EPS
UdG Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Tracejador Interactiu de Lambda-Càlcul Web – TILC(W)

Document: Memòria

Alumne: Oriol Gallart Garangou

Director/tutor: Mateu Villaret Ausellé
Departament: Informàtica, Matemàtica Aplicada i Estadística
Àrea: Llenguatges i sistemes informàtics (LSI)

Convocatòria (mes/any): 06/2015

Índex

1. Introducció

Actualment, a la Universitat de Girona s'ofereix el Grau en Enginyeria Informàtica on al 3r curs es cursa l'assignatura de "Paradigmes i Llenguatges de Programació".

Dins d'aquesta assignatura s'ensenyen diferents paradigmes que no es presenten fins aquest curs i apareix la programació funcional, concretament es presenta la base d'aquesta, el Lambda-Càlcul.

La Teoria del Lambda-Càlcul pot resultar complicada o difícil d'entendre en un primer moment. L'objectiu d'aquest projecte doncs és facilitar aquest aprenentatge mitjançant una aplicació web gràfica que presenti aquests conceptes d'una manera visual.

El que es pretén doncs és crear una eina de caràcter educatiu on-line a disposició dels estudiants que cursen la matèria i per als mateixos professors que imparteixen l'assignatura.

Per entendre millor l'informe farem una descripció a grans trets del Lambda-Càlcul.

1.1 Descripció Informal del Lambda-Càlcul

En lògica matemàtica i computació, el Lambda-Càlcul és un sistema formal dissenyat per investigar la definició i l'aplicació de funcions i la recursivitat.

Per altra banda, el Lambda-Càlcul pot ser anomenat el llenguatge de programació universal més petit, ja que consisteix en una única regla de transformació (anomenada beta-reducció) i una única definició de funció.

El lambda-càlcul també ha influït de manera substancial en els llenguatges de programació funcional, com per exemple el LISP, ML i el Haskell. També ho fa en altres llenguatges no funcionals com el C++, l'Scala o el Go.

En el lambda-càlcul, totes les expressions representen funcions compostes d'un únic argument. Aquest, a la vegada, pot ser un funció d'un sol argument i també el cos de la funció pot ser una nova funció. Totes les funcions són anònimes, estan definides per expressions lambda, que són les que diuen el que es fa amb el seu argument.

Considerem, per exemple, considerem la funció "suma 2":

$f(x) = x + 2$

en lambda-càlcul aquesta funció quedaria expressada com:

$\lambda x.\ x + 2$

(o també es podria expressar com a $\lambda y.\ y + 2$ o $\lambda z.\ z + 2$, ja que el nom de l'argument no importa).

Per tant doncs, el número f(3) vindria donat per la següent expressió:

(λ x. x + 2) 3

L'aplicació de funcions en lambda-càlcul és associativa a l'esquerra:

f x y z = (((f x) y) z)

Si ara considerem la funció que aplica una funció al número 3:

λ f. f 3

podríem passar-li "sumar 2", de manera que ens quedaria:

(λ f. f 3)(λ x. x + 2)

Tot i que no entrarem en detall del perquè fins més avançada la memòria, les tres expressions següents equivalen al mateix:

(λ f. f 3)(λ x. x + 2) = (λ x. x + 2) 3 = 3 + 2

No totes les lambda-expressions poden reduir-se a un valor finit. Considerem la següent expressió:

(λ x. x x) (λ x. x x)

si mirem de visualitzar què passa quan a la primera funció s'hi aplica l'argument:

(λ x. x x)

veurem que tota l'estona el terme es va replicant a si mateix i no s'acaba mai.

Les lambda expressions contenen variables lliures i lligades per alguna λ. Per exemple, la variable 'y' apareix lliure en l'expressió (λ x. y), però apareix lligada a (λ y. y).

Tot i que el lambda-càlcul en si mateix no conté símbols com enters o els signes '+','-', aquests poden ser definits i se'ls sol anomenar definicions o combinadors. Alguns exemples de definicions o combinadors serien els següents:

IDENTITAT = λ x. x

TRUE = λ x. λ y. x

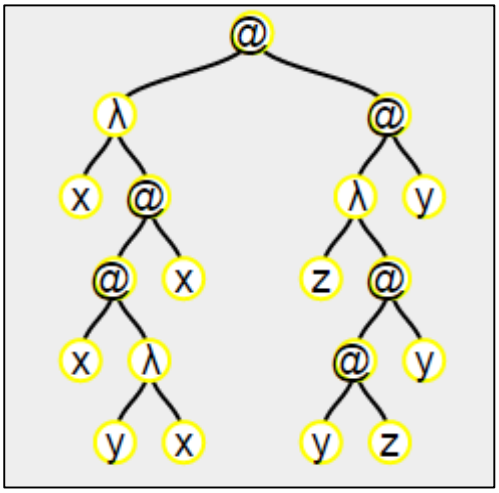
FALSE = λ x. λ y. y

Fins aquí la descripció informal del lambda-càlcul, que permet veure el nivell tècnic d'aquest formalisme i on es pot observar que estudiar-lo no és fàcil. Per tant, eines gràfiques que en permetin la seva visualització poden resultar de gran utilitat per a la comprensió/aprenentatge del mateix.

Posarem un exemple amb el següent Lambda Terme, en la qual hi apareixen tots els parèntesis:

((λx.((x (λy.x)) x)) ((λz.((y z) y)) y))

Veure que dins d'aquest terme hi ha 2 redexs, la seva estructura i el nombre de parèntesis únicament necessaris per a que el Terme sigui el mateix pot resultar complicat si es fa a mà. És per això que mitjançant l'eina podem entrar el Lambda Terme i obtenir els redex restants, la seva representació amb tots els parèntesis o únicament els obligats i com queda representat aquest en forma d'arbre.



Terme amb tots els parèntesis:

((λx.((x (λy.x)) x)) ((λz.((y z) y)) y))

Terme amb els parèntesis obligats:

(λx.x(λy.x) x) ((λz.y z y)y)

Redex restants:

2

1.2 Modificació de seccions de la memòria

S'ha trobat oportú ajuntar en un sol apartat el Marc de Treball i els Estudis i Decisions de la memòria degut al gran nombre d'elements tècnics que apareixen en ella, cosa que farà més comprensible el text.

També s'ha trobat convenient modificar els apartats 8, 9 i 10 de la memòria anomenats "Anàlisi i disseny del sistema", "Implementació i proves" i "Implantació i resultats" per a "Anàlisi, disseny i implementació del sistema" on s'explicarà en paral·lel després de l'anàlisi del sistema el disseny del mateix i els algorismes o elements de cada part més importants (per exemple el Parser) i un altre anomenat "Implantació, proves i resultats" on s'explicaran les proves que es van fer primer al backoffice i posteriorment al frontoffice i llavors es presentaran els resultats. Aquest últim canvi és degut a que els 3 apartats anteriors en l'explicació del TilcW estan força barrejats i la seva separació més adient és la proposada anteriorment.

2. Estudi de viabilitat

Aquest projecte s'ha pogut realitzar a partir dels coneixements adquirits prèviament al llarg de la carrera.

Primerament l'objecte de l'estudi, el Lambda-Càlcul, s'ha treballat a l'assignatura obligatòria de "Paradigmes i Llenguatges de Programació" juntament amb el reforç de conceptes per part del tutor Mateu Villaret, que és qui l'imparteix.

Pel que fa el treball, s'havia realitzat un projecte similar als anys 2007 i 2010, però el que es pretén fer ara és una reescriptura usant el llenguatge de programació anomenat SCALA, que facilitaria la portabilitat a diferents plataformes (no només Windows, com als projectes anteriors) i a la web. Aquest llenguatge s'estudia a l'assignatura opcional "Programació declarativa. Aplicacions" del Grau, dins la branca de 4t curs anomenada "Enginyeria de la Computació", que imparteix el mateix Dr. Villaret.

SCALA, entre altres llenguatges, es pot integrar dins l'eina per crear aplicacions web anomenada "Play 2.0 Framework" que alhora permet exportar-les a aplicació d'escriptori. El Play 2.0 usa un model de treball de patrons de disseny, concretament el patró MVC que l'estudiant ha vist i estudiat dins l'assignatura "Enginyeria del Software 2" i un nivell avançat en coneixements d'HTML5, CSS3 i Javascrip, adquirits durant la seva estada a Dipsalut, dins l'assignatura optativa de 4t de les "Estades a l'Entorn Laboral".

Finalment, com que aquesta eina ha d'estar permanentment activa al web, es necessitava un lloc on allotjar-la, l'elecció usada va ser Heroku. Aquesta plataforma de hosting permet allotjar aplicacions web gratuïtament amb la possibilitat de redirigir-les a un nom de domini propi i no el proporcionat per ells. Per això, es va optar per (el nom que escollim), també gratuït, com a nom web per a l'aplicació.

Per tant, la viabilitat del projecte en quant a coneixements, tecnologia i economia per part del tutor i l'estudiant eren factibles i es va decidir tirar endavant amb el mateix.

3. Metodologia

Per realitzar aquest projecte es va usar la metodologia d'eXtreme Programming (XP). Es va escollir aquesta metodologia perquè ja prediu el fet de que s'incloguin al desenvolupament alguns canvis o nous requeriments a l'aplicació. Per tant, era una bona metodologia perquè ens permetia anar desenvolupant parts del projecte poc a poc, testejar-les i un cop donades per vàlides continuar amb nous requeriments o bé millores dels anteriors.

3.1 Extreme Programming

Aquesta metodologia la va formular Kent Beck l'any 1999, dins el llibre Extreme Programming Explained: Embrace Change. Dins els processos àgils de desenvolupament de software, aquesta metodologia és la més destacada.

Aquest tipus de programació es diferencia de les tradicionals degut a que es dona més prioritat a l'adaptabilitat envers la predicibilitat. És a dir, es considera que els canvis en els requisits de l'aplicació durant el transcurs del seu desenvolupament són naturals, inevitables i fins i tot desitjables.

Aquesta metodologia té 5 principis bàsics:

- **Simplicitat:** és el fonament principal d'aquesta metodologia, ja que si se simplifica el seu disseny al màxim es pot agilitzar el seu desenvolupament i es pot facilitar el seu manteniment.

- **Comunicació:** el codi ha de ser senzill perquè permeti una millor comunicació, aquesta es realitza constantment i és necessària perquè el client (el professor) pugui escollir les característiques i també perquè pugui solucionar els dubtes que puguin aparèixer.

- **Retroalimentació:** es pot saber el que pensa el client en tot moment ja que aquest mateix forma part del projecte, això permet refer el codi que sigui necessari sense haver d'esperar a realitzar un tros molt llarg del projecte, fet que fa perdre molt de temps.

- **Coratge:** la persona que realitza el projecte ha de ser capaç de canviar part del codi quan faci falta, encara que això impliqui més hores i feina. També s'ha d'involucrar al màxim per poder resoldre els problemes que es pugui anar trobant.

- **Respecte:** cada persona ha de valorar la feina de la resta i respectar-la a l'hora de fer la seva. Per tant, han de tenir cura de que el que fan ells no perjudiqui les proves dels companys i han de treballar tan com la resta.

Les característiques principals, les quals posarem un exemple de cada de com s'han treballat en el projecte, de l'eXtreme Programming són:

- Desenvolupament iteratiu i incremental: es va millorant cada cop més, de manera consecutiva.
  En el projecte, cada cop que s'acabava de treballar una part i un cop es començava a testejar, es pensava si sobre aquesta es podia millorar o bé es podia continuar amb noves funcionalitats.
- Proves unitàries contínues: per cada tros de codi implementat, testejar-lo exhaustivament.
  Això és que, un cop acabat un mètode o codi, es testejava amb diferents entrades, de simples a complicades per veure si la sortida era l'esperada.
- Programació en parelles: si es treballa en equip, es pot fer un projecte de més qualitat.
- Treball freqüent del programador amb el client: trobar-se reiteradament amb el client ajuda al programador a fer un millor codi.
  Almenys un cop per setmana, el client i el programador ens anàvem trobant per a veure el codi fet, el programador l'explicava i el client en suggeria millores o el donava per vàlid.
- Correcció dels errors: si es detecta un error en una funcionalitat, es prioritza arreglar-lo a desenvolupar-ne de noves.
  Es van detectar dos errors que podien arribar a ser crítics per la correcta funcionalitat, en aquest cas, es va prioritzar arreglar-los per sobre de continuar desenvolupant l'aplicació.
- Refactorització del codi: és important reescriure parts de codi per poder-ne augmentar-ne la comprensió i un cop fet, repetir les proves unitàries pertinents.
  Un cop acabades funcionalitats prioritàries, s'anava revisant el codi escrit i funcional aplicant reescriptures que permetia l'Scala per fer-lo més entenedor i alhora simple, tan a nivell de comprensió com de computació.
- Propietat compartida del codi: el codi ha de ser accessible a tots els programadors per poder corregir-lo o estendre'l. Això fa que es puguin detectar possibles errors dins el mateix.
- Simplicitat: El més important és que el codi funcioni i no tant que sigui complicat i que no es faci servir mai. D'aquesta manera, s'ha d'anar avançant el projecte i més endavant, si s'escau, augmentar-ne la funcionalitat.
  En un primer moment, s'intentava que el codi es pogués usar, és a dir, que funcionés. Un cop fet i implementades la majoria de funcionalitats, hi havia codis que es feien més funcionals, per exemple, mitjançant immersió.

Aspectes a destacar:

Un cop iniciat el projecte, la seva realització es va anar fent sota la demanda del client (el professor), aquest va anar determinant quines eren les parts més importants del codi que s'havien de desenvolupar abans de passar a fer la part gràfica. Amb això el que s'aconseguia era agilitzar el procés per obtenir ràpidament els algoritmes que servissin realment per la funcionalitat real de l'aplicació, ja que alguns aspectes de Lambda Càlcul no era necessari treballar-los. Tot i així, abans de passar a la part gràfica, es van implementar per a futures expansions del projecte, sota la supervisió i validació del client, i el seu posterior testeig.

De manera excepcional, un cop la part gràfica va ser suficientment funcional, es va decidir contactar amb 4 estudiants que estan cursant l'assignatura de "Paradigmes i Llenguatges de Programació" aquest any per donar-los una primera versió del programa, juntament amb un petit document, on havien d'apuntar deteccions d'errors, deficiències o mancances. Aquesta mesura, acordada per el client i el programador, es va decidir per tal de poder tenir una visió retrospectiva ja que la finalitat del projecte serà l'ús pràctic per part d'estudiants d'aquesta assignatura a l'hora d'estudiar el Lambda-càlcul. Els estudiants van acceptar col·laborar i durant les hores d'estudi van treballar-hi i va servir per trobar errors sobre l'aplicació i en algun algorisme, no correctament implementat, apart de que van fer suggerències útils per a la més ràpida comprensió d'alguns aspectes relacionats amb el Lambda Càlcul que al programador li havien passat per alt o bé havia obviat.

4. Planificació del projecte

Referent a la planificació, el projecte es pot dividir en quatre grans parts: estudi del temari, creació del backoffice i frontoffice i escriptura de la memòria.

Quan ens referim a l'estudi del temari, estem parlant del Lambda Càlcul i de l'Scala. Era interessant que el programador adquirís cert grau d'expertesa d'Scala per a poder fer un projecte d'aquestes característiques ja que a l'assignatura de "Programació Declarativa. Aplicacions" es presenta el llenguatge i els conceptes bàsics, però no s'entra en molta profunditat ni en molts tecnicismes del mateix. Quan parlem del Lambda Càlcul, òbviament, ens referíem a entendre i a veure amb agilitat alguns aspectes del Lambda Càlcul, com serien reconèixer redex, veure si a un terme hi falten tots els parèntesis o posar-ne només els obligats, identificar les variables que ocorren lliures o lligades i aquests conceptes.

Es va decidir centrar-se en la part del desenvolupament del codi intern de l'aplicació en la seva totalitat abans de centrar-se en la part frontal, ja que no tindria sentit preparar una GUI sense saber com seria la base de la mateixa. Així doncs, es va estimar un temps d'entre 4 i 5 mesos, tenint en compte que faltaria també fer la interfície gràfica, que es va estimar entre 2 i 3 mesos de temps.

L'estudi del temari (Lambda Càlcul i el llenguatge Scala) va durar un mes aproximadament, des de mitjans de Setembre fins a mitjans d'Octubre. Si bé és cert que s'han anat introduint conceptes degut a la implementació del frontoffice, a partir d'aquell moment ja es va començar a planificar l'ordre i la preferència dels diferents mètodes del backoffice i a mesura que s'anaven fent, s'anaven testejant unitàriament i es comentaven amb el tutor. El backoffice amb totes les seves funcionalitats bàsiques i avançades (elements que no es fan servir actualment) va acabar-se a finals del mes de Febrer.

Per tant, els temps estimats per a la creació del backoffice s'havien respectat mitjanament. Cal comentar però que durant el desenvolupament del frontoffice s'han afegit al mateix algunes funcions noves, retocat altres per a fer-les realment funcionals per la connexió amb el Play 2.0 i quan hi ha hagut temps, s'han anat polint altres funcions.

La creació del frontoffice mitjançant el Play 2.0 va començar a principi del mes de Març, llegint la documentació del mateix i fent petites proves de funcionament i aplegant els requisits que es creia que seria necessaris, i pensar com es codificarien. Aquest estudi previ i la posterior implementació van ocupar des de l'inici del mes de Març fins a mitjans del mes de Maig, moment el qual es va donar per finalitzada l'eina, la qual ja era funcional. En aquest moment es va passar als estudiants que havien decidit participar i es va començar a redactar la memòria.

El fet de redactar la memòria i acabar de polir o millorar el funcionament de l'eina desenvolupada ha durat aproximadament tres setmanes moment en el qual es dóna per finalitzat completament el projecte i es redacta el treball futur que es pot fer sobre el mateix.

Planificació

Per a una millor comprensió del diagrama de Gantt associat a la planificació del projecte s'ha estat desenvolupant durant 9 mesos aproximadament, es separa el mateix en dues parts (creació del Backoffice per una banda i creació i documentació per una altra) però de totes maneres s'adjunta el diagrama complert al final de l'apartat.

Diagrama per al backoffice:

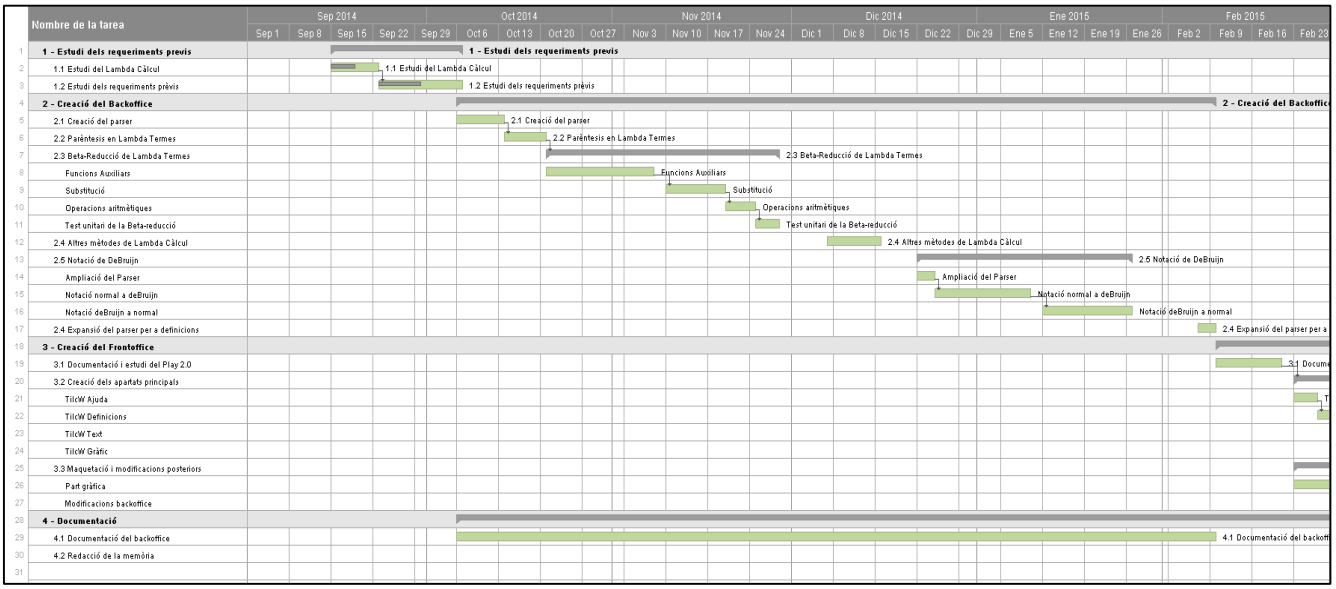
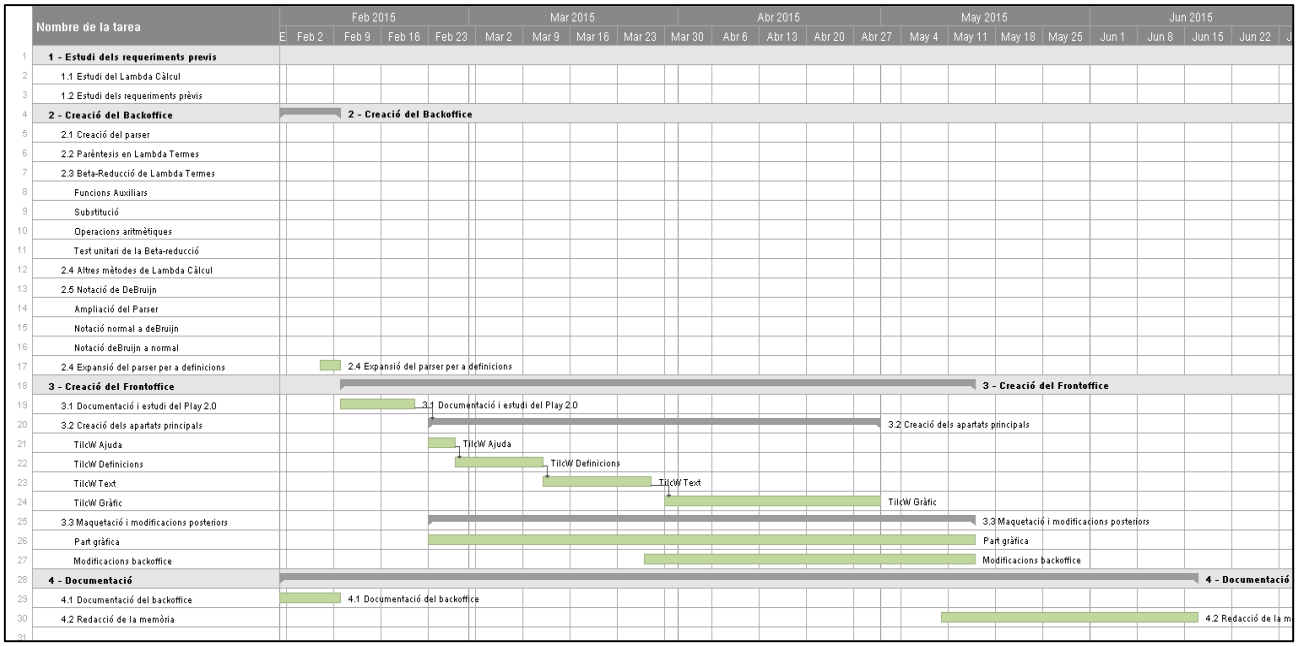


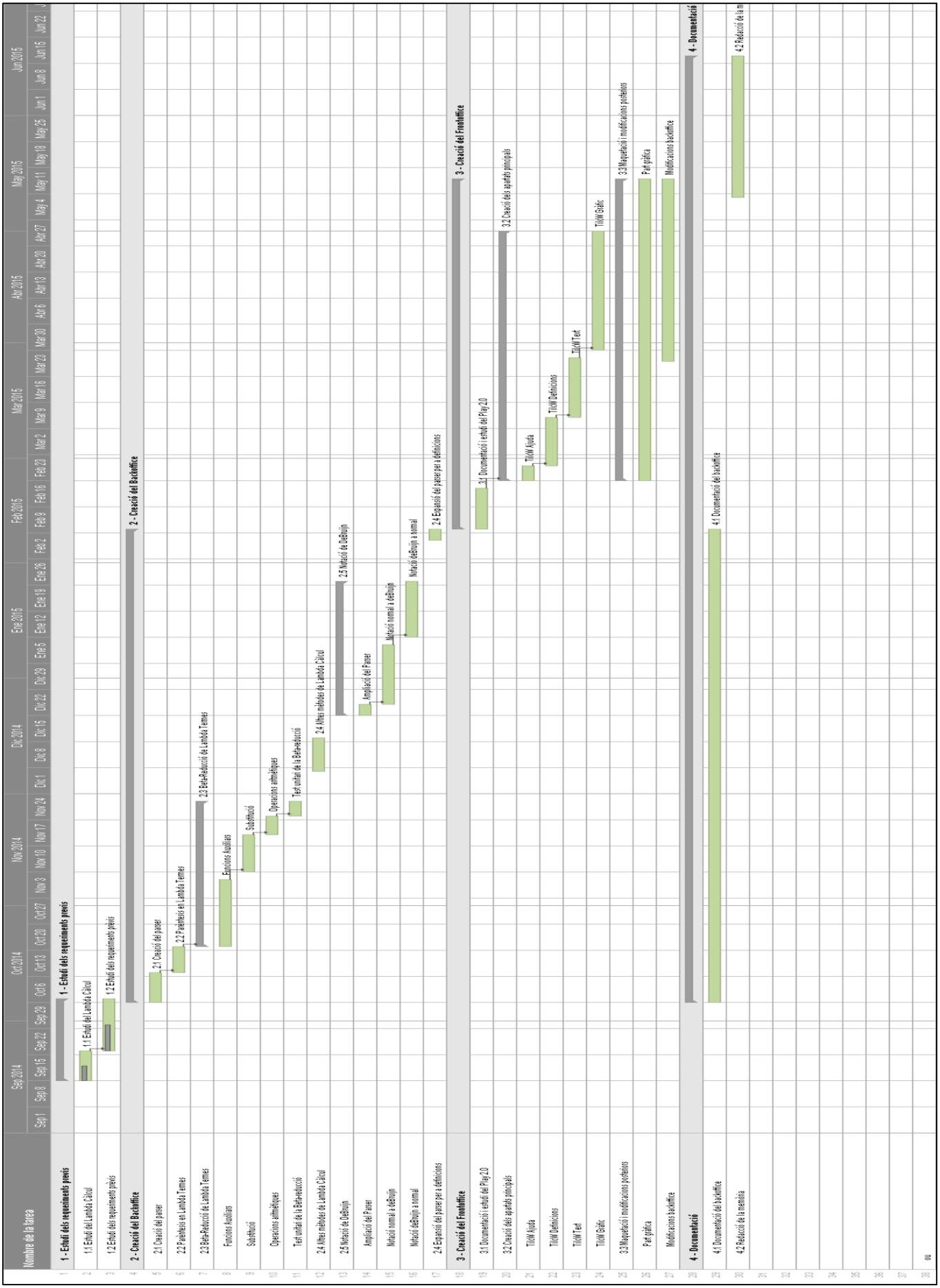
Diagrama frontoffice i documentació:



Temporització

Els temps aproximats per a desenvolupar cada tasca es detallen en la següent taula:

TASCA	TEMPORITZACIÓ
Estudi dels requisits previs	**20h**
• **Lambda Càlcul**	10h
• **Scala**	10h
Backoffice	**75h**
• **Creació del Parser**	**3h**
• **Parèntesis Lambda Termes**	**2h**
• **Beta-Reducció**	**45h**
○ **Funcions auxiliars**	20h
○ **Substitució**	10h
○ **Funcions aritmètiques**	2h
○ **Test**	3h
• **Altres mètodes de Lambda Càlcul**	**8h**
• **Notació de DeBruijn**	**26h**
○ **Ampliació del Parser**	1h
○ **Notació normal a DeBruijn**	10h
○ **Notació DeBruijn a normal**	15h
• **Expansió del parser per a definicions**	**2h**
Frontoffice	**100h**
• **Documentació i estudi de funcionament del Play 2.0**	**15h**
• **Apartats principals**	**75h**
○ **TilcW Ajuda**	5h
○ **TilcW Definicions**	15h
○ **TilcW Text**	20h
○ **TilcW Gràfic**	35h
• **Maquetació i altres modificacions**	**10h**
○ **Maquetació frontoffice**	5h
○ **Modificacions backoffice**	5h
Documentació	**45h**
• **Documentació backoffice**	5h
• **Realització memòria**	40h
TOTAL	**240h**


Nombre de la tarea
Set 2014
Oct 2014
Nov 2014
Des 2014
Ene 2015
Feb 2015
Mar 2015
Abr 2015
May 2015
Jun 2015
1 - Estudi dels requeriments previs
1.1 Estudi del Lambda Càlcul
1.2 Estudi dels requeriments previs
2 - Creació del Backoffice
2.1 Creació del parser
2.2 Parèntesis en Lambda Termes
2.3 Beta-Reducció de Lambda Termes
Funcions Auxiliars
Substitució
Operacions aritmètiques
Test unitari de la Beta-reducció
2.4 Altres mètodes de Lambda Càlcul
2.5 Notació de deBruijn
Ampliació del Parser
Notació normal a deBruijn
Notació deBruijn a normal
2.6 Expansió del parser per a definicions
3 - Creació del Frontoffice
3.1 Documentació i estudi del Play 2.0
3.2 Creació dels apartats principals
TitleW Ajuda
TitleW Definicions
TitleW Test
TitleW Gràfic
3.3 Maquetació i modificacions portaltoss
Part gràfica
Modificacions backoffice
4 - Documentació
4.1 Documentació del backoffice
4.2 Redacció de la memòria

5. Marc de treball i conceptes previs

Degut a la quantitat d'elements i conceptes que intervenen en la realització del projecte, s'ha decidit desglossar i justificar-ne l'ús de manera incremental, és a dir, des del primer concepte sobre el qual s'ha hagut de treballar fins a l'últim que s'ha necessitat per tal de fer el projecte funcional.

