

Grau en Matemàtiques

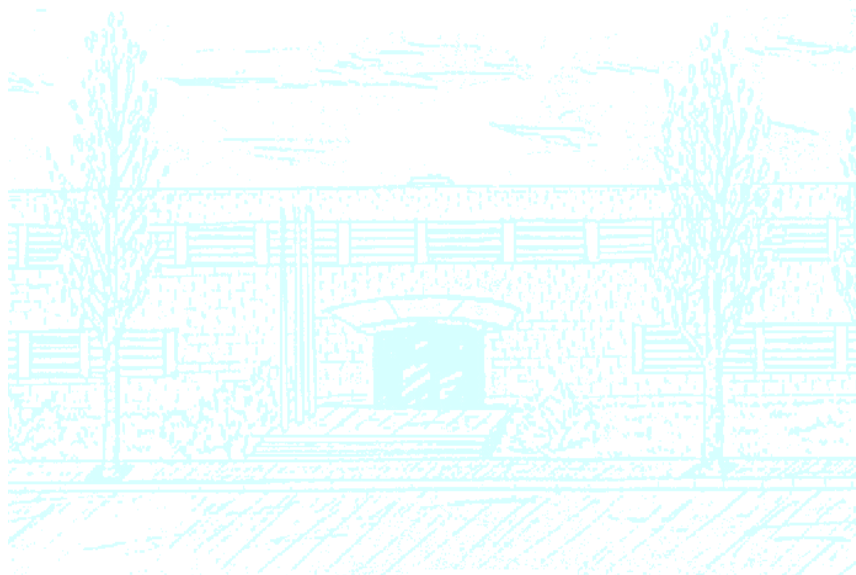
Títol: Reconfiguració de robots cristal·lins

Autor: Joan Soler Pascual

Directora: Vera Sacristán Adinolfi

Departament: Matemàtica Aplicada II

Convocatòria: 2012/2013



Facultat de Matemàtiques
i Estadística

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Treball de fi de grau en Matemàtiques

Reconfiguració de robots cristal·lins

Joan Soler

Directora: Vera Sacristán Adinolfi

Departament de Matemàtica Aplicada II

Resum

L'objectiu d'aquest treball consisteix en crear i implementar un algorisme de reconfiguració per a un tipus de robots encara en fase d'experimentació anomenats moduls. Aquests robots estan formats per un conjunt de peces totes de la mateixa mida i forma anomenades àtoms, que poden interactuar entre elles, connectar i desconnectar-se per poder modificar i deformar la forma final del robot. En concret, els robots moduls dels que parlem en aquest projecte són els anomenats cristal·lins o telecubs que tenen la propietat que poden expandir-se i contraure's. Si aquests moviments es poguessin realitzar de forma suficientment ràpida, aconseguiríem un robot que podria construir estructures d'emergència, o podria arribar a llocs on seria perillós per un humà.

El treball es divideix en dues parts: Primer hem modificat un simulador de robots moduls en Java, perquè accepti les propietats i accions que tenen el nostre tipus de robots. Després hem usat el simulador per modificar un algorisme de reconfiguració per un robot modular que només usa l'espai unió entre la configuració inicial i la final. La nostra implementació es limita al cas bidimensional. Aquest algorisme és centralitzat, i el nostre objectiu ha estat fer-ne una versió distribuïda, és a dir, on els mòduls actuen de forma totalment independent només amb la seva informació i la dels veïns directes. Finalment, hem implementat l'algorisme i l'hem completat amb millores per augmentar-ne la velocitat.

Abstract

The aim of this project is to create and implement a reconfiguring algorithm for a robot type still in development called modular robot. These robots are made of a set of pieces of the same size and shape called atoms, which can interact with each other, connect and disconnect to modify and deform the final shape of the robot. Specifically, the modular robots of this project are called crystalline or telecubs, which have the property to expand and shrink. If these movements could be done fast enough, we could build a robot that would make emergency structures, or could reach places where it would be dangerous for a human to be.

The project is split in two parts: First we modified a modular robot simulator in Java to accept the particular properties and actions of our robot type. Then we used the simulator to modify a reconfiguration algorithm for a modular robot that only uses the space union of the initial and the final setup. Our implementation is limited to two-dimensional case. This algorithm is centralized, and our goal has been to make a distributed version of it, i.e. where the modules operate completely independently only with its own and direct neighbors' information. Finally, we have implemented the algorithm and we have completed upgrades to increase its speed.

Índex general

Capítol 1. Introducció	1
1.1. Robots modulars	1
1.2. Robots cristal·lins	2
1.3. Algorismes distribuïts	3
1.4. Objectiu del treball	4
1.5. Organització de la memòria	4
Capítol 2. El model i la simulació	5
2.1. Moviments atòmics	5
2.2. Moviments modulars	6
2.3. Simulació	8
Capítol 3. Descripció de l'algorisme	13
3.1. Objectiu	13
3.2. Introducció a l'algorisme	13
3.3. Estructura	14
3.4. Estats	15
3.5. Adaptació al simulador	15
Capítol 4. Regles d'actuació	17
4.1. Cerca de l'arrel i creació de l'arbre	17
4.2. Compressió	19
4.3. Expansió	22
Capítol 5. Correcció i complexitat	27
5.1. Definicions bàsiques	27
5.2. Compressió	29
5.3. Expansió	31
5.4. Reconfiguració	32
5.5. Millora de l'algorisme	34
Capítol 6. Creació d'un nou algorisme	37
6.1. Cerca de l'arrel i creació de l'arbre	37
6.2. Compressió	38
6.3. Expansió	38
Capítol 7. Conclusions	43
7.1. Resultats obtinguts	43

7.2.	Dificultats trobades	43
7.3.	Futur del projecte	44
7.4.	Valoració Personal	44
Referències		47
Annex A. Regles de la cerca de l'arrel i creació de l'arbre		49
A.1.	Arbre inicial [S]	49
A.2.	Cadena de missatges de les fulles	59
A.3.	Arbre objectiu [F]	65
Annex B. Regles de la compressió		79
B.1.	ZIP	79
B.2.	SWZIP	82
B.3.	Reparacions en les constants	84
Annex C. Regles de l'expansió		85
C.1.	Expansió del líder	85
C.2.	Seguiment del líder	99
C.3.	Retorn del líder	101
C.4.	Fi de líder	104
Annex D. Moviments Bàsics		105
D.1.	ZIP i UNZIP	105
D.2.	SWZIP	106
D.3.	Posicionament	106
D.4.	Altres moviments	107
Annex E. Compact User Guide		109

Capítol 1

Introducció

Aquest treball està dedicat a la reconfiguració de robots cristal·lins de forma distribuïda. Així doncs per començar explicarem què és un robot modular, i la seva diferència amb els robots especialitzats, quines són les característiques del model del nostre treball, i finalitzarem amb la descripció dels objectius del projecte i l'organització de la memòria.

1.1. Robots modulars

Un robot és un dispositiu que realitza tasques automàticament, ja sigui a través d'un programa definit, amb supervisió humana o utilitzant tècniques d'intel·ligència artificial. Aquests robots estan programats per a fer feines específiques i són molt precisos, però només poden treballar en condicions molt determinades. Per exemple, un robot aspirador no pot pujar escales, per tant no podrà netejar més d'un pis sense l'ajuda humana, o en una cadena de muntatge d'una fàbrica, un robot només pot fer una de les moltes tasques que hi pugui haver. A la Figura 1 podeu veure aquests exemples.



FIGURA 1. Esquerra: L'aspiradora ha estat dissenyada per poder superar petits obstacles però és fàcil bloquejar el seu recorregut amb una caixa. Dreta: En la cadena de muntatge, necessitem molts braços robot perquè cadascun faci una de les tasques del muntatge del cotxe.

Per millorar aquests robots, podríem fer que el robot aspiradora fos capaç de canviar de forma per superar molts més obstacles, per exemple, si es trobés un forat al

terra, podria canviar la seva base per tal d'arribar a l'altre extrem del forat i així creuar-lo o, en el cas del muntatge, podríem fer que el braç robot canviés la seva orientació per poder posar més d'una peça. Per això existeixen els robots modulars autoreconfigurables.

Un robot modular està format per un gran nombre d'unitats independents, les quals anomenarem àtoms, que es poden reorganitzar en una estructura més adequada per a una situació determinada. Per exemple, poden canviar la seva forma per una més llarga i prima per travessar un túnel estret o transformar-se en una estructura com un pont o unes escales. A més, com aquests robots estan formats per un conjunt d'unitats idèntiques també es poden reparar a si mateixos canviant components espatllats per d'altres que funcionen correctament i que estan en posicions on no s'utilitzen. Aquests robots, per tant, són ideals per treballar en espais remots on un humà tindria dificultats per arribar. La Figura 2 mostra diversos exemples de robots modulars que s'han construït fins al moment.

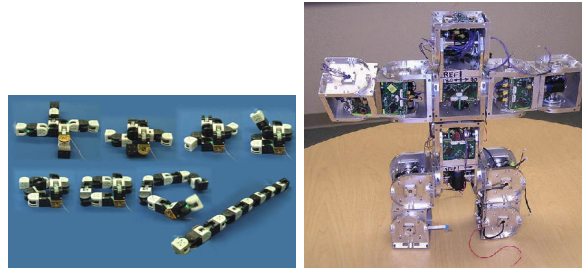


FIGURA 2. A l'esquerra robots del model M-TRAN, a la dreta, un SuperBot.

La dificultat d'aquests robots està en la programació de les reconfiguracions i les tasques, és a dir, en programar el moviment coordinat dels seus mòduls. Aquesta és la finalitat del treball, simular la reconfiguració d'un tipus de robot modular, els robots cristal·lins.

1.2. Robots cristal·lins

En aquest cas treballarem amb uns robots autoreconfigurables homogenis reticulars de forma quadrada¹ com es pot veure a la Figura 3.

Els àtoms tenen uns mecanismes d'expansió i contracció que els permet estendre i retreure les seves cares. A més, cada cara de l'àtom pot enganxar-se i desenganxar-se de la cara d'un àtom adjacent. Així doncs, el conjunt que forma el robot està connectat per les seves cares formant un conjunt connex. Usant aquestes operacions bàsiques de forma adequada, podem fer que uns àtoms es moguin de forma relativa, obtenint una nova configuració del robot. Vegem un exemple molt bàsic a la Figura 4.

¹Treballarem en 2D en tot el projecte, tot i que l'algorisme donat és vàlid a 3D per robots cúbics.

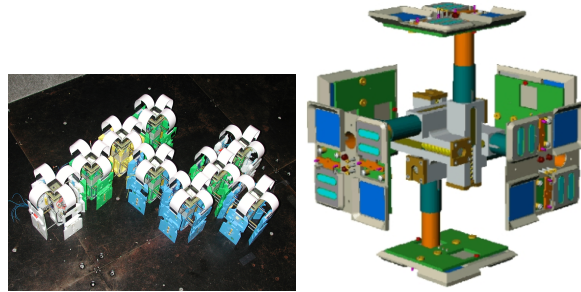


FIGURA 3. Dos exemples de robots cristal·lins, un quadrat, i l'altre cúbic.

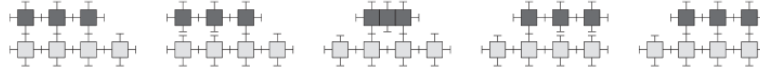


FIGURA 4. Utilitzant les 4 accions bàsiques, els àtoms superiors es desplacen respecte dels inferiors.

Tot i així, hi ha configuracions d'àtoms que no es poden modificar sense cap tipus d'ajuda externa, com per exemple la de la Figura 5. Per assegurar la reconfiguració, considerarem els àtoms agrupats en mòduls, grups de 2×2 àtoms.

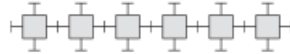


FIGURA 5. Al tenir els àtoms alineats, el robot no pot canviar la seva forma.

1.3. Algorismes distribuïts

La complexitat d'un algorisme de reconfiguració es mesura segons el nombre de passos paral·lels realitzats, i el nombre d'operacions atòmiques. Si reduïm el nombre de passos paral·lels obtindrem una millora en el temps de reconfiguració. Donat que els àtoms estan alimentats per bateries limitades, és interessant reduir el nombre total d'operacions.

Perquè el robot pugui fer múltiples passos paral·lels, construïm un algorisme distribuït, és a dir, un algorisme on cada mòdul actua pel seu compte, sense la necessitat d'un controlador central. L'inici de les accions pot ser enviada a tots els mòduls, però també podem enviar-la a un qualsevol i aquest activarà a la resta. Tots els mòduls actuen en paral·lel prenent les seves pròpies decisions en funció de la seva situació i la dels seus veïns. Per prendre decisions, tots els mòduls són dotats d'unes mateixes regles. Aquesta regla conté unes precondicions i unes accions, si el mòdul compleix les precondicions d'una regla executarà les accions que conté aquesta.

1.4. Objectiu del treball

L'objectiu d'aquest treball és modificar el simulador de Java[5] creat en el projecte [2] perquè pugui simular el nostre tipus de robots cristal·lins. Hem afegit alguns moviments bàsics dels robots i modificat la interfície perquè pugui acceptar la compressió i l'expansió. Un cop modificat el simulador, l'hem usat per simular una versió distribuïda de l'algorisme descrit a [3] i als Capítols 3 i 4. Finalment, hem intentat buscar solucions per millorar el temps d'execució de l'algorisme o implementar un algorisme de zero. Detallem una mica més els objectius:

- Modificar el simulador de Java: Hem arreglat la interfície perquè funcioni amb els nostres mòduls, ja que poden haver dos mòduls en la mateixa posició i hem de poder simular aquesta situació de forma intuïtiva. També hem afegit els nous moviments per introduir un mòdul dins la posició d'un altre, o per poder treure el mòdul o moure'l dins el robot. Inicialment fem la simulació de forma modular.
- Implementar l'algorisme distribuït descrit al Capítol 3 i 4 del treball.
- Implementar un nou algorisme, o modificació de l'anterior millorat, usant el simulador.

1.5. Organització de la memòria

Hem dividit aquest document en sis capítols:

- Capítol 1, Introducció explicant el nostre model per entrar en el context del treball.
- Capítol 2, titulat El model i la simulació, on expliquem els moviments bàsics dels mòduls i com es mostren al nostre programa.
- Capítol 3, titulat Descripció de l'algorisme, introduïm el nostre algorisme de reconfiguració i expliquem com està distribuït.
- Capítol 4, titulat Regles d'actuació, expliquem el nostre algorisme a alt nivell, junt amb els problemes que ens ha donat al simular.
- Capítol 5, titulat Correcció i complexitat, estudiem el nostre algorisme per veure la seva velocitat d'execució i consistència.
- Capítol 6, titulat Creació d'un nou algorisme, on introduïm les idees bàsiques d'un nou algorisme de reconfiguració.
- Capítol 7, titulat Conclusions, on es fa una reflexió sobre el resultat del projecte.

Finalment acabem el treball amb les referències consultades per dur a terme aquest treball i uns annexos on expliquem l'algorisme al detall, veiem els moviments atòmics del robot i per acabar la guia per al funcionament del simulador en anglès.

Capítol 2

El model i la simulació

Abans de començar a descriure els moviments modulars, cal explicar els moviments bàsics que pot realitzar cada robot atòmic dins del mòdul. Aquests moviments no els veurem al simulador en vista modular però són els que el robot real pot fer, i si escollim adequadament aquests moviments atòmics podem obtenir els moviments modulars que utilitzarem al simulador.

2.1. Moviments atòmics

Com hem comentat a la introducció dels robots cristal·lins, aquests robots poden fer quatre accions a cada cara del robot per aconseguir els moviments. Veiem aquests moviments al detall.

2.1.1. Expansió

Si un àtom té alguna cara comprimida, la pot expandir. Si aquesta cara està connectada a un altre àtom, aquesta expansió produeix una separació relativa dels dos àtoms i per tant podem aconseguir moviment del robot. A la Figura 1 podem veure un exemple senzill d'aquesta expansió.

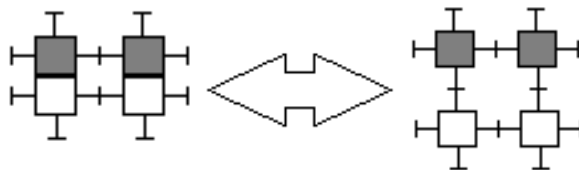


FIGURA 1. Tant els àtoms inferiors com els superiors fan l'*expansió* de les cares que tenen en contacte i provoquen que els superiors es traslladin.

2.1.2. Contracció

De forma anàloga a l'anterior, si un àtom té alguna cara expandida la pot contraure i si té algun àtom connectat a la cara de la compressió, provocarà una translació. El moviment és invers a l'expansió.

2.1.3. Connectar

Una altra acció que pot fer una cara de l'àtom és connectar-se a un veí seu perquè li faci de suport o per fer-li de suport a ell. (Veure Figura 2).

2.1.4. Disconnectar

Finalment, l'acció inversa de connectar, permet disconnectar dos àtoms per independitzar els seus moviments (veure Figura 2). Abans de fer aquesta acció s'ha de comprovar que la unió en qüestió no sigui aresta pont del graf que formen els robots, per tal de no disconnectar el conjunt del robot.

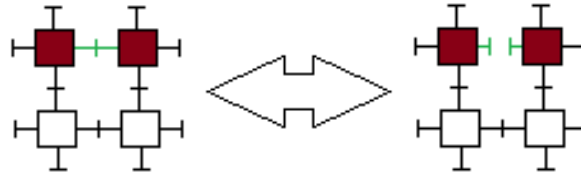


FIGURA 2. Els dos àtoms de color vermell fan una connexió (o desconnexió) de les cares que els uneixen.

2.2. Moviments modulars

Hem vist a la introducció que els àtoms no poden reconfigurar-se en moltes situacions, motiu pel qual introduïm els mòduls de 2×2 àtoms per tenir tota la mobilitat que necessitem. Un mòdul també pot fer els moviments atòmics, usant les cares exteriors del mòdul. A més a més, podem definir altres moviments que usarem per construir el nostre algorisme (la versió atòmica la podem veure a l'Annex D).

2.2.1. Compressió

Si dos mòduls estan totalment expandits, podem fer que els dos es comprimeixin en una mateixa posició. El mòdul que no ha canviat de posició serà el *mòdul amfitrió* i el mòdul que l'ha ocupat serà el *mòdul convidat*. Fem aquesta distinció ja que el mòdul amfitrió és el que executa totes les regles, el mòdul convidat només acceptarà les regles que rebí del mòdul amfitrió, representarem el mòdul convidat amb un quadrat interior, dins del mòdul amfitrió. Podem veure un exemple modular a la Figura 3.

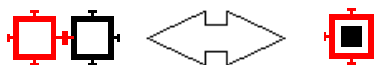


FIGURA 3. El mòdul negre fa la compressió cap al mòdul vermell i passa a ser el convidat.

2.2.2. Descompressió

La descompressió és el moviment invers de la compressió, el mòdul amfitrió envia al convidat a una posició veïna tot mantenint la connexió. Per fer-ho, inicialment aquesta posició que ha d'ocupar el mòdul convidat ha d'estar buida (Veure Figura 3).

2.2.3. Enviament

Aquest moviment consisteix en moure un mòdul convidat d'una posició amb dos mòduls comprimits a una posició veïna on hi ha un mòdul expandit, aquest segon es comprimirà i serà l'amfitrió en acceptar el mòdul convidat. Una condició necessària és que les dues posicions estiguin connectades. A la Figura 4 veiem el procés.

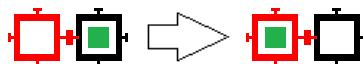


FIGURA 4. Veiem com el mòdul convidat (verd) és enviat a la posició del mòdul vermell, que el rep com a convidat i el mòdul negre finalment queda expandit.

2.2.4. Altres moviments

Existeixen dos moviments més que els robots cristal·lins podrien utilitzar per fer els moviments del nostre algorisme. Tanmateix, podem simular aquestes accions canviant les dades entre els mòduls implicats. Els moviments són els següents:

- *Intercanvi*: Dos mòduls comprimits a la mateixa posició canvien la seva situació de amfitrió-convidat.
- *Enviament doble*: Dues posicions amb mòduls comprimits veïnes intercanvien els seus mòduls convidats. En general s'usa la mateixa funció de crida que *Enviament* i amb la situació inicial decidim quina de les dues executem.

Al simulador, farem els dos moviments amb intercanvis de dades, que es podrien construir amb la combinació dels moviments descrits anteriorment.

Cal tenir en compte també que en la vista atòmica dels mòduls l'algorisme constantment ha de fer un moviment de *Rotació* dels àtoms per poder encarar el mòdul

al fer qualsevol dels moviments descrits en aquesta secció, però no es pot apreciar en forma modular, el detall d'aquest moviment en vista atòmica està descrit a Annex D.

2.3. Simulació

Per poder provar el nostre algorisme, hem modificat un programa creat en el projecte [2] amb el qual podem mostrar com els mòduls executen les regles per fer les accions descrites als apartats anteriors. Tots els mòduls són iguals i apliquen les mateixes regles, que executen si compleixen les condicions. Cada mòdul disposa d'una capacitat limitada d'informació per poder treballar: la seva orientació (N, S, E, W) ; un estat intern; les connexions amb els veïns; un nombre limitat de comptadors i missatges d'entrada-sortida amb els quals pot fer operacions elementals i enviar als seus veïns.

El programa té diferents vistes que ens donen informació dels moviments del robot i on podem manipular les regles i la posició dels robots. Podem veure la Guia d'Usuari a l'annex E, i descarregar el programa a [4]. Vegem les diferents finestres amb més detall:

2.3.1. Universe

Podem veure la finestra a la Figura 5. En aquesta vista podem visualitzar la reconfiguració del nostre robot i veure els moviments que fa en cada iteració. Tenim els botons superiors per controlar les iteracions, pausar el procés, poder tornar a un pas anterior o tornar a avançar, i així poder estudiar en profunditat com el nostre robot executa l'algorisme. Si pausem el procés, podem clicar als mòduls per veure una finestra amb les dades del mòdul seleccionat en la instant pausat.

Aquesta vista ha estat modificada respecte a l'original, per poder visualitzar com un robot es comprimia a la posició d'un altre robot, acció que els robots del programa original no podien fer.

2.3.2. Agents and Rules

En aquesta vista podem veure i modificar els dos documents que corresponen a la informació de la posició inicial dels mòduls (`agents.txt`), i al document que conté la informació de les regles que executaran tots els mòduls a cada iteració (`rules.txt`). Aquests són els documents que el programa obre de forma predeterminada. Tot i així, podem obrir qualsevol altre document guardat prèviament que contingui informació sobre mòduls o regles sempre que segueixi el format establert per el programa (Veure a la Figura 6).

1. *Agents*. En aquest espai, podem escriure l'estat inicial dels mòduls. La seva posició inicial, les connexions amb els seus veïns directes i donar valors al seu estat i els seus comptadors. El programa no permet afegir dos mòduls que inicien a la mateixa posició o sigui comprimits, ja que està pensat perquè les situacions inicial i final dels mòduls sempre estiguin totalment expandides.

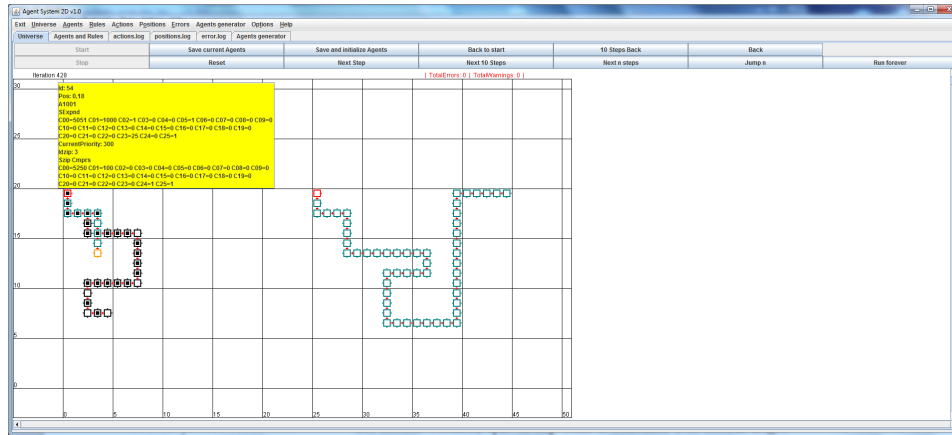


FIGURA 5. A *Universe* podem observar la configuració dels nostres robots a l'espai així com veure i manipular la iteració en la que ens trobem. En la finestra de color groc que apareix si cliquem un mòdul podem veure informació de les dades d'aquest.

2. *Rules*. En l'espai de la dreta, escrivim les regles que executaran a cada iteració tots els mòduls. Cada regla està formada per quatre línies consecutives separades per salts de línia de la següent forma:

Nom de la regla

Prioritat

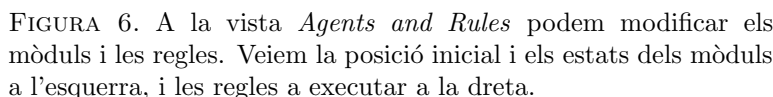
Precondicions que ha de complir el mòdul per executar la regla

Accions

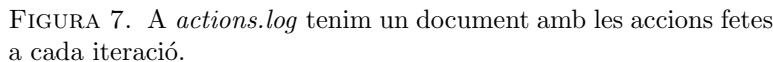
Així doncs, hem d'escriure les precondicions de cada regla pensant en la situació en la què es trobaran els mòduls quan volem que l'executi. Si un mòdul no compleix les condicions, saltarà a la següent regla sense llegir l'acció.

Hem afegit noves funcionalitats al programa original, per poder comprimir i descomprimir dos mòduls i també per poder moure els mòduls comprimits entre posicions. Són les següents:

- ZIP o compressió l'escriurem amb Z, i tot seguit la lletra (N,S,W,E) per indicar la direcció a comprimir-se. Donarà un error si el mòdul està comprimit, o es vol comprimir en una posició on no hi ha cap mòdul o n'hi ha un de comprimit.
- UNZIP, o descompressió del convidat, l'escriurem amb z, tot seguit la lletra de la direcció a descomprimir-se i l'estat començant per S seguit de cinc caselles que volem donar-li. Donarà error si el mòdul ja està descomprimit o es vol descomprimir en una posició on ha hi ha un mòdul.
- SWZIP, o enviament del convidat, l'escriurem amb x, tot seguit escriurem la direcció a fer l'enviament. Donarà error si el mòdul està descomprimit, o a la posició d'enviament no hi ha un mòdul descomprimit.



Un cop executat el programa, en aquesta finestra podem veure les regles que han executat els mòduls a cada iteració com veiem a la Figura 7. El document que conté aquesta informació queda guardat a la carpeta del programa.



En aquest arxiu, també guardat a la carpeta del programa, veiem la posició dels mòduls a cada iteració i les seves dades, missatges i estats en tot moment. A la Figura 8 en tenim un exemple del document.

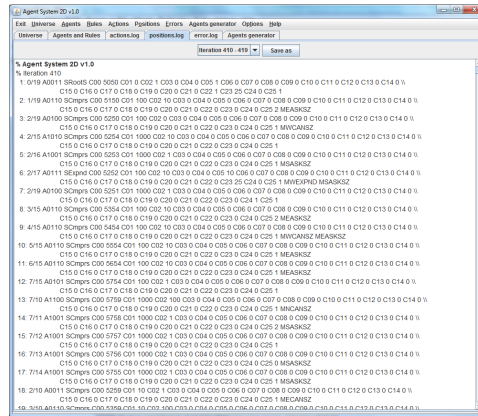


FIGURA 8. A l'arxiu *positions.log* hi tenim la informació dels mòduls en cada iteració.

2.3.5. Errors.log

Aquest tercer arxiu és el document on podem veure els errors del nostre algorisme. Per exemple, si en algun instant provoquem un col·lapse dels mòduls, o sigui tres mòduls volen comprimir-se en la mateixa posició o si un mòdul descomprimit vol fer una descompressió. Tots els errors els veurem en aquest arxiu, separat per iteracions. Per tant, aquest arxiu ens ha servit per construir el nostre algorisme, però amb l'algorisme finalitzat, l'arxiu només indicarà les iteracions com podem comprovar a la Figura 9, ja que no es produeixen errors.

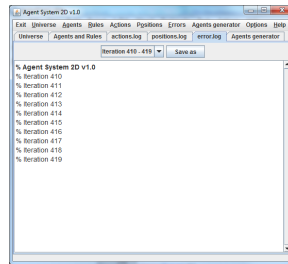


FIGURA 9. Podem veure com l'arxiu *errors.log*, ens mostra totes les iteracions sense cap error.

La llista d'errors del programa ha estat modificada, ja que inicialment no acceptava l'opció que a l'univers hi hagués dos mòduls a la mateixa posició. Hem afegit tots els possibles errors que podem causar al fer moviments que provoquin col·lapses de mòduls o altres moviments no naturals.

2.3.6. Agents generator

Finalment, tenim una finestra on podem crear el nostre robot de forma més visual i intuïtiva que usant el document *agents.txt*. Veiem una graella com la de la Figura 10, on podem interactuar per afegir o eliminar mòduls, i canviar el seu estat o constants. Finalment podem desar el document d'agents per usar-lo a l'univers.

No hem afegit l'opció de tenir mòduls comprimits a l'estructura original dels agents, per tant no ha calgut modificar res en aquest apartat.

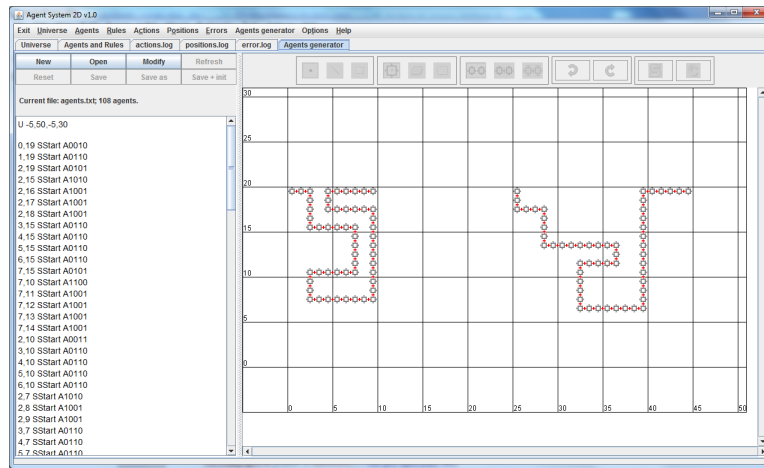


FIGURA 10. Amb els botons superiors podem dibuixar o esborrar els agents i canviar la seva informació. Veiem com a l'esquerra es genera un document d'agents que podem guardar.

Capítol 3

Descripció de l'algorisme

3.1. Objectiu

Volem construir un algorisme distribuït, tal que els mòduls prenguin decisions de forma independent, tan sols usant les seves dades i les dels seus veïns directes. A més a més, volem que els mòduls ocupin el mínim espai necessari, només usem l'espai unió de les formes inicial i final com s'il·lustra a la Figura 1.

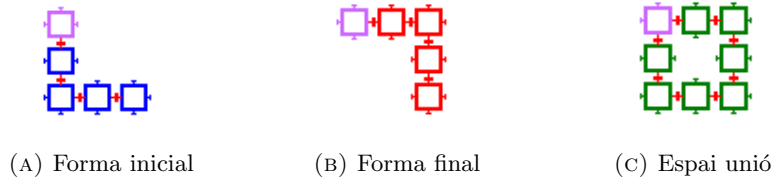


FIGURA 1. En aquest exemple podem veure l'espai unió que podem utilitzar per reconfigurar el robot.

