EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Iztapalapa

INSTITUTO TECNOLÓGICO DE IZTAPALAPA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

LENGUAJES Y AUTOMATAS

TEMA:

IMPLEMENTACIÓN DE INTERACTION NETS MEDIANTE INBLOOBS

PRESENTA:

CHAVARRÍA RODRÍGUEZ JANNET 171080426 (30%)

RODRIGUEZ MARTINEZ JUAN MANUEL 171080152 (30%)

UMEGIDO CAMACHO ERICK JOSUE 171080157 (40%)

PROFESOR:

M.C. Abiel Tomás Parra Hernández

ISC-7AM

MÉXICO, CDMX **JUNIO/2021**

Resumen

Las redes de interacción han sido una herramienta útil para el estudio de los aspectos computacionales de varios formalismos como los sistemas de reescritura de términos o el λ-cálculo, pero también son un paradigma de programación en sí mismas, puede considerarse como un lenguaje de programación visual y así es como fueron introducidas por Yves Lafont comenzando como una generalización de las redes de prueba para la lógica lineal multiplicativa.

En concreto es un formalismo de reescritura gráfica, muy prometedor para la implementación de lenguajes basados en el λ-cálculo, mediante una serie de traducciones del λ-calculus, las redes de interacción han demostrado ser un nuevo paradigma útil para la implementación de lenguajes funcionales, específicamente cuando el control y el intercambio de términos es una prioridad, además de describir las interacciones de los agentes en un escenario multiagente.

En otras palabras, no intentamos diseñar una lógica que describa el comportamiento de algunos sistemas computacionales dados, sino un lenguaje de programación para el que la disciplina de tipo ya es (casi) una lógica.

Palabras clave:

Redes de interacción, lenguaje de programación visual, INblobs net editor.

Abstract

Interaction networks have been a useful tool for studying the computational aspects of various formalisms such as term rewriting systems or the λ-calculus, but they are also a programming paradigm in themselves, it can be considered as a visual programming language and this is how they were introduced by Yves Lafont starting as a generalization of proof networks for multiplicative linear logic.

Specifically it is a graphical rewriting formalism, very promising for the implementation of λ-calculus based languages, through a series of of of translations of the λ-calculus, interaction networks have proven to be a useful new paradigm for the implementation of functional languages, specifically when control and term exchange is a priority, as well as describing agent interactions in a multi-agent scenario.

In other words, we are not trying to design a logic that describes the behavior of some given computational systems, but a programming language for which the type discipline is already (almost) a logic.

KEYWORDS:

Interaction Nets, Visual Programming language, INblobs Net editor.

ÍNDICE GENERAL

Resumen

Abstract

índice de ilustraciones……
índice de tablas ... ...……
introducción……...1
justificación……..2
Objetivo general……..3
Objetivos específicos ... ...……...3
Metodología de trabajo……..4

CAPÍTULO 1.-Generalidades de la disciplina……5
1.1 teoría de la computación ……..…… 5
1.2 Historia ……..…….5
1.3 Teoria de automatas ...……..7
1.4 Teoría de la computabilidad……...8
1.5 Teoría de la complejidad computacional……....10

CAPÍTULO 2.- Marco teórico……11
2.1 lógica lineal……..…11
2.1.1 Estructuras bien formadas……13
2.1.2 Reglas, axiomas y teoremas……13
2.2 Conjuntos computables y no computables……...16
2.2.1 Modos de computación……..18
2.2.2 Máquinas de Turing deterministas .....……...18
2.2.3 Máquinas de Turing no deterministas…….....19
2.2.4 La tesis de Church–Turing……19
2.3 Redes de principio e interacción y confluencia…….....19
2.3.1 Redes y patrones multiplicativos……..…20
2.4 sistemas multiagente……..…… 21
2.5 Paralelismo masivo……23
2.6 sistema de reescritura de grafos…….....24
2.7 Redes de interacción……..…..…25
2.7.1 El cálculo de interacción ligero……..……..……..……..……..……..27
2.7.2 Clasificación de las redes ……..……..……..……..……..……..……....29
2.7.3 reglas de interacción para redes……..……..……..……..……..……..31
2.7.4 combinadores de interacción……..……..……..……..……..……..……...31

CAPITULO 3 Implementación de redes concurrentes…….....35

3.1 Programas de Post–Turing……..……..……..……..……..……..……..……..35
3.2 IN blobs……..……..……..……..……..……..……..……..……..……..……...37
3.2.1 Cómo usar IN blobs……..……..……..……..……..……..……..……..……37
3.2.2 Descripción de la herramienta……..38
3.3 Estructuras de datos……42
3.4 Lógica INblobs……..……..……..……..……42
3.5 Reducción de las redes de interacción……..……..……..……..……..……45
conclusiones……..……..……..……..……..……..……..……..……..……..……47
bibliografía ……..……..……..……..……..……..……..……..……..……..……..48

índice de ilustraciones

Ilustración 1.1 Primeras computadoras……..……6
Ilustración 1.2 Teoría de los autómatas ……8
Ilustración 1.3 Complejidad de una máquina……..……9
Ilustración 2.1 Fluctuación de agente en un sistema……..….21
Ilustración 2.2 representación de tareas en paralelo……..23
Ilustración 2.3 orden de puertos en agentes……25
Ilustración 2.4 representación agente ……..….29
Ilustración 2.5 representación de signos en puertos……..29
Ilustración 2.6 representación de partición……..30
Ilustración 2.7 agentes y puertos……...31
Ilustración 2.8 reglas de interacción……..32
Ilustracion 2.9 representación de multitud ……...32
Ilustracion 2.10 red de multiplexación……...33
Ilustracion 2.11 par de redes de multiplexación……..33
Ilustración 2.12 Construcción de redes de multiplexación……..……..34
Ilustración 3.1 creación nuevo símbolo……39
Ilustracion 3.2. Plantilla de reglas……...40
Ilustracion 3.3. Controles de reducción……40
Ilustración 3.4 Interfaz Inblobs……....41

índice de tablas

Tabla 2.1 Componentes y sintaxis……...12
Tabla 2.2 Equivalencias lógicas……...14
Tabla 3.1. Descripción textual generada por Inblobs……..

Introducción

Este trabajo trata acerca del modelo computacional de redes de interacción, el cómo se comportan, qué tipo de lógica emplea y cómo está relacionada a la máquina de Turing, el documento consta de tres capítulos que a continuación se mencionan, y así mismo se describe el contenido de cada uno de ellos.

En el capítulo uno encontramos las generalidades de la disciplina tales como la teoría computacional, teoría de la computabilidad, aspectos básicos y terminología para el área de ciencias de la computación.

El capítulo dos está conformadas por el marco teórico el cual proporciona un conocimiento de la teoría que le da significado a la investigación, así como a nuestra área a partir de las teorías existentes y cómo pueden generarse nuevos conocimientos, incluyen temas sobre lógica lineal, redes de interacción, tipos de agentes, así como reglas de interacción.

En el capítulo tres, se muestra la implementación de las redes de interacción empleando un software para representar agentes, además se mencionan las similitudes que este modelo tiene con la máquina universal de Turing.

Justificación

Nuestros resultados a futuro pueden proporcionan una evidencia sorprendentemente sólida de la capacidad de IN para aprender simulaciones físicas precisas y generalizar su entrenamiento a nuevos sistemas con diferentes números y configuraciones de objetos y relaciones.

Es sorprendente lo que un sistema como lo son las redes de interacción pueden hacer como por ejemplo puede escalar a problemas del mundo real, es una plantilla prometedora para nuevos enfoques de IA para razonar sobre otros sistemas físicos y mecánicos, comprensión de escenas, percepción social, planificación jerárquica y razonamiento analógico.

Desde el punto de vista de la programación, las redes de interacción pueden considerarse un lenguaje de programación visual en sí mismo; sin embargo, se han utilizado de forma más eficiente como lenguaje de implementación, para codificar programas (de un lenguaje funcional básico) de forma inteligente que permite un control estrecho de las reducciones compartidas y las estrategias de evaluación.

Mostramos cómo se pueden utilizar las redes internas:

- para representar visualmente los programas funcionales y para animar la ejecución de dichos programas mediante la reescritura de gráficos.

- para razonar sobre programas funcionales y realizar transformación visual de programas utilizando el sistema de fusión.

- Aunque se ha avanzado en la creación de una notación visual para programas funcionales (o incluso un lenguaje de programación funcional completamente visual), hace falta más investigación e integración de ciertas teorías aplicables a este modelo computacional.

Objetivo general

Implementación a través de una simulación mediante INBLOBS para la representación de las redes de interacción y su comparación con la máquina de Turing

Objetivos específicos

- Investigación de redes de interacción y sus aplicaciones.
- Entendimiento de las redes de interacción.
- Ejemplificar el uso de redes de interacción mediante la herramienta INblobs (Editor e intérprete de Interacción Nets)
- comparación entre el modelo de redes de interacción con la máquina universal de Turing.

Metodología de trabajo Ágil

Kanban

La estrategia Kanban conocida como 'Tarjeta Visual' útil para los responsables de proyectos. Esta consiste en la elaboración de un cuadro o diagrama en el que se reflejan n columnas de tareas; pendientes, en proceso o terminadas. Este cuadro debe estar al alcance de todos los miembros del equipo, evitando así la repetición de tareas o la posibilidad de que se olvide alguna de ellas. Por tanto, ayuda a mejorar la productividad y eficiencia del equipo de trabajo.

Objetivos del uso de esta metodología

- Balancear la demanda con la capacidad.
- Limitar el trabajo en proceso, mejorar el "flujo" del trabajo, descubrir los problemas tempranamente y lograr un ritmo sostenible.
- Controlar el trabajo (no la gente), coordinar y sincronizar, descubrir los cuellos de botella y tomar decisiones que generen valor.
- Equipos autoorganizados. • Lograr una cultura de optimización incremental.

Capítulo 1 Generalidades de la disciplina

1.1 Teoría de la Computación

La teoría de la computación o teoría de la informática es un conjunto de conocimientos racionales y sistematizados que se centran en el estudio de la abstracción de los procesos, con el fin de reproducirlos con ayuda de sistemas formales; es decir, a través de símbolos y reglas lógicas. La teoría de la computación permite modelar procesos dentro de las limitaciones de dispositivos que procesan información y que efectúan cálculos; como, por ejemplo, el ordenador. Para ello, se apoya en la teoría de autómatas, a fin de simular y estandarizar dichos procesos, así como para formalizar los problemas y darles solución.

1.2 Historia

La teoría de la computación comienza propiamente a principios del siglo XX, poco antes que las computadoras electrónicas fuesen inventadas. En esta época varios matemáticos se preguntaban si existía un método universal para resolver todos los problemas matemáticos. Para ello debían desarrollar la noción precisa de método para resolver problemas, es decir, la definición formal de algoritmo.
Algunos de estos modelos formales fueron propuestos por precursores como Alonzo Church (cálculo Lambda), Kurt Gödel (funciones recursivas) y Alan Turing (máquina de Turing). Se ha mostrado que estos modelos son equivalentes en el sentido de que pueden simular los mismos algoritmos, aunque lo hagan de maneras diferentes. Entre los modelos de cómputo más recientes se encuentran los lenguajes de programación, que también han mostrado ser equivalentes a los modelos anteriores; esto es una fuerte evidencia de la conjetura de Church-Turing, de que todo algoritmo habido y por haber se puede simular en una máquina de Turing, o equivalentemente, usando funciones recursivas.

