RT/LSI-97-15-T

Automatització del càlcul de l'eficiència temporal dels algoritmes

Joan Vancells

Report LSI–97–15–T

UPC
BIBLIOTECA RECTOR GABRIEL FERRATE
Campus Nord

Automatització del càlcul de l'eficiència temporal dels algoritmes

1. Presentació del problema

L'anàlisi d'algoritmes és una branca de la informàtica que aplica raonament matemàtic per determinar certes propietats dels algoritmes que resolen un problema o enunciat particular. El seu objectiu és preveure el comportament d'un algoritme sense haver d'implementar-lo en un computador específic; es tracta de l'estratègia teòrica o a priori (analítica) enfront de l'estratègia empírica o a posteriori (experimental) que consistiria en programar-lo i executar-lo sobre uns exemplars de prova per extreure'n un resultat. Per diferenciar aquestes dues estratègies a vegades es parla d'anàlisi estàtic per referir-se a la primera, normalment acceptada per defecte. En un principi, la propietat única d'estudi era l'eficiència de l'algoritme o quantitat de recursos necessaris per a la seva execució (fonamentalment el temps de càlcul i, en menor mesura, la quantitat de memòria interna i, en alguns casos, la quantitat de memòria secundaria o el nombre de processadors), però amb el temps s'ha anat incorporant moltes altres propietats (algunes d'elles, fins i tot, específiques de cert tipus de programació, per exemple funcional). En el número de gener 1997 d'ACM-SIGPLAN hi ha una sèrie d'articles que exposen les diferents perspectives obertes i línies a seguir en aquest ja extens camp sota el títol genèric de "Program Analysis"[1]. El seu objectiu és el disseny i escriptura d'analitzadors (se sobreentén automàtics) de programes, que es defineixen de la següent manera:

"Un analitzador de programes és un programa que pren com a dada d'entrada un programa (escrit en algun llenguatge, i amb la possibilitat que hi hagi anotacions) i dóna com a sortida resposta a preguntes sobre propietats d'execució que són vàlides per a totes (o a vegades per a algunes de) les possibles execucions d'aquest programa."

(P.Cousot, ACM-SIGPLAN gener 1997 pp. 73-75)

[1] És interessant observar l'evolució del mot algoritme al mot programa d'aquest títol; el primer amaga el caràcter manual d'aquest anàlisi i el segon la seva automatització.

Convé deixar clar d'entrada que aquestes respostes seran necessàriament parcials. Els resultats clàssics de la teoria de la decidibilitat ens diuen que l'acabament d'un programa és en general indecidible i, per tant, cap analitzador de programes pot ser complet. Normalment, es considerarà que només s'analitzen programes que sabem segur que acaben.

Knuth [Knu73] fou el pioner i creador d'aquesta disciplina, i a ell devem els primers (molt extensos i complerts) anàlisis de l'eficiència d'una gran quantitat d'algoritmes. Encara ara és punt de referència obligada per a la majoria de gent que s'han dedicat a l'anàlisi d'algoritmes. Posteriorment ha dedicat molts llibres i articles a exposar el conjunt d'eines matemàtiques necessàries per efectuar aquests anàlisis. Ja en aquest primer text, va establir que, a més de la quantitat de memòria necessària (factor que normalment és fix i més fàcil de calcular), l'anàlisi complet d'un algoritme consistia en determinar quatre quantitats: el temps màxim o cas pitjor, el temp mínim o cas millor, el temp promig i la desviació d'aquest temps promig. Observem que aquestes dues últimes dades presuposen conèixer la distribució de probabilitat de les dades d'entrada en la majoria de casos. Després d'ell, una gran quantitat de treballs s'han dedicat a analitzar amb tot luxe de detalls alguns algoritmes clàssics del món de la programació.

L'especialització va portar a que hi hagués gent que es dedicava a fer anàlisis complets d'algoritmes molt concrets, l'objectiu principal dels quals era l'anàlisi dels temps promig i la consegüent introducció i tractament de qüestions estadístiques, amb la complicació matemàtica inherent. Un exemple típic i que ha rebut molta atenció és el problema de l'ordenació, pel qual es coneixen molts algoritmes diferents que s'han analitzat i comparat amb tot detall. El darrer llibre de Sedgewick i Flajolet [SF96] es pot considerar una bona introducció a l'estat de l'art en l'anàlisi del cas promig d'algoritmes, i les tècniques associades. S'inclouen nombroses referències a articles importants en aquest terreny. D'altra banda, la majoria de llibres d'algorítmica i estructures de dades han incorporat un apartat pràctic d'anàlisi d'algoritmes, centrant-se de seguida en l'estudi del cas pitjor per tenir només una idea o per impossibilitat de saber res a priori de les dades d'entrada. Aquesta segona disciplina ha rebut a vegades el nom de complexitat (computacional) [AHU83].

A part d'això, hi ha la qüestió del nivell de detall amb que fem aquest anàlisi: ho comptem tot o bé en tenim prou amb una aproximació. Es solen distingir dues categories d'anàlisi: microanàlisi i macroanàlisi. El microanàlisi compta el temps necessari, considerant cada operació elemental, per executar l'algoritme. El macroanàlisi és una aproximació i expressa el temps necessari per executar l'operació dominant, normalment utilitzant notació asimptòtica [AHU83, BB87, CLR90]. El més habitual és trobar o bé microanàlisi del cas promig o bé macroanàlisi del cas pitjor. En aquest darrer

excepcionalment s'incorpora la informació del cas promig en notació asimptòtica quan aquesta és prou significativa. L'exemple més típic és el quicksort amb un cas pitjor d'ordre quadràtic però un cas promig d'ordre n*log n, que el diferencia de la resta d'algoritmes quadràtics que resolen aquest problema.

La notació asimptòtica, majoritàriament acceptada com a expressió d'una aproximació i que és una adaptació de la teoria dels nombres clàssica, es va començar a utilitzar a principis dels 70 i va quedar establerta de forma més o menys definitiva en un article de Knuth [Knu76]. En aquest article, s'estableix la O-notació per expressar una fita superior, la W-notació per expressar una fita inferior i la Q-notació per expressar una coincidència de fites inferior i superior (fita ajustada). Brassard [Bra85] va acabar de donar els últims retocs a la proposta de Knuth substituint les igualtats per la notació de teoria de conjunts que és l'enfocament que es segueix actualment. En [BB87] s'amplia la definició a vàries variables.

