FACULTAT D'INFORMATICA
DE BARCELONA

TREBALL FINAL DE GRAU

CREACIO D'UN LLENGUATGE DE PROGRAMACIO PER A XARXES DE TRANSICIONS AUGMENTADES (ATN)

Autor:
Xavier Conejo Micó
Especialitat de computació

Director:
Lluís Padró Cirera
Departament de Ciències de la Computació

Resum

El processament de llenguatge natural, o NLP (Natural Language Processing), al llarg dels últims anys ha anat adquirint cada cop més importància en el món de la tecnologia. Fins i tot grans empreses com Google, Apple o Microsoft ja han fet els seus propis productes sobre aquest tema.

No obstant això, l'estudi del NLP és un món força ampli, començant per les seves finalitats, les quals poden ser molt diferents: traducció d'idiomes a temps real, incorporar-ho en projectes de domòtica, etc. Fins en les eines que s'utilitzen per a treballar-hi, en el nostre projecte ens hem volgut centrar en les xarxes de transició augmentades, o ATN (Augmented Transition Network).

Avui dia la implementació d'un ATN és una mica tediosa, pel fet que els llenguatges de programació més utilitzats no faciliten les eines adients. Per això hem confeccionat el nostre propi llenguatge, enfocat concretament en la realització de ATN's. De manera que la implementació d'un ATN sigui més senzilla, i d'aquesta manera sigui més viable a l'hora d'incorporar-los en projectes més grans sobre NLP.

Resumen

El procesamiento de lenguaje natural, o NLP (Natural Language Processing), a lo largo de los últimos años ha ido adquiriendo cada vez más importancia en el mundo de la tecnología. Incluso las grandes empresas como Google, Apple o Microsoft ya han hecho sus propios productos sobre este tema.

No obstante, el estudio del NLP es un mundo bastante amplio, empezando por sus finalidades, las cuales pueden ser muy diferentes: traducción de idiomas a tiempo real, incorporarlo en proyectos de domótica, etc. Hasta en las herramientas que se utilizan para trabajar sobre este, en nuestro proyecto nos hemos querido centrar en las redes de transición aumentadas, o ATN (Augmented Transition Network).

Hoy día la implementación de un ATN es un poco tediosa, por el hecho de que los lenguajes de programación más utilizados no facilitan las herramientas adecuadas. Por eso hemos querido confeccionar nuestro propio lenguaje, enfocado concretamente en la realización de ATN's. De manera que la implementación de un ATN sea más simple, i de esta manera sea más viable a la hora de incorporarlos en proyectos más grandes sobre NLP.

Abstract

The natural language processing, or NLP, has become increasingly important in the world of technology over the last few years. Even large companies like Google, Apple or Microsoft have already made their own products on this subject.

However, the study of the NLP is a very broad world, starting with its purposes, which can be very different: translation of languages in real time, incorporate it into domotic projects, etc. Even in the tools used to work on this, in our project we wanted to focus on the augmented transition network, or ATN.

Today the implementation of an ATN is a bit tedious, due to the fact that the most used programming languages do not provide the right tools. That's why we wanted to create our own language, focused specifically on the realization of ATN's. So that the implementation of an ATN is simpler, and in this way is more viable when incorporating them into larger NLP projects.

Índex de continugts

Índex de taules

Índex de figures

Índex de codi

1. Introducció

1.1. Contextualització

El processament de llenguatge natural, o NLP (Natural Language Processing), és un camp de la computació, intel·ligència artificial i lingüística que estudia la comunicació entre màquines i persones mitjançant el llenguatge humà.

Tracta de tindre en compte molts aspectes diferents que per als humans ens resulta normal a l'hora de comunicar-nos, des de la pronunciació per a diferenciar les diferents paraules, el to de veu que s'utilitza per indicar diferents emocions, sarcasmes, inclús utilitzar la nostra memòria per a referir-nos a esdeveniments passats.

Avui dia el processament de llenguatges naturals està agafant cada cop més força, pot ser degut al fet que en un món en el qual la tecnologia va agafant un paper cada vegada més important, i la manera més efectiva de comunicar-nos és a través de la parla, l'ideal seria que ens comuniquéssim amb aquesta nova tecnologia amb la mateixa facilitat que si ens comuniquéssim amb una altra persona.

Una vegada hem introduït el concepte del llenguatge natural, parlem ara de les xarxes de transició augmentades, o ATN (Augmented Transition Network). Aquestes són un tipus d'estructura sobre la teoria de grafs, utilitzades en la definició de llenguatges formals, especialment en analitzar llenguatges naturals complexos.

Com hem dit anteriorment el processament de llenguatge natural tracta de diversos factors, els ATN's són una eina que ens ajuden a analitzar frases del llenguatge, de per si sol no ens basta per a assolir el complex món del NLP, però ens pot servir per resoldre una part complexa d'aquest problema.

D’aquesta manera, mitjançant un conjunt de diferents tipus d’eines, cadascuna amb la seva funció, podem arribar a realitzar productes finals. Per exemple, els ATN’s ens poden ajudar a l’hora d’analitzar números, és a dir, que en detectar la frase “dos-cents quaranta i cinc” reconegui el nombre 245. Mentre que amb un altre tipus de tecnologia, com podria ser el machine learning, podríem detectar petits errors a l’hora d’escriure, de manera que si escrivim “dos cents quaranta i cinc”, aquesta primera part ho pogués transcriure a “dos-cents quaranta i cinc” i llavors aquest ATN ara si que entendrà aquesta frase com a 245.

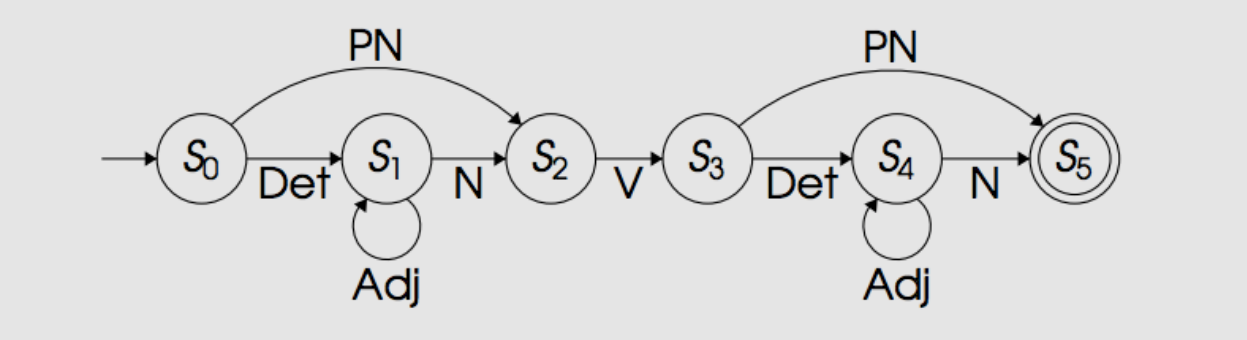

Figura 1.1: Exemple teòric d’un ATN

Centrant-nos en la implementació de xarxes de transició augmentades, podem trobar alguns llenguatges de programació que ens podrien ajudar, el més important potser seria el Lisp. Aquest és el segon llenguatge de programació d'alt nivell, encara utilitzat avui dia, més vell; FORTRAN és qui ocupa la primera posició.

Lisp va sorgir com una notació matemàtica pràctica per als programes de computadora, rapidament es va convertir en un bon llenguatge per a la investigació de la intel·ligència artificial. D'aquí es va convertir a un dels primers llenguatges de programació, el qual va ser pioner de moltes tècniques d'intel·ligència artificial: estructures de dades arborescents, tipus dinàmics, i d’altres.

Avui dia és un dels llenguatges que s'utilitzen per a la implementació de ATN's, tot i que caldria dir que poca gent utilitza encara Lisp, normalment es fan servir variants, com Common Lisp, Scheme, o una de les més noves MacList.

1.2. Motivació

Segons l'últim comentari que hem fet, es podria entendre que si ja existeixen llenguatges aptes per a la confecció de ATN's, perquè necessitem fer-ne un nosaltres.

Com hem comentat anteriorment, un projecte sobre llenguatge natural pot arribar a ser molt complex, inclús que només vulguem comprendre una petita part d'aquesta àrea d'estudi. Per tant es necessiten diverses eines que treballin conjuntament per aconseguir-ho, nosaltres volem que la base del nostre projecte estigui confeccionada amb un llenguatge més comú que els anteriorment esmentats, perquè es pugui complementar en altres projectes d'una manera més eficient.

Però aquesta no és la motivació més important. Com hem comentat en el punt anterior, el processament de llenguatge natural és un camp estudiat en altres rames en el que podem trobar investigadors que comencen a introduir-se en la informàtica, i per tant, els seus coneixements de programació tenen un nivell principiant, com per exemple podrien ser els lingüistes.

A més a més, el director del projecte té el seu propi projecte de processament de llenguatge natural, el Freeling[1], el qual fa ús de xarxes de transició augmentades per a detectors de dates, números, etc. Però actualment és molt pesat de programar, ja que no es disposa de les eines adients, tindre un mòdul destinat a la confecció de ATN's facilitaria molt aquest problema, el nostre llenguatge podria ocupar el lloc d'aquest mòdul.

1.3. Objectius

Ara que ja hem deixat clar el context i la motivació del projecte, continuem amb els objectius. L'objectiu principal és la realització d'un llenguatge de programació orientat a implementar xarxes de transició augmentades de manera que tingui una experiència d'usuari notable, sense deixar de banda la seva eficiència.

A més a més, haurà d'estar implementat en un llenguatge força utilitzat en grans projectes, de manera que sigui viable incorporar el nostre treball en altres treballs d'una major magnitud.

Una vegada terminat, hi ha un objectiu secundari, realitzar el nostre propi ATN amb l'ajuda del nostre propi llenguatge, de manera que puguem demostrar tant que

---

[1] FreeLing és un conjunt d'eines d'anàlisi del llenguatge de codi obert, publicat sota la Affero GNU General Public License de la Free Software Foundation.

El projecte FreeLing va ser creat i actualment és liderat per Lluís Padró com un medi per a passar a disposició de la comunitat ells resultats de la investigació realitzada en el grup d'investigació de processament de llenguatge natural de la UPC.

FreeLing és una biblioteca C ++ que proporciona funcionalitats d'anàlisi lingüístic (anàlisi morfològic, detecció d'entitats nomenades, PoS-tagging, anàlisi sintàctic, Word Sense Disambiguation, Semantic Role Labelling, etc.) per una varietat d'idiomes (anglès, espanyol, portuguès, itàlia, alemany, rus, català, gallec, croata, eslovè, entre d'altres).

funciona correctament com la utilitat que té. Al cap i a la fi, estem confeccionant una eina, hem de fer alguna mostra del que és capaç de fer.

Per últim, com ja hem mencionat anteriorment, en el camp del NLP és buscar integrar diferents tipus d'eines entre elles per a la realització de projectes més grans, per tant, podem buscar un projecte ja en funcionament de manera que en incorporar el nostre llenguatge de programació esdevingui considerables millores, una opció a considerar és l'anteriorment esmentat Freeling.

1.4. Actors

Els actors implicats en el projecte són:

- El *dissenyador*, *desenvolupador* i *beta tester* del llenguatge, que seré jo mateix, l'estudiant.
- El *director del projecte*, en Lluís Padró, que serà l'encarregat de supervisar el treball i si convé, d'ajudar al desenvolupador en les diferents etapes.
- Com a *usuaris finals* del projecte, qualsevol programador que vulgui utilitzar el llenguatge, sobretot tots aquells que vulguin provar de fer el seu propi projecte de processament de llenguatge natural utilitzant ATN's.