En 2007 Nachum Dershowitz y Yuri Gurevich publicaron una demostración de esta conjetura basándose en cierta axiomatización de algoritmos.
Uno de los primeros resultados de esta teoría fue la existencia de problemas imposibles de resolver algorítmicamente, siendo el problema de la parada el más famoso de ellos. Para estos problemas no existe ni existirá ningún algoritmo que los pueda resolver, no importando la cantidad de tiempo o

memoria se disponga en una computadora. Asimismo, con la llegada de las computadoras modernas se constató que algunos problemas resolubles en teoría eran imposibles en la práctica, puesto que dichas soluciones necesitaban cantidades irrealistas de tiempo o memoria para poderse encontrar.

Ilustración 1.1 Primeras computadoras

Desde tiempo inmemorial se sabe que cierta clase de problemas, como la determinación del máximo común divisor de dos números enteros, mediante el algoritmo de Euclides, o la determinación de los números primos, mediante la criba de Eratóstenes, son algorítmicamente solubles, i.e., hay algoritmos o procedimientos mecánicos que permiten obtener la solución del problema en cuestión.

De manera que hasta principios del siglo XX se daba por hecho que existían algoritmos y que el único problema residía en determinarlos. Así pues, si lo que se desea es determinar un algoritmo, no hay ninguna necesidad de definir la clase de todos los algoritmos; eso sólo es necesario si se pretende demostrar que algún problema no es algorítmicamente soluble, i.e., que para dicho problema no hay ningún algoritmo que lo resuelva.

Pero parece ser que fue, por una parte, el problema de la decidibilidad de la lógica de predicados planteado por Hilbert y Ackermann en su libro sobre lógica, publicado en 1928, y, por otra, el asunto de la solubilidad de todo problema matemático, lo que indujo, en aras a resolverlos, a diversos investigadores a partir de 1930, y entre los que cabe mencionar a Gödel, Church y Turing, a proponer diversas formalizaciones del concepto informal de función mecánicamente computable. Debido a que de todas esas

formalizaciones, y de otras propuestas por Kleene, Post y Markoff, se demostró que eran dos a dos equivalentes, se propuso la hipótesis, conocida como Hipótesis de Church-Turing-Post-Kleene, que afirma la coincidencia entre el concepto informal de función parcial mecánica o algorítmicamente computable, y el concepto formal, matemático, de aplicación parcial recursiva. Naturalmente, esa hipótesis, de carácter similar a otras hipótesis propuestas en las ciencias empíricas, no es demostrable, y su fundamento último reside en las equivalencias antes mencionadas.

Hay estudios completos dedicados, en primer lugar, al estudio de diferentes clases de aplicaciones recursivas, desde las recursivas primitivas, hasta las parciales recursivas, pasando por las recursivas generales, así como al de diversas clases de relaciones, entre las que cabe citar a las recursivas primitivas, las recursivamente enumerables y a las recursivas, demostrando además, ciertos teoremas fundamentales de la teoría de la recursión, debidos en gran medida a Kleene; y, en segundo lugar, a la aplicación de la teoría de la recursión a la demostración de la indecidibilidad de la lógica de predicados de primer orden, i.e., a la demostración de que el conjunto de los números de Gödel de los teoremas de la lógica de predicados de primer orden no es recursivo, aunque sí sea recursivamente enumerable; y de los teoremas de incompletitud de Gödel, de los cuales, el primero da cuenta, esencialmente, de la diferencia, en la aritmética, entre las nociones de verdad y demostrabilidad, mientras que el segundo afirma que, bajo ciertas condiciones, no es posible demostrar desde una teoría, la consistencia de la misma, i.e., esencialmente que el infinito no es eliminable en las matemáticas.

1.3 Teoría de autómatas

Esta teoría provee modelos matemáticos que formalizan el concepto de computadora o algoritmo de manera suficientemente simplificada y general para que se puedan analizar sus capacidades y limitaciones. Algunos de estos modelos juegan un papel central en varias aplicaciones de las ciencias de la computación, incluyendo procesamiento de texto, compiladores, diseño de hardware e inteligencia artificial.
Existen muchos otros tipos de autómatas como las máquinas de acceso aleatorio, autómatas celulares, máquinas ábaco y las máquinas de estado abstracto; sin embargo, en todos los casos se ha mostrado que estos modelos no son más generales que la máquina de Turing, pues la máquina de Turing tiene la capacidad de simular cada uno de estos autómatas. Esto da lugar a que se piense en la máquina de Turing como el modelo universal de computadora.

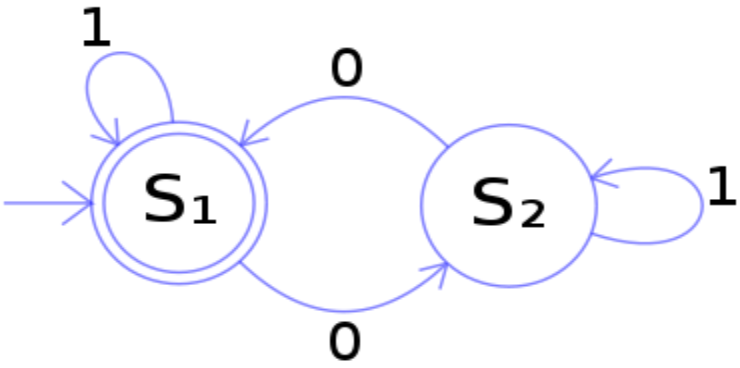


Ilustración 1.2 Teoría de los autómatas

1.4 Teoría de la computabilidad

Esta teoría explora los límites de la posibilidad de solucionar problemas mediante algoritmos, gran parte de las ciencias computacionales están dedicadas a resolver problemas de forma algorítmica, de manera que el descubrimiento de problemas imposibles es una gran sorpresa.

La teoría de la computabilidad, también denominada teoría de la recursión, es una de las cuatro partes que constituyen la lógica matemática, siendo las otras tres, la teoría de conjuntos, la teoría de modelos y la teoría de la demostración, y se ocupa del estudio y clasificación de las relaciones y aplicaciones computables. Además, la teoría de la computabilidad, junto con la teoría de autómatas, lenguajes y máquinas, es el fundamento de la informática teórica y esta, a su vez, de la industria de los ordenadores, los problemas se clasifican en esta teoría de acuerdo a su grado de imposibilidad:

Los computables son aquellos para los cuales sí existe un algoritmo que siempre los resuelve cuando hay una solución y además es capaz de distinguir los casos que no la tienen. También se les conoce como decidibles, resolubles o recursivos.

Los semicomputables son aquellos para los cuales hay un algoritmo que es capaz de encontrar una solución si es que existe, pero ningún algoritmo determina cuando la solución no existe (en cuyo caso el algoritmo para encontrar la solución entraría a un bucle infinito). El ejemplo clásico por excelencia es el problema de la parada. A estos problemas también se les conoce como listables, recursivamente enumerables o reconocibles, porque

si se enlistan todos los casos posibles del problema, es posible reconocer a aquellos que sí tienen solución.

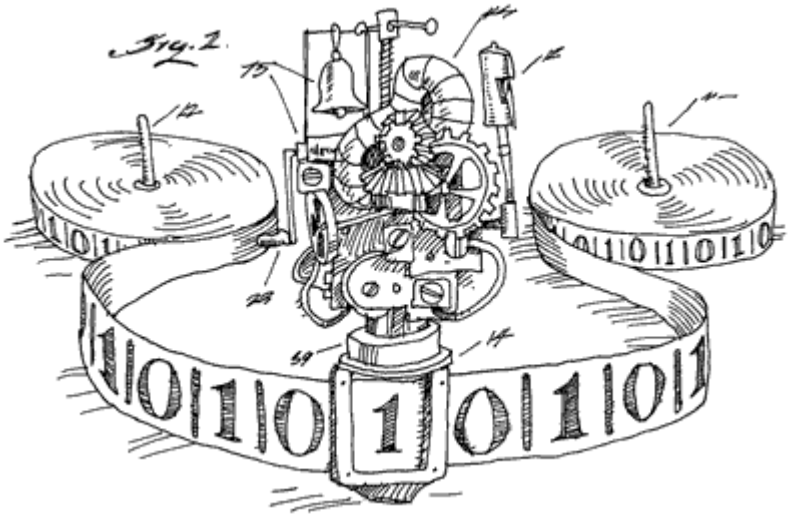


Ilustración 1.3 Complejidad de una máquina

Los incomputables son aquellos para los cuales no hay ningún algoritmo que los pueda resolver, no importando que tengan o no solución. El ejemplo clásico por excelencia es el problema de la implicación lógica, que consiste en determinar cuándo una proposición lógica es un teorema; para este problema no hay ningún algoritmo que en todos los casos pueda distinguir si una proposición o su negación es un teorema.

La herramienta principal para lograr estas clasificaciones es el concepto de reducibilidad:
Un problema A se reduce al problema B si bajo la suposición de que se sabe resolver el problema B es posible resolver el problema A; esto se denota por A <t B, e informalmente significa que el problema A no es más difícil de resolver que el problema B. Por ejemplo, bajo la suposición de que una persona sabe sumar, es muy fácil enseñarle a multiplicar haciendo sumas repetidas, de manera que multiplicar se reduce a sumar.

La teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que se pueden resolver con un algoritmo o equivalentemente con una máquina de Turing. Las preguntas fundamentales de la teoría de la computabilidad son:
¿Qué problemas puede resolver una máquina de Turing?
¿Qué otros formalismos equivalen a las máquinas de Turing?
¿Qué problemas requieren máquinas más poderosas?
¿Qué problemas requieren máquinas menos poderosas?

1.5 Teoría de la complejidad computacional

Aun cuando un problema sea computable, puede que no sea posible resolverlo en la práctica si se requiere mucha memoria o tiempo de ejecución. La teoría de la complejidad computacional estudia las necesidades de memoria, tiempo y otros recursos computacionales para resolver problemas; de esta manera es posible explicar por qué unos problemas son más difíciles de resolver que otros. Uno de los mayores logros de esta rama es la clasificación de problemas, similar a la tabla periódica, de acuerdo a su dificultad. En esta clasificación los problemas se separan por clases de complejidad.

Esta teoría tiene aplicación en casi todas las áreas de conocimiento donde se desee resolver un problema computacionalmente, porque los investigadores no solo desean utilizar un método para resolver un problema, sino utilizar el más rápido. La teoría de la complejidad computacional también tiene aplicaciones en áreas como la criptografía, donde se espera que descifrar un código secreto sea un problema muy difícil a menos que se tenga la contraseña, en cuyo caso el problema se vuelve fácil.