Per a realitzar anàlisis de l'eficiència d'algoritmes cal conèixer i dominar una sèrie de tècniques matemàtiques que han estat reunides en alguns llibres específics [GK81, GKP89, SF97]. Com ja hem dit, l'anàlisi del cas promig és bastant més complex i és l'objectiu principal d'aquests llibres, que inclouen temes com avaluació de sumatoris, resolució d'equacions de recurrència, funcions generatrius, probabilitat discreta, aproximacions asimptòtiques, arbres, permutacions, ... Algunes d'aquestes tècniques apareixen resumides en la majoria de llibres d'algorítmica [AHU83, BB87, CLR90], però només en el que és necessari per calcular la complexitat dels algoritmes: avaluació i afitament (en notació asimptòtica) de sumatoris (per calcular els temps d'execució d'una iteració), i mètodes de resolució de recurrències tant per obtenir valors exactes com aproximacions en notació asimptòtica.

Un cop situat el problema de l'anàlisi de l'eficiència (en temps) d'algoritmes, passem a fer un petit recorregut històric/cronològic per les referències en les que es dediquen a parlar de la seva automatització, i que diseccionarem en profunditat en els seguents apartats. La primera referència a l'automatització del càlcul de l'eficiència dels algoritmes és una proposta de Cohen i Zuckerman [CZ74], en la que es requereix la interacció de l'usuari. Es considera el primer analitzador automàtic (sense assistència) el sistema METRIC construït por Wegbreit [Weg75]. Aquest obre una línia de microanàlisi automàtic del cas promig (el més complex) en la que s'enmarquen també els treballs de Hickey i Cohen [Coh82, HC88]. L'alternativa més important en aquesta línia són els treballs de Flajolet, Salvy i Zimmermann, el sistema LgW [FSZ89, FSZ89b, FSZ91, Sal89, Zim89]. Un altre Zimmermann, però en aquest cas alemany, va desenvolupar el sistema COMPLEXA [Zim88, Zim89] que es pot considerar una extensió del sistema de Wegbreit i, per tant, en la mateixa línia.

L'altra gran línia té un menor interès teòric (i també menor complexitat) però més possibilitats d'aplicació immediata a la pràctica. El seu objectiu és el macroanàlisi automàtic del cas pitjor. El primer antecedent que coneixem és el sistema TIMER [Kas80], que genera fites superior i inferior a la complexitat de programes escrits en un petit llenguatge imperatiu i que necessita la interacció de l'usuari. La següent referència és el sistema ACE de Le Métayer [LeM88], per generació automàtica de funcions de complexitat per a un subconjunt del llenguatge FP. Més o menys a la mateixa època, Rosendahl [Ros89] proposa un sistema alternatiu per a derivar automàticament fites superiors a l'eficiència en el cas pitjor de programes escrits en un subconjunt de Lisp.

Observem que totes les referències són bastant antigues i que es centren quasi exclusivament en llenguatges funcionals; en particular, el macroanàlisi de programes imperatius ha rebut molt poca atenció.

2. Dominis d'aplicació

En aquest segon apartat volem aprofundir en els treballs citats que es poden considerar antecedents més o menys directes del tema objecte del nostre interès: l'automatització del càlcul de l'eficiència en temps d'algoritmes. I ho farem analitzant els aspectes que cobreixen, classificant-los segons els objectius que persegueixen (macro o microanàlisi, cas promig o pitjor, ...) a més del tipus de llenguatges sobre el que actuen (imperatius, funcionals, amb o sense limitacions) i el tipus de problemes o enunciats a que queden restringits.

2.1. Classificació segons els objectius

Menció a part requereix el primer article sobre el tema, el de Cohen i Zuckerman [CZ74], ja que ells mateixos el consideren com un simple primer pas cap a la mecanització de l'anàlisi d'algoritmes. El seu objectiu és el microanàlisi però del temps màxim i mínim d'algoritmes escrits en Algol (amb algunes limitacions poc importants). Es diferencia principalment dels treballs posteriors perque és un sistema interactiu d'ajuda a l'estimació del temps ja que és l'usuari el que dirigeix l'anàlisi a través d'una sèrie de comandes d'execució i d'assignació de probabilitats a les condicions. A més, a través d'unes variables de temps es dóna informació de les característiques del computador i del compilador per tal de poder obtenir un resultat final en unitats de temps (en valor absolut).

Més endavant, l'objectiu esdevé precisament aconseguir sistemes el màxim d'automatitzats possible que retornin expressions del cost en funció de la(s) mida(s) de l'entrada. Per aconseguir això s'hauran de substituir els llenguatges imperatius per llenguatges funcionals (resolució de recurrències en lloc de càlcul de nombre d'iteracions) i s'haurà d'acceptar la limitació a certs tipus de problemes (aquells en que és més senzill calcular el cost). Cal remarcar la divisió del procés en dues etapes que esdevindran típiques: la primera de construcció de la fórmula simbòlica de temps a partir del programa (a nivell de traducció dirigida per la sintaxi i amb la introducció de variables de temps, en aquest cas) i la segona d'assignació de valors a les variables de temps, al nombre d'iteracions del *while* i a la probabilitat de cada expressió booleana, i finalment simplificació de la fórmula via manipulació algebraica (en aquest cas dirigida per l'usuari). Posteriorment rebran el nom d'etapa dinàmica i estàtica (no confondre amb anàlisi estàtic), respectivament. Com a possibles ampliacions de futur es citen el càlcul del temps promig i la introducció d'una comanda en la segona fase per realitzar aproximacions en notació asimptòtica. I com a camí a seguir en aquesta línia proposa anar disminuint la feina de l'usuari en la segona fase incorporant-hi capacitats deductives similars a les requerides a programes de verificació de la correctesa.