1.5. Estat de l'art

1.5.1. Situació actual del processament del llenguatge natural

Avui dia en l'estudi del NLP s'estan fent cada vegada més avenços, grans companyies com Apple, Microsoft, Google o Facebook cada vegada hi inverteixen més: realitzen els seus propis productes, compren petites empreses d'investigació que s'hi dediquen, etc.

Un exemple és la compra de Wavii per part de Google, aquesta era una start-up la qual els seus estudis sobre el llenguatge natural començaven a ser molt prometedors, és així com la gran companyia la va comprar per uns 30 milions de dòlars aproximadament. Google té clar la importància que té llenguatge natural i el terreny que va guanyant any rere any en el camp de la tecnologia.

Tot i així encara queda esperar un temps fins que comencem a veure introduccions significants del llenguatge natural en les activitats del dia a dia. De moment ja podem veure un petit avanç de què ens espera:

- Assistents personals intel·ligents al nostre telèfon mòbil, Siri (Apple) o Cortana (Microsoft) són alguns exemples. Que ens permeten des d'utilitzar funcions simples com indicar a qui volem trucar, fins que ens portin l'agenda al dia, recordar-nos d'esdeveniments pròxims, realitzar cerques de restaurants propers o fins i tot reconèixer quina cançó sona al teu voltant.

- Petites incorporacions a alguns automòbils, les quals ens permeten canviar d'una emissora de ràdio a una altra, trucar a algun contacte de la nostra llista de telèfons mòbils, llegir missatges, etc.

1.5.2. Utilitat de les xarxes de transició augmentades

Els ATN's tenen el seu petit paper dintre dels estudis sobre el processament de llenguatge natural, com ja hem mencionat anteriorment, la seva principal funció és la d'analitzar frases. Abans hem vist algun exemple sobre anàlisi de números, un altre exemple podria ser el d'analitzar frases que ens parlin sobre dates "el tretze d'Abril de 1998…" i el ATN doni com a resultat un objecte Data que contingui la informació rellevant.

Potser l'anterior exemple ha sigut massa simple per entendre la utilitat que tenen les xarxes de transició augmentades, la idea és que en tot un text, el qual pot ser des d'una frase, un paràgraf, fins i tot una pàgina sencera, és analitzat per un ATN concret, o diversos si escau, i segons sigui la finalitat d'aquest serà capaç d'indicar on ha trobat solucions possibles.

Posant un altre exemple més complex "L'aniversari d'en Joan és el 15 d'Abril, dos dies després és el d'en Jordi, i en mig el de la Marta" un bon programa que utilitzi xarxes de transició augmentades podria arribar a ser capaç de detectar totes les diferents dates, i fins i tot l'aniversari de cada persona.

1.5.3. Possibles solucions actuals

Avui dia trobem que ja hi ha llenguatges que ens permeten implementar un ATN, com hem dit anteriorment derivants del llenguatge Lisp entre d'altres. Per tant, actualment ja podríem utilitzar algun d'aquests llenguatges i implementar els nostres projectes.

O bé si volem utilitzar llenguatges més utilitzats, esmentant una de les motivacions anteriorment comentades, podríem fer la nostra pròpia llibreria sobre execució de ATN's en algun d'aquests llenguatges.

Per tant ja hi ha solucions per al problema que volem tractar, tot i que recordem que no tots els usuaris estan acostumats a tractar amb llenguatges de programació, llavors el repte que suposa confeccionar una xarxa de transició augmentada amb una eina que no està especialitzada en aquests podria ser considerable.

1.6. Definició de l'abast

1.6.1. Abast

Per poder assolir un llenguatge de programació per a implementar ATN's d'una manera practica i eficaç, calen destacar dos principals factors: fer un bon estudi sobre els ATN's i realitzar una bona implementació i disseny del llenguatge.

El primer és fàcil d'entendre perquè, si ens disposem a desenvolupar un llenguatge de programació amb el principal objectiu de confeccionar xarxes de transició augmentades, hem de realitzar un estudi previ sobre aquestes, les diferents maneres que les podem recórrer, com encarar-les per a analitzar frases, etc. En conclusió, entendre el millor possible els ATN.

Procedint, volem fer un llenguatge de programació potent, que implementi el ATN desitjat per l'usuari de manera que per a aquest sigui faci'l d'entendre, però alhora el programa sigui eficient i construït en una base capaç de complementar-se amb altres projectes de major magnitud.

Per aconseguir-ho ens recolzarem en l'experiència del director del treball sobre el tema perquè ens indiqui fonts d'informació útils, de manera que puguem aprendre tot allò que necessitem. Li consultarem dubtes que tinguem des del disseny fins a la implementació.

A més a més, podem agafar el projecte del director del treball, el Freeling, com a referència de projecte de major magnitud al qual el nostre llenguatge ha de ser

capaç d'incorporar-se, cosa que augmentarà la potència de la llibreria i la serà d'utilitat a molts més usuaris.

Ara passem a la part de la implementació del llenguatge, com hem dit volem que sigui el més òptim possible, a part que ha de ser compatible amb el Freeling, per tant buscarem les tecnologies que ens permetin que tant l'execució del programa sigui eficient com que ens faciliti l'adaptació a aquest altre projecte.

Una vegada assolit aquest punt, procedirem a realitzar un bon disseny del llenguatge. Hem de recordar que el principal punt és que sigui un llenguatge encarat a implementar ATN's, i tot i que amb el punt anterior assolim la part més teòrica, tampoc podem descuidar la part de l'experiència d'usuari, ha de ser fàcil portar a terme programes sobre conceptes matemàtics.

En resum, l'usuari final hauria de poder implementar un programa que tingui com a base una xarxa de transició augmentada, un concepte teòric complex, de la manera més simple possible, sense perdre vista que ha de ser pragmàtic.

Recordem que en l'estudi del NLP hi ha molts investigadors que són lingüistes i les seves nocions de programació no són les mateixes que un graduat en informàtica.

1.6.2. Obstacles

A continuació es comenten els possibles obstacles que vam plantejar en un principi que podrien haver sorgit durant el projecte i d'altres que potser no havíem contemplat i com els hem resolt.

1.6.2.1. Falta de coneixements sobre ATN

Fins aquest treball no havíem treballat mai amb ATN's de manera que aquest era un dels principals obstacles a l'hora d'implementar el llenguatge que pensàvem que tindríem. Per tant ens preocupava que aquest punt ens treies força temps de la resta del projecte la qual seria més important.

Però durant el desenvolupament del projecte hem vist que no ha sigut un problema, les assignatures cursades al llarg de la carrera ens han proporcionat uns coneixements molt amplis, i tot i no haver estudiat concretament els ATN's, a l'haver estudiat altres tipus de grafs més generals i d'una manera complexa, ens ha servit per a poder entendre aquests més específics però no tan complexos.

Al final ha resultat ser el punt més senzill que hem tingut en al llarg de tot projecte.

1.6.2.2. Disseny

Si el resultat és un llenguatge de programació difícil d'utilitzar, no haurem assolit bé els objectius del treball. Per tant elaborar un bon disseny és important i pot arribar a ser un obstacle a causa del temps que pugui treure de la resta de parts del treball.

Tot i així, una bona solució és buscar altres bons llenguatges i intentar inspirar-nos en el seu disseny, buscar bona documentació sobre el desenvolupament d'un llenguatge nou o provar d'escriure els nostres propis ATN's i veure quina seria la millor manera d'adaptar-ho a un programa.

Aquest ha sigut un obstacle que hem resolt a força de buscar informació i prova i error, fins a l'últim moment hem anat estudiant quina seria la manera més adient

de tornar a reescriure els exemples que hem fet, polint tots els detalls que hem pogut.

1.6.2.3. Calendari

El principal obstacle, com hem esmentat en algun punt anterior, és el temps, serà important determinar les funcionalitats que vulguem que tingui el llenguatge i quines deixar-les en un pla més opcional, de manera que si disposem del temps suficient provar d'implementar-les.

Cal dir que no vam saber tractar bé aquest obstacle, tot i haver-ho tingut en compte des d'un inici del projecte.

1.6.2.4. Tecnologies emprades

Un obstacle que no havíem contemplat al principi del treball, i per això l'afegim una vegada hem terminat el projecte, són les tecnologies que utilitzaríem per a la part de l'escàner i la gramàtica.

Hem tingut problemes a l'hora de buscar una bona tecnologia que ens permetés que les diferents parts del llenguatge funcionessin conjuntament.

Més concretament, el problema que hem tingut no ha sigut el de buscar quina tecnologia utilitzar, ja que rapidament vam trobar una viable, però va resultar que aquesta que vam escollir, n'hi havia diverses variants possibles, per tant vam estar estudiant quina seria la més optima per al nostre cas en concret.

Vam trobar que la documentació oficial d'algunes d'aquestes no estàvem gaire bé detallades, amb el que va derivar en què les nostres principals fonts d'informació fossin altres usuaris. Com vam mal entendre les explicacions i vam

barrejar diferents llenguatges de programació en el mateix codi de la manera errònia.

Com a conseqüència van anar generant-se múltiples errors fins a tal punt que vam veure que no era viable continuar el projecte, fins que vam haver de començar el projecte de nou sense utilitzar cap variant, sinó l'original directament.

La qual cosa ens va endarrerir considerablement en la planificació inicial del projecte.

1.6.3. Metodologia i rigor

Com hem dit en el punt anterior, el temps és crucial, per tant establirem una metodologia adequada per poder aprofitar tot el temps disponible per a realitzar el treball. A continuació es descriuen les metodologies concretes que es portaran a terme.

1.6.3.1. Prototips i tests

Un bon mètode de treball és el de realitzar un primer prototip, el qual sigui molt senzill i no té quasi cap funcionalitat, i a poc a poc anar ampliant-lo. De manera que sempre sigui un llenguatge operatiu.

Un punt pel qual ens vam decantar per aquest mètode, a part del comentat anteriorment, és que en tindre dues parts diferenciades a l'hora d'implementar el llenguatge, la part de la gramàtica i la part de l'intèrpret, és difícil primer terminar una part, i després l'altre. Ja que tot i estar molt diferenciades, una depèn molt de l'altre, per tant és molt freqüent anar fent variacions en les dues parts per separat,

per aquest motiu buscar tasques petites i anar revisant sempre les dues ens ha funcionat força bé.

També és important la implementació de tests per a cada funcionalitat del treball, d'aquesta manera es manté la consistència al llarg del projecte, ja que en implementar una nova funcionalitat potser hem generat errors en un altre que ja havíem terminat anteriorment.

1.6.3.2. Metodoligies Incrementals (SCRUM-Agile)

Com acabem d'exposar, anirem fent petits prototips, però la manera en la qual desenvoluparem cada prototip serà amb metodologies incrementals, SCRUM[2]. Definirem les històries d'usuari, per cadascuna d'aquestes descriurem quines són les tasques necessàries que hem de portar a terme, es farà un càlcul aproximat del temps de cadascuna i d'aquesta manera també podem tindre un major control sobre el temps emprat en el projecte.

En aquest treball utilitzarem una eina que ens servirà per dur a terme el SCRUM, la qual s'anomena Trello. Trello és una pàgina web en la qual pots crear els teus propis marcs de treball per a cada projecte de manera gratuïta.

---

[2] Scrum és un marc de treball per a la gestió de projectes. Inicialment força estès entre els desenvolupadors i mantenidors de programari o d'aplicacions. Amb el temps, Scrum també s'ha fet popular per a la gestió de programes i de múltiples equips de desenvolupament. L'objectiu és desenvolupar i crear un producte en un període determinat on un equip format per diferents persones treballen conjuntament per arribar a un objectiu comú.