5.1 El Lambda-Càlcul

En aquest apartat farem una introducció més profunda del lambda-càlcul que la de l'apartat d'introducció al ser l'objecte d'estudi. Explicarem qui el va formular per primera vegada i els conceptes que es treballaran durant aquest Treball de Fi de Grau, ja que el Lambda-Càlcul conté més conceptes que els que aquí s'explicaran.

Teoria i formulació del Lambda-Càlcul

En lògica matemàtica i computació, el Lambda-Càlcul és un sistema formal dissenyat per investigar la definició, l'aplicació de funcions i la recursivitat. El van introduir per primera vegada Alonzo Church i Stephen Kleene durant la dècada del 1930.

Alonzo Church, al 1936, va usar el Lambda-Càlcul per a resoldre l'*Entscheidungsproblem* (problema de decisió) donant-ne una resposta negativa. El lambda-càlcul es pot usar per a definir de manera precisa i neta del què és una "funció computable". Saber si dos lambda-termes són o no són equivalents no es pot resoldre mitjançant un algorisme, fet pel qual fins i tot abans del problema de la parada de les Màquines de Turing (halting problem) va quedar provada la indecibilitat.

Es pot considerar el lambda-càlcul com el llenguatge de programació universal més petit del món, ja que aquest consisteix en única regla de transformació (substitució de variables mitjançant la Beta-Reducció) i un esquema simple per definir funcions, compost d'un únic patró de substitució. El Lambda-Càlcul ha tingut doncs una gran influència sobre els llenguatges de programació funcional, com ara el LISP, ML o bé el Haskell i ara últimament en llenguatges de caire imperatiu com el C++,el Go o l'Scala.

El lambda-càlcul se l'anomena universal perquè qualsevol funció pot ser computada o avaluada a través d'ell. Per això és equivalent a les màquines de Turing. De totes maneres, el lambda-càlcul no posa tant d'èmfasi en l'ús de les regles de transformació i no considera a les màquines reals capaces d'implementar-lo. Així doncs, estem parlant d'una proposta que s'assembla més al software que al hardware.

La gramàtica del lambda-càlcul és molt simple, ja que compta només de 3 regles molt senzilles, que seran usades per a construir el que anomenarem **Lambda-Expressions**.

- **Variables:** x,y,z,...
- **Aplicacions:** suposem que tenim dos lambda-expressions E i E1 qualsevols. Llavors (E E1) també serà una lambda-expressió. Això significarà que a l'expressió E1 l'hi estem aplicant E.
- **Abstracció:** donada una variable V i una expressió E qualsevols, llavors (λ V.E) és una abstracció.

Notació

Podem ometre parèntesis a l'hora descriure lambda-expressions, però haurem de tenir en compte alguns punts per permetre que el lambda-terme continuï sent el mateix.

1. L'aplicació de les funcions és associativa a l'esquerra. Exemples:

   E E1 E2 significa ((E E1) E1)
   E E1 E2 E3 significa (((E E1) E2) E3)

2. L'àmbit d'una λ V arribarà tant a l'esquerra com sigui possible. Exemples:

   λ V. E E1 significa (λ V. (E E1))
   λ V. E E1 E2 significa (λ V. ((E E1) E2))

3. L'escriptura de λ seguit de variables permetrà ometre λ, però aquestes hi són:
   λ V V1 ... Vn. E significa (λ V.(λ V1.(...(λ Vn.E)))) o bé λ V,V1 ..., Vn. E també significa igualment (λ V1.(λ V2.(...(λ Vn.E)))). Exemples:
   λ x y. E significa (λ x.(λ y. E))

   λ x,y. E significa (λ x.(λ y. E))

Variables lliures i variables lligades

Una variable pot aparèixer amb 2 rols diferents dins una lambda-expressió:

- **Variable lliure:** variable que no apareix dins l'àmbit d'una abstracció λ V.
- **Variable lligada:** variable que si apareix dins l'àmbit d'una lambda-expressió λ V.

Posem-ne un exemple:

(λ x λ y. z x y) (x y)

Veiem que:

- Les variables x y que segueixen la variable z són variables lligades, ja que estan dins l'àmbit de l'abstracció λ x λ y respectiament
- Les variables x i y del terme (x y), en canvi, són lliures, ja que les abstraccions λ x λ y. venen restringides pels parèntesis i per tant ni x ni y estan dins els seu àmbit. També ho és la variable z de la primera expressió, ja que no hi ha cap abstracció de z en la lambda-expressió.

Si ens mirem amb més detall les expressions λ V. E, podem veure que:

λ V representa el paràmetre formal
E representa el cos de la definició

Exemple:

λ x λ y. (z x)

Veiem que λ x y. és el paràmetre de la lambda-expressió i (z x) correspon al cos de la mateixa.

Quan una funció s'aplica a un argument, el que fem és substituir tots els paràmetres formals que contingui el cos de la funció per l'argument:

(λ x λ y. (z x)) (3) és igual a λ y. (z 3), ja que hem canviat les x per (3)

Substitució

Un pas bàsic del lambda-càlcul és la substitució, és a dir, canviar variables per altres lambda-expressions (ja siguin altres variables, aplicacions o abstraccions). Aquesta operació s'ha de realitzar amb certa cura ja que potser capturem variables. Suposem la següent lambda-expressió:

((λ x λ y. x) y)

En aquest cas, hauríem d'aplicar el lambda terme en vermell (λ x λ y. x) al lambda terme colorejat de color verd y. Aquest lambda-terme, que és l'argument de la funció anterior, és la variable y que com es veu, apareix lliure ja que l'àmbit de l'abstracció y queda restringida pels parèntesis abans d'ocórrer y.

Si fem el pas de substitució sense tenir en compte captures de variables ens quedarà el següent:

λ y. y

Ara aquesta variable y, fent la substitució, ha passat de lliure a lligada per λ y.

U una possible solució seria renombrar la variable que lliga i totes les lligades per ella per un altre que al fer la substitució no lliguin la que està lliure.

De totes maneres la definició de la substitució evitant captura de variables es faria seguint aquests passos:

Forma de M		resultat de $M[v \mapsto M']$
v		M'
v'	amb $v \neq v'$	v'
$M_1\ M_2$		$M_1[v \mapsto M']\ M_2[v \mapsto M']$
$\lambda v.\, M_1$		$\lambda v.\, M_1$
$\lambda v'.\, M_1$	on $v \neq v'$ i $v' \notin FV(M')$	$\lambda v'.\, M_1[v \mapsto M']$
$\lambda v'.\, M_1$	on $v \neq v'$ i $v' \in FV(M')$	$\lambda v''.\, M_1[v' \mapsto v''][v \mapsto M']$ on $v'' \notin FV(M')$ ni $v'' \notin FV(M_1)$

Beta-Conversió / Beta-Reducció

Qualsevol aplicació de la forma (λ V. M) N es pot **beta-reduir**, i **totes** les ocurrències de V que apareguin lliures a M han de ser substituïdes per la lambda-expressió N.

Per exemple:

((λ x. x x x) a) és beta redueix a (a a a)

((λ x λ y. z x y) (a b)) és beta redueix a (λ y. z (a b) y)

Per norma, sempre es comença a fer la substitució per el paràmetre (λ V) més extern, és a dir, el que es troba a l'esquerra de tot dins l'abstracció que apareix a l'aplicació.

Beta-Normalització

Usarem les paraules normalització, redex i reducció per referir-nos a les paraules beta-normalització, beta-redex i beta-reducció per fer més entenedor el text a partir d'aquí.

Definirem un redex com al subterme sobre el qual es pot aplicar una reducció. Si volem definir-ho però d'una manera més simple i entenedora, un redex és una aplicació d'una abstracció i una lambda-expressió qualsevol.

(λ V. E)(E1)

Parlarem de que un terme està normalitzat (direm que es troba en forma normal) si ja no queda cap redex dins el terme. Per exemple:

(((λ x. λ y. z x y) a) b)

A l'exemple anterior tenim un redex que podrem beta-reduir:

(((λ x. λ y. z x y) a) b) que es transforma en ((λ y. z a y) b)

En aquest nou terme, ens ha aparegut un redex nou per a reduir:

((λ y. z a y) b) que es transforma en (z a b)

Com que ara ja no ens queda cap redex per reduir, podem dir que (z a b) està en forma normal. Podem dir també amb seguretat que un terme està en forma normal si no apareix en ell una abstracció. Cal tenir en compte però que tot i que quedin abstraccions dins el terme, aquest pot estar en forma normal, és per això de la importància dels parèntesis.

Per exemple:

a (λ b. b y) b ja està en forma normal ja que si col·loquem tots els parèntesis, ens adonarem efectivament que no hi ha cap redex: ((a (λ b. b y)) b)

Ordres de reducció: Normal i Aplicatiu

A l'hora de començar a aplicar beta-reduccions existeixen dos possibles maneres de tractar el terme, que són els anomenat ordres de reducció: l'ordre de reducció normal i l'ordre de reducció aplicatiu. L'única diferència que hi ha entre els dos és l'ordre en què es van aplicant les beta-reduccions dels redex del lambda-terme.

L'ordre de reducció normal redueix sempre el redex més extern dins el lambda-terme. Per contra, l'ordre de reducció normal, el que fa és reduir el redex més intern dins el lambda-terme. Si es dóna el cas que hi hagi dos redex externs o interns al mateix nivell de profunditat dins el terme, sempre es reduirà el redex de més a la l'esquerra.

Tot seguit presentarem un exemple d'un lambda-terme amb dos possibles redexs:

((λ x. x) ((λ x. x) y)) veiem que tenim els redexs: (λ x. x) ((λ x. x) y)

(λ x. x) ((λ x. x) y)

- Per ordre de reducció normal el redex que primer es reduiria seria:

  ((λ x. x) ((λ x. x) y)) ja que és el redex més extern

- Per ordre de reducció aplicatiu el redex que primer es reduiria seria:

  ((λ x. x) ((λ x. x) y)) ja que és el redex més intern

Cal tenir en compte que si un terme té forma normal, si es prova de reduir mitjançant l'ordre de reducció normal s'arribarà a aquesta, en canvi si es prova de reduir mitjançant l'ordre aplicatiu, no podem garantir-ho.

Posarem un exemple que ho demostra:

Lambda-terme proposat: (λ x. y)((λ x. x x x)(λ x. x x x))

Reducció mitjançant Ordre Normal:

1r pas: (λ x. y)((λ x. x x x)(λ x. x x x))

Resultat: y

Reducció mitjançant Ordre Aplicatiu:

1r pas: (λ x. y)((λ x. x x x)(λ x. x x x))

Resultat: (λ x. y)((λ x. x x x) (λ x. x x x) (λ x. x x x))

2n pas: (λ x. y)((λ x. x x x) (λ x. x x x) (λ x. x x x))

Resultat: (λ x. y)(((λ x. x x x)(λ x. x x x) (λ x. x x x)) (λ x. x x x))

3r pas: .....

Com podem veure, si anem reduint per ordre de reducció aplicatiu no arribarem mai a trobar la forma normal del terme perquè aquest anirà creixent per la dreta cada vegada més, mentre que aplicant l'ordre de reducció normal amb un sol pas trobarem la forma normal.

Alpha-Conversió

L'operació de l'alpha-conversió serveix per canviar el nom de les variables lligades dins un lambda-terme. La funció identitat, per exemple:

λ x. x

- La podrem alpha-convertir a λ y. y si renombrem la "x" per "y". El terme doncs serà: λ y. y

Tot i que, com hem dit abans, que l'equivalència entre dos termes és indecidible, es considerarà que si difereixen entre ells només per una o vàries alpha-conversions, seran equivalents. En el cas anterior (λ x. x = λ y. y), i qualsevol terme resultant d'una alpha-conversió, també es pot considerar equivalent del lambda-terme anterior sempre i quan es respectin les dues normes següents.

1. En primer lloc, en l'alpha-conversió s'ha de tenir en compte al convertir una abstracció de la forma (λ V.E) que només hem de canviar les ocurrències de les variables lligades per aquesta abstracció. Mirem el següent exemple:

Si alpha-convertim	λ x. λ x. x	renombrant x per una y,
El resultat correcte és	λ y. λ x. x	
Un resultat incorrecte seria:	λ y. λ x. y	ja que hauríem renombrat una variable no es trobava lligada per l'abstracció λ x.

2. En segon lloc, no podem fer una alpha-conversió per una variable qualsevol si el fet de fer-ho provoca la captura d'altres ocurrències que apareixien lliures. Observem-ho:

Si alpha-convertim	λ x. λ y. x	renombrat x per y,
Obtindrem:	λ x. λ y. y	(resultat incorrecte)

A l'alpha-convertir la variable 'x' per una 'y' el que hem fet és que la abstracció λ y. que abans no lligava cap variable ara lligui a una variable que de fet estava lligada per l'abstracció λ x., per tant hem creat un lambda-terme diferent a l'inicial. El que podem fer és alpha-convertir aquest terme amb un altre nom de variable que no quedi lligada per una altra. Per exemple:

Si alpha-convertim	λ x. λ y. x	renombrat x per z,
Obtindrem:	λ z. λ y. z	(resultat correcte)

Ara ja no capturem la variable, per tant el terme λ z. λ y. z si que és equivalent a λ x. λ y. x després de l'alpha-conversió de la variable x.

Eta-Conversió

Existeix una tercera regla de transformació, la eta-conversió, que ens diu que tota lambda-expressió de la forma λ V. (E V) pot convertir-se en E si i només si dins d'E no apareix lliure la variable V. De tota manera, aquesta regla no la tractarem ja que tot i que s'explica a l'assignatura, no es treballa ni s'estudia dirant la mateixa.

Notació DeBruijn

Existeix una notació per al Lambda-Càlcul, que es va inventar el matemàtic holandès Nicolaas Govert de Bruijn (d'aquí el nom de la notació) que permet escriure els noms de les variables que intervenen en un lambda-terme com a nombres naturals. D'aquesta manera, s'evita el fet del problema de capturar variables dins un lambda-terme al fer la redefinició de les mateixes.

La manera d'escollir aquests nombres segueix la següent mecànica: per cada variable, es compten les abstraccions intermèdies abans d'arribar a la que la lliga dins el lambda-terme. Per les variables lliures, s'escull com a valor el nombre d'abstraccions des de la part més interna del terme i se li suma 1. Per cada variable lliure nova que aparegui, el seu valor serà el nombre correlatiu següent a l'última variable lliure, així no es repeteixen ja que si dues variables lliures tenen el mateix valor, estarem representant la mateixa variable. Això no passa amb les lligades, ja que una variable lligada pot tenir abstraccions intermèdies, i per tant, pot tenir diferents valors però representar la mateixa en notació normal.

Diferents exemples que il·lustren la notació de deBruijn amb totes les variables lligades:

El terme λx. λy. x, en deBruijn s'escriurà de la forma λ λ 2 .

El terme λx. λy. λz. x z (y z), en deBruijn s'escriurà de la forma λ λ λ 3 1 (2 1) .

El terme λz. (λy. y (λx. x)) (λx. z x) en deBruijn s'escriurà de la forma λ (λ 1 (λ 1)) (λ 2 1)

Posem ara un exemple amb variables lliures:

El terme (λx. λy. z x (λu. u x)) (λx. w x), en deBruijn s'escriurà de la forma (λ λ 4 2 (λ 1 3)) (λ 5 1)

Lambda-Càlcul com a llenguatge de programació

A l'assignatura "Paradigmes i Llenguatges de Programació" el lambda-càlcul està vist com un llenguatge de programació, mostrant diferents lambda expressions de diferent complexitat com a entrada d'un programa en la qual la sortida és la seva forma normal.

És a dir, la finalitat d'aquests "programes" serà la forma normal de la lambda-expressio entrada o bé es penjarà.

S'ensenya, a més que els elements bàsics de la programació, com podrien ser enters, booleans, condicionals, tuples o fins i tot la recursivitat es poden escriure com a lambda termes i que el seu comportament és l'esperat.

Fins aquí acaba l'explicació detallada de l'objecte d'estudi. A partir d'aquí es presentaran els conceptes per a poder fer l'aplicació web tal com s'havia plantejat.

5.2 SCALA i ScalaDOC

Primerament, descriurem les característiques principals de l'Scala, el llenguatge de programació que s'ha usat per crear tot el background de l'aplicació i una part de la part visible de cara a l'usuari. Tot seguit, descriurem també l'ScalaDOC, el mètode emprat per fer la documentació de les classes del projecte.

Scala

Scala és un llenguatge de programació modern multi-paradigma dissenyat per a expressar patrons de programació generals d'una manera concisa, elegant i segura respecte als tipus. Integra característiques de orientació a objectes i de llenguatges funcionals.

Aquest llenguatge és purament orientat a objectes, ja que cada valor és un objecte. Els tipus i comportaments d'aquests objectes estan descrits mitjançant classes o "traits". L'abstracció d'una classe és possible mitjançant subclasses i un mecanisme basant en classes mixtes, semblants als interface de java però amb la definició de tipus, variables i implementació de mètodes, anomenades "traïts", que són la substitució neta de l'herència múltiple. També és un llenguatge funcional ja que cada funció és un valor. Scala dóna una sintaxi simple per a definir funcions amb lambda-càlcul i també suporta funcions d'ordre superior, permet passar funcions com a paràmetre d'altres i suporta el *currying*. Té classes per cas i inclou suport per al reconeixement de patrons. Aquests dos elements serveixen per a modelar tipus algebraics, que s'usen en molts llenguatges de programació funcional.

A més a més, la noció de reconeixement de patrons estén de manera natural el procés de dades XML amb l'ajuda de seqüències de patrons ignorant el que hi ha a la dreta. En aquest context, les comprensions de seqüències (iteració sobre Generadors-Filtres-Mapeig) són útils per a formular consultes. Aquestes característiques fan de Scala l'ideal per a desenvolupar aplicacions com a serveis web.

Scala està equipat amb un sistema de tipus expressiu que assegura estàticament que les abstraccions s'utilitzin de manera coherent i segura. En particular el sistema suporta un mecanisme d'inferència de tipus que permet que l'usuari no hagi d'anotar els tipus de manera redundant. Combinades, aquestes característiques proporcionen una base potent per a la reutilització segura de les abstraccions de programació i també per a l'extensió de programari amb tipus de dades segurs.

El disseny de Scala reconeix el fet que, a la pràctica, el desenvolupament d'aplicacions d'un domini específic requereix extensions de llenguatge igualment específiques. Scala proporciona una combinació única de mecanismes de llenguatge que fan fàcil afegir noves construccions al llenguatge, amb suavitat, en forma de biblioteques.

Scala està dissenyat per a interactuar bé amb entorns populars de programació com Java (JRE) i el marc .NET (CLR). En particular, la interacció amb llenguatges orientats a objectes de gran acceptació com Java i C# és força suau. Scala té el mateix model de compilació (compilació separada, càrrega dinàmica de classes) que Java i C#, permetent accedir a milers de biblioteques d'alta qualitat. Degut a ésser un llenguatge dissenyat inicialment per a la màquina virtual JVM, hereta del Java les característiques del lèxic i els tipus bàsics i les seves

operacions així com una sintaxi molt similar. Els punt-i-coma a final de línia es poden ometre (des de la versió 2).

El que en altres llenguatges funcionals es descriu com a unió de tipus discriminats per etiquetes anomenades constructors, aquí es parteix de la classe que identifica el tipus i se'n deriven classes per cada cas de la unió discriminada, amb el constructor per nom i paràmetres corresponents als components del cas.

Variables i mètodes

El tipus de les declaracions es pot ometre si es dedueix de la seva inicialització de manera inequívoca, altrament el compilador ens avisarà que no pot inferir el tipus.

```
val identificador :tipus = expressió     // amb val: variable assignable un sol cop

var identificador :tipus = expressió     // amb var: variable assignable més d'un cop

def identificador :tipus = expressió     // amb def: es recalcula a cada invocació
```

ScalaDOC

ScalaDOC, molt similar al JavaDOC, permet documentar el codi creat en Scala mitjançant l'addició de comentaris als mètodes i variables per tal d'explicar a l'usuari què fa cada mètode o variable, què s'espera a l'entrada i què retornarà a la sortida si l'entrada és correcta. També permet afegir una petita descripció del codi.

Un cop feta aquesta documentació es permet exportar de manera que l'usuari pot visualitzar-la com si es tractés d'un nou package de la llibrera API d'Scala (per posar un exemple, un package seria l'importació d'un Hashmap amb tots els constructors i mètodes que permeten treballar sobre el mateix) per tal de tenir el codi ben documentat, o per si s'ha d'afegir en alguna contribució, seguir els estàndards de documentació. Els creadors d'Scala proveeixen a la seva web una guia d'estil per a documentar correctament el codi, que és la que s'ha seguit per a documentar les classes en les quals s'ha pogut fer d'aquest projecte.

5.3 El Play 2.0 i l'SBT

Play 2.0

Play 2.0 és una eina de codi lliure per a crear aplicacions web en Java o Scala. Va ser creada l'any 2008 per al desenvolupador de codi Guillaume Bort, que va publicar-ne la primera versió estable l'octubre de l'any 2009 i va donar suport a Scala a partir de la versió 1.1. Actualment la versió que s'usa és la 2.3.8, que va ser publicada el Febrer de l'any 2015.

Inicialment el nucli del Play estava escrit amb Java i les plantilles es creaven amb Groovy, però a partir de la versió 2.0 el nucli es va reescriure amb Scala i es va migrar a SBT la construcció i el desplegament de les aplicacions i les plantilles van passar a fer-se amb Scala.

Apart de poder tenir l'aplicació que es desenvolupi on-line, el Play 2.0 permet crear l'aplicació *standalone* o d'escriptori tant per a sistemes Windows com a Linux per a poder-la usar fora d'internet mitjançant una comanda de l'SBT. Aquest fet va motivar a usar el Play 2.0 per al projecte ja que permetria treballar-hi en diferents plataformes i tant en línia com una aplicació que no depengui d'internet per a funcionar.

Tot i que no les enumerarem totes, aquí podem una llista de característiques que suporta Play 2.0 moltes de les quals s'han usat o s'han necessitat per al desenvolupament de l'aplicació:

- El servidor que fa servir és JBoss Netty
- Inclou bases de dades en el propi core per si s'han de fer servir.
- Les plantilles es basen en motors programates en SCALA.
- Permet recarregar l'aplicació al moment mentre es desenvolupa.
- Té parser de JSON i XML.
- Per a dependències, usa SBT.

Algunes plataformes que usen el Play 2.0 Framework per a les seves aplicacions web serien LinkedIn i coursera.

SBT

SBT (Scala Build Tools) és una eina de software lliure creada per al desenvolupador Mark Harrah per a la construcció de projectes amb SCALA i Java, similar a Maven o Ant. El que permeten aquestes eines al programador, entre altres coses, és afegir a un fitxer de configuració possibles dependències i aquests s'encarreguen d'importar-les, com és el cas del Tilc.

5.4 HTML5 i CSS3

Play 2.0, com a aplicació web, usa HTML per a les vistes, i concretament permet usar HTML5, l'última versió d'aquest llenguatge.

Per a l'apartat d'estil, s'ha usat el CSS3, l'última versió d'aquest tipus de maquetació, per a la part d'estilitzar la presentació.

Aquests dos elements, que junts permeten una bona presentació i maquetació del Play 2.0 estan regulats per el w3schools, una web de desenvolupadors d'arreu del món que conté tutorials i exemples d'us per al elements i mètodes disponibles de html, css, javascript i jquery

5.5 Llibreries Javascript i Heroku

En aquest apartat es farà una petita introducció al Javascript, però es farà un incís especial en les llibreries i/o biblioteques que s'han usat i que s'han escrit amb el mateix llenguatge i que permeten a les vistes comunicar-se amb el controlador o per mantenir els elements que apareixen per pantalla desats en variables o bé usar funcions auxiliars per al bon ús de l'aplicació.

Javascript

Javascript és un llenguatge script basat en el concepte del prototipus, és a dir, herència per delegació. Va ser implementat per Netscape Communications i s'usa molt en pàgines web, però també té utilitat en altres aplicacions. Tot i que podria semblar-ho pel seu nom, no és cap derivació del Java, però la seva sintaxi és similar i està inspirada en el llenguatge C. "Javascript" és una marca registrada per Oracle Corporation.

JQuery

JQuery és una biblioteca Javascript que va crear John Resig i que millora l'interacció entre documents HTML i l'arbre DOM amb la gestió d'esdeveniments i afegint també la interacció amb la tecnologia Ajax. L'aventatge d'us d'aquesta llibreria és que es permet el codi fent amb poques línies de text el que amb funcions de Javascript pur en serien força més.

AJAX

AJAX són les sigles de (Javascript Asíncron i XML, en anglès), una tecnologia que permet actualitzar el contingut d'una pàgina web sense haver de recarregar la pàgina sencera. Aquest sistema permet demanar i rebre dades en un segon pla, sense que l'usuari noti cap diferència de comportament en la pàgina que està visitant. La majoria d'aquestes crides es fan mitjançant Javascript.

Podem intuir doncs que l'ús d'AJAX és essencial per a l'aplicació, ja que a excepció del menú de navegació, el d'opcions i les pàgines de definicions i d'ajuda, la resta de l'aplicació ha de ser interactiva i amb contingut dinàmic amb una imperceptibilitat en el temps quan aquest canvii, per fer àgil l'aplicació.