La resta d'articles són més fàcils de classificar segons els seus objectius. Els treballs de Wegbreit [Weg75], Cohen [Coh82, HC88], Flajolet [FSZ89, FSZ89b, FSZ91] i Zimmermann [Zim88, Zim89] es dediquen tots ells al microanàlisi del cas promig. Tot i que el grau d'automatització és molt elevat, segueixen fent pensar més en una eina d'ajuda a l'analista d'algoritmes concrets en detall (pel tipus i format dels resultats obtinguts) que en una eina d'ajuda al programador d'aplicacions. I els de Kasai [Kas80], Le Métayer [LeM88] i Rosendahl [Ros89] centren el seu interès en el macroanàlisi del cas pitjor. Aquest enfocament és conscient que el microanàlisi és més precís però produeix fórmules difícilment tractables, i que el cas promig és més realista però requereix conèixer la funció de distribució de les dades d'entrada, no sempre realista i fiable. És clarament un enfocament més pragmàtic que es proposa obtenir una eina per enginyeria del software. Anem a veure les diferències entre ells.

El sistema pioner de Wegbreit [Weg75] és fruit de la seva época i el seu objectiu és expressar el rendiment d'un programa com una tupla formada per mínim, màxim, mitjana i variància, tal com proposa Knuth [Knu73]. Veu l'anàlisi de rendiment de programes com una derivació d'expressions en forma tancada del comportament esperat del temps d'execució en funció de l'entrada d'aquests programes. La mecanització d'aquest procés té, segons ell, tres aplicacions principals: com a eina d'enginyeria del software (ajuda al programador a entendre el comportament del programa, tal com Le Métayer proposarà més tard [LeM88]), en un sintetitzador automàtic de programes, i

com a part del compilador d'un llenguatge de molt alt nivell. I es realitza en tres fases: assignació d'un cost a cada component (en primitives segons una taula de costos, en procediments analitzant el seu cost i en procediments recursius passant a la següent fase), anàlisi de la recursió (construcció de conjunts d'equacions en diferències) i resolució d'equacions en diferències usant diverses tècniques. L'interès per l'automatització completa (sense assistència, característica del seu únic precedent [CZ74]) és la causa de les limitacions d'aquest sistema. Posteriorment el sistema COMPLEXA de Zimmermann [Zim88, Zim89] eliminarà algunes d'aquestes limitacions a la llum de nous resultats en resolució de recurrències afegint-hi mecanismes de control més elaborats i resolent recurrències més complexes. Articles posteriors de Wegbreit estudien possibles utilitzacions del seu sistema: que les expressions de rendiment obtingudes de forma automàtica es converteixin en guia per dirigir la transformació del programa en un de més eficient [Wegbreit76] (idea que desenvoluparà més tard Ait-Ameur en la seva tesi [AiA92]) o bé adaptar les tècniques de verificació formal per demostrar que el temps d'execució màxim o promig està correctament descrit per les especificacions d'eficiència del programa [Weg76b].

El següent article de Cohen [Coh82] (recordem que és el co-autor de la nostra primera referència [CZ74]) es pot considerar que realitza un estudi de l'estat de l'art del nostre tema en aquella època. Comença situant el problema amb unes definicions prèvies centrant de seguida el seu interès en el microanàlisi, fa una breu referència a articles previs relacionats [CZ74, Weg75], i sobretot comenta treballs anteriors que han de servir de base pel seu objectiu: resultats recents en resolució de tipus especials d'equacions en diferències, exemples d'anàlisis no trivials realitzats en altres articles (per donar una idea de la dificultat de l'empresa), propostes de metodologies per a realitzar l'anàlisi (per part de Knuth basicament) i una proposta en ferm de formalització d'anàlisi d'algoritmes inspirada en tècniques de verificació tal com havia indicat Wegbreit (model de Ramshaw [Ram79], al qual tornarem més endavant). I tot orientat al microanàlisi del cas promig amb una preocupació especial pel tractament de probabilitats des del punt de vista expressiu i amb la manipulació algebraica simbòlica com a cor del sistema (corresponents a la primera i segona fase, dinàmica i estàtica, respectivament). El següent article, amb Hickey [HC88], insisteix en demostrar que els mètodes formals poden ser usats com a fonament teòric per implementar un compilador de rendiment capaç de generar automàticament equacions que expressen el rendiment promig de programes (amplia el seu interès a semàntiques probabilistes i gramàtiques probabilistes amb atributs) i mostra amb alguns exemples en llenguatge funcional pur (FP) com funciona el seu mètode. En cap moment, però, es parla d'un sistema ja implementat com en altres articles. Centren tot l'esforç en la incorporació de funcions de distribució de probabilitat de l'entrada a

l'anàlisi en cas promig d'un programa, qüestions més de semàntica i expressivitat, previs al disseny d'un sistema. No tenim notícies d'avenços recents en la proposta.

La següent referència en ordre cronològic són els treballs de Flajolet, Salvy i Zimmermann. Anteriorment Flajolet s'havia dedicat a estudiar mètodes matemàtics per l'anàlisi d'algoritmes i estructures de dades, que han quedat resumits en el llibre ja citat [SF96]. Un article del 1989 [FSZ89] inicia l'interès per la seva automatització. Aquest article presenta el sistema Lambda-Upsilon-Omega, LgW, dissenyat per a realitzar anàlisis automàtics de classes ben definides d'algoritmes operant sobre estructures de dades descomposables. De fet és un programa assistent de l'analista que amaga una gran quantitat d'informació d'expert matemàtic i s'ha de considerar com una eina d'ajuda al càlcul exacte del cost d'algoritmes en cas promig, via manipulacions algebraiques i analítiques, expressats en un llenguatge de descripció d'algoritmes i amb unes característiques molt concretes i limitades. El sistema està format per dos components principals: el subsistema analitzador algebraic ALAS [Zim90], implementat en CAML dialecte de ML, que cerca correspondències generals entre estructures de dades i especificacions d'algoritmes per una banda i equacions sobre funcions generatrius per l'altra; i el subsistema analitzador analític ANANAS [Sal89], conjunt de rutines algebraiques escrites en MAPLE, que agafa les funcions generatrius generades per ALAS i n'intenta extreure una expansió asimptòtica dels seus coeficients de Taylor. Un tercer subsistema, SOLVER, permet la connexió dels dos subsistemes principals, aprofitant per fer algunes simplificacions senzilles. En un report intern del mateix any [FSZ89b] s'explica amb més detall el funcionament del sistema i s'exposen els enunciats inicialment provats amb els resultats obtinguts. Un article posterior [FSZ91] incorpora algunes novetats, i el sistema (presentat com a prototipus) es va millorant en versions posteriors.

