El sistema Axis i el seu ús a l'entorn Excalibur

Carles Moreno
Núria Montolío
Xavier Franch

Report LSI-94-6-T

Títol

El sistema Axis i el seu ús a l'entorn Excalibur

Autors

Carles Moreno, Núria Montolío, Xavier Franch

Resum

El llenguatge de programació multiparadigma Merlí combina els estils equacional i imperatiu en una única notació. L'entorn d'execució Excalibur està dissenyat per suportar l'execució de programes Merlí i, per això, ha de tenir un sistema de reescriptura capaç de manipular els mòduls especificats. En aquest report es presenta un estudi sobre la integració del llenguatge i del sistema de reescriptura Axis (construïts als laboratoris de HP a Bristol) dins d'Excalibur com eina d'execució d'especificacions Merlí interpretades amb semàntica inicial.

EL SISTEMA AXIS I EL SEU ÚS A L'ENTORN EXCALIBUR[1]

Es presenta en aquest report un estudi sobre el comportament del llenguatge i del sistema de reescriptura Axis com suport per a l'execució d'especificacions Merlí dins de l'entorn de programació Excalibur.

L'entorn **Excalibur** està dissenyat per suportar un mètode de disseny de programes conegut com **prototipatge**, en el qual un programa és el resultat d'una seqüència de versions preliminars o **prototipus**, que poc a poc s'aproximen a la implementació final. Hi ha diverses estratègies de prototipatge; una d'elles consisteix a obtenir els programes per una aplicació successiva d'etapes de refinament començant a partir d'una especificació formal (possiblement incompleta) del problema, requerint que el prototipus existent a cada moment sigui executable. Excalibur està orientat a aquest cicle de vida [Fra92] el qual té algunes conseqüències; en destaca la necessitat de disposar de dues notacions diferents (però fortament interrelacionades) per especificar el problema i per implementar-lo i d'aquí sorgeix el llenguatge **Merlí** [FBB93]: Merlí és un llenguatge d'especificació, d'una banda, i un llenguatge d'implementació, de l'altra, i ofereix prou construccions incorporades com per relacionar ambdues parts. Per això es pot classificar Merlí com llenguatge **multiparadigma**. Com a nexe d'unió d'aquests dues vessants actua el concepte de **tipus abstracte de dades**, que és la unitat de disseny de les aplicacions; els tipus abstractes de dades s'encapsulen en **universos**, que s'anomenen **universos d'especificació** quan contenen l'especificació del tipus abstracte o bé **universos d'implementació** quan en contenen una implementació. Els universos d'especificació s'expressen usant equacions per assertar propietats mentre que els universos d'implementació estan escrits en estil imperatiu, recordant en molts aspectes als llenguatges modulars com ara Ada.

En aquest report es presenta una proposta concreta per a l'execució d'especificacions que es basa en una traducció dels universos d'especificació Merlí a un llenguatge anomenat **Axis** [Axis88], executable dins d'un sistema de reescriptura amb el mateix nom. Axis és un producte dissenyat i implementat al *Information Systems Centre* dels laboratoris HP a Bristol; el llenguatge permet especificar tipus abstractes de dades usant equacions. La seva similitut amb la part equacional de Merlí i l'amabilitat de HP en proporcionar gratuïtament no només el sistema i la documentació pertinent sinó també una estació de treball, va decidir als integrants del projecte Excalibur a estudiar la viabilitat de la integració d'Axis dins de l'entorn i, posteriorment, a implementar un traductor de Merlí a Axis. En el report es dóna una breu explicació dels dos llenguatges, es mostra el sistema de reescriptura, es presenta el mètode general de traducció acompanyat d'uns quants exemples d'especificacions Axis obtingudes a partir d'especificacions Merlí i finalment es mostra l'arquitectura del traductor.

[1] Aquest treball està parcialment recolçat per la beca TIC92-0667 de la CICYT.

1. Merlí com a llenguatge d'especificació

Merlí permet especificar equacionalment el comportament dels tipus abstractes de dades usant algunes construccions bàsiques que poden combinar-se per formar universos, que són les unitats atòmiques d'encapsulament del llenguatge. A continuació, es mostren aquests constructors sense entrar massa en detall; podeu consultar el report del llenguatge per a una descripció més acurada [FBB93]:

- Definició de nous gèneres, operacions i propietats: per a cada nou tipus que es vulgui definir, es dóna el seu nom a la clàusula "tipus", el nom i l'aritat de les seves operacions a la clàusula "operacions" i les propietats que compleixen aquestes operacions a les clàusules "equacions" i "errors". Alguns dels tipus i/o de les operacions es poden prefixar amb la paraula clau "privat" o "privada", que els oculta als universos usadors; de la resta, en direm símbols **visibles** o exportats. Les propietats no errònies s'expressen en estil equacional, essent les equacions parells de termes amb variables; en general, les equacions són condicionals, essent les condicions termes booleans. La lògica per expressar equacions permet associar semàntica inicial a les especificacions i, per això, es pot emprar un sistema de reescriptura per manipular termes, tal com es descriurà a la resta del report.
  Les propietats errònies es tracten de manera especial: seguint les idees de [ADJ78], s'introdueix implícitament un valor erroni a cada tipus en definició; tots els termes que expressen propietats errònies s'identifiquen i s'encapsulen a la clàusula especial "errors" i la resta de propietats poden suposar-se lliures d'error. A més, els valors d'error es propaguen.
- Ús d'universos ja existents: un univers U pot servir-se'n d'un altre univers V mitjançant la clàusula "usa", que introdueix dins U els símbols exportats per V. A més, el conjunt de símbols exportats de V passa a formar part del conjunt de símbols exportats de U, llevat que l'ús de l'univers V es precedeixi de la paraula clau "privat".
- Reanomenament de símbols amb la clàusula "reanomena", per llegibilitat o en parametritzar.
- Ocultació de símbols: mitjançant la clàusula "amaga", per ocultar alguns símbols als usuaris de l'univers en curs o bé per redefinir-los.
- Parametrització i instància: el comportament d'un tipus pot especificar-se en funció d'un o més paràmetres formals de manera que no s'està definint un tipus concret sinó una família de tipus semblants. Els paràmetres s'encapsulen en uns universos especials anomenats "de caracterització" (a la seva capçalera apareix la paraula clau "caracteritza") on s'especifica, per a cadascun d'ells, si és un gènere o una operació; en el segon cas, si s'escau, es defineixen les propietats que ha de complir. Els noms dels universos de caracterització apareixen a la capçalera de l'univers parametritzat (també dit **genèric**), tancats entre parèntesis i precedits opcionalment per un identificador. Aquest identificador serveix per prefixar les referències als símbols i és indispensable quan un mateix univers de caracterització apareix a la capçalera més d'un cop.
  Quan es vol definir un tipus concret a partir de la definició parametritzada s'efectua una **instància** de l'univers genèric, la qual cosa significa associar als paràmetres formals uns paràmetres reals que han de complir les propietats establertes en els universos de caracterització corresponents. Generalment, després de la instància es duen a terme alguns reanomenaments de símbols. La instància pot ser privada o no, depenent de si es volen exportar els símbols instanciats o no.

Hi ha d'altres mecanismes a Merlí que no s'introdueixen per no ser necessaris en la lectura d'aquest report.