La teoría de la complejidad computacional clasifica las funciones computables según el uso que hacen de diversos recursos en diversos tipos de máquina.

Capítulo 2

Marco teórico

2.1 Lógica lineal

Introducida por Jean-Yves Girard en el año de 1986, como una forma de extender la lógica clásica que comúnmente se basa en sus posiciones estables ciertas o falsas true or false.

Por ejemplo:

Tenemos una presuposición de A y otra de A entonces B y al momento de llevarlas a cabo podemos deducir AB pero A se sigue manteniendo, la lógica lineal nos va a hacer mención de que estas suposiciones previas son consumibles, esto significa por la misma operación al poder deducir AB habrá de haber sido consumida.

Esto en una oración más entendible o una lógica clásica:

A = Esta nublada

B = puede llover

Sin embargo, en la lógica lineal las premisas son consumibles.

Tenemos la premisa A como “tengo 8 dólares”.

Premisa B; “un café vale 8 dólares”.

La lógica lineal va a estar ahí para poder representar algunos aspectos de la vida cotidiana en la cual no es posible mantener las premisas:

Esto quiere decir si yo tengo 8 dólares y un café vale 8 dólares, si yo deduzco que ya tengo el café entonces A sido consumida y deja de existir.

Por lo tanto, la lógica lineal va a ser una versión refinada de la lógica clásica; esto significa que no debe de ser tomada como una nueva lógica.

Girard, en 1995 dijo que, “esta debe de ser vista como una extensión de la lógica usual con la introducción de nuevas conectivas”.

Con el paso del tiempo la lógica lineal consolida su aporte a la ciencia, logrando trascender a través del impacto que alcanzó en la línea de tiempo de su nacimiento. Entre sus principales contribuciones nos presenta una nueva visión de los alcances que se pueden lograr con la lógica clásica.

Componentes formales y sintaxis.

símbolo	Significado	Tipo	Lógica tradicional
&	And	aditivos	∧ (and)
Å	Or		∨ (or)
Ä	And	multiplicativos	∧ (and)
Ã	Or		∨ (or)
⊤ 1	Top One		Top
⊥ 0	Bottom Zero		bottom
$A^{\perp}$	Linear negation	Negación	¬A (negación)
	Implicación lineal		Conjunción
!A	Of course	exponenciales	Reglas estructurales (debilitación y contracción)
?A	Why not		

Tabla 2.1 componentes y sintaxis

2.1.1 Estructuras bien formadas.

Tenemos que estas serán las que ocuparemos dentro de la lógica lineal.

Una fórmula bien formada no se diferencia mucho de la clásica, si una es una constante lógica o una variable proposicional, entonces es una fórmula bien formada.

Si A es una variable proposicional y tiene una negación lineal también es una fórmula bien formada.

Si A y B son variables proposicionales y están unidas con una colectiva anilla entonces también van a ser una fórmula bien formada.

Ejemplo de una fórmula bien formada:

2.1.2 Reglas, axiomas y teoremas

Las reglas de inferencia son las reglas que gobiernan a las operaciones deductivas por las que una o dos fórmulas ya probadas se pasan a una tercera.

La lógica lineal se basa en un conjunto de axiomas en los axiomas de la lógica lineal clásica, de igual forma cada una ayudó a definir el siguiente grupo de axiomas.

Axiomas consecuentes:

Axioma consecuente lógico

- $A \vdash A$

Constantes lógicas

- $\vdash A$
- $\Gamma \vdash \Delta \top$
- $\bot \vdash$
- $0, \Gamma \vdash \Delta$

En la Lógica lineal se introducen ciertas equivalencias lógicas, cabe destacar que la expresión A lógicamente equivalente a B se debe de interpretar como A implica B por B implica A.

1	$\vdash A^{\perp\perp} \equiv A$	Ley de doble negación
2	$\vdash A \tilde{A} A^{\perp}$	Excluido medio
3	$\vdash (A \ddot{A} B)^{\perp} \equiv (A^{\perp}) \tilde{A} (B^{\perp})$	Ley de De Morgan
4	$\vdash (A \tilde{A} B)^{\perp} \equiv (A^{\perp}) \ddot{A} (B^{\perp})$	
5	$\vdash (A \tilde{A} B)^{\perp} \equiv (A^{\perp}) \mathring{A} (B^{\perp})$	
6	$\vdash (A \mathring{A} B)^{\perp} \equiv (A^{\perp}) \tilde{A} (B^{\perp})$	
7	$\vdash (!A)^{\perp} \equiv ? (A^{\perp})$	
8	$\vdash (?A)^{\perp} \equiv ! (A^{\perp})$	
9	$\vdash A \ddot{A} (B \mathring{A} C) \equiv (A \ddot{A} B) \mathring{A} (A \ddot{A} C)$	Ley distributiva
10	$\vdash A \tilde{A} (B \tilde{A} C) \equiv (A \tilde{A} B) \tilde{A} (A \tilde{A} C)$	Ley distributiva
11	$\vdash (!A) \ddot{A} (!B) \equiv !(A \& B)$	
12	$\vdash (?A) \tilde{A} (?B) \equiv ? (A \mathring{A} B)$	
13	$\vdash A \equiv A^{\perp} \tilde{A} B$	

tabla 2.2 equivalencias lógicas

Los teoremas se dividen en dos grupos:

Semántica de fases: caracterizada por la probabilidad, es utilizada para mostrar varias propiedades de la lógica lineal como la invisibilidad, eliminación de corte, normalización fuerte y de civilidad de unos fragmentos

Cálculo concurrente: introducido como una generalización de las restricciones concurrentes en el que los procesos pueden consumir información por medio de la operación de petición, también sintetiza pruebas que pueden ser evaluadas como programas funcionales.

Ejemplo de demostración en lógica lineal:

-(AXB) $^{\perp\perp)}$Ã(A$^{\perp}$))

Esto se lee como; A por B con una negación lineal que a su vez es una implicación lineal con la negación lineal de B para negación lineal de A.

Inicialmente se utiliza el axioma de derivación A, deriva a, A.

Después se utiliza la regla de diferencia de negación lineal por la derecha para pasar del otro lado de la derivación una variable proposicional con una negación. Quedando A, . Aplicamos la misma regla diferenciada en B.

A continuación, se utilizará la regla de inferencia de intercambio a la derecha. Que nos dice que tenemos una variable proposicional A y B y la podemos intercambiar por B y A.

CON RESPECTO A= , A

CON RESPECTO B= , B

Después utilizaremos la regla de inferencia Ä A, a la derecha. Esta regla nos dice que si tenemos una derivación de A, y una derivación de B; se pueden unir las dos variables (Ä) proposicionales con un conector lineal Ä.

El resultado de aplicar la regla de inferencia es:

⊢B$^{\perp}$,A$^{\perp}$,A

Lo siguiente seria aplicar una negación lineal por la izquierda. Esta regla en diferencia nos dice que, si tenemos una variable posicional A, la podemos pasar del lado izquierdo del símbolo de derivación.

Esta acción nos cambia la variable proposicional movida con una negación lineal.

En este caso tomamos AXB como una variable proposicional y esta variable proposicional es la que moveremos al lado izquierdo teniendo. Negación de A por B deriva negación A, negación de B.

$$(A)^{\perp} \vdash B^{\perp}, A^{\perp}$$

Lo siguiente sería utiliza la regla de diferencia de par por la derecha; esta regla diferencial nos dice que si tenemos dos variables proposicionales A y B las podemos unir con un conector lógico lineal de par.

En nuestro caso las variables proposicionales que utilizaremos serán negación de B y negación de A; quedando como la fórmula completa: negación de A por B que deriva la negación de B de la negación de A.

$$(A)^{\perp} \vdash (B^{\perp})\,(A^{\perp})$$

Finalmente, para terminar con el ejemplo, se utiliza la regla de diferencia de implicado lineal a la derecha. Esta regla nos dice que si tenemos una variable proposicional A que deriva a una variable proposicional B, podemos unir las dos variables proposicionales moviendo la variable A, a la derecha y uniéndose con la colectiva lógica del implicado lineal.

Concluimos con: negación de A por B implicador lineal negación de B para negación de A.

$$(A)^{\perp}\ \ (\,(B^{\perp})\,(A^{\perp}))$$

2.2 Conjuntos computables y no computables

La teoría de la recursión se originó en la década de 1930, con el trabajo de Kurt Gödel, Alonzo Church, Alan Turing, Stephen Kleene y Emil Post, los resultados fundamentales que obtuvieron los investigadores estabilizaron el concepto de función computable como la manera correcta de formalizar la idea sobre cálculos efectivos.
Estos resultados llevaron a Stephen Kleene a acuñar dos nombres, "Tesis de Church" y "Tesis de Turing", hoy en día ambos se consideran como una única hipótesis, la Tesis de Church-Turing, la cual establece que cualquier función

que sea computable por un cierto algoritmo es una función computable. Aunque en un principio era algo un tanto escéptico, alrededor del año 1946, Gödel defendió esta tesis:
"Tarksi ha subrayado en su lectura (y creo justamente) la gran importancia del concepto de recursividad general (o computabilidad de Turing). En mi opinión esta importancia se debe en gran medida al hecho de que, con este concepto, por fin se ha conseguido darle una noción absoluta a una interesante noción epistemológica, es decir, una que no depende del formalismo elegido.
Con una definición sobre cálculos efectivos aparecieron las primeras pruebas de que hay ciertos problemas en las matemáticas que no pueden ser decididos de una manera eficaz. Church y Turing, inspirados por las técnicas usadas por Gödel para probar sus teoremas sobre la incompletitud, demostraron por separado que no es posible decidir el problema de una manera eficaz. Este resultado demostró que no existe un procedimiento algorítmico que pueda decidir de manera correcta si ciertas proposiciones matemáticas son verdaderas o no.
Muchos problemas en las matemáticas han sido demostrados ser indecidibles una vez se establecieron estos primeros ejemplos. En 1947, Márkov y Post publicaron por separado sus trabajos mostrando que el problema de las palabras para los semigrupos no puede ser decidido de una manera eficaz. Ampliando este resultado, Pyotr Novikov y William Boone demostraron independientemente en la década de 1950 que el problema de las palabras para los semigrupos no se puede resolver de una manera efectiva: no hay ningún procedimiento eficaz que, dada una palabra en un grupo, decida si el elemento representado por la palabra es el elemento identidad del grupo.