De forma paral.lela altres autors van centrar el seu interés en l'automatització de l'anàlisi de la complexitat, però en aquest cas de forma puntual i sense continuïtat. La primera referència és el sistema TIMER [Kas80] que genera fites superior i inferior a la complexitat de programes escrits en un llenguatge imperatiu tipus C reduït. Requereix anotacions de l'usuari (bàsicament indicacions del rati en que cada brancament és escollit). No hi ha noves referències a l'anàlisi automàtic de la complexitat del cas pitjor fins al sistema ACE de Le Métayer [LeM88] i es pot considerar l'alternativa més clara al sistema METRIC de Wegbreit (el sistema de Flajolet és posterior i molt limitat, [HC88] donen una idea però no presenten cap implementació concreta i sembla que no està al corrent del sistema TIMER). Justifica per qüestions pragmàtiques el seu interès pel macroanàlisi del cas pitjor: dificultat del microanàlisi (s'obtenen fórmules intractables) i manca d'informació pel cas promig (fiabilitat de la funció de distribució de les dades

d'entrada, en cas d'existir, a part de la complexitat afegida al tractar amb probabilitats). També se li simplifica el problema pel fet d'analitzar programes escrits en llenguatge funcional pur (FP), ja que això li permet basar el seu sistema en la poderosa àlgebra associada amb aquest tipus de llenguatges. Això fa que la idea del sistema només sigui extensible a altres llenguatges funcionals purs. D'entrada, com que fem macroanàlisi, no cal considerar el cost de funcions primitives ni formes funcionals i per tant, el seu problema es redueix a calcular el nombre de crides recursives necessàries per avaluar la funció-programa. I després, només ha de transformar la equació recursiva resultant en una equació no recursiva a través d'una sèrie de simplificacions i transformacions prèvies a l'aplicació del principi d'inducció recursiva (matching amb algun dels patrons predefinits) amb la gran avantatge que no calen transformacions "exactes" sinó simples aproximacions compatibles amb el propi objectiu del sistema.

I per acabar aquest repàs inicial sota el punt de vista dels objectius que es persegueixen en els antecedents del nostre tema, passem a comentar el darrer dels treballs apareguts en la línia de macroanàlisi del cas pitjor. La proposta de Rosendahl [Ros89], resum amb algunes millores de la seva tesi de l'any anterior, té com a objectiu l'automatització del càlcul de la complexitat com a fita superior en el cas pitjor. Però, de la mateixa manera que el treball de Hickey i Cohen en l'altra línia, només estableix un marc teòric o mètode a seguir per construir un sistema automàtic que derivi fites superiors a la complexitat i, per tant, es centra més en els aspectes de demostració de la correctesa d'aquest possible sistema que, d'altra banda, explica amb més detall i sense gaires incògnites obertes (com és el cas de Hickey i Cohen, més incomplert). També distingueix dues fases, una primera de derivació del que anomena una versió comptadora de passos (step-counting) de l'algoritme que exterioritza la propietat interna que és el seu temps d'execució, i una segona de càlcul de la funció fita del temps que s'expressa com una interpretació abstracta de la versió comptadora de passos obtinguda en la primera fase. També és comenten els treballs anteriors relacionats, bàsicament els que nosaltres hem repassat. I a l'igual que la majoria d'aquests, presenta com a possibles aplicacions de la funció de complexitat obtinguda l'ajuda a la construcció d'algoritmes més eficients sobretot com a guia d'un sistema de transformació de programes (ja hem dit que Ait-Ameur fa cas d'aquets sugerència en la seva tesi [AiA92], en la que utilitza el treball de Rosendahl).

2.2. Classificació segons el tipus de llenguatge

El segon aspecte respecte el qual ens interessa classificar aquest conjunt de treballs és el tipus de llenguatge, i les seves limitacions, en que estan escrits els programes que es vol analitzar de forma automàtica. Excepte [CZ74] i [Kas80] que ho fan amb llenguatges

imperatius (i que inclouen anotacions o informació addicional proporcionada per l'usuari), la resta ho fan sobre llenguatges funcionals, tot i que amb grans diferències quant a limitacions entre els uns i els altres. De fet les limitacions en el llenguatge, en molts casos, produeixen limitacions importants en el tipus de problemes a tractar (sobretot en el cas de llenguatges funcionals), i per tant està molt relacionat amb el tercer aspecte a comparar.

Ja hem dit que [CZ74] analitzen programes escrits en Algol 60 amb les següents restriccions: 1) no permet variables locals, *go-to's*, etiquetes, arrays per valor ni procediments recursius; 2) s'han d'enumerar els paràmetres formals; i 3) el valor de la variable de control així com les expressions que representen els valors inicial i final d'una sentència *for* no es poden canviar dins de l'àmbit de la sentència. En [Kas80] s'analitzen programes escrits en un llenguatges tipus C, bàsicament semblant al cas anterior.

El sistema METRIC és un prototipus que analitza programes LISP (en concret Interlisp de Xerox) simples (com els que s'utilitzarien en un curs d'introducció a la programació en LISP: append, reverse, nth, substitute, flatten, member i union), tot i que en les conclusions es fa algun comentari sobre possibles ampliacions del sistema per tractar contruccions típiques imperatives que no conté el Lisp.

El sistema LgW rep com a entrada l'especificació d'un algoritme i la seva estructura de dades escrita en un format especial anomenat ADL (Algorithm Description Language). La limitació en aquest cas va més lligada a les estructures de dades sobre les que operen els algoritmes, i per tant, és més una limitació de la classe d'algoritmes sobre els que es pot aplicar (veure punt 2.3).

El sistema ACE utilitza una versió no estricta de FP (especificada en un apèndix) com a llenguatge font, amb la particularitat pròpia d'aquest llenguatge: només té una estructura de dades, la seqüència. Com ja hem dit la tria del llenguatge és molt important en aquest cas, ja que la seva poderosa àlgebra és la base de la major part del sistema. Es comenta la seva possible adaptació (bàsicament de la primera part del sistema) a d'altres llenguatges funcionals purs mantenint FP com a llenguatge intern ja que la seva estructura és la base del sistema. Per tant, la major part de les idees no són aplicables a d'altres sistemes que treballin amb programes no funcionals.