És un algorisme on s'usa el mínim d'informació possible. A més, com els mòduls actuen en paral·lel, el temps d'execució és lineal en el nombre de mòduls, tal com és demostra en el Capítol 5.

3.2. Introducció a l'algorisme

En primer lloc, hem dissenyat una versió distribuïda de l'algorisme publicat a [3]. La idea general d'aquest algorisme és construir un arbre generador de la forma inicial i un de la final que comparteixen arrel. Per fer la transformació, es comprimeix l'arbre inicial i després s'expandeix en l'arbre final guiat per un mòdul que anomenarem líder. Aquesta compressió i expansió utilitza la particularitat dels mòduls dels robots cristal·lins on dos mòduls poden ocupar la mateixa posició.

Aquesta idea és molt senzilla si els dos arbres ocupen espais disjunts (excepte per l'arrel que la tindran sempre en comú). Quan no són disjunts, però, es poden produir els anomenats *deadlocks*, o sigui, quan volem expandir a una posició ocupada

per un mòdul que vol comprimir-se, produint un embós. L'algorisme proposa incorporar a l'arbre generador de la nova estructura els mòduls de la vella que es troben al llarg del camí.

Seguint aquesta idea hem fet un algorisme semblant, però de forma distribuïda, els mòduls prenen decisions sense l'ajut d'un controlador central, únicament utilitzant la seva informació i la que els poden donar els seus veïns directes.

A continuació descriurem el conjunt de regles que hem desenvolupat.

3.3. Estructura

Les regles per dur a terme la reconfiguració es poden agrupar en tres grans conjunts:

- **Elecció de l'arrel i creació de l'arbre:** Per organitzar els mòduls, usem un arbre d'expansió. Primer hem d'escollir quin mòdul serà l'arrel. Per fer-ho, tots els mòduls que no tinguin cap altre mòdul a la posició veïna nord ni a l'oest, són possibles arrels. Aquests mòduls envien un missatge amb la seva posició als seus veïns, que alhora la transmeten als seus respectius veïns, fent un recorregut en amplada del graf. Si un mòdul rep el mateix missatge des de dos o més direccions diferents, indica que hi ha un cicle, i el mòdul en qüestió el trencarà. Si un mòdul rep missatges diferents, compara les posicions donades i tria la que es troba més al oest-nord. Finalment, els missatges arriben a les fulles, que al seu torn els retornen als seus pares. Quan un missatge arribi a un candidat, comprova si és el seu propi missatge i si l'és, aquest mòdul es converteix en l'arrel, en cas contrari, torna a ser un mòdul esclau.
- **Compressió:** Comprimit l'arbre inicial, per així poder aconseguir espai per crear l'arbre final. Les fulles de l'arbre es comprimeixen cap al seu pare, i aquest envia el mòdul comprimit en direcció a l'arrel recorrent l'arbre. Reptint el procés, aconseguim comprimir tot l'arbre.
- **Expansió:** Per accelerar el procés, aquesta part s'inicia tant bon punt arriba el primer mòdul comprimit a l'arrel. El líder inicialment està a la posició de l'arrel, i sempre que arriba un mòdul comprimit a la seva posició, es mou en la direcció que correspon per formar l'arbre final. El líder té establertes unes preferències en les direccions ($S \rightarrow E \rightarrow W \rightarrow N$). Per tant, entre totes les possibles de l'arbre final, escull la que tingui més preferència. Pot trobar-se diferents casos:
 - Si la direcció a moure's està buida, s'expandirà sense cap problema.
 - Si la direcció està ocupada i el mòdul que l'ocupa està connectat a ell, aquest passa a ser el líder.
 - Si la direcció està ocupada i el mòdul no està connectat al líder, es connecten i el mòdul receptor es desconnecta del seu antic pare per no crear cicles. El seu nou pare serà el líder i finalment s'intercanvien.

Sempre que el líder arriba a una fulla de l'arbre final, recorre l'arbre en direcció a l'arrel fins que troba una nova intersecció no completada.

3.4. Estats

Per tal de distingir les diverses fases, les regles fan ús de diversos estats en què es poden trobar els mòduls:

- **Arrel** [**RootS**]/[**RootF**]: Comú a les formes inicial i final, i té la funció d'arrel dins l'estructura d'arbre generador.
- **Start** [**Start**]: Mòduls inactius de la forma inicial, esperen a que algun veí els envii un missatge per començar.
- **Final** [**Final**]: Mòduls inactius de la forma final.
- **Forward** [**Fwrds**]/[**Fwrdf**]: Mòdul en procés de creació de l'arbre. Ha rebut un missatge del seu possible pare, i ha d'enviar un missatge als seus possibles fills per crear l'arbre.
- **Back** [**Backs**]/[**Backf**]: Mòdul en procés de creació de l'arbre. Ha rebut una confirmació de que la seva branca de l'arbre ha arribat a fulles i ha d'enviar un missatge al seu pare perquè arribi el missatge a l'**Arrel**.
- **Líder** [**LIDER**]: Inicialment és l'**Arrel** i es tracta del mòdul que conté tota la informació de l'estructura final. Posteriorment, recorre tot l'arbre, per col·locar els altres nodes en la posició que els correspon.
- **Compress** [**Cmprs**]: Mòdul amb l'objectiu d'arribar a l'**Arrel** o a un mòdul en **Expand** de l'arbre. Si és fulla es comprimeix al seu pare, i aquest li indica el nou pare per arribar al seu objectiu.
- **Expand** [**Expnd**]: Mòdul ja fixat pel **Líder** en la posició final amb l'objectiu de fer arribar els mòduls al **Líder** de l'arbre. Quan un mòdul **Compress** arriba a un mòdul **Expand** o a l'**Arrel**, s'envia en la direcció on sigui el **líder**.

3.5. Adaptació al simulador

Hi ha algunes diferències de l'algorisme per al robot real amb el del simulador. En el robot, alguns mòduls coneixerien la informació de la forma final que volen construir. En el simulador no s'ha pogut fer així: Per simular-ho construïm la forma final al costat de la inicial, el líder inicialment busca la posició de l'arrel de l'arbre final, obté una posició relativa i sempre que ho necessiti pot buscar la informació que necessita a l'arbre final.

Capítol 4

Regles d'actuació

4.1. Cerca de l'arrel i creació de l'arbre

Aquesta part de l'algorisme està adaptat d'un algorisme fet en un projecte anterior (Veure [1] per més informació).

4.1.1. Objectiu

Es tracta de fixar, d'entre tots els mòduls que formen el robot, quin d'ells farà d'arrel.

L'arrel és el mòdul que inicia i finalitza l'enviament de dades per les fases posteriors.

A més a més, de forma simultània, els missatges usats per buscar l'arrel serveixen perquè els mòduls puguin detectar els cicles de l'estructura que trencaran per formar un arbre.

4.1.2. Estratègia

1. *Configuració Inicial.* Inicialment, els mòduls de la configuració inicial tenen estat [Start] i estan enganxats a tots els seus veïns.

Escollim com a arrel de l'arbre generador a construir el mòdul que, d'entre els que estan més a l'esquerra, està més amunt, tal i com podem veure a la Figura 1.

Al iniciar, determinem quins són els possibles candidats d'entre tots els mòduls. Per fer-ho, tots els que no en tenen cap altre connectat al nord ni al oest, es converteixen en candidats. Definim el seu estat com [CanbS]¹ i enviem una senyal amb la seva posició als veïns com a la Figura 2.

Aquest missatge es propaga pels mòduls veïns connectats, que actualitza la posició relativa al mòdul que ha creat el missatge i el reenvia a tots els veïns excloent el pare. El missatge és un nombre de 4 dígit, on els 2 primers indiquen la coordenada x de la posició relativa i els 2 restants la coordenada y . Aquests valors s'inicialitzaran a

¹La lletra **S** fa referència a **Start** per diferenciar la configuració inicial de la final.

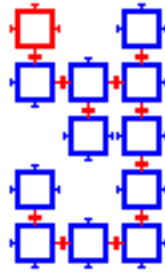


FIGURA 1. En aquest exemple de reconfiguració podem veure de color vermell, quin seria el mòdul arrel.

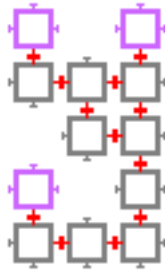


FIGURA 2. Podem veure de color lila, els candidats a arrel d'aquesta reconfiguració.

50 50. Així podem guardar qualsevol posició en un mapa de 100×100 en un valor de 4 dígits.

Aquests missatges crearan col·lisions en dos casos:

Si hi ha cicles, el missatge que col·lisiona serà el mateix, ja que ve del mateix candidat. Llavors el mòdul sap que hi ha un cicle en l'estructura. Usant la preferència de direccions ($S \rightarrow E \rightarrow W \rightarrow N$) el mòdul escull una de les direccions d'on han arribat els missatges per trencar el cicle.

Si tenim més d'un candidat a arrel, els missatges poden donar dues posicions diferents. El mòdul les compara, i es queda amb la millor posició. El mòdul del qual prové el missatge bo queda marcat com a pare, i el mòdul reenvia el missatge als altres veïns, inclòs el que havia enviat el missatge del candidat descartat. Així aconseguim que els missatges es propaguin per tota la configuració.

Quan un mòdul rep un missatge, canvia l'estat a **[ForwS]** i guarda la direcció del seu pare situat a la direcció d'on rep el missatge.

Finalment, quan un missatge arriba a una fulla, el mòdul canvia a l'estat **[BackS]** i retorna un missatge de confirmació al seu pare.

El pare no accepta cap missatge dels seus fills fins que no li han arribat de tots ells. Quan els tinguin tots, canvia també a estat **[BackS]** alhora que guarda en un

comptador les adreces de tots els seus fills i finalment retorna el missatge al seu pare.

Quan el candidat a arrel rep el missatge de **Back** de tots els seus fills, canvia l'estat a **[RootS]**, estat final de l'arrel.

2. *Configuració Objectiu.* L'elecció de l'arrel i la creació de l'arbre objectiu funciona amb regles anàlogues, només canvien els estats. Substituïm la **S** de **[Start]**, per **F** de **[Final]** per diferenciar-los.

Duplicuem el procés per diferenciar en tot moment els dos arbres, ja que aquest arbre només existeix per simular la informació que el robot tindria de l'arbre objectiu, però a la realitat no existeix aquest arbre físicament.

Així doncs, després de repetir tot el procés, quan obtenim l'arrel **[RootF]**, aquesta propaga un missatge que posa tots els mòduls en estat **[Slave]** i atura el procés.

Per més detalls, veure Annex A.

4.1.3. Modificacions

Aquest algorisme inicialment estava pensat per construir tan sols un arbre d'una configuració de robots. Per tant, hem hagut de fer alguns retocs perquè no hi hagués problemes al tenir dos arbres simultanis.

Tots els estats de construcció de l'arbre estan duplicats, així podem utilitzar els mateixos missatges. Per diferenciar els arbres, hem usat la terminació **S** i **F**, per definir mòdul de l'arbre inicial o final. Així, si un mòdul rep un missatge i ell té la terminació **S**, el mòdul seguirà les regles de l'arbre inicial (anàleg amb **F** per l'arbre objectiu).

A més a més, a l'algorisme inicial hi havia un conjunt de regles per acabar amb el procés. Aquestes regles les hem usat a l'arbre final però no amb l'arbre inicial, on les hem substituït per les de l'inici de la compressió, que explicarem a la següent secció.

4.2. Compressió

4.2.1. Objectiu

En aquesta fase, es vol comprimir les branques del arbre inicial, sense ocupar espais que inicialment no estiguessin ocupats. Per la manera com hem definit els mòduls, aquests poden modificar la seva forma a nivell atòmic per acceptar un altre mòdul a la seva posició i així alliberar espai per permetre el moviment dels altres mòduls.

4.2.2. Estratègia

Quan l'arrel passa al seu estat `[RootS]`, envia una senyal als seus fills perquè comencin la compressió. Aquests propaguen el missatge als seus fills alhora que canvien a l'estat `[Cmprs]`.

Si un mòdul `[Cmprs]` és fulla de l'arbre, demana al seu mòdul pare si pot comprimir-se a la seva posició. Si el seu pare li dona permís llavors es comprimeix utilitzant l'acció `Zip`.

El mòdul pare dona permís si no té cap altre mòdul comprimit, no està executant cap altra regla i, si té més d'un fill que li demana permís, el que tingui més preferència de direcció.

A més, quan dos mòduls estan comprimits, també demanen als seus pares si poden enviar un mòdul comprimit i quan reben confirmació, utilitzant l'acció `Swap-Zip` el mòdul amfitrió desplaça el convidat al seu pare. Repetint el procés, els mòduls es van comprimint cap a l'arrel i en el moment que el primer mòdul comprimit arriba a l'arrel, comença el procés d'expansió.

Per més detalls, veure Annex B.

4.2.3. Problemes

Tot i que aquesta fase és la més senzilla del procés, ja que no hi ha estats intermedis que ens puguin bloquejar en casos puntuals i només enviem missatges als nostres veïns directes, el fet que necessitem de tres passos per poder completar una compressió (demandar permís, donar el permís, i comprimir finalment), ha provocat alguns problemes de intrusió en el procés de compressió. Vegem-ho:

1. *Compressió simultània 1:* Té lloc quan dos mòduls volen comprimir-se a un mateix pare al mateix temps. Els dos mòduls envien el missatge per demanar permís i el pare, si en aquell moment no té cap mòdul comprimit a la seva posició, accepta els dos missatges, i llavors els dos mòduls volen comprimir-se a la mateixa iteració, provocant un col·lapse a la posició del pare.

Solució: Al establir prioritats de direccions, solucionem el problema. L'ordre establert és $(S \rightarrow E \rightarrow W \rightarrow N)$, així si el mòdul pare rep dos o tres missatges dels seus fills, només contestarà el missatge de la direcció amb més preferència.

2. *Compressió simultània 2:* Té lloc si un mòdul pare té dos fills, que es volen comprimir però amb un instant temporal de diferència. El mòdul més ràpid envia primer el missatge, i el pare l'accepta però, abans que es comprimeixi, l'altre fill envia un missatge per comprimir-se. Si aquest segon fill es troba en una direcció amb més preferència que el primer, el mòdul pare també li envia un missatge de confirmació. Per tant, el primer mòdul es comprimeix, i a l'instant següent, ho fa el segon fill, provocant altre cop un col·lapse a la posició del pare.

Solució: Afegim al pare un booleà que indica si està en procés de compressió: Quan rep el primer missatge, aquesta constant és certa, i al següent instant, aquesta constant és canviada a fals. Mentre la constant és certa, no accepta cap missatge

de moviment de cap altre mòdul. Aquest bloqueig d'un instant és suficient ja que al tercer pas, el primer mòdul es comprimeix abans de que el pare llegeixi els missatges, quan ho fa, per tant, el mòdul ja s'ha comprimit i no accepta més compressions.

4.2.4. Models de prova

Per comprovar tots els conflictes possibles de les compressions, s'han fet diferents configuracions dels robots que provoquen els problemes observats.

1. *Compressió simultània 1:* En els models de les figures 3 i 4 provoquem el problema de la compressió simultània 1. El mòdul pare escull quin dels fills que li envien missatges té més prioritat i li dóna preferència a fer la compressió. El fill que té el pare al sud és el de més preferència, llavors sempre és el primer a comprimir-se si es dóna el cas.

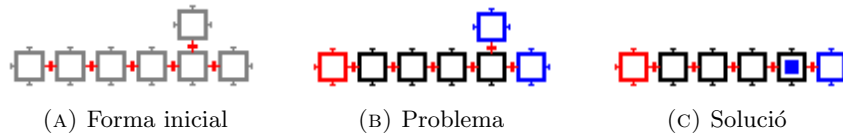


FIGURA 3. En aquesta configuració (A), els dos mòduls de color blau de (B) volen comprimir-se alhora. A la figura (C) veiem la posició final, on el mòdul del nord del pare s'ha comprimit.

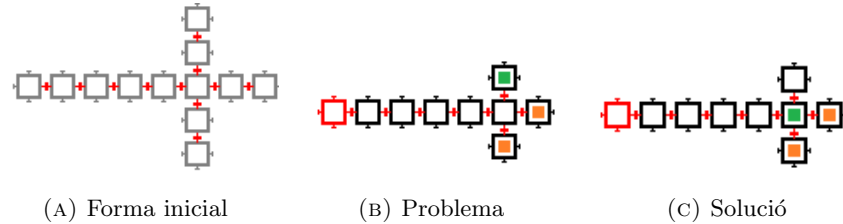


FIGURA 4. En un instant d'aquesta compressió (B), tres mòduls (el verd i els taronges) volen comprimir-se alhora. A l'imatge (C) podem veure com es comprimeix primer el mòdul veí del nord (verd) del pare, mentre els altres esperen (taronges).

2. *Compressió simultània 2:* Un model per veure la compressió simultània 2 és la Figura 5. El mòdul que ve de l'est arriba abans que el del nord, llavors, tot i que el mòdul del nord té més preferència, el pare accepta al de l'est. Abans de fer la compressió, arriba el mòdul del nord i demana permís, però el pare no l'accepta perquè espera un altre mòdul.

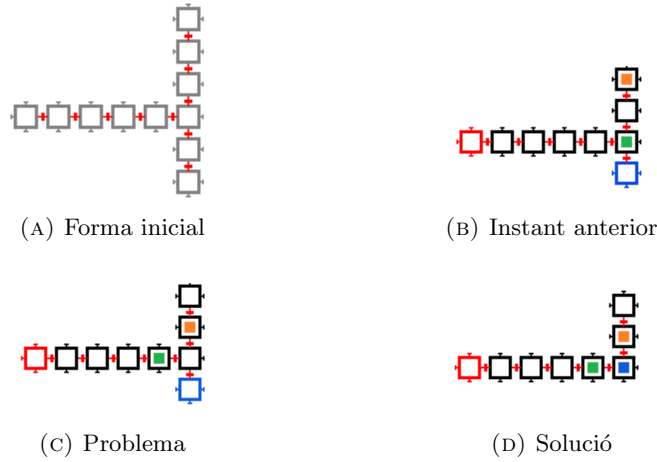


FIGURA 5. En un instant (B) d'aquesta compressió, un mòdul comprimit (verd) en una intersecció deixa la seva posició. El mòdul que està al sud de la intersecció (blau) demana permís per comprimir-se, i se li dóna al mateix moment que arriba un mòdul comprimit del nord (taronja). Així doncs, al moment del problema (C), tenim un mòdul que ja té permís per comprimir-se, i un altre amb més preferència que demana comprimir-se. A (D) podem veure com primer hem completat la compressió que ja s'havia permès (blau) i no s'ha provocat un col·lapse.

4.3. Expansió

4.3.1. Objectiu

Quan l'arrel comparteix espai amb un altre mòdul comprimit, està en condicions de començar la construcció de la forma final.

Primer de tot volem crear un líder, l'únic mòdul que coneix la informació de l'arbre final i que ha de recórrer tot l'arbre, seguit per tots els altres mòduls en compressió.

El líder busca les posicions de l'arbre final que falten per omplir, quan li arribi un mòdul comprimit, el col·loca a la posició corresponent i repeteix el procés fins a completar l'arbre.

Si algun mòdul bloqueja el seu moviment, l'afegeix a l'arbre final i aquest al mateix temps trenca el seu contacte amb el seu antic pare i en conseqüència el cicle format.

4.3.2. Estratègia

Primer simulem que un mòdul coneix la informació de l'arbre objectiu, per fer-ho l'arrel [RootS] busca en direcció Est, un mòdul amb estat [RootF] l'arrel de l'arbre objectiu. La distància entre les dues arrels l'emmagatzema el líder, que

inicialment està en la posició de l'arrel. El **líder** doncs, quan necessiti informació de l'arbre final, busca el mòdul que està a aquesta distància cap a l'est.

Un cop feta la connexió entre el **líder** i l'arbre objectiu, l'arrel té estat `[RootL]`² indicant que fa les dues funcions, arrel i **líder**.

Quan arriba el primer mòdul comprimit a l'arrel, comença l'acció d'expansió del **líder**. Copia la informació del mòdul en la seva posició relativa a l'arbre objectiu, i l'usa per saber en quines direccions ha d'expandir-se.

El mòdul comprimit passa a estat `[Expnd]`³ indicant que ja està fixat en una posició correcta de l'arbre objectiu. El **líder** es mou cap a la direcció preferent de totes les escollides, i el mòdul **Expand** guarda la direcció per enviar tots els mòduls comprimits que arribin a la seva posició.

Al moure el **líder**, ens podem trobar en dos casos diferents:

- **Si la direcció a ocupar és buida:** El **líder** s'expandeix, i marca la seva anterior posició com a pare.
- **Si la direcció està ocupada:** El **líder**, envia un missatge al mòdul, per confirmar que pot fer el canvi. Sempre que un mòdul rep un missatge, finalitza totes les accions que estigui portant a terme. Quan acaba amb totes les accions, el mòdul ha de comprovar si està connectat amb el **líder** i, si no ho està, es connectarà a ell, i es desconnectarà del seu antic pare per no formar un cicle. Finalment, el mòdul marca al líder com el seu pare i li envia un missatge de confirmació per fer el canvi de líder, que s'executa de forma instantània.

Cal tenir en compte que un mòdul **líder** només pot rebre mòduls comprimits dels seus veïns en **expand**, i un mòdul **expand** només rep els mòduls que provenen de un **compress** quan no arriben mòduls des de el seu pare **Expand**. Per tant, en el cas que s'uneixin parts de l'arbre de compressió a l'arbre objectiu sempre té preferència l'expansió. Així evitem possibles embossos que es puguin crear a l'arbre.

Quan el **líder** arriba a una posició de fulla en l'arbre final, retorna un missatge al seu pare per canviar de líder (que s'executa al moment). Aquest missatge el retorna fins que arriba a un mòdul que té alguna branca a l'arbre final que encara no s'ha creat.

Per més detalls, veure Annex C.

4.3.3. Problemes

Al crear models de prova amb aquesta fase es quan trobem la majoria de problemes, ja que la compressió també s'està executant.

1. *Bloqueig de compressió 1:* Com el **líder** fa totes les accions abans que la resta, pot provocar algun problema en la compressió. Suposem un cas on el mòdul **líder**

²El líder sempre està indicat amb l'estat `[LIDER]` excepte en el cas que sigui l'arrel que és `[RootL]`.

³En el cas que sigui la posició de l'arrel, l'estat és `[RootS]` altre cop, tot i que fa les mateixes funcions que l'**Expand**.

vol comprimir-se a un mòdul connectat en estat [Cmprs] no comprimit. Aquest mòdul **compress** vol rebre un mòdul del seu fill en la compressió, i li envia el missatge de confirmació alhora que rep un missatge del líder. Al pas següent, aquest mòdul es converteix en el **líder**, i per tant no accepta el mòdul de la compressió, provocant un error en aquest procés.

Solució: El mòdul **líder** sí que accepta els mòduls que es vulguin comprimir, però no confirma cap compressió per part de un mòdul **Compress**, és a dir, aconseguim que el **líder** segueixi sense acceptar cadenes de mòduls de part de la compressió, però no aturem una compressió ja començada.

2. *Bloqueig de compressió 2:* Suposem un cas on el **líder** es vulgui expandir en direcció a un mòdul fulla no comprimit en estat de [Cmprs], on aquest vol comprimir-se al seu pare de compressió diferent del **líder**. El mòdul en **compress** envia un missatge al seu pare per comprimir-se, alhora que rep un missatge del **líder** per canviar d'estat, es connecten i tallen el cicle. El pare del mòdul **compress** canvia les constants per saber que estarà comprimit, i envia un missatge de confirmació, però el fill ja no està connectat, per tant no pot fer la compressió. En aquest cas, el mòdul pare quedava amb les constants errònies.

Solució: Quan un mòdul rep un missatge de confirmació per comprimir-se i no pot fer-ho, retorna un missatge per indicar que no ho ha fet. Així, si un mòdul rep un missatge de negació, reinicia les seves constants.

3. *Compressió prematura:* Repetim el cas anterior, però en aquest cas la compressió va un iteració per davant. El **líder** comprova que en la direcció en què ha d'avançar hi ha un mòdul i envia el missatge i tot seguit envia el lideratge en la direcció, ja que el líder té la preferència més alta en les seves accions. Si el comprimit va un instant avançat, quan envia el **líder**, ell ja ha fet la compressió i per tant, el **líder** s'envia a una posició buida.

Solució: Sempre que un líder envii l'informació, esperarà un missatge de confirmació del nou líder, si no rep el missatge, tornarà a ser ell el líder.

4.3.4. Models de prova

1. *Bloqueig de compressió 1.* El bloqueig de compressió 1 el podem veure a la Figura 6. El **líder** vol passar el lideratge al mòdul en compressió de l'est, que està descomprimit, però ha acceptat al seu fill la compressió. A l'instant següent, aquest mòdul té estat [LIDER] i també rep el mòdul comprimit.

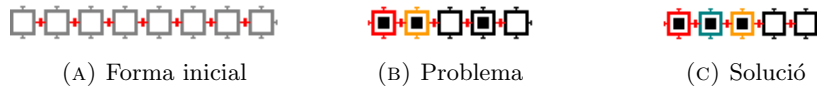
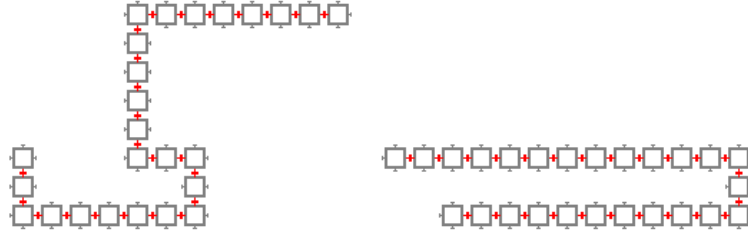
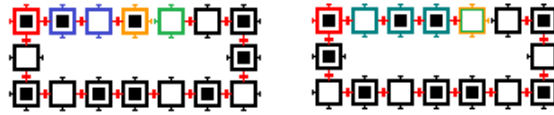


FIGURA 6. En l'exemple, el mòdul taronja a (B) és el **líder** de la configuració: Vol moure's a l'est on un mòdul vol rebre un comprimit. Podem comprovar que a l'instant següent (C), el mòdul s'ha comprimit i ara té estat [LIDER].

2. *Bloqueig de compressió 2.* Vegem el bloqueig de compressió 2 amb un model que podem veure a la Figura 7. Podem observar un mòdul en estat de compressió que, com és la fulla de l'arbre, vol comprimir-se cap al seu pare al nord. Al mateix moment, els mòduls en expansió que provenen de l'oest li aturen el moviment. Després del moviment d'expansió, el seu pare és una fulla de l'arbre de compressió i no queda comprimit.



(A) Forma inicial i final

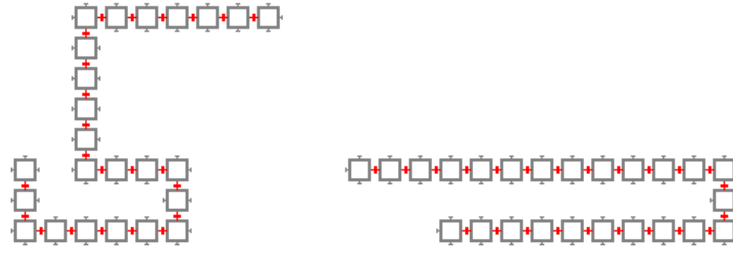


(B) Problema

(C) Solució

FIGURA 7. En un instant d'aquest exemple (B), tenim un mòdul fulla (verd) que es vol comprimir a l'est i ja ha enviat la petició, al mateix instant que el líder (taronja) li vol passar el lideratge. Podem veure a (C) com el mòdul ha passat a ser el líder i no ha fet la compressió, a més de connectar-se al antic líder i desconnectar el seu pare.

3. *Compressió prematura.* En aquest model de la Figura 8 podem observar un mòdul en estat de compressió que com és la fulla de l'arbre, vol comprimir-se cap al seu pare al nord. Al instant següent, el mòdul líder en expansió que prové de l'oest li vol donar el lideratge. A continuació el mòdul fa la compressió, i el líder perd el seu candidat, i per tant, torna a recuperar el líder al següent pas.



(A) Forma inicial i final

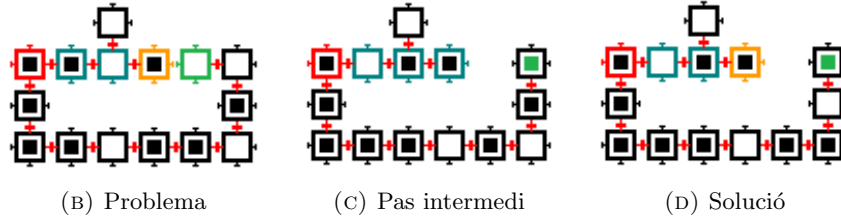


FIGURA 8. En aquest exemple, semblant a l'anterior, tenim la diferència de que en l'instant (B) el mòdul que és vol comprimir (verd) ja ha rebut el permís. El líder (taronja) li vol donar el lideratge però llavors completa la compressió. En l'instant (C), l'antic líder espera una resposta del mòdul i al no rebre resposta, torna a agafar el lideratge a (D).

Capítol 5

Correcció i complexitat

Volem demostrar que l'algorisme donat és correcte, o sigui que amb les nostres regles aconseguim formar un arbre generador inicial amb l'arrel proposada, que manté l'estructura d'arbre generador a cada pas i que aconsegueix formar l'estructura final. També és demostra que aquest algorisme necessita $O(n^2)$ moviments atòmics per completar-se, executats en $O(n)$ passos paral·lels.

5.1. Definicions bàsiques

Definim una malla 2D on cada casella correspon a la mida d'un mòdul d'un robot expandit, és a dir, el que ocupen quatre àtoms expandits connectats formant un quadrat 2×2 . Les caselles contenen un enter $\{0,1,2\}$ que indica el nombre de mòduls que ocupen aquella posició; així una 2-casella vol dir que hi ha dos mòduls a la mateixa posició. En una casella amb 2 mòduls distingirem l'amfitrió i el convidat: l'amfitrió és el mòdul que ocupava la 1-casella abans de convertir-se en una 2-casella; el convidat és el mòdul que es comprimeix a la posició de l'amfitrió.

Definim doncs, els conjunts S i F com les cel·les que ocupen els arbres d'expansió formats pels mòduls amfitrió. Inicialment es corresponen amb les configuracions **Start** i **Final** dins d'aquesta malla; així com un mòdul especial l_0 que té accés a la informació de l'arbre F . La creació de l'arbre s'executa en temps lineal respecte del nombre de mòduls amb comunicació local usant només l'espai unió $S \cup F$.

Mantenim l'estructura d'arbre en tot l'algorisme, fent que els mòduls amfitrió estiguin connectats en tot moment. A més, les caselles de S i F mantindran una relació pare-fill dins l'arbre: per cada casella $c \in S$ definim la seva casella pare $P(c) \in S$ com aquella que aconsegueix que, pel node amfitrió n que ocupa c , el pare en l'arbre és el node amfitrió que ocupa $P(c)$.