D3

La llibreria D3 (sigles de Data-Driven Documents) és una llibreria JavaScript sota les directrius de la llicència BSD la qual permet crear visualitzacions dinàmiques de dades al navegadors web. Està desenvolupada per un grup de desenvolupadors que van iniciar-la al febrer de l'any 2008, i actualment ja va per la versió 3.5.5.

Aquesta llibreria està fortament lligada amb SVG, l'HTML5 i el CSS3 per a la representació de les dades. Permet representar-les de moltes formes (arbres, grafs, diagrames de tot tipus,etc) i dóna molta flexibilitat en les sortides que doni ja que permet aplicat classes CSS als elements generats. La majoria d'aquestes dades s'entren o bé amb JSON, CSV o geoJSON tot i que si és necessari, poden usar-se funcions auxiliars perquè una entrada en altres formats sigui acceptada.

S'ha usat aquesta llibreria per a generar l'arbre binari que representa el Lambda Terme que l'usuari entri i es representa dins un canvas on l'usuari pot fer zoom i moure l'arbre, entre altres.

Apprise

Aquesta llibreria permet mostrar els típics alert de Javvascript però d'una manera més agradable. De fet, no és que permeti mostrar-los d'una manera més agradable, si no que són elements propis seus, on també es disposa d'un fitxer CSS per a personalitzar-ne la sortida. Com que en el nostre cas no ha calgut, s'ha deixat amb les opcions que duu per defecte, però es pot canviar la lletra, el color fons, el número de botons, i tota una sèrie d'elements que per a l'ús de l'aplicació no s'ha trovat convenient canviar-los.

JSON

En el cas d'aquest projecte, les dades passades per a la creació de l'arbre que representa el LTerme es fan amb JSON, així que s'explicarà una mica què és de manera que es pugui entendre com es crea l'arbre.

JSON són les sigles de JavaScript Object Notation, i aquest és un estàndard per a passar objectes de dades consistents en parelles atribut-valor de manera que sigui comprensible per un humà. Tot i que deriva del JavaScript, és un format de dades independent, per la qual el codi per a generar dades JSON o bé parsejar-les està ja a disposició de molt llenguatges. En el cas del Tilc, el controlador parseja el LTerme en format string que rep del model i alhora envia a la vista l'string corresponent a l'arbre en format string, on a la vista el Javascript s'encarrega d'obtenir de nou l'arbre en format JSON, per tal que la llibreria D3 pgui representar-lo.

Heroku

Heroku és una plataforma de servei de computació al núvol. Això vol dir que Heroku ofereix de manera gratuïta als seus clients tenir a internet aplicacions. Heroku està en un sistema operatiu Ubuntu i opera des del Juny del 2007. Al principi només suportava aplicacions desenvolupades en Ruby, però més endavant es va extendre el suport a altres com ara Java, Node.js o Scala, i és per això que s'ha decidit usar-la per a tenir l'aplicació disponible en línia. Es pot obtenir l'aplicació llesta per a penjar a Heroku mitjançant la comanda de l'SBT *stage deployHeroku*.

Amb tot l'explicat anteriorment, s'han descrit ja tots els elements que es necessiten per a què es puguin entendre els elements que s'explicaran més endavant referents als requisits i al disseny i esquema de l'aplicació.

5.6 Col·laboracions externes

El tutor Mateu Villaret, qui m'ha ensenyat Lambda Càlcul, l'objecte d'estudi i SCALA, el llenguatge de programació amb què s'ha escrit majoritàriament l'aplicació . El tutor, alhora ha estat el col·laborador que ha fet de client seguint la metodologia escollida per a desenvolupar el projecte, l'eXtreme Programming, corregint elements i suggerint-ne d'altres a mesura que avançava el projecte.

Els companys de feina de Dipsalut Astrid, David i Miquel els quals m'han ajudat a entendre i a programar correctament amb HTML,CSS3, JQuery i Ajax, requisits indispensables per al bon funcionament de l'aplicació.

Els companys del LAP, Marc i Joan, que em van proporcionar informació i aplicacions fetes per ells del Play 2.0, exemples en Javascript i em van aconsellar l'ús de la llibreria Javascript D3 per a la generació del gràfic, així com documentació.

Finalment, la col·laboració dels alumnes que actualment estan cursant Paradigmes i Llenguatges de Programació Nuri Banús, Lourdes Boix i Rubén Calle els quals m'han ajudat a detectar errors potencials, a afegir noves funcionalitats i en la maquetació del producte final que es presenta quan han hagut d'estudiar l'apartat de Lambda Càlcul per a l'assignatura.

6. Requisits del sistema

Per a poder usar el TilcW l'usuari necessita un ordinador o bé un smartphone o tauleta. Per a aquests dos últims es requereix una connexió a internet. És possible treballar amb el TilcW mitjançant aquests dispositius ja que s'han adaptat les plantilles del TilcW a fins a unes resolucions de 800x600 i la majoria dels dispositius mòbils arriben o superen aquesta resolució.

Si parlem de la versió sense internet, no es pot usar als dispositius mòbils ja que és un script que necessita de la terminal i de llegir uns propis fitxers descomprimits i per tant no és possible executar-ho. Per a ordinadors, es pot usar tant en ordinadors de sobretaula com en ordinadors portàtils tant en plataformes Windows, Linux o MacOS.

L'únic requisit indispensable és un navegador web. S'han provat els següents i el TilcW ha funcionat correctament:

- Chromium
- Icewheasel
- Google Chrome
- Mozilla Firefox
- Safari
- Internet Explorer 10 i superiors

Si qualsevol d'aquests navegadors està instal·lat, el TilcW es podrà executar al navegador. No requereix cap llibreria Java addicional o algun altre programari, ja que ve tot integrant en el seu core.

La direcció web per executar-lo, si es té internet és:

http://tilcw.herokuapp.com

I si es vol fer servir la versió de sobretaula, sense internet, un cop executat l'script (bat o sh), és:

http://localhost:9000

7. Anàlisi, disseny i implementació del sistema

Un cop es va començar a desenvolupar la part del backoffice, o més ben dit, la part no visible del TilcW, va sortir també el problema de com desenvolupar la part gràfica de l'aplicació.

En un primer moment, es va pensar en desenvolupar per separar el que seria l'aplicació d'escriptori del de l'aplicació web, amb el cost temporal que això comportaria. Primerament, es van començar a buscar eines per a la versió d'escriptori amb diferents eines de creació d'aplicacions amb Scala, on es van veure eines com ScalaFX (Que té el seu homòleg de Java, més conegut potser, que és el JavaFX) o bé plugins de les eines de programació que s'usava per a crear applets amb Scala, però no es va trobar molta informació.

Després de parlar amb el client, es va decidir aparcar la cerca temporalment, centrar-se en acabar tot el backoffice o les parts que faltaven per a desenvolupar i centrar-se en poder fer una aplicació web. Després de fer una cerca i veure varies aplicacions que permetrien desenvolupar el TilcW com a aplicació web es va trobar l'eina Play 2.0 Framework. Es va decidir no seguir buscant més per a desenvolupar-lo com a aplicació d'escriptori ja que el Play permetria crear una aplicació tant web com d'escriptori un cop aquesta estigues acabada. A més Heroku, una eina de serveis al núvol, tenia suport per a aplicacions per al Play 2.0 i oferia els mateixos de manera gratuïta.

Tot i que més endavant s'explicarà amb molt detall com s'integra el TilcW al Play, dir que aquest es basa en un patró de disseny anomenat Model-Vista-Controlador, que es considera oportú fer-ne una introducció en aquest punt.

El Patró MVC

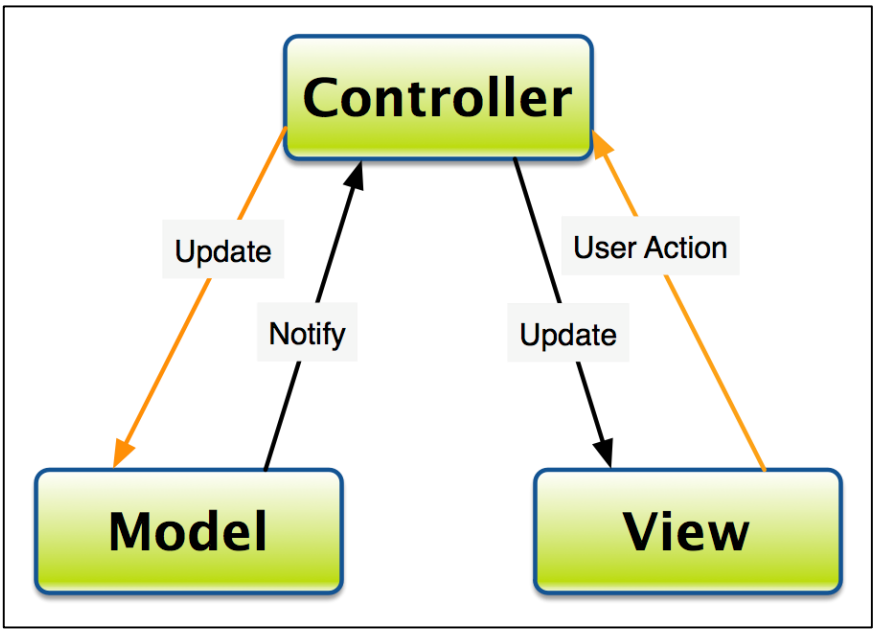
Play 2.0 usa per definició l'estructura del Patró MVC (Model-Vista-Controlador) per a crear les aplicacions web, per això s'ha trobat coherent fer-ne una petita introducció abans d'explicar amb més detall el Play 2.0.

Definició del Patró

Aquest patró normalment se'l coneix com a patró de patrons degut que és un patró que n'engloba 3 en un mateix disseny. Aquests són l'Strategy (implementat pel controlador), l'Observer (implementat pel model) i finalment el Composite (implementat per la vista).

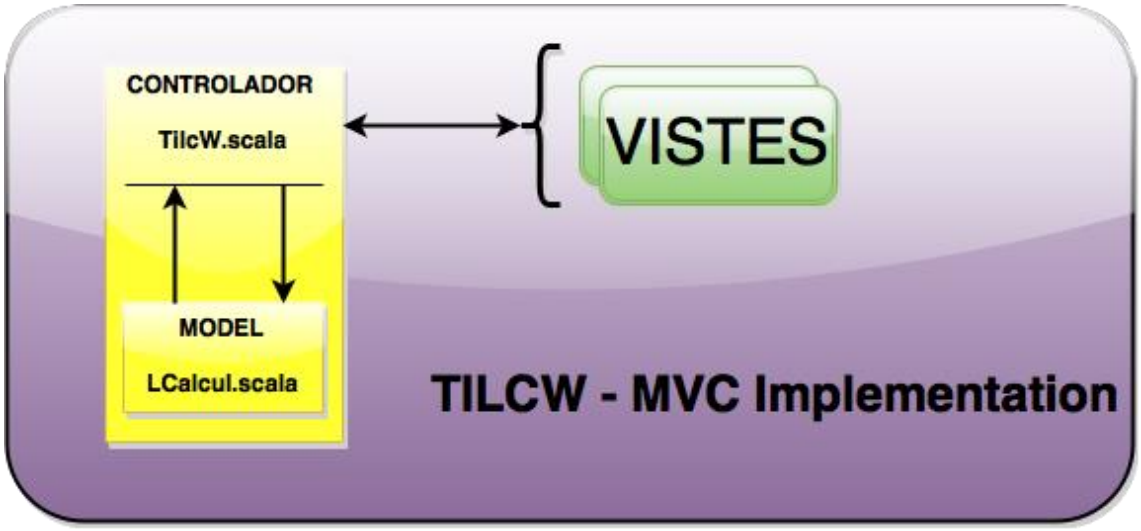
Les parts d'aquest patró queden clarament diferenciades ja que tenim una part que interactua amb l'usuari (Vista), una part no visible que controla (Controlador) possibles canvis que l'usuari pugui fer sobre la vista i finalment una tercera tampoc visible anomenada model que és amb la qual l'usuari interactua a través de les vistes. El controlador és l'encarregat d'actualitzar el model en funció dels canvis que faci l'usuari i retornar a les vistes els canvis sobre el model que s'hagin pogut produir.

Tot i que s'explicarà amb detall durant aquest apartat de la memòria quina és cada part del projecte en el patró, es presenta aquí esquema molt senzill del patró per entendre'l gràficament:



El patró MVC a alt nivell al TilcW

Vist l'esquema anterior, a molt alt nivell veiem aquí sota els noms de les classes del TilcW, demostrant doncs que l'aplicació manté l'estructura del Patró MVC però amb alguna possible desviació sobre el que seria l'estructura pura. Això és degut a que enlloc de tenir el model fora del controlador, s'ha decidit tenir-ne una instància dins el propi controlador, per agrupar i permetre treballar a l'aplicació d'una manera més flexible:

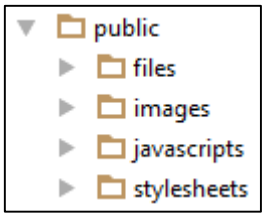


El Play 2.0 comunica el controlador amb el servidor mitjançant un fitxer de rutes a pàgines amb les comandes GET/POST/PUT/DELETE. En aquest punt va aparèixer un problema i és que algunes parts de les vistes del Tilc Gràfic i el Tilc Text havien de ser elements d'una mateixa pàgina, no pàgines noves. Després d'estar-hi treballant uns dies, es va decidir usar formularis html que s'enviessin al Controlador i ell gestionaria de qui venia i a on ho tornava. Així doncs, el fitxer de rutes va quedar de la següent manera:

```
# Home page
GET         /                       controllers.TilcW.tilcg

# Text page
GET         /text                   controllers.TilcW.tilct

# Definitions page
GET         /def                    controllers.TilcW.tilcd

# Help page
GET         /help                   controllers.TilcW.tilca

# Grafic Text
POST        /                       controllers.TilcW.opcions

# Forms Text
POST        /text                   controllers.TilcW.opcions

# Forms Def
POST        /def                    controllers.TilcW.opcions

# Map static resources from the /public folder to the /assets URL path
GET         /assets/*file           controllers.Assets.at(path="/public", file)
```

Fitxer "routes" de l'aplicació TilcW

Com podem veure, cada GET i POST invoca un mètode del Controlador que ell ho gestiona com a una nova pàgina, i els tres POST criden al mateix mètode que mitjançant un "case" al controlador sap qui l'ha invocat i què ha de fer. L'últim element, el get, és el que permet des de qualsevol vista/controlador de l'aplicació cridar els fitxers que estan a:

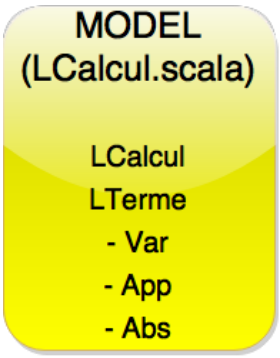


Fitxer de recursos públics de l'aplicació TilcW

Aqui dins i trobem possibles imatges compartides, llibreries javascript globals (jQuery, D3, Apprise), els fitxers CSS3 de cada vista i fitxers varis que interessi tenir dins la carpeta "files".

Tot seguit es passarà a descriure cada un dels elements, primerament a un alt nivell, és a dir, el seu esquema i una petita explicació i llavors els elements que hi intervenen i perquè. En el cas del Model i el Controlador però, no s'explicaran totes i cada una de les operacions ni tampoc en el cas de les Vistes les funcions Javascript que hi intervenen, la justificació del seu ús i el perquè de la implementació ja que moltes d'aquestes operacions i funcions són auxiliars de les principals per a fer més entenedor el codi.

El Model



Com podem veure més que model, s'hauria d'anomenar models, ja que la classe *LCalcul.scala* incorpora dues subclasses, la LCalcul i la classe abstracta LTerme, que mitjançant Scala es representa com a una *case class* que s'explicarà seguidament amb més detall. Aquest és l'objecte que canvia cada vegada que l'usuari interactua amb ell a través de les vistes. El Controlador informa a la classe LCalcul quina operació aplicar sobre la classe LTerme de manera que aquest vagi evolucionant o involucionant en funció de l'operació aplicada.

Si haguéssim de pensar en un Model-Vista-Controlador pur, el que es podria fer seria afegir totes les operacions i el parser dins el propi controlador i l'aplicació funcionaria igualment, però el controlador passaria a ser un fitxer de codi de més de 2000 línies, on, segurament i per modularitzar el codi, o s'acabaria fent en una classe apart el conjunt d'operacions dins el propi fitxer del controlador. Es va començar fent-ho així ja que en un principi no es va desenvolupar l'aplicació al Play 2.0 en paral·lel amb la del backoffice i no s'ha cregut convenient canviar la modularització actual.

Començarem presentant la classe `LTerme`, que és la que ens representa el Lambda Terme, sobre el qual es treballarà tota l'estona. La classe es codifica així:

```
abstract class LTerme
case class Var(nom: String) extends LTerme
case class App(a: LTerme, b: LTerme) extends LTerme
case class Abs(a: Var, b: LTerme) extends LTerme
```

La classe és un Lambda Terme i tot i que un cop s'ha parsejat correctament la cadena de text el que realment queda és una estructura de `Var`, `App` o bé `Abs` en funció del terme entrat. Aquí ho podem veure:

```
val a:App = App(Var("x"),Var("x"))
val b:LTerme = lc.LCalcul("x x")
```

```
a: models.App = App(Var(x),Var(x))
b: models.LTerme = App(Var(x),Var(x))
```

Com es pot veure a la imatge anterior, tota cadena de text parsejada correctament quedarà codificada com una estructura de *case class* de LTerme, però l'exemple és per mostrar en part que si es vol, es pot definir un element de forma individual ja que veiem que "`a`" és una `App` però també es comporta com un LTerme. És per això que enlloc de diferents classes, s'ha usat per implementar un LTerme mitjançant aquesta estructura on cada representació de LTerme es pot escriure en una sola línia.

Notes:

- Tot i que quan parlem de notació deBruijn parlem de nombres enters, els codifiquem a la *case class* com a String quan els codifiquem.
- En l'apartat del Treball Futur es parla d'una ampliació del Lambda Càlcul al Lambda Càlcul que permet inferir tipus a les variables. Veiem que aquest canvi no suposaria un gran problema sobre la classe LTerme ja que o bé els tipus es codificarien com a "String" o s'afegiria únicament una nova case class (anomenada VarTipada, per exemple) amb la qual representar-les.
- Aprofitant que es fa l'apunt del comportament del tipus i s'explica a l'apart 5 el comportament del compilador d'Scala referent als tipus, s'afegeix aquí una imatge on es demostra el que s'explica anteriorment (`a` no s'especifica el tipus, `b` és una aplicació que estén `LTerme` i c si és pròpiament un `LTerme`):

```
val a = App(Var("x"),Var("x"))
val b:App = App(Var("x"),Var("x"))
val c:LTerme = App(Var("x"),Var("x"))
```

Aquí acaba l'explicació sobre la implementació de la classe que representa l'objecte d'estudi. A l'apartat de les proves i resultats es veuran exemples més detallats i amplis del mateix.

La classe `LCalcul` són el parser i el conjunt d'operacions que representen el Lambda Càlcul, interactuant amb el Lambda Terme definit, i permetent per exemple que enlloc d'un conjunt de classes es pugui veure la representació d'un Lambda Terme amb tots els parèntesis o només els obligats, poder aplicar passos de reducció, passar de notació normal a notació de DeBruijn i tot el conjunt complet d'operacions. La seva capçalera és la següent:

```
class LCalcul extends StdTokenParsers with PackratParsers
```

Com s'ha dit i com es desprèn a la capçalera de la classe, aquesta incorpora un parsejador. Apart de l'importació de la base del parser `l'StdTokenParsers` es fa ús d'un parser anomenat `PackratParser`, que permet evitar recursivitat a l'esquerra, cosa que succeeix si es fa una Gramàtica Lliure de Context per al Lambda Càlcul i se'n generen les regles pertinents. El que farem serà veure i analitzar aquestes regles, i, com es veurà a l'apartat de proves de la memòria, funcionen correctament tot i haver-se hagut de modificar per afegir noves funcionalitats al parser. Així doncs, a l'entrar l'usuari una cadena de text pot:

- Obtenir un Lambda Terme compost per una o més variables, o qualsevol combinació de Aplicacions i Abstraccions que defineixin un Lambda Terme.
- Entrar una nova definició si es segueix la sintaxi d'entrada correcta. Això és el nom d'una variable seguida d'un = i un lambda terme vàlid.

La gramàtica simple per obtenir el terme seria com la donada aquí:

entry → *lterme*

lterme → *app* | *abs* | *variable*

variable → "Char" (lletra o dígit)

De manera recursiva i al llarg de tota la cadena de text el que fa el parser és mirar d'obtenir un Lambda Terme a partir de la cadena. Aquesta gramàtica és recursiva a l'esquerra, per tant abans de fer-la, cal reestructurar les regles com ho fem en el parser:

```
type Tokens = StdLexical
val lexical = new LambdaLexer
lexical.delimiters ++= Seq("λ", ".", ",", "(", ")","\\","|","/","=")

type P[+T] = PackratParser[T]
lazy val entry = definicio | lterme
lazy val definicio = ident~"="~lterme^^{...}
lazy val lterme: P[LTerme] =  app | nApp
lazy val nApp = variable | parentesi | absDB | abs
lazy val absDB: P[Abs] = ("λ" | "\\" | "|" | "/")~("." | ",")~ lterme ^^ {
  case x ~ y  => Abs(Var(""),y)}
lazy val abs: P[Abs] = ("λ" | "\\" | "|" | "/")~> variable ~ ("." | ",")  ~ lterme ^^ {
  case x ~ ("." | ",") ~ y  => Abs(x,y)}
lazy val app: P[App] = lterme ~ nApp ^^ { case x ~ y => App(x,y) }
lazy val variable: P[Var] = ident^^Var
lazy val parentesi: P[LTerme] = "(" ~> lterme <~ ")"


class LambdaLexer extends StdLexical {
  override def letter = elem("letter", c => c.isLetterOrDigit && c != 'λ')
}
```

El Parser del TilcW

Per entendre bé què fa el parser donada una cadena de text, cal entendre primer uns quants elements que apareixen a la imatge superior.

El primer que veiem és el *lexical.delimiters*, això és que donat un seguit d'elements quin són delimitadors i no s'han de considerar Tokens (identificadors). Així doncs, ens cal afegir tots els elements que volem que no puguin formar part del nom d'una variable o que formin part de la notació.

Els caràcters especials que apareixen per definir les regles i els seus significats:

- | = La disjunció lògica. La usem per dir que un element pot ser vàries coses.
- ~> = "El que hi hagi a la meva dreta ignorant el de l'esquerra"
- <~ = "El que hi hagi a la meva esquerra ignorat el de la dreta"
- ~ = "El que hi hagi a l'esquerra seguit del que hi hagi a la dreta"
- ^^ = "si és correcte, llavors {El que sigui}"

Com que volem permetre també l'entrada de termes en format deBruijn, apart d'afegir-hi la regla específica, a la classe LambdaLexer cal dir-li que una lletra no és només un "letter" sinó que és un "isLetterOrDigit" i que el caràcter "Lambda" no és una lletra, altrament es permetria aquest com a nom de variable i això no ha de ser possible.

Per acabar les modificacions al parser, com que es necessitava poder entrar definicions, es va implementar una nova regla recursiva que donada la següent expressió "variable=LambdaTerme" ens generés una entrada a un mapa de definicions, on variable seria la clau i LambdaTerme el valor i el terme que la representaria.

El que farem ara serà mostrar alguna de les funcions o mètodes més importants o interessants de la classe LCalcul, ja sigui per a veure'n el funcionament o per entendre l'avantatge d'usar programació funcional i imperativa per contra d'usar-ne només una de les dues. A l'apartat de les proves es veuran exemples del correcte funcionament de les mateixes.

Dir que per a moltes funcions, el que s'ha fet és aplicar la tècnica de la immersió apresa a "Estructura de Dades i Algorísmica" de manera que hi ha una funció que rep el Lambda Terme i que té a sota una crida a una funció que porta el mateix nom amb el prefix "*i*", ja sigui perquè moltes funcions requereixen ajustar prèviament atributs (com poden ser la neteja de Maps de dades) o requerir-ne més que només el Lambda Terme. És per això que es mostraran aquestes i no les que usa l'usuari realment. Totes aquestes funcions estan implementades de manera privada perquè els usuaris no puguin accedir-hi de manera directa.

Començarem veient el mètode que permet crear el Lambda Terme, un cop l'usuari ha entrat la cadena de text que el representa:

```
def LCalcul(nom: String): LTerme = {
  val a: LTerme = toLTerm(nom)
  if(a!=null) {
    if (esDB(a) || esLT(a)) {
      a
    } else null
  }else null
}
```

Funció LCalcul de la classe LCalcul.scala

El primer que es fa és enviar al parser la cadena de text `nom` i si és correcte, s'ha de fer un altre pas entremig, que és comprovar que l'usuari no hagi escrit un Lambda Terme que contingui notació de deBruijn i notació normal alhora. Si és així, el terme entrat tampoc s'ha d'acceptar com a vàlid. Si el resultat és correcte, "`a`" representarà el Lambda Terme sobre el qual l'usuari podrà treballar.

Tot seguit, veurem una altra funció interessant, que és al de mostrar el Lambda Terme amb tots els parèntesis obligats. Com gairebé la majoria de funcions que es presentaran i també les que no, s'usa la recursivitat amb *pattern-matching* per a recórrer l'arbre que representa el Terme i fer-hi modificacions o tractar-lo. Vegem-la:

```
def show(term: LTerme):String = {
  term match {
    case Var(x) => x
    case App(x, y) => "("+show(x)+" "+show(y)+")"
    case Abs(x, y) => "(λ"+show(x)+"."+show(y)+")"
  }
}
```

Funció show de la classe LCalcul.scala

Com s'ha comentat a dalt, aquesta funció rep l'arrel de l'arbre que representa el Lambda Terme creat i el que fa és recorre'l de dalt a baix mitjançant l'ús del *pattern-matching* per anar creant una cadena de text que representarà el mateix però amb tots els parèntesis obligats.

Una altra funció molt útil, ja que sense ella no es pot crear l'arbre gràficament, és la funció per recórrer tota l'estructura generant una cadena de text que el controlador s'encarregarà posteriorment de parsejar a JSON. Es mostra aquí una part i a sota s'expliquen els atributs usats per a la implementació i perquè són necessaris:

```
private def icreaJSON(t:LTerme):String ={
  t match {
    case Var(x) => "\"name\":"+"\""+x+"\",\"represent\":"+"\""+show(t)+"\",\"isRedex\":\"no\""+",\"actiu\":\"\""
    case App(x,y) =>
```

Funció icreaJSON a la classe LCalcul.scala

- **`name`**: El nom del node. A una variable és el seu nom pròpiament, a una aplicació "@" i a una abstracció "λ".
- **`represent`**: El terme que representa. Òbviament, l'arrel representarà tot el terme i les variables, la part més petita. Això és útil a l'hora de treballar amb l'string que apareix per pantalla.
- **`isRedex`**: Si és l'inici d'un redex, aquest valor val si. Per la resta val no. Tot i que gairebé tots els valors és no, el cost d'afegir-lo és pràcticament nul però molt útil per marcar a l'arbre els redex.
- **`actiu`**: Indica si aquest element és actiu o no. Ens referim a actiu si no té el color base d'un node de l'arbre.
- (opcional) **`freq`**: Marca les vegades que el redex ha aparegut dins l'arbre. Això és útil per si el mateix redex apareix més d'una vegada, poder-lo identificar dins la cadena de text.
- (opcional) **`children`**: cadena que representa els fills d'un node. El que conté és el text recursiu de la crida a icreaJSON.