1.6.3.3. Reunions amb el director

Com hem esmentat anteriorment en aquest treball necessitarem la guia del director perquè ens aconselli al llarg del treball. Seria aconsellable reunir-se amb el director periòdicament, d'aquesta manera el projecte estarà en un continu desenvolupament. Aprofitant al màxim el temps del qual disposem.

Tot i que cal dir que al final no hem pogut seguir amb la periodicitat desitjada amb les reunions a causa de problemes personals, hem anat mantenint un contacte també via email.

1.6.3.4. Còpies de seguretat

És important anar realitzant còpies de seguretat, de fet tindre diverses d'aquestes. Ja que sempre podem tindre algun problema i perdre tot el projecte, o un error molt més comú, en implementar una funcionalitat nova generar tal quantitat d'errors que és més senzill recuperar una còpia de seguretat i començar de nou aquesta nova funcionalitat.

Una bona eina és el control de versions Git[3], el qual gestiona perfectament aquest problema que acabem d'esmentar, i a més a més, fa una còpia de seguretat en la xarxa, de manera que podem estar tranquils de no perdre el projecte.

[3] Git és un programari de sistema de control de versions dissenyat per Linus Torvalds, pensat en l'eficiència i confiabilitat de manteniment de versions d'aplicacions amb una enorme quantitat de fitxers de codi font.

2. Planificació del projecte

2.1. Planificació temporal

En aquesta part de la documentació parlarem sobre la planificació temporal del treball que es va fer en el començament d'aquest mateix, i explicarem els canvis que ha patit i per quins motius.

Exposarem les principals tasques que el formen, explicant en què consisteixen cadascuna, quina és la seva finalitat i es farà un càlcul aproximat del temps que ens dura portar-les a terme.

També exposarem un diagrama temporal de Gantt que ens servirà per veure amb perspectiva les durades de les tasques i si pot haver-hi algun conflicte de dependències entre elles o algun altre problema.

Aquest treball va començar aproximadament el 26 de Setembre i conclourà el 18 d'abril (29 setmanes). Ho tindrem en compte per a quan exposem com ha anat la distribució de tasques.

2.2. Descripció de les tasques

A continuació expliquem les diferents tasques dins del projecte, les quals no ha patit cap canvi des de l'inici d'aquest, les explicarem en ordre temporal d'elaboració.

2.2.1. Planificació del projecte

Aquesta primera tasca és la qual ens trobem actualment, és la primera tasca que hem de realitzar per a especificar els objectius, l'abast i la planificació del treball.

El podem dividir en:

- Abast i Contextualització
- Planificació temporal
- Gestió econòmica i sostenibilitat

La funció principal d'aquesta part és destinar el temps necessari a organitzar el temps disponible que tenim per realitzar el projecte en realitzar aquest mateix.

2.2.2. Disseny i implementació del projecte

Ara procedirem a com hem planificat el projecte. Com hem dit anteriorment en el nostre projecte l'hem planificat com a prototips, de manera que anirem fent cicles en una sèrie d'apartats, a continuació els expliquem.

2.2.2.1. Set up

Prèviament a la realització del treball, dedicarem un temps a preparar tots els dispositius que necessitem, és a dir, a tindre a punt tot el software que haurem d'utilitzar, les llibreries, iniciar el projecte tant a la plataforma de Git com la de Trello, etc.

És una part molt tediosa, però és més eficient destinar una part prèvia per a després no tindre imprevistos amb versions desactualitzades de software, o amb llibreries incompletes i que no podem utilitzar.

2.2.2.2. Sèrie de prototip

Cada sèrie consta de les següents etapes:

- *Anàlisis de noves funcionalitats*. Buscarem quines noves funcionalitats hem d'incorporar al projecte, de manera anem avançant fins al nostre objectiu final.

- *Desglossament dels casos d'ús en tasques*. Procedim a descompondre cada cas d'ús en una sèrie de tasques, que ens serviran per a dur a terme el SCRUM. Hi haurà tres tipus principals de tasques: Disseny, Implementació i Tests.

- *Disseny*. Aquesta tasca té la finalitat de definir el disseny modular del projecte, és a dir, incorporar les noves funcionalitats de la manera correcta, tenint en compte aquelles que ja han sigut implementades i si es pot reaprofitar algunes parts d'aquestes, en el cas que no sigui possible, haurem de trobar la millor manera d'incorporar aquesta nova funcionalitat mantenint la coherència del projecte.

- *Implementació*. Una vegada hem dissenyat les funcionalitats corresponents, cal implementar- les. Intentarem sempre fer-ho de la manera més òptima, per a això podrem consultar referències adients a cada problema que haurem de resoldre i acudir al director en el cas que fos necessari.

- *Tests*. Tot i que en la part d'implementació també es fa un petit testeig, les tasques de tests tenen la finalitat d'implementar una sèrie de tests que comprovaran si les noves funcionalitats actuen correctament, de manera que quan fem de noves podrem comprovar si hem generat algun nou error en alguna ja implementada.

- *Cerca de bugs*. L'última part és comprovar que les funcionalitats d'anteriors sèries continuen operant sense errors, per tant només hem d'executar tots els tests prèviament implementats i comprovar que és així. En cas d'haver-hi algun error, procedirem a arreglar-lo, si aquest resulta ser força important, o dit amb altres paraules, d'una magnitud considerable, podem o bé tornar a la fase de disseny i tornar a procedir des d'aquesta per tal de resoldre aquest error, o bé eliminar la funcionalitat causant de l'error d'aquest cicle del prototip, i tornar a afegir-la en la següent sèrie.

Com ja hem dit, aquest conjunt de procediments, s'aniran executant cíclicament, agrupant acumulacions de funcionalitats per cada sèrie, segons la magnitud total de les funcionalitats.

2.2.3. Tasca final

En un principi ja hauríem de tindre el nostre propi llenguatge funcionant al complet, per tant ens queda un últim objectiu que assolir, la implementació d'uns programes amb el nostre propi llenguatge.

Com a mínim realitzarem un exemple, perquè serveixi de mostra. Seran programes senzills però d'utilitat real.

2.3. Duració de les tasques

En aquest apartat parlarem de la duració de les tasques, primer de tot parlarem de la distribució del temps que teníem planificada, procedirem amb valoracions d'alternatives que vam valorar al començament del projecte i finalment exposarem els imprevistos que hem tingut i s'ha vist modificada la distribució inicial.

2.3.1. Distribució de temps

A la següent taula s'indiquen els temps previstos de cadascuna de les tasques:

Tasca	Temps (hores)
Planificació del projecte	50
Set up	30
Anàlisis de noves funcionalitats	5 hores/cicle * 3 = 15 (aprox. 3 cicles)
Descripció dels casos d'ús	5 hores/cicle * 3 = 15 (aprox. 3 cicles)
Desglossament dels casos d'ús	5 hores/cicle * 3 = 15 (aprox. 3 cicles)
Disseny	15 hores/cicle * 3 = 45 (aprox. 3 cicles)
Implementació	60 hores/cicle * 3 = 180 (aprox. 3 cicles)
Tests	15 hores/cicle * 3 = 45 (aprox. 3 cicles)
Cerca de bugs	15 hores/cicle * 3 = 45 (aprox. 3 cicles)
Tasca final	120
Total	**560**

Taula 1: Distribució del temps previst per cada tasca

2.3.2. Valoració d'alternatives

Si aconseguim realitzar les tasques en el temps establert, seria el cas ideal, però segurament tindrem imprevistos durant la realització del treball.

Pot ser que hi hagi imprevistos que ens desviïn del temps estimat, calculem un marge d'error d'un 10%, és a dir, unes 56 hores.

Si ens sorgeix més imprevistos, una solució per la qual podem optar és la de treure funcionalitats, debut a seguir sèries de prototips, les primeres funcionalitats que realitzarem seran les més essencials, de manera que a mesura que anem avançant al llarg del treball, romandran les menys essencials, per tant podem considerar a no afegir-les al projecte i deixar-les com a treball futur.

2.3.3. Imprevistos

A continuació exposarem els imprevistos que hem tingut al llarg del treball, el primer imprevist a destacar és un problema que vam tindre durant la tasca de *Set up*. Aquest ja l'hem explicat en l'apartat d'obstacles, concretament és el punt *1.6.2.3. Tecnologies emprades*.

Com ja l'hem explicat anteriorment no tornarem a concretar que va passar, però si recordem que en qüestió de temps va ser un imprevist força greu, ja que vam haver de començar de nou el projecte.

Un altre imprevist que ens va impedir l'avanç del projecte va ser ocasionat per diversos problemes personals, entre els quals consta un petit accident que vaig patir que em va deixar indisposat un parell de setmanes.

El conjunt d'aquests va derivar en què des de mitjans de Desembre fins a finals de Gener no pogués avançar el projecte.

Tasca	Temps (hores)
Planificació del projecte	50
Set up (incloent imprevist)	30 + 125 (temps d'un cicle) = 155
Anàlisis de noves funcionalitats	5 hores/cicle * 3 = 15 (aprox. 3 cicles)
Descripció dels casos d'ús	5 hores/cicle * 3 = 15 (aprox. 3 cicles)
Desglossament dels casos d'ús	5 hores/cicle * 3 = 15 (aprox. 3 cicles)
Disseny	15 hores/cicle * 3 = 45 (aprox. 3 cicles)
Implementació	60 hores/cicle * 3 = 180 (aprox. 3 cicles)
Tests	15 hores/cicle * 3 = 45 (aprox. 3 cicles)
Cerca de bugs	20 hores/cicle * 3 = 60 (aprox. 3 cicles)
Tasca final	120
Total	**700**

Taula 2: Distribució del temps aproximat per cada tasca

2.4. Recursos

Els recursos que necessitarem per portar a terme el treball, són molt simples, els dividirem en software i hardware.

- Recursos software:
  - *Sistema operatiu*: a totes les tasques, en el nostre cas serà Mac i Ubuntu

- *Trello*: per dur a terme el SCRUM
- *Git*: ens facilitarà el control de versions

- Recursos hardware:
  - *Ordinador*, amb prestacions mínimes, el nostre projecte no requereix d'unes necessitats gaire grans, en el nostre cas utilitzarem un Macbook Air, degut que és l'ordinador que disposem.

2.5. Diagrama de Grantt

El següent diagrama de Grantt representa de forma gràfica la planificació explicada en aquest apartat.

Figura 2.1: Diagrama de Gantt del projecte

2.6. Gestió econòmica i sostenibilitat

2.6.1. Gestió econòmica

Aquest apartat tracta sobre l'estimació econòmica de tot el treball, tenint en compte els recursos esmentats en els apartats anteriors. Podem desglossar aquesta estimació en quatre parts diferents: costos directes per activitat, costos indirectes, contingències i imprevistos.

2.6.1.1. Costos directes

El cost emprat en recursos humans, és a dir, el salari dels treballadors en aquest treball és nul, ja que tots els realitzarà l'estudiant. Per tant no ho tindrem en compte, així doncs ens queden tres tipus diferents: Recursos humans, Hardware i Software.

2.6.1.1.1. Recursos humans

Tot i que els recursos humans d'aquest projecte estaran compostos per l'estudiant i el director, expossarem els costos del dissenyador, desenvolupador, beta tester i del director del treball.

Rol	Preu per hora (€/hora)	Temps (hores)	Cost per rol (€)
Director	50	125	6,250
Analista	30	60	1,800
Dissenyador	35	200	7,000
Desenvolupador	35	275	9,625
Beta tester	25	40	1,000
Total		700	25,675

Taula 3: Depeses de recursos humans

2.6.1.1.2. Hardware

El hardware que emprarem en aquest treball és realment simple, com ja hem esmentat en apartats anteriors. Tot i així és imprescindible comptar amb aquest material, a continuació detallarem les despeses d'aquest.