Si tenim dos cel·les c_1, c_2 definim les següents operacions:

- $\text{ZIP}(c_1, c_2)$: S'usa quan c_1, c_2 estan ocupades per mòduls n_1 i n_2 respectivament i es té que $c_2 = P(c_1)$, el mòdul n_1 és una fulla i n_1 i n_2 estan descomprimits. Després de l'acció, c_1 queda buida i c_2 passa a ser una 2-casella, sent n_2 l'amfitrió de n_1 .

- $\text{UNZIP}(c_1, c_2)$: S'aplica quan c_2 és una 2-casella i c_1 és buida. És la funció inversa de ZIP .
- $\text{SWZIP}(c_1, c_2)$: Si c_1 és una 2-casella i c_2 és una 1-casella, el convidat de c_1 es mou a c_2 .
- $\text{ATTACH}(c_1, c_2)$: Quan els dos mòduls amfitrió de c_1 i c_2 estan desconnectats, es connecten físicament.
- $\text{DETACH}(c_1, c_2)$: Si els dos mòduls amfitrió de c_1 i c_2 estan connectats físicament, trenquen la connexió.

A l'Annex D podem veure el detall atòmic d'aquests moviments. El robot també pot enviar i rebre missatges dels veïns en temps $O(1)$, però l'enviament de missatges no provoca canvis en l'estructura del robot.

En el nostre algorisme, primer trobem un arbre generador de l'estructura del nostre robot, procés que és clar que podem fer en temps lineal respecte el nombre de mòduls. Veiem que els nostres moviments bàsics, no modifiquen aquesta estructura d'arbre, és a dir, que al final de cada operació, obtenim com a resultat un arbre.

LEMA 5.1 *Les operacions: ZIP , UNZIP i SWZIP mantenen l'estructura d'arbre del robot i poden ser executades en temps $O(1)$.*

DEMOSTRACIÓ. Al text [3] podem verificar que els nostres moviments bàsics en forma atòmica no provoquen desconexions de forma individual, i s'explica una solució per evitar les possibles desconexions que poden causar els àtoms de dos mòduls veïns al fer un moviment al mateix instant.

Un cop comentada l'estructura atòmica, veiem que els nostres moviments no afecten a l'estructura d'arbre. L'operació SWZIP només modifica les posicions dels mòduls convidats, pel que no afecta l'arbre general. L'operació ZIP treu una fulla de l'arbre S , i UNZIP afegeix una fulla, pel que cap dels dos pot desconnectar l'estructura, i continua essent un arbre.

Finalment, vegem fàcilment la complexitat de les operacions bàsiques: donat que qualsevol moviment d'àtoms entre dos mòduls pot ser realitzat en temps lineal respecte a la mida dels mòduls, la complexitat de les operacions és de $O(1)$ per cada mòdul. $\square$

Ara que ja tenim definides les operacions bàsiques, podem començar a demostrar l'efectivitat del nostre algorisme.

En primer lloc, al llarg de les Seccions 5.2 a 5.4 veiem la correcció i complexitat d'una versió simplificada de l'algorisme.

A la Secció 5.5 expliquem la millora posterior i com adaptar les demostracions anteriors.

5.2. Compressió

A la primera fase, repetim la funció PASCOMPRS constantment per moure els mòduls cap a l'arrel l_0 . És a dir, comprimim les fulles cap als seus pares i les movem en direcció a l'arrel per dins de l'arbre. Si repetim aquest procés infinitament al final obtenim un arbre on totes les posicions de S que no són fulles, són 2-caselles. Tot i així, per començar la segona fase no cal esperar a que l'arbre es comprimeixi totalment, només cal esperar a que l'arrel sigui una 2-casella tal com s'explicarà a la Secció 5.5. Aplicarem aquestes dues funcions:

PASCOMPRS(s):

Per totes les caselles $n \in S$, excepte per l_0 , executar en paral·lel:

Si $P(n)$ és una 1-casella & n és el fill de més prioritat de $P(n)$
& tots els fills de $P(n)$ són fulles o 2-caselles:

 Si n és una 2-casella: SWZIP($n, P(n)$).

 Si n és una 1-casella fulla: ZIP($n, P(n)$).

DEFINICIÓ 5.2. Un arbre *totalment comprimit* és aquell en que tots els seus nodes són 2-caselles o fulles.

COMPRS(s):

Repetir fins que S estigui totalment comprimit:

$S \leftarrow \text{PASCOMPRS}(S)$.

Tal i com hem dit al paràgraf anterior, la restricció de *que S estigui totalment comprimit* no es necessària però simplifica l'estudi de l'algorisme. A més, la condició de que l'acció de PASCOMPRS(s) només s'executi quan *tots els fills de $P(n)$ són fulles o 2-caselles* és un afegit a l'algorisme que en redueix la velocitat. A la Secció 5.5 es demostra que el nostre algorisme és tant o més eficaç que aquest.

Amb aquestes dues funcions podem trobar algunes propietats de l'algorisme, que ens serviran per comprovar la consistència dels processos.

LEMA 5.3 *sigui S el conjunt de caselles ocupades per un conjunt de mòduls connectats en un arbre, llavors PASCOMPRS(s) retorna un conjunt de cel·les associat a un arbre que conté el mateix nombre de mòduls i manté la connectivitat. També és cert per COMPRs(s).*

DEMOSTRACIÓ. PASCOMPRS usa dues operacions bàsiques: ZIP i SWZIP. Pel Lema 5.1, aquestes operacions mantenen l'arbre. Quan una casella c està involucrada en ZIP o SWZIP, sabem que $P(c)$ és una 1-casella no-fulla i per tant, no està involucrada en cap altra operació en paral·lel. A més, totes les caselles estan involucrades

com a màxim en una operació en cada unitat de temps. Per tant, la demostració per COMPRES(s) és evident¹. $\square$

DEFINICIÓ 5.4. L'alçada d'una casella de S és la longitud del camí més llarg fins a una fulla del sub-arbre que el té com arrel. Per tant, les fulles tenen alçada 1.

LEMA 5.5 *Segui $r \in S$ amb alçada $h \geq 2$. A la iteració $h - 1$ de COMPRES(s), la cel·la de r esdevé una 2-casella per primer cop.*

DEMOSTRACIÓ. Abans de la primera iteració, l'espai ocupat per S són tot 1-caselles. Ho demostrem per inducció en h .

En el cas base $h = 2$, els fills de r són fulles. A la primera iteració del PASCOMPRES, la fulla amb més prioritats es comprimeix cap a r .

Vist el cas base, ho suposem per nodes d'alçada $h - 1$ i ho volem veure pels nodes d'alçada h . Suposem doncs que r té com a mínim un fill amb alçada $h - 1$, que anomenarem f , i que per hipòtesi d'inducció, a la iteració $h - 2$ es converteix en una 2-casella. Aleshores, a la següent iteració, r rep per primer cop un mòdul amb l'acció PASCOMPRES. $\square$

DEFINICIÓ 5.6. Definim la profunditat d'una casella com la distància a l'arrel. Així, l'arrel tindrà profunditat 0. La profunditat de l'arbre h_0 és el màxim de les profunditats de totes les seves caselles.

LEMA 5.7 *A la iteració i de COMPRES(s) considerem r una 2-casella amb alçada h que en aquesta iteració envia un mòdul a $P(r)$. Llavors, al final de la iteració $i + 1$, r és una fulla o una 2-casella altre cop.*

DEMOSTRACIÓ. Ho demostrarem per inducció sobre h . Considerem el cas base $h = 2$. Al final de la iteració i , tot fill de r és fulla, i a la següent iteració $i + 1$ un d'ells es comprimeix cap a r .

Per $h > 2$, considerem la iteració $j < i$ on r rep el mòdul convidat que després enviarà cap a $P(r)$ al pas i . A l'inici de la iteració j tots els fills de r són fulles o 2-caselles. Segui f el fill que envia el mòdul cap a r :

- Si f ha usat ZIP, llavors r té un fill menys, però tots els altres fills continuen sent fulles o 2-caselles fins la iteració $i + 1$.
- Si f ha usat SWZIP llavors, per hipòtesi d'inducció, al final de la iteració $j + 1 \leq i$, f és una fulla o una 2-casella.

Per tant, a la iteració $j + 2 \leq i + 1$, r aconsegueix totes les condicions per rebre un mòdul llevat la de ser 1-casella, condició que s'assoleix quan a la iteració i envia el mòdul convidat cap a $P(r)$. $\square$

LEMA 5.8 COMPRES(s) acaba com a màxim en $2h_0 - 1$ iteracions de PASCOMPRES.

¹Al simulador hi ha iteracions en què un mòdul executa més d'una regla, però per construcció només una d'elles inclou una acció, les altres corresponen a manipulació d'estats, missatges i/o constants.

DEMOSTRACIÓ. Vegem que al completar la iteració $h_0 - 1 + d$, tots els mòduls a profunditat menor o igual que d de S són 2-caselles o fulles. Demostrem-ho per inducció sobre d . El cas base és el de l'arrel a $d = 0$. Pel Lema 5.5, l'arrel esdevé una 2-casella a la iteració $h_0 - 1$.

Ara considerem una casella p a profunditat $d - 1$ tal que té un fill, ja que si és una fulla ja hem acabat. Per hipòtesi d'inducció, p i totes les caselles amb menys profunditat són 2-caselles al final de la iteració $i = h_0 - 1 + (d - 1)$, i p ha estat l'última cel·la en convertir-se en 2-casella d'entre totes aquestes. Durant la iteració i , només el fill f amb més prioritats canvia, al enviar un mòdul a p , ja sigui amb ZIP si és una fulla o amb SWZIP si és una 2-casella. Pel Lema 5.7, si f era una 2-casella, al acabar la iteració $i + 1$ serà fulla o 2-casella. A més, tots els altres fills de p (que eren 2-caselles o fulles) es queden igual perquè tots els avantpassats de p ja eren 2-caselles i per tant p no tornarà a ser una 1-casella.

Així si fem $d = h_0$, o sigui el cas de la casella amb més profunditat, el lema queda demostrat. $\square$

DEFINICIÓ 5.9. Direm que un arbre té l'*arrel comprimida* quan no té dues 1-caselles no-fulles connectades.

LEMA 5.10 *Sigui S el conjunt de cel·les associat a un arbre amb l'arrel comprimida. Llavors, després d'un PASCOMPRS(S), S continua sent un arbre amb l'arrel comprimida.*

DEMOSTRACIÓ. Considerem qualsevol casella $c \in S$. Volem veure que després d'una iteració de PASCOMPRS o bé és una 2-casella, o bé és una fulla, o bé és una 1-casella que només està connectada a 2-caselles i/o fulles.

- Si PASCOMPRS fa que c sigui una 2-casella o una fulla ja ho tenim.
- Si c envia un mòdul cap a $P(c)$, cap dels seus fills pot enviar un mòdul cap a c , pel que c és una 1-casella i $P(c)$ és una 2-casella. Considerem doncs y una cel·la fill de c que és 1-casella abans de la iteració de PASCOMPRS. Com S és un arbre amb l'arrel comprimida, cap fill de y pot ser una 1-casella i aleshores, al fer PASCOMPRS, y rep un mòdul d'algun dels seus fills i passa a ser 2-casella.

Per tant, S continua essent un arbre amb l'arrel comprimida. $\square$

5.3. Expansió

Després de la primera fase de compressió volem que el mòdul l_0 , que té la informació sobre F , faci un recorregut DFS² per tot F mentre el completa amb els mòduls que falten. Els altres mòduls el segueixen, repetint PASCOMPRS, fins que l_0 els dona una ordre diferent. El mòdul expandeix F usant els mòduls que arriben a la seva posició.

Vegem l'algorisme:

²Depth First Search

```

PASEXPNS( $s_l, F$ ):

d = Nova casella a visitar del DFS en F.
c = 2-casella on  $l_0$  és un mòdul convidat.

PAS 1: Si d és una 0-casella: UNZIP(c,d).

PAS 2: Si d no és una 0-casella:
      Si  $c \neq P(d)$ : ATTACH(c,d) and DETACH(d,P(d));
      SWZIP(c,d).

PAS 3: P(c) = d;
      P(d) = NULL;
      Marcar c com "visitada";
      Afegir (c,d) a  $S_l$ .

PAS 4: Mentre d no sigui una 2-casella, repetir:
      (a) Per totes les 1-caselles fulla  $u \in S_l$ , fer:
          Si u està marcada "visitada", eliminar u de  $S_l$ .
      (b) PASCOMPRS( $S_l$ ).
      Si  $l_0$  no és un mòdul convidat fer: SWITCH(d).

```

La funció SWITCH apareix a l'Annex D, però és un moviment atòmic que no usem al nostre algorisme.

```

EXPNS( $S, F$ ):

 $S_l = S$ ;
Repetir fins que  $l_0$  arribi a la posició final del recorregut DFS:
   $S_l \leftarrow \text{PASEXPNS}(S_l, F)$ .

```

Al repetir PASEXPNS, el mòdul líder busca les posicions que falten per completar F i les caselles que queden a S_l són les 2-caselles o les 1-caselles per on el líder encara no ha passat. Els mòduls d'aquestes es mouen cap al líder i així poden expandir-se fins a completar l'arbre F .

5.4. Reconfiguració

Ara que ja tenim les fases de compressió i expansió, podem construir el nostre algorisme de reconfiguració:

```

RECONFIG( $S, F$ ):

fase 1:  $S \leftarrow \text{COMPRS}(S)$ ;
fase 2:  $\text{EXPNS}(S, F)$ ;

```

LEMA 5.11 *L'algorisme de RECONFIG manté l'arbre connectat físicament.*

DEMOSTRACIÓ. Ja hem vist al Lema 5.3, que la funció COMPRS retorna un arbre S amb tots els mòduls inicials del robot. Per tant, falta veure que la funció EXPNS manté l'estructura d'arbre si l'entrada ho és:

- Al pas 1, UNZIP manté l'arbre per el Lema 5.1.
- Al pas 2, d ja és un mòdul de S . Si $c \neq P(d)$, connectem c amb d , creant un cicle que immediatament trenquem desconnectant d i $P(d)$. Finalment fem el SWZIP per avançar el líder l_0 que pel Lema 5.1 sabem que manté la connectivitat³.
- El pas 3 no afecta la connexió física de l'arbre ja que només s'hi actualitzen dades⁴.
- Finalment al pas 4, S també conserva l'estructura d'arbre ja que:
 - Al pas 4(a) només fem canvis a l'arbre temporal S_l .
 - A 4(b), PASCOMPRS manté l'arbre pel Lema 5.3.

L'únic que falta demostrar és que sempre que una 1-casella fulla c' fa un ZIP a la funció PASCOMPRS del pas 4, c' mai ha estat visitada pel líder ja que no volem $S \neq S_l$. Però és suficient veure que S_l mai conté una 1-casella fulla que ja ha estat visitada. Al aplicar PASCOMPRS podem tenir 2 casos: Si aquesta casella és una 1-casella fulla al començar el pas 4 i ha estat visitada, l'eliminem de S_l . Si no ho ha estat, vol dir que $c' \neq d$ i llavors pot fer ZIP per convertir-se en un mòdul convidat del seu pare. Si és una 2-casella fulla i el líder no està en la seva casella, al fer PASCOMPRS només pot fer el moviment SWZIP, altrament no podrà fer res.

□

A la fase de COMPRS, produïm un arbre amb l'arrel comprimida, amb l_0 en una 2-casella. Vegem que la fase d'EXPNS manté aquesta propietat, encara que l_0 es mogui:

LEMA 5.12 *PASEXPNS manté S_l com un arbre d'arrel comprimida on l_0 és una 2-casella. A més a més, el procediment es pot fer en $O(1)$.*

DEMOSTRACIÓ. La demostració es semblant a la del Lema 5.10. Ja que l'estructura S_l és idèntica a l'estructura de S , amb l'excepció que algunes branques han estat retallades, i el Lema 5.11 demostra que S_l està connectat físicament.

Sigui S_l^i l'arbre enviat al PASEXPNS a la iteració i . Al Pas 1 o Pas 2 de la funció, l'arbre S_l^i és modificat físicament, però pel Lema 5.11 els canvis resulten un nou arbre que anomenem S_l^{i+1} . Aquests passos només afecten les caselles c i d . Si ara c és una 2-casella ja està. Si és una 1-casella com provenien d'un arbre d'arrel comprimida, o els fills de c són fulles o 2-caselles, o els fills d'aquests tenen aquesta propietat. El pas 4(a) només afecta a les 1-casella fulles visitades, així no afecta a la propietat de l'arrel comprimida de l'arbre. Podem tenir dos casos al pas 4(b):

³A la pràctica, al simulador intercanviarem estats entre els mòduls.

⁴Cal destacar que la relació pare-fill es defineix d'aquesta manera per tal que quedin correctament indicats quan el líder retorna per una branca durant el DFS.

- Si S_l^{i+1} s'ha obtingut al fer un UNZIP, llavors c i d són 1-caselles al començar el pas 4. Si tots els fills de c són 2-caselles o fulles, al fer la funció PASCOMPRS, c torna a ser una 2-casella i per tant està ben comprimit. Si no és així, algun dels fills de c és una 1-casella, definim c' a un d'aquests i veiem que després d'un PASCOMPRS, esdevé una 2-casella. Però com S_l és un arbre d'arrel comprimida, si c' és una 1-casella els seus fills són tots 2-caselles o fulles, per tant, al fer el Pas 4, un d'aquests fills es mou cap a c' i pel Lema 5.10 el sub-arbre format pels descendents de c' també ho és. A més, en el cas pitjor, si fem dues iteracions més al Pas 4, un mòdul arriba a d formant una 2-casella.
- Si S_l^{i+1} s'obté al fer un SWZIP, aleshores d ja és una 2-casella al començament del pas 4, tots els mòduls que segueixen a l_0 fan el PASCOMPRS i ja tindrem un arbre amb arrel comprimida pel Lema 5.10.

Així doncs, el temps d'execució de PASEXPNS és $O(1)$ perquè ja hem vist que les accions UNZIP i PASCOMPRS ho són pels lemes 5.1 i 5.8. $\square$

TEOREMA 5.13 *L'algorisme de RECONFIG permet reconfigurar qualsevol conjunt connex de mòduls en qualsevol altre amb el mateix nombre mòduls, i la reconfiguració és duu a terme en $O(n)$ passos paral·lels. Cada mòdul executa $O(n)$ moviments i comunicacions amb els seus veïns*

DEMOSTRACIÓ. Usant el Lema 5.8, la fase 1 usa temps $O(n)$. Pel Lema 5.12 cada iteració del PASEXPNS es pot fer en temps constant, per tant EXPNS usa temps $O(n)$ ja que el DFS és una cerca de com a molt $O(n)$. $\square$

5.5. Millora de l'algorisme

Un cop descrit i analitzat l'algorisme anterior, hem fet unes quantes modificacions per millorar-lo. Vegem que l'algorisme modificat manté las propietats de l'anterior, afegint la capacitat d'executar-se en un entorn asíncron.

Vegem l'algorisme:

PASCOMPRS2(s):

Per totes les caselles $n \in S$, excepte per l_0 ,
executar en paral·lel:

Si (n és una 1-casella fulla | n és una 2-casella) & $P(n)$ és una 1-casella:

n demana permís per comprimir.

Si n és una 1-casella:

De tots els missatges rebuts per comprimir, escull el de la direcció amb més prioritats i envia confirmació.

Si $P(n)$ és una 1-casella & n rep confirmació:

Si n és una 2-casella: SWZIP($n, P(n)$).

Si n és una 1-casella fulla: ZIP($n, P(n)$).

COMPRS2(s):

Repetir fins que S estigui totalment comprimit:
 $S \leftarrow \text{PASCOMPRS2}(S)$.

PASEXPNS2(s_l, F):

d = Nova casella a visitar del DFS en F .
 c = 2-casella on l_0 és el mòdul amfitrió.

PAS 1: Si d és una 0-casella:

UNZIP(c, d).

Enviar l'estat de líder de c a d .

PAS 2: Si d no és una 0-casella:

c envia missatge a d demanant permís per comprimir-se.

Si d rep un missatge per comprimir i no ha d'executar cap moviment fer:

d envia missatge a c confirmant compressió.

Si c rep un missatge de confirmació:

Si $c \neq P(d)$: ATTACH(c, d) and DETACH($d, P(d)$);

SWZIP(c, d).

Enviar l'estat de líder de c a d .

PAS 3: $P(c) = d$;

$P(d) = \text{NULL}$;

Marcar c com "visitada";

Afegir (c, d) a S_l .

PAS 4: Mentre d no sigui una 2-casella, repetir:

(a) Per totes les 1-caselles $u \in S_l$, fer:

Si u està marcada "visitada", eliminar u de S_l .

(b) PASCOMPRS2(S_l).

EXPNS2(S, F):

$S_l = S$;

Repetir fins que l_0 arribi a la posició final del recorregut DFS:

$S_l \leftarrow \text{PASEXPNS2}(S_l, F)$.

RECONFIG2(s, F):

fase 1: $S \leftarrow \text{COMPRS2}(S)$;

fase 2: EXPNS2(S, F);

Es pot apreciar que els canvis a la funció PASCOMPRS2 acceleren el procés en alguns casos, ja que en arribar a un pare amb més d'un fill no cal esperar a la resta de fills per enviar el mòdul convidat.

Això provoca que el nostre arbre no sigui exactament un arbre d'arrel comprimida, tot i que ho seria si executéssim PASCOMPRS2(T) un nombre suficient (finit) de vegades. Ara bé, és immediat comprovar que sempre és possible aplicar alguna regla, en altres paraules, sempre hi ha alguna compressió a fer (fulles) o a avançar, de tal manera que els mòduls arriben sempre a la fase d'expansió.

Es pot pensar que el fet que l'arbre no sigui d'arrel comprimida pot fer que el temps necessari per tal que l_0 torni a ser 2-casella quan hem fet una expansió no sigui $O(1)$, però sabem que com a màxim un mòdul pot trigar el temps que hem guanyat en accelerar la compressió. Per tant, en el cas pitjor, l'algorisme perdrà tot el temps que ha guanyat en la compressió, esperant els mòduls que han d'arribar a l_0 .

Cal remarcar que ara per tal de fer una operació bàsica triguem 3 iteracions enlloc d'una, però el canvi és proporcional de manera que encara tenim un temps d'execució de reconfiguració de $O(n)$ i, de fet, dues de les iteracions tenen un cost inferior ja que no fan operacions bàsiques, sinó que només envien missatges. A més, hem guanyat la capacitat de executar el programa en un context asíncron, ja que, encara que no tots els mòduls estiguin a la mateixa iteració, totes les accions que impliquen moviment dels mòduls estan precedides per uns senyals de confirmació que els mòduls no acceptaran si ja han acceptat un altre moviment.

Capítol 6

Creació d'un nou algorisme

En aquest capítol volem introduir un nou algorisme de reconfiguració intentant millorar l'estudiat al llarg d'aquest treball. La idea inicial és utilitzar les constants del robot, per comptar quants mòduls necessita enviar el robot per cada branca de l'arbre objectiu per així evitar moviments innecessaris al seguir constantment un líder. Per fer-ho, necessitem que els mòduls puguin accedir a la informació de l'arbre objectiu, és a dir, necessiten la capacitat d'emmagatzemar tota la informació de les posicions de l'estructura final. Tot i així, per un robot modular és molt més costós fer un moviment físic que enviar un missatge o emmagatzemar dades. Per tant, reduïm el nombre de moviments a canvi d'emmagatzemar més informació als mòduls.

Les fases de l'algorisme són les mateixes que per l'anterior però modificant les accions. Vegem aquestes modificacions.

6.1. Cerca de l'arrel i creació de l'arbre

Usem el mateix procés, però necessitem obtenir algunes dades dels arbres: el nombre de fills que té cada mòdul i el nombre de fills que té cadascun dels respectius fills per evitar un enviament constant de missatges pare-fill per obtenir la informació.

En aquesta fase, en que recorrem tot l'arbre, aprofitem aquesta cerca per obtenir les dades esmentades. Vegem el procés a seguir:

- (1) Escollim els candidats a arrel i aquests comencen l'enviament de missatges per trobar el millor candidat.
- (2) Els missatges es propaguen entre veïns desconnectant els cicles o descartant els candidats que reben missatges de candidats millors.
- (3) Si aquests missatges arriben a una fulla:
 - La fulla activa 5 constants: una per comptar el seu nombre de fills (0 perquè és fulla), i quatre per comptar els fills de cadascun dels seus fills (0 si el mòdul no té fills en la direcció pertanyent).
 - Envia un missatge de retorn al seu pare amb la posició relativa del seu candidat, i el seu nombre de fills.

- (4) Els missatges de retorn són acceptats pel mòdul pare, però aquest no segueix la cadena fins que no arriben els missatges de tots els seus fills. Quan un pare rep els missatges, emmagatzema les constants que pertanyen al nombre de fills i segueix la cadena, enviant la posició del millor candidat a líder, i el nombre de fills corresponents.

Aquest missatge de retorn, finalment arriba al millor candidat que canvia d'estat a [RootS] (o [RootF] en l'arbre objectiu) i envia una cadena de missatges per començar la compressió (l'arbre objectiu atura el procés).

6.2. Compressió

Aquesta fase és anàloga a l'explicada en l'algorisme anterior. Només hem afegit a les regles que quan un mòdul amfitrió rep un mòdul convidat en compressió, també actualitza les constants que donen el nombre de fills, ja que el mòdul ha perdut un fill.

6.3. Expansió

La fase d'Expansió és la que ha tingut més modificacions. Hem eliminat el mòdul líder, ja que ara tots els mòduls actuen de forma independent amb els comptadors de fills. Quan el primer mòdul en compressió arriba a l'arrel, comença la fase d'expansió. L'objectiu del procés es pot descriure així: Si un mòdul en expansió o l'arrel, rep un mòdul convidat, llegeix les constants que indiquen el nombre de fills, i intenta enviar el mòdul en la direcció on faltin més fills per enviar fins a completar l'arbre.

Aquest enviament però, ha de ser de usant diferents normes segons el que troba al mòdul en la direcció a expandir. Veiem els casos diferents que es pot trobar:

- Si a la direcció objectiu tenim un mòdul en estat [Expnd], el mòdul amfitrió pregunta si pot enviar un mòdul convidat i si rep resposta afirmativa de l'altre mòdul, envia el convidat i actualitza les constants necessàries. No modifiquem l'estructura de l'arbre, però així enviem els mòduls a les seves posicions per l'interior de l'arbre d'expansió.
- Si ha d'enviar el mòdul a una posició buida, el mòdul amfitrió fa UNZIP, ja que no pot tenir cap problema extern. Tot seguit, actualitza les constants i envia un missatge al seu nou fill perquè canviï a estat [Expnd] i agafi les dades necessàries de l'arbre objectiu.

Aquests dos casos són els únics que ens podem trobar si l'arbre inicial i l'objectiu són disjunts. Tot i així en els algorismes de reconfiguració, trobem les dificultats quan no s'acompleix aquesta propietat. Com hem afegit les constants per comptar, si al fer l'expansió usem els mòduls de la compressió que ens bloquegen, provoquem que els mòduls perdin el compte dels que s'han d'enviar a cada branca de l'arbre. És impossible evitar aquest fet en alguns casos, ja que hem de poder expandir d'alguna

manera quan trobem un bloqueig però, tot i així volem intentar perdre el compte el mínim possible.

Si usem la idea aplicada en l'algorisme anterior de crear un cicle amb l'arbre de compressió i tallar al mateix instant, en moltes situacions obtenim que un gran nombre de mòduls de l'arbre de compressió s'afegeixen a l'arbre d'expansió. Tots aquests mòduls no han estat comptats pels mòduls de les generacions anteriors d'aquesta branca i provoquem que els comptadors no compleixin la seva utilitat.

És cert que podem intentar arreglar aquest problema enviant un missatge de tornada cap a l'arrel amb el compte dels mòduls de compressió que hem afegit pel camí, però aquests missatges no són instantanis i, per tant, en molts casos no arreglem el problema¹. Veiem doncs, com actuem en cada situació:

- Si un mòdul [Expnd] vol enviar el seu convidat a una posició ocupada per un mòdul en compressió, envia un missatge **WARD1** a aquesta direcció. Quan un mòdul en compressió rep aquest missatge, acaba totes les accions que estava executant en aquell moment i canvia d'estat a [Ward1], tot just després es connecta amb el mòdul del que ha rebut el missatge i obté les dades de l'arbre objectiu, però conservant també les dades de l'arbre de compressió, agafa la funció de mòdul intersecció entre els dos arbres. Com es queda anclat a l'arbre final, en una constant X guarda el nombre de mòduls que té en la seva posició al moment de convertir-se en [Ward1].
- Un mòdul [Ward1] té sempre preferència pel procés de compressió. Per tant, aquest mòdul pregunta sempre al seu pare de compressió, si pot rebre un mòdul comprimit:
 - Si rep resposta afirmativa i el seu fill de compressió li pot enviar un mòdul comprimit, l'envia cap allà. També accepta tants mòduls de l'arbre d'expansió com indica el comptador X . Amb aquest procés aconseguim controlar tots els mòduls usats per crear el guardià i no perdem el compte. Quan no té més mòduls fills de compressió i el valor de la constant es 0, desconnecta el seu pare de compressió i canvia a estat [Expnd].
 - Si el seu pare de compressió no pot acceptar mòduls, llavors deixa pas als mòduls d'expansió i continua amb el procés d'expansió. Pot trobar-se amb dos casos:
 - * Si el [Ward1] ha d'expandir-se en una direcció diferent de les ocupades per l'arbre de compressió, aplica aquesta acció com si fós un mòdul [Expnd] (A la Figura 1 podem veure com actua un [Ward1]).
 - * Si el mòdul ha d'anar a una posició ocupada per un mòdul [Cmprs], vol dir que l'expansió i la compressió poden crear més conflictes. El mòdul canvia a l'estat [Ward2] i envia un missatge en la direcció a ocupar amb el missatge **WARD2** per canviar el seu estat. També envia el valor de la constant X perquè afegeixi a la constant els mòduls que conté i li retorni.
- Un mòdul amb l'estat [Ward2] ha d'estar connectat a un o més mòduls amb el mateix estat². Si no és el cas, un dels mòduls del grup [Ward2] és el pare

¹La idea dels missatges l'usem també al nostre algorisme però només en un cas, reduint l'error.

²Si no és així, canvia el seu estat a [Ward1].

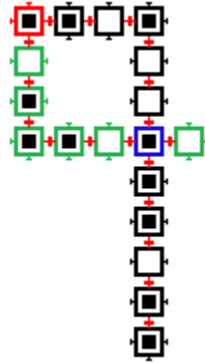


FIGURA 1. El mòdul blau està actuant de **Ward1**, controla als mòduls d'expansió (verds) perquè no col·lapsin amb els mòduls de compressió (negre).

de tots i és l'únic del grup que té una connexió amb un mòdul de compressió pare. Aquest mòdul, actua de forma semblant a com ho fa el **[Ward1]**, però sempre que necessita enviar un mòdul de compressió, el demana als seus fills **[Ward2]**, que fan una cadena de missatges fins a trobar el mòdul per enviar. Si ha d'enviar un mòdul d'expansió, l'envia a través dels seus fills de compressió, i espera a rebre resposta. Com aquest procés és força lent perquè en una cadena de **[Ward2]** han d'enviar molts missatges d'anada i tornada per comunicar-se, una altra opció és fer que el mòdul **[Ward2]** opti per donar preferència a l'expansió sempre que arribin mòduls del seu pare d'expansió, així evitem un embós a l'arbre. A la Figura 2 podem veure un esquema d'un **Ward2**.