A la fig. 1 apareix una especificació genèrica per a una variant de les piles on, en lloc d'una operació per consultar el cim, hi ha un predicat que esbrina si un element és o no és dins la pila. El paràmetre formal *elem* es defineix com gènere a l'univers de caracterització *ELEM*; la igualtat sobre els elements existeix implícitament (es el mecanisme de deducció equacional) i s'usa a les equacions condicionals.

```
univers PILA (ELEM) defineix
    usa Bool
    tipus pila
    operacions
        init: → pila
        push: pila elem → pila
        pop: pila → pila
        existeix?: pila → bool
    error pop(init)
    equacions
        var p: pila; v, w: elem fvar
        pop(push(p, v)) = p
        existeix?(init, v) = fals
        [v = w] ⇒ existeix?(push(p, v), w) = cert
        [¬ (v = w)] ⇒ existeix?(push(p, v), w) = existeix?(p, w)
funivers
```

```
univers ELEM caracteritza
    tipus elem
funivers
```

Fig. 1: especificació genèrica de les piles i caracterització dels seus elements.

2. El llenguatge d'especificació Axis

El llenguatge Axis permet especificar equacionalment tipus abstractes de dades i algoritmes sense concurrència, de manera similar a Merlí, Pluss [PLUSS84] i OBJ2 [OBJ85], però emfasitzant la inicialitat i, per tant, l'executabilitat de les especificacions. La interpretació de les equacions es fa mitjançant la reescriptura condicional de termes. El llenguatge és modular i presenta, a l'igual que Merlí, mecanismes de construcció d'especificacions complexes a partir de components més simples: parametrització i instanciació de mòduls, reanomenaments, compartició, còpia i unió de sub-especificacions.

Les especificacions poden ser de tres tipus: especificacions no parametritzades, especificacions parametritzades o especificacions de caracterització; les dues primeres s'encapsulen en mòduls encapçalats per la paraula clau "SPEC" i les últimes en mòduls encapçalats per "PROPS" (per abreujar, els anomenem **mòduls SPEC** i **mòduls PROPS**, respectivament).

2.1 Mòduls SPEC no parametritzats

Els mòduls SPEC serveixen per especificar un tipus de dades determinat i venen delimitats per les paraules clau "SPEC" i "ENDSPEC". La paraula "SPEC" va seguida de l'identificador del mòdul. Sintàcticament, un mòdul SPEC senzill, que no depèn de cap altre, té la forma (v. [Axis88] per a la descripció BNF):

SPEC <id del mòdul>
SORT(S) <identificador/s del gènere/s> {els singulars i els plurals són equivalents}
OP(S) <declaració de les operacions>
FORALL <declaració de les variables>
AXIOMS: <equacions>
ENDSPEC

2.1.1 La signatura

La signatura està formada per un o més gèneres i diverses operacions. Un gènere és un identificador per al conjunt de valors que pertanyen a un tipus de dades i pot coincidir amb l'identificador del mòdul SPEC. Una operació és l'identificador d'una funció, juntament amb el nombre i el tipus dels seus paràmetres i el tipus del resultat (que és únic).

En tot mòdul podem distingir entre la signatura **posseïda**, que és aquella part a la qual els axiomes del mòdul poden canviar el significat i la signatura **heretada**, que no es local al mòdul i tal que els axiomes no poden canviar-li el significat. En el cas de mòduls senzills, la signatura posseïda correspon als gèneres i operacions definits al propi mòdul i la signatura heretada és el gènere intern "bool" i les operacions booleanes definides en el mòdul intern "Boolean".

Les operacions poden seguir la notació prefix o misfix. El caràcter "_" serveix per indicar la posició dels arguments. Després del caràcter ":" s'han d'indicar els gèneres als quals corresponen cadascun dels arguments del domini (llevat del cas d'operacions constants -sense cap argument-) i després del caràcter " -> " el gènere al què correspon el resultat. Es pot declarar més d'una operació per línia. Alguns exemples en són:

```
op1 _ _ _ : s1 s2 s3 -> s3
zero : -> nat                {és una constant}
_ and _ : bool bool -> bool
```

Per expressar les propietats d'associativitat i de commutativitat de les operacions binàries de signatura **S S -> S** es disposa dels atributs "ASSOC", "COMM" i "ASSOC COMM" que s'escriuen a continuació del gènere del resultat de la operació i entre parèntesi. Així, si declarem una operació * com associativa i commutativa, **_ * _ : S S -> S (ASSOC COMM)**, significa que * satisfà les equacions:

```
FORALL X, Y, Z : S
        X * Y = Y * X
        X * (Y * Z) = (X * Y) * Z
```

És important la utilització dels atributs en comptes de les equacions equivalents atès que en utilitzar-se com regles de reescriptura conduirien a càlculs sense fi.

Les especificacions Axis poden usar implícitament alguns gèneres i operadors pre-definits. Concretament, existeix un gènere "bool" amb les constants "true" i "false" i les operacions lògiques "and", "or" i "not", automàticament disponibles a totes les especificacions. A més, per a cada gènere S existeix una operació d'igualtat **_ == _ : S S -> bool** (corresponent a la deducció equacional) i també una operació condicional **IF _ THEN _ ELSE _ ENDIF** definida com:

```
IF true THEN s1 ELSE s2 ENDIF  = s1
IF false THEN s1 ELSE s2 ENDIF = s2
```

essent **s1** i **s2** variables de gènere **S**. En aquesta operació condicional s'avalua primer la condició i després només la branca apropiada.

2.1.2 La secció FORALL

La secció "FORALL" ha de contenir les declaracions de totes les variables que surtin a les equacions. No poden haver-hi dues variables amb el mateix identificador i els gèneres de les variables han d'estar declarats en el mateix mòdul "SPEC" o bé ser accessibles des d'altres mòduls. El format de la secció és:

FORALL <id var1>: <gènere var1> ... <id varn>: <gènere varn>

Les variables del mateix gènere poden agrupar-se separades amb ",".

2.1.3 La secció AXIOMS

Els axiomes o equacions defineixen el comportament de les operacions i poden ser condicionals. Una equació és un parell de termes del mateix gènere separats pel símbol " = " i quantificats universalment respecte de les variables que contenen: l'equació determina que, per a qualsevol valor de les variables, els valors denotats pel terme de l'esquerra i pel terme de la dreta són iguals. La part condicional (si existeix) comença amb la paraula reservada "IF" i ve seguida d'una expressió booleana (es a dir, que retorna un valor del gènere "bool"). A més, el terme esquerre ha de contenir com a mínim una operació i el terme dret i la condició no poden contenir variables que no surtin a la part esquerra.

La secció ve encapçalada per la paraula reservada "AXIOM" (o "AXIOMS") seguida per ":" i per les equacions. Per claredat i també per facilitar la depuració en compilar el mòdul, també es poden agrupar les equacions amb múltiples aparicions de la paraula "AXIOMS", etiquetades (l'etiqueta precedeix els missatges dels errors de compilació); en aquest cas, el format es:

```
AXIOMS <etiq_grup1>:
        <grup d'equacions-1>
AXIOMS <etiq_grup2>:
        <grup d'equacions-2>
...
ENDSPEC/ENDPROPS
```

Per exemple, donades les operacions sobre el tipus "integer":

```
_ + _: integer integer -> integer
_ * _: integer integer -> integer
_ ** _: integer integer -> integer
alfa _ _ _ : integer integer integer -> integer
```

i suposant que *x*, *y* i *z* són variables de tipus "integer", obtindríem:

```
AXIOMS for + :
        x + 0 = x
        x + y = y + x
AXIOMS for alfa: alfa x y z = (x + y) * z
AXIOMS for **: x ** y = 1  IF (y == 0)
```

Les equacions s'interpreten com parells ordenats terme esquerre -> terme dret, i s'utilitzen com regles per reescriure un terme en un altre substituint repetidament les parts esquerra per les parts

dreta corresponents, fins que s'obté un terme irreduïble que no coincideix amb la part esquerra de cap regla (l'anomenat terme **canònic**). Les equacions condicionals es tracten com regles de reescriptura condicionals on la substitució de la part esquerra per la part dreta només s'aplica si la condició es compleix.