Producte	Preu (€)	Unitats	Vida útil (anys)	Amortització (€)
Ordinador portàtil	1,700.00	1	5 - 10	113.33
Monitor	80.00	1	5 - 10	5.33
Total	1,780.00	2	5 - 10	163.33

Taula 4: Despeses de Hardware

2.6.1.1.3.Software

El software que emprarem en aquest treball té cost zero, com que o bé son open source, software proporcionat per la facultat, o bé ja bé inclòs amb el hardware.

Producte	Preu (€)	Unitats	Vida útil (anys)	Amortització (€)
Sistema operatiu MAC	0.00	1	5 - 10	0.00
Pages	0.00	1	5 - 10	0.00
Sublime Text	0.00	1	5 - 10	0.00
Git	0.00	1	5 -10	0.00
Total	0.00	1	5 -10	0.00

Taula 5: Despeses de Software

Per últim, afegir que la vida útil és un càlcul aproximat, sobretot en el software, ja que no es farà malbé i les actualitzacions són gratuïtes, per tant la vida útil del software depèn del hardware on s'executa, per això hem posat les mateixes dades.

2.6.1.2. Costos indirectes

Per costos indirectes del treball, considerem les despeses generals, les quals tenen en compte, la despesa de l'electricitat, la qual utilitzem en fer funcionar tot el hardware esmentat, entre d'altres; la despesa del telèfon mòbil i serveis d'internet; i per últim el cost del transport per acudir a les cites amb el director del treball.

Producte	Preu	Unitats	Amortització (€)
Electricitat	0.11 €/Kwh	400	44.00
Telèfon	40.00 €/mes	4	160.00
Transport	20.00 €/mes	4	80.00
Total	104 €/mes		284.00

Taula 6: Despeses generals

2.6.1.3. Contingències

A continuació mostres una taula resum de totes les d'aquest apartat, tot i que ho fem segons el preu amortitzat, ja que per a un ordinador per exemple, no

s'utilitzarà només per aquest projecte ni només durant la durada d'aquest:

Despeses	Amortització (€)
Recursos humans	25,675.00
Hardware	163.33
Software	0.00
Generals	284.00
Total	26,122.33

Taula 7: Pressupost total

Tot i haver fet tots aquests càlculs, al llarg d'un treball poden haver-hi imprevistos, és millor tindre'l's en compte quan estem realitzant el càlcul del pressupost, i així tindrem un pressupost realitzat a l'alça, que sempre és millor que no pas ens falti de capital per a realitzar el projecte que en sobri.

Un bon percentatge dels imprevistos podria ser del 5%.

Import	Percentatge	Import Total
26,122.33	5%	1,306.1165

Taula 8: Pressupost total amb imprevistos

Finalment realitzem el últim càlcul, el qual consisteix a calcular el pressupost final tenint en compte els impostos, en el nostre cas només s'era el 21% de l'IVA.

Import	Impost	Import Total
1,306.1165	21%	274.284465

Taula 9: Pressupost total amb impostos

2.6.2. Sostenibilitat

2.6.2.1. Dimensió econòmica

El pressupost final vist anteriorment es troba dins de les dimensions d'un treball d'aquest estil. Fem un repàs ràpid als costos, els menys importants serien aquells relacionats amb l'electricitat, internet, transport i d'altres, aquests costos estan presents en qualsevol tipus de treball, per tants són imprescindibles.

Per una altra banda tenim els costos de hardware i software, cal remarcar la importància d'aquests recursos no només ens serveixen per a aquest treball, sinó per a molts d'altres, per tant, l'amortització es força gran.

En altres treballs del mateix tipus el pressupost és semblant, tot i que costa molt de comparar, ja que com hem dit, els mateixos materials s'utilitzen per a diversos projectes.

2.6.2.2. Dimensió social

L'aportació social del projecte és considerable, recordem que l'objectiu és desenvolupar un llenguatge de programació encarat a desenvolupar ATN's. Per

tant, tots aquells programadors que ho desitgin, el podran utilitzar per a fer els seus propis projectes.

Remarquem també que, com vam dir, ja existeixen llenguatges que permeten implementar ATN's, però d'una manera molt més costosa, amb el nostre llenguatge podrem aconseguir que altres especialistes del processament del llenguatge natural que tinguin menys nocions de programació, com ara podrien ser lingüistes, l'utilitzessin sense cap mensa de dificultat. A més a més, remarcar el que és un software lliure, per tant es disposa aquesta nova tecnologia a l'abast de tothom.

2.6.2.3. Dimensió ambiental

El dany al medi ambient del nostre projecte és mínim, per una banda, el transport que utilitzarem al llarg del projecte serà transport públic, d'aquesta manera minimitzarem el dany a la capa d'ozó.

A part d'aquest dany, només tenim el que pugui causar l'electricitat consumida, que com podem veure en les taules anteriors, serà un cost insignificant, per tant el dany al medi ambient no serà considerable.

Per últim, si tenim en consideració el hardware que utilitzem, cosa que bé no podríem fer a causa de la llarga vida d'aquests, també són uns danys mínims, ja que existeixen molts projectes d'ONG's sobre buscar noves utilitats a hardwares vells o espatllats, com per exemple, agafar les parts que encara funcionen per fer-ne de nous i reutilitzar-los en països menys desenvolupats.

2.6.2.4. Puntuació final de sostenibilitat

A continuació es mostra una taula resum amb les diferents puntuacions de les a àrees del projecte analitzades en aquests apartats.

Sostenible?	Econòmica	Social	Ambiental
Planificació	Viabilitat econòmica	Millora de qualitat de vida	Viabilitat ambiental
Valoració [0, 10]	9	0,5	9
Resultats	Cost final versus previsió	Impacte social	Consum de recursos
Valoració [-10, +10]	0	2,5	9
Riscos	Adaptació a canvis	Danys socials	Danys ambientals
Valoració [-20, 0]	0	0	0
Valoració total	**30**		

Taula 10: Matriu de sostenibilitat

3. Xarxes de transició augmentades

A partir d'aquest punt ens començarem a centrar en el que és el projecte en si, començarem per explicar en profunditat les xarxes de transició augmentades, sobretot el com funcionen.

3.1. Definició

Una xarxa de transició augmentada, com ja hem comentat anteriorment, és un tipus d'estructura de la teoria de grafs, més concretament són una extensió de les xarxes de transició recursives, o Relative Transition Network (RTN). Els ATN's s'utilitzen en l'anàlisi del llenguatge natural, els quals són normalment emprats en l'anàlisi de frase de manera que s'aconsegueix interpretar el significat d'aquesta.

3.1.1. Funcionament

Les xarxes de transició augmentada es poden entendre millor si les veiem com un tipus de màquina d'estats finits, com un autòmat finit o una màquina de turing, els quals estan compostos per una sèrie d'*estats*, els quals es poden connectar entre ells mitjançant una sèrie de *transicions*, on s'analitza una entrada donada, en el cas en concret dels ATN, aquesta entrada són frases, paràgrafs o textos entre d'altres.

A més a més, cada estat executa un llistat d'instruccions programables, d'igual manera que és tractes d'una subrutina.

Podem diferenciar tres tipus d'estats, els inicials, els finals i els normals. Els estats normals no tenen cap característica especial, els inicials són els primers que s'executaran cada vegada que comenci a executar-se aquest ATN, i per últim, els

finals són aquells que detecten una anàlisi complet, de manera que és una possible solució.

En altres tipus d'autòmats, les transicions que uneixen uns nodes amb uns altres, solen ser condicions booleanes, de manera que si s'interpreten com a "true" l'execució pot continuar cap a aquest següent node. En els ATN's és semblant, però afegint que és en aquest moment on s'analitza la frase objectiva. En cada aresta s'analitza una paraula, un exemple senzill seria que en cada aresta hi indiquem una paraula, si són iguals, s'avalua com a "true".

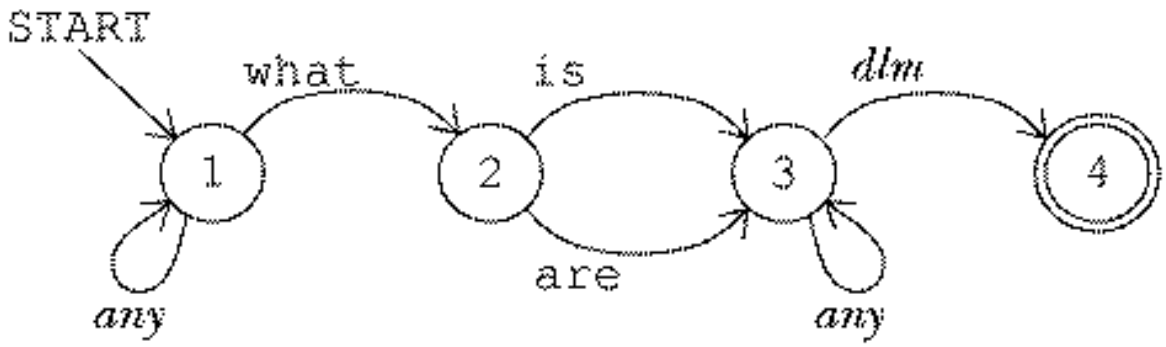

Figura 3.1: Exemple senzill d'un ATN

Ara que hem vist amb profunditat que és i com funciona un ATN, podem entendre el perquè el seu objectiu final és el d'anàlisi de textos. Aquest model compleix moltes de les metes establertes per la naturalesa del llenguatge en què capta les regularitats del llenguatge natural.

3.2. ATN’s en el nostre llenguatge de programació

Procedirem a explicar el com s'executaran els ATN's en el nostre llenguatge de programació.

3.2.1. Elements del llenguatge

Començant per deixar clar que l’element principal són els ATN’s, podem prosseguir per establir que és obvi que cada estat pot representar una crida a una subrutina, per tant, el nostre llenguatge ha de ser capaç de tindre tot un conjunt d'instruccions igual que molts altres llenguatges: aritmètiques, booleanes, estructures de dades, crides a funcions, etc.

Veiem una necessitat llavors que el nostre llenguatge sigui capaç de no tindre només ATN's com a element principal, sinó també funcions que es puguin executar en els estats d'aquests, o inclús en altres funcions.

Tornant a les xarxes de transició augmentades, trobem un altre necessitat alhora de què els estats han poder de comunicar-se entre si, és a dir, poder d'alguna manera, compartir dades. A més a més, ens pot interessar que aquestes dades també siguin visibles des d'algunes subrutines, per tant concloem en la definició d'unes variables globals, les quals poden ser accessibles des de qualsevol part del programa.

Prosseguim, com hem esmentat abans, també hem de diferenciar els tres tipus diferents d’estats. Els normals no requereixen cap tipus d'indicació, els inicials per una altra part cal que estiguin llistats per a quan hàgim d'executar el ATN poder iniciar en cadascun d’aquests.

Els finals en canvi, podem optar per no indicar-los, i si en el codi a executar en un estat determinat detectem un output, entenem que és un estat final, tot i així, decidim indicar-los igual que els estats inicials, ja que per motius que esmentarem

més endavant un mateix estat no pot ser final i inicial alhora, i d'aquesta manera creiem que a l'usuari li serà fàcil de veure si comet aquest error.

```
ATN main {
    initial states: start;
    final states: final;

    STATE start {
        action { }
        transition {
            to start if ($ != "what");
            to s1 if ($ == "what");
        }
    }

    STATE s1 {
        action { }
        transition {
            to s2 if ($ == "is" || $ == "are");
        }
    }

    STATE s2 {
        action { }
        transition {
            to final if ($ == "?");
            to s2 if ($ != "?");
        }
    }

    STATE final {
        action {
            @ = "OUTPUT CORRECT";
        }
        transition { }
    }
}
```

Codi 1: Exemple del codi de la figura 3.1.