- L'estat **[RExpnd]** apareix quan l'arrel **[RootS]** o un mòdul amb estat **[RExpnd]**³ ha d'expandir cap a una direcció ocupada per un mòdul de compressió. Aquest estat indica que el mòdul fa les accions d'expansió, però que ha d'actuar com ho fa una arrel en el procés. Aquest canvi és degut a que aquests mòduls tenen ocupat el camí que usa l'arbre de compressió per tornar els mòduls cap a l'arrel i per tant no podem tornar els mòduls de compressió usats. Si en la nostre reconfiguració hem d'usar aquesta tècnica podem tenir problemes amb els comptadors, però podem millorar la situació usant els missatges de tornada amb el compte de mòduls usats que hem explicat anteriorment.

Com en algun cas, podem perdre el compte d'alguns mòduls, hem afegit una regla de seguretat: Si un mòdul rep un mòdul comprimit i no té cap fill que encara necessiti mòduls (o no té cap fill si és una fulla), tornar aquest convidat al mòdul pare. Amb aquesta regla, si tenim algun mòdul estraviat per l'arbre, tornarà en direcció a l'arrel fins que no trobi el camí correcte a seguir.

Aquest algorisme no ha estat programat completament, però en el procés hem trobat diferents problemes amb els estats de **[Ward1]** i **[Ward2]**, doncs arribàvem a diferents casos d'embussament dels mòduls. Tot i així, hem aconseguit arribar a una forma final força estable.

³Estat que és creat en aquesta mateixa situació.

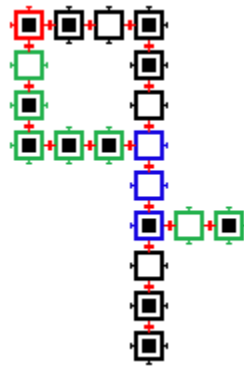


FIGURA 2. Els mòduls blaus tenen estat `Ward2` i han creat un pont format per la intersecció entre l'arbre de compressió (negre) i el d'expansió (verd). En aquest instant un mòdul comprimit està travessant aquest pont.

Capítol 7

Conclusions

7.1. Resultats obtinguts

L'objectiu principal del treball era aconseguir implementar l'algorisme de reconfiguració en el simulador. Per fer-ho, primer necessitàvem adaptar el nostre programa perquè acceptés els moviments dels robots cristal·lins.

Aquests objectius han estat complerts de forma satisfactòria. El programa permet usar tots els moviments explicats en el treball sense cap problema i l'algorisme que hem implementat ha superat tots els casos de prova a la perfecció.

7.2. Dificultats trobades

La dificultat més gran que he trobat en aquest treball, ha sigut la preparació del programa. Possiblement per un informàtic no hagués sigut tant problema però, tot i que m'agrada molt programar, mai havia tingut l'ocasió de treballar amb un programa de tal magnitud. A més, en Java, llenguatge de programació amb el que no havia treballat mai i, tot i que els codis de programació són semblants, ha dificultat encara més la comprensió del programa.

Un cop fets els canvis suficients per aconseguir que el programa pogués simular els nostres robots, vaig necessitar un temps de comprensió del simulador, ja que el codi intern del programa per implementar regles no és senzill. Tampoc ho ha sigut aprendre a pensar en totes les possibles situacions que es pot trobar un mòdul i com pot afrontar-les només utilitzant la seva pròpia informació (i la dels seus veïns directes). Tot i així, el simulador ha sigut una eina essencial en la implementació de les regles, ja que ens ha permès veure errors en les nostres regles que no et planteges que es puguin donar fins que no te les trobes.

A més, s'han d'implementar regles diferents per cada direcció del mòdul tot i que conceptualment fan el mateix. Al implementar-les usava casos senzills de prova en una direcció, i després construïa la mateixa regla per cada direcció però sempre havies de revisar les 4 direccions per evitar errors de escriptura. Finalment, després de moltes regles escrites, vaig agafar l'hàbit de seguir un ordre determinat alhora

d'implementar les direccions de cada regla. Així, sempre escrivia la regla pensant en el Sud, i feia el símil amb les altres direccions. Al agafar l'hàbit vaig aconseguir escriure-les molt ràpidament.

El temps per treballar en el projecte també ha sigut essencial, inicialment volíem implementar els dos algorismes però ha sigut impossible. Encara que porto pensant durant tot el curs en el nou algorisme, a la hora d'escriure les regles mai funcionen a la primera i el nivell de complexitat d'aquest segon algorisme era molt més alt que el programat inicialment, per tant un petit canvi, provocava errors en altres situacions que ni ens havíem plantejat. Finalment, quan començàvem a veure la llum en l'algorisme, ens vam adonar que no teníem tant temps com creiem, i vam decidir deixar de banda l'algorisme, per dedicar tot el temps possible a la escritura d'aquesta memòria.

7.3. Futur del projecte

Aquest treball tan sols és una petita part d'un projecte de creació dels robots cristal·lins, però tot i així crec que el programa pot ser útil per fer proves i crear nous algorismes per al robot. Altres tasques que ens havíem proposat en el nostre treball i que no s'han pogut complir per falta de temps són les següents:

- Implementació d'un segon algorisme de reconfiguració millorat.
- Modificar el simulador per aconseguir mostrar la vista atòmica dels moviments del robot en el simulador.
- Extensió a 3D. El simulador només pot representar figures planes, però en un futur es podria millorar i estendre les regles a la tercera dimensió.
- Implementació d'un algorisme de translació. Havíem comentat inicialment entre les idees per aquest treball, en construir un algorisme per als robots cristal·lins que servís per traslladar el robot.

7.4. Valoració Personal

Els últims dies de treball mentre feia la memòria, no estava satisfet perquè m'hagués agradat haver completat el segon algorisme, ja que hem treballat moltíssim en aquest algorisme i finalment només ha pogut ser un petit capítol del treball. Tot i així, al acabar el treball penso en com veia aquest projecte a l'inici, en com m'explicava coses la Vera dels robots i, tot i que m'atreia el que m'explicava, no entenia gaire el funcionament dels robots. Penso en el primer dia que em vaig enfrontar al codi Java del programa i va faltar poc perquè em vingués una depressió. Com després d'unes quantes setmanes, la Vera em va fer el comentari: "hauriem de començar a escriure l'algorisme"; i jo vaig pensar en què encara estava intentant entendre la primera funció del simulador.

I crec que, amb el poc temps que he tingut, he après moltíssim d'aquest treball. He après a programar en un altre llenguatge, encara que només sigui una base. La forma d'escriure les regles ha sigut com un joc mental, que m'ha fet treballar molt el cap per pensar i puc assegurar que he gaudit, com a matemàtic que sóc. Finalment,

el fet d'haver d'escriure un treball d'aquesta magnitud, que no he gaudit, però m'ha servit d'experiència. És difícil mostrar al públic tota la feina feta, explicar el treball de manera que una persona que no coneix en profunditat el tema el pogués entendre. No tinc del tot clar haver-ho aconseguit, però estic content de com ha quedat finalment.

Possiblement no he pogut aportar molt en aquest projecte, però espero que la feina feta serveixi d'utilitat en aquest gran projecte d'investigació, de la mateixa manera que m'ha ajudat a mi a la hora de fer aquest treball la feina feta per altres persones implicades en el projecte. Reconec que no hagués sigut capaç de fer cap algorisme sense l'ajuda d'aquest programa, i jo sol no hagués pogut crear un programa que fes res semblant. És difícil aportar coses noves, i per aquesta part, estic content ja que, crec que tot i que no està acabat, les meves idees sobre el nou algorisme poden ajudar en un futur a alguna persona que vulgui continuar aquest estudi de la reconfiguració.

Vull agrair a l'Àngel Rodríguez, per haver-me ajudat en la programació del simulador, sense la seva ajuda hagués tingut greus problemes per entendre certes parts del codi del programa. Finalment agrair també a la Vera Sacristán, per la seva paciència al veure que no avançava, i guiar-me buscant solucions als meus bloquejos, sobretot en aquests últims dies per escriure la memòria, crec que sense la seva ajuda no hagués sigut capaç de mostrar el treball fet en aquest document.

Gràcies a tots dos.

Referències

- [1] R. Wallner. *A System of Autonomously Self-Reconfigurable Agents*. Diploma Thesis, Institute for Software Technology, Graz University of Technology, 2009.
- [2] O. Aichholzer, T. Hackl, V. Sacristán, B. Vogtenhuber, and R. Wallner. Simulating distributed algorithms for lattice agents. To appear in *Proc. XV Spanish Meeting on Computational Geometry*, June 26-28, 2013.
- [3] G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O' Rourke, V. Pinciu, S. Ramaswami, V. Sacristán, S. Wuhler. Efficient Constant-Velocity Reconfiguration of Crystalline Robots, *Robotica*, Vol. 29, N. 1, pp. 59-71, 2011.
- [4] *Crystal Simulation*. Disponible a la Web: <http://www-ma2.upc.edu/vera/CrystalSimulation>
- [5] The Eclipse Foundation, Eclipse IDE for Java Developers, <http://www.eclipse.org>.

Annex A

Regles de la cerca de l'arrel i creació de l'arbre

En aquest apartat es detallen les regles utilitzades en la fase de la cerca de l'arrel i creació de l'arbre. L'objectiu i l'estratègia general d'aquest conjunt de regles es descriuen a la Secció 4.1 d'aquest treball.

Detallarem les regles per als mòduls de l'arbre inicial. Aquesta fase té una versió gairebé anàloga per els mòduls de l'arbre final que incloem a la segona part d'aquest annex amb menys detall.

Totes les regles comencen per [x], on x indica la secció a la que pertany la regla en qüestió.

A.1. Arbre inicial [S]

A.1.1. Inici dels candidats

Inicialment tots els mòduls de l'arbre inicial comencen amb l'estat [Start]. Aquestes dues regles comencen l'algorisme, buscant quins són els candidats a arrel de l'arbre. Per fer-ho, busquem quins són els mòduls que no tenen cap mòdul al nord ni a l'oest, aquests canvien d'estat a [CanbS], i inicien la cadena de missatges enviant la seva posició relativa als seus veïns.

```
[S]Start Message as a Candidate to E
100
A001* SStart
SCanbS C000+50500000 #E01+00005150
```

```
[S]Start Message as a Candidate to S
100
A00*1 SStart
SCanbS C000+50500000 #S01+00005051
```

A la constant *C000* guardem la posició relativa al mòdul candidat i enviem als mòduls veïns la seva posició usant el canal de missatges $\# * 01$. Aquesta posició inicial dels candidats serà el valor 5050, on la posició horitzontal és indicada amb les centenes, i la posició vertical amb unitats. Així, la posició de l'est respecte a un mòdul, té 100 unitats més i la posició al sud, té 1 unitat més. Al escriure les posicions com a valors numèrics, podem comparar posicions relatives buscant sempre la que tingui un valor més gran ja que és la posició de les de més a l'esquerra, la de més amunt.

A.1.2. Cadena de missatges de candidats

Un cop enviats els primers missatges des dels candidats, els mòduls veïns no-fula que rebin els missatges els tracten de la forma següent:

- (1) Si un mòdul rep un o més missatges dels seus veïns, escull el valor, d'entre tots els missatges i la seva posició, el més gran. Si aquest valor prové d'un missatge l'emmagatzema a *C000*. Reinicia a 0 el valor que indica la posició dels seus fills directes *C002* i assigna a *C001* la posició del mòdul que li ha enviat el missatge, que és el seu mòdul pare a l'arbre.
- (2) Si arriben dos o més missatges amb el mateix valor a un mòdul, vol dir que la nostra estructura té cicles, ja que han arribat missatges procedents del mateix mòdul candidat per dos camins diferents. En aquesta situació, el mòdul que ha rebut els missatges desconnecta tots els veïns dels quals ha rebut el missatge excloent-ne un, escollit per preferència de direccions.

En tots dos casos, després d'actualitzar les dades, el mòdul envia un missatge per continuar la cadena a tots els seus veïns menys el marcat com a pare i canvia a un estat intermig [*WaitS*] per no provocar interferències amb els missatges de tornada que explicarem més endavant.

Cal fer remarca en el fet que si un candidat rep un missatge d'un altre candidat, vol dir que l'altre candidat és millor, i per tant aquest mòdul deixa de ser-ho i canvia també a l'estat [*WaitS*], com farien la resta de mòduls normalment.

1. *Tractament d'un missatge que provenen del Nord:*

```
[S]Forward from N->W
200
A11** >#N01C000 !>#W01#N01 !>#E01#N01 !>#S01#N01 !=#W01#N01
      !=#E01#N01 !=#S01#N01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #W01-#N010100

[S]Forward from N->E
200
A1*1* >#N01C000 !>#W01#N01 !>#E01#N01 !>#S01#N01 !=#W01#N01
      !=#E01#N01 !=#S01#N01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #E01+#N010100
```

```
[S]Forward from N->S
200
A1**1 >#N01C000 !>#W01#N01 !>#E01#N01 !>#S01#N01 !=#W01#N01
      !=#E01#N01 !=#S01#N01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #S01+#N010001
```

2. *Tractament d'un missatge que provenen del Oest:*

```
[S]Forward from W->N
200
A11** >#W01C000 !>#N01#W01 !>#E01#W01 !>#S01#W01 !=#N01#W01
      !=#E01#W01 !=#S01#W01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#W010000 C001+01000000 C002+00000000 #N01-#W010001
```

```
[S]Forward from W->E
200
A*11* >#W01C000 !>#N01#W01 !>#E01#W01 !>#S01#W01 !=#N01#W01
      !=#E01#W01 !=#S01#W01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#W010000 C001+01000000 C002+00000000 #E01+#W010100
```

```
[S]Forward from W->S
200
A*1*1 >#W01C000 !>#N01#W01 !>#E01#W01 !>#S01#W01 !=#N01#W01
      !=#E01#W01 !=#S01#W01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#W010000 C001+01000000 C002+00000000 #S01+#W010001
```

3. *Tractament d'un missatge que provenen del est:*

```
[S]Forward from E->N
200
A1*1* >#E01C000 !>#N01#E01 !>#W01#E01 !>#S01#E01 !=#N01#E01
      !=#W01#E01 !=#S01#E01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#E010000 C001+00100000 C002+00000000 #N01-#E010001
```

```
[S]Forward from E->W
200
A*11* >#E01C000 !>#N01#E01 !>#W01#E01 !>#S01#E01 !=#N01#E01
      !=#W01#E01 !=#S01#E01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#E010000 C001+00100000 C002+00000000 #W01-#E010100
```

```
[S]Forward from E->S
200
A**11 >#E01C000 !>#N01#E01 !>#W01#E01 !>#S01#E01 !=#N01#E01
      !=#W01#E01 !=#S01#E01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#E010000 C001+00100000 C002+00000000 #S01+#E010001
```

4. *Tractament d'un missatge que provenen del Sud:*

```
[S]Forward from S->N
200
A1*1 >#S01C000 !>#N01#S01 !>#W01#S01 !>#E01#S01 !=#N01#S01
      !=#W01#S01 !=#E01#S01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#S010000 C001+00010000 C002+00000000 #N01-#S010001
```

```
[S]Forward from S->W
200
A*1*1 >#S01C000 !>#N01#S01 !>#W01#S01 !>#E01#S01 !=#N01#S01
      !=#W01#S01 !=#E01#S01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#S010000 C001+00010000 C002+00000000 #W01-#S010100
```

```
[S]Forward from S->E
200
A**11 >#S01C000 !>#N01#S01 !>#W01#S01 !>#E01#S01 !=#N01#S01
      !=#W01#S01 !=#E01#S01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#S010000 C001+00010000 C002+00000000 #E01+#S010100
```

5. *Cicle simple que prové del Nord, Oest:*

```
[S]Forwrd from N,W->E, detach W
200
A111* >#N01C000 =#W01#N01 !>#E01#N01 !>#S01#N01 !=#E01#N01
      !=#S01#N01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #E01+#N010100
      A10** MWDetac
```

```
[S]Forward from N,W->S, detach W
200
A11*1 >#N01C000 =#W01#N01 !>#E01#N01 !>#S01#N01 !=#E01#N01
      !=#S01#N01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #S01+#N010001
      A10** MWDetac
```

6. *Cicle simple que prové del Nord, Est:*

```
[S]Forward from N,E->W, detach E
200
A111* >#N01C000 =#E01#N01 !>#W01#N01 !>#S01#N01 !=#W01#N01
      !=#S01#N01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #W01-#N010100
      A1*0* MEDetac
```

```
[S]Forward from N,E->S, detach E
200
A1*11 >#N01C000 =#E01#N01 !>#W01#N01 !>#S01#N01 !=#W01#N01
      !=#S01#N01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #S01+#N010001
      A1*0* MEDetac
```

7. *Cicle simple que prové del Nord, Sud:*

```
[S]Forward from N,S->W, detach S
200
A11*1 >#N01C000 =#S01#N01 !>#W01#N01 !>#E01#N01 !=#W01#N01
      !=#E01#N01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #W01-#N010100
      A1**0 MSDetac
```

```
[S]Forward from N,S->E, detach S
200
A1*11 !A1001 >#N01C000 =#S01#N01 !>#W01#N01 !>#E01#N01 !=#W01#N01
      !=#E01#N01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #E01+#N010100
      A1**0 MSDetac
```

8. *Cicle simple que prové del Oest, Est:*

```
[S]Forward from W,E->N, detach E
200
A111* >#W01C000 =#E01#W01 !>#N01#W01 !>#S01#W01 !=#N01#W01
      !=#S01#W01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#W010000 C001+01000000 C002+00000000 #N01-#W010001
      A*10* MEDetac
```

```
[S]Forward from W,E->S, detach E
200
A*111 >#W01C000 =#E01#W01 !>#N01#W01 !>#S01#W01 !=#N01#W01
      !=#S01#W01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#W010000 C001+01000000 C002+00000000 #S01+#W010001
      A*10* MEDetac
```

9. *Cicle simple que prové del Oest, Sud:*

```
[S]Forward from W,S->N, detach S
200
A11*1 >#W01C000 =#S01#W01 !>#N01#W01 !>#E01#W01 !=#N01#W01
      !=#E01#W01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#W010000 C001+01000000 C002+00000000 #N01-#W010001
      A*1*0 MSDetac
```

```
[S]Forward from W,S->E, detach S
200
A*111 >#W01C000 =#S01#W01 !>#N01#W01 !>#E01#W01 !=#N01#W01
      !=#E01#W01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#W010000 C001+01000000 C002+00000000 #E01+#W010100
      A*1*0 MSDetac
```

10. *Cicle simple que prové del Est, Sud:*

```
[S]Forward from E,S->N, detach S
200
A1*11 >#E01C000 =#S01#E01 !>#N01#E01 !>#W01#E01 !=#N01#E01
      !=#W01#E01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#E010000 C001+00100000 C002+00000000 #N01-#E010001
      A**10 MSDetac
```

```
[S]Forward from E,S->W, detach S
200
A*111 >#E01C000 =#S01#E01 !>#N01#E01 !>#W01#E01 !=#N01#E01
      !=#W01#E01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#E010000 C001+00100000 C002+00000000 #W01-#E010100
      A**10 MSDetac
```

11. *Cicles dobles:*

```
[S]Forward from N,W,E, detach W,E
200
A1111 >#N01C000 =#W01#N01 =#E01#N01 !>#S01#N01 !=#S01#N01
      !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #S01+#N010001
      A100* MWDetac MEDetac
```

```
[S]Forward from N,W,S, detach W,S
200
A1111 >#N01C000 =#W01#N01 =#S01#N01 !>#E01#N01 !=#E01#N01
      !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #E01+#N010100
      A10*0 MWDetac MSDetac
```

```
[S]Forward from N,E,S, detach E,S
200
A1111 >#N01C000 =#E01#N01 =#S01#N01 !>#W01#N01 !=#W01#N01
      !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#N010000 C001+10000000 C002+00000000 #W01-#N010100
      A1*00 MEDetac MSDetac
```



```
[S]Forward from W,E,S, detach E,S
200
A1111 >#W01C000 =#E01#W01 =#S01#W01 !>#N01#W01 !=#N01#W01
      !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SWaitS C000+#W010000 C001+01000000 C002+00000000 #N01-#W010001
      A*100 MEDetac MSDetac
```

12. *Canvi d'estat a Forwd:*

```
[S]Change from Wait_ to Forwd
200
SWaitS
SForwS
```

Després d'haver fet un enviament de missatges de la posició dels candidats, l'estat dels mòduls canvia a [WaitS] en aquest estat rep missatges de posició, i de desconnexió però ignora els missatges de tornada que explicarem més endavant. Amb aquest estat, arreglem un problema de superposició de missatges, ja que els mòduls només tenen espai per guardar un missatge String de cada direcció.

A.1.3. Missatges rebuts a les fulles

Els mòduls segueixen la cadena de missatges fins que aquest arriba a una fulla que, com no té cap fill al que enviar el missatge, finalitza la cadena. Quan una fulla rep un missatge, comença una cadena de missatges cap a l'arrel, per comprovar que tot l'arbre sigui correcte. Per indicar que un mòdul està en procés de tornada a l'arrel, usem l'estat [BackS], les fulles canvien a aquest estat i envien un missatge al seu pare indicant que ha arribat a una fulla.

```
[S]Leaf from N
200
A1000 >#N01C000 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#N010000 C001+10000000 MNBack_
```

```
[S]Leaf from W
200
A0100 >#W01C000 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#W010000 C001+01000000 MWBack_
```

```
[S]Leaf from E
200
A0010 >#E01C000 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#E010000 C001+00100000 MEBack_
```

```
[S]Leaf from S
200
A0001 >#S01C000 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#S010000 C001+00010000 MSBack_
```

Després d'haver fet un enviament de missatges de la posició dels candidats, l'estat dels mòduls canvia a [WaitS] en aquest estat rep missatges de posició, i de desconnexió però ignora els missatges de tornada que explicarem més endavant. Amb aquest estat, arreglem un problema de superposició de missatges, ja que els mòduls només tenen espai per guardar un missatge String de cada direcció.

A.1.4. Missatges rebuts a les fulles

Els mòduls segueixen la cadena de missatges fins que aquest arriba a una fulla que, com no té cap fill al que enviar el missatge, finalitza la cadena. Quan una fulla rep un missatge, comença una cadena de missatges cap a l'arrel, per comprovar que tot l'arbre sigui correcte. Per indicar que un mòdul està en procés de tornada a l'arrel, usem l'estat [BackS], les fulles canvien a aquest estat i envien un missatge al seu pare indicant que ha arribat a una fulla.

```
[S]Pseudo Leaf N,W, detach W
200
A1100 >#N01C000 =#N01#W01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#N010000 C001+10000000 MNBack_ A1000 MWDetac
```

```
[S]Pseudo Leaf N,E, detach E
200
A1010 >#N01C000 =#N01#E01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#N010000 C001+10000000 MNBack_ A1000 MEDetac
```

```
[S]Pseudo Leaf N,S, detach S
200
A1001 >#N01C000 =#N01#S01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#N010000 C001+10000000 MNBack_ A1000 MSDetac
```

```
[S]Pseudo Leaf W,E, detach E
200
A0110 >#W01C000 =#W01#E01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#W010000 C001+01000000 MWBack_ A0100 MEDetac
```

```
[S]Pseudo Leaf W,S, detach S
200
A0101 >#W01C000 =#W01#S01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#W010000 C001+01000000 MWBack_ A0100 MSDetac
```

```
[S]Pseudo Leaf E,S, detach S
200
A0011 >#E01C000 =#E01#S01 !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#E010000 C001+00100000 MEBBack_ A0010 MSDetac
```

```
[S]Pseudo Leaf N,W,E, detach W,E
200
A1110 >#N01C000 =#N01#W01 =#N01#E01
```

```
      !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#N010000 C001+10000000 MNBack_ A1000 MWDetac MEDetac

[S]Pseudo Leaf N,W,S, detach W,S
200
A1101 >#N01C000 =#N01#W01 =#N01#S01
      !(!SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#N010000 C001+10000000 MNBack_ A1000 MWDetac MSDetac
```

```

[S]Pseudo Leaf N,E,S, detach E,S
200
A1011 >#N01C000 =#N01#E01 =#N01#S01
      !(SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#N010000 C001+10000000 MNBack_ A1000 MEDetac MSDetac

[S]Pseudo Leaf W,E,S, detach E,S
200
A0111 >#W01C000 =#W01#E01 =#W01#S01
      !(SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#W010000 C001+01000000 MWBack_ A0100 MEDetac MSDetac

[S]Pseudo Leaf N,W,E,S, detach W,E,S
200
A1111 >#N01C000 =#N01#W01 =#N01#E01 =#N01#S01
      !(SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS C000+#W010000 C001+10000000 MNBack_ A1000
      MWDetac MEDetac MSDetac

```

Semblant al cas anterior, si un mòdul és desconnectat del seu veí perquè aquest segon ha reconegut un cicle, pot trobar-se en el cas de passar a ser una fulla. Aquest mòdul doncs, ha de fer les mateixes accions que una fulla, i enviar un missatge al seu pare indicant que ha arribat a una fulla.

```

[S]Leaf from N if detach message received
200
A1000 !(MWDetac !MEDetac !MSDetac) =C0011000
      !(SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS MNBack_

[S]Leaf from W if detach message received
200
A0100 !(MNDetac !MEDetac !MSDetac) =C0010100
      !(SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS MWBack_

[S]Leaf from E if detach message received
200
A0010 !(MNDetac !MWDetac !MSDetac) =C0010010
      !(SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS MEBBack_

[S]Leaf from S if detach message received
200
A0001 !(MNDetac !MWDetac !MEDetac) =C0010001
      !(SCanbS !SStart !SForwS !SBackS !SWaitS)
SBackS MSBack_

```

Aquests missatges de **Back** són els que provocaven conflictes amb els de desconnexió al ser els dos missatges string.

A.2. Cadena de missatges de les fulles

Quan un mòdul rep el missatge que prové d'una fulla, vol dir que la cadena de missatges de l'arrel a arribat al final d'un dels camins que ell pertany. Si tots els camins dels seus fills arriben a una fulla, vol dir que la seva branca ja està construïda. El mòdul llavors ha d'enviar el missatge al seu pare amb la informació de que tots els seus camins han acabat per continuar la nova cadena amb l'objectiu d'arribar al seu candidat arrel.

1. *Actualització de fills:* Per fer-ho, el mòdul afegeix a la constant *C002* la direcció de la que prové el missatge de fulla, ja que només pot rebre aquests missatges dels seus fills. Així al final del procés, quan ha rebut tots els missatges fulla, té a la constant *C002* la direcció de tots els seus fills.

```
[S]Store back message from N
100
SForwS MNBack_ !=C0021000 !=C0021100 !=C0021010 !=C0021001
           !=C0021110 !=C0021101 !=C0021011 !=C0021111
C002+C0021000
```

```
[S]Store back message from W
100
SForwS MWBack_ !=C0020100 !=C0021100 !=C0020110 !=C0020101
           !=C0021110 !=C0021101 !=C0020111 !=C0021111
C002+C0020100
```

```
[S]Store back message from E
100
SForwS MEBack_ !=C0020010 !=C0021010 !=C0020110 !=C0020011
           !=C0021110 !=C0021011 !=C0020111 !=C0021111
C002+C0020010
```

```
[S]Store back message from S
100
SForwS MSBack_ !=C0020001 !=C0021001 !=C0020101 !=C0020011
           !=C0021101 !=C0021011 !=C0020111 !=C0021111
C002+C0020001
```

Hem afegit a les funcions la precondició que no tingui ja informació d'aquest fill, per evitar contar el fill més d'una vegada.

2. *Enviament del missatge de tornada:* Un cop que ha rebut el missatge de **Back** de tots els seus fills, que ho pot comprovar si tots els veïns que té connectat són pares o fills, envia un missatge de **Back** al seu pare, per continuar la cadena.

[S]Relay Back message N->W
100
A1100 =C0010100 =C0021000 SForwS
MWBack_ SBackS

[S]Relay Back message N->E
100
A1010 =C0010010 =C0021000 SForwS
MEBack_ SBackS

[S]Relay Back message N->S
100
A1001 =C0010001 =C0021000 SForwS
MSBack_ SBackS

[S]Relay Back message W->N
100
A1100 =C0011000 =C0020100 SForwS
MNBack_ SBackS

[S]Relay Back message W->E
100
A0110 =C0010010 =C0020100 SForwS
MEBack_ SBackS

[S]Relay Back message W->S
100
A0101 =C0010001 =C0020100 SForwS
MSBack_ SBackS

[S]Relay Back message E->N
100
A1010 =C0011000 =C0020010 SForwS
MNBack_ SBackS

[S]Relay Back message E->W
100
A0110 =C0010100 =C0020010 SForwS
MWBack_ SBackS

[S]Relay Back message E->S
100
A0011 =C0010001 =C0020010 SForwS
MSBack_ SBackS

[S]Relay Back message S->N
100
A1001 =C0011000 =C0020001 SForwS
MNBack_ SBackS

[S]Relay Back message S->W
100
A0101 =C0010100 =C0020001 SForwS
MWBack_ SBackS

[S]Relay Back message S->E
100
A0011 =C0010010 =C0020001 SForwS
MEBack_ SBackS

[S]Relay Back message N,W->E
100
A1110 =C0010010 =C0021100 SForwS
MEBack_ SBackS

[S]Relay Back message N,W->S
100
A1101 =C0010001 =C0021100 SForwS
MSBack_ SBackS

[S]Relay Back message N,E->W
100
A1110 =C0010100 =C0021010 SForwS
MWBack_ SBackS

[S]Relay Back message N,E->S
100
A1011 =C0010001 =C0021010 SForwS
MSBack_ SBackS

[S]Relay Back message N,S->W
100
A1101 =C0010100 =C0021001 SForwS
MWBack_ SBackS

[S]Relay Back message N,S->E
100
A1011 =C0010010 =C0021001 SForwS
MEBack_ SBackS

[S]Relay Back message W,E->N
100
A1110 =C0011000 =C0020110 SForwS
MNBack_ SBackS

[S]Relay Back message W,E->S
100
A0111 =C0010001 =C0020110 SForwS
MSBack_ SBackS

[S]Relay Back message W,S->N
100
A1101 =C0011000 =C0020101 SForwS
MNBack_ SBackS

[S]Relay Back message W,S->E
100
A0111 =C0010010 =C0020101 SForwS
MEBack_ SBackS

[S]Relay Back message E,S->N
100
A1011 =C0011000 =C0020011 SForwS
MNBack_ SBackS

[S]Relay Back message E,S->W
100
A0111 =C0010100 =C0020011 SForwS
MWBack_ SBackS

[S]Relay Back message N,W,E->S
100
A1111 =C0010001 =C0021110 SForwS
MSBack_ SBackS

[S]Relay Back message N,W,S->E
100
A1111 =C0010010 =C0021101 SForwS
MEBack_ SBackS

[S]Relay Back message N,E,S->W
100
A1111 =C0010100 =C0021011 SForwS
MWBack_ SBackS


```
[S]Relay Back message W,E,S->N
100
A1111 =C0011000 =C0020111 SForwS
MNBack_ SBackS
```

A.2.1. Creació de l'arrel

Finalment, si un mòdul candidat rep missatges de tornada de tots els seus fills, aquest mòdul és el millor candidat a arrel. Seguint aquesta idea si rep un missatge de tornada, guarda la direcció del seu fill tal com ho fan tots els altres mòduls, i després comprova si té algun altre mòdul connectat. Si no és el cas, vol dir que no pot tenir més fills, i per tant s'ha acabat la cadena de missatges de tornada. Aquest mòdul és l'arrel, canvia l'estat a [RootS] i envia una última cadena de missatges als seus fills, per començar la següent fase de compressió.

```
[S]Store BackToCandy E
100
SCanbS A0011 =C0020000 MEBack_ !MSBack_
C002+00100000
```

```
[S]Store BackToCandy S
100
SCanbS A0011 =C0020000 MSBack_ !MEBack_
C002+00010000
```

```
[S]Make Root E
100
SCanbS A0010 MEBack_
SRootS C002+00100000 MESlave C023+00000000
```

```
[S]Make Root S
100
SCanbS A0001 MSBack_
SRootS C002+00010000 MSSlave C023+00000000
```

```
[S]Make Root E,S at the same time
100
SCanbS A0011 =C0020000 MEBack_ MSBack_
SRootS C002+00110000 MESlave MSSlave C023+00000000
```

```
[S]Make Root E, S
100
SCanbS A0011 =C0020010 MSBack_
SRootS C002+C0020001 MESlave MSSlave C023+00000000
```

```
[S]Make Root S, E
100
SCanbS A0011 =C0020001 MEBack_
SRootS C002+C0020010 MESlave MSSlave C023+00000000
```

Veiem en les funcions de creació de l'arrel, que inicialitza una constant *C023* amb valor 0. Aquesta constant ens serveix per simular que l'arrel coneix la informació de l'arbre objectiu, expliquem com al final d'aquesta secció.