En 1970, Yuri Matiyasevich demostró (usando los resultados de Julia Robinson) el Teorema de Matiyasevich, el cual implica que el décimo problema de Hilbert no tiene una solución eficaz; este problema preguntaba si había o no un procedimiento mediante el cual se pudiera decidir si una ecuación diofántica sobre los números enteros tiene una solución entera. La lista de problemas indecidibles contiene ejemplos adicionales sobre problemas sin soluciones computables.
El estudio sobre qué construcciones matemáticas pueden ser llevadas a cabo de una forma eficaz se denomina a veces matemática recursiva; el Handbook of Recursive Mathematics (Ershov et al. 1998) cubre muchos de los resultados conocidos en este campo.
Función computable en un modelo de computación
Supongamos definido un modelo de computación M

· Debemos asociar a cada M–programa p una función [[p]] sobre la(s) estructura(s) de dato(s)
D [[p]]: D– → D
posiblemente parcial (es posible que p no pare sobre algún dato de entrada), definida como sigue: para cada d ∈ D, [[p]](d) = resultado de ejecutar p con entrada d.
Definición: Una función f: D– → D es M–computable si existe un M–programa p tal que [[p]] = f.

2.2.1 Modos de computación

Dependen de la semántica y del concepto de solución.

1. Modo secuencial-determinista: La ejecución de un programa sobre un dato es única. La solución de un problema es la solución lógica exacta para cada entrada.
2. Modo paralelo: El programa especifica diversas tareas que se pueden ejecutar en distintos procesadores, compartiendo información. Solución exacta.
3. Modo no determinista. La ejecución no es única: existen instrucciones que pueden elegir entre 2 o más opciones. I Un programa resuelve un problema si alguna de sus posibles ejecuciones ofrece una solución exacta para el dato de entrada.
4. Modo probabilista: Los programas son deterministas, pero contienen instrucciones que se ejecutan sólo con cierta probabilidad. Un programa resuelve un problema si, con cierta probabilidad, obtiene una solución exacta.

2.2.2 Máquinas de Turing deterministas

consta de
1. Un conjunto finito, Q, cuyos elementos se llaman estados.
2. Un alfabeto Σ (denominado alfabeto de entrada).
3. Un alfabeto Γ tal que Σ ∪ {B} ⊆ Γ, (B ∈/ Σ), denominado alfabeto de trabajo o de cinta).
4. q0 ∈ Q (se denomina estado inicial).
5. F ⊆ Q (sus elementos se llaman estados finales o de aceptación).
6. Un conjunto finito P ⊆ Q × Γ × (Γ ∪ {R, L}) × Q (con R, L ∈/ Γ) tal que: Para todo q ∈ Q y s ∈ Γ, (q,s,t1, q1) ∈ P y (q,s,t2, q2) ∈ P =⇒ t1 = t2 y q1 = q2

2.2.3 Máquinas de Turing no deterministas

Si en la definición anterior eliminamos la ´ultima condición decimos que se trata de una máquina de Turing no determinista.
1. Q – el conjunto de estados.
2. Σ - el alfabeto de entrada.
3. Γ - el alfabeto de cinta.
4. δ: Q×Γ→P (Q×Γ× {I, D}) – la función de transición.
5. q0∈Q – el estado inicial.
6. qsi∈Q – el estado aceptador.
7. qno∈Q – el estado que rechaza.

2.2.4 La tesis de Church–Turing

· La tesis de Church–Turing afirma que la clase de las funciones numéricas cuyos valores pueden calcularse siguiendo algún algoritmo coincide con la clase de las funciones Turing–computables (o recursivas).
· Esta tesis sostiene que el concepto informal “función computable mediante un algoritmo” coincide con un concepto formal “función recursiva”.
· Por tanto, esta afirmación no puede ser demostrada formalmente, solo podemos validarla en la práctica demostrando que todos los modelos de computación propuestos son equivalentes.
· Esto es lo que ha ocurrido hasta la fecha, lo que muestra que la noción de función computable es muy estable y, a la vez, es interpretado por muchos investigadores como un respaldo a la tesis.

2.3 Redes de principado e interaccion

los gráficos de conexión usan nomenclatura neta y un formalismo gráfico, introducimos ahora el concepto de principado que caracteriza a las redes de interacción propiamente dichas.
Una asignación de principado mapea cada etiqueta de agente a un subconjunto de sus puertos llamados puertos principales. Indicaremos los puertos principales de conformidad con Lafont dibujándolos con flechas gruesas.
Una conexión de puerto en una red donde dos puertos principales se encuentran se llama corte y el nodo de puerto correspondiente se llama nodo de corte. Si ninguno de los puertos es principal, la conexión del puerto se denomina enlace y el nodo del puerto se denomina nodo de enlace. Si solo un puerto es principal, la conexión del puerto se llama arco principal, dirigido

de la manera obvia, con el nodo de puerto llamado nodo de arco. El Principado restringe la reescritura aplicando la siguiente regla invariante.
Regla invariante: en cada regla, los dos bordes de puerto que conectan a los dos agentes en el LHS son ambos principales. Por lo tanto, la reescritura neta es (intencionalmente) un análogo de la eliminación de cortes
Una regla reflexiva es aquella en la que en el LHS, ambos agentes tienen la misma etiqueta de agente y ambos bordes de puerto en el corte tienen la misma etiqueta de puerto. Una regla reflexiva posee claramente una inversión de su LHS que intercambia todos los pares de puertos etiquetados de manera similar en la periferia del LHS.

Una regla reflexiva es simétrica sif el RHS es invariante bajo la inversión que intercambia todos los pares de puertos del RHS correspondientes a los pares de puertos etiquetados de manera similar en la periferia del LHS. Aquí hay un par de propuestas sencillas. El primero es una simple adaptación de una propuesta de Lafont.
Si cada agente tiene como máximo un puerto principal (de modo que cada par de agentes puede formar el LHS de una regla como máximo de una manera), cada par de agentes forma el LHS de como máximo una regla, y cualquier regla reflexiva es simétrica , entonces la reescritura neta es Church-Rosser.
En una red o patrón G, dos redes con imágenes distintas g1 (L1) y g2 (L2), pueden superponerse en no más de una parte de las imágenes g1 (l1 (K1)) y g2 (l2 (K2)), de los dos patrones de interfaz K1 y K2 de cualquier regla para la que g1 (L1) y g2 (L2) son imágenes redex.
Un resultado estándar en la reescritura de doble expulsión da una propiedad CR de un paso para tal situación (ver las referencias citadas). Observando además que con las restricciones dadas, cada corte en G puede hacer que el par de agentes a los que se une se reescribe como máximo de una manera posible.

2.3.1 Redes y patrones multiplicativos

En la descripción anterior, diferentes instancias de agentes etiquetados con L en una red enhebrada podrían admitir particiones completamente no relacionadas de sus puertos, para cualquier etiqueta de agente L. En esta y las secciones restantes, aún pueden admitir diferentes particiones, pero estas ya no estarán relacionadas.
Una asignación de partición mapea cada etiqueta de agente a un conjunto de particiones de sus puertos. Este es el acicate para que impongamos otro invariante. Subprocesamiento invariante: dado un patrón de subprocesos, la

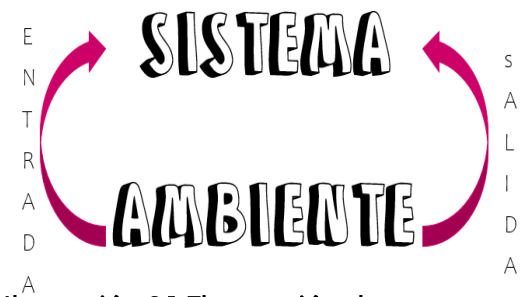
partición en los bordes del puerto de un nodo de agente inducida por cualquier subproceso es siempre un miembro del conjunto asignado a su etiqueta. Además, si (A, F) es un patrón enhebrado, entonces cualquier elección de particiones para agentes de entre los asignados a sus etiquetas se unen para formar un subproceso, y F es exactamente el conjunto de subprocesos dado por todas las posibles opciones de este tipo. Esto claramente restringe las posibles redes enhebradas de manera bastante severa.

Una asignación de partición conmutada s es una asignación de partición en la que para cada etiqueta de agente L, para cada partición de los puertos de L en s (L), se distingue un conjunto de puertos en la partición. Llamamos particiones que poseen miembros tan distinguidos, particiones conmutadas.

Dada una asignación de partición conmutada, un patrón de subprocesos (A, F) es un patrón multiplicativo si f para cada subproceso en F, descartando todos los bordes de puerto no distinguidos del patrón, da como resultado un bosque de expansión (SF) del patrón .
Un bosque en expansión es obviamente solo un subgrafo del patrón en su representación gráfica bipartita, que contiene todos los nodos y es acíclico. Para las redes multiplicativas, tenemos las mismas construcciones, excepto que no hay nodos de puerto libres o aislados en el patrón original (aunque puede haber algunos después de eliminar los bordes de los puertos no distinguidos), y para las redes multiplicativas especiales exigimos además que el número de conexiones conectadas componentes en el SF es igual al número de componentes conectados en la red original (por lo que el descarte de los bordes de puerto no distinguidos poda las periferias de los hiper-árboles de enhebrado originales pero no los desconecta).

2.4 Sistemas multiagentes

los agentes son autónomos y capaces de actuar independientemente, exhibiendo control sobre su estado interno.



Por lo tanto; es un sistema de computación capaz de realizar acciones autónomas en un ambiente.

Ilustración 2.1 Fluctuación de agente en un sistema

El modelo de un agente dentro un sistema multiagente es un proceso del sistema operativo.

- se utiliza usualmente un sistema de red utilizando protocolos de investigación.
- Es un sistema distribuido en el cual los nodos o elementos son sistemas de Inteligencia Artificial.
- se comunican a través de un mecanismo de comunicación interproceso.
- es un sistema distribuido y produce un resultado en conjunto inteligente.

Leonado Garrido(2013), dice que; es un sistema que, aunque no existe una definición formal y precisa de lo que es un agente, estos son por lo general. el comportamiento inteligente se genera. sistemas basados en agentes y SMA.

Es un sistema construido por un número de agentes que interactúan entre sí, para interactuar satisfactoriamente, los agentes necesitan las habilidades de cooperación, coordinación y negociación.

Un sistema multiagente está definido como; un conjunto de agentes autónomos, generalmente heterogéneos y potencialmente independientes, que trabajan en común resolviendo un problema.

como características de este tipo de sistemas (MAS) son:

- Actividades conjuntas y cooperación: es el proceso por el que ciertos agentes participantes generan deberes mutuamente dependientes en actividades conjuntas (planes).
- Conflictos: surgen cuando al resolver un problema hay una o varias circunstancias.
- Negociación: se resuelve en un plan común; que es óptimo cuando se da una situación de paridad conjunta.
- Compromisos y planificación de actividades: se forman un conjunto de restricciones sobre las acciones y creencias de cada agente con respecto al resto.
- modelo del conocimiento y su comunicación: constituida por el lenguaje que se comunicara entre agentes y está representado por conocimiento común.

2.5 Paralelismo masivo

Definido como la capacidad de un sistema para ejecutar más de un hilo de ejecución (proceso) al mismo tiempo.

Ejemplo. Tenemos una CPU con diferentes procesos ejecutándose y representado en cada color en qué procesador se están ejecutando; el verde en un procesador, el morado es otro procesador, el rosa es otro y así por cada color de flecha.