Una funció molt útil, que serveix per a totes les operacions de reducció entre altres és el mètode de `hihaRedex`, que comprova si dins un terme hi ha un redex. Mitjançant aquesta funció podem buscar el redex més intern, o el més extern o bé aturar la reducció ja que podem saber si queden o no redex mitjançant aquesta funció.

```
def hihaRedex(terme: LTerme): Boolean = {
  lazy val redex: Boolean = terme match {
    case App(Abs(x, y), z) => true
    case App(x, y) => hihaRedex(x) || hihaRedex(y)
    case Abs(x, y) => hihaRedex(y)
    case Var(x) => false
  }
  redex
}
```

Funció hihaRedex de la classe LCalcul.scala

A l'hora de generar una nova variable, per si calia substituir-ne algunes degut a que substitucions altres quedarien lligades, es va haver de crear una funció que permetés aquest canvi. El programador va pensar en tenir una llista de variables en format String, però el client va suggerir una cosa millor. Definir una variable base (es va optar per "x") seguit d'@ i un

número en format de comptador que indiqués el nombre de variables generades fins al moment. Es va usar aquest mètode fins que els alumnes que van provar l'aplicació i el propi client va veure que no era del tot entenedor usar l'"@", que confon més que no ajuda. Així doncs, s'ha decidit que per a la generació de noves variables s'usaria x seguit d'un guió baix i de la n-èssima variable generada, començant per 0. Es pot veure la implementació de la funció aquí:

```
private def generarNovaVariable(): String = {
  val nouValor:String = "x_" ++ nGenerats.toString
  nGenerats+=1
  nouValor
}
```

Funció generarNovaVariable de la classe LCalcul.scala

Per acabar la introducció a les classes del model, mostrarem els dos mètodes gairebé bàsics del desenvolupament del Lambda Terme del TilcW, com són el que fa la reducció normal i el que fa la reducció aplicativa del Lambda Terme. Si bé és cert que el mètode que aplica la beta reducció és el mètode "`substitució`", aquests dos mètodes fan que mentre hi hagi redex dins un Lambda Terme, aquest es vagi reduint mitjançant l'ordre de reducció normal o aplicatiu, respectivament.

```
def redueixNormal(terme: LTerme): LTerme = {
  var termeReduit: LTerme = terme
  nGenerats=0
  while (hihaRedex(termeReduit)) {
    termeReduit = leftOutermost(termeReduit)
    nRedex += 1
  }
  termeReduit
}
```

Mètode redueixNormal de la classe LCalcul.scala

```
def redueixAplicatiu(terme: LTerme): LTerme = {
  var termeReduit: LTerme = terme
  nGenerats=0
  while (hihaRedex(termeReduit)) {
    termeReduit = leftInnermost(termeReduit)
    nRedex += 1
  }
  termeReduit
}
```

Mètode redueixAplicatiu de la classe LCalcul.scala

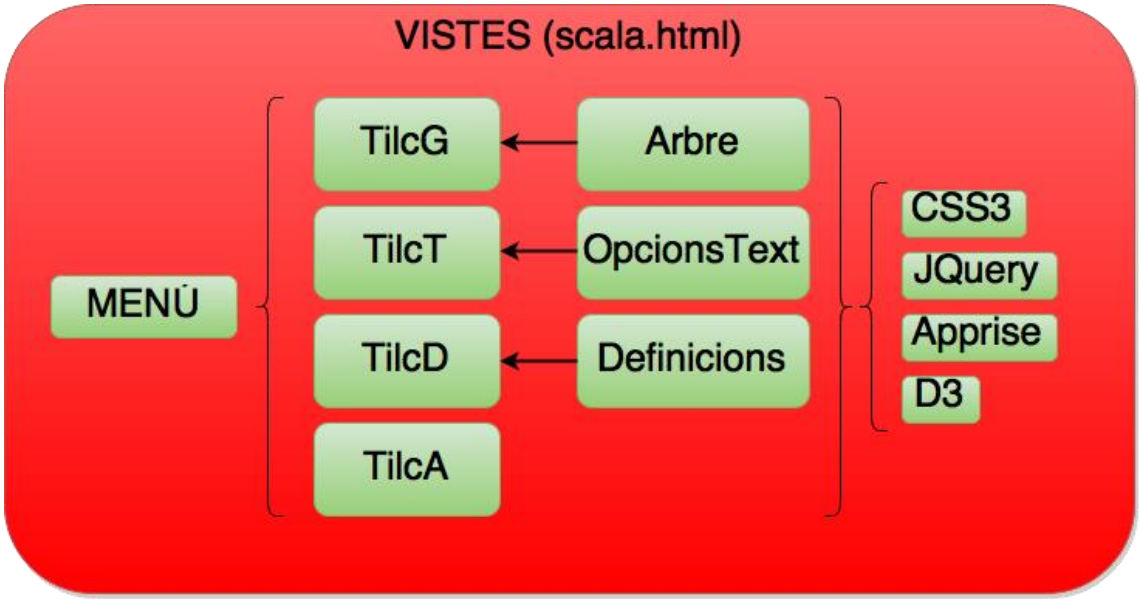
Les Vistes

Les 4 vistes amb les quals pot interactuar l'usuari són les següents:

- TilcG – Permet tractar el Lambda Terme Gràficament.
- TilcT – Permet tractar el terme en forma de text.
- TilcG – Permet entrar definicions i esborrar-ne.
- TilcA – Permet visualitzar una petita ajuda del funcionament de l'aplicació.

Aquestes 4 vistes usen vistes auxiliars i també fitxers CSS3 per a la maquetació. Les 4 usen a més funcions javascript integrades dins la mateixa vista, per comunicar-se amb el controlador mitjançant GET i POST de manera asíncrona ja que es va provar de fer-ho en fitxers externs però llavors les crides AJAX no funcionaven.

El seu esquema conceptual és el següent:



El menú és una vista comuna a les 4 vistes, i llavors TilcG/T/D tenen cadascun una vista dinàmica interna anomenada Arbre, OpcionsText i Definicions que permeten rebre les característiques de l'arbre, les característiques del Lamba Terme en el format text i la vista amb les definicions carregades a l'aplicació. Com que han de ser elements dinàmics i les vistes estan considerats com a objectes pròpiament, cal fer-ho en vistes apart i llavors incloure-les dins la vista principal un cop estan refrescades.

L'avantatge d'aquestes vistes, com indica la seva extensió (scala.html) és que es pot barrejar codi Scala amb Html si s'introdueix amb el caràcter comodí "`@`", i això permet carregar vistes dins vistes de manera estàtica o crear un codi Html a mida mitjançant elements de la programació Scala (`@if`,`@while`,etc). Per exemple, el menú:

```
<div class="navegacio">
        <!-- Crida al menú lateral de navegació -->
    @menu()
</div>
```

Tros de codi extret de la vista TilcG.scala.html

Com veiem, la vista menú queda inclosa dins el div de navegació. Si poséssim dos menú seguits evidentment apareixeria dos cops o bé, si el menú depengués d'un paràmetre i en tinguéssim dos, podríem mitjançant un "`@if(condició)`" escollir quin menú mostrar, però no és el cas. Les vistes poden tenir elements com a paràmetre, en el nostre cas usem sempre un títol (que podem invocar a on sigui del codi mitjançant `@titol`) i en el cas de les

pàgines dinàmiques, una primera versió de les mateixes però buides. Un defecte que té això és que si es fa una crida a un element del controlador de manera directa, al fer crides asíncrones, o bé serà un valor "*undefined*" o bé un valor no actualitzat. És en part per això que s'usen aquestes vistes auxiliars, que són retornades pel controlador, i per tant contenen els valors actualitzats.

Com ja s'ha dit, les crides al controlador es fan totes de manera molt similar, i ocupen moltes línies de codi per a ser implementades. Així doncs, aquí es mostrarà la implementació de l'entrada del Lambda Terme i s'explicaran les funcions que no usen crides al servidor i que per tant són "funcions" Javascript, com a exemple.

```
$("#entrarLTerme").submit(function(e)
{
    var formURL = $(this).attr("action");
    var aux = $("#ent").text();
    $.ajax(
        {
            url : formURL,
            type: "POST",
            data : { valor: aux, op: "0",pag: "2" },
            success:function(data)
            {
                ordres=[];
                index=0;
                $("#n").prop("checked", true);
                $("#db").prop("checked", false);
                $("#veureOpcions").html(data);
                termeActual = $(data).find("#tp").html();
                backup = termeActual;
                $("#termeAnterior").html(backup);
                $("#ent").text(termeActual);
            },
            error: function()
            {
                apprise('El terme entrat no es corresponia a un LTerme, torna-ho a provar.');
            }
        });
    e.preventDefault(); //STOP default action
    e.unbind(); //unbind. to stop multiple form submit.
});
```

Funció Javascript referent a la crida per entrar un Lambda Terme en format text extreta de TilcT.scala.html

On els valors més significatius són els següents:

`aux`: conté el text escrit al "p" identificat com a **`ent`** i és el possible Lambda Terme.

Si hi ha hagut èxit a la crida al servidor, se'ns retornen del servidor el Lambda Terme amb tots els parèntesis, es posa l'índex de reduccions i la llista de reduccions a 0, es guarda una còpia del terme per quan l'usuari vulgui fer un reset i es posa a l'entrada el terme. Si hi ha hagut algun error, apprise ens avisa de l'error amb un alert.

El següent llistat de funcions usen una estructura molt similar i són vàlids tant per al TilcW Gràfic i el TilcW Text. El que fan aquestes funcions és recollir de les vistes els canvis que apliqui l'usuari i enviar-los al servidor, per reomplir els camps de les vistes que necessitin ser actualitzats:

- Entrar el Lambda Terme en una àrea de text.
- Escollir l'ordre de reducció sobre el qual es vol reduir el terme, ja sigui el normal l'aplicatiu o de manera lliure (únicament en el cas del TilcW Gràfic).
- Recular o avançar en les transformacions del Lambda Terme.
- Veure directament la forma normal del Lambda Terme.
- Reiniciar el Lambda Terme.
- Borrar el Lambda Terme entrat.
- Cambiar la vista de la notació del Lambda Terme, o normal o en deBruijn i viceversa.

Vistes TilcW Gràfic i TilcW Text

El TilcW Gràfic i Text tenen una última operació comuna implementada via funció Javascript i requerida pel client un cop l'aplicació va ser funcional. Podria ser pesat que, si es vol veure el Lambda Terme d'una manera o d'una altra, s'hagués de reescriure de nou cada vegada. És per això que s'han implementat dues funcions, anomenades *"carregar"* (que s'executa automàticament al carregar la pàgina) i una altra sense nom pròpiament que, al canviar de pestanya, s'encarreguen de carregar o descarregar de la sessió del navegador (en una variable anomenada **`sessionStorage`**) possibles dades que hi hagi referents al TilcW de manera que al canviar de pestanya es mantingui tant el Lambda Terme entrat com en el punt de reducció en què s'ha deixat en l'altra pestanya i viceversa. En veurem un exemple d'ús a l'apartat de les proves.

L'apartat Gràfic té una funció anomenada *`actualitzarArbre(var)`* on var representa el node de partida en format JSON la qual crea l'arbre corresponent a aquest codi, és l'encarregada de generar el tros de codi HTML corresponent al nodes que es mostraran per pantalla, el color i les característiques de cadascun d'ells. Aquesta funció ha de ser cridada sempre que es faci un canvi a la vista que impliqui un nou arbre i també dins la funció *carregar*, ja que al moure's d'una pestanya a una altra, l'usuari potser ha aplicat una reducció (o ha reculat dins l'històric) i per tant l'arbre a mostrar ja no és el mateix.

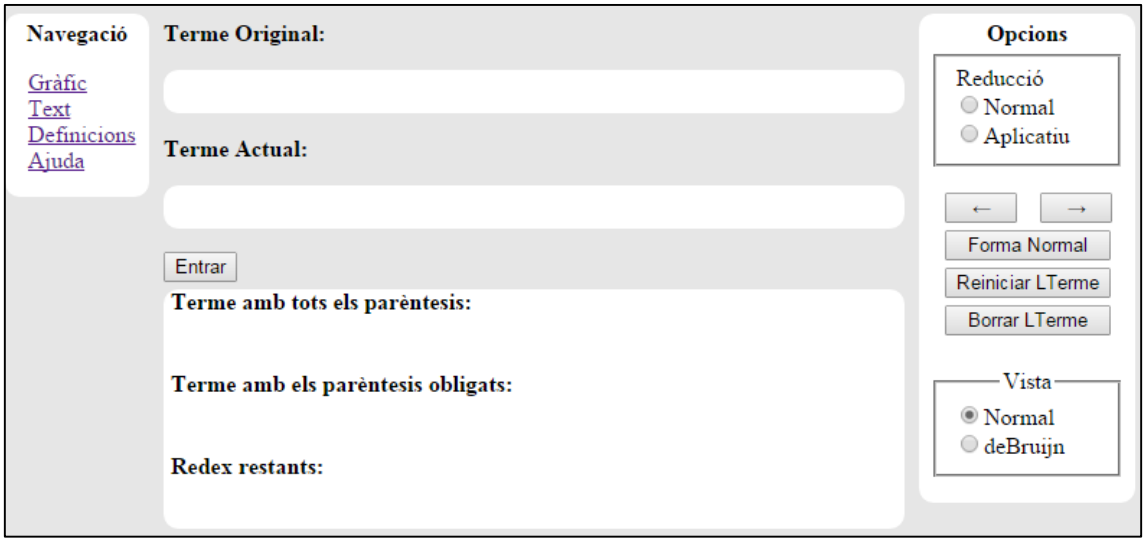
Vista TilcW Definicions

Aquesta vista és en la qual l'usuari podrà entrar, actualitzar i esborrar definicions. Com s'ha comentat, aquesta vista és totalment funcional, però no la connexió de les definicions dins l'entrada del Lambda Terme. Es compon bàsicament de dues crides al controlador, una que afegeix una nova definició o n'actualitza una d'existent o bé una que permet eliminar-ne de ja entrades. La part central és una vista auxiliar dinàmica que s'explicarà més avall.

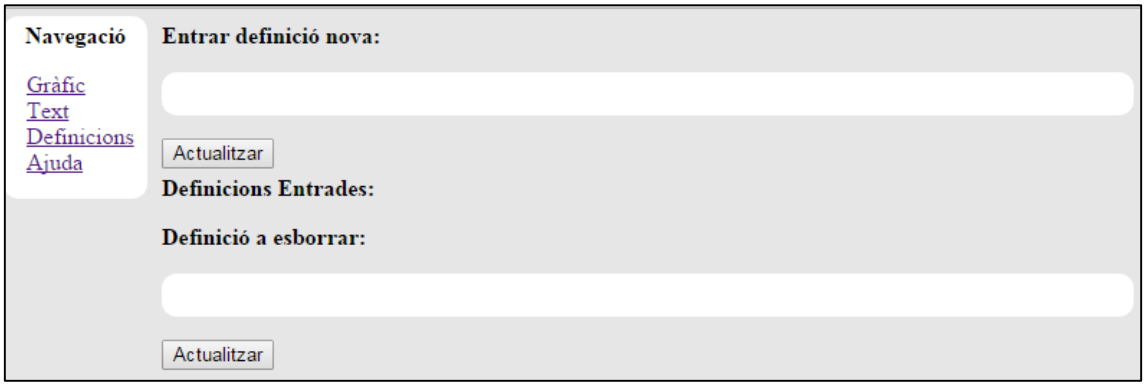
Vista TilcW Ajuda

La vista d'ajuda és la més simple de totes, però no per això ha de ser la menys important. Conté una ajuda sobre com usar les diferents pestanyes del programa i dos links (només 1 serà visible si estem en l'aplicació d'escriptori) on un permetrà baixar un manual d'usuari en pdf i l'altre el TilcW de versió d'escriptori.

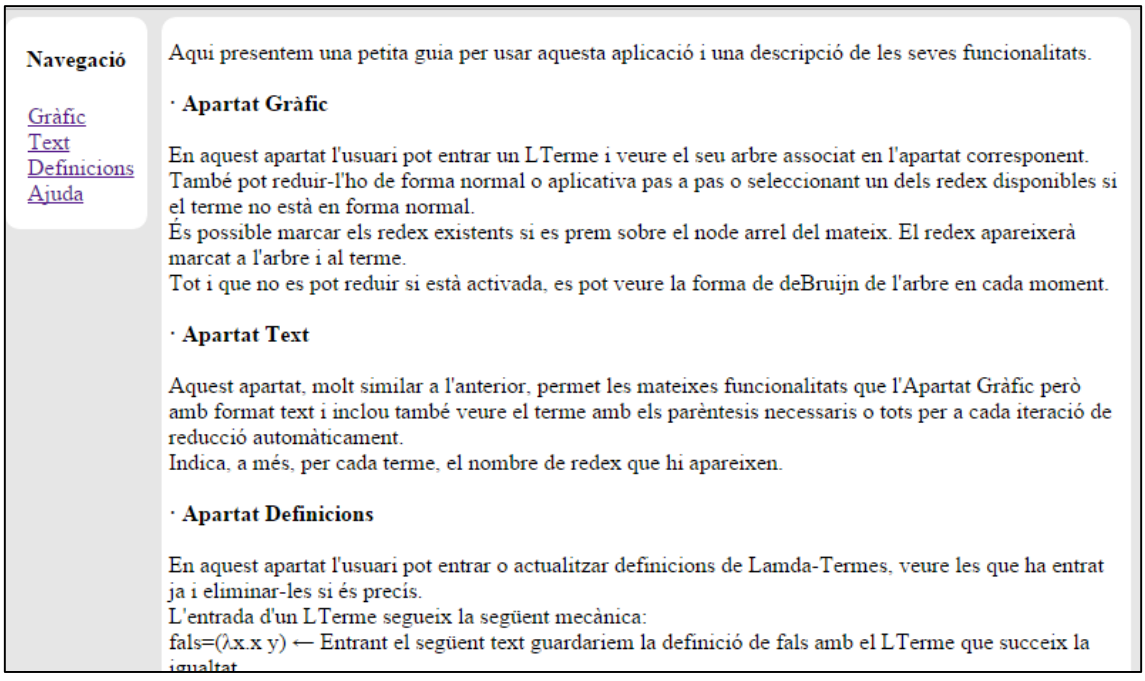
Aquí mostrem una vista de cadascuna d'elles en el seu estat inicial:



Pantalla inicial del TilcG



Pantalla inicial del TilcD



Fragment de la pantalla d'ajuda del TilcA

Ara explicarem les 3 vistes auxiliars que apareixen al TilcW, i que són, com diem, elements dinàmics que varien en funció del que esculli l'usuari.

Vista arbre

Aquesta vista envia del servidor a la vista TilcG el conjunt de dades com serien l'String JSON a parsejar per a construir l'arbre, l'avís sobre si el terme està normalitzat i les cadenes de text del terme anterior i el terme actual.

Vista opcionsText

Aquesta vista incorpora més informació que la vista arbre, ja que aquesta ens mostra el terme amb tots els parèntesis, només els obligats i mostra també el nombre de redexs restants o avisa a l'usuari si està en forma normal. Per contra, no incorpora el terme JSON com a la vista explicada al paràgraf anterior.

Vista definicions

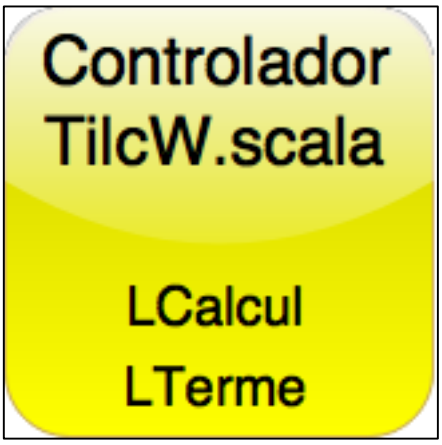
Aquesta vista informa a TilcG de la mida del mapa i, mitjançant una llista per comprensió, crea una "unordered list" amb les claus i valor de cada definició. És interessant veure aquest tros de la vista ja que com s'ha comentat anteriorment, aquestes vistes permeten barrejar codi Scala amb HTML5, però no se n'ha posat cap exemple gràfic:

```
<h4>Definicions Entrades:</h4>
<ul id="lectura">
    @for((key,value) <- TilcW.getDef) {
        @key @("=") @value._1 <br>
    }
</ul>
```

Creació de la llista de les definicions a definicions.scala.html

Aquest doncs és el conjunt de vistes del patró MVC que componen l'aplicació del TilcW i que permeten que sigui com està actualment. Com s'ha explicat a l'apartat 3, s'ha mirat de premiar la funcionalitat sobre la correctesa del codi, de manera que segurament moltes parts, combinant la potència de l'Scala amb l'HTML5 podrien ser més simples o escriure's d'una manera més senzilla. Aquesta tasca de correcció es tindrà en compte en futures revisions.

El Controlador



Com hem dit anteriorment, el controlador és l'objecte encarregat d'actualitzar les vistes si i el model ha canviat al fer un canvi l'usuari. En el nostre cas és, per exemple, que l'usuari hagi volgut avançar fent un pas de reducció sobre el Terme que hagi entrat anteriorment.

El primer problema plantejat va ser que el Play 2.0 redirigeix cada "canvi" en una nova pàgina, i no és el que volíem, sinó que donada una pàgina part del contingut variés de forma dinàmica sense haver-ho de fer la totalitat de la pàgina.

El segon problema va ser doble però la seva solució va ser més bona de l'esperada. Com s'ha vist, la pàgina routes.conf redirigeix cada element GET/POST/PUT/DELETE a una funció. Degut a això, més d'una pàgina no pot tenir la mateixa direcció tot i ser funcions diferents. És a dir, per exemple, no podíem aplicar 2 GET a la pàgina "/" tot i una anar a tilcg i l'altra ser opcions. Així doncs, es va decidir que si es volia usar pàgines dinàmiques a dins de pàgines estàtiques no es podria fer mitjançant GET *request* sinó mitjançant POST. Això també aniria millor perquè via POST l'usuari no veu els paràmetres que s'envien i la informació que s'envia pot ser molt major que via GET.

Per fer això, es va decidir fer les crides POST mitjançant "*forms*" HTML, a qual consten d'una crida mitjançant Ajax al servidor i aquest s'encarrega de gestionar-les. En el nostre cas, el servidor era el controlador i per tant va afegir-se a ell un atribut val que era el nostre Form que contenia tres camps de tipus text. Un seria el valor del que s'enviava, l'altre, anomenat op, seria l'opció escollida i, finalment, pag, ja que al voler-ho unificar, podríem mitjançant aquest atribut saber de quina de les tres pàgines que requerien del form procedia la crida. Aqui en tenim la implementació:

```
val opcionsDefinicions = Form(
    tuple("valor" -> text,
    "op" -> text,
    "pag" -> text)
)
```

Formulari opcionsDefinicions de la classe Controlador del TilcW

Com es veu a l'arxiu routes, la funció que serveix per a enviar el formulari es diu "`opcions`". Aquesta, mitjançant el *pattern-matching*, el que va fent és separar la tripleta (de més a menys separació possible) per tal de saber quina acció ha d'aplicar a cada cas. Les referències són les següents:

pàgina 1: definicions, pàgina 2: text, pàgina 3: gràfic

Pàgina	Opció	Definició	Comanda
1	0	X	Afegir definició
1	1	X	Eliminar definició
2	0	X	Entrar LTerme
2	1	X	Recular Reducció
2	2.1	X	Reducció Ordre Normal
2	2.2	X	Reducció Ordre Aplicatiu
2	3	X	Reiniciar LTerme
2	4	X	Borrar LTerme
2	5	0	Notació normal to DeBruijn
2	5	1	Notació DeBruijn to normal
2	7	X	Obtenir Forma Normal
3	2.1	X	Reducció Ordre Normal
3	2.2	X	Reducció Ordre Aplicatiu
3	2.3	X	Reducció Ordre Lliure
3	3	X	Reiniciar LTerme
3	4	X	Borrar LTerme
3	5	0	Notació normal to DeBruijn
3	5	1	Notació DeBruijn to normal
3	7	X	Obtenir Forma Normal

Taula de referències d'accions del Formulari al controlador de l'aplicació TilcW

Nota: Com potser es pot observar, l'ordre de les opcions passa del 5 al 7. això és degut a que el número 6 s'havia reservat per a una funció no disponible actualment, i s'ha respectat aquest ordre.

En aquest punt, és interessant veure els atributs dels quals disposa el controlador, per entendre una mica millor com funciona el mateix:

```
val b = new LCalcul
var termeactual:LTerme = null
var termeoriginal:LTerme = null
var termeDB:LTerme = null
var ordreReduccio: ArrayBuffer[String] = new ArrayBuffer[String]()
var historicLT: ArrayBuffer[LTerme] = new ArrayBuffer[LTerme]()
var historicShow: ArrayBuffer[String] = new ArrayBuffer[String]()
var historicShow0: ArrayBuffer[String] = new ArrayBuffer[String]()
var historicShowd: ArrayBuffer[String] = new ArrayBuffer[String]()
var historicShow0d: ArrayBuffer[String] = new ArrayBuffer[String]()
var historicJSON: ArrayBuffer[JsValue] = new ArrayBuffer[JsValue]()
var anteriorsGrafic: ArrayBuffer[Html] = new ArrayBuffer[Html]()
var anteriorsText: ArrayBuffer[Html] = new ArrayBuffer[Html]()
var index:Int=0
var isDebruijn:Boolean = false
var normalitzat:String = "fals"
```

Llistat d'atributs de la classe Controlador de l'aplicació TilcW

Com s'ha dit, aquesta classe incorpora una instància de la classe LCalcul, que és la que s'usa per a fer la majoria de canvis sobre el Lambda Terme. També tenim 3 instàncies de la classe Lambda Terme, per agilitat. La primera és el Lambda Terme cambiant, la segona serveix per a guardar una còpia del primer terme entrat, com a "*backup*". Finalment tenim una última instància que ens representa el terme en notació de DeBruijn, així no hem de passar d'un a l'altre i viceversa, només cal que canviem aquesta instància cada vegada que volguem mostrar el terme en DeBruijn.

El fet de tenir tantes llistes és perquè no està implementat en la classe Lambda Càlcul (i de fet, tampoc es podria) donat un lambda terme saber-ne la forma anterior. Així doncs, aquí guardem com era, l'ordre amb què s'havia reduït (**important!** si no s'aplica el mateix ordre, l'aplicació no deixa recular) i les cadenes de JSON generades. A l'hora d'avançar, es mira si cal crear un nou element a la llista o cal actualitzar l'existent, ja que podem tenir el mateix ordre de reducció però no el mateix terme, així que caldrà sempre que s'avanci modificar un registre de la taula. Si no existissin aquestes llistes, no seria possible recular en el Lambda Terme ja que no sabem d'on partim. Evidentment, els atributs `index`, `isDebruijn` i `normalitzat` són atributs auxiliars que ens permeten saber en quin element de la taula ens trobem, el tipus de vista de l'arbre i si el terme ja està en forma normal o no.