2.2 Mòduls PROPS

Els mòduls PROPS serveixen per especificar els requeriments que han de satisfer els paràmetres reals d'un mòdul parametritzat: els paràmetres formals d'un mòdul SPEC sempre fan referència a mòduls PROPS (v. apartat 2.4). Sintàcticament, són similars als mòduls SPEC, substituint les paraules reservades "SPEC" i "ENDSPEC" per "PROPS" i "ENDPROPS", però semànticament no tenen una interpretació inicial sinó laxa i, per això, les àlgebres que formen el model no han de complir cap condició d'inicialitat.

A tall d'exemple, el mòdul pre-definit "Triv" especifica la classe de totes les àlgebres ja que que no imposa cap requeriment als paràmetres actuals excepte que hi hagi un gènere:

```
PROPS Triv
        SORT element
ENDPROPS
```

Incloent-hi una constant al gènere, especifiquem la classe de totes les àlgebres amb un conjunt portador no buit.

```
PROPS Triv2
        SORT element
        OPS c :-> element
ENDPROPS
```

Una classe molt útil és aquella dels tipus que presenten una relació d'ordre:

```
PROPS Preorder
        SORTS element
        OPS _ <= _ : element element -> bool
        FORALL e1, e2, e3: element
        AXIOMS:
                e1 <= e1 = true
                e1 <= e3 = true IF (e1 <= e2) and (e2 <= e3)
ENDPROPS
```

2.3 Jerarquització de mòduls: USING i INCLUDING

Els mòduls SPEC i PROPS es poden formar en base a d'altres mòduls mitjançant els constructors USING i INCLUDING. Amb USING es permet un accés "només de lectura" a una especificació existent, mentre que amb INCLUDING s'importa una especificació permetent-ne la modificació. Puntualitzant segons les definicions de signatura posseïda i heretada vistes anteriorment, un mòdul amb USING hereta tant la signatura posseïda com l'heretada de l'expressió de mòduls associada, mentre que un mòdul amb INCLUDING posseeix la signatura posseïda de l'expressió de mòduls i hereta la signatura heretada d'aquesta.

La sintaxi és:

SPEC <id_spec> / PROPS <id_props>
[USING <expressió de móduls>]
[INCLUDING <expressió de mòduls>]
...

Una expressió de mòduls por ser l'identificador d'un mòdul SPEC no parametritzat, la instanciació d'un mòdul parametritzat o l'aplicació de la unió (+) o reanomenaments (WITH) a expressions de mòduls.

Algunes propietats de les construccions USING i INCLUDING són:

- Transitivitat: si *A* usa *B* i *B* usa *C*, llavors *A* usa *C*.
- Un mòdul PROPS només pot usar i incloure mòduls SPEC.
- Els mòduls PROPS no poden ser usats ni inclosos per cap altre mòdul.
- La signatura d'un mòdul no es pot amagar ni parcial ni totalment i per tant sempre serà totalment visible a mòduls usadors via USING o INCLUDING (no existeixen operacions privades com a Merlí).

2.4 Parametrització i instància

La parametrització permet definir especificacions genèriques i, per tant, obtenir múltiples especificacions concretes amb instàncies d'un mòdul parametritzat. La sintaxi per a la definició d'un mòdul parametritzat és:

SPEC <id_mòdul> (<dp1> && <dp2> ... && <dpn>)

on <dpi> és la declaració del paràmetre formal, formada per <id_param>: <id_mòdul Props>.

Dins del mòdul parametritzat pot fer-se referència als gèneres dels paràmetres formals i a les seves operacions com si fossin gèneres i operacions definits al propi mòdul. En el cas de tenir múltiples paràmetres amb el mateix mòdul PROPS associat, la disambiguació de gèneres es fa mitjançant l'identificador del paràmetre segons el format: <id_param>'s <id_gènere>.

Per exemple, les tuples de dos elements es poden definir dins d'un mòdul parametritzat (essent "Triv" el mòdul PROPS presentat anteriorment):

SPEC Tupla (X: Triv && Y: Triv)
SORT tupla
OPS primop _ _ : tupla -> X's element
FORALL x1, x2: X's element; y: Y's element
...
ENDSPEC

La instància és el procès de substitució de cada paràmetre formal per una expressió de mòduls que satisfaci els requeriments del mòdul PROPS associat. La seva sintaxi és:

<id_mòdul_param.> (<mòdul1> FIT <assocs1> && ... <mòdul*n* > FIT <assocs*n* >)

on <mòdul*i* > pot ser una expressió de mòduls (habitualment un identificador de SPEC no parametritzat) o bé un identificador de paràmetre formal i on <assocs*i* > estableix la correspondència entre els gèneres i les operacions que són paràmetres reals i els corresponents paràmetres formals. Les correspondències se separen amb "," i és indiferent l'ordre en què es facin; sintàcticament, són de la forma: <id_parformal> AS <id_s_parreal>. Hi ha algunes correspondències per defecte (v. [Axis88]).

A l'exemple de les tuples, podem obtenir instàncies de tuples de dos elements del mateix o diferent tipus, perquè el mòdul "Triv" només requereix que l'expressió de mòduls que actuï com paràmetre actual tingui un gènere. Si volem tenir tuples amb el primer element de tipus natural i el segon booleà cal escriure:

```
SPEC TuplesNatBool
        USING Tupla (Natural FIT element AS nat && Boolean FIT element AS bool)
ENDSPEC
```

Per il·lustrar la parametrització amb operacions, suposem un mòdul per a les piles, com ara **SPEC Pila (X: Preorder)**; llavors, podríem fer una instància per obtenir les piles de naturals, associant una operació ">=" dels naturals a l'operació "<=" de **Preorder**:

```
SPEC PilaNat
        USING Pila (Natural FIT element AS nat,
                                    _ <= _ : element element -> bool AS _ >=_)
ENDSPEC
```

La instància d'un mòdul posseeix la unió de: 1) els gèneres declarats en el mòdul parametritzat; 2) les operacions declarades en el mòdul parametritzat (amb les aritats i els abasts reanomenats segons els paràmetres actuals); 3) la signatura posseïda per cada expressió de mòduls amb INCLUDING instanciada. Per altra banda, hereta la unió de: 1) la signatura posseïda i heretada dels paràmetres reals; 2) la signatura posseïda i heretada de la expressió de mòduls amb USING instanciada; 3) la signatura heretada de cada expressió de mòduls amb INCLUDING instanciada.

2.5 Unió de mòduls (+)

Es poden combinar dos mòduls per formar-ne un tercer utilitzant l'operació "+". La unió de mòduls és associativa i commutativa. El mòdul resultat posseeix una signatura buida i hereta la unió de les signatures posseïdes i heretades del tots els seus arguments.