3.2.2. Execució del llenguatge

Per terminar aquest punt, explicarem amb detall l'execució que farà el nostre llenguatge de programació. Passarem per alt l'explicació de l'execució de subrutines, ja que no tenen cap dificultat.

En el llenguatge hi ha tres tipus d'elements principals, com ja hem esmentat anteriorment, les variables globals, les funcions i els ATN's. De manera que el codi podrà tindre aquests elements, les variables globals i les funcions només tindran importància quan se'n facin ús d'elles, per tant podem centrar-nos en les xarxes de transició augmentades.

Primerament el llenguatge buscarà el ATN amb el nom “main”, ja que aquest serà el que s'executarà. Tot seguit executarà cada estat inicial, per cadascun d'aquest el procediment és el mateix, primer dur a terme el codi determinat en l'estat, tot seguit comprovarà totes les transicions possibles.

Durant l'execució dels estats inicials encara no s'ha analitzat cap paraula de l'entrada, és en les transicions de l'estat inicial en el qual s'inicia l'anàlisi de la primera paraula, d'aquesta manera, a mesura que es vagi avançant d'un estat a un altre s'aniran analitzant una paraula rere altre successivament, a no ser que en la transició s'indiqui, mitjançant un caràcter reservat, que s'omet l'anàlisi de la paraula actual.

Expliquem ara el procés de les transicions, per cada una s'executarà la condició indicada, si es determina com a “true” procedirà a l'estat indicat, si per una altra banda resulta “false” no s'executarà aquest estat.

Hi ha dos tipus de transicions, les més senzilles són una expressió booleana, o en el seu defecte una funció que retorna un valor booleà, sigui un o l'altre, aquest és l'únic moment del programa en el qual es pot fer referencia a la paraula a

analitzar. L'altre tipus consta d'un identificador referent a un altre ATN del programa.

Centrant-nos primer en les més simples, com acabem de dir, procedirà a executar el següent estat, i així successivament, de manera que si en algun moment arribem a un estat final, en aquest es crearà un output que el programa recollirà i al final de l'execució de tot el codi, retornarà tots els outputs recollits.

Però tot i arribar a un estat final, pot ser que encara tinguem paraules a analitzar, per tant el ATN s'anirà executant de la mateixa manera, si arriba el moment en què hem analitzat tot el text, el codi es donarà per finalitzat. Observem el cas en el qual es terminen abans els estats possibles a executar que la frase a analitzar, llavors es tornarà a executar el ATN "main", però no començarem a analitzar tot el text des de zero, sinó que procedirem des del punt el qual ens havíem quedat.

Un exemple seria el cas de tindre un ATN que analitzi números i la meva entrada sigui "Pel meu aniversari vaig aconseguir cinc-centes tretze pessetes.", és obvi que fins a la paraula "cinc" el ATN només descartarà les altres, això ho farà degut això que acabem d'explicar, totes aquestes paraules del començament no resultaran en "true" a cap transició de cap estat inicial, donant com a resultat que el ATN comenci des del principi una altra vegada però amb una paraula a tractar menys, d'aquesta manera arribarà el moment de tractar "cinc", i llavors el ATN podrà entendre el número i una vegada hagi tractat aquest nombre al ser el següent element a analitzar "pessetes" no el sabrà interpretar i començarà de nou, d'aquesta manera si el text fos més llarg i més endavant hi hagués altres números seria capaç de detectar-los tots.

Amb això podríem concloure tota una execució d'un ATN, a continuació explicarem el tipus de transicions les quals contenen l'identificador d'un altre ATN. Bàsicament, abans de procedir amb el ATN actual (el qual anomenarem A) executarà el ATN determinat (el qual anomenarem B), això vol dir que la paraula a

analitzar que pertanyia a aquesta transició passarà a ser la primera que haurà d'analitzar B. Per tant, el ATN B s'executarà de manera que acabem de descriure, quan termini, es procedirà a realitzar, per cadascun dels outputs donats per B, l'estat indicat en la transició que s'havia indicat en A. A més a més, per cada una d'aquestes execucions, s'actualitzarà la paraula a analitzar del ATN A, sent ara la següent de l'última paraula del matching del output en qüestió.

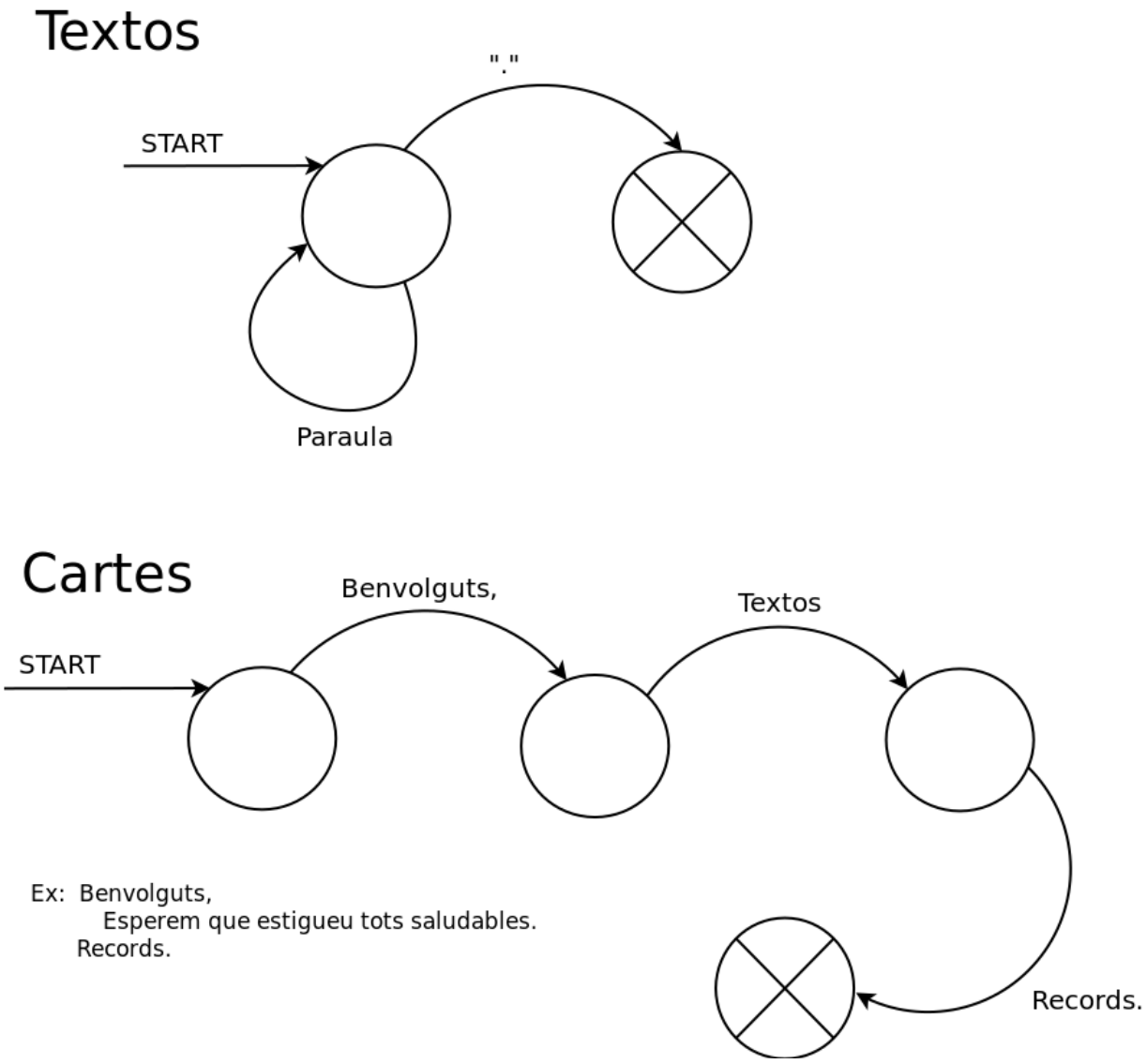

Figura 3.2: Exemple ATN Cartes utilitzant ATN Textos

4. Desenvolupament del llenguatge

En aquest punt explicarem el desenvolupament del llenguatge, quin software hem escollit i per quins motius, el disseny modular que hem portat a terme i la implementació.

4.1. Tria de les eines

Com hem anat comentant al llarg del projecte, volem que l'execució del llenguatge sigui la més òptima possible, a més a més, que tenim el Freeling com a exemple de projecte al qual hem d'incorporar el nostre treball. Tenint en compte que el Freeling està implementat en C++, és un llenguatge compilat, per tant el temps d'execució és millor que un interpretat i és un llenguatge força utilitzat en projectes d'aquest àmbit, concloem que implementarem tot el nostre projecte amb C++.

Un altre punt que hem de tindre en compte a l'hora de triar les següents eines que farem servir en el treball, és que si incorporem aquest al Freeling, aquest últim és un Open Source, per tant qualsevol llibreria o software que utilitzem en la implementació també hauria de ser-ho, ja que si no podríem arribar a tindre algun problema amb les llicències.

Tenint això present, procedim amb la tria de com implementarem tota la part del lèxic i la gramàtica, ja que la semàntica implementarem la nostra pròpia classe intèrpret, però per als altres dos camps ja hi ha tota una col·lecció de codi lliure del quan ens podem servir.

Ens decantem per Bison[4] i Flex[5], els quals treballen conjuntament, el primer s'encarrega de la gramàtica mentre que el segon del lèxic. Ens decantem per aquests per ser els més utilitzats que compleixen que es poden compilar a C++ i són open source.

Cal dir que abans de decantar-nos per l'ús de Bison i Flex vam estar estudiant altres possibilitats, com són Bison++ o Bisonc++ i Flex++ o Flexc++, les quals en una primera lectura de les seves finalitats semblava que es compenetrarien de millor de manera amb un projecte en c++ com és el nostre cas. Però finalment vam veure que aquests altres projectes eren derivacions del Bison i moltes funcionalitats que disposa aquest, no les disposen les seves derivants, funcionalitats que per a nosaltres eren essencials per a dur a terme la realització del nostre llenguatge, per aquest motiu finalment vam triar l'ús de Bison i Flex.

4.2. Disseny modular

En aquest punt parlarem dels diferents mòduls que hem utilitzat. Primer de tot llistarem els principals elements que hem de tindre en compte.

Comencem per aquells que ja hem esmentat anteriorment, tindrem el mòdul de l'escàner, el qual estarà implementat amb el Flex, aquest es comunicarà directament amb l'encarregat de les regles gramaticals, el parser, el qual estarà confeccionat amb el Bison.

---

[4] Bison és un generador sintàctic de propòsit general que converteix una gramàtica lliure de context anotada en un LR determinista o analitzador LR generalitzat (GLR) que empra taules d'analitzador LALR. D'aquest s'han derivat altres projectes, com Bison++ o Bisonc++

[5] Flex és el generador analític lèxic més ràpid. És una eina per generar escàners, és a dir, programes que reconeixen patrons lèxics en el text.

L'altre gran mòdul que ens queda és el de l'intèrpret, en el qual tindrem els tres conjunts d'elements principals que hem esmentat anteriorment, les variables globals, les funcions i els ATN's. Serà el parser qui s'hagi d'encarregar d'anar omplint aquestes tres estructures. També veiem que l'òptim és que aquests conjunts siguin diccionaris, ja que cada element tindrà un identificador i informació associada.