Finalment, Rosendahl [Ros89] descriu un sistema que pot analitzar programes escrits en un subconjunt de primer ordre de Lisp tot i que es comenta la seva possible aplicació a llenguatges imperatius i funcionals *lazy*, sempre i quan la versió comptadora de passos resultant de la primera fase estigui escrita en Lisp de primer ordre. En el cas de

llenguatges imperatius això suposa una traducció a funcional, no sempre trivial. I, per tant, a tots els efectes té les mateixes limitacions que altres sistemes, només és aplicable a llenguatges funcionals.

2.3. Classificació segons el tipus d'enunciat

I per acabar ens interessa comentar les limitacions, en cas d'haver-n'hi, en el tipus d'enunciats sobre els que es pot fer l'anàlisi automàtic (inherents al llenguatge o no), o si més no en el tipus de problemes sobre els que els autors han demostrat o comprovat la seva utilitat o èxit. Les dues propostes que treballen amb llenguatges imperatius, [CZ74] i [Kas80], són vàlides per qualsevol tipus d'enunciats, amb les anotacions de l'usuari es pot fer front a qualsevol dificultat. En els antecedents teòrics del sistema TIMER [AKM79] s'analitzava un llenguatge més restringit (loop programs) en que només es podien utilitzar bucles amb un nombre fix d'iteracions, de fet també és fix a efectes de càlcul si incorporem una anotació-fita a tot bucle.

En el moment en què entrem en l'anàlisi de programes sense anotacions escrits en llenguatge funcional (en lloc de calcular el nombre d'iteracions s'ha de resoldre equacions de recurrència) és quan apareixen les limitacions. El sistema METRIC és el més limitat de tots, només serveix per analitzar programes Lisp pur simples, ja que només sap resoldre recurrències lineals amb coeficients constants. El sistema COMPLEXA amplia el ventall de programes analitzables en incorporar el tractament de recurrències més complexes.

L'article de Cohen proposa un mètode per microanalitzar programes i dóna com a exemple (indirecció a d'altres treballs d'ell mateix) una sèrie de programes no trivials amb els quals ho han provat: el mètode de Strassen de multiplicació de matrius, algoritmes de parsing deterministes. Recordem que és un mètode manual amb l'ajuda del computador, no un sistema automatitzat. En el següent article [HC88] (centrat en l'expressivitat) es generalitza aquest mètode sobre un llenguatge funcional pur FP i es donen tres exemples simples d'anàlisi del cas promig: concatenació de dues llistes, inversió d'una llista i superinversió d'una llista, a més d'un de més complex d'ordenació amb l'ajuda d'un arbre binari. Però també en aquest cas s'ha de considerar com un pas més per arribar a un analitzador automàtic, forma part de l'estudi teòric previ.

El sistema LgW es centra en l'anàlisi d'una important classe d'algoritmes que operen amb estructures de dades "descomposables", això vol dir tipus de dades que es poden especificar (explícitament o recursivament) amb l'ajuda de poques construccions ben conegudes de teoria de conjunts (unió de tipus, producte cartesià o records, seqüències o

llistes, conjunts i multiconjunts). Els enunciats concrets en que s'ha provat el sistema es poden dividir en tres categories: llenguatges regulars i autòmats finits (cadenes d'addició i problemes d'optimització relacionats, comportament a llarg terme de controls d'estat finit en sistemes i problemes combinatoris expressables en termes de llenguatges regulars), termes i arbres (algoritmes de derivació, alguns algoritmes de simplificació, una classe de sistemes de reescriptura, algoritmes d'unificació i problemes relacionats d'ocurrència de patrons) i problemes combinatoris (problemes d'arbres aleatoris, grafs, particions i particions ordenades, alguns problemes especials de permutacions). En [FSZ89b] apareixen els passos seguits (especificació del problema, programa font en ADL, anàlisi algebraica, resolució d'eqüacions i anàlisi asimptòtica) i els resultats obtinguts en un total de 18 exemples.

El sistema ACE està pensat per analitzar programes en FP i l'estructura del llenguatge és la seva base. Per tant, les limitacions del sistema estan íntimament relacionades amb les limitacions del llenguatge: els algoritmes que no es poden expressar de forma natural amb FP no es poden analitzar. Els dos principals trets del llenguatge FP són: té una sola estructura de dades, la seqüència, amb la qual es fa difícil expressar alguns algoritmes; i no està governat per cap sistema de tipus, informació que també pot ser d'interés per l'anàlisi. S'ha analitzat amb ACE alguns algoritmes raonablement llargs no especialment escrits per a aquest propòsit: programes d'ordenació (*quicksort*, inserció, intercanvi,...), programes numèrics, programes sobre grafs, programes de cerca, un *parser*, un programa de reconeixement de patrons, ... Es comenta la seva flexibilitat i extensibilitat, es pot augmentar el seu camp d'aplicació enriquint el sistema amb nous axiomes i noves definicions recursives, però cal anar amb compte per no violar la propietat d'acabament de les operacions de simplificació.

Finalment, Rosendahl també comenta les restriccions del mètode que proposa (deixant de banda, com tots els altres sistemes, que no resol el problema de la parada). Només funciona per programes en que la recursió està controlada per restriccions estructurals, com la longitut d'una llista. Per solucionar-ho es pot estendre el domini però no sempre s'aconsegueix l'aproximació. De fet és la restricció de tots els sistemes que no permeten anotacions i que analitzen programes funcionals. El mètode s'ha provat amb èxit en una sèrie de programes que inclouen algoritmes amb matrius, ordenació i un *parser* determinista.

3. Tècniques utilitzades

Per acabar d'aprofundir en els diferents treballs que tracten l'automatització del càlcul de l'eficiència dels algoritmes, farem un repàs de les tècniques que utilitzen per aconseguir els seus objectius i, en alguns casos, farem referència als resultats teòrics previs en que es recolzen o s'inspiren aquestes propostes. Hi ha dos grans temes que són preocupació comú principal en la majoria d'articles: les tècniques de resolució d'equacions de recurrència (que s'han anat ampliant mentre apareixien aquests articles, i que tenen una gran importància en l'anàlisi de programes funcionals) i les eines de manipulació algebraica simbólica per simplificar expressions abans de la presentació de resultats (també amb el temps s'han anat perfeccionant eines com MACSYMA i MAPLE, de gran ajuda per als analistes).

En el primer treball citat [CZ74], s'utilitzen eines típiques de la construcció de compiladors en la primera fase: un analitzador sintàctic que condueix la traducció per obtenir fórmules de temps a partir del programa a analitzar; i en la segona fase, a part d'associar valors a una sèrie de paràmetres, hi ha un conjunt de procediments per simplificació de fòrmules (associat a la comanda eval).