Per exemple, el mòdul següent hereta les signatures posseïdes i heretades de "Natural" i d'una instància de "Parell":

```
SPEC Exemple2
        USING Natural + Parell(Natural FIT element AS nat)
        ...
ENDSPEC
```

2.6 Reanomenaments (WITH)

La clàusula WITH permet reanomenar els gèneres i les operacions d'una expressió de mòduls. La sintaxi és idèntica a la definida anteriorment a l'apartat d'instància amb l'etiqueta de <assocs*i* >. El resultat d'un WITH és un mòdul que posseeix la signatura posseïda de l'expressió de mòduls i hereta la signatura heretada de l'expressió, en els dos casos aplicant-hi els reanomenaments oportuns.

2.7 Sobrecàrrega i disambiguació

Axis suporta la sobrecàrrega de gèneres i d'operacions introduint el mateix identificador o declaració d'operació en diferents mòduls. L'ambigüitat es resol utilitzant l'identificador del mòdul posseïdor com a qualificador. Així, la notació **M's S** indica que **S** és un símbol posseït pel mòdul **M**. No cal qualificar un gènere o una operació en el mòdul que el posseeix; si la

referència és ambigua s'assumeix el gènere o la operació local. En resoldre l'ambigüitat entre operacions amb idèntics identificador, domini i abast el qualificador precedeix a tot el terme.

3. El sistema Axis

El sistema Axis [Arn88] està format per un compilador, un intèrpret per a la reescriptura de termes i una base de dades d'especificacions. La implementació està feta en Common Lisp i és executable sobre una estació de treball HP9000/300.

El sistema és bàsicament interactiu i es manipula a través d'una interfície amb l'editor Gnu Emacs o bé els entorns Lisp, Nmode o Poplog. A les proves efectuades, via Emacs, només s'ha pogut executar Axis des de la Korn Shell o la Bourne Shell, i és recomanable la utilització de XWindows durant el desenvolupament i la prova d'especificacions.

En realitzar la instal·lació, es creen sota el directori principal, anomenat "axis", quatre subdirectoris que contenen l'executable, alguns exemples, el fitxer LISP d'interfície amb Emacs, una llibreria amb les especificacions més comunes (piles, cues, llistes, etc.) i la documentació del sistema.

La seqüència d'accions més comuna en una sessió de treball correspondria a l'algorisme:

```
repetir
    repetir
        editar mòdul
        inserir mòdul a la base de dades
        compilar mòdul
    fins que el mòdul compili bé
    preparar la reescriptura
    repetir
        reescriure terme
    fins que es vulgui
fins que es dongui per acabada la sessió
```

La base de dades es divideix en **clusters**, que serveixen per agrupar els mòduls relacionats i permetre el desenvolupament separat de components d'una mateixa especificació. Les referències a clusters es fan mitjançant un identificador entre els caràcters "[" i "]". La base de dades conté inicialment els clusters "[BuiltIn]" (amb les especificacions "Boolean", "Natural" i "Identifier"), "[Library]" (buida) i "[User]" (buida; és el cluster inicial). És possible incloure un o tots els mòduls d'un cluster en un altre. Cal remarcar que la base de dades no és permanent i desapareix en acabar la sessió en curs (a aquest respecte, veure la comanda "AUTOLOAD").

3.1 Texts Axis

Els texts axis són els fitxers font per al sistema i es consideren dividits en **particions**. Cada partició comença amb una paraula reservada i acaba a la següent paraula reservada o bé al final del text. La partició actual és aquella on es troba el cursor en un moment donat. Tots els lexemes escrits entre "% " o "%%" i el fi de línia es consideren comentaris.

Les paraules reservades d'inici de partició (que denoten diferents tipus de particions) són "SPEC", "PROPS", "COMPILE", "SETUP", "REWRITE", "IN~CLUSTER", "AUTOLOAD"

i "INCLUDE". La càrrega d'una partició SPEC o PROPS té com a resultat la inserció del mòdul en el cluster actual de la base de dades. Si el mòdul ja existia, tots els mòduls depenents es marquen com a no-compilats per mantenir la consistència. La càrrega dels altres tipus de particions provoca l'execució de la comanda amb el mateix nom que el de la partició sobre la partició actual.

3.2 Comandes Axis

A continuació exposarem les comandes més rellevants que ofereix el sistema i el seu efecte:

- COMPILE nomcluster id_mòdul: compila el mòdul especificat emmagatzemant la seva representació interna en la base de dades. Es compilen automàticament i prèviament tots els mòduls sense compilar dels quals en depenguès.
- SETUP nomcluster id_mòdul: prepara la base de regles per a reescriptura respecte al mòdul especificat, que ha d'estar correctament compilat.
- REWRITE terme: reescriu el terme donat segons l'últim mòdul del què s'ha fet SETUP. A més, el sistema dóna el nombre de reescriptures que s'han fet i el temps d'execució. Dóna un missatge d'error si el terme és sintàcticament incorrecte o si no s'ha preparat cap mòdul. Dels mòduls PROPS o SPEC genèrics no es pot fer reescriptura.
- IN~CLUSTER nomcluster: el cluster especificat passa a ser el cluster actual, creant-lo si no existia.
- AUTOLOAD nomcluster nomdirectori: permet associar fitxers que contenen una única especificació amb mòduls d'un cluster. Per a tots els fitxers del directori especificat que tinguin extensió ".ax" es crearà una entrada com a mòdul amb el seu nom sense l'extensió en el cluster especificat. En utilitzar el mòdul per primer cop, es carregarà del fitxer.
- DELETE nomcluster id_mòdul: esborra el mòdul especificat de la base de dades marcant com a no-compilats tots els mòduls que en depenen.
- SHOW~DEPS nomcluster id_mòdul: mostra la llista de mòduls dels quals depèn el mòdul especificat i la llista dels mòduls que depenen d'ell.
- MODULES nomcluster: llista els noms i els estats dels mòduls existents al cluster especificat.
- INIT~DATA~BASE: inicialitza la base de dades de forma que contingui només les especificacions internes.

3.3 Interfície amb EMACS

L'arxiu LISP que actua d'interfície amb Emacs és el "gnuaxis.el" i pot modificar-se segons criteris propis. En general: per entrar al sistema, cal entrar ESC-x run-axis; per sortir del sistema, Ctrl-c Ctrl-c i per introduir comandes, Ctrl-c Ctrl-v. Sovint és més convenient i ràpid treballar amb combinacions de tecles que amb comandes o càrrega de particions que no siguin SPEC ni PROPS; les més destacades són: Ctrl-c Ctrl-i, càrrega de la partició actual; Ctrl-c Ctrl-c, compilació de la partició actual; Ctrl-c Ctrl-s, prepara la partició actual per a reescriptura; Ctrl-c Ctrl-m, llista els mòduls del cluster actual; Ctrl-x Ctrl-f, càrrega d'un fitxer a Emacs.

4. Traducció d'especificacions Merlí a Axis

La traducció de Merlí a Axis no ha estat massa dificultosa, ateses les semblances existents entre ambdós llenguatges d'especificació. Cal remarcar, però, alguns punts importants:

- L'ordre de les clausules és totalment estricte en Axis. Ens hem aprofitat d'aquest fet en tractar els errors, tal com s'explica a continuació.