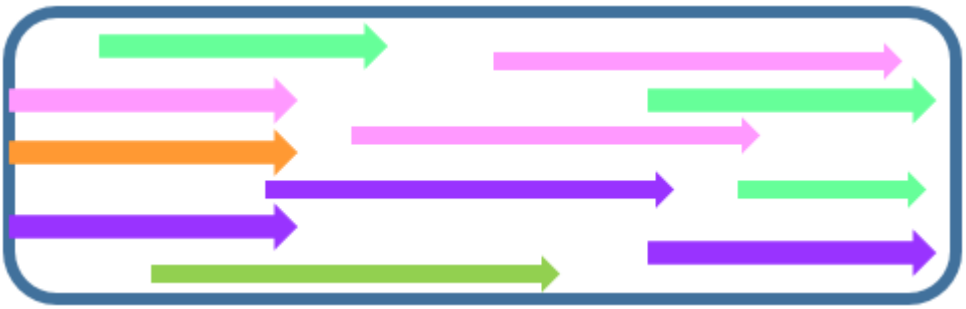


Ilustración 2.2 representación de tareas en paralelo

Por tanto, ahora la situación es diferente, estos son procesos paralelos porque se están ejecutando a la vez en 5 procesadores a la vez.

Pero ya en el caso de las flechas de color tanto el rosa y el morado del final, a pesar de que coexisten, se están ejecutando en el mismo procesador, y por lo tanto no pueden ejecutarse a la vez. Se ejecuta uno y luego el otro, se ejecutan por turnos.

Por lo tanto, las flechas de en medio, son procesos no paralelos, porque ni siquiera coexisten, no se ejecutan a la vez ni siquiera existe.

El paralelismo implica concurrencia, si varios procesos se ejecutan al mismo tiempo; es decir, que son paralelos eso implica que coexisten y por tanto son concurrentes. Por tanto, cuando hay paralelismo hay concurrencia.

Pero de lo contrario no siempre es tiempo, puede serlo, pero no es la implicación. La concurrencia no implica paralelismo. Si varios procesos conviven a la vez, es decir, que son concurrentes; puede ser que se ejecuten a la vez o no, por tanto, no son necesariamente paralelos por ser el hecho de ser concurrentes

Arleinginformatica, nos dice que, "se debe de disponer de suficientes procesadores como para que todas las operaciones que puedan ser

ejecutadas en paralelo puedan ser asignadas a procesadores separados. Esto ofrece de una forma de ejecutar un programa en el menor tiempo posible."

El proceso de los sistemas de multiprocesamiento generalmente son la confiabilidad y la disponibilidad muy altas, como así, también el incremento del poder de computación. El diseño modular proporciona una flexibilidad importante y facilita la expansión de la capacidad.

Características:

- Es indicado de forma específica por un programador mediante una "construcción de concurrencia".
- Se pueden utilizar procesadores separados para ejecutar cada proposición.
- Es susceptible de errores de programación difíciles de detectar y depurar.
- El programador puede omitir situaciones donde se aplica el paralelismo.

2.6 sistema de reescritura de grafos

Un sistema de reescritura de grafos puede tener un conjunto potencialmente grande de reglas para aplicar a un grafo, el orden en que se aplican las reglas puede alterar en gran medida el gráfico final cuando se considera la reescritura general de gráficos, en el caso de las redes de interacción, la propiedad de confluencia fuerte garantiza que todas las
secuencias de reducción a la forma normal completa son equivalentes, sin embargo, este no es el caso si utilizamos una noción de reducción que no alcanza una forma normal completa (por ejemplo, la reducción a la forma normal de). Además, en el caso de las redes de interacción, aunque el grafo final no cambie, el tamaño y la disposición del grafo durante el proceso de reescritura pueden diferir según las reglas y el lugar en el que se apliquen primero.
Por lo tanto, los usuarios pueden querer no aplicar reglas a ciegas, sino crear una estrategia en torno a estas reglas para dirigir la reescritura.
La reescritura estratégica se ha estudiado para los sistemas de reescritura de términos, y hay lenguajes que permiten al usuario especificar una estrategia y aplicarla.
En este trabajo se emplea un lenguaje para definir estrategias para sistemas de reescritura de grafos, donde no sólo la estrategia necesita tener en cuenta las reglas y las secuencias de reglas, sino también la ubicación y la propagación en un grafo, se complica por el hecho de que en un grafo no

existe la noción de raíz, por lo que las estrategias de reescritura de términos estándar de reescritura de términos basadas en recorridos descendentes o ascendentes no tienen sentido en este entorno), por ello, es crucial a futuro desarrollar un lenguaje específico para tratar las estrategias de las redes de interacción que también puede aplicarse a los sistemas generales de reescritura de grafos.

2.7 Redes de interacción

Las redes de interacción (INs) son un modelo de reescritura de gráficos introducido por Lafont , inspirado en los Proof-nets de la Lógica lineal multiplicativa, que consisten en agentes y aristas (número de argumentos necesarios para que dicha función pueda calcular), vinculados entre sí a través de sus puertos.

Las redes de interacción son la instancia de un sistema de reescritura abstracto, y por lo tanto sus propiedades se estudian utilizando la terminología estándar de ese campo., las redes de interacción poseen propiedades de localidad y confluencia que permiten que todos los pares activos de una red determinada se reduzcan simultáneamente, sin interferencias.

Los agentes se representan gráficamente de la siguiente manera, donde indicamos el puerto principal con una flecha de acuerdo con Lafont; nótese que los puertos están ordenados.

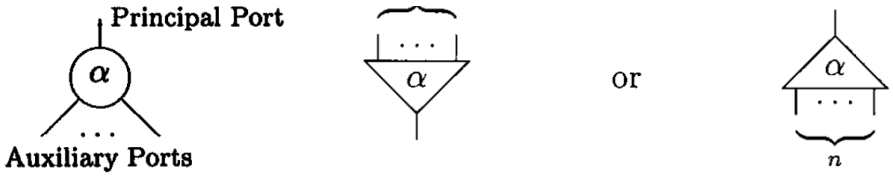


ilustración 2.3 orden de puertos en agentes

Un agente individual es una instancia de un símbolo particular que se caracteriza por su nombre α y su aridad $n \geq 0$. La aridad define el número de puertos auxiliares asociados a cada agente, además de los puertos auxiliares, un agente posee un puerto principal.

Los agentes se escriben como constructores algebraicos con aridad igual al número de puertos auxiliares. Los pares activos son entonces ecuaciones, igualdades entre términos con variables, donde un cable que une dos hojas (dos puertos auxiliares) en dos términos de este tipo (árboles) está representado por dos ocurrencias de la misma variable, las variables están

permitidas como miembros en las ecuaciones, para permitir descripciones modulares, cada variable aparece exactamente dos veces en la red.

Todos los tipos de agentes pertenecen a un conjunto Sigma llamada firma, una red construida sobre Σ es un grafo que tiene agentes como nodos, donde aristas del grafo están conectadas a puertos en los agentes, de manera que hay como máximo una arista conectada a cada puerto de la red. Las aristas pueden estar conectadas a dos puertos del mismo agente. Los puertos principales de los agentes se representan con algún signo distintivo, como una flecha.

La interfaz de una red es el conjunto (ordenado) de los extremos libres de las aristas (en particular, para una red que consta sólo de una arista, la interfaz contiene los dos extremos de la arista).

Un sistema de red de interacción se especifica dando un conjunto Σ de símbolos, y un conjunto R de reglas de interacción, cada símbolo $\alpha \in \Sigma$ tiene una aridad (fija) asociada aridad

La aparición de un símbolo $\alpha \in \Sigma$ se llamará agente, si la aridad de α es n entonces el agente tiene n+1 puertos: uno distinguido llamado el puerto principal representado por una flecha, y n puertos auxiliares etiquetados como x1,...,xn correspondiente a la aridad del símbolo.

Un par activo es cualquier par de agentes (α,β) en una red, con una arista que conecta sus puertos principales, si una red no contiene ningún par activo, decimos que no tiene forma normal

Hay dos instancias especiales de una red: un cableado (una red que no contiene agentes, sólo aristas entre puertos libres), y la red vacía (que no contiene agentes ni aristas).

Una red de interacción que consiste únicamente en bordes se denomina cableado y generalmente se denota como omega . Un árbol t con su raíz x se define inductivamente como una arista x, o como agente alpha con su puerto principal libre x y sus puertos auxiliares x_{i} conectado a las raíces de otros árboles t_{i}.

2.7.1 El cálculo de interacción ligero

Proporciona una semántica precisa para las redes de interacción. Maneja la aplicación de reglas, así como el recableado y la conexión de puertos y agentes.

Utiliza los siguientes componentes:

Símbolos Σ que representan agentes, denotados por α,β,γ.

Nombres N que representan puertos, denotados por x,y,z,x1,y1,z1,.... . Denotamos secuencias de nombres por x,y,z.

Los términos T son nombres o símbolos con un número de subterráneos, correspondientes a la aridad del agente: $t = x \mid \alpha(t1,...,tn)$. s,t,u denotan términos, s,t,u denotan secuencias de términos.

Ecuaciones E denotadas por t = s donde t,s son términos, representando conexiones en una red. Obsérvese que t = s es equivalente a s = t. Δ,Θ denotan conjuntos múltiples de ecuaciones.

Configuraciones C que representan una red por $ht \mid \Delta i$. t es la interfaz de la red, es decir, sus puertos que no están conectados a un agente, todos los nombres de una configuración aparecen como máximo dos veces, estos se denominan vinculados.

Reglas de interacción R denotadas por $\alpha(x) = \beta(y) \dashrightarrow \Theta$. α,β es el par activo del lado izquierdo (LHS)de la regla y el conjunto de ecuaciones Θ representa el lado derecho (RHS).

Decimos que un conjunto de reglas de cálculo de interacción R no es ambiguo si se cumple lo siguiente

- para todos los pares de símbolos (α,β), existe como máximo una regla $\alpha(x) = \beta(y) \dashrightarrow \Theta$ o $\beta(y) = \alpha(x) \dashrightarrow \Theta \in R$.

- si un agente interactúa consigo mismo, es decir, $\alpha(x) = \alpha(y) \dashrightarrow \Theta \in R$, entonces Θ es igual a Δ (como conjuntos múltiples, módulo orientación de las ecuaciones), donde Δ se obtiene de Θ intercambiando todas las ocurrencias de x e y.

La reescritura de una red se modela aplicando cuatro reglas de reducción a una configuración con respecto a un conjunto determinado de reglas de interacción R.

Las cuatro reglas de reducción del cálculo ligero se definen como:

Comunicación: h t | x = t,x = u,Δi com -⟶ h t | t = u,Δi

Sustitución: h t | x = t,u = s,Δi

sub -⟶ h t | u[t/x] = s,Δi, donde u no es un nombre y x ocurre en u.

Recoger h t | x = t,Δi col -⟶ h t[t/x] | Δi, donde x ocurre en t.