Finalment, el que retorna sempre una request del controlador a la vista és una `Action`. Aquí mostrem les `Action` que retorna el nostre controlador.

```
/**...*/
def tilcg = Action {
  Ok(views.html.tilcg.render("Tilc(W)-Gràfic",views.html.arbre()))
}

/**...*/
def tilct = Action {
  Ok(views.html.tilct.render("Tilc(W)-Text",views.html.opcionsText()))
}

/**...*/
def tilcd = Action {
  Ok(views.html.tilcd.render("Tilc(W)-Definicions",getDef.size.toString))
}

/**...*/
def tilca = Action {
  Ok(views.html.tilca.render("Tilc(W)-Ajuda"))
}

/**...*/
def opcions = Action {implicit request =>
  val (definicio,opcio,pagina) = opcionsDefinicions.bindFromRequest.get
  pagina match{
    case "1" =>     opcio match {...}
                    Ok(views.html.definicions())
    case "2" =>     opcio match {...}
                    Ok(views.html.opcionsText())
    case "3" =>     opcio match {...}
                    Ok(views.html.arbre())
  }
}
```

Retorn de les Actions del controlador a TilcW

8. Implantació, proves i resultats

En aquest apartat anirem explicant quines proves s'han anat fent durant el transcurs de tot el projecte. Primer mostrarem les proves fetes al backoffice, a la majoria de funcions implementades. Algunes d'aquestes no són requisits funcionals de l'aplicació però si funcions útils per a testejar l'aplicació. Altres, en canvi, si que són requerides ja que sense elles l'apartat gràfic no funcionaria.

Per a fer tot el conjunt de proves del backoffice, s'ha usat un *worksheet* (que s'ha anomenat proves), un software propi d'Scala que permet veure els resultats en paral·lel sense haver de recompilar cada vegada sempre i quan el codi no s'hagi tocat. Altrament si que caldrà. A més, permet fer-ho en mode interactiu, és a dir, a mesura que escrius van apareixent els resultats. Les captures de pantalla derivades de les proves provenen d'aquest tipus de fitxer. El primer que veurem és l'entrada de dos Lambda Termes diferents i com es veuen amb tots els parèntesis i llavors amb només amb els obligats. Són els següents:

(λx.x(λy.x y) (λx.y x)) (λx.x y)

((λx.((x (λy.x)) x)) ((λz.((y z) y)) y))

```
lc.show(a)                                  res0: String = ((λx.((x (λy.(x y))) (λx.(y x)))) (λx.(x y)))
lc.showO(a)                                 res1: String = (λx.x(λy.x y) λx.y x) λx.x y

lc.show(b)                                  res2: String = ((λx.((x (λy.x)) x)) ((λz.((y z) y)) y))
lc.showO(b)                                 res3: String = (λx.x(λy.x) x) ((λz.y z y)y)
```

El client també va voler que s'implementessin diferents lambda termes vistos a classe, com serien els números, la suma, el producte, el successor, el predecessor i el factorial, i que a més servirien de test per a comprovar que s'estava treballant de manera correcta ja que si no donaven els valors aritmètics correctes, alguna cosa no es feia bé. Es va crear a més una altra funció inversa a la creació d'un número, que donat un Lambda Terme n'obtenia l'enter corresponent (en format text). Com que els resultats d'aquestes operacions impliquen l'obtenció de la forma normal per ordre de reducció normal, ja es testejava també si la beta reducció i la substitució funcionaven. En aquestes dues captures de pantalla podem veure un seguit d'operacions que s'han aplicat i les seves sortides en una altra captura, veient doncs que si que s'han obtingut els resultats esperats:

```
val c = lc.creaNumero("3")
val d = lc.creaNumero("2")
val e = lc.suma(c,d)
val f = lc.producte(c,d)
val g = lc.successor(f)
val h = lc.predecessor(f)
val i = lc.creaNumero("4")
val j = lc.factorial(i)


lc.show(c)
lc.show(d)
lc.show(e)
lc.descodificaNumero(e)
lc.show(f)
lc.descodificaNumero(f)
lc.show(g)
lc.descodificaNumero(g)
lc.show(h)
lc.descodificaNumero(h)
lc.show(j)
lc.descodificaNumero(j)
```

Entrada per comprovar que les funcions entrades donaven la sortida correcta

```
res4: String = (λf.(λx.(f (f (f x)))))
res5: String = (λf.(λx.(f (f x))))
res6: String = (λf.(λx.(f (f (f (f (f x)))))))
res7: String = 5
res8: String = (λf.(λx.(f (f (f (f (f (f x))))))))
res9: String = 6
res10: String = (λf.(λx.(f (f (f (f (f (f (f x)))))))))
res11: String = 7
res12: String = (λf.(λx.(f (f (f (f (f x)))))))
res13: String = 5
res14: String = (λk.(λl.(k (k (k (k (k (k (k (k (k (k (k (k (k (k (k (k (k (k (k (k (k (k (k (k l))))))))))))))))))))))))))
res15: String = 24
```

Sortida de les funcions “suma(3,2)”, producte(3,2), successor(6), predecessor(6) i factorial(4)

Per acabar les proves que es facin dins el backoffice, comprovarem que el fet de provar que un Lambda Terme estigui escrit en notació normal o deBruijn funcioni correctament. A la part de l’esquerra podem veure l’entrada i a la dreta la sortida, donat un terme escrit en notació normal i un escrit en notació de deBruijn:

```
val lc = new LCalcul
val a:LTerme = lc.LCalcul("(\\x.x)y")
val b:LTerme = lc.LCalcul("(\\.1)0")
lc.esDB(a)
lc.esLT(a)
lc.esDB(b)
lc.esLT(b)
```

```
a: models.LTerme = App(Abs(Var(x),Var(x)),Var(y))
b: models.LTerme = App(Abs(Var(),Var(1)),Var(0))
res0: Boolean = false
res1: Boolean = true
res2: Boolean = true
res3: Boolean = false
```

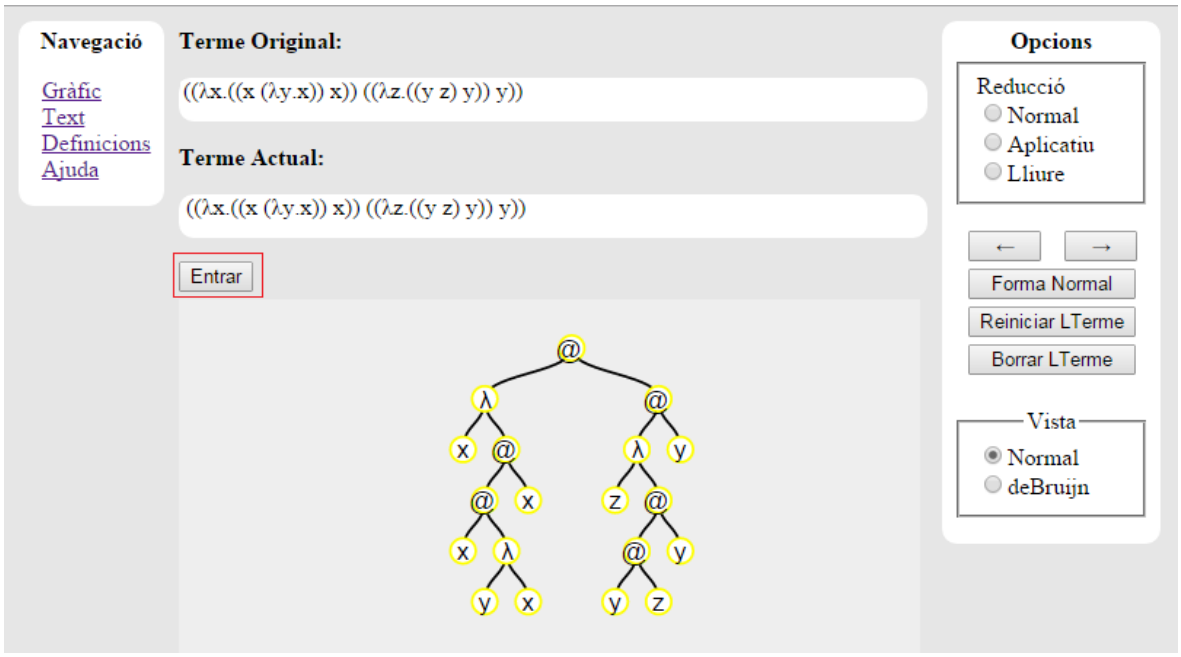
Entrada i sortida de les funcions que comproven si un terme està escrit en notació normal o de deBruijn

Nota: Ara es detallaran el conjunt de proves sobre les diferents opcions que apareixen al menú lateral del TilcW. Aquestes proves, per a més qualitat, s’han fet sobre la versió de 1024px del TilcW, però la mida del canvas de l’arbre sempre serà d’amplada el màxim que s’allargui la pantalla i fins a un màxim de 1025px d’alçada. Com que estem parlant d’una aplicació online, tots els camps de temps editables s’hi ha desactivat el corrector ortogràfic del navegador altrament faria acte de presència i marcaria les variables com a faltes d’ortografia.

Proves de funcionament al TilcW Gràfic

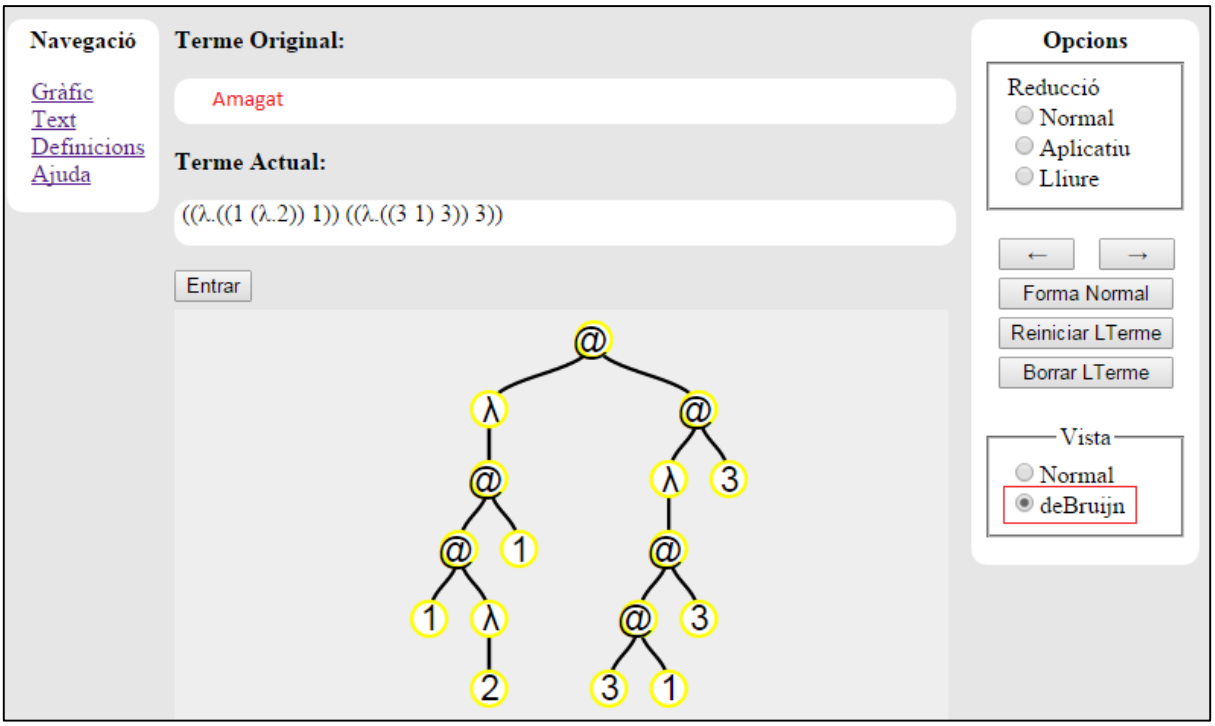
Per a veure que l'aplicació es comporta com s'esperava, el que es fa és anar descrivint el funcionament dels elements del menú d'opcions i mostrar, mitjançant captures de pantalla, que funcionen com s'esperava.

El primer que farem serà entrar un lambda terme. Quan aquest s'ha introduït, apareixerà per pantalla la seva representació en forma d'arbre, juntament amb la seva escriptura amb tots els parèntesis tant a la finestra del terme original com a la finestra del terme actual. Sempre es mostrarà així per tal que els alumnes no tinguin confusions a l'hora de veure el terme.



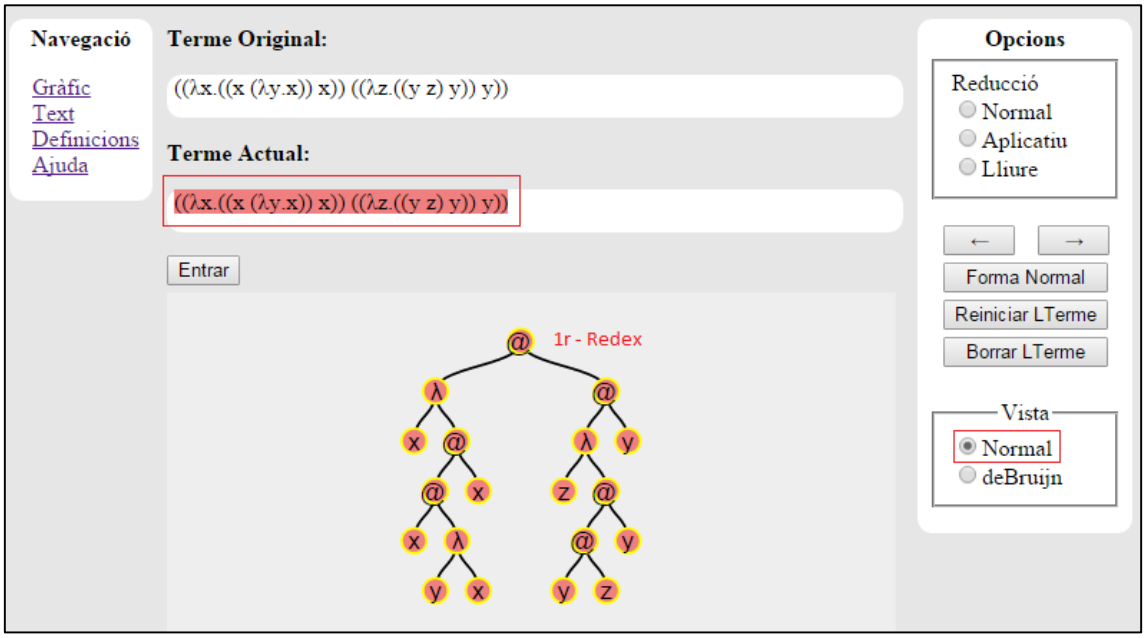
Entrada d'un Lambda Terme

Si volem veure aquest terme en notació de DeBruijn, només cal que activem el checkbox corresponent en les opcions del menú. Podrem alternar aquesta vista en cada pas de reducció del Lambda Terme i marcar els redexs existents, però no podrem aplicar el pas de reducció ni veure el redex aplicat quan estem en aquesta vista. Aqui en veiem un exemple:

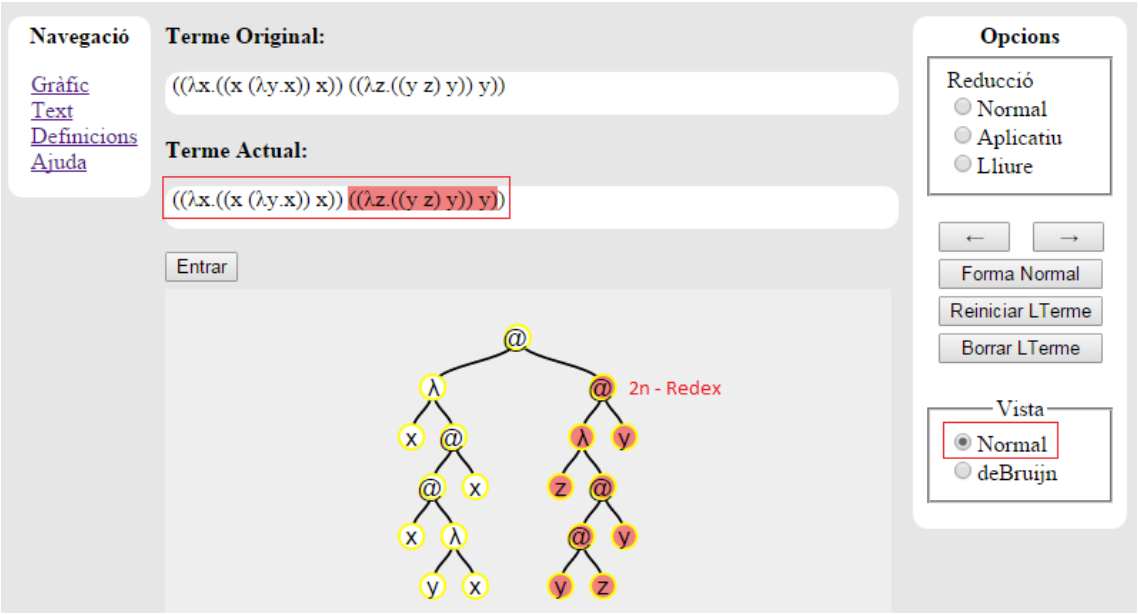


Vista d'un Lambda Terme en notació de DeBruijn

Podem marcar els redexs existents a l'arbre si aquest en conté prement sobre el node que és l'aplicació arrel (inici de redex) del terme. Si el tornem a prémer el que farem serà desmarcar-lo o bé si premem sobre un node que no és inici de redex si no hi ha cap redex marcat no passarà res, altrament el desmarcarem. Aquest canvi quedarà reflexat sobre l'estructura d'arbre i sobre la cadena de text que representa el terme actual. En aquestes dues captures de pantalla es mostren els dos termes que té l'arbre:



Redex marcat dins l'arbre que representa el Lambda Terme

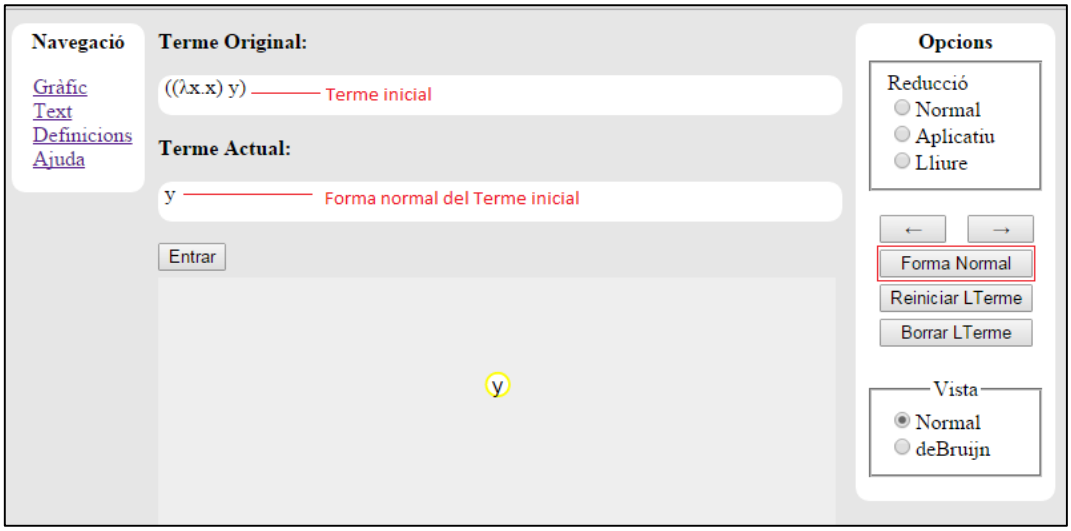


Redex marcat dins l'arbre que representa el Lambda Terme

Nota: Que hàgim parlat de "1r"i "2n" redex és totalment indiferent. No existeix ordre de numeració dels mateixos, simplement és una manera d'anomenar-los.

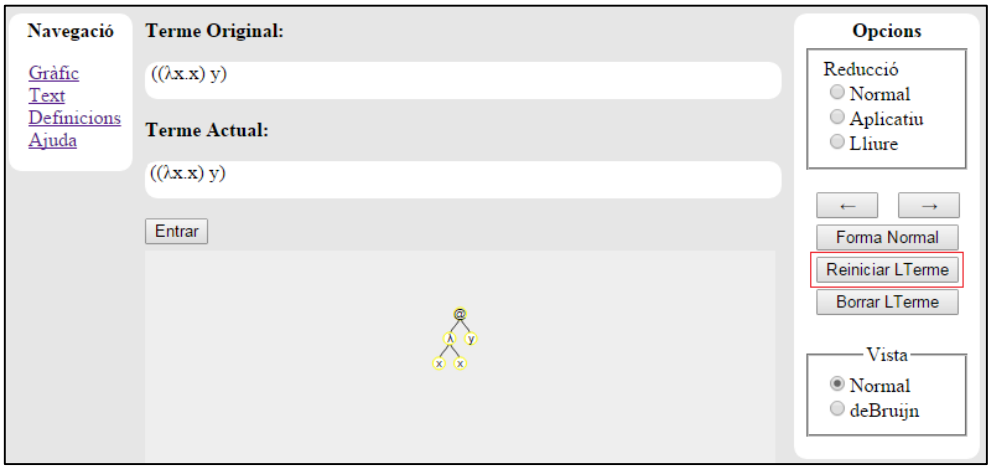
Seguim ara comprovant que els botons funcionen de la manera correcta. Per fer-ho, usarem un altre terme, el terme ((λx.x)y), que conté un únic redex i la seva forma normal és y. Provarem el botó de Forma Normal. Si donat un Lambda Terme el qual té forma normal, aquest botó ens la donarà directament, per fer-ho, l'aplicació redueix completament el terme amb el mètode de reduir normal fins que no hi hagi més redex dins el terme, per assegurar-se que la troba. Cal comentar que ho farà sigui el pas de reducció que estiguem fent. A l'hora de recular llavors, però, no es tindran en compte els passos intermedis. És a dir, si tenim un terme que a través de 5 passos de reducció en podem obtenir la forma normal i al 2n pas premem aquest botó, l'històric de reduccions constarà únicament de 3 passos, els 2 primers ordres i el pas del 2n a Forma Normal. Si es dona el cas que es fa a la primera iteració, el pas de recular durà a l'inici. El que es mostra a la part del terme anterior en aquest cas és el terme original, perquè l'usuari pugui veure des d'on s'ha partit.

Així doncs, pel que hem comentat, ((λx.x)y) ---FN--> y. Comprovem-ho:



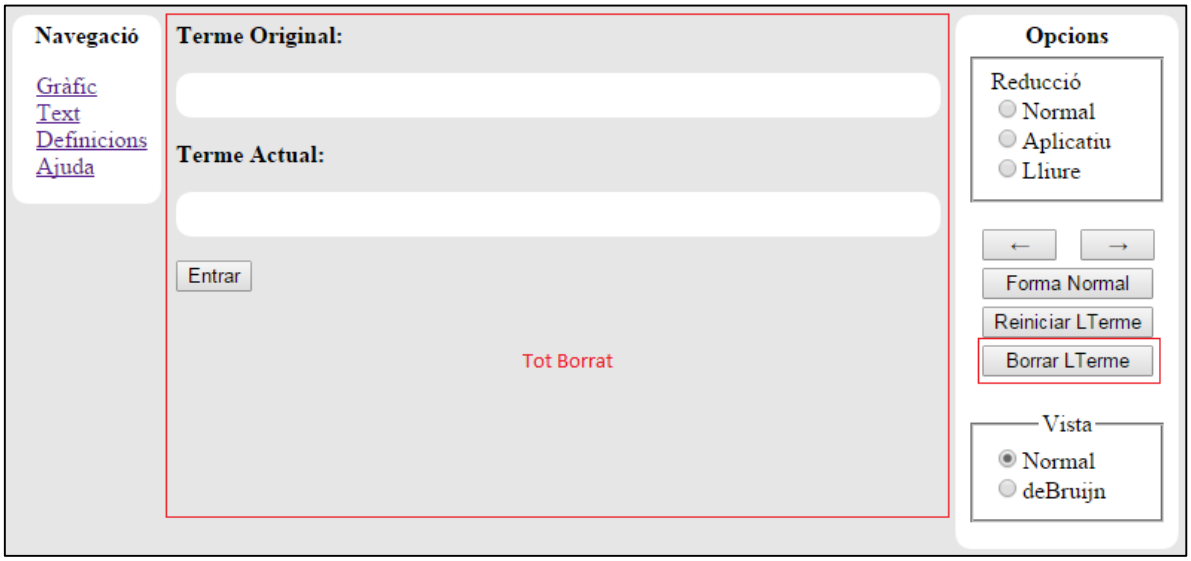
Forma Normal del terme ((λx.x)y) obtinguda mitjançant el botó de "Forma Normal"

Si premem el botó reiniciar en aquest moment, o en qualsevol moment de la reducció d'un Lambda Terme, obtindrem el terme entrat inicialment i tot l'històric quedarà eliminat. Si just en el punt de la imatge anterior premem el botó "Reiniciar LTerme", doncs ens quedarà de nou el terme entrat inicialment.