- La declaració de les variables utilitzades a les equacions és obligatòria en Axis, mentre que és opcional a Merlí. Per això, a la traducció cal generar les variables necessàries.
- En Axis no es poden declarar símbols privats; totes les operacions i gèneres introduïts en un univers són accessibles des de qualsevol altre univers que l'usi. Per això, la invisibilitat es garanteix en el procès de traducció i no en el codi Axis resultant.
- En instanciar un univers genèric, Axis obliga a que el paràmetre real associat a una operació sigui també una operació, mentre que Merlí permet que sigui una expressió arbitrària correcta. En aquest cas, la traducció genera una nova operació constant que es defineix igual a l'expressió i que és la que s'associa al paràmetre formal.
- Axis presenta algunes restriccions a les relacions d'ús (per exemple, no es pot usar més d'un mòdul amb USING, i un univers PROPS no pot ser usat per cal altre) que no són a Merlí. Per això, de vegades és necessari modificar la distribució en universos en traduir una jerarquia.
- Axis no té tractament d'errors; per tant, s'han de generar les equacions corresponents. En concret, per a cada tipus abstracte *t* que intervé en una especificació, es generen dues operacions:

  error-*t*: $\rightarrow t$, que simbolitza el valor erroni del tipus, i

  correcte-*t*?: $t \rightarrow$ bool, que comprova la correctesa d'un valor del tipus.

  Llavors, per a cada terme *A* declarat com a erroni a l'especificació Merlí, el traductor genera l'equació Axis *A* = *error-t*; per altra banda, la resta d'equacions es protegeix convenientment mitjançant premises per evitar inconsistències. Degut a la importància de l'ordre d'aparició de les equacions en Axis, és imprescindible que apareguin primer les equacions d'error que les altres.

A la fig. 2 apareix la traducció a Axis de l'especificació Merlí per a les piles (v. fig. 1) on es reflecteixen aquests fets i d'altres no tan rellevants. Destaquem que alguna condició és redundant (per exemple, preguntar si *correcte-pila init*) però que apareix per l'automatisme del procès de traducció.

Cal remarcar algunes mancances de procès de traducció actualment existent. Per començar, les equacions Merlí es tradueixen a equacions Axis, les quals s'interpreten directament com a regles de reescriptura; caldria afegir, doncs, un algoritme de Knuth-Bendix per estudiar i, eventualment, solucionar la completitud i la terminació finita del sistema de reescriptura generat. En concret, durant aquest procés fóra necessari estudiar l'associativitat i commutativitat de les operacions i, si és el cas, eliminar les equacions que donen lloc a aquestes propietats i etiquetar les operacions com a tals en l'especificació Axis. Per últim, algunes facilitats addicionals de Merlí no descrites aquí (per exemple, els tipus escalars) no es tradueixen a Axis.

5. Exemples

Els següents exemples, traduccions d'especificacions en llenguatge Merlí tretes de [Fra91] i de [Fra93], pretenen il·lustrar algunes possibilitats i limitacions del sistema Axis com llenguatge d'especificació de tipus abstractes de dades i com sistema de reescriptura.

```
SPEC PILA (X0: ELEM)
    USING Bool
    SORTS pila
    OPS
        init: → pila
        push _ _: pila elem → pila
        pop _: pila → pila
        existeix? _ _: pila → bool
        error-pila: → pila
        correcte-pila? _: pila → bool
    FORALL p: pila; v, E0: elem
    AXIOMS:
        (pop init) = error-pila
        (pop (push p v)) = p IF (correcte-pila? (push p v))
        (existeix? init  v) = false IF (correcte-pila? init) and (correcte-elem? v)
        (existeix? (push  p v) E0) = true IF (v == w) and (correcte-pila? (push p v)) and
                                              (correcte-elem? v) and (correcte-elem? E0)
        (existeix? (push  p v) E0) = (existeix? p E0)
                                     IF (not (v == w)) and (correcte-pila? (push p v))
                                        and (correcte-elem? v) and (correcte-elem? E0)
        (push p v) = error-pila IF (not correcte-pila? p) or (not correcte-elem? v)
        (pop p) = error-pila IF (not correcte-pila? p)
        (existeix? p v) = error-bool IF (not correcte-pila? p) or (not correcte-elem? v)
        (correcte-pila? error-pila) = false
        (correcte-pila? init) = true
        (correcte-pila? (pop p)) = (correcte-pila? p)
        (correcte-pila? (push p v)) = (correcte-pila? p) and (correcte-elem? v)
ENDSPEC

PROPS ELEM
    USING Bool
    SORTS elem
    OPS
        error-elem: → elem
        correcte-elem? _: elem → bool
    AXIOMS:
        (correcte-elem? error-elem) = true
ENDPROPS
```

Fig. 2: traducció a Axis de l'especificació genèrica de les piles de la fig. 1.

A) Enriquiment de l'especificació interna "Bool", necessari per incorporar el tractament d'errors a especificacions posteriors i que, a més, introdueix l'operació d'implicació "=>". En aquest cas no és necesari utilitzar la clàusula USING per referenciar el mòdul que s'està enriquint.

```
SPEC MiBool
        OPS     _ => _: bool bool -> bool
                error-bool: -> bool
                correcte-bool? _: bool -> bool
        FORALL b, b0: bool
        AXIOMS:
                (true => true) = true IF (correcte-bool? true) and (correcte-bool? true)
                (true => false) = false IF (correcte-bool? true) and (correcte-bool? false)
                (false => b) = false IF (correcte-bool? false) and (correcte-bool? b)
                (b => b0) = error-bool IF (not correcte-bool? b) or (not correcte-bool? b0)
                (correcte-bool? ( b => b0)) = (correcte-bool? b) and (correcte-bool? b0)
                (correcte-bool? true) = true  (correcte-bool? false) = true
                ...
ENDSPEC
```

B) Es presenta l'especificació parametritzada del tipus "Conjunt" i, a continuació, dues instàncies (conjunt de naturals i conjunt de booleans). S'utilitza l'especificació "MiBool" tot just introduïda i es defineixen les operacions i els axiomes pertinents per efectuar el tractament d'errors. S'ha de remarcar que la reescriptura mitjançant l'equació de commutativitat d'"afegir" iniciaria un procés sense fi, i que l'alternativa d'utilitzar l'atribut COMM en la declaració de la operació no podria considerar el tractament d'errors.

```
SPEC Cjt ( X0: Elem )
        USING MiBool
        SORTS cjt
        OPS     crea:  -> cjt
                afegir _ _: cjt elem  -> cjt    (COMM)
                error-cjt: -> cjt
                correcte-cjt? _ : cjt -> bool
        FORALL  n, m: elem; s:cjt
        AXIOMS for afegir:
        (afegir (afegir s n) n ) = (afegir s n) IF (correcte-cjt?  (afegir s n)) and (correcte-elem? n)
        (afegir (afegir s n) m ) = (afegir (afegir s m) n )
                IF (correcte-cjt? (afegir s n)) and (correcte-elem? m)
        (afegir s n) = error-cjt IF (not correcte-cjt? s) or (not correcte-elem? n)
        AXIOMS for correcte-cjt?:
        (correcte-cjt? error-cjt) = false
        (correcte-cjt? crea) = true
        (correcte-cjt? (afegir s n)) =(correcte-cjt? s) and (correcte-elem? n)
ENDSPEC
```