Per tant, ens hem de plantejar quins mòduls representaran cada un d'aquests tres elements diferents. La conclusió a la qual vam arribar és la de què, per una part, les funcions les podem representar mitjançant un arbre sintàctic, el qual rep el nom de Abstract Syntax Tree (AST). Aquests s'utilitzen molt a l'hora de confeccionar un llenguatge de programació, són molt útils per a representar tot tipus de subrutines. Veiem llavors que també els podem utilitzar per a representar el codi a executar en els estats dels ATN's.

La llibreria que utilitzàrem per a confeccionar l'arbre serà una que ja és implementada el projecte del Freeling, la qual s'anomena *tree*. Per la nostra part només ens caldrà implementar el node d'aquest arbre.

El mòdul del node és en veritat molt simple, tenint una cadena de caràcters que conté la informació del token determinat, el qual mai tindrà valor nul, i a més, també podem emmagatzemar la informació rellevant a aquest token, sigui un booleà, enter, real o una cadena de caràcters.

Prosseguim amb el mòdul que representarà de les xarxes de transició augmentades, aquest contindrà un llistat d'identificadors que farà referencia als estats inicials, i un altre de molt semblant que ens servirà per contenir els estats finals, a més a més, tindrà una cadena de caràcters per emmagatzemar l'identificador del mateix ATN. Per terminar la part més important, un conjunt que representarem fent ús d'un diccionari que ens faci servei per a poder contenir tots els estats.

Per últim ens queda el mòdul de les variables globals. Al cap i a la fi aquest hauria de ser el mateix que utilitzarà l'intèrpret per a gestionar les variables i estructures de dades locals, ja que no són diferents. Aquest mòdul, el qual anomenarem data, ha de poder contenir tots els diferents tipus de dades possibles al llenguatge: booleans, enters, reals, cadenes de caràcters, vectors de data i diccionaris de data. També haurà de tindre implementades totes les funcions necessàries per a tractar amb la informació que conté, per exemple, retornar un valor concret del diccionari a partir de la seva key, o fins i tot, modificar el mateix diccionari, sigui afegint o eliminant dades.

4.3. Implementació

En els següents punts explicarem la implementació del treball, en alguns punts profunditzarem més que en altres. Per exemple, l'intèrpret serà el mòdul que més importància li donarem, ja que és on recau la complexitat de major magnitud.

4.3.1. Lèxic

Aquesta secció basicament tracta d'establit els tokens del nostre llenguatge de programació, bàsicament són les paraules que seran aptes per a aquest.

En l'àmbit d'implementació és de les parts més simples, ja que en gran part només hem d'indicar quina paraula clau és cada token, paraules clau o símbols com per exemple: if, while, for, or, and, +, -, *, &&, etc.

Per una altra banda també definim com s'han d'expressar els valors, per exemple els nombres reals consten d'un dígit seguit d'un punt i un altre digito, mentre que les cadenes de caràcters s'han de contenir entre comelles, i d'igual manera amb qualsevol altre valor que vulguem, com poden ser els booleans o els enters.

Per terminar la part del lèxic només comentar que és en aquesta secció on també s'implementen els comentaris del llenguatge, en el nostre cas, en detectar els símbols “/*” ometrà qualsevol caràcter fins a trobar “*/”.

4.3.2. Gramàtica

Passem a un punt una mica més complex, una vegada ja tenim traduïdes les paraules clau, símbols i altres en tokens, podem treballar amb aquest per a fer les nostres regles gramaticals, les quals ens serviran per definir l'ordre en què s'han de trobar aquests perquè tinguin un sentit en el llenguatge.

Mostrarem un parell d'exemples simples, per tal d'il·lustrar com és la gramàtica en el nostre llenguatge, comencem amb un exemple d'una de les regles gramaticals més senzilles:

```
// if-then-else (else if & else are optionals)
ite_stmt    : IF OPARENTHESIS expr CPARENTHESIS block_instructions else_if else
```

Codi 2: Regla gramatical if then else

On les paraules en majúscules són tokens i les minúscules regles gramaticals. Primer de tot trobem la definició de la regla gramatical que estem definint (ite_stmt), la qual s'inicia amb el token “IF”, seguit d'una expressió emmarcada entre uns parèntesis seguit de dues regles finals, “else_if” i “else”.

Aquestes altres dos regles, a diferència del “ite_stmt" poden ser regles buides, veiem l'exemple del else:

```
// else
else    : // nothing
            {
                $$ = new tree<ASTN*>(new ASTN(L"ELSE"));
            }
        | ELSE block_instructions
            {
                tree<ASTN*>* t = new tree<ASTN*>(new ASTN(L"ELSE"));
                tree<ASTN*>* body = $2;

                t->add_child(*body);
                $$ = t;
            }
        ;
```

Codi 3: Regla gramatical else

Observem com els diferents casos se separen mitjançant el símbol “|”, el primer cas és buit, mentre que el segon consta del token else seguit d'un bloc d'instruccions. Parlem ara del codi que trobem després de cada cas, aquest s'executarà si la regla determinada fa match, aquest ens permet anar construint l'estructura de dades de l'intèrpret mencionades anteriorment. El Bison ens proporciona les variables $i, on i representa quina posició de la regla gramatical fa referencia, per exemple en el cas de: ELSE block_instructions, equival a ELSE: $1 i block_instructions: $2.

Els valors que tenen els token es defineixen en la part del lèxic, els quals o bé no tenen cap valor com és el cas del “ELSE”, o bé en el cas de ser token que representen un valor, com podria ser el cas del token que representa un real, ens proporciona el valor d'aquest real.

Una vegada hem exposat les regles gramaticals, procedirem per explicar quina és l'estructura que hem escollit per al nostre llenguatge. Un programa consta d'una sèrie d'elements principals, els quals hi ha tres diferents:

1. *Variables globals*. Només és la definició d'aquestes, tant la inicialització com la utilització esdevé en els altres elements.

2. *Funcions*. Com el la gran majoria de llenguatges, el bon ús de subrutines pot ser tant pragmàtic per al programador com eficient a l'hora de l'execució. Aquestes consten d'una sèrie d'instruccions, com poden ser: “if then else”, “while”, “for”, “do while”, etc. Diverses d'aquestes poden utilitzar expressions, tota expressió ha de retornar un valor, alguns exemples són les operacions aritmètiques, crides a funcions, operacions booleanes.

3. *ATN*. Aquest és el principal element del nostre llenguatge, bàsicament segueix la mateixa estructura que hem explicat en punts anteriors, on l'execució d'un estat és d'igual manera que en les funcions i les transicions senzillament avaluen una expressió la qual, si és certa, procedeix a l'estat determinat.

4.3.3. Semàntica

Finalment arribem a la part més interessant de la implementació, l'intèrpret, el qual és l'encarregat de la semàntica del llenguatge. Aquest és també el mòdul que s'haurà d'instanciar quan es vulgui fer d'ús d'aquest projecte, per tant començarem per explicar les funcions creadores que disposa aquesta classe, i així anirem procedint per les funcions públiques més importants.

4.3.3.1. Funcions creadores

Hi ha dos tipus diferents de creadores, la més recomanable d'utilitzar és la que requereix que li passi per paràmetre el nom del fitxer on haurà d'estar el codi del programa del nostre llenguatge de programació.

```
Atn::Atn(std::wstring name) :
    m_atn(),
    m_func(),
    m_global(),
    m_scanner(*this),
    m_parser(m_scanner, *this),
    m_column(0),
    m_row(1),
    m_location(0)
{
    ifstream file( converter.to_bytes(name) );
    if (file) {
        switchInputStream(&file);
        parse();
        file.close();
    }
    else {
        throw runtime_error("Error reading the input file: " + converter.to_bytes(name));
    }
}
```

Codi 4: Creadora de l'intèrpret

Com veiem, primer inicialitza els valors del mòdul per defecte, per després accedir al fitxer del codi. Una vegada ha aconseguit el programa que s'haurà d'executar, canvia el canal d'entrada de l'escàner mitjançant la funció publica switchInputStream, ara ja podrem fer l'anàlisi gramatical del codi, pel que utilitzarem la subrutina parse, i ja tindrem el codi llest per a executar.

L'altra creadora que tenim és la creadora buida, la qual només inicialitza per defecte els valors de la classe. Per tant el canal d'entrada per al codi serà l'estàndard, si volem canviar-ho, haurem de cridar a la subrutina switchInpuStream. I per últim, haurem d'executar el parse. Aquestes dues últimes funcions, no fan gaire més que cridar a les rutines internes del Flex i Bison respectivament.

```
Atn::Atn() :
    m_atn(),
    m_func(),
    m_global(),
    m_scanner(*this),
    m_parser(m_scanner, *this),
    m_column(0),
    m_row(1),
    m_location(0)
{ }
```

Codi 5: Creadora de l'intèrpret buida

```
void Atn::switchInputStream(std::istream *is) {
    m_scanner.switch_streams(is, NULL);
    m_func.clear();
    m_global.clear();
}
```

Codi 6: Funció per canviar el canal d'entrada de l'escàner

```
int Atn::parse() {
    m_column = m_location = 0;
    m_row = 1;
    return m_parser.parse();
}
```

Codi 7: Funció per realitzar el anàlisi gramatical

4.3.3.2. Debug del parse

Per a poder comprovar que el parse s'ha efectuat correctament hem implementat una funció per escriure pel canal de sortida estàndard les tres diferents estructures que tenim a l'intèrpret. A continuació posem un petit exemple d'una sortida.

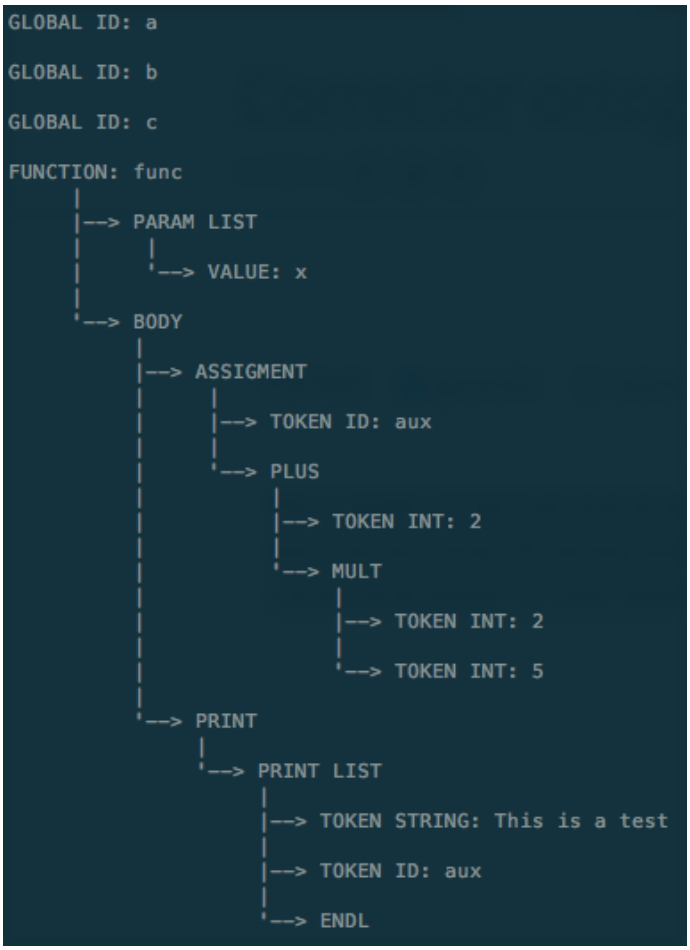

Figura 4.1: Exemple de sortida del debug del parser

4.3.3.3. Execució del ATN

En aquest punt ja estem preparats per a iniciar l'execució del nostre codi, per a això disposem d'una funció, a la qual ha de rebre per paràmetre el llistat de paraules a analitzar. A més a més, aquesta retornarà un llistat de tots els outputs detectats.