Botó de Reiniciar Lambda Terme al menú d'opcions del TilcW Gràfic

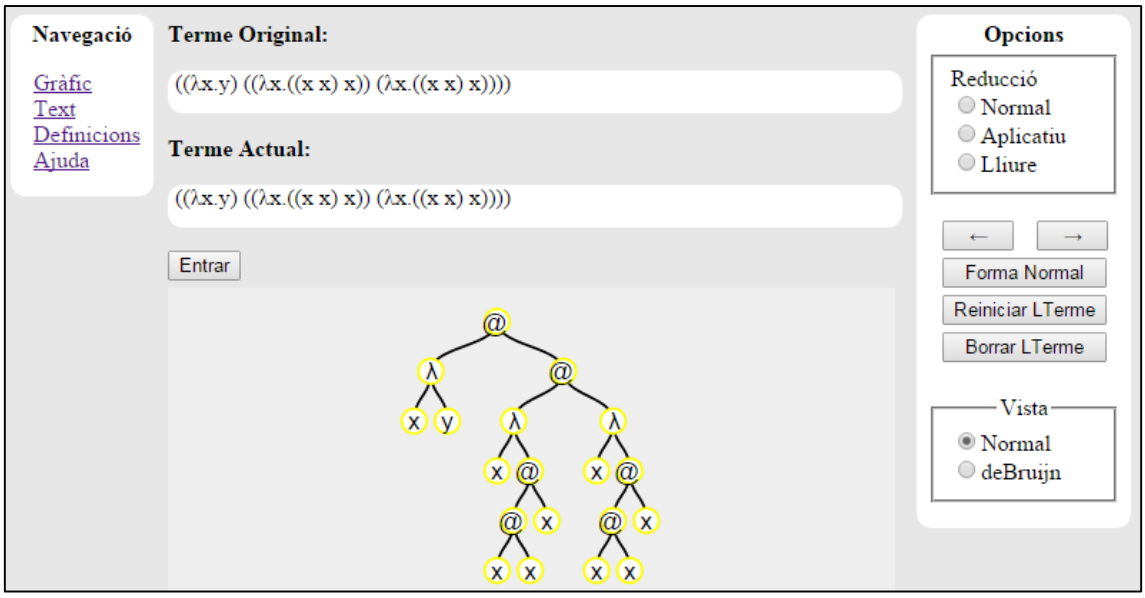
Si el que interessa és començar de nou, amb un altre Lambda Terme, tenim l'opció d'eliminar-lo. Per contra al botó de "Reset LTerme" el botó "Borrar LTerme" elimina completament tot el Lambda Terme, el seu arbre associat i la sessió del navegador, perquè al canviar de pestanya no carregui el Lambda Terme que s'ha esborrat. El seu aspecte, després d'eliminar el terme anterior, és el següent:



Pestanya TilcW Gràfic després de prémer el botó "Borrar LTerme"

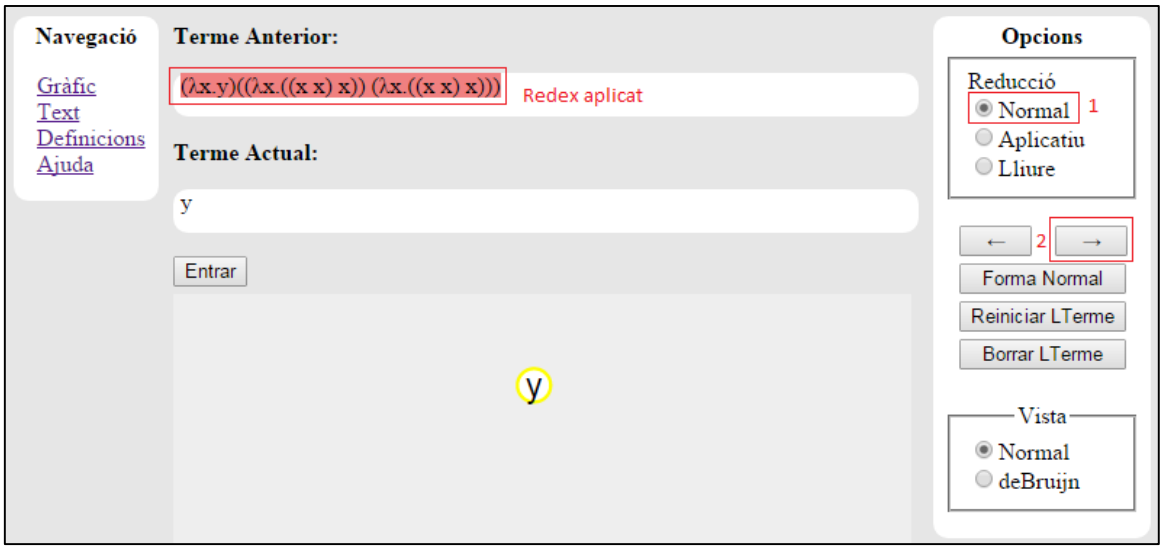
Per fer les proves de funcionament de reduccions per ordre normal i aplicatiu usarem un terme força usat a classe de paradigmes, ja que si es va reduint per ordre aplicatiu no es pot trobar mai la forma normal, mentre que aquesta apareix si es fa un sol pas de reducció normal. El terme és:

((λx.y) ((λx.((x x) x)) (λx.((x x) x))))



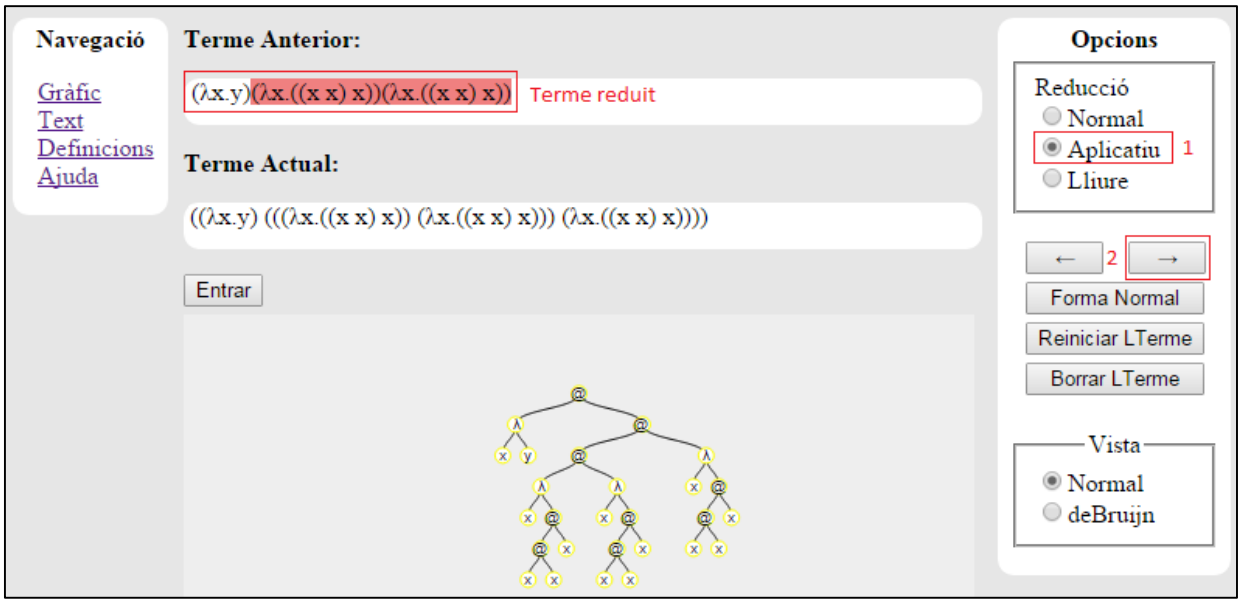
Lambda Terme proposat per explicar les proves de la reducció per odre nomal i aplicatiu

Si fem ara una aplicació per odre de reducció normal, ens ha de quedar el terme y, i a la part superior, la reducció que s'ha aplicat:



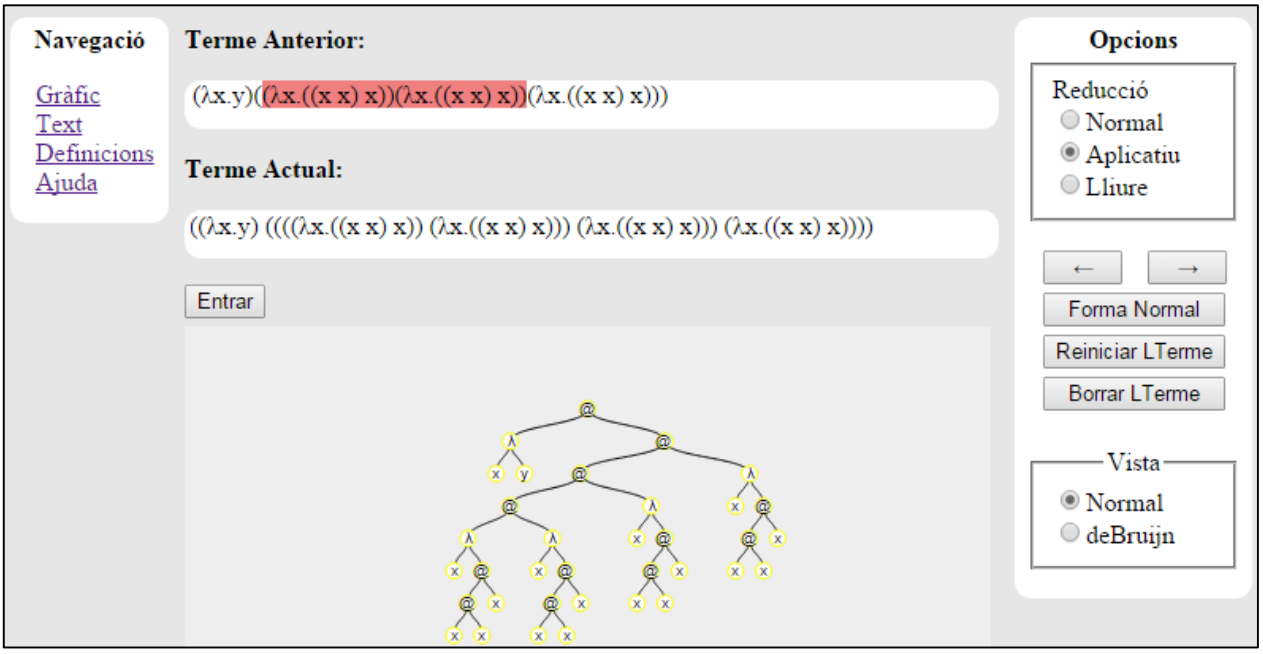
Resultat d'aplicar una reducció per ordre normal al Lambda Terme Proposat

Fem ara del mateix terme 2 reduccions per ordre aplicatiu. Aquest terme, si es redueix per ordre aplicatiu, es va copiant l'última part del mateix. Per tant, fent-ho així no aconseguirem mai la forma normal. Vegem aquests dos passos:



Primera reducció per ordre aplicatiu del terme proposat

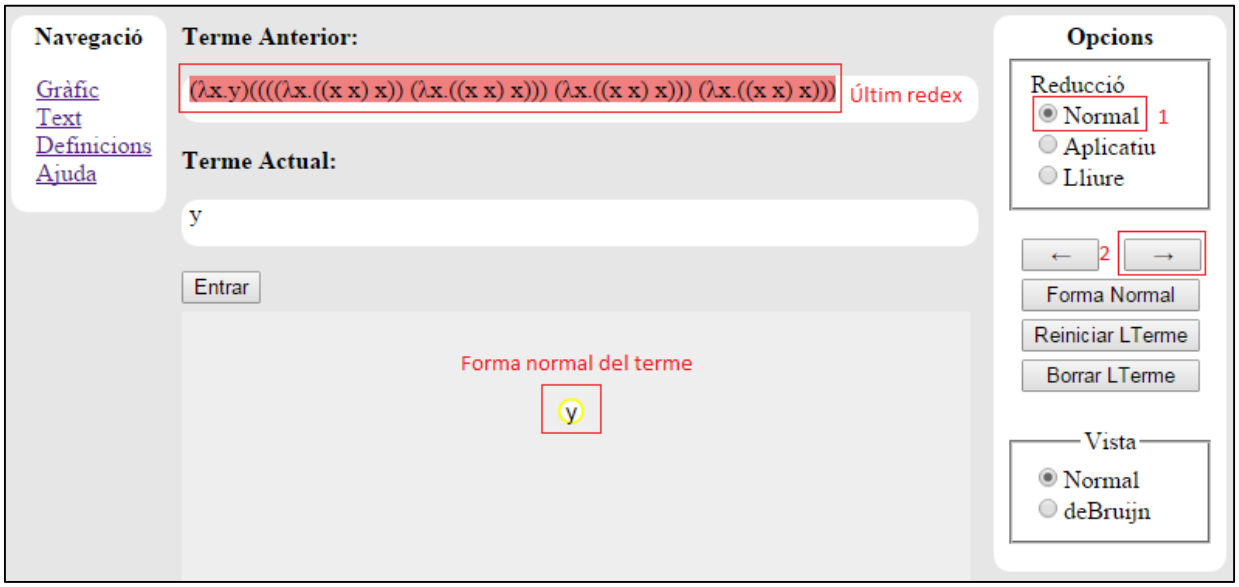
Fem ara el següent pas de reducció per ordre aplicatiu, i veiem com creix el terme i com es va indicant el terme anterior:



Segona reducció per ordre aplicatiu del terme proposat

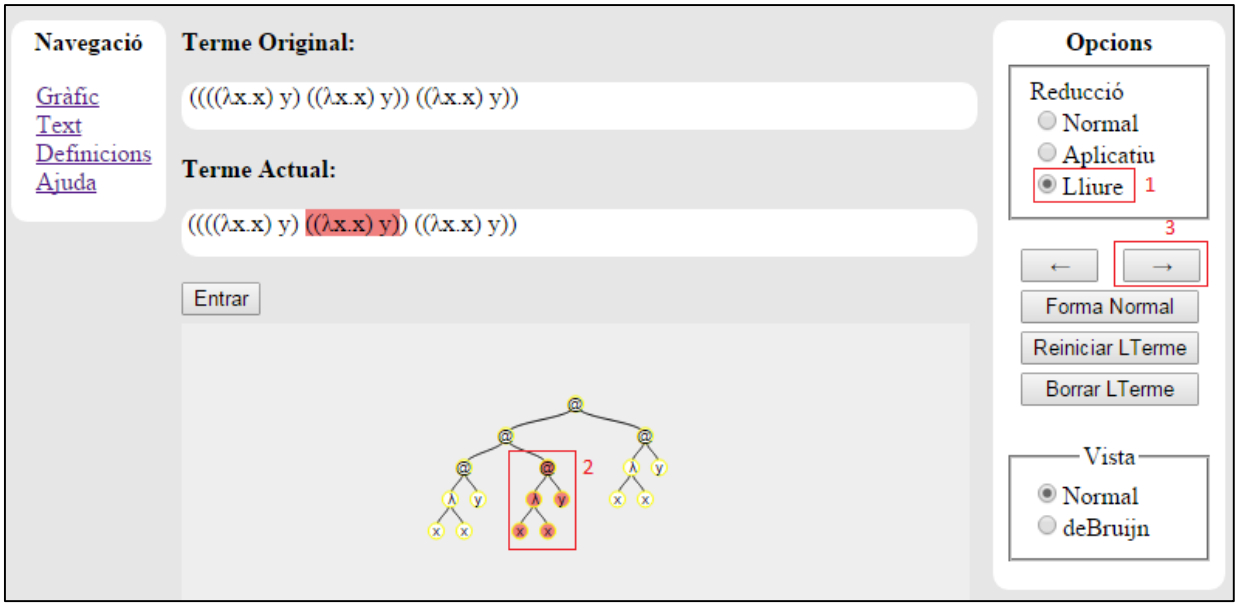
D'aquesta manera podríem anar seguint fins que la memòria s'acabés o l'arbre no hi cabés, ja que com veiem, no trobarem mai la forma normal aplicant aquest ordre de reducció.

Apliquem un ordre de reducció normal sobre aquest últim pas per veure que l'aplicació funciona correctament. Veiem que l'arbre queda reduït a la forma normal del terme (y), per tant doncs la commutació d'ordres de reducció és factible i funcional:



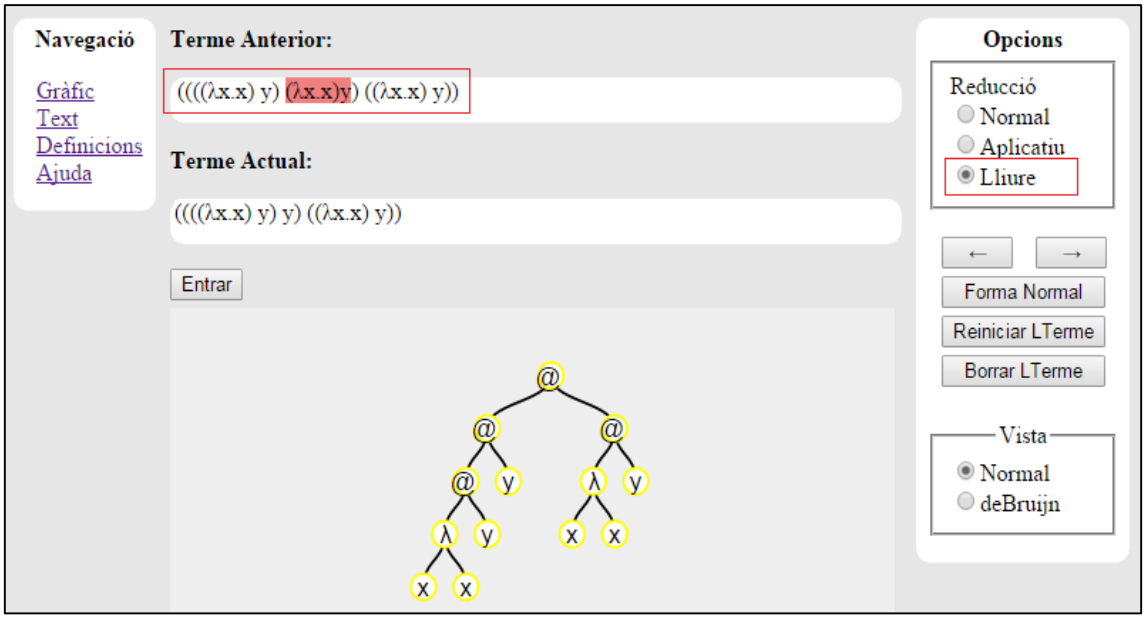
Per fer les proves de la reducció del TilcW Gràfic lliure usarem un que conté 3 redex, un que es pot reduir de manera aplicativa, un que es pot reduir de manera normal i finalment un que, si s'usa una o altra, no es reduirà degut a que està al mig. És aquí on entra en joc l'opció de reduir de manera lliure. El terme proposat és un terme que conté 3 vegades seguides el mateix redex, i nosaltres marquem aquí el redex central:

((λx.x)y)((λx.x)y)((λx.x)y)



Lambda Terme amb 3 redex

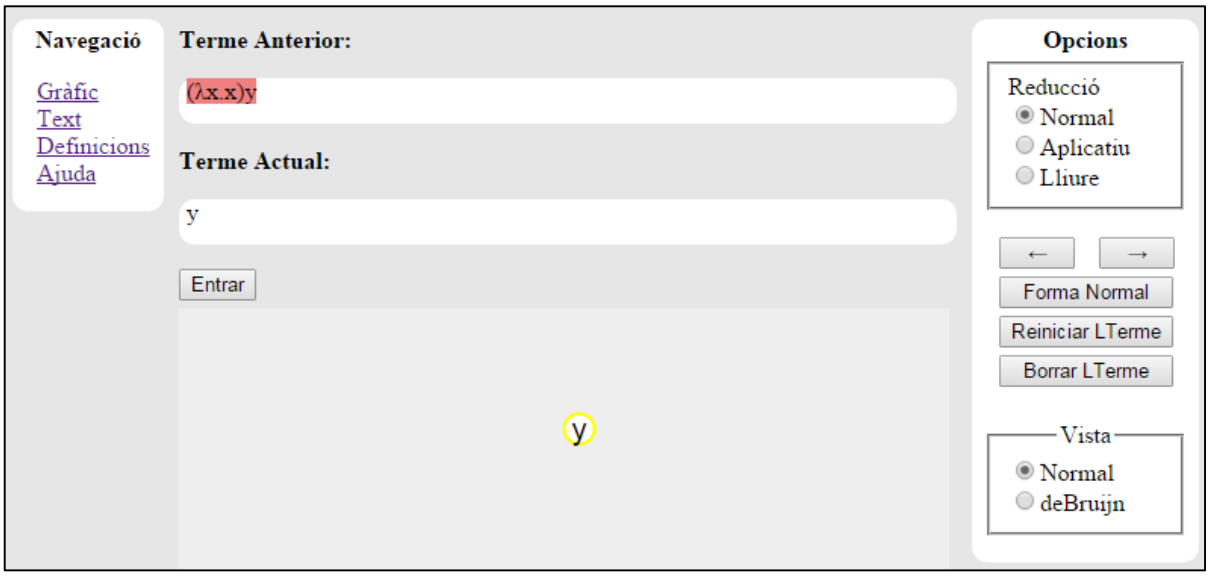
Veiem el resultat d'aplicar la reducció per el redex seleccionat d'aquest terme:



Resultat de l'aplicació de la reducció pel redex central del Lambda Terme proposat

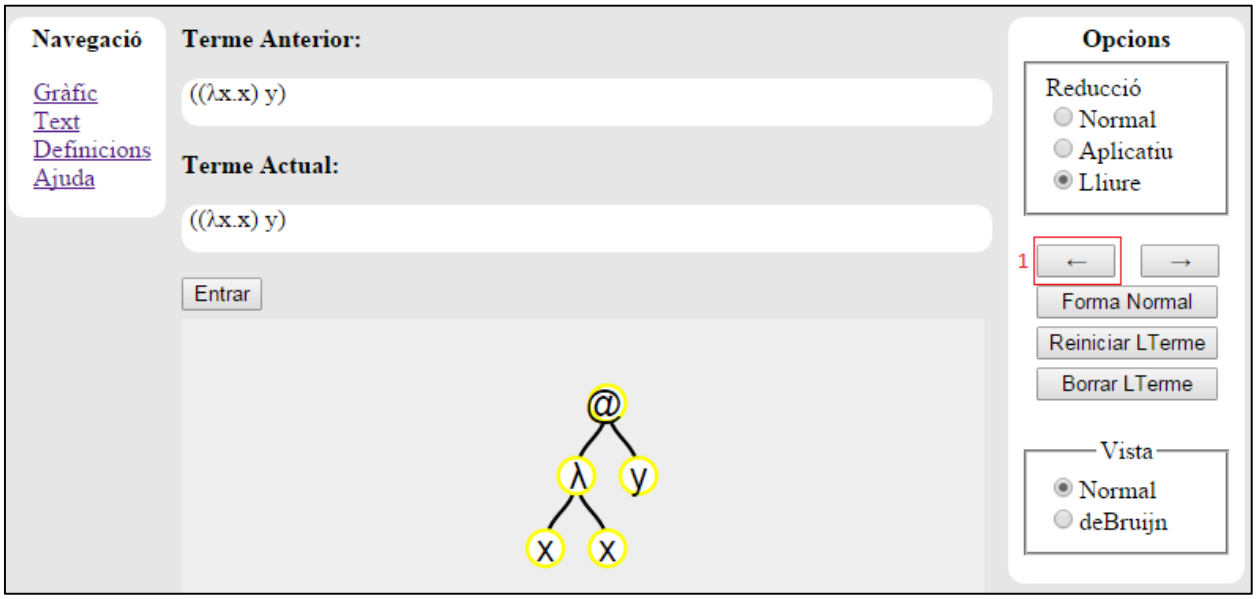
Un exemple del botó de recular seria el que es presentarà a continuació. El fet de recular consisteix en anar a buscar dins l'històric de reduccions l'índex anterior a l'actual. L'aplicació no permetrà la reducció si la reducció en aquest pas no era la mateixa que estigui activada al checkbox en aquell moment. Si no és el cas, avisarà a l'usuari quina és la que s'hi havia fet. Posarem un exemple de reducció:

((λx.x)y)



Terme proposat després del primer pas de reducció

Si ara reculem prement el botó corresponent, el terme tornarà a quedar com l'original, que en aquest cas és el terme que hi havia abans d'aplicar aquesta reducció:



Terme "y" després de recular un pas de reducció. Ha tornat a ser l'anterior ((λx.x)y)

Alertes disponibles mitjançant la llibreria Apprise:

En aquest apartat gràfic l'aplicació ens avisarà mitjançant "*alerts*" d'apprise si:

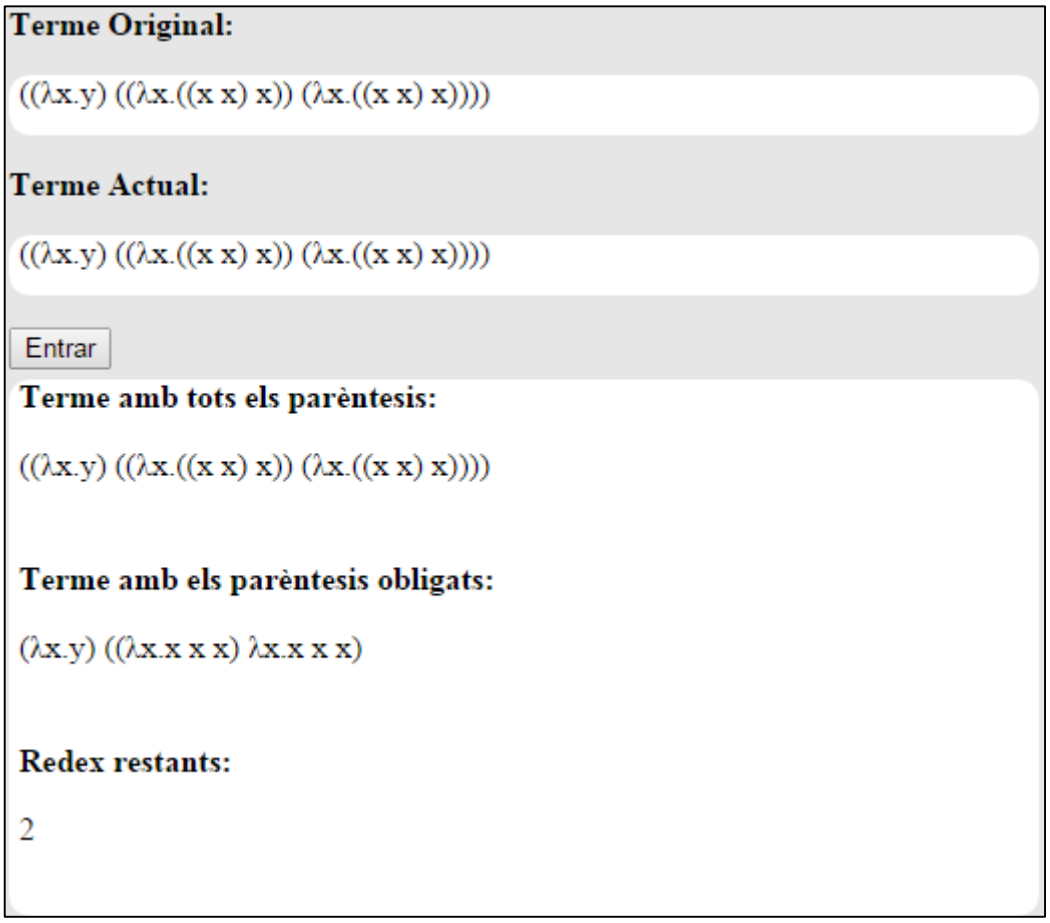
- Algun error de connexió entre la vista i el servidor. Normalment en aquests casos l'error s'arreglarà tornant a prémer el botó o amb la tecla F5.
- Es vol aplicar un pas de reducció si s'està mirant l'arbre en notació de deBruijn.
- S'està fent recular dins l'històric de Lambda Termes en un ordre de reducció diferent a l'aplicat en aquell pas. Estem parlant de casos com per exemple recular en ordre normal i s'ha reduït en aplicatiu. Apprise ja avisa de l'ordre usat en aquell punt.
- Es vol reduir de manera lliure però no s'ha especificat el redex. En aquesta versió de l'aplicació cal escollir el redex marcant-lo a l'arbre abans de prémer el boto d'avançar.
- El terme ja està en forma normal i es vol continuar reduint.

Proves de funcionament al TilcW Text

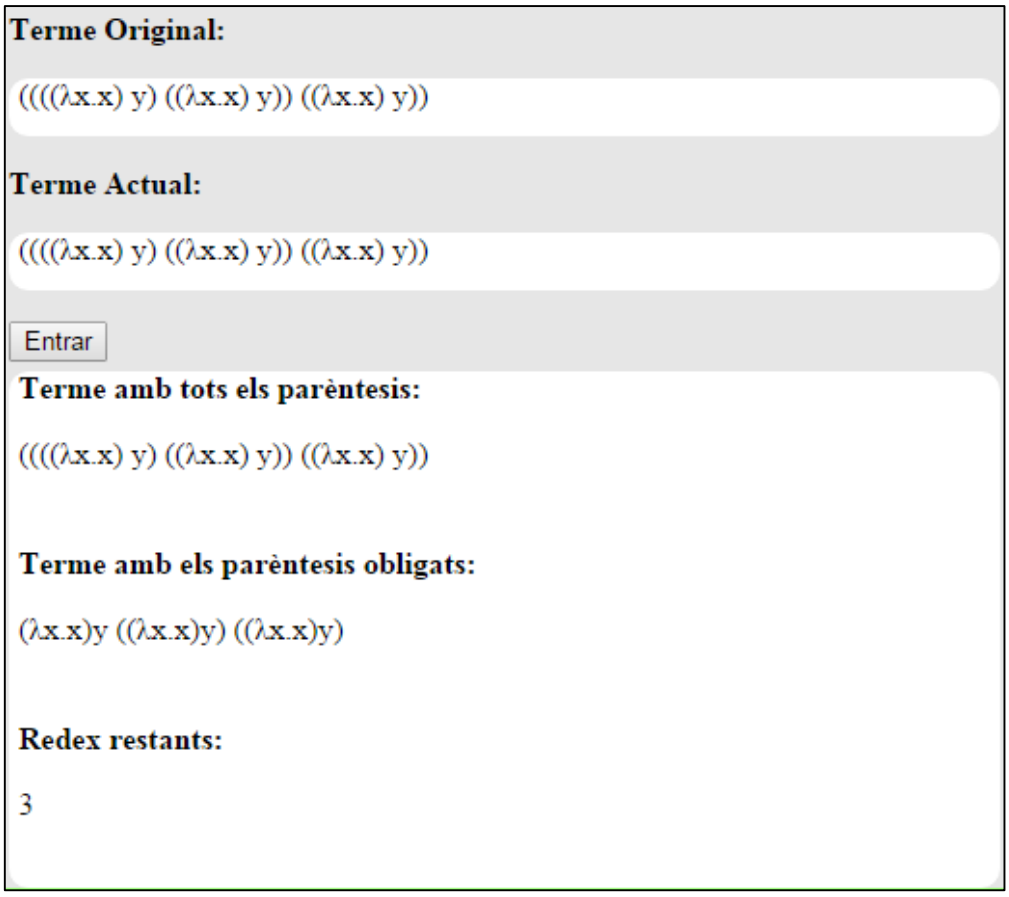
Les diferències entre el TilcW Gràfic i el Text són que a l'hora de marcar el redex aplicat, aquest no es mostra mitjançant un background color, sinó que queda subratllat i en cursiva sobre a on s'ha aplicat. Com hem dit, en aquest cas es mostra en cada pas del terme tots els parèntesis o només els obligats i el nombre de redex que queden dins el terme.