El paràmetre formal "Elem" exigeix que el paràmetre actual tingui almenys un gènere i tres operacions anàlogues a "error-elem", "correcte-elem?" i "equ":

```
PROPS Elem
        USING MiBool
        SORTS elem
        OPS     error-elem: -> elem
                correcte-elem? _ : elem -> bool
                equ _ _ : elem elem  -> bool
        FORALL e0, e1: elem
        AXIOMS:
                (correcte-elem? error-elem) = false
                (equ e0 e1 ) = error-bool IF (not correcte-elem? e0) or (not correcte-elem? e1)
                ...
ENDPROPS

SPEC CjtBool
        USING Cjt (MiBool FIT elem AS bool, error-elem: -> elem AS error-cjt,
                                correcte-elem? _ : elem -> bool AS correcte-cjt? _ ,
                                equ _ _: elem elem -> bool AS  _ == _ )
        WITH cjt AS cjtbool
        OPS
                error-cjtbool : -> cjtbool
                correcte-cjtbool? _ : cjtbool -> bool
        AXIOMS:
                correcte-cjtbool? error-cjtbool = false
ENDSPEC
```

C) Especificació del tipus "Unió" amb dos paràmetres formals que tenen la mateixa definició i disambiguació mitjançant identificador. La instància "BoolNat" utilitza dos paràmetres actuals diferents i depèn dels mòduls "FNat" (que surt més endavant) i "Bool" (per l'ús del signe "+").

```
SPEC Unio2 (X : Triv && Y : Triv)
        SORT unio2
        OPS     aunio1 _ : X's element -> unio2
                aunio2 _ : Y's element -> unio2
                t1 _ : unio2 -> bool
                t2 _ : unio2 -> bool
                sel1 _ : unio2 -> X's element
                sel2 _ : unio2 -> Y's element
        FORALL x: X's element  y: Y's element
        AXIOMS:
                t1 (aunio1 x) = true     t2 (aunio2 y) = true
                sel1 (aunio1 x) = x      sel2 (aunio2 y) = y
ENDSPEC

SPEC BoolNat
USING FNat+Bool+Unio2 (Bool FIT X's element AS bool, FNat FIT Y's element AS natural)
WITH unio2 AS boolnat, t1 _ : unio2 -> bool AS bool?, t2 _ : unio2 -> bool AS natural?
ENDSPEC
```

D) Aquí es presenta l'especificació parcial parametritzada d'un graf dirigit i etiquetat. Els

paràmetres reals determinen els tipus dels vèrtexs i de les etiquetes. S'utilitza l'especificació de conjunt parametritzat "Set" de la llibreria Axis (veure [DoH88]), que s'instanciarà automàticament quan ho faci "Graf". Es resalta la necessitat de declarar separadament operacions amb idèntica aritat, domini i abast, la col·locació de comentaris per evitar la possibilitat de procés infinit amb la commutativitat d'afegeix, l'ús de les operacions implícites "==" i "IF_THEN_ELSE_ENDIF", i la instància implícita sense utilitzar assignacions amb FIT.

```
SPEC Graf (V: Vertex && A: Etiq)                PROPS Vertex
        USING Set (V)                                   SORT vertex
        WITH set AS cjtvertex                   ENDPROPS
        SORTS graf
        OPS     crea: -> graf
                afegeix _ _ _ _ : graf vertex vertex etiq -> graf
                esborra _ _ _ : graf vertex vertex -> graf
                exist-arest? _ _ _ : graf vertex vertex -> bool
                valor _ _ _ : graf vertex vertex -> etiq
                succes _ _ : graf vertex -> cjtvertex
                predec _ _ : graf vertex -> cjtvertex
                exist-cami? _ _ _ : graf vertex vertex -> bool
        FORALL v, v1, v2, v3, v4: vertex   e, e1,e2: etiq   g: graf
        AXIOMS for afegeix:
                % afegeix (afegeix g v1 v2 e1) v3 v4 e2 =
                        % IF (v1==v3) and (v2==v4) THEN afegeix g v1 v2 e2
                        %                          ELSE afegeix(afegeix g v3 v4 e2) v1 v2 e1
                        % ENDIF
        AXIOMS for esborra:
                esborra crea v1 v2 = crea
                esborra (afegeix g v1 v2 e) v3 v4 = IF (v1==v3) and (v2==v4)
                                                    THEN g
                                                    ELSE afegeix (esborra g v3 v4) v1 v2 e
                                                   ENDIF
        AXIOMS for exist-arest:
                exist-arest? crea v1 v2 = false
                exist-arest? (afegeix g v1 v2 e) v3 v4 =
                        ((v1==v3) and (v2==v4)) or (exist-arest? g v3 v4)
                ...
ENDSPEC

SPEC GrafN
        USING Graf(Natural && Natural)
ENDSPEC
```

E) En traduir a Axis l'especificació dels tipus dels naturals i dels enters definits a la biblioteca estàndar de Merlí (veure [Fra93], pp 6-10), s'han trobat algunes limitacions en el sistema:

- impossibilitat de redefinir objectes interns (0, 1, 2, etc.);
- impossibilitat d'utilització d'un mòdul PROPS per un altre PROPS;
- impossibilitat d'utilització de paràmetres en mòduls PROPS;
- impossibilitat d'afitar els naturals (si la constant d'afitament és interna, p.ex. 50, dóna error; si és una operació constant definida per l'usuari té conflictes per sobrecàrrega i no

opera com seria desitjable);
- problemes per sobrecàrrega d'operacions i constants.

L'aproximació que hem realitzat en Axis per solucionar aquests problemes té el format:

```
SPEC NatDef
        SORT natural
        OPS     cero : -> natural
                succ _ : natural -> natural
                pred _ : natural -> natural
        FORALL x: natural
        AXIOMS:
                succ pred x = x   pred succ x = x
ENDSPEC

SPEC Limite                                     PROPS ValNat
        USING NatDef                                    USING NatDef
        SORT limite                                     OPS val: -> natural
        OPS                                     ENDPROPS
                ocho : -> natural %    {aquesta constant representa la fita màxima dels naturals}
ENDSPEC

SPEC Nat (Max: ValNat)
        USING NatDef+Bool
        OPS     uno : -> natural ...
                ocho :      -> natural
                maxnat : -> natural
                _ + _ : natural natural -> natural  (ASSOC COMM)
                _ - _ : natural natural -> natural
                ...
                error-nat : -> natural
        FORALL  x, y : natural
        AXIOMS for succ and pred:
                maxnat = val    uno = succ cero    dos = succ uno    ...
        AXIOMS for operations:
                (succ x) = error-nat IF ( x >= maxnat ) %    L'anterior no es verifica
                (x + cero) = x
                (x + succ y) = succ (x + y)
                ...
ENDSPEC

SPEC FNat
        USING Nat (Limite FIT val : -> natural AS ocho)
ENDSPEC  %  Tampoc és possible utilitzar expressions després del AS

SETUP FNat  % La reescriptura dels següents tres termes no dóna els resultats esperats (ocho
                en forma de succ(succ...succ(cero)..) i false)
REWRITE maxnat
REWRITE ocho
REWRITE tres > maxnat
```

L'especificació dels enters s'ha fet de forma molt semblant, utilitzant l'instància "FNat" anterior. S'han obtingut missatges d'error per *parse* múltiple en reescriure termes com **dos - uno** o **dos * (tres + dos)**. En aquests casos s'havia preparat la base de regles dels enters (amb SETUP) i s'esperava que no hi hagués problemes d'ambigüitat entre les definicions de les operacions naturals i enteres en base a l'idea de localitat.

F) En executar reescriptures de termes extensos corresponents a la traducció a Axis de l'especificació d'una taula de símbols modular que apareix a [Fra91], pp.27-32, el sistema ha donat missatges de falta de memòria; termes semblants però més reduits s'han reescrit satisfactòriament.