Interacción h t | α(t1) = β(t2),Δi int -⟶ h t | Θ′

,Δi, donde α(x) = β(y) -⟶ Θ ∈ R. Θ′ denota Θ donde

todos los nombres vinculados en Θ reciben nombres nuevos y x,y se sustituyen por t1,t2.

Las reglas de reducción com -⟶ y sub

-⟶ sustituyen los nombres por términos: esto resuelve explícitamente las conexiones entre agentes que son generadas por las reglas de interacción. col-⟶ también sustituye los nombres, pero sólo para la interfaz. Naturalmente, int-⟶ modela la aplicación de las reglas de interacción: una ecuación correspondiente a un LHS se sustituye por las ecuaciones del RHS.

Ejemplo Las reglas de adición de números naturales simbólicos se expresan en el cálculo ligero

ligero como sigue:

+(y,r) = S(x) -⟶ +(y,w) = x, r = S(w) (1)

+(y,r) = Z -⟶ r = y (2)

Sea -⟶ la relación de reducción inducida por las cuatro reglas de reducción y un conjunto de reglas de interacción R. Si R no es ambigua, entonces

-⟶ satisface la confluencia uniforme.

La principal diferencia del cálculo ligero con el calculo de interacción es que la regla de indirección del cálculo de interacción estándar se divide ahora en com -⟶ y sub

-⟶ . Sin embargo, esto no afecta a la propiedad mostrada en todos los pares críticos (es decir, las divergencias críticas de un paso en la reducción de una configuración) pueden unirse en un solo paso.

Es necesario que R no sea ambiguo para evitar el no determinismo en la aplicación de int-⟶ regla, que podría llevar a divergencias no unibles.

2.7.2 Clasificación de redes

Lafont también introdujo una disciplina de tipos para las redes de interacción, utilizando un conjunto de tipos constantes (atom, nat, list-nat,). Para cada agente, los puertos se clasifican entre entrada y salida. un puerto de entrada se le asignará un tipo r- y a un puerto de salida un tipo z+.

Una red está bien tipificada si los puertos de entrada están conectados a puertos de salida del mismo tipo. En lo que sigue siguiente, para simplificar, consideraremos sólo un tipo. En otras palabras sólo distinguiremos entre puertos de entrada (marcados con el signo -) y puertos de salida. (marcados con el signo +).

Por ejemplo, para los agentes $\in$ y δ consideramos las siguientes tipificaciones:

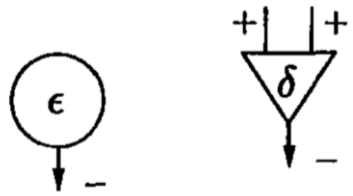


ilustración 2.4 representación agente

Suponiendo que α tiene un puerto principal positivo, las redes de las reglas de interacción anteriores pueden pueden tipificarse de la siguiente manera:

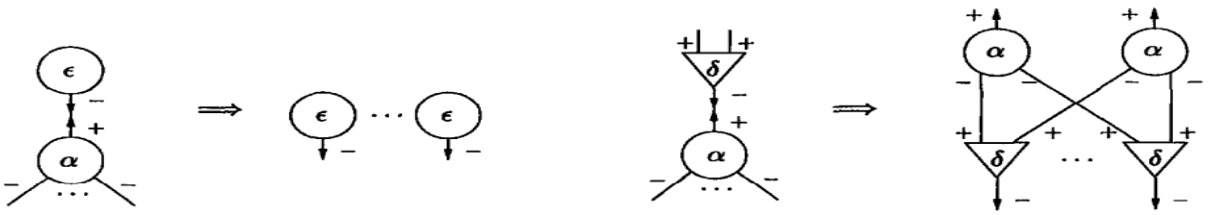


ilustración 2.5 representación de signos en puertos

Los agentes pueden dividirse en constructores y destructores: si el puerto principal de un agente es un puerto de salida, el agente es un constructor, en caso contrario es un destructor.

En el ejemplo anterior, ε y δ son destructores, mientras que α es un constructor, la división entre constructores y destructores tiene su origen en

el sistema lógico que inspiró el formalismo de las redes de interacción: los destructores y los constructores están asociados a las reglas de introducción izquierda y derecha de los operadores lógicos.

Para cada agente, los puertos auxiliares se dividen en particiones (la noción de partición tiene también su origen en el cálculo secuencial que inspiró el formalismo).

Cada agente $\alpha \in \Sigma$ tiene un puerto principal y un conjunto (posiblemente vacío) de puertos auxiliares que se dividen en una o varias clases, cada una de ellas llamada una partición. Un mapeo de particiones establece, para cada agente en Σ, la forma en que sus puertos auxiliares se agrupan en particiones, las particiones dadas por Lafont para ε y δ son las siguientes ε, que no tiene ningún puerto auxiliar, tiene una partición, que está vacía,; para δ ambos puertos auxiliares están en la misma partición (δ tiene una partición que contiene dos puertos de salida).

La noción de partición se introdujo con el propósito de definir una clase de redes de interacción, llamadas redes semi-simples, que son libres de bloqueo, es decir, una clase de redes que no pueden crear círculos viciosos de puertos principales, como los que se muestran en el diagrama siguiente durante el cálculo:

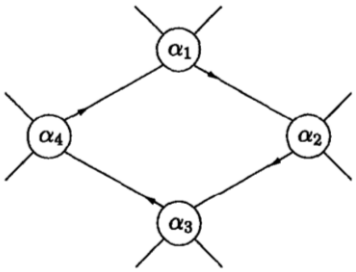


Ilustración 2.6 representación de partición

2.7.3 reglas de interacción para redes.

Las reglas deben cumplir dos condiciones: las interfaces del lado izquierdo y del lado derecho son iguales (lo que implica que los puertos libres se conservan durante la reducción), y hay como máximo una regla para cada par de agentes, por lo que no hay ambigüedad en cuanto a la regla a aplicar.

Un conjunto IR de reglas de interacción que son reglas de reescritura de redes en las que el lado izquierdo es una red formada por dos agentes conectados en sus puertos principales, y el lado derecho es una red arbitraria con la única restricción de que debe tener la misma interfaz que el lado izquierdo. Hay como máximo una regla para cada par de agentes.

El siguiente diagrama muestra la forma general de una regla de interacción, utilizando agentes de aridad 3 y 4, respectivamente, el lado derecho N es una red cualquiera, que puede contener ocurrencias de los agentes del lado izquierdo (representamos las redes con líneas discontinuas).

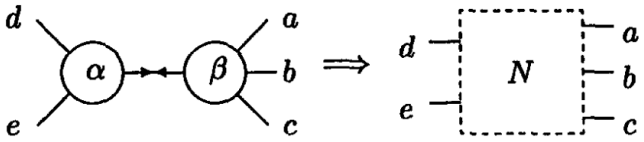


Ilustración 2.7 agentes y puertos

Obsérvese que la interfaz se mantiene; hay el mismo número de puertos libres antes y después de la interacción. Utilizamos nombres (a, b, c,d, e) para indicar la correspondencia entre los puertos libres en los lados izquierdo y derecho de la regla, pero a menudo los omitiremos cuando no haya ambigüedad.

Además de un conjunto de reglas de interacción, las interacciones interdependientes entre los agentes(nodo), tienen que ser consideradas en el modelo de comportamiento individual para una buena coordinación y sincronización de las acciones de los agentes distribuidos.

2.7.4 combinadores de interacción

El primer sistema de combinadores de interacción fue presentado por Lafont, que definió un sistema completo de interacción utilizando ocho agentes. Con una ingeniosa codificación de una red que puede ser duplicada, Lafont consiguió definir un sistema con sólo tres agentes y seis reglas de interacción (imagen x), este sistema es el que utilizaremos a lo largo de este artículo.

Los tres agentes del sistema de Lafont son: γ (un constructor de aridad 2), δ (un duplicador de aridad 2), y ε (un borrador de aridad 0), que dibujamos de la siguiente manera: Cada red que construyamos se construirá conectando ocurrencias de estos agentes entre sí. se mostrará cómo este sistema de combinadores puede utilizarse para codificar tanto la eliminación de cortes en la lógica lineal como la β-reducción en el λ-cálculo.

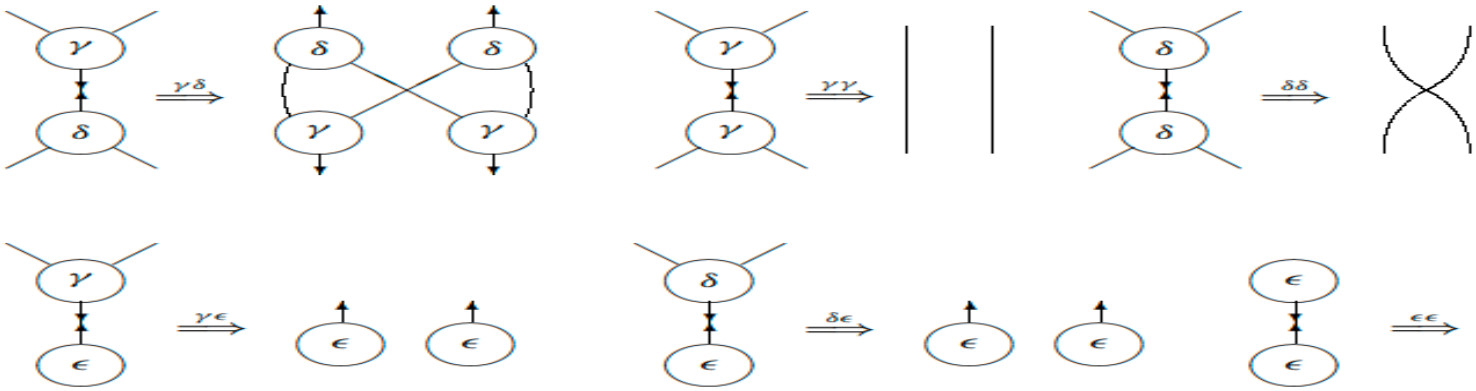
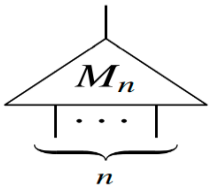


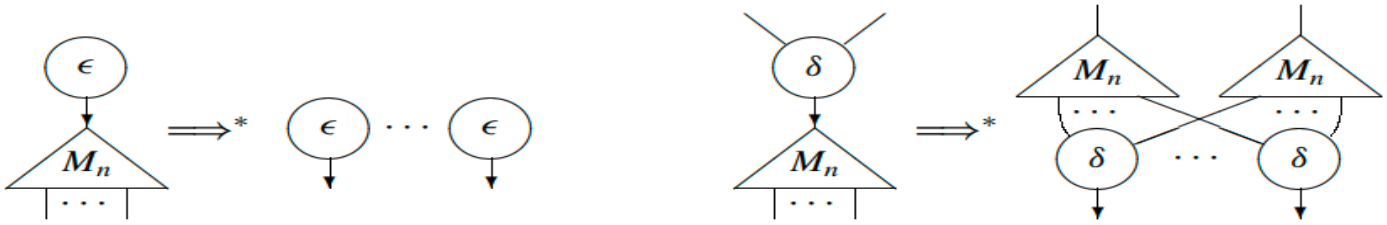
ilustración 2.8 reglas de interacción

Multiplexación. Será útil para la traducción, y para el razonamiento sobre las redes, proporcionan algunas abreviaturas (macros) que se construyen a partir de los combinadores, el agente constructor γ puede utilizarse como un agente multiplexor binario: agrupa las dos aristas de los puertos auxiliares en una arista del puerto principal. Simétricamente, γ también puede utilizarse como agente demultiplexor, y la regla de interacción γγ puede entonces entenderse como la eliminación de la arista compartida entre los agentes. Esta idea puede generalizarse a las redes de multiplexación n-arias como explicamos a continuación.