A.2.2. Cadena de missatges de Compresió

L'arrel ha començat la cadena de missatges per començar la fase de compressió. Quan un mòdul rep aquest missatge canvia a l'estat de compressió [*Cmprs*] i envia el missatge a tots els seus fills. Si el mòdul és una fulla, no ha d'enviar cap missatge ja que no té cap fill i s'acaba la cadena.

```
[S]Store and forward Slave to N
200
A1*** !(MWSlave !MESlave !MSSlave) SBackS
SCmprs MNSlave
```

```
[S]Store and forward Slave to W
200
A*1** !(MNSlave !MESlave !MSSlave) SBackS
SCmprs MWSlave
```

```
[S]Store and forward Slave to E
200
A**1* !(MNSlave !MWSlave !MSSlave) SBackS
SCmprs MESlave
```

```
[S]Store and forward Slave to S
200
A***1 !(MNSlave !MWSlave !MESlave) SBackS
SCmprs MSSlave
```

```
[S]Store Slave on Leaf
200
!(!(A1000 MNSlave) !(A0100 MWSlave) !(A0010 MESlave)
  !(A0001 MSSlave)) SBackS
SCmprs
```

A.2.3. Cerca de l'arbre objectiu

Per simular que el líder coneix la informació de l'arbre objectiu, un cop hem trobat l'arrel, aquesta comença a buscar l'arrel de l'arbre objectiu. Aquesta arrel objectiu hem d'imposar al simular els dos arbres que sempre estigui en direcció est de l'arrel original, l'arrel inicial doncs, buscarà la distància fins l'arrel objectiu que guardarà en la constant *C023*, indicant la posició relativa on es troba l'arbre final i així poder buscar informació en qualsevol moment.

```
[S]Search Final Tree
100
SRootS !TC23,0,CanbF !TC23,0,RootF
C023+C0230001

[S]Copy Final Tree
1000
SRootS TC23,0,RootF =C0220000
SRootL C002+00000000 C004+C23,0,C0020000 C022+C0220001
```

La primera funció, te com a condició trobar un mòdul amb l'estat [RootF] de l'arrel objectiu, o [CanbF] estat d'un candidat, per si l'arbre final encara no ha acabat la seva creació. Quan l'arrel troba la casella i l'arbre objectiu s'ha completat, l'arrel inicial canvia d'estat a [RootL] indicant que és l'arrel de l'arbre i també és el líder inicial. Usarem la constant *C004* per indicar els fills que ha de tenir el mòdul en aquella posició, obtinguts de l'arbre final. La constant *C022* fa la funció de booleà, per no crear més d'un líder.

A.3. Arbre objectiu [F]

En aquesta secció construïm l'arbre de l'estructura final del robot, el construïm perquè el robot del simulador pugui obtenir la informació. En el robot original aquest arbre no existeix, però el líder el coneix en la seva memòria.

Tot el procés és gairebé anàleg al de l'arbre inicial, però substituint els estats:

- Start → Final
- RootS → RootF
- CanbS → CanbF
- BackS → BackF
- FrwdS → FrwdF
- WaitS → WaitF
- Cmprs → Slave

Així, al final de la fase, els mòduls no comencen el procés de compressió, i és queden aturats.

[F]Start Message as a Candidate to E

100

A001* SFinal

SCanbF C000+50500000 #E01+00005150

[F]Start Message as a Candidate to S

100

A00*1 SFinal

SCanbF C000+50500000 #S01+00005051

[F]Forward from N->W

200

A11** >#N01C000 !>#W01#N01 !>#E01#N01 !>#S01#N01 !=#W01#N01

!=#E01#N01 !=#S01#N01 !(SCanbF !SFinal !SForwF !SBackF !SWaitF)

SWaitF C000+#N010000 C001+10000000 C002+00000000 #W01-#N010100

[F]Forward from N->E

200

A1*1* >#N01C000 !>#W01#N01 !>#E01#N01 !>#S01#N01 !=#W01#N01

!=#E01#N01 !=#S01#N01 !(SCanbF !SFinal !SForwF !SBackF !SWaitF)

SWaitF C000+#N010000 C001+10000000 C002+00000000 #E01+#N010100

[F]Forward from N->S

200

A1**1 >#N01C000 !>#W01#N01 !>#E01#N01 !>#S01#N01 !=#W01#N01

!=#E01#N01 !=#S01#N01 !(SCanbF !SFinal !SForwF !SBackF !SWaitF)

SWaitF C000+#N010000 C001+10000000 C002+00000000 #S01+#N010001

[F]Forward from W->N

200

A11** >#W01C000 !>#N01#W01 !>#E01#W01 !>#S01#W01 !=#N01#W01

!=#E01#W01 !=#S01#W01 !(SCanbF !SFinal !SForwF !SBackF !SWaitF)

SWaitF C000+#W010000 C001+01000000 C002+00000000 #N01-#W010001

[F]Forward from W->E

200

A*11* >#W01C000 !>#N01#W01 !>#E01#W01 !>#S01#W01 !=#N01#W01

!=#E01#W01 !=#S01#W01 !(SCanbF !SFinal !SForwF !SBackF !SWaitF)

SWaitF C000+#W010000 C001+01000000 C002+00000000 #E01+#W010100

[F]Forward from W->S

200

A*1*1 >#W01C000 !>#N01#W01 !>#E01#W01 !>#S01#W01 !=#N01#W01

!=#E01#W01 !=#S01#W01 !(SCanbF !SFinal !SForwF !SBackF !SWaitF)

SWaitF C000+#W010000 C001+01000000 C002+00000000 #S01+#W010001

[F]Forward from E->N

200

A1*1* >#E01C000 !>#N01#E01 !>#W01#E01 !>#S01#E01 !=#N01#E01
 !=#W01#E01 !=#S01#E01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#E010000 C001+00100000 C002+00000000 #N01-#E010001

[F]Forward from E->W

200

A*11* >#E01C000 !>#N01#E01 !>#W01#E01 !>#S01#E01 !=#N01#E01
 !=#W01#E01 !=#S01#E01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#E010000 C001+00100000 C002+00000000 #W01-#E010100

[F]Forward from E->S

200

A**11 >#E01C000 !>#N01#E01 !>#W01#E01 !>#S01#E01 !=#N01#E01
 !=#W01#E01 !=#S01#E01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#E010000 C001+00100000 C002+00000000 #S01+#E010001

[F]Forward from S->N

200

A1**1 >#S01C000 !>#N01#S01 !>#W01#S01 !>#E01#S01 !=#N01#S01
 !=#W01#S01 !=#E01#S01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#S010000 C001+00010000 C002+00000000 #N01-#S010001

[F]Forward from S->W

200

A*1*1 >#S01C000 !>#N01#S01 !>#W01#S01 !>#E01#S01 !=#N01#S01
 !=#W01#S01 !=#E01#S01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#S010000 C001+00010000 C002+00000000 #W01-#S010100

[F]Forward from S->E

200

A**11 >#S01C000 !>#N01#S01 !>#W01#S01 !>#E01#S01 !=#N01#S01
 !=#W01#S01 !=#E01#S01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#S010000 C001+00010000 C002+00000000 #E01+#S010100

[F]Forward from N,W->E, detach W

200

A111* >#N01C000 =#W01#N01 !>#E01#N01 !>#S01#N01 !=#E01#N01
 !=#S01#N01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#N010000 C001+10000000 C002+00000000 #E01+#N010100
 A10** MWDetac

```
[F]Forward from N,W->S, detach W
200
A11*1 >#N01C000 =#W01#N01 !>#E01#N01 !>#S01#N01 !=#E01#N01
      !=#S01#N01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#N010000 C001+10000000 C002+00000000 #S01+#N010001
A10** MWDetac
```

```
[F]Forward from N,E->W, detach E
200
A111* >#N01C000 =#E01#N01 !>#W01#N01 !>#S01#N01 !=#W01#N01
      !=#S01#N01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#N010000 C001+10000000 C002+00000000 #W01-#N010100
A1*0* MEDetac
```

```
[F]Forward from N,E->S, detach E
200
A1*11 >#N01C000 =#E01#N01 !>#W01#N01 !>#S01#N01 !=#W01#N01
      !=#S01#N01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#N010000 C001+10000000 C002+00000000 #S01+#N010001
A1*0* MEDetac
```

```
[F]Forward from N,S->W, detach S
200
A11*1 >#N01C000 =#S01#N01 !>#W01#N01 !>#E01#N01 !=#W01#N01
      !=#E01#N01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#N010000 C001+10000000 C002+00000000 #W01-#N010100
A1**0 MSDetac
```

```
[F]Forward from N,S->E, detach S
200
A1*11 !A1001 >#N01C000 =#S01#N01 !>#W01#N01 !>#E01#N01 !=#W01#N01
      !=#E01#N01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#N010000 C001+10000000 C002+00000000 #E01+#N010100
A1**0 MSDetac
```

```
[F]Forward from W,E->N, detach E
200
A111* >#W01C000 =#E01#W01 !>#N01#W01 !>#S01#W01 !=#N01#W01
      !=#S01#W01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#W010000 C001+01000000 C002+00000000 #N01-#W010001
A*10* MEDetac
```

```
[F]Forward from W,E->S, detach E
200
A*111 >#W01C000 =#E01#W01 !>#N01#W01 !>#S01#W01 !=#N01#W01
      !=#S01#W01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#W010000 C001+01000000 C002+00000000 #S01+#W010001
      A*10* MEDetac
```

```
[F]Forward from W,S->N, detach S
200
A11*1 >#W01C000 =#S01#W01 !>#N01#W01 !>#E01#W01 !=#N01#W01
      !=#E01#W01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#W010000 C001+01000000 C002+00000000 #N01-#W010001
      A*1*0 MSDetac
```

```
[F]Forward from W,S->E, detach S
200
A*111 >#W01C000 =#S01#W01 !>#N01#W01 !>#E01#W01 !=#N01#W01
      !=#E01#W01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#W010000 C001+01000000 C002+00000000 #E01+#W010100
      A*1*0 MSDetac
```

```
[F]Forward from E,S->N, detach S
200
A1*11 >#E01C000 =#S01#E01 !>#N01#E01 !>#W01#E01 !=#N01#E01
      !=#W01#E01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#E010000 C001+00100000 C002+00000000 #N01-#E010001
      A**10 MSDetac
```

```
[F]Forward from E,S->W, detach S
200
A*111 >#E01C000 =#S01#E01 !>#N01#E01 !>#W01#E01 !=#N01#E01
      !=#W01#E01 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#E010000 C001+00100000 C002+00000000 #W01-#E010100
      A**10 MSDetac
```

```
[F]Forward from N,W,E, detach W,E
200
A1111 >#N01C000 =#W01#N01 =#E01#N01 !>#S01#N01 !=#S01#N01
      !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
SWaitF C000+#N010000 C001+10000000 C002+00000000 #S01+#N010001
      A100* MWDetac MEDetac
```

[F]Forward from N,W,S, detach W,S

200

A1111 >#N01C000 =#W01#N01 =#S01#N01 !>#E01#N01 !=#E01#N01
 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#N010000 C001+10000000 C002+00000000 #E01+#N010100
 A10*0 MWDetac MSDetac

[F]Forward from N,E,S, detach E,S

200

A1111 >#N01C000 =#E01#N01 =#S01#N01 !>#W01#N01 !=#W01#N01
 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#N010000 C001+10000000 C002+00000000 #W01-#N010100
 A1*00 MEDetac MSDetac

[F]Forward from W,E,S, detach E,S

200

A1111 >#W01C000 =#E01#W01 =#S01#W01 !>#N01#W01 !=#N01#W01
 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SWaitF C000+#W010000 C001+01000000 C002+00000000 #N01-#W010001
 A*100 MEDetac MSDetac

[F]Change from Wait_ to Forwd

200

SWaitF
 SForwF

[F]Leaf from N

200

A1000 >#N01C000 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SBackF C000+#N010000 C001+10000000 MNBack_

[F]Leaf from W

200

A0100 >#W01C000 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SBackF C000+#W010000 C001+01000000 MWBack_

[F]Leaf from E

200

A0010 >#E01C000 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SBackF C000+#E010000 C001+00100000 MEBack_

[F]Leaf from S

200

A0001 >#S01C000 !(!SCanbF !SFinal !SForwF !SBackF !SWaitF)
 SBackF C000+#S010000 C001+00010000 MSBack_


```

[F]Pseudo Leaf N,W, detach W
200
A1100 >#N01C000 =#N01#W01 !(SCanBF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#N010000 C001+10000000 MNBack_ A1000 MWDetac

[F]Pseudo Leaf N,E, detach E
200
A1010 >#N01C000 =#N01#E01 !(SCanBF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#N010000 C001+10000000 MNBack_ A1000 MEDetac

[F]Pseudo Leaf N,S, detach S
200
A1001 >#N01C000 =#N01#S01 !(SCanBF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#N010000 C001+10000000 MNBack_ A1000 MSDetac

[F]Pseudo Leaf W,E, detach E
200
A0110 >#W01C000 =#W01#E01 !(SCanBF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#W010000 C001+01000000 MWBack_ A0100 MEDetac

[F]Pseudo Leaf W,S, detach S
200
A0101 >#W01C000 =#W01#S01 !(SCanBF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#W010000 C001+01000000 MWBack_ A0100 MSDetac

[F]Pseudo Leaf E,S, detach S
200
A0011 >#E01C000 =#E01#S01 !(SCanBF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#E010000 C001+00100000 MEBBack_ A0010 MSDetac

[F]Pseudo Leaf N,W,E, detach W,E
200
A1110 >#N01C000 =#N01#W01 =#N01#E01
      !(SCanBF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#N010000 C001+10000000 MNBack_ A1000 MWDetac MEDetac

[F]Pseudo Leaf N,W,S, detach W,S
200
A1101 >#N01C000 =#N01#W01 =#N01#S01
      !(SCanBF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#N010000 C001+10000000 MNBack_ A1000 MWDetac MSDetac

[F]Pseudo Leaf N,E,S, detach E,S
200
A1011 >#N01C000 =#N01#E01 =#N01#S01
      !(SCanBF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#N010000 C001+10000000 MNBack_ A1000 MEDetac MSDetac

```

```

[F]Pseudo Leaf W,E,S, detach E,S
200
A0111 >#W01C000 =#W01#E01 =#W01#S01
      !(SCanbF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#W010000 C001+01000000 MWBack_ A0100 MEDetac MSDetac

[F]Pseudo Leaf N,W,E,S, detach W,E,S
200
A1111 >#N01C000 =#N01#W01 =#N01#E01 =#N01#S01
      !(SCanbF !SFinal !SForwF !SBackF !SWaitF)
SBackF C000+#W010000 C001+10000000 MNBack_ A1000
      MWDetac MEDetac MSDetac

[F]Leaf from N if detach message received
200
A1000 !(MWDetac !MEDetac !MSDetac) =C0011000
      !(SCanbF !SFinal !SForwF !SBackF !SWaitF)
SBackF MNBack_

[F]Leaf from W if detach message received
200
A0100 !(MNDetac !MEDetac !MSDetac) =C0010100
      !(SCanbF !SFinal !SForwF !SBackF !SWaitF)
SBackF MWBack_

[F]Leaf from E if detach message received
200
A0010 !(MNDetac !MWDetac !MSDetac) =C0010010
      !(SCanbF !SFinal !SForwF !SBackF !SWaitF)
SBackF MEBBack_

[F]Leaf from S if detach message received
200
A0001 !(MNDetac !MWDetac !MEDetac) =C0010001
      !(SCanbF !SFinal !SForwF !SBackF !SWaitF)
SBackF MSBack_

[F]Store back message from N
100
SForwF MNBack_ !=C0021000 !=C0021100 !=C0021010
      !=C0021001 !=C0021110 !=C0021101 !=C0021011 !=C0021111
C002+C0021000

```

```
[F]Store back message from W
100
SForwF MWBack_ !=C0020100 !=C0021100 !=C0020110
      !=C0020101 !=C0021110 !=C0021101 !=C0020111 !=C0021111
C002+C0020100
```

```
[F]Store back message from E
100
SForwF MEBack_ !=C0020010 !=C0021010 !=C0020110
      !=C0020011 !=C0021110 !=C0021011 !=C0020111 !=C0021111
C002+C0020010
```

```
[F]Store back message from S
100
SForwF MSBack_ !=C0020001 !=C0021001 !=C0020101
      !=C0020011 !=C0021101 !=C0021011 !=C0020111 !=C0021111
C002+C0020001
```

```
[F]Relay Back message N->W
100
A1100 =C0010100 =C0021000 SForwF
MWBack_ SBackF
```

```
[F]Relay Back message N->E
100
A1010 =C0010010 =C0021000 SForwF
MEBack_ SBackF
```

```
[F]Relay Back message N->S
100
A1001 =C0010001 =C0021000 SForwF
MSBack_ SBackF
```

```
[F]Relay Back message W->N
100
A1100 =C0011000 =C0020100 SForwF
MNBack_ SBackF
```

```
[F]Relay Back message W->E
100
A0110 =C0010010 =C0020100 SForwF
MEBack_ SBackF
```

```
[F]Relay Back message W->S
100
A0101 =C0010001 =C0020100 SForwF
MSBack_ SBackF
```

[F]Relay Back message E->N
100
A1010 =C0011000 =C0020010 SForwF
MNBack_ SBackF

[F]Relay Back message E->W
100
A0110 =C0010100 =C0020010 SForwF
MWBack_ SBackF

[F]Relay Back message E->S
100
A0011 =C0010001 =C0020010 SForwF
MSBack_ SBackF

[F]Relay Back message S->N
100
A1001 =C0011000 =C0020001 SForwF
MNBack_ SBackF

[F]Relay Back message S->W
100
A0101 =C0010100 =C0020001 SForwF
MWBack_ SBackF

[F]Relay Back message S->E
100
A0011 =C0010010 =C0020001 SForwF
MEBack_ SBackF

[F]Relay Back message N,W->E
100
A1110 =C0010010 =C0021100 SForwF
MEBack_ SBackF

[F]Relay Back message N,W->S
100
A1101 =C0010001 =C0021100 SForwF
MSBack_ SBackF

[F]Relay Back message N,E->W
100
A1110 =C0010100 =C0021010 SForwF
MWBack_ SBackF

[F]Relay Back message N,E->S
100
A1011 =C0010001 =C0021010 SForwF
MSBack_ SBackF

[F]Relay Back message N,S->W
100
A1101 =C0010100 =C0021001 SForwF
MWBack_ SBackF

[F]Relay Back message N,S->E
100
A1011 =C0010010 =C0021001 SForwF
MEBack_ SBackF

[F]Relay Back message W,E->N
100
A1110 =C0011000 =C0020110 SForwF
MNBack_ SBackF

[F]Relay Back message W,E->S
100
A0111 =C0010001 =C0020110 SForwF
MSBack_ SBackF

[F]Relay Back message W,S->N
100
A1101 =C0011000 =C0020101 SForwF
MNBack_ SBackF

[F]Relay Back message W,S->E
100
A0111 =C0010010 =C0020101 SForwF
MEBack_ SBackF

[F]Relay Back message E,S->N
100
A1011 =C0011000 =C0020011 SForwF
MNBack_ SBackF

[F]Relay Back message E,S->W
100
A0111 =C0010100 =C0020011 SForwF
MWBack_ SBackF

[F]Relay Back message N,W,E->S
100
A1111 =C0010001 =C0021110 SForwF
MSBack_ SBackF

[F]Relay Back message N,W,S->E
100
A1111 =C0010010 =C0021101 SForwF
MEBack_ SBackF

[F]Relay Back message N,E,S->W
100
A1111 =C0010100 =C0021011 SForwF
MWBack_ SBackF

[F]Relay Back message W,E,S->N
100
A1111 =C0011000 =C0020111 SForwF
MNBack_ SBackF

[F]Make Root E
100
SCanbF A0010 MEBack_
SRootF C002+00100000 MESlave

[F]Make Root S
100
SCanbF A0001 MSBack_
SRootF C002+00010000 MSSlave

[F]Store BackToCandy E
100
SCanbF A0011 =C0020000 MEBack_ !MSBack_
C002+00100000

[F]Store BackToCandy S
100
SCanbF A0011 =C0020000 MSBack_ !MEBack_
C002+00010000

[F]Make Root E,S at the same time
100
SCanbF A0011 =C0020000 MEBack_ MSBack_
SRootF C002+00110000 MESlave MSSlave

```

[F]Make Root E, S
100
SCanbF A0011 =C0020010 MSBack_
SRootF C002+C0020001 MESlave MSSlave

[F]Make Root S, E
100
SCanbF A0011 =C0020001 MEBack_
SRootF C002+C0020010 MESlave MSSlave

[F]Store and forward Slave to N
200
A1*** !(MWSlave !MESlave !MSSlave) SBackF
SSlave MNSlave

[F]Store and forward Slave to W
200
A*1** !(MNSlave !MESlave !MSSlave) SBackF
SSlave MWSlave

[F]Store and forward Slave to E
200
A**1* !(MNSlave !MWSlave !MSSlave) SBackF
SSlave MESlave

[F]Store and forward Slave to S
200
A***1 !(MNSlave !MWSlave !MESlave) SBackF
SSlave MSSlave

[F]Store Slave on Leaf
200
!(A1000 MNSlave) !(A0100 MWSlave) !(A0010 MESlave)
  !(A0001 MSSlave)) SBackF
SSlave

```


Annex B

Regles de la compressió

En aquest apartat es detallen les regles utilitzades en la fase de compressió. L'objectiu i l'estratègia general d'aquest conjunt de regles es descriuen a la Secció 4.2 d'aquest treball.

Les regles d'aquesta secció comencen per [C] per indicar que és una regla de la fase de compressió.

B.1. ZIP

Després de crear l'arbre inicial, tots els mòduls canvien l'estat a [Cmprs] amb un últim enviament de missatges. Quan una fulla té aquest estat, utilitza l'acció ZIP per comprimir-se al seu pare. Podem separar aquesta compressió en tres fases:

- La fulla descomprimida envia un missatge al seu pare amb el nom ASK_Z, que usa per preguntar si es pot comprimir.
- El mòdul que rep un missatge ASK_Z decideix si es possible rebre un mòdul convidat, ja que és probable que rebi missatges amb altres accions. Si el pot rebre, envia un missatge de confirmació CAN_Z al seu mòdul fill.
- Si el mòdul fulla rep el missatge de confirmació CAN_Z utilitza l'acció ZIP per comprimir-se al seu pare.

B.1.1. ASK_Z

Els mòduls fulla en estat [Cmprs] no comprimits envien un missatge al seu pare amb ASK_Z, és a dir, preguntant si es poden comprimir.

```
[C]Send Msg Leaf to zip N
300
A1000 SCmprs =C0250000 !MNCAN_Z
MNASK_Z
```

```
[C]Send Msg Leaf to zip W
300
A0100 SCmprs =C0250000 !MWCAN_Z
MWASK_Z
```

```
[C]Send Msg Leaf to zip E
300
A0010 SCmprs =C0250000 !MECAN_Z
MEASK_Z
```

```
[C]Send Msg Leaf to zip S
300
A0001 SCmprs =C0250000 !MSCAN_Z
MSASK_Z
```

La constant C025 fa la funció de booleà, per indicar si un mòdul està comprimit o no. És a dir, si val 0, aquest mòdul no està comprimit, per contra si val 1, aquest mòdul ja està comprimit. En les precondicions també hi ha una altre condició perquè el mòdul no envii el missatge si ja ha rebut una confirmació, ja que si no, aquest missatge s'enviava altre cop després d'haver fet la compressió, en la direcció que correspongués i provocava un error.

B.1.2. CAN_Z

Quan un mòdul fulla envia un missatge preguntant per si pot comprimir-se, el pare li envia una resposta afirmativa, sempre que ell també estigui descomprimit, per evitar un col·lapse, i no tingui cap fill que vulgui fer un intercanvi (SWZIP) que explicarem a l'Annex B.2. A més, en aquest algorisme donarem preferència a l'expansió abans que la compressió, quan estiguin les dues fases executant-se alhora, podem tenir conflictes amb els missatges d'expansió. Per aquesta raó, hi ha precondicions que diuen que no ha de rebre cap missatge d'expansió.

```
[C]Recieve Msg Leaf to zip N
300
A***1 !(!SRootL !SRootS !SCmprs !SExpnd) =C0250000 MSASK_Z !MNASK_Z
      !MWASK_Z !MEASK_Z !MSEXPND !MNEXPND !MEEXPND !MWEXPND
MSCAN_Z C025+C0250002 C002-C0020001
```

```
[C]Recieve Msg Leaf to zip W
300
A**1* !(!SRootL !SRootS !SCmprs !SExpnd) =C0250000 !MNASK_Z !MWASK_Z
      MEASK_Z !MSEXPND !MNEXPND !MEEXPND !MWEXPND
MECAN_Z C025+C0250002 C002-C0020010
```

```
[C]Recieve Msg Leaf to zip E
300
A*1** !(!SRootL !SRootS !SCmprs !SExpnd) =C0250000 !MNASK_Z MWASK_Z
      !MSEXPND !MNEXPND !MEEXPND !MWEXPND
MWCAN_Z C025+C0250002 C002-C0020100
```

```
[C]Recieve Msg Leaf to zip S
300
A1*** !(!SRootL !SRootS !SCmprs !SExpnd) =C0250000 MNASK_Z !MSEXPND
      !MNEXPND !MEEXPND !MWEXPND
MNCAN_Z C025+C0250002 C002-C0021000
```

Podem veure que si es compleix la regla, el comptador *C025* té el valor 2, és un pas intermedi per evitar conflictes, més tard veurem que els mòduls que tenen aquest valor el retornen a 1. Amb aquest retard fem que el mòdul pare no pugui fer cap acció més fins que no hagi rebut el seu convidat. Modifiquem també la constant *C002* per eliminar el mòdul fulla que ha de fer la compressió.

Són les mateixes regles per cada direcció, però per donar prioritats, les direccions amb menys prioritat també tenen la condició de no rebre cap missatge d'una direcció amb una prioritat més alta.

B.1.3. Moviment Zip

Finalment, quan un mòdul fulla descomprimit rep un missatge *CAN_Z*, farà l'acció de compressió sempre i quan encara estigui en condicions de poder fer-la. Aquest mòdul ha de modificar la constant *C025*, per indicar que està comprimit, i la constant *C024* per indicar que és un mòdul convidat (amb el valor 1).

```
[C]Zip N
300
A1000 SCmprs MNCAN_Z =C0250000
C025+C0250001 C024+C0240001 ZN
```

```
[C]Zip W
300
A0100 SCmprs MWCAN_Z =C0250000
C025+C0250001 C024+C0240001 ZW
```

```
[C]Zip E
300
A0010 SCmprs MECAN_Z =C0250000
C025+C0250001 C024+C0240001 ZE
```

```
[C]Zip S
300
A0001 SCmprs MSCAN_Z =C0250000
C025+C0250001 C024+C0240001 ZS
```

B.2. SWZIP

Per continuar la compressió, tots els mòduls 2-caselles volen enviar el seu mòdul convidat en direcció al seu pare. Per fer-ho, seguim un procés semblant al de les fulles, però usant l'acció **SWZIP**.

B.2.1. ASKSZ

De forma anàloga a l'anterior, els mòduls comprimits amfitrió, envien un missatge al seu pare **ASKSZ** preguntant si pot enviar el seu mòdul convidat.

```
[C]Send Msg ziped to swzip N
300
A1*** SCmprs =C0011000 =C0250001 !=C0240001
MNASKSZ
```

```
[C]Send Msg ziped to swzip W
300
A*1**SCmprs =C0010100 =C0250001 !=C0240001
MWASKSZ
```

```
[C]Send Msg ziped to swzip E
300
A**1* SCmprs =C0010010 =C0250001 !=C0240001
MEASKSZ
```

```
[C]Send Msg ziped to swzip S
300
A***1 SCmprs =C0010001 =C0250001 !=C0240001
MSASKSZ
```

Podem observar que en les precondicions no hem afegit, pensant en la versió de ZIP, la condició de **!M*CANSZ**. Però només cal observar que en aquest cas, el mòdul que fa l'acció de **SWZIP** és l'amfitrió, que no és mou de la posició inicial.

B.2.2. CANSZ

De forma anàloga al ZIP, si un mòdul està en condicions de rebre un convidat, i rep un missatge de **ASKSZ**, envia un missatge de confirmació. Cal veure que les fulles tenen preferència sobre les 2-caselles, així donem preferència a eliminar les branques.

```
[C]Recieve Msg Zip to swzip N
300
A***1 !(!SRootL !SRootS !SCmprs !SExpnd) =C0250000 MSASKSZ !MNASK_Z
      !MWASK_Z !MEASK_Z !MNASKSZ !MWASKSZ !MEASKSZ !MSEXPND !MNEXPND
      !MEEEXPND !MWEXPND
MSCANSZ C025+C0250002
```

```
[C]Recieve Msg Zip to swzip W
300
A**1* !(SRootL !SRootS !SCmprs !SExpnd) =C0250000 MEASKSZ !MNASK_Z
      !MWASK_Z !MSASK_Z !MNASKSZ !MWASKSZ !MSEXPND !MNEXPND !MEEXPND
      !MWEXPND
MECANSZ C025+C0250002
```

```
[C]Recieve Msg Zip to swzip E
300
A*1** !(SRootL !SRootS !SCmprs !SExpnd) =C0250000 MWASKSZ !MNASK_Z
      !MSASK_Z !MEASK_Z !MNASKSZ !MSEXPND !MNEXPND !MEEXPND !MWEXPND
MWCANSZ C025+C0250002
```

```
[C]Recieve Msg Zip to swzip S
300
A1*** !(SRootL !SRootS !SCmprs !SExpnd) =C0250000 MNASKSZ !MSASK_Z
      !MWASK_Z !MEASK_Z !MSEXPND !MNEXPND !MEEXPND !MWEXPND
MNCANSZ C025+C0250002
```

En aquest cas no hem canviat els fills, ja que el pare rep un mòdul d'una 2-casella.