% Els següentes termes s'han reescrit correctament:
REWRITE consulta (declara (declara (usa (usa (declara (declara (usa (declara (declara (declara crea "m1" "A" "cA") "m2" "A" "cA2") "m1" "B" "cB") "m1" "m2") "m4" "G" "cG") "m2" "D" "cD") "m2" "m3") "m1" "m4") "m3" "A" "cA3") "m3" "B" "cB3") "m3" "A"
REWRITE declaracions (declara (usa (declara (declara (usa (declara (declara (declara crea "m1" "A" "cA") "m2" "A" "cA2") "m1" "B" "cB") "m1" "m2") "m4" "G" "cG") "m2" "D" "cD") "m1" "m4") "m3" "A" "cA3") "m1"

% El següent terme esgota la capacitat del sistema (16 Mb):
REWRITE declaracions (declara (declara (usa (usa (declara (declara (usa (declara (declara (declara crea "m1" "A" "cA") "m2" "A" "cA2") "m1" "B" "cB") "m1" "m2") "m4" "G" "cG") "m2" "D" "cD") "m2" "m3") "m1" "m4") "m3" "A" "cA3") "m3" "B" "cB3") "m1"

G) En especificar l'exemple del joc del 4 en ratlla ([Fra91], pps. 48-53) es manifesta la impossibilitat de definir operacions privades (amplada, alçada, ...). A tots els exemples en general i més particularment en aquest, s'ha observat una velocitat considerable en efectuar un nombre elevat de reescriptures.

```
SPEC enratlla
        USING Color+Integer+Bool % se suposa que els colors usats són només blanc i negre
        SORTS enratlla
        OPS     init _ _ : nat nat -> enratlla
                juga _ _ _ : enratlla nat color -> enratlla
                he-guanyat? _ : enratlla  -> bool
                diagd _ _ _ _ : enratlla int int int -> bool
                ...
        FORALL  t: enratlla    n, n1, n2: nat   c: color     x, y, dx, dy, ni: int
        AXIOMS:
                ...
                he-guanyat? (init n1 n2) = false
                he-guanyat? (juga t n c) = (verticals (juga t n c) n )  or
                                ( diagd (juga t n c) (+(alcada-col t (n-2))) (+(n-3)) (+4)) or
                                ( diage (juga t n c) (+(alcada-col t (n-2))) (+(n+3)) (+4)) or
                                ( horitz (juga t n c) (+(alcada-col t (n+1))) (+(n-3)) (+4) )
                ...
                diagd t x y ni = (exit? t x y (+1) (+1)) or (diagd t (succ x) (succ y) (pred ni))
                diagd t x y (+0) = false
                ...
ENDSPEC
```

% Nombre de reescriptures: 16.669. Temps: 8.08 segons
REWRITE he-guanyat? (juga (juga (juga (juga (juga (init 10 6) 5 blanc) 3 blanc) 6 blanc) 7 negre) 4 blanc)

% Nombre de reescriptures: 24.705. Temps: 10.74 segons
REWRITE diagd (juga (juga (juga (juga (juga (juga (juga (juga (juga (juga (juga (juga (juga (init 10 6) 3 negre) 4 negre) 4 negre) 5 negre) 5 negre) 5 negre) 6 negre) 6 negre) 6 negre) 6 negre) 6 blanc) 5 blanc) 3 blanc) 4 blanc) + 0 + 1 + 4

6. Arquitectura del traductor

La traducció d'especificacions Merlí a Axis es basa en la construcció prèvia d'un arbre sintàctic que guarda l'estructura de l'especificació original; aquest arbre és manipulable per altres eines de l'entorn Excalibur i per això la seva existència és obligada, malgrat que la reescriptura de termes s'efectui *via* Axis. La construcció de l'arbre es basa en la descripció lèxica i sintàctica del llenguatge d'especificació mitjançant expressions regulars (per a l'anàlisi lèxica) i una gramàtica lliure de context (per a l'anàlisi sintàctica); hem optat per usar les conegudes eines Lex i Yacc que, donades aquestes descripcions, permeten associar codi C a executar cada vegada que a l'entrada es reconegui l'expressió o estructura sintàctica.

A la fig. 3 es mostra el procès seguit per construir el traductor. Els fitxers *equac.ll* i *expre.ll* contenen la descripció lèxica de la part d'especificació de Merlí i els fitxers *equac.yy* i *expre.yy* la descripció sintàctica; es diferencien dos fitxers a cada part atès que la sintaxi de les expressions, que resideix als fitxers *expre*, és la mateixa a la part equacional i a la imperativa i serà compartida. El fitxer *arbre.c* conté diverses rutines d'interès general per a la manipulació de l'arbre: arrelament, obtenció de fills i consulta; a més, al fitxer *gen_axis.c* hi ha les rutines especialitzades per a la traducció de l'arbre a una especificació Axis. Per altra banda, s'usen dos fitxers més, *ts.c* i *tg.c*, que contenen rutines per a la construcció i consulta de dues taules de símbols que contenen els identificadors vigents a l'univers en procès de traducció; *tg.c* manega una taula de les variables que apareixen en el codi Axis a partir de la traducció (en general, el procès de traducció afegeix noves variables a l'especificació) i *ts.c* una taula dels identificadors de l'univers. La definició de les estructures de dades per a l'arbre i les taules de símbols resideixen en el fitxer *union.h*. Per últim, el fitxer *fitxers.c* conté diverses rutines per emmagatzemar l'arbre i la taula de símbols a disc i vice-versa; així, els símbols definits i usats per un univers resideixen en un suport permanent i el seu ús queda facilitat.

Pel que fa l'ús del traductor (v. fig. 4), tota especificació Merlí d'un tipus abstracte *T* resideix en un fitxer *T.m*. La traducció d'aquest fitxer resulta en tres fitxers de sortida: *T.ax*, que conté la traducció a Axis de l'especificació Merlí; *T.arb*, que conté l'arbre sintàctic corresponent i el fitxer *T.ts*, amb la taula de símbols que conté els identificadors visibles de l'univers. A continuació, expliquem breument l'estructura de l'arbre i de la taula de símbols.

L'arbre sintàctic és una estructura de dades dinàmica que es representa mitjançant un arbre binari segons l'estratègia fill-esquerre, germà dret (cada node apunta al primer fill i al germà dret). El seu objectiu és reproduir l'estructura del programa obviant el sucre sintàctic. Els nodes són d'una categoria donada (per exemple, *IDENTIFICADOR*, *REANOMENAMENT*, etc.) i, segons la categoria, tenen uns valors determinats (per exemple, per a la categoria *IDENTIFICADOR* hi ha els valors *UNIVERS, GÈNERE, OPERACIÓ*, ...). L'arbre es forma mitjançant rutines semàntiques escrites en C associades a la corresponent regla de la definició

sintàctica per a Yacc. El posterior recorregut preordre de l'arbre tradueix l'estructura a Axis aplicant un repertori d'accions, cadascuna associada a una categoria.

La taula de símbols de l'univers s'ha implementat usant dispersió (*hashing*) amb estratègia *open adressing*, amb *rehashing* lineal. La funció de dispersió consisteix en una manipulació contundent de la codificació ASCII dels identificadors. Cada entrada de la taula conté l'identificador d'un objecte, la categoria (que és la mateixa que en l'arbre sintàctic), l'apuntador a l'identificador de l'univers que defineix els símbols (que resideix també a la taula), un indicador de si el símbol és o no és privat i, si és un símbol d'operació, una llista que en codifica la signatura. Pel que fa a la taula de variables de la traducció, la implementació és seqüencial atès que n'hi haurà poques; els noms de les variables es crea a partir d'un prefix no ambigu del gènere i un nombre que es va incrementant.

