

  let Dt in {Euclid A B Dt}
  on D>Dt
   say "Hi ha un divisor comú de `A` i `B` més petit que `D`."
   fail
  end
end
```

Les dues estratègies es descriuran així:

```
proc {BuscaPerEuclidi A B D}
  make Ds a list of integers with A|B|Rs
  make T a table with rows [Ds none|Qs]
  say "Disposem els valors `A` i `B` en una taula de dos files com
       segueix"
  say T
  {Euclidi A B Rs Qs D}
end
```

El procediment `BuscaPerEuclidi` serveix per a formar la pantalla d'entrada a l'algorisme d'Euclidi. La resta és cosa del procediment `Euclidi`:

```
proc {Euclidi A B Rs Qs D}
   if B=0 then
    let D be A
    let Rs be []
    let Qs be []
    say "Hem arribat a zero: El mcd és `A`"
   else
    make R, Q integers
    let R|Rt be Rs
    let Q|Qt be Qs
    say "Dividiu `A` per `B`, poseu el quocient a la casella sota `B`
i el resta a la casella de la dreta."
    unless Q*B+R=A
      say "La divisió de `A` per `B` està mal feta."
      fail
    end
    {Euclidi B R Rt Qt D}
   end if
end
```

La segona estratègia consisteix a descomposar en factors primers `A` i `B`:

```
proc {BuscaPerDescomposicio A B D}
  say "Anem a descomposar 'A' i 'B' en factors primers".
  {Descomposa A Fa}
  {Descomposa B Fb}
  {McdPerFactors Fa Fb D}
end
```

No detallem aquí la composició de `Descomposa`. Direm només que `Fa` i `Fb` són membres del tipus `bag of integers`, que és quelcom que es troba a mig camí entre un *conjunt* i una *llista*: A diferència del conjunt, un element és pot repetir; però a diferència de la llista l'ordre no és important. L'usem aquí com a estructura de dades adient per a representar descomposicions en factors primers: La descomposició $3^2 \cdot 5 \cdot 7$ correspondria a la *bossa* $\{3, 3, 5, 7\}$.

5.3 Expressions binàries

El següent exemple demana trobar aproximacions per defecte i excés a un número racional donat, tenint en compte que les aproximacions han de ser números en coma flotant d'una certa llargada de mantissa binària. És un dels exercicis que posem a *Mètodes matemàtics 1*.

```
proc {AproximacionsMaquina A L}
```

```
  % Troba aproximacions del racional A
  make A1 a radix 2 rational
  make A2 a radix 2 rational
  say "Trobeu les aproximacions per defecte i excés al número 'A' en
una màquina binària amb longitud de mantissa `L`."
  say "Per defecte: `A1`."
  say "Per excés: `A2`."

 {BinAprox A A1 A2 L}
end

proc {BinAprox A A1 A2 L}
  % Aqui falta definir IsMachine

  unless A1<=A
   say "L'aproximació per defecte ha de ser menor que el número"
   fail
  end

  unless A2>=A
   say "L'aproximació per excés ha de ser més gran que el número"
   fail
  end

  unless {IsMachine A1}
   say "`A1` no és un número de màquina"
   fail
  end

  unless {IsMachine A2}
   say "`A2` no és un número de màquina"
   fail
  end

  let Epsilon be 2**(-L)
  unless A1+A*Epsilon>A
   say "Hi ha una aproximació per defecte més gran."
   fail
  end

  unless A2-A*Epsilon<A
   say "Hi ha una aproximació per excés més petita."
   fail
  end

end
```