També es permeten les reduccions normals i aplicatives, però no en forma lliure. Si s'ha aplicat una reducció d'ordre lliure mitjançant el TilcW Gràfic i es recula, el que passarà serà que és que s'avisarà que es salta i es donarà el terme resultant del pas. Com que al avançar es sobreescriuen com hem explicat a l'apartat anterior aquests passos, no hi haurà cap més problema.

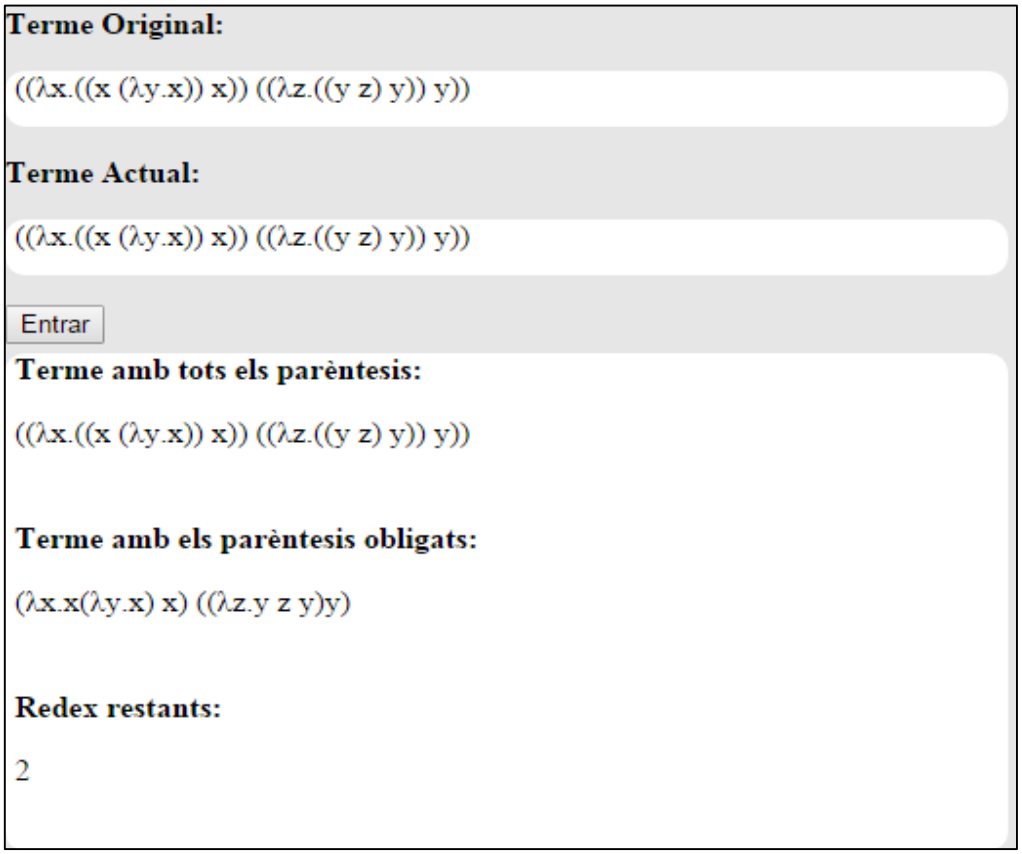
Apart d'això, ara mostrarem els termes d'exemple de l'apartat anterior per mostrar com funciona aquest apartat i finalment una reducció, per mostrar també la notació emprada per marcar el redex que s'ha aplicat al terme al avançar.



Primer Terme d'exemple per a l'apartat TilcW Text

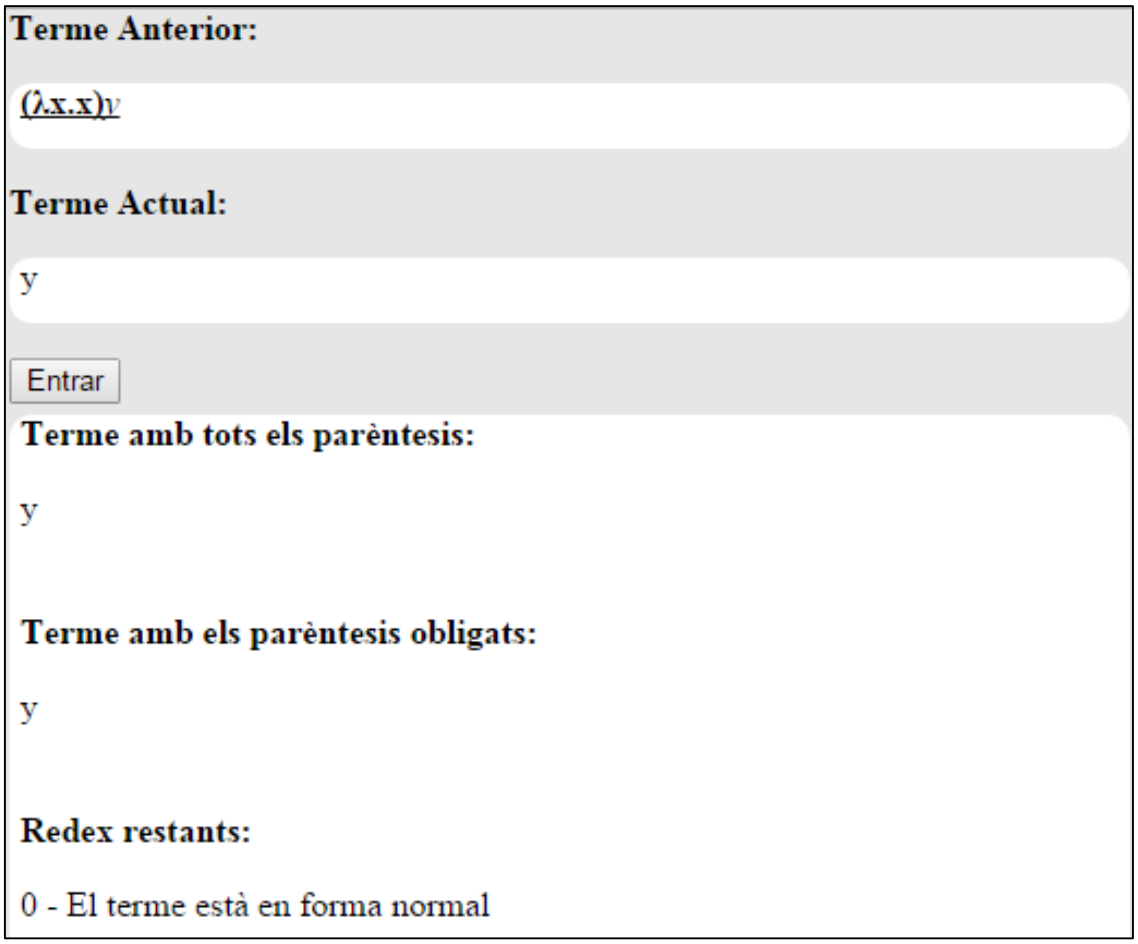


Segon terme d'exemple per a l'apartat TilcW Text



Tercer i últim terme d'exemple per a l'apartat TilcW Text

Ara veurem com és la notació del terme anterior en aquest apartat del TilcW.



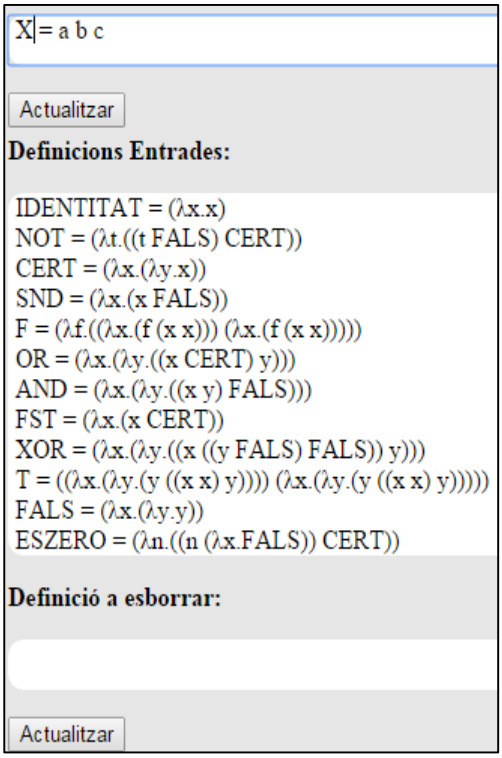
Vista d'un Lambda Terme reduït

Les alertes de la llibreria Apprise estan programades perquè saltin en els mateixos casos que al TilcW Gràfic però amb l'excepció de que en aquest cas avisarà que estem reculant per "sobre" d'un ordre de reducció lliure aplicat en l'apartat gràfic.

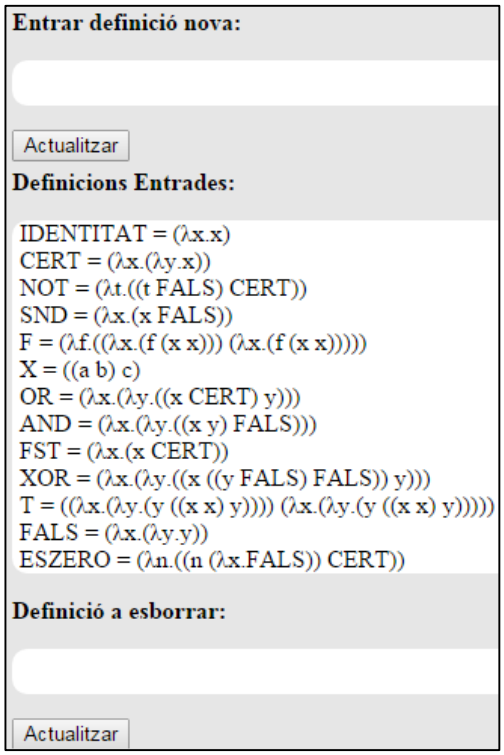
Proves de Funcionament al TilcW Definicions

Per comprovar el correcte funcionament de l'apartat de l'entrada de definicions, mostrarem com queda el mapa de definicions al entrar-ne una de nova, al actualitzar-ne una o a l'eliminar-ne una altra.

Començarem mostrant com s'entra correctament una definició nova:

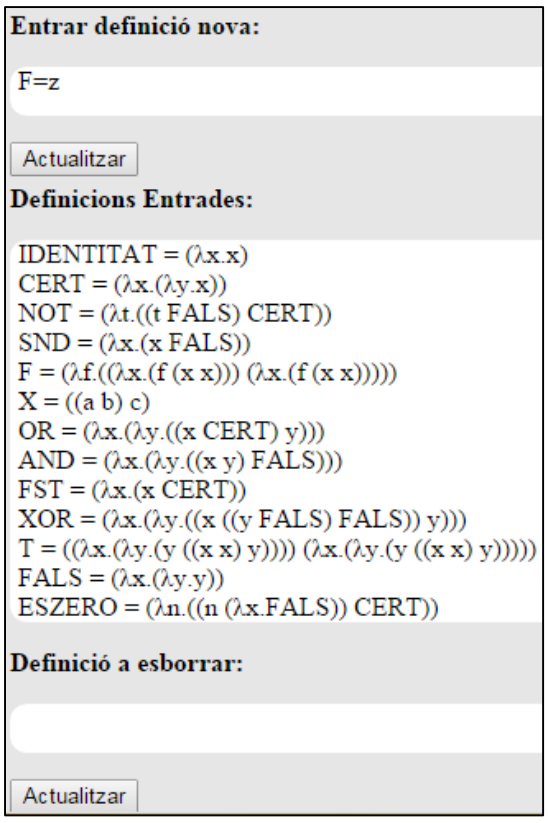


Definicions abans d'introduir la definició X

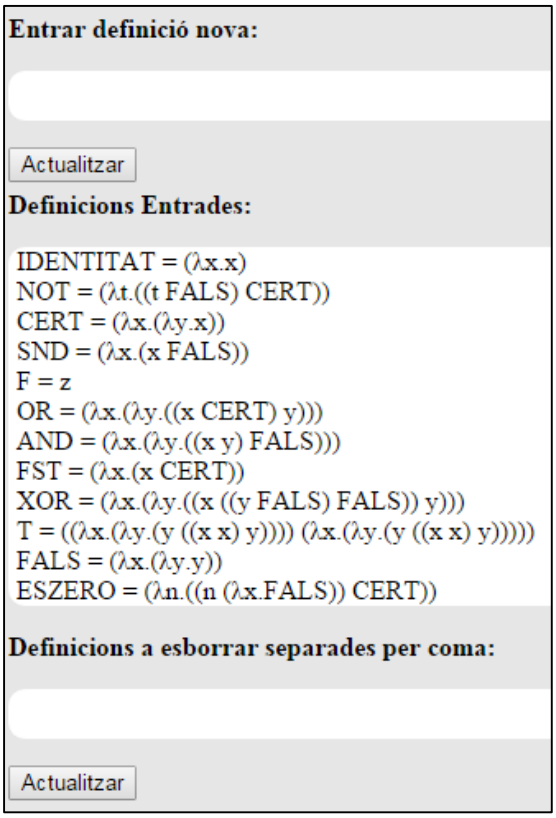


Definicions un cop entrada la definició X

Ara el que farem serà actualitzar una definició ja entrada anteriorment. Podem veure el seu valor abans (F) i el seu valor després:

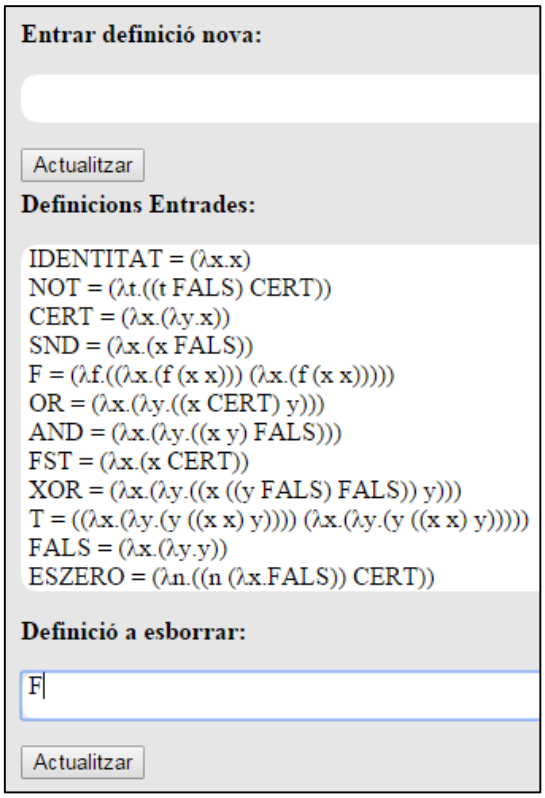


Actualització del valor de la definició Z (abans)

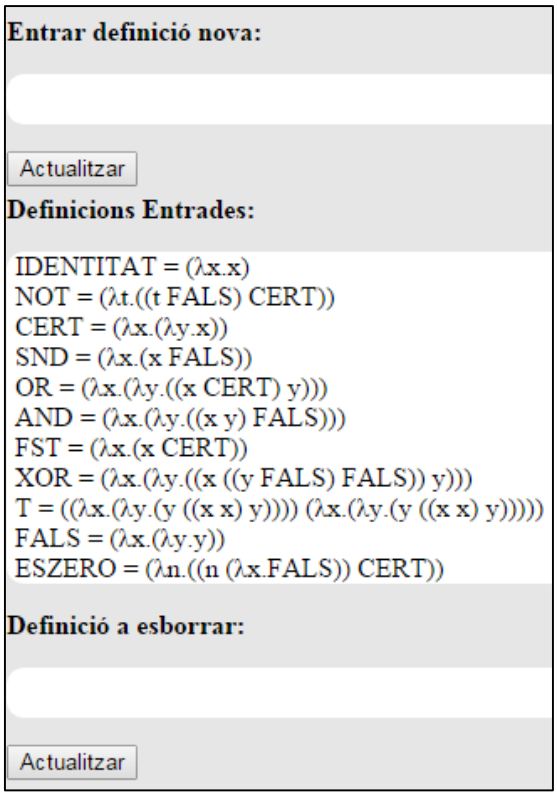


Actualització del valor de la definició Z (després)

Ara provarem d'eliminar una definició i veurem que un cop fet, desapareix del mapa de definicions entrades en l'aplicació:



Definicions entrades abans de borrar F



Definicions entrades després de borrar F

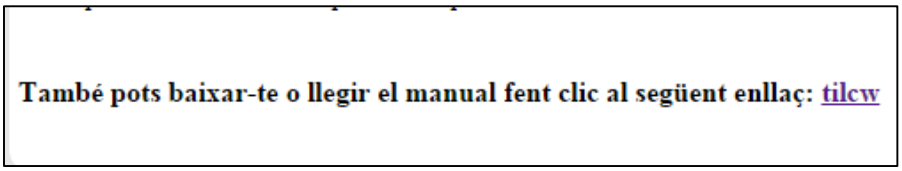
Els avisos que dóna l'Apprise en aquest apartat són:

- Errors al entrar/borrar/actualitzar alguna definició.
- Confirmació i nom de la definició entrada/borrada/actualitzada correctament.

Proves de Funcionament al TilcW Ajuda

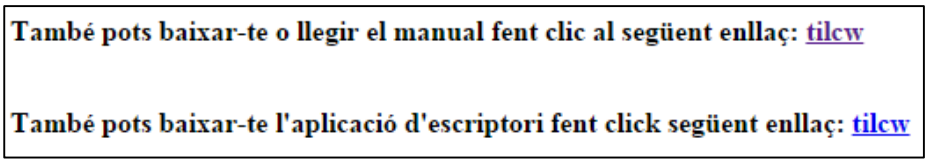
L'apartat d'ajuda no té més que un HTML estàtic on hi ha un petit element que si que varia depenent de si s'usa l'aplicació online o d'escriptori. Es tracta de l'últim paràgraf.

Si estem usant l'aplicació d'escriptori veurem:



Últim paràgraf de l'ajuda del TilcW ajuda en la versió d'escriptori

I si usem la versió online:



Últim paràgraf de l'ajuda del TilcW ajuda en la versió online

Això és degut a que la versió online ha de permetre baixar l'aplicació d'escriptori i el manual en pdf de la mateixa, mentre que la versió d'escriptori, evidentment, només ha de permetre baixar el manual.

8.1 Comentaris i retrospeccions dels companys que han provat el TilcW

Com s'ha comentat, hi ha hagut una sèrie de companys que han provat el TilcW i que, apart d'aportar elements per a l'apartat del treball futur, se'ls hi ha demanat que donessin una petita retrospecció sobre el mateix. Això és el que han dit:

Lourdes Boix Piqué:

"L'ús del TilcW m'ha proporcionat una molt bona manera d'estudiar. Tant amb la seva part gràfica, que et representa l'arbre i permet veure pas a pas com aquest va modificant-se a mesura que es va realitzant la reducció, fins trobar (o no) la forma normal; Com amb la seva part textual, on se t'indica entre d'altres com quedaria el terme introduït amb tots els parèntesis, permetent aprofundir i assegurar els coneixements adquirits pels alumnes durant les diverses classes.

Trobo que és una bona manera de seguir practicant, ja que et permet comprovar qualsevol tipus de fórmula o exercici que se t'acudeixi, exercici el qual podries no estar del tot segur de com resoldre'n la normalització, com col·locar-ne els parèntesis... I gràcies al programa, pots comprovar si la teva resposta anava o no anava ben encaminada.

A més a més, permet que s'utilitzi força intuïtivament, ja que tant els colors com l'estil són molt visuals i, per tant, crec que tothom pot aprofitar-lo i adquirir una bona base sòlida de coneixements de lambda-càlcul, arrelant a través del seu ús, els coneixements adquirits."

Rubén Calle Molina:

"El TilcW ha estat una eina de gran ajuda per resoldre dubtes i exercicis de la part de lambda càlcul corresponent a l'assignatura Paradigmes i Llenguatges de Programació de Tercer del Grau en Enginyeria Informàtica, la part gràfica és molt útil i sobretot molt visual, és realment ràpid d'escriure un lambda terme i veure'n la seva forma normal, validar si les reduccions que estem fent són correctes, així com en cas contrari i gràcies al seu "step-by-step" ens permet veure on ens hem equivocat. L'eina és molt intuïtiva, fàcil d'utilitzar tot i la seva complexitat interna, però com bé sabem és una de les tasques que comporta el software, fer que la complexitat quedi oculta als usuaris. En resum, és una eina útil i que en el futur pot adquirir noves funcionalitats que faran d'ella una eina encara més complerta."

9. Conclusions

Si tenim en compte els objectius plantejats a l'inici del treball i als que al final s'han assolit, es pot dir que el treball ha complert les expectatives inicials, ja que s'ha pogut creat el que en un principi es va proposar, una aplicació web per a què els usuaris que ho desitgin puguin estudiar Lambda-Càlcul d'una manera visual i gràfica que segurament és més agradable que fer-ho de la manera tradicional, mitjançant papers i exercicis.

Si bé és cert que hi ha hagut problemes i imprevistos durant la creació i el desenvolupament, no ha calgut modificar el guió inicial previst i el projecte s'ha pogut realitzar sense més entrebancs.

De totes maneres, si ens ho mirem d'una manera més constructiva, el que permet fer l'aplicació respecte al que podria ser, per part del client i el programador, és que s'haurien pogut implementar més elements i opcions (Comentades a l'apartat de "Treball Futur") però que degut a la limitació de temps i la impossibilitat d'avarca-ho tot per part del programador han limitat les possibilitats de l'eina. De totes maneres, s'ha mirat de deixar tot documentat i preparat de manera que si el mateix programador o algun altre estudiant en un futur no molt llunyà vol continuar la feina, aquesta tasca d'ampliació de funcionalitats no sigui extremadament difícil.

Referent a la realització del projecte, aquest no hagués estat possible si no s'haguessin cursat les assignatures de "Paradigmes i Llenguatges de programació" (Obligatòria) i la de "Programació declarativa. Aplicacions" (Optativa) on gràcies a la primera es va conèixer l'objecte d'estudi, el Lambda Càlcul, i amb la segona es va aprendre el llenguatge de programació base de l'aplicació, l'Scala. Dos conceptes que vaig trobar interessants i he tingut la possibilitat d'unir-los en un projecte que a més permetria a futurs estudiants fer més amè l'aprenentatge del Lambda Càlcul.

Això també ha estat possible gràcies a les assignatures "Metodologia i Tecnologia de la Programació I i II", amb la qual vaig aprendre a programar, "Estructures de Dades i Algorísmica" on vaig aprendre tècniques avançades de programar, com serien la immersió o un ús eficient de la recursivitat i també "Enginyeria del Software I i II" on van mostrar-me patrons de disseny, dels quals el Play n'usa 4.

Per tant, puc dir que el que he après durant la carrera m'ha servit per desenvolupar la totalitat del projecte i sense els conceptes apresos en diferents assignatures això no hagués estat possible.

10. Treball futur

En aquest apartat el que es mirarà és fer un llistat de possibles millores a l'eina, o ampliacions de la mateixa, ja sigui a elements que faltin que hagi requerit el client, a millorar que hagi vist el propi desenvolupador o bé suggerències fetes per els alumnes que han pogut testejar l'aplicació i usar-la.

No és que el TilcW no sigui funcional, si no que en una aplicació d'aquestes característiques, i amb un nivell de flexibilitat tan elevat, les possibles millores són moltes. Aquí se'n detallaran algunes.

La idea serà que, per cada millora o fet es farà una comparativa sobre com és actualment i la proposta de millora, així a més es pot fer una valoració de les millores. El llistat no està organitzat per prioritats, si no que són millores que han sorgit i el client i el programador han anat anotant. Algunes estan referides a l'aplicació en general i altres al lambda terme (tant de l'apartat gràfic i text) concretament. En aquest apartat s'indicaran els trets bàsics per a millorar-la.

Referents a l'aplicació:

Multi llenguatge

Com a requeriment extra, l'aplicació es pot fer multi llenguatge.

- Consideracions: És possible fer-ho degut a que al tractar-se d'una aplicació web els temes d'internacionalització estan força avançats en la majoria d'aplicacions i el Play 2.0 no és una excepció. El cas és però que, per a cada llenguatge s'ha d'escriure el text de cada botó, menú, ajuda, etc de manera que tots els elements estiguin traduïts i, a l'hora d'executar-se en funció de la selecció d'idioma s'ha de fer que es mostri una altra. Així doncs, no és un requeriment excepcionalment difícil però les hores de treball per a fer-les són proporcionals al nombre de llenguatges a afegir així que es va decidir no afegir-lo com a requeriment principal.

Un últim apunt és que, al poder-se desenvolupar com a aplicació d'escriptori però que corre sobre navegador, la traducció multilingüe per a la versió d'escriptori ja estaria feta en el moment de passar l'aplicació a *standalone*.

Interfície

El fet de ser un únic programador i un client fa que l'elecció de colors i font de les lletres pugui resultar complicada a l'inici, i que tot depengui dels gustos o preferències del programador. Aquest fet implica que potser no a tothom li serveixi o li sigui accessible la interfície actual del TilcW, en quan a temàtica de color o font de text així que es podria optar o per buscar-ne una de més correcta.

NOTA: Les dos millores anteriors es poden agrupar sota un mateix paraigua, que, juntament amb altres elements que podrien requerir-ho podrien formar una pestanya nova, anomenada, per exemple “opcions” o “configuració” on l’usuari es pogués adaptar a gust l’aplicació.

Opcions Lambda Càlcul

En Lambda Càlcul existeixen altres operacions que no estan contemplades en el TilcW, com seria alfa conversió de termes, o fins i tot preguntar-se si un terme pot eta-reduir-se, el fet d’afegir un nou apartat amb aquestes opcions, podria ser una possible millora en un futur.

Algorisme de Hindley-Milner

No és una millora estricament necessària, però si molt útil ja que a la branca de Computació el professor Mateu Villaret presenta el Lambda Càlcul Tipat, així com l’algorisme de Hindley-Milner, que permet tipar el Lambda-Càlcul.

- Consideracions: Aquest apartat hauria de ser independent a l’arbre “Gràfic” i “Text”, hauria de permetre entrar un Lambda Terme, mostrar-lo en forma d’arbre i finalment mirar de determinar el tipus de les variables. Aquesta millora és costosa en un principi perquè implicaria la creació d’un nou element dins el Model “LTerme”, afegint que hi poden haver Variables i Abstraccions amb tipus. Degut a la complexitat que això comportaria, no s’ha decidit incloure aquest apartat en el projecte inicial.

Referents al Lambda Terme:

En aquest apartat són moltes les millores que es poden fer, i les dividiren en millores pròpies del Lambda Terme que comparteixen tant el terme de l’apartat Text com el terme de l’apartat Gràfic.