En el sistema de Wegbreit també hi ha una fase d'anàlisi de la recursió que intenta establir una correspondència entre l'expressió de cost recursiu i un conjunt d'equacions en diferències en tres pasos: reducció a forma normal, construcció d'equacions de recurrència per avaluació simbòlica i discriminació de casos i, finalment, projecció en els enters. Això és el que fa que només sapiga resoldre equacions de recurrència lineals amb coeficients constants, de fet les úniques necessàries per analitzar programes Lisp simples com és el seu objectiu. El sistema de Zimmermann és el que s'encarregarà d'afegir la resolució de recurrències més complexes i, per tant, d'ampliar el tipus d'algoritmes que es poden analitzar. Seguint amb el sistema METRIC, les equacions en diferències obtingudes en la fase d'anàlisi de la recursió es resolen en la següent fase per mitjà d'una de les següents tècniques: suma directa, pattern matching, eliminació de variables, anàlisi del cas pitjor i millor, i diferenciació de funcions generatrius.

Hickey i Cohen proposen un prototipus de sistema basat en els treballs teòrics de Kozen sobre semàntica de programes probabilistes (intenta establir un significat matemàtic precís de programes que utilitzen generadors de nombres aleatoris) i de Ramshaw sobre correcció de programes amb assercions de rendiment [Ram79] (inspirat en els sistemes de verificació de Floyd-Hoare). És Ramshaw el que en la seva tesi va considerar per primer cop l'anàlisi del cas promig d'algoritmes dividit en dues parts: una primera fase dinàmica en que cal usar el coneixement sobre conceptes de programació com són els condicionals, les iteracions i la recursivitat, i la comprensió de l'estructura matemàtica del

domini de dades, per a caracteritzar la distribució de probabilitats del paràmetre d'eficiència sota estudi amb l'ajuda d'una construcció matemàtica (relació de recurrència); i una segona fase estàtica en que s'utilitzen tècniques matemàtiques pures per trobar una solució de la recurrència, sigui exacte o asimptòtica. El seu objectiu és construir un sistema formal en que la primera fase pugui reduir-se a manipulació simbòlica, i proposa el sistema freqüència (probabilitats però sense la obligació de sumar 1) per dissenyar un sistema formal en que l'anàlisi del rendiment en promig d'un programa es pot demostrar que és correcte aplicant certes regles d'inferència a un conjunt d'axiomes. Inicialment comenta l'anàlisi del cas pitjor però al veure que el més important és identificar quin és aquest cas pitjor i el reste ja no té tan interés, es centra del tot en el cas promig i sobretot en problemes d'expressivitat formal dels aspectes que intervenen en aquest anàlisi del temps promig. Els posteriors articles de Hickey i Cohen segueixen en aquesta línia amb la presentació d'una semàntica probabilista de programes FP i la introducció de gramàtiques probabilistes amb atributs per representar les distribucions d'entrada i sortida. No es coneixen treballs posteriors, però tot sembla indicar que més que com a càlcul d'eficiència d'algoritmes pot ser d'interés en la verificació d'assercions de rendiment (com un article de Wegbreit [Weg76b]). En [FSZ91] es comenta el gran poder expressiu d'aquesta proposta, però es veu difícil que s'hi pugui desenvolupar un càlcul complet (amb regles de simplificació i formes normals).

El sistema LgW està basat, en la part algebraica (ALAS), en metodologies recents d'anàlisi combinatori que observen correspondències sistemàtiques entre definicions de tipus estructural i funcions generatrius de comptatge, i, en la part analítica (ANANAS), en correspondències en part clàssiques i en part noves entre singularitats de funcions analítiques i el creixement asimptòtic dels seus coeficients de Taylor.

El sistema ACE està basat en el concepte de transformació de programes introduïda per Darlington i Burstall, i consta d'una llibreria ampliable amb més de 1000 regles de transformació (el seu punt crític és precisament el trobar la regla adequada en cada moment). Consta de dues parts: la primera calcula, a partir de la definició recursiva de la funció, el nombre de crides recursives necessàries per a la seva avaluació, i la segona (la més important) transforma aquesta expressió recursiva en una de no recursiva. Aquesta segona consta a la vegada de cinc passos: simplificació (usant un conjunt d'axiomes de FP que actuen com a regles de transformació), factorització (en cas de que no sigui possible aplicar el principi d'inducció recursiva, s'intenta expressar la funció de complexitat com a composició amb una funció de mesura), partició (quan la factorització falla, s'eliminen condicions que no són rellevants per l'avaluació de la complexitat), aplicació del principi d'inducció recursiva (les operacions anteriors tenen com a objectiu produir una equació recursiva que coincideixi amb un patró predefinit i per fer el

matching es proporciona una llibreria de les definicions recursives més usuals, tan exactes com aproximacions aprofitant que l'objectiu del sistema és la complexitat asimptòtica del cas pitjor) i finalment substitució (si l'aplicació del principi d'inducció recursiva no és possible). Aquesta proposta es fonamenta en la potència del àlgebra de programes funcionals, expressades per mitjà de les regles de transformació que s'apliquen en els passos anteriors.

El mètode o marc teòric proposat per Rosendahl per construir un sistema d'anàlisi automàtic de la complexitat es centra més en la qüestió de la demostració de la correctesa d'aquest mètode (demostrar que realment deriva una fita superior a la complexitat). I per tant, es mou més en el terreny de les diferents semàntiques (denotacional, operacional, ...) del llenguatge (subconjunt de Lisp) adaptades a les seves necessitats. També distingueix dues fases: la primera de derivació de la versió comptadora de passos (*step-counting*, notació introduïda en [AKM79]) que exterioritza la propietat interna del programa que indica el temps de càlcul; i la segona, que calcula la funció fita superior del temps de càlcul que s'expressa com una interpretació abstracta de la versió comptadora de passos de la primera fase. La primera fase consisteix en un programa transformador que a partir d'un programa en llenguatge font qualsevol retorna el mateix programa amb l'afegit de noves funcions que donen el seu temps de càlcul. Es proporciona l'esquema de traducció dirigit per la sintaxi de que consta aquesta primera fase, i la major part de la feina és adaptar una semàntica per poder portar a terme la demostració de correctesa d'aquest primer pas. La segona fase és més interessant i innovadora, és el primer treball que utilitza la interpretació abstracta com a tècnica per construir un analitzador autòmatic de la complexitat. Per la seva importància comentarem aquesta tècnica més endavant. Tornant a l'article de Rosendahl, en aquesta segona fase fa una interpretació abstracta de la versió comptadora de passos centrant el seu interès en demostrar la seva correctesa: el valor retornat per la interpretació abstracta serà major que tots els valors que pugui retornar la interpretació estandard per totes les possibles entrades; i per obtenir el programa que calcula la funció fita del temps només cal composar la versió comptadora de passos amb la funció de longitut invertida.