Un output consta de tres valors, el primer és la posició del text en la qual s'ha inicialitzat el matching, el segon és la posició següent en finalitzar el match, i per

últim el valor determinat pel codi que s'ha volgut retornar, el qual s'exigeix que sigui una cadena de caràcters.

```
def: run(vector<wstring> in), return vector<Output>:
    map<wstring, Data*> copy_global;
    for it = m_global.begin() to it != m_global.end() do:
        copy_global[it->first] = new Data(*(it->second));
        ++it;
    end

    vector<OutputInternal> v = executeAtn(L"main", copy_global, in, 0, 0);
    checkFinalOutput(v);

    return out;
end
```

Codi 8: Funció d'execució del programa

Aquesta subrutina procedirà a realitzar una còpia de les variables globals, ja que aquestes en ser punters poden veure's modificades, tot seguit buscarà el ATN amb identificador main, procedirà a executar-ho, farà un petit preprocessat a la sortida d'onada per la funció executeATN, el qual consisteix a traduir el llistat de OutputInternal a un llistat de Output.

Abans de tancar el punt de la implementació veiem necessari explicar el funcionament de la funció executeAtn, tot i així, en canvi creiem que podem obviar algunes de les subrutines que utilitza, per exemple, l'avaluació d'expressions o instruccions són funcions molt tedioses d'una complexitat no gaire elevada.

4.3.3.3.1. Funció per l'execució d'un ATN

```
def: executeAtn(wstring atnname, map<wstring, Data*> global, vector<wstring> in, int init, int act),
     return vector<OutputInternal>:

    // Get the data of the ATN
    ATNN atn = m_atn.find(atnname);
    // Inicializate output
    vector<OutputInternal> finalOutput;

    // Execute all the initial states at the start of the input
    vector<wstring> initials = atn.getInitials();
    vector<wstring> finals = atn.getFinals();
    map<wstring, tree<ASTN*>*> states = atn.getStates();
    for i = 0 to i < initials.size() do:
        wstring stateInit = initials[i];
        iterator st = states.find(stateInit);
        bool fin = finals.contain(stateInit);
        vector<OutputInternal> output = executeState(in, st, init, act, global, finals, states, fin);
        finalOutput.insert(output);
        ++i;
    end

    return finalOutput;
end
```

Codi 9: Funció d'execució d'un ATN

Per dur a terme l'execució d'un ATN requerim el nom del ATN a executar, per a poder buscar-lo en el conjunt de ATN's; el conjunt de variables globals, ja que tenim la possibilitat que el ATN que procedim a executar hagi sigut cridat per un altre ATN, per tant hem de rebre les variables globals actualitzades; el text a analitzar, podríem considerar el de tindre aquest valor conservat en el mòdul, però si el passem per paràmetre, podem facilitar l'execució en multiprocés; i per últim la posició de l'inici del match i l'actual, aquests dos les necessitem, ja que si aquest ATN resulta que l'estem executant des d'un altre ATN, hem de saber per quina posició del text ens trobem.

Per tant a continuació haurem de recórrer el llistat d'estats inicials i executar cadascun, per a això necessitarem obtenir la informació necessària per a cridar a la subrutina executeState, la qual ens retornarà les sortides de cadascun d'aquests estats.

De manera que guardarem tots els outputs, i aquest conjunt serà la sortida que haurem de retornar per a finalitzar aquesta funció.

4.3.3.3.2. Funció per l'execució d'un estat

Veiem que la clau del executeAtn és la subrutina executeState, això és a causa del fet que un ATN consta bàsicament de l'execució dels seus estats, per tant podríem dir que aquest és el cor del treball, a continuació l'exposem.

Primer de tot executarem la llista d'instruccions que té l'estat donat, seguidament comprovarem si aquest és final o no, ja que si ho és, haurem d'agafar el valor emmagatzemat en la variable reservada per a l'output (@), i si encara tenim paraules a analitzar, tornarem a executar el ATN main.

Una vegada ens hem encarregat de la part del codi de l'estat, ens centrarem a tractar les transicions d'aquest. Primerament comprovem que encara disposem de text per analitzar, si és així, per cada transició comprovem quina expressió s'ha de complir, tenim diversos casos:

1. *ATN*. Tenim el cas en què no sigui una expressió, sinó que la condició que s'ha de complir és l'execució d'un ATN. Per a això executarem el ATN determinat i tot seguit procedirem a executar l'estat indicat en aquesta transició per a cadascun dels outputs del ATN.

2. *NULL*. Podem trobar-nos amb què l'expressió sigui la cadena de caràcters "NULL", de ser així executarem el següent estat sense analitzar la paraula actual, de manera que sigui aquest nou estat al qual cridem l'encarregat d'analitzar-la. Es pot donar el cas en què l'expressió sigui una condició booleana tal que comenci de la següent manera: "NULL &&", de ser així, el cas és molt semblant al que acabem de mencionar, exceptuant que només es pot executar el següent estat si l'altra part de l'expressió booleana s'avalua a "true".

3. *Expressió booleana*. L'últim cas que ens queda és el que sigui una expressió booleana simple, també s'admeten funcions a condició que retornin un valor booleà.

Una vegada hem tractat totes les transicions, ens queda comprovar si efectivament, alguna d'aquestes ha aconseguit cridar a algun altre estat, en el cas negatiu, tornarem a executar el ATN main tenint en compte que la paraula actual ha sigut analitzada.

4.3.4. Altres mòduls

Hem volgut fer aquest últim punt per a explicar els altres mòduls dels quals no hem parlat, ja que són molt simples, com poden ser la classe del ATN o AST Node. La part d'implementació d'aquestes són elementals, només consten de donar els valors corresponents les dades que contenen i obtenir aquests mateixos, assegurant en tot moment la consistència del projecte. Per aquest motiu creiem que només requereixen una petita menció.

5. Ús del llenguatge

Creiem necessària una secció en la qual exposem com s'ha d'utilitzar de forma correcta el nostre llenguatge. Farem una explicació general, centrant-nos en els ATN's.

En l'exemple que exposem en la següent pàgina, veiem, totes les parts que hem anat explicant al llarg de la documentació. Primerament s'han d'introduir variables globals, funcions o ATN's, els dos primers elements són bàsics, com a molt esmentar la manera en què indiquem que una variable la rebem per referència, com és el cas de la variable "b" en la funció "f".

Tot seguit, explicarem el ATN, primer de tot indiquem el llistat d'estats inicials, tot seguit el de finals, i finalment els conjunts d'estats que componen el ATN en qüestió. Cada estat està compost per dos elements, per una banda el conjunt d'instruccions a executar i el de transicions, cal dir que aquests dos elements no tenen per què seguir l'ordre que acabem d'anomenar.

```
global glb1, glb2;

function f(a, &b) {
    print << a << " " << b << endl;
    return a == b;
}

atn main {
    initial states: start;
    final states: final;

    state start {
        transition {
            to final if (second);
            to final if ("NULL" && $.substring(0, 3) = "dos");
        }

        action {
            print << "START MAIN" << endl;
        }
    }

    state final {
        action {
            @ = "FINAL MAIN";
            print << "FINAL MAIN" << endl;
        }
        transition { }
    }
}

atn second {
    initial states: start;
    final states: final;

    state start {
        action {
            print << "START SECOND" << endl;
        }
        transition {
            to final if (f($,"uno"));
        }
    }

    state final {
        action {
            @ = "FINAL SECOND";
            print << "FINAL SECOND" << endl;
        }
        transition { }
    }
}
```

Codi 10: Exemple d'utilització del nostre llenguatge de programació

5.1. Exemple final

Per últim mostrarem un exemple útil en projectes sobre NLP, un ATN que analitza nombres. També mostrarem algunes solucions.

```
global num, _1D, _2D, _3D, _4D, _45D;
function init () {
   num = 0;
   _1D = {
      "uno": 1,
      "dos": 2,
      "tres": 3,
      "cuatro": 4,
      "cinco": 5,
      "seis": 6,
      "siete": 7,
      "ocho": 8,
      "nueve": 9
   };
   _2D = {
      "diez": 10,
      "once": 11,
      "doce": 12,
```

```
    "trece": 13,
    "catorce": 14,
    "quince": 15,
    "dieciséis": 16,
    "diecisiete": 17,
    "dieciocho": 18,
    "diecinueve": 19,
    "veinte": 20,
    "veintiuno": 21,
    "veintidós": 22,
    "veintidos": 22,
    "veintitrés": 23,
    "veintitres": 23,
    "veinticuatro": 24,
    "veinticinco": 25,
    "veintiséis": 26,
    "veintiseis": 26,
    "veintisiete": 27,
    "veintiocho": 28,
    "veintinueve": 29
};
```

```
_3D = {
   "treinta": 30,
   "cuarenta": 40,
   "cincuenta": 50,
   "sesenta": 60,
   "setenta": 70,
   "ochenta": 80,
   "noventa": 90
};
_4D = {
   "ciento": 100,
   "doscientos": 200,
   "trescientos": 300,
   "cuatrocientos": 400,
   "quinientos": 500,
   "seiscientos": 600,
   "sietecientos": 700,
   "ochocientos": 800,
   "nuevecientos": 900
};
```

```
   _45D = {
      "cien": 100
   };
}

function func (b, n) {
   if (b.contain(n)) {
      num = num + b[n];
   }
   return b.contain(n);
}

atn ATN3D {
   initial states: start;
   final states: D1, D2, D3, D4;

   state start {
      action { }
      transition {
         to D1 if (func(_1D, $));
         to D1 if (func(_2D, $));
         to D1 if (func(_45D, $));
```

```
      to D3 if (func(_3D, $));
      to D4 if (func(_4D, $));
   }
}

state D1 {
   action {
      @ = num;
   }
   transition { }
}

state D3 {
   action {
      @ = num;
   }
   transition {
      to D35 if (true);
   }
}
```

```
   state D35 {
      action { }
      transition {
         to D1 if (func(_1D, $));
      }
   }

   state D4 {
      transition {
         to D1 if (func(_1D, $));
         to D1 if (func(_2D, $));
         to D3 if (func(_3D, $));
      }
      action {
         @ = num;
      }
   }
}
```

```
atn main {
   initial states: S1;
   final states: F1;

   state S1 {
      action {
         init();
      }
      transition {
         to M1 if (ATN3D);
         to F1 if ($ == "mil");
      }
   }

   state M1 {
      action { }
      transition {
         to F1 if (num != 1 && $ == "mil");
      }
   }
```

```
   state F1 {
      action {
         if (num == 0) {
            num = 1;
         }
         num = num * 1000;
         @ = num;
      }

      transition {
         to F2 if (ATN3D);
      }
   }

   state F2 {
      action { }
      transition { }
   }
}
```

Codi 11: Programa per analitzar nombres

Introduirem la següent frase: “El otro día fui a comprarme tres cromos, y para mi sorpresa me vino uno de regalo. Ahora ya tengo trescientos cuarenta y ocho cromos en mi casa.”