Variables lliures i lligades

Tot Lambda Terme conté variables lliures i lligades, i tot hi veure el terme descompost en forma d’arbre, si el Lambda Terme és molt llarg o enrevessat, es pot fer difícil de veure.

- Consideració: És podrien fer dos botons que marquessin quines variables estan lligades i quines estan lliures, respectivament.

Definicions

Tot i dir que no hi havia ordre a la llista de funcionalitats pendents a fer i cap funció prioritzava a la resta, aquesta potser ho hauria de ser ja que no s’ha pogut acabar d’implementar correctament.

- Consideracions: Donat un arbre amb definicions al mateix, aquestes, que per defecte representen variables, s’haurien de poder desplegar ja que, al desplegar definicions poden aparèixer nous redex dins el lambda terme. Per posar-ne un exemple, donarem aquestes tres definicions:

  CERT=λx.λy.x
  FALS=λx.λy.y
  NOT=λt.t FALS CERT

  FALS i CERT com a variables no representen res, però si es despleguen seguides una rera l’altra, apareixen nous redexs. Aquesta part és important, i per això es demana que les variables corresponents a definicions s’entrin en majúscula ja que tot i mostrar-

se com a variables, el programa internament sap que s'estan referint a definicions i a més a més l'usuari pot identificar-les més fàcilment.

Colorpicker

Tot i no ser una millora que millori massa el comportament de l'aplicació, es podria afegir un *"colorpicker",* és a dir, una paleta de colors dinàmica per escollir el color dels nodes de l'arbre, el color dels nodes que tindran les variables lligades o les lliures, si es marca un únic redex el color del mateix, etc.

- Consideració: Existeixen llibreries javascript que permeten implementar aquest element, amb diferents maneres de visualització. Estudiar quina d'aquestes és la més òptima a implementar per a l'aplicació podria ser una bona manera d'implementar la millora.

Marcatge de tots els redex

Una altra opció no disponible actualment és el fet de que tots els redex dins el terme es puguin mostrar al mateix moment, a l'arbre i a la cadena de text. Pot semblar una millora en si fàcil d'implementar, però el muntatge actual de l'arbre no la fa fàcil, així doncs, per fer-la, abans s'haurien de canviar trossos importants de codi per a poder veure-les correctament.

Visió d'un redex

Actualment, el marcatge dels redex a l'arbre es fan d'un en un, és a dir, si es prem un node que és l'inici d'un redex, aquest queda marcat sobre l'arbre. El que es marca realment són els nodes de l'arbre, que canvien de color.

- Consideració: enlloc de marcar els nodes de l'arbre marcar amb un rectangle el mateix, sense el canvi de colors dels nodes, sinó que aquests quedessin emmarcats.

Observacions vistes pels companys que han provat l'aplicació

Els companys/es que han provat l'aplicació s'han trobat algun problema a l'hora de visualitzar Lambda Termes amb el sentit de que, per exemple, un terme que al reduir mitjançant la forma aplicativa el terme es replica, o bé un terme que enlloc de contraure's només s'expandeix. Proposen que per a aquests casos, o altres de similars, si n'hi ha, l'usuari pugui saber-o d'alguna manera ja que l'aplicació no dona error si cada cop tenim més redex i no menys, ja que no és un error com a tal, però per algú que aprèn Lambda Càlcul pot ser complicat d'assimilar.

- Consideració: fer certes comprovacions condicionals a l'hora d'avançar o prémer el botó de FN i fer els "alerts" corresponents amb l'Apprise (recordem-ho, la llibreria que ens permet llançar els missatges d'error o confirmació quan cal) seria una manera de aplicar aquesta millora.

Finalment s'exposen una sèrie de millores que es podrien aplicar sobre l'arbre, la majoria de les quals són idees extretes de l'últim Tilc, fet per David Ruiz al 2010.

Menú contextual

Al prémer un node amb el botó dret s'hauria de poder desplegar un petit menú amb opcions, com ara "marcar", "desplegar", etc.

- Consideració: Existeixen llibreries javascript que permeten fer-ho, però la complicació de aplicar-les i annexar-les amb la construcció actual de l'arbre és més aviat costosa i no és viable de cara al projecte actual.

Navegació de l'arbre

Altres millores possibles a l'arbre serien que al prémer un node amb el botó dret aquest es pogués estirar cap a una altra part (com es pot fer amb l'arrel), per si l'usuari vol tractar una part del Lambda Terme per separat, enlloc del conjunt sencer.

També afegir 4 botons direccionals a l'apartat del menú per moure'ns pel canvas, i no únicament amb el ratolí podrien facilitar la navegació sobre el mateix.

Resum:

Com hem vist, l'aplicació tot i estar funcional, conté encara un conjunt molt ampli de millores i/o correccions, que es poden aplicar en futurs PFG o bé pot seguir el programador desenvolupant pel seu compte per al client, en l'ordre que aquest decideix organitzar-les.

11. Bibliografia

Hankin, C. (2004). *An introduction to Lambda Calculi for a Computer Scientists* (2a ed.). Strand: King's College London Publications.

Hilton, P., Bakker, E., Canedo, F. (2013). *Play for Scala* (1a ed.). Shelter Island, NY: Manning Publications Co.

Petrella, A. (2013). *Learning Play! Framework 2* (1a ed.). Birmingham: Packt Publishing.

Odersky,M. (2014). *Scala By Example*. Recuperat des de

http://www.scala-lang.org/docu/files/ScalaByExample.pdf

Villaret Ausellé, M. (2012). *Làmbda-Calcul-Breu* [Apunts acadèmics]. UdGMoodle.

Ruiz i Franco, D. (2010). *Tracejador Interactiu de Lambda-Calcul amb tipus (Tilc(T))*

Griba2010. (2006). *Javascript*. Recuperat des de

https://ca.wikipedia.org/wiki/JavaScript

Yug. (2012). *D3.js*. Recuperat des de

https://en.wikipedia.org/wiki/D3.js

MindMatrix. (2012). *JSON.* Recuperat des de

https://en.wikipedia.org/wiki/JSON

Sfermigier. (2010). *SBT.* Recuperat des de

https://en.wikipedia.org/wiki/SBT

Griba2010. (2006). *Scala*. Recuperat des de

https://ca.wikipedia.org/wiki/Scala_(llenguatge_de_programació)

Ysidoro. (2013). *Heroku*. Recuperat des de

http://es.wikipedia.org/wiki/Heroku

Anònim. (2015). *The DeBruijn Index*. Recuperat des de

https://en.m.wikipedia.org/wiki/De_Bruijn_index

Panphilos. (2006). *Ajax (programació)*. Recuperat des de

https://ca.wikipedia.org/wiki/Ajax_(programació)

Bort,G. (2007). *Play Framework* . Recuperat des de

https://www.playframework.com/

Varis. (1999). *THE WORLD'S LARGEST WEB DEVELOPER SITE*. Recuperat des de

http://www.w3schools.com/

12. Manual d'usuari i/o instal·lació

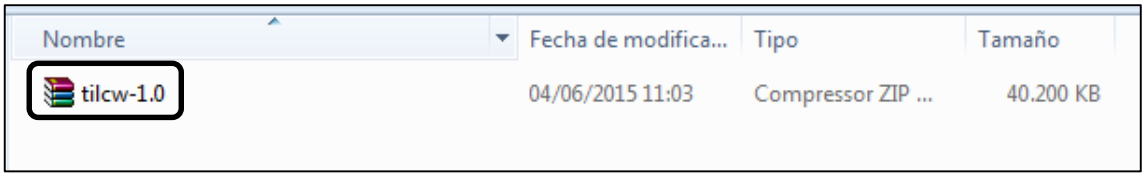
En aquest apart es pretén fer una explicació de com funciona l'aplicació desenvolupada i què fa cada element de la mateixa.

Com que l'aplicació es pot usar on-line i en versió d'escriptori, explicarem primerament com permetre que funcioni mitjançant la versió d'escriptori. Tot i que l'aplicació serveix tant per entorns Windows i Linux es posarà d'exemple com executar-ho en un entorn Windows puntualitzant en cada coment si algun pas difereix ja que la majoria són els mateixos.

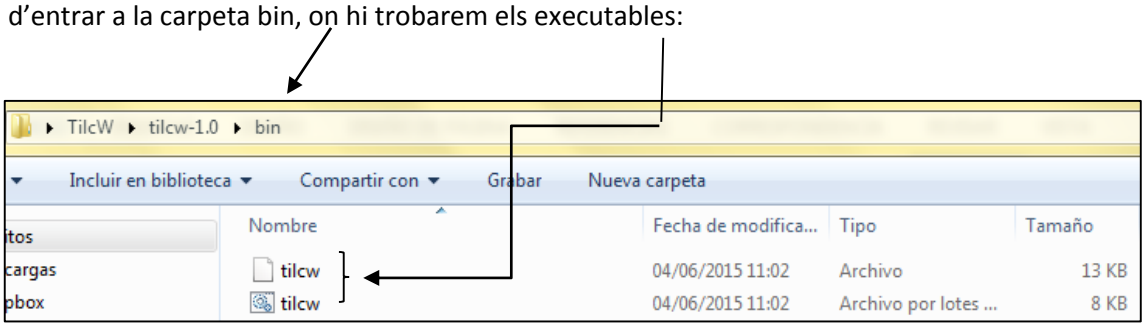
La versió d'escriptori pot trobar-se prement el link de l'última línia a la següent direcció:

http://tilcw.herokuapp.com/help

El primer que s'haurà de fer és descomprimir l'arxiu a la carpeta on es desitgi, on hi ha el TilcW comprimit.

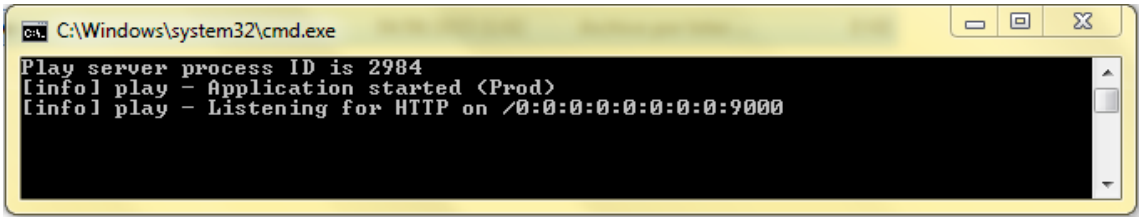


Un cop fet, dins la carpeta tilcw-X hi trobarem un seguit de carpetes i arxius, nosaltres hem d'entrar a la carpeta bin, on hi trobarem els executables:

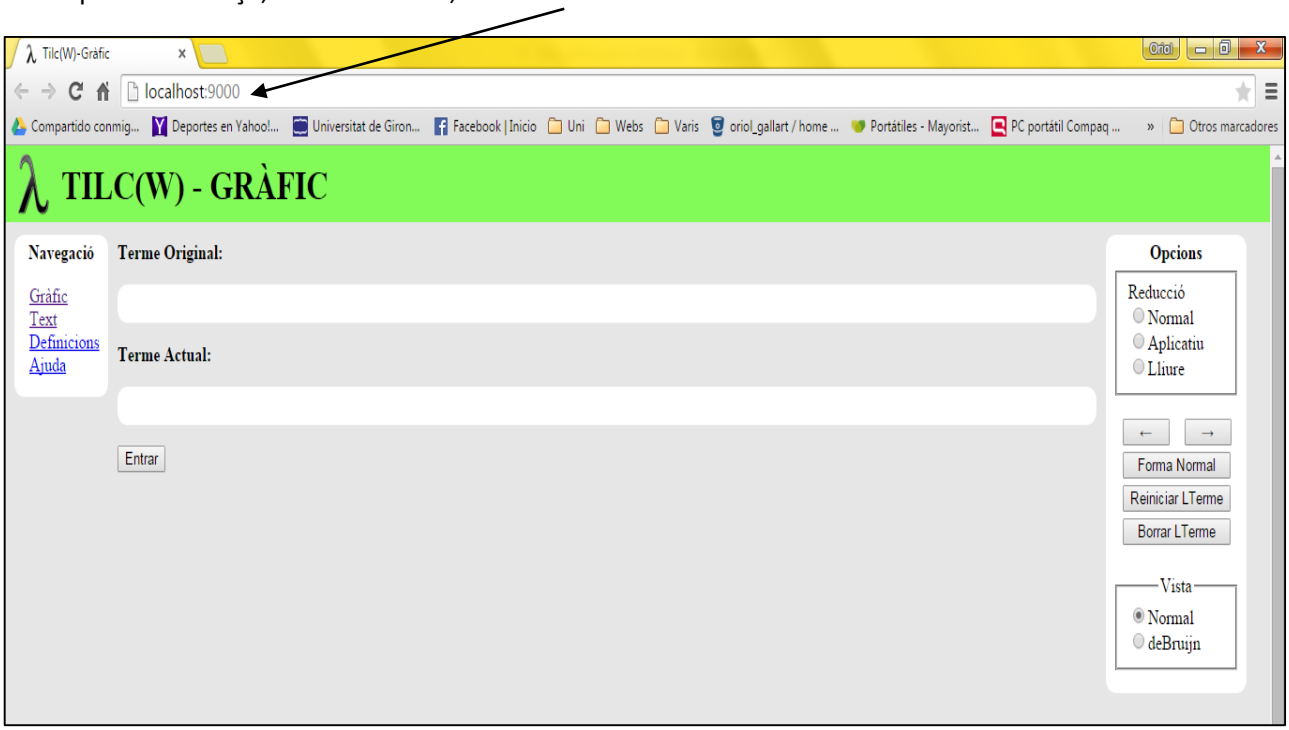


Nota: Per a sistemes Windows haurem d'executar el fitxer .bat, i per a Linux, donar permisos d'execució al fitxer .sh i llavors ja el podrem executar.

Un cop fet això, s'obrirà la línia de comandes on al cap de pocs segons ens dirà que a **localhost:9000** ja hi ha el servidor escoltant. A partir d'aquest moment, ja es pot obrir qualsevol navegador disponible per part de l'usuari i entrar a l'adreça corresponent, on s'obrirà la pantalla principal del TilcW. Per a tancar l'aplicació haurem de tancar també la línia de comandes.



Aquí tenim una vista de si, un cop obert l'executable, anem al Google Chrome (per exemple) i posem l'adreça, *localhost:9000*, tal com hem comentat:



Nota: La pestanya per defecte és la de l'apartat gràfic, però es pot anar a qualssevol altra via direcció web.

Si per contra volem entrar a l'aplicació i disposem de connexió a internet podem fer-ho a la direcció *http://tilcw.herokuapp.com/*. Aquí adjunto una relació de direccions entre versions:

	Versió Online	Versió Escriptori
Gràfic	http://tilcw.herokuapp.com/	http://localhost:9000/
Text	http://tilcw.herokuapp.com/text	http://localhost:9000/text
Definicions	http://tilcw.herokuapp.com/def	http://localhost:9000/def
Ajuda	http://tilcw.herokuapp.com/help	http://localhost:9000/help

Tot seguit, un cop explicat com entrar o iniciar l'aplicació el que farem serà explicar cada element del web i la seva funcionalitat, tot i que les proves d'execució es poden veure als apartats 9 i 10 d'aquesta memòria, ja que aquí només s'explica l'ús, no s'exemplifica.

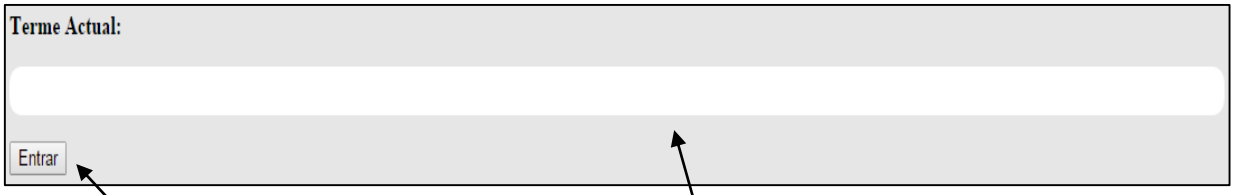
Es pot navegar per l'aplicació amb el menú persistent que hi ha a l'esquerra de l'aplicació, on se'ns redirigirà a cada un dels apartats corresponents (Gràfic, Text, Definicions o l'apartat d'ajuda)

Seguint l'ordre d'importància de l'aplicació, descriurem què fa cada apartat de manera breu i llavors les opcions que hi apareixen a cadascun d'ells.

Apartat Gràfic

Aquest apartat permet, un cop entrat un Lambda Terme, veure'n la seva estructura en forma d'arbre i interactuar amb la mateixa.

L'entrada del Lambda terme es fa mitjançant una àrea de text on es podrà entrar el Lambda Terme en format cadena de tex que, si és un text que realment representi un Lambda Terme, el servidor ens tornarà la serva estructura. L'àrea de text ve representada de la següent forma:



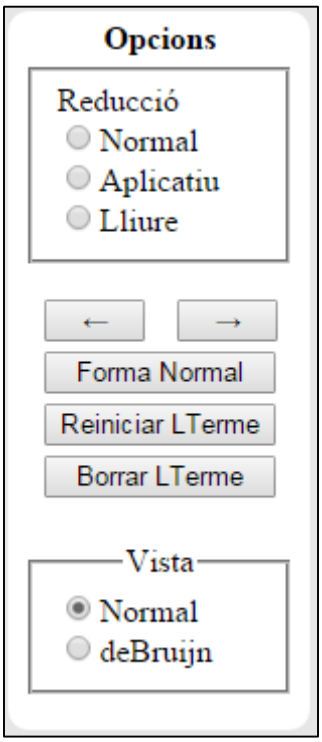
Enviar el LTerme al servidor

Àrea de text on escriure el Lambda Terme

La manera d'introduir un Lambda Terme en l'apartat Gràfic i el de Text, apart d'explicar-ho breument a l'ajuda del programa, s'exposa aquí:

- El valor de "Lambda" pot introduir-se amb els següents caràcters:
    - "λ", "\", "|", "/" per a permetre més agilitat a l'usuari.Per al punt en la notació d'una (λV.E), serveixen els caràcters "," i "."
- La introducció dels elements és independent del nombre d'espais entre elements. De totes maneres, per a una notació correcta, seria convenient només separar amb 1 espai cada element.
- Les variables poden contenir números i lletres, de mida indefinida per defecte. Recordem que:
    - Per ser un terme en notació normal, no cal cap restricció en el nom de les variables.
    - Per ser un terme en notació deBruijn, els noms de les variables han de poder ser parsejats a enter i una abstracció de la forma (λV.E) ha d'escriure's sense la V, com: (λ.E)

El menú d'opcions es troba a la dreta, i són els següents:



De dalt a baix, les funcionalitats són les següents:

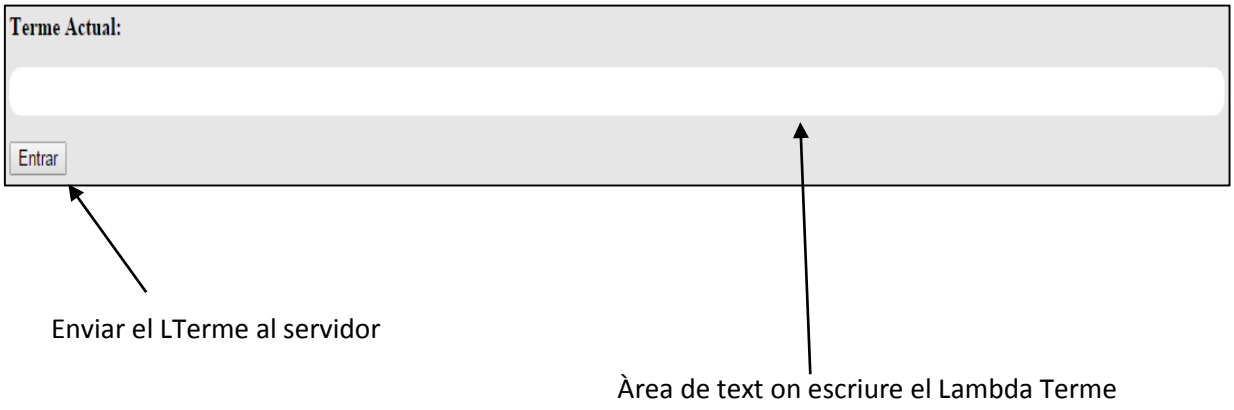
1. Elecció de l'ordre de reducció.
    a. Ordre de reducció normal: Permet, combinat amb el botó d'avançar(→), beta-reduir el terme mitjançant l'ordre de reducció normal si aquest no es troba en forma normal.
    b. Ordre de reducció aplicatiu: Si el terme no està ja en forma normal, seleccionant aquest element s'aplicarà una beta-reducció mitjançant l'ordre de reducció aplicatiu.
    c. Ordre de reducció "lliure": Es possible escollir, si hi ha més d'un redex, quin es vol reduir. Per fer-ho, s'ha de marcar el redex sobre l'arbre i llavors aquesta opció (o viceversa) per llavors seguir endavant. L'aplicació el que farà serà reduir el terme per el redex seleccionat.
2. Botons d'avançar i/o recular.
    a. Recular (←): Si el terme no és l'original, aquest botó permetrà recular un pas en la reducció del terme, que quedarà reflectit tant en l'arbre com en una àrea de text que marca el redex anterior.
    b. Avançar(→): Mentre el terme no estigui en forma normal, podrem avançar un pas de reducció (Normal, Aplicatiu o d'elecció lliure), mitjançant aquest botó. Si el terme ja està en forma normal, l'aplicació ens avisarà i no ens deixarà avançar més.

3. Forma normal: És possible obtenir la forma normal d'un terme si en té (o bé si no ho és, encara) directament prement aquest botó. El que farà l'aplicació serà tornar-la directament fent la reducció per ordre normal.
4. Reiniciar LTerme: Aquest botó posa tots elements a 0 amb l'excepció del Terme que s'ha entrat, és a dir, permet començar de 0 amb el Lambda Terme entrat sense necessitat de re introduir-lo de nou.
5. Borrar LTerme: Mitjançant aquest botó podrem eliminar completament el LTerme introduït de la memòria de l'aplicació per introduir-ne un de nou.
6. Elecció de Vista:
    a. Normal: Aquesta vista és l'activada per defecte i, de fet, la més usual. Permet veure el Lambda Terme en notació base del Lambda Càlcul, on hi apareixen variables (lligades o lliures).
    b. DeBruijn: Amb aquest element premut, veurem el Lambda Terme entrat amb notació de deBruijn, on enlloc de variables tindrem números enters, que marquen la distància en abstraccions entre la Lambda i la variable que lliga.

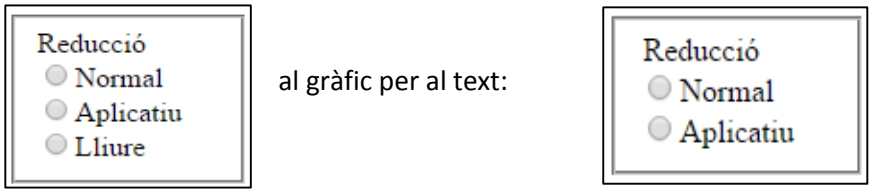
Apartat Text

Aquest apartat és diferent a l'apartat gràfic, i enlloc de mostrar el Lambda Terme en forma d'arbre, el mostra amb tots els parèntesis, només amb els obligats i finalment quants redex queden dins el mateix o bé avisa si ja està en forma normal.

L'entrada del Lambda Terme és la mateixa que la de l'apartat gràfic, hi ha una àrea de text per entrar-lo:



Les opcions del menú de text respecte a la del menú gràfic és que al text no es permet una reducció a escollir del terme, només es pot seleccionar ordre normal o ordre aplicatiu. La resta de botons es comporten de la mateixa manera. És a dir l'únic canvi és:



NOTA: Per facilitar-ne l'ús, es va decidir implementar la funcionalitat que quan es canvia d'un apartat a un altre (Gràfic a Text o viceversa) el Lambda Terme persisteix mentre no es tanqui la pestanya de navegació, moment en què caldrà re introduir el terme. això és interessant ja com que les funcionalitats que apareixen no són exactament les mateixes, a l'estudiant pot interessar-li navegar entre les dues.

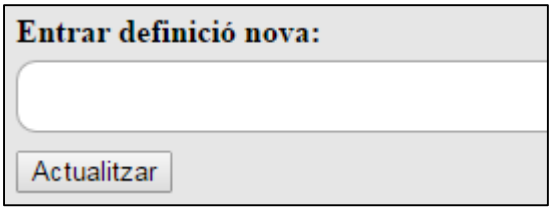
Apartat definicions

Actualment aquest apartat és funcional, però no permet interactuar amb els apartats de Text i Gràfic.

L'apartat es divideix en tres parts, la d'entrada de definicions, la de la vista de definicions ja carregades o la d'eliminar-ne alguna de les ja entrades.

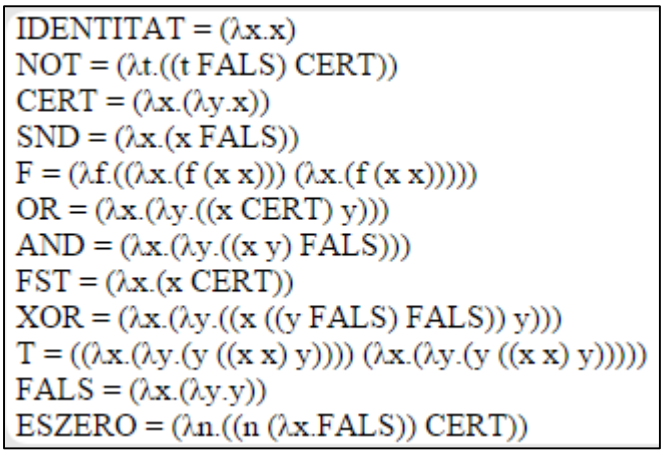
La introducció d'una definició seguirà el patró "nom definició=Terme". Podem veure la sintaxi de com entrar termes en l'apartat Gràfic d'aquest mateix capítol.

El primer és una àrea de text on l'usuari pot entrar una nova definició. L'entrada de les mateixes segueix l'ordre de nom, seguit d'un igual i el Lambda Terme que representa.

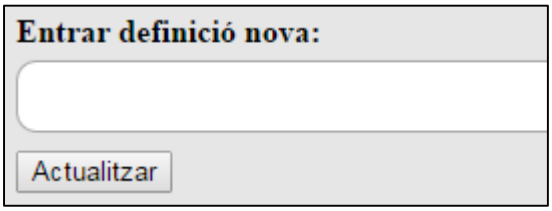


Un cop premut el botó "Actualitzar", si la definició és correcta, ja apareixerà a la llista de definicions entrades. Pot donar-se el cas que l'usuari vulgui usar el mateix nom per a una nova definició. Si vol actualitzar-la, només cal que introdueixi el mateix nom, i el programa ja s'encarregarà d'actualitzar-la.

A la part central de l'apartat trobem la llista de definicions. Per defecte estarà buit, però ara el mostrem amb una sèrie de definicions entrades per veure com es mostra i que és funcional:



Finalment, al peu es dóna l'opció d'eliminar una o vàries definicions entrades. Pot fer-se mitjançant la següent àrea de text:



NOTA IMPORTANT: Les definicions, per diferenciar-les amb la resta de variables, haurien d'estar escrites amb majúscules, però si s'entra alguna definició en minúscula l'aplicació no privarà l'entrada de la mateixa. El fet de no fer-ho pot arribar a confondre a l'estudiant ja que quan la funció estigui implementada, les definicions, mentre estiguin plegades, es tractaran com una variable més.

Apartat ajuda

En aquest últim apartat es mostra a l'usuari una petita ajuda de sobre com usar els diferents apartats de l'aplicació per a treballar amb la mateixa.