B.2.3. Moviment Swzip

Si rep un missatge de confirmació CANSZ, el mòdul amfitrió envia al seu pare el mòdul convidat.

```
[C]Swzip N
300
A1*** SCmprs MNCANSZ =C0250001 !=C0240001
C025-C0250001 xN
```

```
[C]Swzip W
300
A*1** SCmprs MWCANSZ =C0250001 !=C0240001
C025-C0250001 xW
```

```
[C]Swzip E
300
A**1* SCmprs MECANSZ =C0250001 !=C0240001
C025-C0250001 xE
```

```
[C]Swzip S
300
A***1 SCmprs MSCANSZ =C0250001 !=C0240001
C025-C0250001 xS
```

B.3. Reparacions en les constants

Com hi ha moments en que les fases d'expansió i compressió s'estan executant alhora, podem tenir conflictes amb els missatges. Per solucionar els problemes descrits a les Seccions 4.2.3 i 4.3.3, hem afegit algunes constants i missatges de correcció.

```
[C] DisAttach repair swaps S
300
A***0 !(MSCANSZ !MSCAN_Z)
MSNO_SZ
```

```
[C] DisAttach repair swaps N
300
A0*** !(MNCANSZ !MNCAN_Z)
MNNO_SZ
```

```
[C] DisAttach repair swaps E
300
A**0* !(MECANSZ !MECAN_Z)
MENO_SZ
```

```
[C] DisAttach repair swaps W
300
A*0** !(MWCANSZ !MWCAN_Z)
MWNO_SZ
```

Si estem en un enviament de missatges per fer un ZIP o SWZIP, i els dos mòduls en la operació es desconnecten a causa de les regles d'expansió, quan el mòdul fill rep resposta afirmativa del seu pare, envia un missatge al seu pare NO_SZ, és a dir, que no ha pogut comprimir-se.

```
[C] NO SWAP msg
300
M*NO_SZ >C0250000
C025+00000000
```

Si el mòdul pare rep el missatge NO_SZ, vol dir que el seu fill no ha pogut comprimir-se, per tant, canvia el valor de C025 a 0 altre cop.

```
[C]Repair C25
300
=C0250002
C025+00000001
```

Aquesta regla modifica el valor de la constant C025 a 1 dels mòduls amfitrió que han rebut un mòdul, indicant que poden continuar amb qualsevol altre acció.

Annex C

Regles de l'expansió

En aquest apartat es detallen les regles utilitzades en la fase d'expansió. L'objectiu i l'estratègia general d'aquest conjunt de regles es descriuen a la Secció 4.3 d'aquest treball.

Al acabar la creació de l'arbre, un líder s'ha creat a l'arrel, és a dir, hem canviat l'estat de l'arrel a [RootL] per indicar que el líder de l'expansió està en aquesta posició. Quan arriba el primer mòdul convidat de la compressió, a la posició de l'arrel, ja estem en condicions de començar l'expansió.

Les regles d'aquesta secció comencen per [E] per indicar que és una regla de la fase d'expansió.

C.1. Expansió del líder

El primer moviment el fa el mòdul arrel-líder que només pot expandir en dues direccions (E,S) per com ha estat escollida l'arrel. A més, podem tenir dos casos en la direcció que volem expandir, que la posició estigui buida, o per altra banda, que estigui ocupada per un mòdul en compressió on aquest per tant, està connectat a l'arrel.

Les regles d'aquesta secció estan ordenades de la següent forma: Primer busquem les expansions que pot fer el líder a posicions buides o a aquelles en que està ocupada per un mòdul en compressió ja connectat al líder. Després busquem les expansions on el líder ha de fer la connexió.

C.1.1. Expansió del líder a una posició buida

En el cas que el líder escull una posició buida per expandir-se, no trobem cap problema. El mòdul convidat ocupa la posició buida fent l'acció UNZIP i el líder li dóna el lideratge amb el missatge LIDER. Veiem les regles per al cas on el líder està a l'arrel.

[E]Expand Empty E Root

300

SRootL =C0200000 =C0040010 E1,0 =C0250001 =C0240000
 SRootP C020+00010000 C025-C0250001 C002+C0020010 C005+00000010
 C004-C0040010 zESPLIDR MELIDER #E02+00000001

[E]Expand Empty S Root

300

SRootL =C0200000 =C0040001 E0,-1 =C0250001 =C0240000
 SRootP C020+00010000 C025-C0250001 C002+C0020001 C005+00000001
 C004-C0040001 zSSPLIDR MSLIDER #S02+00000001

[E]Expand Empty SE->S Root

300

SRootL =C0200000 =C0040011 E0,-1 =C0250001 =C0240000
 SRootP C020+00010000 C025-C0250001 C002+C0020001 C005+00000001
 C004-C0040001 zSSPLIDR MSLIDER #S02+00000001

Veiem el significat de cada constant:

- C002 són els fills que ja té fixats aquest mòdul en l'arbre d'expansió.
- C004 indica els fills que encara li falten per fer la copia de l'arbre objectiu.
- C005 senyala la direcció a la qual es troba el líder, així si a un mòdul amb estat [Expnd] li arriba un mòdul comprimit, l'envia en aquesta direcció.
- C020 provoca un retràs del mòdul que envia el líder així no executa cap altre regla abans de la següent iteració, l'usarem en tots els enviaments de missatges.
- C024 és el booleà que indica si un mòdul és un mòdul convidat.
- C025 és el booleà que indica si un mòdul està comprimit.

Així doncs, l'antic líder queda en estat [RootP] esperant la confirmació del seu nou líder i ha enviat el líder alhora que ha fet l'acció UNZIP. El missatge numèric del canal #*02, serveix per indicar al nou líder que ja no està comprimit, més endavant veure'm la regla que aplica l'acció.

Un cop vist el cas del líder a l'arrel, els moviments d'expansió a una posició buida per la resta de caselles és anàleg, només hem canviat els estats corresponents al líder i a un mòdul expansió de l'arbre que no sigui arrel.

[E]Expand Empty N

300

SLIDER =C0200000 =C0041000 E0,1 =C0250001 =C0240000
 SPExpN C020+00010000 C025-C0250001 C002+C0021000 C005+00001000
 C004-C0041000 zNSPLIDR MNLIDER #N02+00000001

[E]Expand Empty W

300

SLIDER =C0200000 =C0040100 E-1,0 =C0250001 =C0240000
 SPExpN C020+00010000 C025-C0250001 C002+C0020100 C005+00000100
 C004-C0040100 zWSPLIDR MWLIDER #W02+00000001

[E]Expand Empty E

300

SLIDER =C0200000 =C0040010 E1,0 =C0250001 =C0240000

SPExpn C020+00010000 C025-C0250001 C002+C0020010 C005+00000010

C004-C0040010 zESPLIDR MELIDER #E02+00000001

[E]Expand Empty S

300

SLIDER =C0200000 =C0040001 E0,-1 =C0250001 =C0240000

SPExpn C020+00010000 C025-C0250001 C002+C0020001 C005+00000001

C004-C0040001 zSSPLIDR MSLIDER #S02+00000001

[E]Expand Empty SE->S

300

SLIDER =C0200000 =C0040011 E0,-1 =C0250001 =C0240000

SPExpn C020+00010000 C025-C0250001 C002+C0020001 C005+00000001

C004-C0040001 zSSPLIDR MSLIDER #S02+00000001

[E]Expand Empty SW->S

300

SLIDER =C0200000 =C0040101 E0,-1 =C0250001 =C0240000

SPExpn C020+00010000 C025-C0250001 C002+C0020001 C005+00000001

C004-C0040001 zSSPLIDR MSLIDER #S02+00000001

[E]Expand Empty SN->S

300

SLIDER =C0200000 =C0041001 E0,-1 =C0250001 =C0240000

SPExpn C020+00010000 C025-C0250001 C002+C0020001 C005+00000001

C004-C0040001 zSSPLIDR MSLIDER #S02+00000001

[E]Expand Empty EW->E

300

SLIDER =C0200000 =C0040110 E1,0 =C0250001 =C0240000

SPExpn C020+00010000 C025-C0250001 C002+C0020010 C005+00000010

C004-C0040010 zESPLIDR MELIDER #E02+00000001

[E]Expand Empty EN->E

300

SLIDER =C0200000 =C0041010 E1,0 =C0250001 =C0240000

SPExpn C020+00010000 C025-C0250001 C002+C0020010 C005+00000010

C004-C0040010 zESPLIDR MELIDER #E02+00000001

[E]Expand Empty WN->W

300

SLIDER =C0200000 =C0041100 E-1,0 =C0250001 =C0240000

SPExpn C020+00010000 C025-C0250001 C002+C0020100 C005+00000100

C004-C0040100 zWSPLIDR MWLIDER #W02+00000001

```
[E]Expand Empty SEW->S
300
SLIDER =C0200000 =C0040111 E0,-1 =C0250001 =C0240000
SPExpn C020+00010000 C025-C0250001 C002+C0020001 C005+00000001
      C004-C0040001 zSSPLIDR MSLIDER #S02+00000001
```

```
[E]Expand Empty SEN->S
300
SLIDER =C0200000 =C0041011 E0,-1 =C0250001 =C0240000
SPExpn C020+00010000 C025-C0250001 C002+C0020001 C005+00000001
      C004-C0040001 zSSPLIDR MSLIDER #S02+00000001
```

```
[E]Expand Empty SWN->S
300
SLIDER =C0200000 =C0041101 E0,-1 =C0250001 =C0240000
SPExpn C020+00010000 C025-C0250001 C002+C0020001 C005+00000001
      C004-C0040001 zSSPLIDR MSLIDER #S02+00000001
```

```
[E]Expand Empty EWN->E
300
SLIDER =C0200000 =C0041110 E1,0 =C0250001 =C0240000
SPExpn C020+00010000 C025-C0250001 C002+C0020010 C005+00000010
      C004-C0040010 zESPLIDR MELIDER #E02+00000001
```

C.1.2. Expansió del líder a una posició ocupada, connectada

Si el líder vol expandir-se a una posició ocupada, aquesta serà de compressió. Com ja estan els dos mòduls connectats només cal enviar un missatge perquè passi a ser el líder i actualitzar les constants necessàries. Cal remarcar que el missatge és el mateix que hem usat a la secció anterior, ja que el mòdul que rep el lideratge, ha de fer les mateixes accions en els dos casos, el missatge l'estudiem en unes regles que explicarem a la següent secció.

```
[E]Expand Full E Root
300
SRootL =C0200000 =C0040010 F1,0 T1,0,Cmprs =C0250001 =C0240000
C020+00010000 C005+00000010 C002+C0020010 C004-C0040010 MELIDER
```

```
[E]Expand Full S Root
300
SRootL =C0200000 =C0040001 F0,-1 T0,-1,Cmprs =C0250001 =C0240000
C020+00010000 C005+00000001 C002+C0020001 C004-C0040001 MSLIDER
```

```
[E]Expand Full SE->S Root
300
SRootL =C0200000 =C0040011 F0,-1 T0,-1,Cmprs =C0250001 =C0240000
C020+00010000 C005+00000001 C002+C0020001 C004-C0040001 MSLIDER
```

[E]Expand Full Lider Attached N

300

SLIDER =C0200000 =C0041000 A1*** T0,1,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0021000 C005+00001000 C004-C0041000 MNLIDER

[E]Expand Full Lider Attached S

300

SLIDER =C0200000 =C0040001 A***1 T0,-1,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020001 C005+00000001 C004-C0040001 MSLIDER

[E]Expand Full Lider Attached W

300

SLIDER =C0200000 =C0040100 A*1** T-1,0,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020100 C005+00000100 C004-C0040100 MWLIDER

[E]Expand Full Lider Attached E

300

SLIDER =C0200000 =C0040010 A**1* T1,0,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020010 C005+00000010 C004-C0040010 MELIDER

[E]Expand Full Lider Attached SE->S

300

SLIDER =C0200000 =C0040011 A***1 T0,-1,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020001 C005+00000001 C004-C0040001 MSLIDER

[E]Expand Full Lider Attached SW->S

300

SLIDER =C0200000 =C0040101 A***1 T0,-1,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020001 C005+00000001 C004-C0040001 MSLIDER

[E]Expand Full Lider Attached SN->S

300

SLIDER =C0200000 =C0041001 A***1 T0,-1,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020001 C005+00000001 C004-C0040001 MSLIDER

[E]Expand Full Lider Attached EW->E

300

SLIDER =C0200000 =C0040110 A**1* T1,0,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020010 C005+00000010 C004-C0040010 MELIDER

[E]Expand Full Lider Attached EN->E

300

SLIDER =C0200000 =C0041010 A**1* T1,0,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020010 C005+00000010 C004-C0040010 MELIDER

```
[E]Expand Full Lider Attached WN->W
300
SLIDER =C0200000 =C0041100 A*1** T-1,0,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020100 C005+00000100 C004-C0040100 MWLIDER
```

```
[E]Expand Full Lider Attached SEW->S
300
SLIDER =C0200000 =C0040111 A***1 T0,-1,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020001 C005+00000001 C004-C0040001 MSLIDER
```

```
[E]Expand Full Lider Attached SEN->S
300
SLIDER =C0200000 =C0041011 A***1 T0,-1,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020001 C005+00000001 C004-C0040001 MSLIDER
```

```
[E]Expand Full Lider Attached SWN->S
300
SLIDER =C0200000 =C0041101 A***1 T0,-1,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020001 C005+00000001 C004-C0040001 MSLIDER
```

```
[E]Expand Full Lider Attached EWN->E
300
SLIDER =C0200000 =C0041110 A**1* T1,0,Cmprs =C0250001 =C0240000
C020+00010000 C002+C0020010 C005+00000010 C004-C0040010 MELIDER
```

C.1.3. Tractament dels missatges LIDER

El missatge que envia el líder, es rebut per un mòdul en compressió [Cmprs] o per un mòdul acabat de descomprimir amb les regles de la expansió a un espai buit amb estat PLIDR, usem aquestes regles per als dos casos per a que el mòdul sigui el líder.

```
[E]Change to Expand S
300
!(SCmprs !SPLIDR) MNLIDER
SLIDER C023 + 0,1,C0230000 C000 + C23,0,C0000000
C004 + C23,0,C0020000 C001+00001000 C002+00000000
```

```
[E]Change to Expand E
300
!(SCmprs !SPLIDR) MWLIDER
SLIDER C023 + -1,0,C0230000 C000 + C23,0,C0000000
C004 + C23,0,C0020000 C001+00000100 C002+00000000
```

```
[E]Change to Expand W
300
!(SCmprs !SPLIDR) MELIDER
SLIDER C023 + 1,0,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000010 C002+00000000
```

```
[E]Change to Expand N
300
!(SCmprs !SPLIDR) MSLIDER
SLIDER C023 + 0,-1,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000001 C002+00000000
```

En el cas que el mòdul rep el líder al fer un UNZIP, hem d'arreglar les constants que indiquen si estem comprimits. Hem usat el missatge numèric del canal **#*02** per diferenciar aquest cas.

```
[E]Change to Expand S Repair C25
300
SPLIDR MNLIDER =#N020001
C025-C0250001 C024-C0240001
```

```
[E]Change to Expand E Repair C25
300
SPLIDR MWLIDER =#W020001
C025-C0250001 C024-C0240001
```

```
[E]Change to Expand W Repair C25
300
SPLIDR MELIDER =#E020001
C025-C0250001 C024-C0240001
```

```
[E]Change to Expand N Repair C25
300
SPLIDR MSLIDER =#S020001
C025-C0250001 C024-C0240001
```

C.1.4. Expansió a una posició ocupada, desconnectada

L'últim cas d'expansió que pot trobar-se el líder, és una posició ocupada per un mòdul de compressió on no hi hagi connexió física entre els dos. En aquest cas fem un enviament de senyals amb resposta per evitar el problema descrit a la . El líder envia un missatge DISAL i espera la confirmació del seu mòdul objectiu.

```
[E]Expand Full Lider DISAttached N
300
SLIDER =C0200000 =C0041000 A0*** T0,1,Cmprs =C0250001 =C0240000
MNDISAL C020+00010000
```

[E]Expand Full Lider DISAttached S
300
SLIDER =C0200000 =C0040001 A***0 T0,-1,Cmprs =C0250001 =C0240000
MSDISAL C020+00010000

[E]Expand Full Lider DISAttached W
300
SLIDER =C0200000 =C0040100 A*0** T-1,0,Cmprs =C0250001 =C0240000
MWDISAL C020+00010000

[E]Expand Full Lider DISAttached E
300
SLIDER =C0200000 =C0040010 A**0* T1,0,Cmprs =C0250001 =C0240000
MEDISAL C020+00010000

[E]Expand Full Lider DISAttached SE->S
300
SLIDER =C0200000 =C0040011 A***0 T0,-1,Cmprs =C0250001 =C0240000
MSDISAL C020+00010000

[E]Expand Full Lider DISAttached SW->S
300
SLIDER =C0200000 =C0040101 A***0 T0,-1,Cmprs =C0250001 =C0240000
MSDISAL C020+00010000

[E]Expand Full Lider DISAttached SN->S
300
SLIDER =C0200000 =C0041001 A***0 T0,-1,Cmprs =C0250001 =C0240000
MSDISAL C020+00010000

[E]Expand Full Lider DISAttached EW->E
300
SLIDER =C0200000 =C0040110 A**0* T1,0,Cmprs =C0250001 =C0240000
MEDISAL C020+00010000

[E]Expand Full Lider DISAttached EN->E
300
SLIDER =C0200000 =C0041010 A**0* T1,0,Cmprs =C0250001 =C0240000
MEDISAL C020+00010000

[E]Expand Full Lider DISAttached WN->W
300
SLIDER =C0200000 =C0041100 A*0** T-1,0,Cmprs =C0250001 =C0240000
MWDISAL C020+00010000

```
[E]Expand Full Lider DISAttached SEW->S
300
SLIDER =C0200000 =C0040111 A***0 T0,-1,Cmprs =C0250001 =C0240000
MSDISAL C020+00010000
```

```
[E]Expand Full Lider DISAttached SEN->S
300
SLIDER =C0200000 =C0041011 A***0 T0,-1,Cmprs =C0250001 =C0240000
MSDISAL C020+00010000
```

```
[E]Expand Full Lider DISAttached SNW->S
300
SLIDER =C0200000 =C0041101 A***0 T0,-1,Cmprs =C0250001 =C0240000
MSDISAL C020+00010000
```

```
[E]Expand Full Lider DISAttached ENW->E
300
SLIDER =C0200000 =C0041110 A**0* T1,0,Cmprs =C0250001 =C0240000
MEDISAL C020+00010000
```

El mòdul que rep el missatge DISAL fa totes les preparacions per ser el líder, sempre i quan no hagi rebut un missatge per completar un ZIP, que llavors ignora el lideratge perquè pot buidar la posició perquè a la següent iteració l'utilitzi l'actual líder.

El mòdul que ha de convertir-se en líder és connecta al líder actual d'on prové el missatge, li retorna un missatge de confirmació EXPDL, i després és desconnecta del seu actual mòdul pare per trencar el cicle. Després de fer la desconnexió, canvia a estat líder i canvia les constants necessàries per ser el líder.

Aquest pas ha estat fet en dues regles diferents perquè el mòdul no faci la connexió i desconnexió alhora, pas que pot portar problemes en un robot real.

```
[E]Change to Expand DisAttach 1 S==PN
300
SCmprs MNDISAL =C0011000 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MNEXPDL A1***
```

```
[E]Change to Expand DisAttach 2 S==PN
300
SCmprs MNDISAL =C0011000 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A0*** SLIDER C023 + 0,1,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00001000 C002+00000000
```

```
[E]Change to Expand DisAttach 1 S==PW
300
SCmprs MNDISAL =C0010100 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MNEXPDL A1***
```

```
[E]Change to Expand DisAttach 2 S==PW
300
SCmprs MNDISAL =C0010100 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A*0** SLIDER C023 + 0,1,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00001000 C002+00000000
```

```
[E]Change to Expand DisAttach 1 S==PE
300
SCmprs MNDISAL =C0010010 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MNEXPDL A1***
```

```
[E]Change to Expand DisAttach 2 S==PE
300
SCmprs MNDISAL =C0010010 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A**0* SLIDER C023 + 0,1,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00001000 C002+00000000
```

```
[E]Change to Expand DisAttach 1 S==PS
300
SCmprs MNDISAL =C0010001 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MNEXPDL A1***
```

```
[E]Change to Expand DisAttach 2 S==PS
300
SCmprs MNDISAL =C0010001 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A***0 SLIDER C023 + 0,1,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00001000 C002+00000000
```

```
[E]Change to Expand DisAttach 1 E==PN
300
SCmprs MWDISAL =C0011000 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MWEXPDL A*1**
```

```
[E]Change to Expand DisAttach 2 E==PN
300
SCmprs MWDISAL =C0011000 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A0*** SLIDER C023 + -1,0,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000100 C002+00000000
```

```
[E]Change to Expand DisAttach 1 E==PW
300
SCmprs MWDISAL =C0010100 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MWEXPDL A*1**
```



```
[E]Change to Expand DisAttach 2 E==PW
300
SCmprs MWDISAL =C0010100 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A*0** SLIDER C023 + -1,0,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000100 C002+00000000
```

```
[E]Change to Expand DisAttach 1 E==PE
300
SCmprs MWDISAL =C0010010 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MWEXPDL A*1**
```

```
[E]Change to Expand DisAttach 2 E==PE
300
SCmprs MWDISAL =C0010010 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A**0* SLIDER C023 + -1,0,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000100 C002+00000000
```

```
[E]Change to Expand DisAttach 1 E==PS
300
SCmprs MWDISAL =C0010001 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MWEXPDL A*1**
```

```
[E]Change to Expand DisAttach 2 E==PS
300
SCmprs MWDISAL =C0010001 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A***0 SLIDER C023 + -1,0,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000100 C002+00000000
```

```
[E]Change to Expand DisAttach 1 W==PN
300
SCmprs MEDISAL =C0011000 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MEEXPDL A**1*
```

```
[E]Change to Expand DisAttach 2 W==PN
300
SCmprs MEDISAL =C0011000 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A0*** SLIDER C023 + 1,0,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000010 C002+00000000
```

```
[E]Change to Expand DisAttach 1 W==PW
300
SCmprs MEDISAL =C0010100 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MEEXPDL A**1*
```

```
[E]Change to Expand DisAttach 2 W==PW
300
SCmprs MEDISAL =C0010100 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A*0** SLIDER C023 + 1,0,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000010 C002+00000000
```

```
[E]Change to Expand DisAttach 1 W==PE
300
SCmprs MEDISAL =C0010010 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MEEXPDL A**1*
```

```
[E]Change to Expand DisAttach 2 W==PE
300
SCmprs MEDISAL =C0010010 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A**0* SLIDER C023 + 1,0,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000010 C002+00000000
```

```
[E]Change to Expand DisAttach 1 W==PS
300
SCmprs MEDISAL =C0010001 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MEEXPDL A**1*
```

```
[E]Change to Expand DisAttach 2 W==PS
300
SCmprs MEDISAL =C0010001 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A***0 SLIDER C023 + 1,0,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000010 C002+00000000
```

```
[E]Change to Expand DisAttach 1 N==PN
300
SCmprs MSDISAL =C0011000 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MSEXPDL A***1
```

```
[E]Change to Expand DisAttach 2 N==PN
300
SCmprs MSDISAL =C0011000 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A0*** SLIDER C023 + 0,-1,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000001 C002+00000000
```

```
[E]Change to Expand DisAttach 1 N==PW
300
SCmprs MSDISAL =C0010100 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MSEXPDL A***1
```

```
[E]Change to Expand DisAttach 2 N==PW
300
SCmprs MSDISAL =C0010100 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A*0** SLIDER C023 + 0,-1,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000001 C002+00000000
```

```
[E]Change to Expand DisAttach 1 N==PE
300
SCmprs MSDISAL =C0010010 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MSEXPDL A***1
```

```
[E]Change to Expand DisAttach 2 N==PE
300
SCmprs MSDISAL =C0010010 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A**0* SLIDER C023 + 0,-1,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000001 C002+00000000
```

```
[E]Change to Expand DisAttach 1 N==PS
300
SCmprs MSDISAL =C0010001 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
MSEXPDL A***1
```

```
[E]Change to Expand DisAttach 2 N==PS
300
SCmprs MSDISAL =C0010001 !MNCAN_Z !MSCAN_Z !MECAN_Z !MWCAN_Z
A***0 SLIDER C023 + 0,-1,C0230000 C000 + C23,0,C0000000
      C004 + C23,0,C0020000 C001+00000001 C002+00000000
```

Quan el mòdul objectiu ha fet el canvi a líder, ha enviat un missatge de confirmació al anterior líder. Aquest antic líder haurà perdut l'estat de líder a causa de la constant C020. El líder quan rep el missatge, comprova que el seu veí és líder i actualitza les constants. Fa la comprovació per evitar el problema de *Compressió prematura* on el mòdul objectiu és comprimeix quan ha de rebre el líder. En el cas de no trobar el líder, aplica una regla de correcció que veurem més endavant, i torna a agafar el lideratge.

```
[E]Expanded Full Lider DISAttached N
300
SExpnd T0,1,LIDER =C0250001 =C0240000 MNEXPDL
A1*** C002+C0021000 C005+00001000 C004-C0041000
```

```
[E]Expanded Full Lider DISAttached S
300
SExpnd T0,-1,LIDER =C0250001 =C0240000 MSEXPDL
A***1 C002+C0020001 C005+00000001 C004-C0040001
```

```
[E]Expanded Full Lider DISAttached W
300
SExpnd T-1,0,LIDER =C0250001 =C0240000 MWEXPDL
A*1** C002+C0020100 C005+00000100 C004-C0040100
```

```
[E]Expanded Full Lider DISAttached E
300
SExpnd T1,0,LIDER =C0250001 =C0240000 MEEXPDL
A**1* C002+C0020010 C005+00000010 C004-C0040010
```

C.1.5. Reparació del líder

Hem vist que sempre que el líder fa una regla aquesta constant canvia el seu valor a 1, això és degut a que no volem que el líder faci més d'una acció alhora per evitar col·lapsaments, però no sempre pot perdre l'estat [LIDER] sense rebre confirmació. Amb aquestes dues regles fem que el líder sigui un mòdul [Expnd] al haver fet la seva funció.

```
[E]Change Lider to Expand S
300
!(SLIDER !SPExpnd) =C0200001
SExpnd C020+00000000
```

```
[E]Change Root Lider to RootS S
300
!(SRootL !SRootP) =C0200001
SRootS C020+00000000
```

En alguns casos, un mòdul que rep el lideratge no l'accepta. En les precondicions d'aquestes regles volem comprovar si el mòdul no ha actualitzat les seves dades, condició que vol dir que no ha rebut confirmació del nou líder i per tant, ha de recuperar el líder.

```
[E]IM THE Root LIDER
300
SRootS =C0050000 !=C0040000 !MNEXPDL !MSEXPDL !MEEXPDL !MWEXPDL
SRootL
```

```
[E]IM THE LIDER
300
SExpnd =C0050000 !MNEXPDL !MSEXPDL !MEEXPDL !MWEXPDL
SLIDER
```

C.2. Seguiment del líder

Hem vist com s'expandeix el líder, però la condició comuna és que el líder ha d'estar comprimit per poder expandir-se. Així doncs, falta que els altres mòduls arribin al líder. Sempre que el líder avança, la constant *C005* s'actualitza per indicar la direcció cap a on es troba el líder, fent la funció de punter. En les regles d'aquesta secció, els mòduls comprimits en estat [Expnd] intenten enviar el seu convidat cap a la direcció que marca aquesta constant. Les regles que usem per moure els mòduls són anàlegs a les que hem vist per fer la compressió de les 2-caselles a l'Annex B. La única variació és que el mòdul que ha de fer la confirmació, pot rebre un missatge pel canal *#*01* enviat pel líder avisant que ha arribat a una fulla, i està tornant enrere cap a la següent intersecció per continuar el camí. Les regles pel missatge estan explicades en aquest annex més endavant.

```
[E]Send Msg ziped to Expand-F S
300
A***1 !(SRootS !SRootL !SExpnd) =C0050001 F0,-1 !(T0,-1,Expnd
      !T0,-1,LIDER !T0,-1,RootS !T0,-1,RootL) =C0250001 =C0240000
MSEXPND
```

```
[E]Receive Msg ziped to Expand-F S
300
A1*** !(SRootS !SExpnd !SLIDER) =C0250000 !=C0051000 !MSEXPND
      !MWEXPND !MEEXPND MNEXPND !=#*010001
MNCANEX C025+C0250002
```

```
[E]Receive Root Msg ziped to Expand-F S
300
A1*** SRootL =C0250000 !MSEXPND !MWEXPND !MEEXPND MNEXPND
      !=#*010001
MNCANEX C025+C0250002
```

```
[E]Expand-F S
300
A***1 !(SRootS !SRootL !SExpnd ) MSCANEX =C0250001 !=C0240001
C025-C0250001 xS
```

```
[E]Send Msg ziped to Expand-F N
300
A1*** !(SRootS !SRootL !SExpnd) =C0051000 F0,1 !(T0,1,Expnd
      !T0,1,LIDER !T0,1,RootS !T0,1,RootL) =C0250001 =C0240000
MNEXPND
```

```
[E]Receive Msg ziped to Expand-F N
300
A***1 !(SRootS !SExpnd !SLIDER) =C0250000 !=C0050001 MSEXPND
      !MWEXPND !MEEXPND !=#*010001
MSCANEX C025+C0250002
```

[E]Receive Root Msg ziped to Expand-F N

300

A**1* SRootL =C0250000 MEXPND !MWEXPND !MEXPND !=#*010001
MSCANEX C025+C0250002

[E]Expand-F N

300

A1*** !(SRootS !SRootL !SExpnd) MNCANEX =C0250001 !=C0240001
C025-C0250001 xN

[E]Send Msg ziped to Expand-F W

300

A*1** !(SRootS !SRootL !SExpnd) =C0050100 F-1,0 !(T-1,0,Expnd
!T-1,0,LIDER !T-1,0,RootS !T-1,0,RootL) =C0250001 =C0240000
MWEXPND

[E]Receive Msg ziped to Expand-F W

300

A**1* !(SRootS !SExpnd !SLIDER) =C0250000 !=C0050010 MEXPND
!MWEXPND !=#*010001
MECANEX C025+C0250002

[E]Receive Root Msg ziped to Expand-F W

300

A**1* SRootL =C0250000 MEXPND !MWEXPND !=#*010001
MECANEX C025+C0250002

[E]Expand-F W

300

A*1** !(SRootS !SRootL !SExpnd) MWCANEX =C0250001 !=C0240001
C025-C0250001 xW

[E]Send Msg ziped to Expand-F E

300

A**1* !(SRootS !SRootL !SExpnd) =C0050010 F1,0 !(T1,0,Expnd
!T1,0,LIDER !T1,0,RootS !T1,0,RootL) =C0250001 =C0240000
MEXPND

[E]Receive Msg ziped to Expand-F E

300

A*1** !(SRootS !SExpnd !SLIDER) =C0250000 !=C0050100 MWEXPND
!=#*010001
MWCANEX C025+C0250002

```

[E]Receive Root Msg ziped to Expand-F E
300
A*1** SRootL =C0250000 MWEXPND !=#*010001
MWCANEX C025+C0250002

[E]Expand-F E
300
A**1* !(SRootS !SRootL !SExpnd ) MECANEX =C0250001 !=C0240001
C025-C0250001 xE

```

C.3. Retorn del líder

Finalment, el líder després de fer les expansions seguint una branca, arriba a una fulla de l'arbre objectiu. En aquest moment, per seguir el DFS ha de tornar enrere fins arribar a un mòdul amb algun fill no visitat. Veiem el procés de forma més detallada:

C.3.1. Branca completada.