La interpretació abstracta, introduïda per Cousot [CC77], és un esquema general que facilita la inferència de propietats de programes en temps d'execució. Normalment pren com a punt de partida alguna semàntica estàtica precisa del programa (basada en la semàntica estandard operacional) que en general no pot ser computada en temps finit però si aproximada amb seguretat. Això s'aconsegueix substituint el domini concret per un domini abstracte que conté menys informació i les operacions en el domini concret per operacions abstractes en el domini abstracte. La correspondència entre propietats

concretes i abstractes s'estableix per una funció d'abstracció i una de concretització que és una conexió de Galois que formalitza la pèrdua d'informació.

La introducció de la interpretació abstracta està motivada pel desig de justificar l'especificació d'analitzadors de programes respecte a alguna semàntica formal. La idea central és una transformació discreta o aproximada de propietats des d'una semàntica exacta o concreta en una semàntica abstracta o aproximada (aspecte constructiu).

L'objectiu de la interpretació abstracta és provar la solidesa dels mètodes d'anàlisi de programes respecte a la semàntica o encara millor dissenyar formalment aquests mètodes per aproximació de la semàntica de programes. Des del punt de vista teòric, l'objectiu de la interpretació abstracta és dissenyar jerarquies de semàntiques interrelacionades especificant a varios nivells de detall el comportament de programes quan són executats per computadors, i des del punt de vista pràctic, el seu objectiu és dissenyar eines per l'anàlisi automàtic de programes que determinin estàticament propietats dinàmiques de programes. La interpretació abstracta és a la semàntica formal el que l'anàlisi numèrica és a l'anàlisi matemàtica. Els problemes amb una solució analítica no coneguda es poden resoldre numèricament donant solucions aproximades.

Després de l'article fundacional citat [CC77] que s'enmarcava en el context dels llenguatges imperatius, diferents autors l'han adaptat al context de llenguatges funcionals i lògics en els quals ha acabat tenint més acceptació i anomenada. En el mateix numero de ACM SIGPLAN del gener 1997 en que apareix la visió de l'anàlisi de programes sota l'óptica de la enginyeria del software [LeM97], Cousot fa un breu repàs d'aquest tema sota la perspectiva de la interpretació abstracta.

4. Una proposta

Després d'un estudi en profunditat de les diferents propostes (poques) que s'han fet en el terreny de l'automatització del càlcul de l'eficiència en temps d'algoritmes arribem a la conclusió que es poden classificar en dos grans grups: un primer que està pensant desenvolupar una eina d'ajuda al microanàlisi del cas promig i que té com a objectiu molt llunyà la seva total automatització (són conscients que hi ha molt camí per recòrrer i desenvolupen prototipus d'aproximació especialitzats en cert tipus d'algoritmes, el seu granet de sorra a un projecte faraònic i quasi inabastable); i un segon grup, més pragmàtic, que en té prou amb automatitzar el càlcul d'una fita superior del cas pitjor però que tot i que el seu objectiu no és tan llunyà, forcen el seu assoliment introduïnt limitacions insalvables: es centren en el paradigma funcional (més facilment automatitzable pel seu fonament matemàtic però d'ús molt restringit a la pràctica pels

seus problemes d'eficiència i d'expressivitat de certs tipus d'enunciats) i no donen gaire importància a la simplificació i presentació del resultat en una notació normalitzada (notació assimptòtica, per exemple). Estem d'acord en la total automatització (sense assistència) però no a costa de perdre generalitat, encara que calgui introduir anotacions en le programa font. De fet, les propostes del segon grup no es poden considerar generals (ni en cas de considerar la restricció als llenguatges funcionals) ja que només permeten tractar la recursivitat controlada per restriccions estructurals.

En el camp de l'enginyeria del software, qualsevol proposta s'ha de situar forçosament en el grup pragmàtic. Per això, estem interessats en automatitzar el càlcul de l'eficiència asimptòtica de programes imperatius, entenent el qualificatiu "imperatiu" en el sentit més ampla possible (incloent-hi orientació a objectes, per exemple), segons s'ha exposat en [FV97]. Volem fer un analitzador de programes imperatius que calculi la complexitat (macroanàlisi cas pitjor) i l'expressi amb la notació asimptòtica O-gran (resultat en escala clàssica: logarítmic, lineal, quasi-lineal, quadràtic, ...). La idea inicial és treballar sobre un llenguatge imperatiu reduït però suficientment complet (estructures de control habituals i, en particular, el bucle, i que permet crides a accions i funcions, a més d'incloure un mecanisme de definició de tipus bàsics) i tractar el temps d'execució com un atribut sintetitzat de la gramàtica donant les regles de càlcul adequades a aquest llenguatge. De moment, és necessari afegir a tots els bucles una fita o nombre màxim d'iteracions, però en un futur volem estudiar la possibilitat d'estalviar-ho a l'usuari, si més no en alguns casos. També introduïm el concepte d'esquelet per especificar patrons de combinació d'operacions amb una eficiència potencialment diferent de la que es calcularia amb les regles estandard.

A part de l'estudi de casos d'estalvi de fita, anirem enriquint el nostre llenguatge amb noves construccions i potencialitats fins a arribar a un llenguatge amb les característiques dels existents d'amplia difusió tractant les succesives adaptacions del nostre sistema. També tenim pensat ampliar el càlcul a notacions asimptòtiques alternatives (Ω-notació per expressar una fita inferior i la Θ-notació per expressar una fita ajustada) per tal d'obtenir un estudi més complet de l'eficiència dels programes. I finalment, deixem una porta oberta a una possible generalització del càlcul automàtic a d'altres propietats no funcionals d'interès.