Ilustracion 2.9 representación de multitud

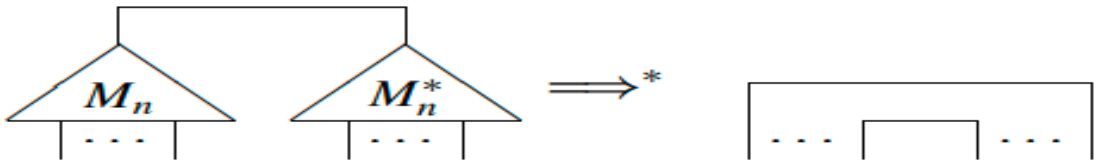
Una red Mn, que agrupa n aristas en una sola, dibujada de la siguiente manera:



Ilustracion 2.10 red de multiplexación

Es una red de multiplexación si se puede borrar con ε y duplicado con δ:

Nótese que para que estas dos propiedades se mantengan la red Mn debe estar libre de agentes δ y también libre de pares activos y bloqueos.
Un par de redes (Mn, M∗ n) es un par multiplexor cuando tanto Mn como M∗ n son redes de multiplexación
redes que además satisfacen la siguiente propiedad:



Ilustracion 2.11 par de redes de multiplexación

Para este par de redes, diremos que M∗ n es la red de demultiplexación de Mn.
Las redes de multiplexación: Para todo n ≥ 0, Mn puede construirse sin usar δ o .
Los pares de multiplexación: Para todo n ≥ 1, (Mn, M∗
n) pueden construirse sin utilizar δ o . Para n = 0,
(Mn, M∗n) puede construirse sin usar δ.

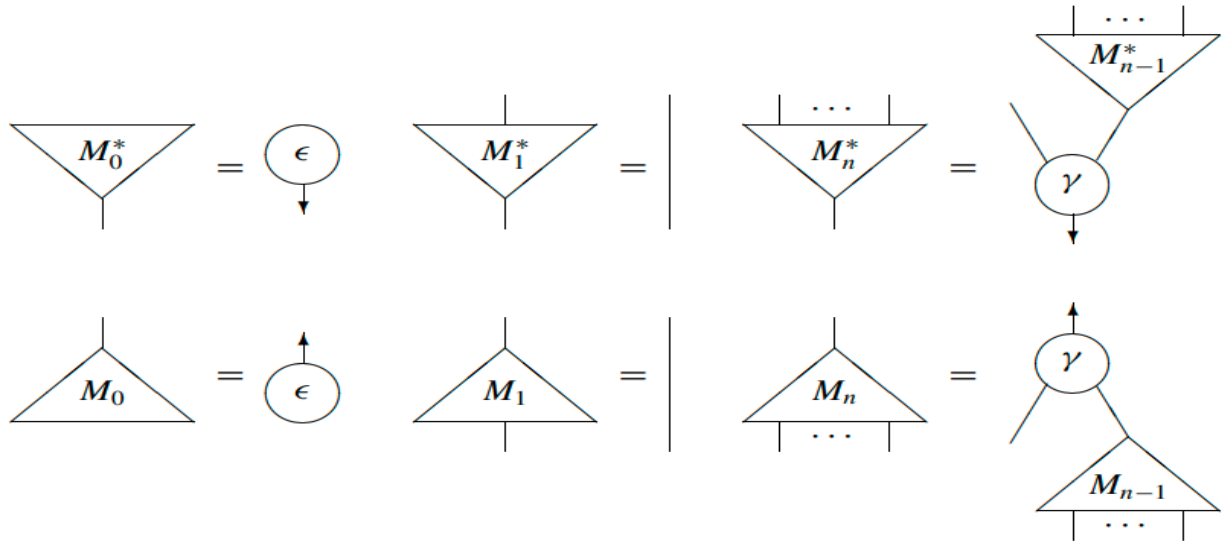


ilustración 2.12 Construcción de redes de multiplexación.

Capítulo 3 Implementación de redes concurrentes

3.1 Programas de Post–Turing

La búsqueda de un modelo abstracto de computación capaz de lidiar con una clase más amplia de algoritmos fue un tema de investigación activo durante los primeros años, los candidatos propuestos exploraron una amplia variedad de enfoques y características, y muchos han tenido un impacto duradero en el campo de la informática.
Ejemplos incluyen:

- El cálculo lambda de Alonso Church, modelo simple para lenguajes de programación funcionales;
- La lógica combinatoria de Schonfinkel y Curry, un modelo de programación funcional sin variables;
- La teoría de las funciones recursivas de Kleene;
- Los sistemas de reescritura de cadenas de Post y otros, en los que el estado del sistema almacenado como una cadena de símbolos evoluciona bajo un conjunto de reglas de reescritura.

Cada uno de estos modelos tenía como objetivo capturar un amplio conjunto de algoritmos prácticos, incluidos aquellos cuyos requisitos de tiempo y espacio eran funciones ilimitadas de sus datos de entrada.
A medida que se propusieron y compararon nuevos formalismos, hubo una clara necesidad de un punto de referencia para determinar su poder relativo como modelo general de cálculo. Lo que surgió como una medida estándar de la potencia de un modelo es el conjunto de funciones enteras soportadas por ese modelo: un modelo que podía soportar la suma y multiplicación de enteros arbitrarios, por ejemplo, era más poderoso que un modelo alternativo que podía sumar pero no multiplicar.

Consideremos el siguiente lenguaje de programación (formal), T, para el cálculo con palabras sobre un alfabeto finito Σ (aunque el alfabeto de trabajo del lenguaje contendrá un símbolo extra B ∈/ Σ).

· Un programa de T, es una sucesión finita de instrucciones I1, . . ., In.
· Instrucciones (algunas pueden estar etiquetadas):
o Para cada símbolo s ∈ Σ ∪ {B} y cada etiqueta L, la instrucción IF s GOTO
o Para cada símbolo s ∈ Σ ∪ {B}, la instrucción PRINT s
o Instrucciones de desplazamiento: RIGHT y LEFT

· Un programa controla un dispositivo que inspecciona una cinta infinita por ambos extremos, dividida en celdas.
· En cada momento la cabeza lectora puede leer el contenido de una celda (un símbolo de Σ ∪ {B}).
· El dispositivo puede sobrescribirlo (PRINT), moverse a una celda adyacente (RIGHT, LEFT) o cambiar de instrucción (IF s GOTO L).
Para calcular una función f: Σ∗× (n) · · · ×Σ ∗− → Σ ∗, dada la n–tupla (v1, . . ., vn) escribimos en la cinta
B v1 B v2 B · · · B vn B
con la cabeza lectora en la primera celda de la cadena.

· El programa ejecuta secuencialmente cada instrucción pasando a la siguiente.
· En el caso de una instrucción IF s GOTO A, si la cabeza lectora lee el símbolo s, se pasa a la primera instrucción etiquetada por A (si existe). En caso contrario se pasa a la siguiente instrucción (si existe).
· El programa se detiene si el proceso de ejecución descrito le envía a una instrucción inexistente.
· Si f (v1, . . ., vn) = x, al detenerse el programa, el contenido de la cinta debe ser Bx y la cabeza lectora debe estar en la celda marcada con B.

Dependen de la semántica y del concepto de solución.

1. Modo secuencial-determinista: La ejecución de un programa sobre un dato es única. La solución de un problema es la solución lógica exacta para cada entrada.

2. Modo paralelo: El programa especifica diversas tareas que se pueden ejecutar en distintos procesadores, compartiendo información. Solución exacta.

3. Modo no determinista. La ejecución no es única: existen instrucciones que pueden elegir entre 2 o más opciones. I Un programa resuelve un problema si alguna de sus posibles ejecuciones ofrece una solución exacta para el dato de entrada.
4. Modo probabilista: Los programas son deterministas, pero contienen instrucciones que se ejecutan sólo con cierta probabilidad. Un programa resuelve un problema si, con cierta probabilidad, obtiene una solución exacta.

3.2 INblobs

Blobs es un editor visual para grafos dirigidos, implementado en Haskell utilizando la biblioteca wxHaskell. Blobs fue producido por un equipo que integraba a los autores de Dazzle, un editor de grafos bayesianos, que se dieron cuenta de que su front-end podría ser útil para otros proyectos Haskell completamente diferentes. Este front-end ha sido extraído de Dazzle, y puesto a disposición con algunas características añadidas de la comunidad de programación funcional.
Citando a los autores, "Blobs es un front-end para dibujar y editar diagramas gráficos". El editor descrito en el presente artículo está construido sobre Blobs y proporciona funcionalidades adicionales al editor front-end, relativas a la edición de redes de interacción en particular la presencia de puertos en los nodos, y también funcionalidades back-end - edición del sistema de interacción, generación de descripciones textuales/configuraciones, y reducción de redes de interacción.
INblobs hereda todas sus capacidades de edición de Blobs, en particular la colocación de nodos y aristas mediante apuntar y hacer clic; la posibilidad de seleccionar un subgrafo con el ratón y realizar acciones sobre él; acciones de deshacer/rehacer; y guardar/cargar el estado actual de la aplicación a/desde un archivo.

3.2.1 Cómo usar INblobs

Seleccione un símbolo de agente presionando su botón en el panel izquierdo (paleta de símbolos).
Haga clic con el botón derecho (o presione Ctrl y haga clic) en un lienzo, nodo o borde para acceder a un menú contextual.
Para crear un nodo, haga clic con la tecla Mayús en un lienzo en blanco.
Para crear un borde, seleccione (haga clic) el puerto de origen, luego presione Mayús y haga clic en el puerto de destino.
Para eliminar un nodo o borde, selecciónelo y presione RETROCESO o SUPR, o use el menú contextual.
Para reorganizar el diagrama, haga clic y arrastre los nodos a donde los desee.
Para que un borde se vea más ordenado, agregue un punto de control desde su menú contextual y arrastre el punto a donde lo desee.
Puede agregar varios elementos a la selección actual haciendo meta-clic en los nodos y puntos de control adicionales. (Meta = tecla Apple o tecla CONTROL). Una selección múltiple se puede arrastrar o eliminar como una única selección.