Quan el líder arriba a un mòdul que no té més fills que no hagin estat visitats, envia un missatge pel canal #*01 al seu pare que indica que el líder ha de tornar enrere. Aquest mòdul canvia el punter al seu pare, i canvia la constant C020 per ser un mòdul d'expansió a la iteració següent.

```

Leaf Return Father N
300
SLIDER =C0200000 =C0040000 =C0011000 !=C0200001
#N01+00000001 C020+00010000 C005+00001000

```

```

Leaf Return Father S
300
SLIDER =C0200000 =C0040000 =C0010001 !=C0200001
#S01+00000001 C020+00010000 C005+00000001

```

```

Leaf Return Father W
300
SLIDER =C0200000 =C0040000 =C0010100 !=C0200001
#W01+00000001 C020+00010000 C005+00000100

```

```

Leaf Return Father E
300
SLIDER =C0200000 =C0040000 =C0010010 !=C0200001
#E01+00000001 C020+00010000 C005+00000010

```

C.3.2. Retorn del líder al pare

Quan un mòdul rep el missatge de retorn sempre accepta el lideratge, i no pot acceptar cap altre regla fins al torn següent. Amb la combinació d'aquestes regles i les anteriors, sempre que el líder arriba a una fulla, fa el camí invers per tornar a una branca de l'arbre no completada.

```
[E]Return Signal Lider PS
300
SExpnd =#*010001 =C0010001 =C0040000
SLIDER
```

```
[E]Return Signal Lider PN
300
SExpnd =#*010001 =C0011000 =C0040000
SLIDER
```

```
[E]Return Signal Lider PE
300
SExpnd =#*010001 =C0010010 =C0040000
SLIDER
```

```
[E]Return Signal Lider PW
300
SExpnd =#*010001 =C0010100 =C0040000
SLIDER
```

Aquesta és la regla anàleg per el mòdul arrel.

```
[E]Receive Signal Root Lider
300
SRootS =#*010001
SRootL C005+00000001
```

C.3.3. Enviament del líder al següent fill

Si un mòdul rep el líder de retorn i té algun fill que encara no ha estat visitat, canvia d'estat a [LIDER]. Per tant, a la següent iteració pot fer l'expansió al següent fill no visitat.

```
[E]Receive Signal Lider S
300
SExpnd =#*010001 =C0040001
SLIDER C005+00000001
```

```
[E]Return Signal Lider N
300
SExpnd =#*010001 =C0041000
SLIDER C005+00001000
```



```
[E]Return Signal Lider E
300
SExpnd =#*010001 =C0040010
SLIDER C005+00000010

[E]Return Signal Lider W
300
SExpnd =#*010001 =C0040100
SLIDER C005+00000100

[E]Receive Signal Lider SE->S
300
SExpnd =#*010001 =C0040011
SLIDER C005+00000001

[E]Receive Signal Lider SW->S
300
SExpnd =#*010001 =C0040101
SLIDER C005+00000001

[E]Receive Signal Lider SN->S
300
SExpnd =#*010001 =C0041001
SLIDER C005+00000001

[E]Return Signal Lider EW->E
300
SExpnd =#*010001 =C0040110
SLIDER C005+00000010

[E]Return Signal Lider EN->E
300
SExpnd =#*010001 =C0041010
SLIDER C005+00000010

[E]Return Signal Lider WN->W
300
SExpnd =#*010001 =C0041100
SLIDER C005+00000100
```

Regles anàlegs per al mòdul arrel.

```
[E]Return Signal Lider Root
300
SRootL =#*010001
SRootS C005+C0040000
```

C.4. Fi de líder

Per acabar l'algorisme, quan el líder retorna a l'arrel, i aquesta no té més fills per visitar, haurem acabat l'algorisme de reconfiguració. Per tant, canviem l'estat del líder a [RootS] per eliminar el líder.

```
[E]End Signal Lider Root
```

```
300
```

```
SRootL =C0040000
```

```
SRootS
```

Annex D

Moviments Bàsics

En aquest apartat presentem la representació gràfica dels moviments atòmics per al nostre algorisme, podem comprovar que no hi ha desconnexions en cap moment.

D.1. ZIP i UNZIP

Versió atòmica del moviment ZIP (i UNZIP si invertim els passos).

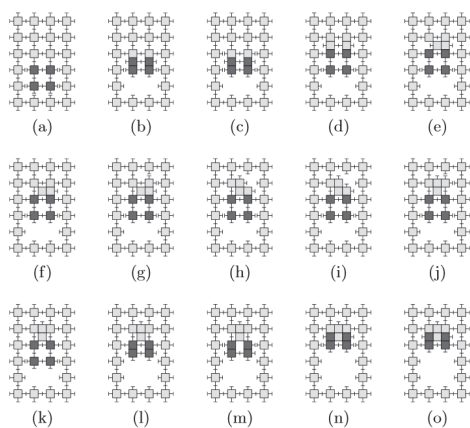


FIGURA 1. Detall del moviment de ZIP, i UNZIP fent el moviment invers.

D.2. SWZIP

Versió atòmica del moviment **SWZIP**.

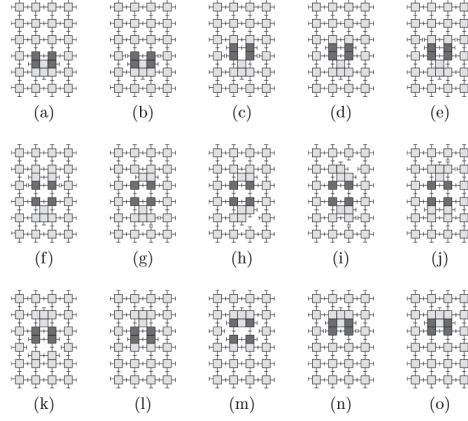


FIGURA 2. Detalls de l'enviament del mòdul convidat en l'acció SWZIP.

D.3. Posicionament

Per fer els moviments anteriors, és fàcil observar que necessitem encarar els àtoms d'una forma determinada abans de començar el moviment atòmic, usem aquest moviment per direccionar els àtoms d'un mòdul. En el treball no necessitem aquest moviment perquè no es pot apreciar en forma modular.

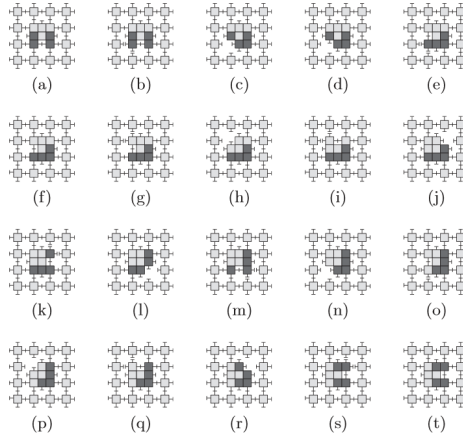


FIGURA 3. Detalls del moviment de *Posicionament*, usat per l'*UNZIP* O *SWZIP* abans d'enviar el mòdul cap a n_2 . El mòdul n_1 fa una rotació per encarar-se cap a la direcció de n_2 .

D.4. Altres moviments

Aquests moviments no els usem en el nostre algorisme, ja que els substituïm per l'enviament de missatges amb dades, tot i així, si és vol usar uns robots amb menys capacitat d'informació, podríem usar aquests moviments:

D.4.1. SWZIP2

Aquest moviment intercanvia els mòduls convidats entre dos amfitrions veïns.

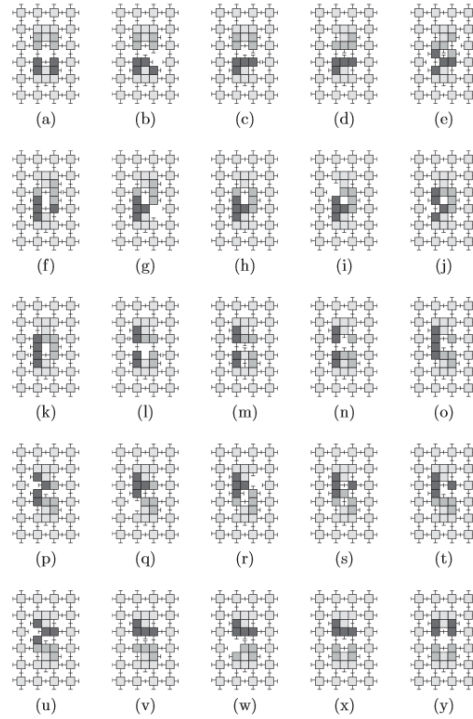


FIGURA 4. Detalls de l'intercanvi de mòduls convidats en l'acció SWZIP amb dues 2-caselles.

D.4.2. SWITCH

Amb aquest moviment, els dos mòduls comprimits d'una 2-casella, canvien els seus papers.

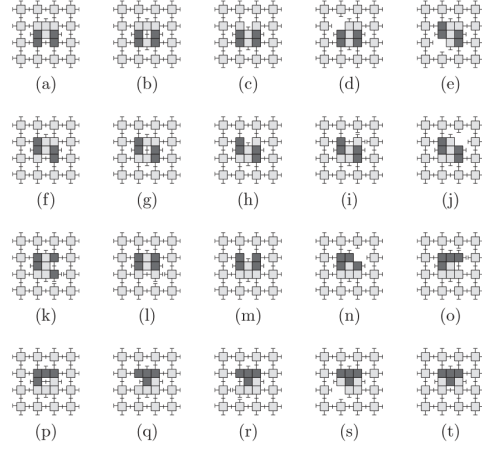


FIGURA 5. Detalls del SWITCH.

Annex E

Compact User Guide

Compact User Guide of the Agent System

B. Vogtenhuber, R. Wallner

extended by O. Rodríguez, J. Soler and revised by V. Sacristán

June 19, 2013

Contents

1	Introduction	3
2	Program flow	3
3	Program inputs	4
3.1	Definition of the initial setting	6
3.2	Definition of the rules	7
3.2.1	The precondition	7
3.2.2	The action	9
3.2.3	Include rules	11
3.2.4	State colors	11
3.3	Program options	12
3.3.1	Layout options	12
3.3.2	Running options	14
3.3.3	Behavior options	15
3.3.4	Limitations	15
4	Program window	16
4.1	Universe	16
4.2	Agents and rules	18
4.2.1	Agents panel	18
4.2.2	Rules panel	20
4.3	actions.log	22
4.4	positions.log	23
4.5	error.log	23
4.6	Agents generator	23
4.6.1	Agents panel	23
4.6.2	Universe panel	24
4.7	Exit menu	27
4.8	Universe menu	27
4.9	Agents menu	27
4.10	Rules menu	27

4.11	Agents generator menu	27
4.12	Options menu	27
4.12.1	Layout options	27
4.12.2	Running options	27
4.12.3	Behavior options	27
4.12.4	Limitations	28
4.13	Help menu	28
5	Limitations	28
6	Program outputs	28
7	Requirements	30
8	Installation	31

1 Introduction

This document describes the functionality of the simulation programs of the Agent System. The design of the system, the chosen representation of the universe as well as examples for the definition of input files are described in [?]. Therefore this document mainly is considered to give a description of the used syntax in input and output files as well as to describe the functionality of the graphical user interface.

2 Program flow

This section is dedicated to explain how the system evaluates the precondition and the priority and how it executes the action of each agent. Since the system supports priorities of the rules, the evaluation model is more complex than it would be without the usage of priorities. But priorities provide a helpful extension of the rules to enable advanced rule sets with simple rules.

The program starts by reading the initial set of agents as well as the set of rules. In every calculation step, the following operations are performed (in the order listed here; alternatively sending messages can be done before performing actions).

1. Check and get valid rules (each step for all agents)
 - (a) check all rules whether they would apply (ignoring priorities) and store valid rules sorted by priority in descending order
 - (b) store the highest priority of valid rules as current priority (if > 0) and set the agents priority to `PRIORITY_OPEN`
 - (c) for all agents sorted by priority in descending order (if priority is `PRIORITY_OPEN`):
do while not all agents have `PRIORITY_FIXED` :
 - fetch current rules to current priority
 - for all rules: check priority-conditions
 - if they are fulfilled, set priority to `PRIORITY_FIXED`
 - if one of them is not fulfilled, remove this rule from the specified agent and if the agents rule list is empty, reduce the current priority to the highest priority of the remaining rules
 - if a circular dependency between rules on different agents is detected, remove all related rules
 - (d) remove all rules with priority lower than the priority of the agent
2. For all agents, for all previously stored rules for the agent: perform from the actions (if they are part of the action):
 - (a) detach
 - (b) compute the attachment component
 - (c) move the attachment component (including collision detection test)

- (d) update attachments
 - (e) update the state
 - (f) swap agents
 - (g) zip agents
 - (h) unzip agents
 - (i) swap-zip agents
3. Send and deliver messages
- (a) for all agents, for all previously stored rules for the agent:
 - do all calculations (in the order they are listed in the rule) and send numerical messages to the post-office
 - send all text messages to the post-office
 - (b) for all messages at the post office: deliver them to their recipients.

3 Program inputs

The program `AgentSystem2D.jar` uses three text files as input.

The file `files/config.txt` is a mandatory file. If it doesn't exist or an error on reading this file occurs, an error display is shown instead of the main window. A similar message appears if any file in the directory `img` does not exist.

The files `files/agents.txt` and `files/rules.txt` are the standard input files for the initial setting. If one of these files doesn't exist or doesn't match the specifications respectively described in sections 3.1 and 3.2 an error dialog occurs at the beginning. The dialog can be closed and necessary files can be chosen (see Section 4.2).

Syntax to launch the program:

```
java [parameters] AgentSystem2D.jar
```

The program can be alternatively started with commandline parameters. If input files are chosen with commandline parameters (arguments `-a` or `-r`), these files must exist instead of the files listed above. Available parameters are:

- `-a <agents_file>` this file is used instead of `files/agents.txt`
- `-r <rules_file>` this file is used instead of `files/rules.txt`
- `-n <number_of_steps>` performs after start the given number of steps with delay (see sections 3.3.2 and 4.1).
- `-nx <number_of_steps>` same as option `n` but additionally the program is stopped after performing the given number of steps.
- `-j <number_of_steps>` performs after start the given number of steps in jump mode (see Section 4.1).

- `-jx <number_of_steps>` same as option `j` but additionally the program is stopped after jumping the given number of steps.
- `-e` performs after start an endless run with delay (see sections 3.3.2 and 4.1).
- `-ex` same as option `e` but additionally the program is stopped if no further rule can be performed. Note that the endless run only stops if none of the preconditions of any rule is fulfilled. It doesn't stop if any rule tries to perform a step, but an error on performing occurs (see Section 4.1).
- `-s <agentsfile>` stores automatically the resulting agents configuration after performing multiple steps. Hence this argument is just allowed in combination with the `-nx`, `-jx` or `-ex` argument.
- `-h` help to commandline arguments

The file `files/config.txt` contains the options of the program (see Section 3.3). The file `files/agents.txt` contains the definition of the initial setting, meaning the initial positions, states, attachments, and counters of all agents (see Section 3.1). The definition of what an (arbitrary) agent will do is contained in the file `files/rules.txt` (see Section 3.2). In all files, one-line-comments can be written by pre-pending the percentage-symbol (%) to the comment (meaning everything in the line after % is ignored). White spaces as well as empty lines, are ignored. Thus these two symbols may not be used as text. Further, for any kind of pattern matching, the asterisk (*) matches every character.

The error handling when starting the program is done in different ways because the program on the one hand can be used by manually starting and navigating it, or on the other hand it can be used by batch processing. To enable both cases with a maximum of usability, the error cases are handled as follows:

1. If the arguments contain syntax errors or irregular combinations of arguments are detected, a command line help with the information about the usage of the program is shown. The GUI does not start in that case.
2. If the program is started without any argument, but there exist errors in the agents or the rules files, then a dialog box appears after starting the graphical user interface with an appropriate message.
3. If the program is started with the arguments `-a` and/or `-r`, but no other parameters, and errors are found in the agents or in the rules files, the same behavior as in Case 2 appears.
4. If the program is started with the arguments `-a` and/or `-r`, but no other parameters, and any of the input files is not found, the GUI starts and an error message about this fact is displayed in a dialog box.
5. If any of the command line arguments `-nx`, `-jx` or `-ex` is used and any of the agents or the rules files is not found, then an error message is shown in the command line to ensure that a batch routine can proceed its work without interruption.

6. If the same arguments as in Case 5 are used and all files are found, but an error is detected in the agents or the rules file, then the program starts the GUI and shows the error to inform the user.

Moreover note that the usage of one of the arguments `-nx`, `-jx`, `-ex` has a side effect. When starting the program with one of those parameters, the **STOP** button is enabled to give the user the possibility to stop the process. But if the **STOP** button is pressed, then the program does not stop processing until the given number of steps are performed!

3.1 Definition of the initial setting

The initial setting is stored in the file `files/agents.txt`. Each line of the file defines one agent by its initial (global) coordinates (mandatory) plus (optional) its state, its attachments and initializations for (some of) its counters. Optionally it is possible to change the size of the universe in the agents file with the parameter `U`.

Initial (global) position `x,y`

The initial position is written as integer `x`-, `y`-coordinates, separated by a comma (see Section 3.3.4 for the range of coordinates).

State `S_____`

The state of an agent consists of exactly 5 characters, written with a leading `S`. Default (if no argument given): `S*****` (no information).

Attachments `A_____`

The attachments of an agent are written as `A` followed by 4 boolean values (0 for not attached, 1 for attached), in the order North, West, East, South. Default (if no argument given): `A0000` (nothing attached)

Counters `C__ _____`

Each agent has 25 integer counters, `C00,...,C24`, which can be set to any 16-bit integer between `-32767` and `32767` (`-32768` is defined as no-number). Default (if no argument given): any not explicitly initialized counter is set to zero.

Limitation of the universe size `U minX,maxX,minY,maxY`

This parameter must be positioned at the beginning of the file. If this parameter is used the option `coordinates_range` of the file `files/config.txt` is overridden for this agents configuration (see Section 3.3.4)!

Small example:

```
U -5,20,20,0 % optional line
0,0 A0010 S1ST-- C0 -5 C1 17
1,0 A0100 S2ND--
3,0 SALONE
5,6
5,7
6,6 C0 1
7,3 C1 -234    C5 43    C7 999
```

3.2 Definition of the rules

The definition of what the agents will do is stored in the file `files/rules.txt`. Each rule definition consists of 4 lines:

1. the name of the rule
2. the priority of the rule
3. the precondition
4. the action

The name is a nonempty string. The priority is either a positive integer (at least 1 and at most 32767), where higher priority 'wins' lower priority, or it is written with the sign \$. In the latter case the rule is a non-priority rule. The precondition defines whether or not an agent may apply the rule. Finally, the action defines what has to be performed, if an agent applies a rule.

3.2.1 The precondition

An agent can apply a rule if it fulfills all the statements included in the precondition of the rule. The precondition may consist of the following parts, where each part is optional. In total, the precondition of a rule cannot be empty. Additionally each "sub-condition" of the precondition can be negated (see the end of this section).

Neighbors N____

The definition of the direct neighbors (North, West, East, South). For each of them

- 0 denotes empty (no agent),
- 1 denotes full (an agent), and
- * denotes don't care.

For example `N01*1` denotes that there must be an empty field to the North of the agent, and the West and South fields need to be full.

Empty field `Edx,dy, EC__,dy, Edx,C__`

An empty field requirement. Written as an E, followed by the relative coordinates of the field that needs to be empty, separated by a comma. Alternatively instead of each value `dx` or `dy` a name of any counter can be inserted, where a counter starts with a C, followed by the two digit number of the counter. If a counter name is written, the value of the counter of the current agent is inserted as this coordinate. Of course it is possible to mix numbers and counters.

Note that the usage of counter values for coordinates may lead to relative coordinates outside the universe. In that case a warning is shown but the rule is performed.

Filled field `Fdx,dy`

A full field requirement. The restrictions and the syntax are the same as in the "Empty field" condition.

Neighboring priorities P_____

The priority of (the applied rule/s) of the direct neighboring positions (North, West, East, South). For each of them:

< denotes that the priority of the corresponding agent needs to be strictly smaller,

= denotes smaller or equal, and

* denotes don't care.

Note that the priority of an empty field is always zero.

Remote smaller priority Ldx,dy

A strictly smaller priority field requirement. The L is followed by the relative coordinates of the agents that is required to have smaller priority, separated by a comma. The usage for counters is the same as in the "Empty field" condition.

Remote smaller or equal priority Qdx,dy

A less or equal priority field requirement. Syntax and usage is analogous to those of the "Remote smaller priority" condition.

Attachments A_____

The attachment states to the direct neighbors (North, West, East, South), where

0 denotes not attached,

1 denotes attached, and

* denotes don't care.

For example **A0**1** denotes that the agent must not be attached North and needs to be attached South (implying that an agent to the South must exist).

State S_____

Every agent has a state, consisting of exactly 5 characters. This state can be required to match a simple pattern, where an asterisk matches any character.

For example **SA***Z** denotes that the state of the agent must start with A and end with Z.

State of a remote agent Tdx,dy,_____

This option is a combination of the "Filled field" and the "State" preconditions. Written as a T, followed by the relative coordinates of the field that is required to be full, and ended by the state that the remote agent must have. The usage of counters is the same as in the "Empty field" condition.

Incoming text messages

M*_____, MN_____, MW_____, ME_____, MS_____

Every agent has four text messages from its direct neighbors (*=any, N=North, W=West, E=East, S=South), consisting of exactly 5 characters. Any of these messages can be required to match a pattern, where the asterisk matches any character but at most four asterisks are allowed.

Numeric comparisons on counters and incoming numeric messages

$<(-)_{----} (-)_{----}, >(-)_{----} (-)_{----}, =(-)_{----} (-)_{----}$

Every agent has 25 counters C00...C24, and additionally $4 * 8 = 32$ numeric messages from its direct neighbors:

- #N01, #N02, #N03, #N04, #N05, #N06, #N07, #N08, #W01, #W02, #W03, #W04, #W05, #W06, #W07, #W08, #E01, #E02, #E03, #E04, #E05, #E06, #E07, #E08, #S01, #S02, #S03, #S04, #S05, #S06, #S07, #S08
- Use #*01, #*02, #*03, #*04, #*05, #*06, #*07, #*08 to check whether a numeric message is received from any direction at the specified channel.

Any of these numeric values can be required to fulfill a comparison with respect to any other such value or to any four digit number, where a negative number contains additionally a leading -.

Note that messages at different channels are independently from each other.

Remote numeric comparisons $V_{dx,dy,C_{--}} (-)_{----}, W_{dx,dy,C_{--}} (-)_{----}$

These options compare the first value with the second value:

V: the first value is strictly smaller than the second;

W: the first value is smaller or equal than the second.

The first numeric value is a counter from another agent at relative coordinates dx, dy . It requires the agent to exist. The second numeric value can be a counter, a numeric message from a neighbor or any four digit number, where a negative number contains additionally a leading -. Recall that every agent has 25 counters and $4 * 8 = 32$ numeric messages from its direct neighbors (enumerated in the previous “Numeric comparison” condition).

Negation !

A negation can be applied to any part of the precondition. The negation means that the result of a “sub-condition” will be negated.

Parenthesis ()

To enable grouping of conditions as well as offering a disjunction between different conditions, parenthesis can be set around a group of conditions. It is also possible to interleave parenthesis (example: $!(N**1*!(A101* L-3,2))$).

3.2.2 The action

The action of a rule defines what happens when an agent applies the rule. It may consist of the following parts, where each part is optional (but in total the action cannot be empty).

Position change $P_{dx,dy}$

Written as a P, followed by the relative coordinates of the field to which the agent moves (separated by a comma). The usage for counters is the same as in the “Empty field” condition.

Attachments A_{----}

Attachment behavior for the four directions North, West, East, South (in this order), with the following possibilities for each direction:

- 0 for detaching (if was attached) before moving, and staying detached afterwards
- 1 for detaching (if was attached) before moving, and attaching afterwards (if possible)
- * for detaching (if was attached) before moving, and attaching afterwards if was attached before (and possible)
- + (only allowed if the agent is attached at the beginning) staying attached all the time. If the agent moves, every (directly or indirectly) attached agent is moved in the same way.

Default (if no argument given): **A****** (try attaching as before in all directions).

State S_____

New state of the agent, consisting of exactly 5 characters. An asterisk denotes that the according character remains unchanged.

Default (if no argument given): **S******* (no change).

Outgoing text messages

M*_____, **MN_____**, **MW_____**, **ME_____**, **MS_____**

Messages that are sent to neighbors (*=all, N=North, W=West, E=East, S=South), consisting of exactly 5 characters. At most four asterisks are allowed (for example **MSRUN**** is allowed, but not **MS*******).

Assign calculation results to counters and outgoing numerical messages

C__ _ _ _ _ _, **#__ _ _ _ _ _**
C__ _ dx,dy,C__ _ _ _ _, **#__ _ dx,dy,C__ _ _ _ _**
C__ _ C__,C__,C__ _ _ _ _, **#__ _ C__,C__,C__ _ _ _ _**

Every calculation action starts with a position to write the result to (counter or outgoing message), followed by the operation to be performed, and two (readable) values on which the operation is performed.

Possible operations are + (add), - (subtract), * (multiply), / (divide), M (modulo), A (maximum), and I (minimum).

As values for an operation, either four-digit-numbers, or internal counters, or one remote counter, or incoming numerical messages can be used. Every agent has 25 counters C00 ... C24 which can be read and written. Further, it has 32 (incoming) numeric messages from its direct neighbors, which it can only read, and 32 numeric (outgoing) messages to its direct neighbors, which it can only write.

The remote counter is defined by first indicating the coordinates (**dx,dy** or **C__,C__**) of the module and then the counter to be used. Note that the coordinates are restricted to three digits (-999 to 999), and that it is required that there exists an agent at the indicated location.

Available message senders/recievers:

#N01, #N02, #N03, #N04, #N05, #N06, #N07, #N08,
#W01, #W02, #W03, #W04, #W05, #W06, #W07, #W08,
#E01, #E02, #E03, #E04, #E05, #E06, #E07, #E08,
#S01, #S02, #S03, #S04, #S05, #S06, #S07, #S08

Use **#*01, #*02, #*03, #*04, #*05, #*06, #*07, #*08** to send a numeric message to

all neighbors at the specified channel. Note that a calculation with undefined result (for example division or modulo by 0) leads to an error. If such an error occurs no value will be sent to the receiver of the result.

Swap *xN, xW, xE, xS*

Written as a **X**, followed by the desired swap neighbor.

Zip *zN, zW, zE, zS*

Written as a **Z**, followed by the desired neighbor direction to zip, both agents must be uncompressed.

Unzip *zNS_____, zWS_____, zES_____, zSS_____*

Written as a **z**, followed by the desired neighbor direction to unzip and the new state of the uncompressed agent with the same format as **state**.

Swap-zip *xN, xW, xE, xS*

Written as a **x**, followed by the desired neighbor direction to swap-zip.

3.2.3 Include rules

To enable combinations of different rule sets for one agent configuration, it is possible to include other rule files. The syntax for an include is

```
include:<file_path>:<Priority_Offset>
```

where include statements must be added at the beginning of the file. If just a file name is given the file will be searched in the **files** directory of the program. If a relative path definition is given in the include statement, the path separator sign must be chosen from the user according to the operating system. Note that spaces are not allowed in the path according to Section 3. It is possible to insert as many include statements as desired and it is also possible to cascade include statements. If include statements are cascaded priority offsets will be accumulated.

If an error in any of the included files occurs, an error message is shown in the specified line of the main rules file. Additionally an error message occurs if the include statement is behind the init block (see Section 3.2.4) or any defined rule (see Section 4.2.2).

For example `include:foreign_rules.txt:100` means that all rules from the file `foreign_rules.txt` are included in the ruleset, where the priority of each rule defined in `foreign_rules.txt` will be incremented by 100.

Note that the usage of include statements can simplify the definition of complex rulesets. But this can lead to some problems, for example the usage of the same state name in different files or the usage of same counters in different files, what definitively would lead to undesired behavior.

3.2.4 State colors

A helpful function for visualization is the color of agents. The standard color for attached and detached agents can be defined in the file `files/config.txt` file or alternatively in the options menu. But in the visualization it is also essential which state an agent at a specified step has. Of course it is possible to check the state in the tooltip function

(see Section 3.3.1). But to get an overview about the state of all agents this would need a lot of time. Therefore a color can be assigned to each state of the agents.

Assigning colors to states can be made in the current rules file. In order to do so an init block is inserted in the rules file. The init block needs to be located after the include lines and before the rules. In the init block each line assigns a color to a state. Assigning a color to a state can be made in two different ways. The first possibility is to write the name of the state followed by the name of the color, where possible colors are BLACK, BLUE, CYAN, GRAY, MAGENTA, ORANGE, PINK, RED, YELLOW. The names are listed additionally in the help menu (described in Section 4.13). An alternative option is to write the specified state followed by a # and the hexadecimal RGB color code.

The following is an example of a rules file:

```
include:foreign_rules.txt:50

_init_start_
SRIGHT CYAN
SDOWN_ #ff0000 %means the same as SDOWN_ RED
_init_end_

Go Down
100
SDOWN_ >C000000
P0,-1 C00-C000001 A++++
```

Note that color names are written with capital letters and that the hexadecimal code uses lower case letters. Further in the hexadecimal code the first two digits stand for the red color, the middle two digits for the green color and the last two digits for the yellow color. If an agent gets a state where no color definition is made in the rules file, the agent gets the standard color as defined in the options (see Section 3.3.1). If any error on reading the state colors is found, an error message is written in the specified line (see Section 4.2.2).

3.3 Program options

The definition of available options is stored in the file `files/config.txt`. All options that are available can be set in the file or in the Options menu(see Section 4.12). The syntax of options is always the same: `[OPTION_NAME]=[OPTION_VALUE]`.

3.3.1 Layout options

Available layout options are color options, font options [and the grid option].

1. Color options

The color option value has the syntax: `[R] , [G] , [B]` whereas R means the red color value, G means the green color value and B means the blue color value. The values for R, G and B can be chosen in the range 0-255.

Example:

`color_bg=255,255,255`

The following colors can be modified:

Background color Color of the universe background.

- Option name: `color_bg`
- Standard value: `[255,255,255]`: (White)

Grid color Color of the universe grid.

- Option name: `color_grid`
- Standard value: `[0,0,0]`: (Black)

Color of not attached agent Color of agents who are nowhere attached, and for attachment arms which aren't attached.

- Option name: `color_agent_not_att`
- Standard value: `[0,255,0]`: (Green)

Color of attached Agent Color of agents who are anywhere attached.

- Option name: `color_agent_att`
- Standard value: `[0,124,0]`: (Dark Green)

Color of attached arm Color of attachment arms which are attached.

- Option name: `color_attached_arm`
- Standard value: `[255,0,0]`: (Red)

Standard text color Color of the iteration number in the universe.

- Option name: `color_text`
- Standard value: `[0,0,0]`: (Black)

Error text color Color of the error message text in the universe.

- Option name: `color_error_text`
- Standard value: `[255,0,0]`: (Red)

Color of tooltip background Color of the tooltips background.