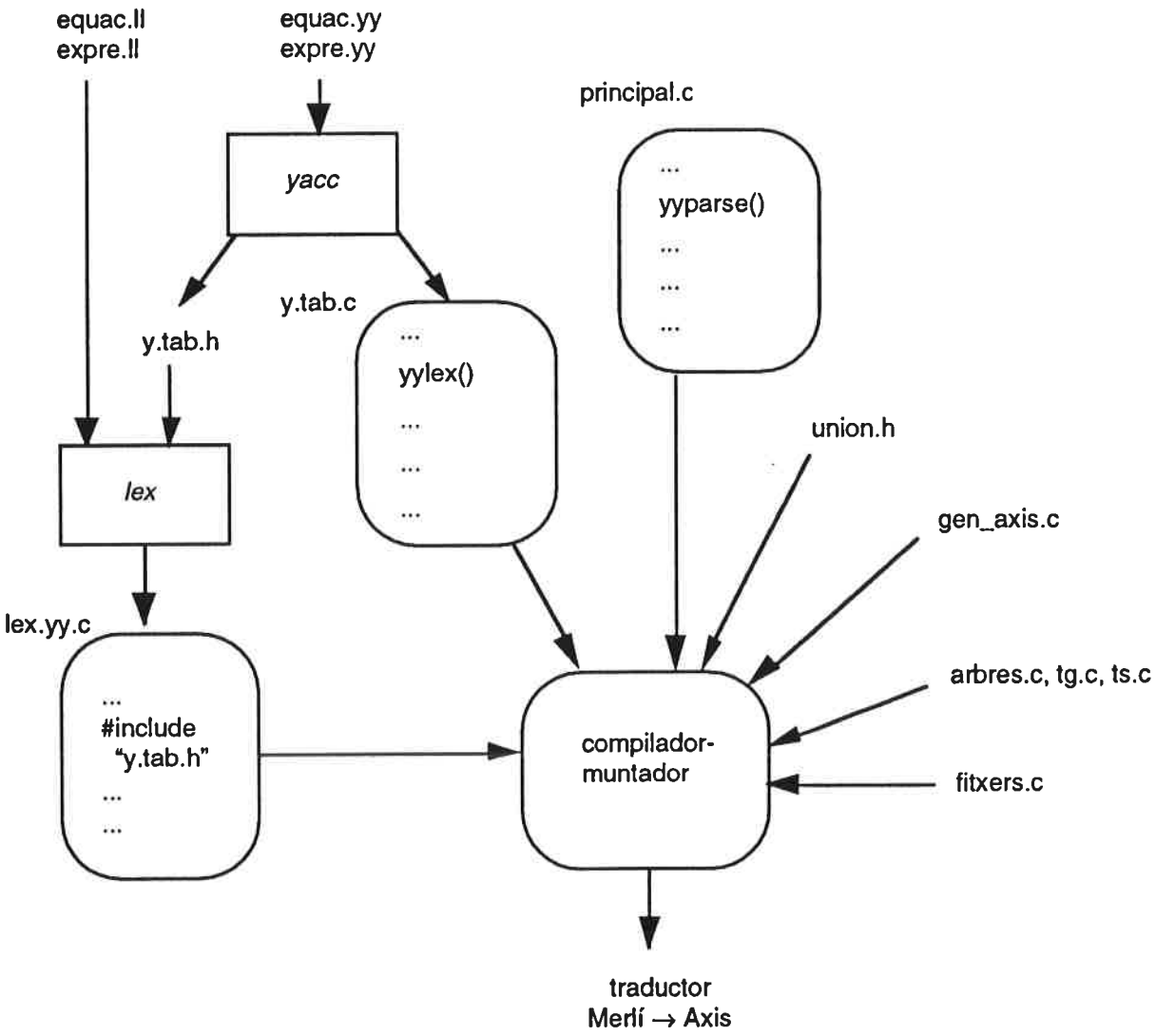

Fig. 3: construcció del traductor Merlí → Axis.

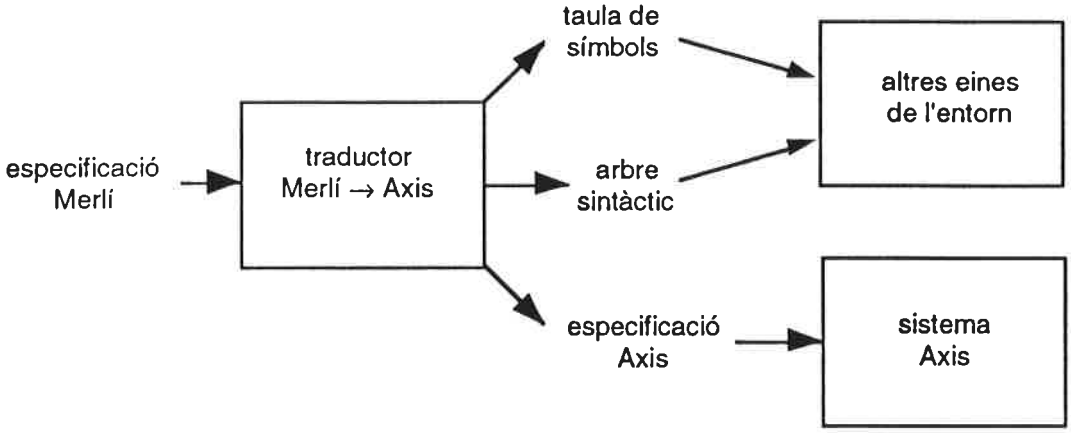

Fig. 4: manipulació d'una especificació Axis.

Referències

[ADJ78] J.A. Goguen, J.W. Thatcher, E.G. Wagner. "An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types". A *Current Trends in Programming Methodology*, Vol. IV, Prentice-Hall, 1978.

[Arn88] P. Arnold. "The User Manual for the Axis System". Technical Report HPL-ISC-TR-88-034,Software Engineering Dept, HP Labs, Bristol UK, 1988.

[Axis88] P. Arnold, D. Coleman, C. Dollin, R. Gallimore, T. Rush. "An introduction to the Axis specification language". Technical Report HPL-ISC-TR-88-031, Software Engineering Dept, HP Labs, Bristol UK, 1988.

[DoH88] C. Dollin, F. Hayes. "The Axis Library Manual". Technical Report HPL-ISC-TM-88-033, Software Engineering Dept, HP Labs, Bristol UK, 1988.

[FBB93] X. Franch, P. Botella, X. Burgués. "Report de definició del llenguatge MERLÍ". Report LSI-93-13-T.

[Fra91] X. Franch. "Especificació Algebraica de Tipus Abstractes de Dades: Estudi de Casos". Report LSI-91-5.

[Fra92] X. Franch. "Supporting Prototyping with a Multiparadigm Language". Report LSI-92-12-R.

[Fra93] X. Franch. "Especificació d'una biblioteca de tipus". Report LSI-93-23-R.

[Mon92] N. Montolío. "Part Equacional del Projecte Excalibur". Documentació interna del projecte Excalibur (sense publicar).

[OBJ85] K. Futatsugi, J. Goguen, J.-P. Jouannaud, J. Meseguer. "Principles of OBJ2". A *Simposium of Principles of Programming Languages*, pp. 52-66, 1985.

[PLUSS84] M.C. Gaudel. "A First Introduction to PLUSS". Technical Report METEOR, 1984.