Aquesta és la sortida que obtenim:

```
Output de l'entrada:
El otro día fui a comprarme tres cromos,
y para mi sorpresa me vino uno de regalo.
Ahora ya tengo trescientos cuarenta y ocho cromos en mi casa.

        INICI DE MATCHING:      15 - uno
        FINAL DEL MATCHING:     15 - uno
        INFORMACIÓ INDICADA:    1

        INICI DE MATCHING:      6 - tres
        FINAL DEL MATCHING:     6 - tres
        INFORMACIÓ INDICADA:    3

        INICI DE MATCHING:      22 - trescientos
        FINAL DEL MATCHING:     22 - trescientos
        INFORMACIÓ INDICADA:    300

        INICI DE MATCHING:      23 - cuarenta
        FINAL DEL MATCHING:     23 - cuarenta
        INFORMACIÓ INDICADA:    40

        INICI DE MATCHING:      25 - ocho
        FINAL DEL MATCHING:     25 - ocho
        INFORMACIÓ INDICADA:    8

        INICI DE MATCHING:      22 - trescientos
        FINAL DEL MATCHING:     23 - cuarenta
        INFORMACIÓ INDICADA:    340

        INICI DE MATCHING:      23 - cuarenta
        FINAL DEL MATCHING:     25 - ocho
        INFORMACIÓ INDICADA:    48

        INICI DE MATCHING:      22 - trescientos
        FINAL DEL MATCHING:     25 - ocho
        INFORMACIÓ INDICADA:    348
```

Figura 5.1: Sortida del programa d'analitzar nombres

Veiem com detecta tots els números existents en el codi, en canvi si cometéssim una falta d'ortografia, com per exemple dieciseis (sense accent), no el detectaria, ja que com hem vist prèviament, fa una comparació amb paraula a paraula amb el que hem indicat.

6. Treball futur

El nostre llenguatge de programació podria considerar-se que està terminat, ja que hem aconseguit complir el principal objectiu, executar ATN's. No obstant això, encara el podríem ampliar considerablement, a continuació fem un llistat de millores que hem considerat.

Una millora considerable seria la incorporació d'expressions regulars, ja que ens permetrien comparar el text a analitzar no només amb cadenes de caràcters.

També creiem que podria ser útil poder transmetre dades als estats que volem executar, d'igual manera que les funcions poden rebre paràmetres.

Hem observat que la tria de definir variables globals ens resulta força útil, ja que podem cridar un ATN dintre d'un altre, potser seria interessant establir la possibilitat que cada ATN pogués definir les seves pròpies variables globals.

En dissenyar el llenguatge de programació vam considerar que la definició de classes no seria del tot rellevant, al cap i a la fi recordem que és un llenguatge simple i se centra en l'execució de xarxes de transició augmentades, no obstant si podríem implementar que tant les estructures de dades com les variables poguéssim emmagatzemar funcions, d'aquesta manera un objecte del nostre llenguatge podria tindre variables, estructures de dades i funcions, no seria del tot una classe, però si ens podria fer servei per suplir la falta d'aquestes.

Un punt molt important és el d'afegir funcions natives, per exemple, hem incorporat el nostre llenguatge al Freeling, aquest té tota una sèrie de funcions que permeten obtenir més informació de cada paraula a analitzar, seria una manera de resoldre un problema que ja hem esmentat anteriorment, les faltes d'ortografia dels usuaris, entre d'altres. A l'hora de desenvolupar el nostre projecte hem tingut en compte aquesta possible millora, i cal dir que incorporar noves subrutines natives no és gaire complexa, tot depèn de la funció en qüestió que es vulgui implementar.

7. Conclusions

7.1. Valoracions d'objectius

Recordem que els objectius del projecte eren: realitzar un llenguatge de programació encarat a la implementació de ATN's, el qual ha de tant pràctic d'utilitzar per a usuaris poc habituats a programar com eficaç a l'hora de l'execució. A més a més, ha de ser factible incorporar-ho en altres projectes de NLP de major magnitud.

Veiem que el nostre llenguatge no sembla requerir uns amplis conceptes de programació per a utilitzar-ho, a l'hora que permet l'execució de ATN's, inclús uns dins dels altres, per tants podríem considerar com assolits aquests dos objectius.

Pel que fa a la incorporació a projectes més grans, com hem dit hem aconseguit incorporar aquest treball al Freeling, per tant queda assolit. Tot i que sí que seria convenient comprovar com s'adapta a altres projectes.

L'últim objectiu a tractar és el del temps d'execució, cal dir que esperàvem un temps d'execució millor, però aquest és un punt el qual sempre hem d'intentar continuar millorant. A continuació mostrem un parell d'exemples on indicarem els seus temps, observeu que l'única diferencia entre els dos textos és que mentre en el primer trobem "36 años" i en el segon "treinta y seis años", per tant en el segon haurà d'avaluar un nombre més.

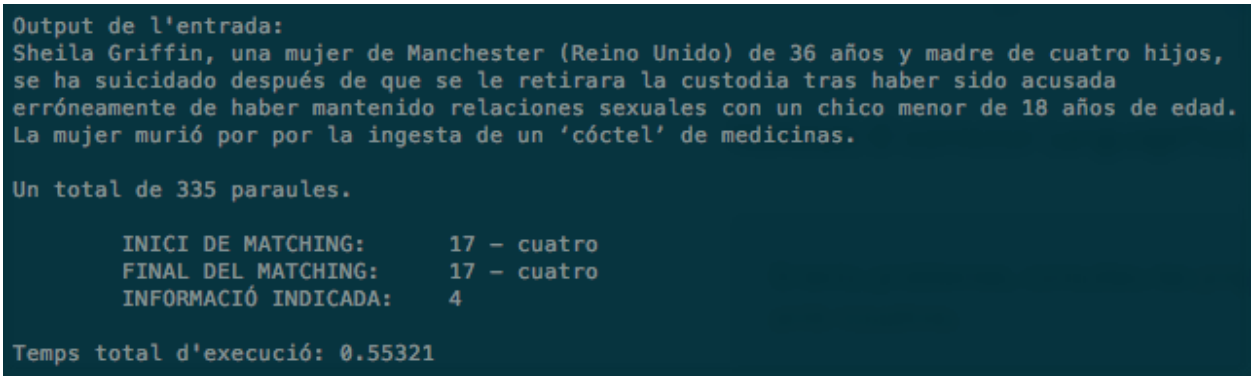

Figura 7.1: Exemple amb 36

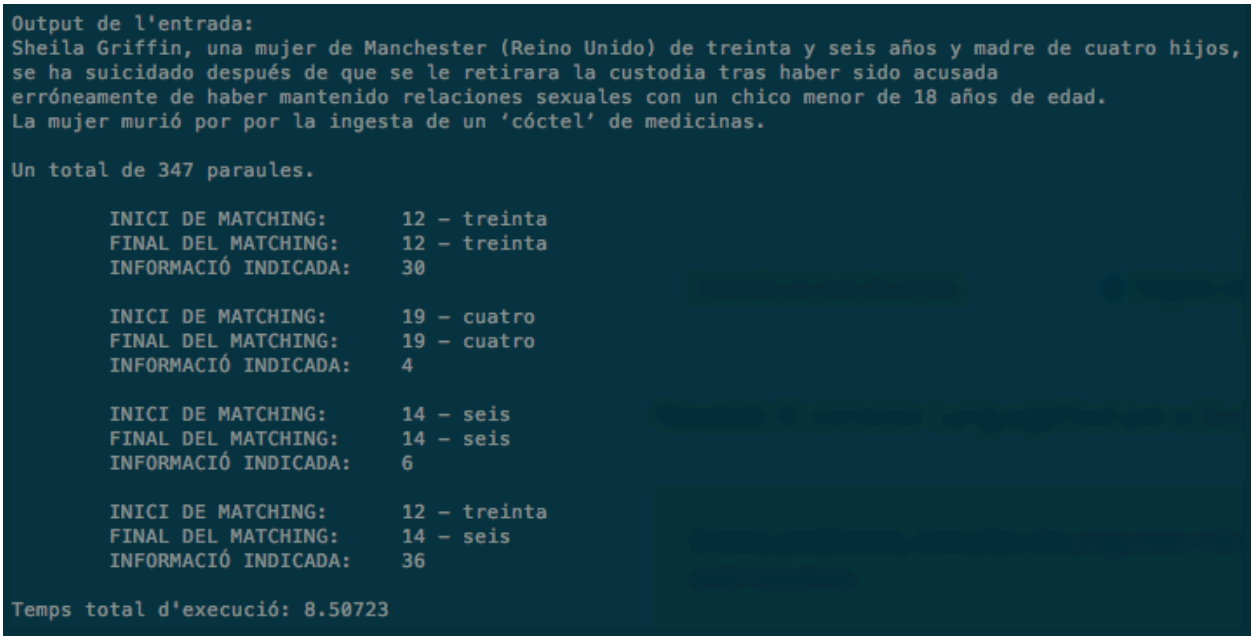

Figura 7.2: Exemple amb treinta y seis

7.2. Conclusions finals

Com a conclusions finals podem dir que el treball ha sigut força satisfactori, per una part hem confeccionat una eina que podria ser utilitzada per altres desenvolupadors, i així poder fer una petita aportació a aquest camp d'estudi.

Per part de l'estudiant podem dir que he après força sobre el processament del llenguatge natural, sobretot sobre les xarxes de transició augmentades. El qual ja era un tema que m'interessava i ha sigut una bona manera d'aprendre més.

Llicències de software

- **Bison:** A causa de a la seva utilització en el parser. Bison és programari lliure; es pot redistribuir-lo i/o modificar-lo sota els termes de la Llicència pública general de GNU publicada per la Fundació per al Programari Lliure; ja sigui la versió 3 de la Llicència, o (a la seva elecció) qualsevol versió posterior.

- **Flex:** A causa de la seva utilització en el escàner. Flex porta el copyright utilitzar per al software BSD, lleugerament modificat, perquè es va originar en el Lawrence Berkeley (no Livermore!) Laboratory, el qual opera sota un contracte amb el Departament d'Energia.

- **Freeling:** A causa de l'utilització de la llibreria tree, Freeling és codi obert, publicat sota la Llicència Pública General GNU Affero de la Free Software Foundation.

Glossari d'abreviatures

- **ATN:** Augmented Transition Network
- **NLP:** Natural Language Processing
- **AST:** Abstract Syntax Tree
- **ATNN:** ATN Node

Referencies

[1] Freeling, “Freeling - homepage” URL: http://nlp.lsi.upc.edu/freeling/index.php?option=com_content&task=view&id=25&Itemid=62.

[2] Flexc++, “Flexc++ - User guide” URL: https://fbb-git.github.io/flexcpp/manual/flexc++.html.

[3] Manpages, “Flexc++ - manual” URL: http://manpages.org/flexc.

[4] Bison, “Bison - manual” URL: https://www.gnu.org/software/bison/manual/bison.html.

[5] Bisonc++, “Bisonc++ - User guide” URL: https://fbb-git.github.io/bisoncpp/manual/bisonc++.html.

[6] Progtools, “How to use C++ with Bison” URL: http://www.progtools.org/compilers/tutorials/cxx_and_bison/cxx_and_bison.html.

[7] Jonathanbread, “Flex and Bison in C++” URL: http://www.jonathanbeard.io/tutorials/FlexBisonC++.

[8] Aquamentus, “Flex and Bison manual” URL: http://aquamentus.com/flex_bison.html#3.

[9] Gnuu, “Writing Your Own Toy Compiler Using Flex, Bison and LLVM” URL: http://gnuu.org/2009/09/18/writing-your-own-toy-compiler/

[10] Department of Information Systems Faculty of Information Technology Brno University of Technology Boˇzetechova 2, 612 00 Brno, CZ, “Augmented Transition Networks” URL: http://www.fit.vutbr.cz/~rudolfa/grants.php?file=%2Fproj%2F533%2Ffmnl03-atn.pdf&id=533.

[11] Bilal M. Bataineh, Emad A. Bataineh , “An Efficient Recursive Transition Network Parser for Arabic Language” *Proceedings of the World Congress on Engineering 2009 Vol II WCE 2009*, July 1 - 3, 2009, London, U.K.