Amb aquesta declaració de principis queda clar que la nostra proposta comparteix els objectius inicials dels treballs de Le Métayer i Rosendahl però centrats en la programació imperativa com en els més antics articles de Cohen i Kasai. I afegim com a novetat a totes les referències conegudes el fet d'utilitzar la notació asimptòtica (la més ampliament acceptada) com a forma d'expressió dels resultats. També reprenem la idea d'incorporar anotacions de fita en les iteracions com proposaven Cohen i Kasai, però intentant

estalviar-nos-les en la majoria de casos per tendir a un sistema més proper als METRIC i COMPLEXA. El concepte d'esquelet per afinar més els anàlisis del nostre sistema és també una novetat respecte als treballs estudiats. Tot i el seu interés, els punts de contacte amb les propostes d'anàlisi automàtic del cas promig (principalment, els treballs del grup de Flajolet i Hickey i Cohen) seran mínims, per no dir inexistents. És un tema completament diferent.

Comentar, per acabar, que volem connectar aquest treball amb el projecte ComProLab [Fra+97] de selecció automàtica d'implementacions de components en base a les seves característiques no funcionals. L'eficiència és una de les característiques no funcionals dels programes, i el projecte aquí proposat permetrà que aquesta característica es calculi automàticament, millorant l'estat actual del sistema que requereix l'escriptura explícita per part del programador de l'eficiència de les implementacions.

Bibliografía

[AHU83] A.V.Aho, J.E. Hopcroft, J.D. Ullman. "Data Structures and algorithms". Addison-Wesley, 1983.

[AiA92] Y. Ait-Ameur. "Application des techniques d'interpretation abstraite aux developpements formels de programmes". Ph.D. Thesis, ENSAE, 1992.

[AKM79] A. Adachi, E. Kasai, E.Moriya. "A theoretical study on the time analysis of programs". LNCS 74, Springer Verlag, 1979.

[BB87] G. Brassard, P.Bratley. *Algorithmique. Conception et analyse.*Ed. Masson, 1987.

[Bra85] G. Brassard. "Crusade for a better notation". SIGACT News, 16(4), 1985.

[CC77] P. Cousot, R. Cousot. "Abstract interpretation: A unified lattice model for static analysis of programs by construction or aproximation of fixpoints". *Proceedings ACM POPL*, Los Angeles (California), 1977.

[CLR90] T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[Coh82] J.Cohen. "Computer-assisted microanalysis of programs". *Communications ACM,* 25(10), 1982.

[CZ74] J. Cohen, C. Zuckerman. "Two languages for estimating program efficiency". *Communications ACM,* 17(6), 1974.

[Fra+97] X. Franch, P. Botella, X. Burgués, J.M. Ribó. "ComProLab: A Component Programming Laboratory". *Proceedings 9th Software Engineering and Knowledge Engineering (SEKE)*, Madrid(España), 1997.

[FSZ89] P. Flajolet, B. Salvy, P. Zimmermann. "Lambda-Upsilon-Omega: An assistant algorithms analyzer". A *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes,* LNCS 357, Springer Verlag, 1989.

[FSZ89b] P. Flajolet, B. Salvy, P. Zimmermann. "Lambda-Upsilon-Omega: The 1989 Cookbook". Research Report 1073, Institut National de Recherche en Informatique et en Automatique, 1989.

[FSZ91] P. Flajolet, B. Salvy, P. Zimmermann. "Automatic average-case analysis of algorithms". *Theoretical Computer Science* 79, 1991.

[FV97] X. Franch, J. Vancells. "Cálculo automático de la eficiencia asintótica de programas imperativos". III Jornadas de Informática de Puerto de Santa Maria, Cádiz (España), 1997.

[GK81] D.H. Greene, D.E. Knuth. *Mathematics for the Analysis of Algorithms.* Birkhäuser, 1981.

[GKP89] R. Graham, D.E. Knuth, O. Patashnik. *Concrete Mathematics.* Addison-Wesley, 1989.

[HC88] T. Hickey, J. Cohen. "Automating program analysis". *Journal ACM,* 35, 1988.

[Kas80] T. Kasai. "An automatic time analysis system". Technical Report RIMS-335, Kyoto University, 1980.

[Knu73] D.E. Knuth. *The Art of Computer Programming, vol 1: Fundamental Algorithms.* Addison-Wesley, 1973.

[Knu76] D.E. Knuth. "Big Omicron and Big Omega and Big Theta". SIGACT News, (82), 1976.

[LeM88] D. Le Métayer. "ACE: An automatic complexity evaluator". *ACM TOPLAS* 10(2), 1988.

[LeM97] D. Le Métayer. "Program analysis for software engineering: new applications, new requirements, new tools". *ACM SIGPLAN* , Gener 1997.

[Ram79] L.H. Ramshaw. "Formalizing the analysis of algorithms". Technical Report SL-79-5, Xerox Palo Alto Research Center, Palo Alto, CA.

[Ros89] M. Rosendahl. "Automatic Complexity Analysis".A *FPCA 89*, 1989.

[Sal89] B. Salvy. "Fonctions Génératrices et assymptotique automatique". Research Report 967, INRIA, 1989.

[SF96] R. Sedgewick, P. Flajolet. *An Introduction to the analysis of algorithms.* Addison-Wesley, 1996.

[Weg75] B. Wegbreit. "Mechanical program analysis". *Communications ACM,* 18(9), 1975.

[Weg76] B. Wegbreit. "Goal-Directed program transformation". *IEEE Transactions on Software Engineering*, SE-2(2), 1976.

[Weg76b] B. Wegbreit. "Verifying program performance". *Journal ACM,* 23(4), 1976.

[Zim88] W. Zimmermann. "How to mechanize complexity analysis". Technical report GMD-Karlsruhe, 1988.

[Zim89] W. Zimmermann. "Automatische komplexitätsanalyse von funktionalen programmen". Ph.D. Thesis, University of Karlsruhe, 1989.

[Zim90] P. Zimmermann. "Alas: un système d'analyse algébrique". Rapport de recherche 968, INRIA, 1989.

[Zim91] P. Zimmermann. "Séries génératrices et analyse automatique d'algorithmes". Ph.D. Thesis, Ecole Polytechnique (Palaiseau, France), 1991.

UPC
BIBLIOTECA RECTOR GABRIEL FERRATE
Campus Nord