- Option name: `color_tooltip`
- Standard value: `[255,255,0]`: (Yellow)

Color of tooltip text Color of the tooltips text.

- Option name: `color_tooltip_text`
- Standard value: `[0,0,0]`: (Black)

2. Font options

Font options are font type and font size for the agents, rules, actions, positions, and error panels.

Font type Type of the font in the agents, rules, actions, positions and error panel.

- Option name: `current_font`

- Standard value: `[SansSerif]`.

Font size Size of the font. Available range is from 6 to 32 dots per inch.

- Option name: `current_font_size`
- Standard value: `[12]`.

3.3.2 Running options

These options are used for better usability and performance.

1. History options

The program has a history option to provide reverse steps.

- Option name: `history`
- Available values:
 - `all`: All iterations are stored in the history. Take into account, though, that the system is limited to a maximal stored number of history entries (see Section 5).
 - `last20`: The last 20 iterations are stored in the history (at most 20 back steps are possible).
 - `off`: No history entries are stored. Thus no back steps are possible.
- Standard value: `all`

2. Logging options

For details to the log files see Section 6.

- Option name: `logging`
- Available values:
 - `all`: All log files are written.
 - `error`: Just the error.log file is written.
 - `off`: No log files are written.
- Standard value: `all`

3. Delay options

Delay between multiple steps (10 Steps Back, Next 10 Steps, Next n steps, Run forever; see Section 4).

- Option name: `timeout`
- Available values:
 - The timeout in milliseconds.
- Standard value: `500`

4. Rules warning option

This option enables or disables the warning for duplicate rule names (see Section 4.2.2).

- Option name: `rules_warning`
- Available values:
 - `on`: Rules warnings are shown in rules panel.
 - `off`: Rules warnings are not shown in rules panel.
- Standard value: `on`

3.3.3 Behavior options

These options are concerning the actions in each iteration.

1. Agent options

This option takes effect on applying rules on each agent. In the standard way all rules which can be applied to one agent are performed in the order they are listed in the rules file `files/rules.txt`. This option allows to change this in the following way: rules which don't have an action including a position change can be applied before rules which imply a position change. This option has an effect on each agent, but does not influence the global order of the rules performance: rules are always applied to the agents in the order agents are listed in `files/agents.txt`.

- Option name: `position_changes_last`
- Available values:
 - `on`: All stored rules of an agent implying a position change are performed after all rules without position changes.
 - `off`: All rules are applied in the order listed in the rules file.
- Standard value: `off`

2. Message delivering option

In the standard way messages are sent and delivered after performing all actions (described in [?]). With this option this can be changed so that after checking all rules all messages are sent and delivered and after that all actions are performed.

- Option name: `perform_messages_first_move_last`
- Available values:
 - `true`: All messages are sent before performing all actions to all agents.
 - `false`: All messages are sent after performing all actions to all agents.
- Standard value: `false`

3.3.4 Limitations

Limitations to the universe range can be set. Attention: This option does not have any effect if the parameter `U` is used in the agents file (see Section 3.1).

1. Coordinates range

This option has the syntax: `min_X`, `max_X`, `min_Y`, `max_Y`, where the values must be integers.

- Option name: `coordinates_range`
- Example:
 - `coordinates_range=-5,70,-5,27`
- Standard value: `-5,70,-5,27`

4 Program window

The main window consists of a menu and a tabbed panel with five tabs. The Short key: `'Ctrl'+‘T’` allows to navigate to the next tab, and the Short key: `'Ctrl'+‘Shift’+‘T’` sends to the previous tab.

4.1 Universe

This panel visualizes all read agents in the universe. In the upper side of the universe panel the iteration is shown and if an error occurs a message is shown, too. If more than one error appears in one iteration a + sign is added in front of the error field. Clicking on the + sign opens a window with all error messages appeared in this iteration, where the first error is on the first position in the window. If more than 10 errors appears, an additional line

```
< < < < Further errors occurred! For details see error.log! > > > >
```

is shown in the last line of this window. Furthermore, the number of `TotalErrors` and `TotalWarnings` is shown in the upper side of the universe panel.

Several navigation buttons are available (back steps are only available if the history function is on, see Section 3.3.3). For multiple steps buttons (10 Steps Back, Next 10 Steps, Next *n* steps, Run forever) a timeout between the steps is passed (see Section 3.3). Alternatively for each button at least one short key is available.

Universe panel functions:

Start Before iterations can be performed this button has to be clicked.

Short keys: `'s'`, `'S'`.

Reset A click on this button resets the configuration to its initial values. This means that agents and rules are read once again and all history entries are removed.

Short key: `'F2'`.

Back to start A click on this button resets the universe to its initial position (Iteration0)

Short key: `'Home'`.

10 Steps Back With this button 10 reverse steps are performed.

Short keys: `'PgUp'`, `'CursorUp'`.

Back Just to go one step back.

Short keys: `'CursorLeft'`, `'Backspace'`.

Next Step Go one step forward.

Short keys: `'CursorRight'`, `'Enter'`.

Next 10 Steps Go 10 steps forward. If no further rule can be performed a message is shown.

Short keys: 'PgDown', 'CursorDown'.

Next n steps A click on the button opens a dialog where the number of steps to move can be introduced. The dialog has a text field to insert the desired number of steps, a button **Next** to perform the chosen number of steps and a button **Cancel** to close the dialog without performing steps. After clicking on **Next** the steps are performed. If no further rule can be performed a message is shown.

Jump n A click on this button performs a chosen number of steps without visualization and without timeout (see Section 3.3.2, item 3). After clicking on the button a dialog appears. The dialog has a text field to insert the desired number of steps, a button **Jump** to perform the chosen number of steps and a button **Cancel** to close the dialog without performing steps. After clicking on **Jump** the steps are performed. To update the user about the remaining steps a message **Wait n steps** is shown on the top of the universe panel (where n indicates the remaining number of steps). If no further rule can be performed a message is shown.

Short keys: 'j', 'J'.

Run forever Performs endless forward steps. If no further rule can be performed a message is shown.

Short key: 'End'.

Stop Stops executing steps (10 Steps Back, Next 10 Steps, Next n Steps, Run forever).

Short keys: 'Esc', 'Pause'.

Tooltip function A helpful function to check the current values of each agent is the Tooltip function. A click on an agent shows a tooltip window.

The information displayed on this windows is:

always shown: (on each agent)

Id

Global position

Attachments

Counter values

Current priority

optionally shown (if information is stored)

State

Text messages

Numeric messages

Id zip

State zip

Zip counter values

Speed This function is only available with the

Short key: `'Ctrl'`.

If multiple steps are performed a timeout between steps is passed. To avoid this, the timeout can be disabled. But this can be done in another way, too: while multiple steps are performed the Short key `'Ctrl'` can be hold and the timeout while holding the key is 0.

4.2 Agents and rules

The tab consists of two text panels. On the left side the current agents file is shown and on the right side the current rules file is shown. To enable changing focus between the two text panels if editable, the mouse or alternatively the short key: `'Ctrl'+ 'P'` can be used. If just one text panel is editable the focus grabs to that panel.

The font type and size of the agents and rules panels as well as of the actions, positions and error panel can be modified in the options menu (see Section 3.3.1).

Short key: `'Ctrl'+ '+'` increments the font size.

Short key: `'Ctrl'+ '-'` decrements the font size.

4.2.1 Agents panel

The panel has on the upper side buttons to create, open and modify agents files. The text panel contains the current agents file. Lines with successfully read agents (valid initial setting) are shown in black, comment lines are shown in gray and lines with errors are shown in red. An agent definition is valid if and only if all settings in the specified lines fulfill the specification. For further information see Section 3.1.

Possible error messages:

- `"% AGENT already exists! Ignored!"` occurs if an agent on this position is already stored.
- `"% Position OUT_OF_RANGE! Ignored!"` is shown if a position value of an agent isn't in the coordinate range of the universe (see also Sections 5 and 3.1).
- `"% % Counter OUT_OF_RANGE! Ignored!"` occurs if any counter value inserted is out of the specified range (see Section 3.1).
- `"% ATTACHMENTS ERROR: not symmetrical! Attachment ignored!"` is displayed if an attachment definition between two agents exists that is not symmetrical.
- The message `"% ATTACHMENTS ERROR: to attach agent doesn't exist! Attachment ignored!"` shows that an attachment definition refers to a non existent agent.
- `"% SYNTAX ERROR OR IRREGULAR LIMITATIONS! Ignored!"` occurs if a syntax error on an agent definition or the universe limitation is found, or if the universe limitation is not at the beginning of the file.

New To create a new agents file. After clicking on the button the text panel will be cleared. Agents can be defined and saved. To interrupt the process click on Reset. To store the new setting click on Save.
Short key: 'Ctrl'+ 'N'.

Open Opens an agents setting file. A dialog to save all log files appears before opening the file (see button Save) because when opening the agents file the log files are overwritten.
Short key: 'Ctrl'+ 'O'.

Modify With this button the current agents setting can be modified.
Short key: 'Ctrl'+ 'M'.

Reset If a new agents setting is created or an old one is modified, this button discards the new setting (the last read agents become active).
Short key: 'Ctrl'+ 'R'.

Save To save an existing agents setting. If the button is clicked all log files are overwritten and the new agents setting is read immediately. No dialog appears to store existing log files!
Short key: 'Ctrl'+ 'S'.

Save as To save a new agents setting or a modified one. If this button is clicked all log files are overwritten and the new agents setting is read immediately. A dialog to save all log files appears before loading the new setting.
Short key: 'Ctrl'+ 'W'.

Undo If the agents file is modified in the agents panel this function undoes the last modification. If modifications in the rules panel are done too, the current focus decides on which panel the function is performed. This can be done via the menu or the
Short key: 'Ctrl'+ 'Z'.

Redo If the agents file is modified in the agents panel and undo steps are performed this function redoes the last undone operation. If modifications in the rules panel are done too, the current focus decides on which panel the function is performed. Redo can be done via the menu or the
Short key: 'Ctrl'+ 'Y'.

Find If the agents file is editable in the agents panel a find dialog is opened with the
Short key: 'Ctrl'+ 'F'.
Available options in this dialog are search direction, search scope, wrap search and case sensitive search. After clicking **Find** the dialog will be closed and eventually the found text will be marked. The opened dialog can be canceled with the short key 'Esc' and the find process can be started with the short key 'Enter', too. If the rules file is also editable, the current focus determines on which panel the find process takes effect.

Find next The next occurrence of an already searched text can be found with the Short key 'F3'.
Search options are the same as for "Find".

Replace A replace dialog for the agents panel can be opened with the Short key 'Ctrl'+ 'H'.
Finding options are the same as in the "Find" dialog. The dialog is opened until the **Close** button or the short key 'Esc' is pressed. The button **Find** or the short key 'Enter' searches the next occurrence in the text; the button **Replace** replaces the first occurrence of found text after the current cursor position; the button **Replace and Find** replaces the first occurrence and searches the next occurrence of the searched text. Finally the button **Replace All** replaces all occurrences of the searched text. If the rules file is also editable, the current focus determines on which panel the find and replace process takes effect.

4.2.2 Rules panel

The panel has on the upper side buttons to create, open and modify rules files. The text panel contains the current rules. Rules with successfully read priority, preconditions and actions are shown in black, comment lines are shown in gray and rules with errors are shown in red. A rule definition is valid if and only if all settings in the specified lines fulfill the specifications. For further information see Section 3.2.

Possible error messages:

- If an error on reading an include file is found the message "% Error: Specified file not found or error in nested file! Please check it!" is displayed. Possible causes are: the file doesn't exist; the path to the file contains spaces, which is not allowed; some rule in some include file (they can be nested) contains irregular expressions or some file contains an irregular include statement. It would be possible that a negative priority offset causes problems, too.
- If an include statement is found at an irregular position the message "% Error: Include statement only allowed at the beginning of the file!" is shown.
- If an irregular syntax in an include line or in any of the agent colors is found the message "% Syntax error!" is shown.
- If the priority isn't an integer > 0 the message "% Wrong syntax! Rule ignored!" is displayed in the second line of the rule.
- "% Irregular Precondition! Rule ignored!" shows that the precondition contains either a syntax error or that an **Empty Field**, **Filled Field**, **Smaller Priority**, **Smaller or Equal Priority** or **State of a remote agent** refers to a position outside the universe.
- "% Irregular Precondition! Priority conditions and NonPriority conditions are mixed! Rule ignored!" is shown if a NOT condition (!) contains priority and non priority conditions (see Section 3.2.1).

- "% Irregular Action! Rule ignored!" occurs if a syntax error in the action line is found or if a `Position change` would lead to move outside the universe.

Possible warnings (can be disabled, see Section 3.3.2):

- "% WARNING: DUPLICATE RULE NAME!" shows that a rule with the same name has already been defined.

In the rules panel, the following options can also be found:

New To create a new rules file. After clicking on the button the text panel will be cleared. Rules can be defined and saved. To interrupt the process click on Reset. To store the new definitions click on Save.

Short key: 'Ctrl'+ 'Shift'+ 'N'.

Open Opens a rules file. A dialog to save all log files appears before opening the new definition (see button Save) because when the rules file is opened, the log files are overwritten.

Short key: 'Ctrl'+ 'Shift'+ 'O'.

Modify With this button the current rules file can be modified.

Short key: 'Ctrl'+ 'Shift'+ 'M'.

Reset If a new rules definition is created or an old one is modified this option discards the new definition, and the last read rules become active.

Short key: 'Ctrl'+ 'Shift'+ 'R'.

Save To save an existing rules definition or a modified one. If this button is clicked all log files are overwritten and the new rules file is read immediately. No dialog appears to store existing log files!

Short key: 'Ctrl'+ 'Shift'+ 'S'.

Save as To save a new rules definition or a modified one. If this button is clicked all log files are overwritten and the new rules file is read immediately. A dialog to save all log files appears before loading the new definition.

Short key: 'Ctrl'+ 'Shift'+ 'W'.

Undo If the rules file is modified in the rules panel this function undoes the last modification. If modifications in the agents panel are done, too, the current focus decides on which panel the function is performed. Undo is possible via the menu or the

Short key: 'Ctrl'+ 'Z'.

Redo If the rules file is modified in the rules panel and undo steps are performed, this function redoes the last undone operation. If modifications in the agents panel are done too, the current focus decides on which panel the function is performed. Redo can be done via the menu or the

Short key: 'Ctrl'+ 'Y'.

Find If the rules file is editable in the rules panel a find dialog is opened with the Short key: `'Ctrl'+‘F’`. Available options in this dialog are search direction, search scope, wrap search and case sensitive search. After clicking **Find** the dialog will be closed and eventually the found text will be marked. The opened dialog can be canceled with the short key `'Esc'` and the finding process can be started with the short key `'Enter'`, too. If the agents file is also editable, the current focus determines on which panel the find process takes effect.

Find next The next occurrence of an already searched text can be found with the Short key `'F3'`. Search options are the same as for “Find”.

Replace A replace dialog for the rules panel can be opened with the Short key `'Ctrl'+‘H’`. Options are the same as for “Find”. The dialog is opened until the **Close** button or the short key `'Esc'` is pressed. The button **Find** or the short key `'Enter'` search for the next occurrence in the text; the button **Replace** replaces the first occurrence of found text after the current cursor position; the button **Replace and Find** replaces the first occurrence and searches for the next occurrence of the search text. Finally the button **Replace All** replaces all occurrences of searched text. If the agents file is also editable, the current focus determines on which panel the find and replace process takes effect.

4.3 actions.log

This panel shows the log file `files/actions.log` (see Section 6). The panel contains in its upper side a combo box to select the iterations to be visualized. If an iteration range is performed more than once a message `later iterations:` is inserted to the text panel.

Let us examine an example. First the iterations 1-48 are performed. Then 20 back steps are done, and finally the iterations 29-42 are performed. If now on the actions panel the set `Iteration 40-49` is selected, the following lines are shown:

```
% Iteration 40
  Agent 1 (28/9): DOWN
% Iteration 41
  Agent 1 (28/8): RIGHT
% Iteration 42
  Agent 1 (29/8): RIGHT
% Iteration 43
  Agent 1 (30/8): RIGHT
% Iteration 44
  Agent 1 (31/8): RIGHT
% Iteration 45
% Iteration 46
  Agent 1 (32/8): RIGHT
% Iteration 47
```

```

    Agent 1 (33/8): RIGHT
% Iteration 48
    Agent 1 (34/8): RIGHT

```

later iterations:

```

% Iteration 40
    Agent 1 (28/9): DOWN
% Iteration 41
    Agent 1 (28/8): RIGHT
% Iteration 42
    Agent 1 (29/8): RIGHT

```

Combo box iteration n-m Selects the shown iteration range.

Save as Button to store the `files/actions.log` file to another file.
Short key: ‘Ctrl’+‘S’.

4.4 positions.log

This panel shows the log file `files/positions.log`, which stores the situation of all agents at each iteration (for more details on the content of this file, see Section 6). The handling is analogous to the one described in Section 4.3.

4.5 error.log

This panel shows the log file `files/error.log`, which stores all warning and error messages (see Section 6 for more details on the content of this file). The handling is analogous to the one described in Section 4.3.

4.6 Agents generator

The agents generator is dedicated to create and modify sets of agents. This tab shows on the left side a text panel, which shows the agents file, similar to the text panel for the agents tab (compare with Section 4.2.1). On the right side of the tab a graphical representation of the loaded set of agents is shown. When the program starts, the agents generator loads the same file as the universe. Nevertheless, the agents generator is independent from the current universe. This means that the modification of agent sets does not influence the current setting on the “Universe” tab. Furthermore the text in the left panel and the graphical representation to its right are equivalent, i.e. a text/graphical representations of the same file. An exception to this fact happens if the content of the text panel is modified by hand; for details see the “Refresh” option of the Agent panel in Section 4.6.1.

4.6.1 Agents panel

The left part of the agents generator contains the text panel, which has the same functionality as the agents panel on the “Agents and rules” tab. In this case, though, the

agents are shown on the universe of the creator panel instead of the current universe. Moreover the dialog to save log files does not appear in the agents panel of the agents creator if, for example, agents are saved. The “Undo” and “Redo” functions work a little bit differently from the functions described in Section 4.2.1 (see more details in Section 4.6.2). Additionally the following two options are included on the agents panel of the agents creator:

Save + init On clicking this button the current agents setting will be stored and shown in the creator universe.
Short key: ‘Ctrl’+‘I’.

Refresh If the agents panel is modified, the graphical presentation needs to be updated through this refresh option. On refreshing the text panel, syntax errors are checked, too.
Short key ‘F5’.

4.6.2 Universe panel

The universe panel can be used to create and modify sets of agents in a visual way, by using the mouse of the computer. Note that all functions to change the agents set in the universe panel are only available if the agents file in the agents panel is editable. All modifications on the agents are refreshed immediately in the text panel to the left of the universe panel. Contrary to this, any modification in the agents panel is only updated in the universe panel when using the “Refresh” function.

Available functions on the universe panel are “Add agents”, “Remove agents”, “Select agents and set their state” and “Change the values of counters on selected agents” as well as the “Undo” and the “Redo” function. Note that most of the actions are done by using the left mouse button (if the user is right handed, else the right mouse button)! For the “Add”, “Remove” and “Select” functions the following modes are possible:

MODE: Single agent



If this option is selected, all actions are applied on just one single agent (for example, an agent is added to the universe).

MODE: Line of agents



If the “Line mode” is chosen, an “Add”, “Remove” or “Select” action is applied on all agents which are within the drawn line. If, for example, the “Add” action and the “Line mode” are selected, then a mouse click starts creating a line of modules. When the mouse is dragged (by holding the left mouse button), a line of agents is created, from the middle of the starting point to the current position of the mouse pointer. When the left mouse button is released, all agents are added to the universe.

MODE: Rectangle of agents



The “Rectangle mode” works similar to the “Line mode”: on dragging the mouse left button, a blue rectangle is drawn. On releasing the left mouse button, all cells enclosed in the drawn rectangle contain an agent which is added to the universe.

In the following the three actions “Add”, “Remove” and “Select” agents are described in detail:

ACTION: Add agent



If this option is selected, agents are added to the current universe. To show which mode is selected when adding agents, a blue line in the line mode and a blue rectangle in the rectangle mode is drawn when dragging the mouse. Note that adding agents does not change any state or counter value of any previously added agent! After adding agents, the cursor in the text panel is set to the agent on the end point of the line or rectangle. Furthermore, several options for attachments on adding agents are available (see the options below).

ACTION: Remove agents



This is the reverse of the adding action, and works similarly. Either a single agent, a line of agents, or a rectangle of agents can be removed from the universe. The line and the rectangle on dragging the mouse is drawn in red color.

ACTION: Select agents



This option allows to choose a set of agents and further to modify their state and the value of their counters(see the actions “Change state” and “Change counters”). When dragging the mouse, a gray line is drawn if the line mode is chosen, and a gray rectangle if the rectangle mode is selected. Agents are selected by using the left mouse button and unselected by using the right mouse button (if the user is right handed, else the inverse). Selected attached agents are drawn in red color, selected detached agents in orange. Furthermore every selected agent is surrounded by a gray rectangle. Additionally it is possible to select groups of agents by holding the Short key ‘Ctrl’!. If the ‘Ctrl’ key is hold, and new agents are selected, the old selection is extended by the new selection, otherwise the old selection is replaced by the new selection. Note that is also possible to change the “mode” of selection without losing the current selection, because the selection is only removed if a new action is performed or the agents are stored or reset.

When adding agents to the universe, the new agents can be either detached from all their neighbors, or they can be attached only to the currently added agents. Finally, they can also be attached to all existing neighbors. Therefore, adding agents can overwrite attachments of existing agents!

OPTION: Detach all when adding



If the “Add” action and the “Detach” option are chosen, all new agents are detached from all their neighbors, meaning that attachments of previously existing agents are removed.

OPTION: Attach along cursor when adding



This option attaches all currently added agents to each other, but not with previously existent agents. Recall, though, that if new agents are added in the same

positions where previously added agents were already present, no attachments are removed, but new attachments can be added.

OPTION: Attach all when adding

All currently added agents are attached to all existing neighbors, including to previously added ones. Note that attachments of previously added agents are overwritten, if new agents are added on the same positions by using this option.

ACTION: Change state

The state of all selected agents can be modified by using this action. A dialog box to insert the new state is shown. The Short key: ‘Enter’ sets the new state if it is possible (equivalent to the OK button) and closes the dialog, and the Short key: ‘Esc’ closes the dialog without changing the state of any agent.

ACTION: Change counters

The values of any counters of all selected agents can be changed by this action. A click on this button opens a dialog. Either all counters or a selection of counters can be selected, and their new values can be set. The new counter value can be inserted in a text field. The correctness of the inserted values is checked before changing the value. The Short key: ‘Enter’ sets the new counter values if it is possible (equivalent to the OK button), and the Short key: ‘Esc’ closes the dialog without changing any value.

ACTION: Undo

This option works for all actions which are performed on the creator universe (“Add” or “Remove” agents, “Change state” or “Change counters”). The “Undo” operation of the text panel to the left of the universe is only stored if the “Refresh” function of the text panel is applied. For example, if a set of agents is loaded, then an agent is removed from in the text panel and finally the key ‘F5’ is pressed, then the “Undo” operation works as desired. If changes are performed in the text panel without “Refreshing” the universe, and then an action is done using the mouse, then all modifications done in the text panel (previous to the mouse action) are lost!

Short key: ‘Ctrl’+‘Z’

ACTION: Redo

The “Redo” operation works if any “Undo” operation is performed prior. Short key: ‘Ctrl’+‘Y’

Zoom The size of the grid and the agents can be increased by pressing the Short key: ‘Ctrl’+‘+’ and decreased by pressing the Short key: ‘Ctrl’+‘-’.

The colors used to display agents are the same as the ones used in the original universe. This means that the “standard” colors for agents are green, and dark green for attached

agents (if the colors of the agents are not changed in the options of the program). If a rules file is loaded containing color definitions for states, then the colors defined in the rules file are used. An exception to this is the selection of agents, which always uses the colors orange (for unattached agents) and red (for attached agents).

4.7 Exit menu

To end the program.

4.8 Universe menu

This menu contains all functions of the universe panel. For details see Section 4.1.

4.9 Agents menu

This menu contains all agent functions of the panel “Agents and Rules”. More details can be found in Section 4.2.1.

4.10 Rules menu

This menu contains all rules functions of the panel “Agents and Rules”. More details can be found in Section 4.2.2.

4.11 Agents generator menu

See Section 4.6.

4.12 Options menu

After clicking on this menu a dialog with option values is opened. The values which can be set are described in Section 3.3. In the dialog the options can be modified and stored. Modifications can be discarded and options can also be reset to their standard values.

4.12.1 Layout options

Available layout options, color options [and the grid option]. In contrast to the color options in the file, in the panel the colors can be chosen from a color chooser dialog without knowledge about RGB-values. See Section 3.3.1.

4.12.2 Running options

Described in Section 3.3.2.

4.12.3 Behavior options

Described in Section 3.3.3.

4.12.4 Limitations

Limitations to the universe range are available. See Section 3.3.4 for details.

Attention:

- If new limitations are stored, the current universe state is invalid. Thus all agents and rules must be read once again and all log files are overwritten. To avoid data loss a dialog window to save all log files is shown.
- This option does not have any effect if the parameter `U` is used in the agents definition (see Section 3.1).

4.13 Help menu

This menu contains an entry **Shortkeys** which gives an overview of available short keys. “Color names for states” lists all available names, which can be used in the rules file (refer to Section 3.2.4). Finally it contains an entry **About**.

5 Limitations

The system has the following restrictions.

- The program needs a minimal display resolution of 800×600 pixels. If the display width is less than or equal to 1024 the buttons of the universe panel are arranged in two rows. If the width is greater than 1024 the buttons are arranged in one row.
- The duration of one step in the program differs on different modes. The fastest mode is the jump-mode because only the last step is shown. Multiple steps forward (Next 10 steps, Next n steps, Run forever) in speed-mode (see Section 4.1) are almost as fast as jump steps. Single steps forward are a little bit slower than multiple steps. Backwards steps are the slowest steps, because all agents are displayed at each step.
- If the option History (Section 3.3.2) is set to **all** the application is limited because of limited memory resources. If resources are exhausted an **OutOfMemory** error occurs. In this case an error dialog is shown. On the dialog the button **End Program** exists to end the program.

On Windows XP with a standard stack size of 64MB this limit is reached with approximately 35000 agents stored in the history. This can be reached for instance with 200 agents in 175 iterations if all agents act at each step.

The size of the stack can be set manually with a JVM-Flag `-Xmx[n]m` whereas the value for n is the stack size in MB.

6 Program outputs

In addition to the graphical representation, the program generates three output files:

- `files/positions.log`

- files/actions.log
- files/error.log.

For better readability of these files, the agents are given numerical id's according to the order they appear in the file `files/agents.txt`, starting with number 1.

The file `files/positions.log` contains a complete state history of the agents. Starting from the initial setting, and then after every (global) calculation step, all agents are listed in the order of their numerical id's. For each agent one line is printed, containing its numerical id, its current global position, its attachments, its state, the values of all its counters, and all current incoming messages (numerical and text).

Example with numerical messages:

```
% Iteration 4
1: 0/5 A0010 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
2: 1/6 A0011 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
3: 2/6 A0101 SFORWD C0 0 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
4: 1/5 A1110 SFORWD C0 1 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
5: 2/5 A1110 SUNDEF C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 #N2 #W2
6: 3/5 A0101 SUNDEF C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
7: 3/4 A1010 SUNDEF C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
8: 4/4 A0100 SUNDEF C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
```

Example with text messages:

```
% Iteration 16
1: 0/5 A0010 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
2: 1/6 A0011 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
3: 2/6 A0101 SFORWD C0 0 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 MSWRONG
4: 1/5 A1110 SFORWD C0 1 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 MEOK___
5: 2/5 A1110 SFINIS C0 2 C1 3 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
6: 3/5 A0101 SFINIS C0 0 C1 4 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
7: 3/4 A1010 SFINIS C0 4 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
8: 4/4 A0100 SFINIS C0 0 C1 5 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0

% Iteration 17
1: 0/5 A0010 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 MEOK___
2: 1/6 A0011 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 MEWRONG MSWRONG
3: 2/6 A0101 SFINIS C0 0 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
4: 1/5 A1110 SFINIS C0 1 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
5: 2/5 A1110 SFINIS C0 2 C1 3 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
6: 3/5 A0101 SFINIS C0 0 C1 4 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
7: 3/4 A1010 SFINIS C0 4 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
8: 4/4 A0100 SFINIS C0 0 C1 5 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
```

The file `files/actions.log` contains a complete history of which agent applied which rules, and when. In every global calculation step, the printing is done after the decision is taken of which agents will perform which rules, but before the rules are applied. For every agent and every applied rule one line is printed, containing the numerical id of the agent, its current position, and the name of the rule to be applied.

The following example shows an `actions.log` file. In iteration 3 agent 4 applies two rules. The possibility of performing multiple rules on one agent is a very powerful tool, which allows simplifying the rules set. It must be handled very carefully!

```

% Iteration 1
  Agent 1 (0/5): MAKE_LMOST_TRY
  Agent 2 (1/6): MAKE_LMOST_TRY
% Iteration 2
  Agent 1 (0/5): SEND_LMOST
  Agent 2 (1/6): SEND_LMOST
% Iteration 3
  Agent 3 (2/6): STORE_FW_MSG_FROM_WEST
  Agent 4 (1/5): STORE_FW_MSG_FROM_NORTH
  Agent 4 (1/5): STORE_FW_MSG_FROM_WEST
% Iteration 4
  Agent 3 (2/6): FW_MSG_IF_W
  Agent 4 (1/5): FW_MSG_IF_N_AND_W_C1__
% Iteration 5
  Agent 5 (2/5): STORE_FW_MSG_FROM_NORTH
  Agent 5 (2/5): STORE_FW_MSG_FROM_WEST
% Iteration 6
  Agent 5 (2/5): FW_MSG_IF_N_AND_W_C1__
% Iteration 7
  Agent 6 (3/5): STORE_FW_MSG_FROM_WEST
% Iteration 8
  Agent 6 (3/5): FW_MSG_IF_W
% Iteration 9
  Agent 7 (3/4): STORE_FW_MSG_FROM_NORTH
% Iteration 10
  Agent 7 (3/4): FW_MSG_IF_N

```

The file `files/error.log` contains a complete history of occurred errors. In the following example warning and error messages are shown. Each warning starts with `warning(` and ends with a `)`; error messages start with the string `error(` and end with a `)`.

```

% Iteration 122
warning(Perform Rule: COLLISION IN UNIVERSE)
  error(Consistency Error B:
    Universe[32,15] = 16
    Agent[13] = 32,15)
% Iteration 123
% Iteration 124
warning(Internal Error! Agent on 32,15 mark Attached to agent 10, but isn't there!)
warning(Internal Error! Agent on 32,15 mark Attached to agent 10, but isn't there!)

```

7 Requirements

The program is developed in Java 1.4.2.15. in order to run the application Java JRE 1.4 or higher must be installed.

8 Installation

To install the program, unzip the zip file in a folder of your choice. The program folder contains the program file `AgentSystem2D.jar` and a folder `files` with the files

- `config.txt`
- `agents.txt`
- `rules.txt`

The file `files/config.txt` is necessary to run the program.

The files `files/agents.txt` and `files/rules.txt` are used as the starting initial setting.

The files

- `files/actions.log`
- `files/positions.log`
- `files/error.log`.

are not initially necessary: they are created on starting the program.