La interfaz de una red o una regla se define explícitamente mediante agentes de interfaz especiales.
La red de la parte inferior es la red que se va a reducir o convertir a una configuración textual.
Las dos redes en la parte superior son el lado izquierdo y el lado derecho de la regla de interacción seleccionada actualmente (de la lista de reglas a la izquierda).
Para agregar una nueva regla, presione el botón Agregar nueva regla o haga clic con el botón derecho del mouse en Reglas (la raíz del árbol de reglas). Luego agregue agentes al lienzo en la parte superior.
Alternativamente, utilice el asistente de creación de reglas. Al presionar este botón aparecerá un cuadro de diálogo donde se pueden elegir los dos agentes que interactúan.
El asistente generará automáticamente el lado izquierdo de la regla. También es posible elegir lo que se genera en el lado derecho de la regla:
una copia del lado izquierdo para editar manualmente después (útil para reglas con lados similares); los agentes de interfaz del lado izquierdo; nada (genera un lado derecho en blanco, no recomendado)

Para las reglas entre uno de los símbolos {& epsilon, & delta, c} y cualquier símbolo, existen plantillas que crean automáticamente la regla completa; en el asistente de creación de reglas, marque la opción Plantilla de reglas y elija los dos símbolos.
Haga coincidir los agentes de interfaz en una regla seleccionando el agente de interfaz deseado en el lado izquierdo y presionando Mayús y haciendo clic en el agente de interfaz correspondiente en el lado derecho, aparecerá un cuadro con el mismo número en ambos agentes.

Las etiquetas de borde y nodo se pueden hacer visibles seleccionando el comando apropiado en el menú Ver, las etiquetas de borde son útiles porque facilitan la selección de bordes: para seleccionar un borde simplemente haga clic en su etiqueta.

3.2.2 Descripción de la herramienta

El usuario debe dar un nombre y una lista de puertos, esta operación tiene efectos a nivel del sistema de interacción (correspondiente a incluir en él una nueva declaración de símbolos), pero también a nivel geométrico nivel, ya que las coordenadas de los puertos en los agentes correspondientes también deben ser dado. La representación del agente es un objeto ovalado cuya longitud se extiende hasta acomodar su nombre en el interior. Todo esto se

hace en el Asistente para crear nuevos símbolos que es accesible desde la ventana principal a través de un botón.

El usuario también puede cargar una paleta de formas diferente, lo que permite un objeto visual alternativo representación de agentes (las paletas se pueden diseñar y programar con wxHaskell código)

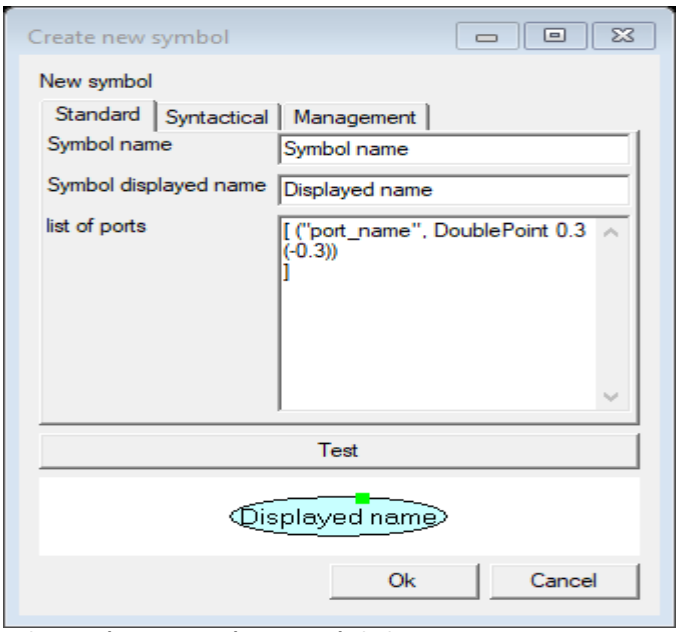
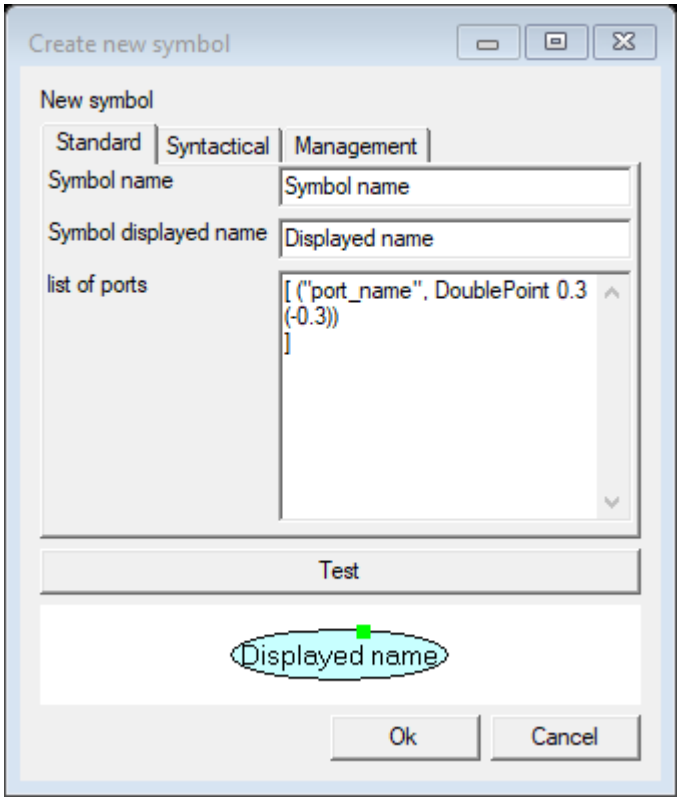


Ilustración 3.1 creación nuevo símbolo

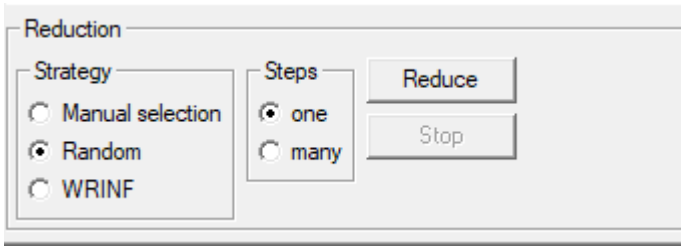
Una característica muy útil del asistente para la creación de reglas es la capacidad de definir reglas desde plantillas predefinidas. Ciertos agentes tienen patrones fijos de interacción; Ejemplos de estos incluyen los agentes de borrador y duplicador. Por ejemplo, un paso de interacción

Entre un agente duplicador y cualquier otro agente α con N puertos auxiliares genera dos agentes α y n duplicadores. Este patrón se codifica como una plantilla que puede ser utilizado en el asistente: basta con seleccionar α y el duplicador para la regla esperada para ser creado, cualquiera que sea la aridad de α. La figura x muestra una regla creada por la plantilla referida.



Ilustracion 3.2. Plantilla de reglas

El usuario puede optar por reducir el par activo seleccionado actualmente; un azar par activo; o bien el par activo correspondiente a una de las estrategias implementadas en la herramienta, en cuyo caso, esas estrategias se enumerarán en el área reducción de la ventana principal.



Ilustracion 3.3. Controles de reducción.

Por el momento las estrategias están codificadas en la herramienta, pero dicha codificación es relativamente fácil y tan nuevas estrategias de IN, por ejemplo, que archivar formas canónicas más rápido o mantener las redes más pequeñas, se puede agregar a INblobs por solicitud a sus mantenedores.

La reducción débil a la estrategia normal de la forma de la interfaz [1.7] es tal caso. El usuario alternativamente, puede reducir la red, de acuerdo con cualquiera de las estrategias disponibles, en muchos pasos a la forma normal, teniendo la posibilidad de detener la evaluación en cualquier hora

INblobs ha sido diseñado para que se pueda editar simultáneamente un sistema de redes de interacción y una red de interacción, hay restricciones naturales que resultan directamente del formalismo: la red actual sólo puede utilizar agentes que están definidos en el sistema de interacción actual, y para que un par activo se reduzca, debe incluirse en el sistema una regla de interacción que coincida.

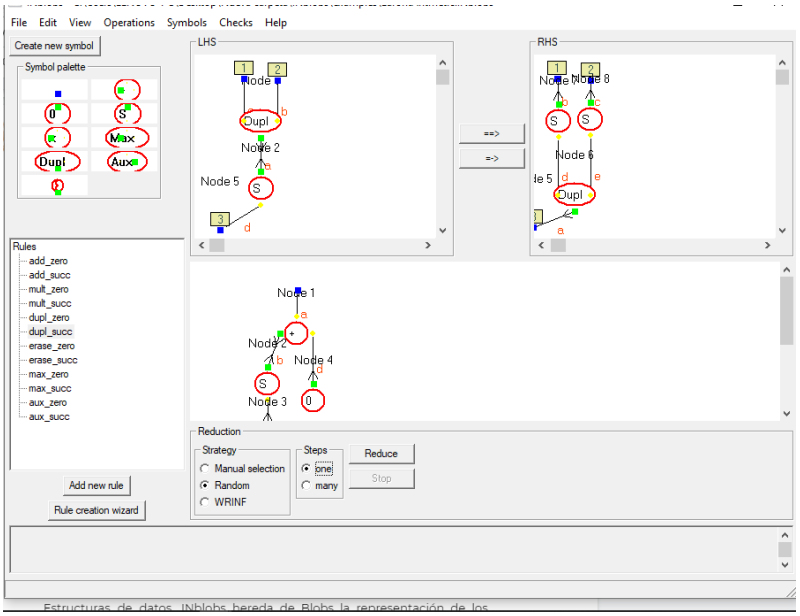


Ilustración 3.4 Interfaz INblobs

La figura muestra la ventana principal de la herramienta, a la izquierda se encuentra la paleta de símbolos, junto con la lista de reglas de interacción, el área del botín de la derecha muestra la red de interacción actual, y en la parte superior se muestra la regla de interacción actualmente seleccionada.

3.3 Estructuras de datos.

INblobs hereda de Blobs la representación de los grafos como pares de intmaps, que se adaptan para contener información específica de las redes de interacción. El intmap del agente asocia a cada nodo, además de la misma información que en Blobs, información relativa a sus puertos tanto simbólica, como si el puerto es el principal o no, y geométrica, como la ubicación de los puertos en el nodo. El intmap de aristas también asocia a cada arista una nueva información, a saber, los puertos a los que está conectada en cada agente.
Edición básica de redes. Las redes de interacción pueden editarse en el panel inferior derecho de la ventana principal de la herramienta, y en el panel superior como parte de una regla de interacción. Recordemos que la interfaz de una red está formada por todos sus puertos libres, es decir, los que no tienen ninguna arista conectada.

Este enfoque permite una identificación clara y explícita de la interfaz. Observamos que la forma más habitual de representar visualmente los puertos libres no puede utilizarse, ya que no hay forma de dibujar una arista colgante en Blobs, los agentes (incluidos los agentes de interfaz) se crean a partir de la paleta de símbolos actual; los bordes se añaden entonces mediante simples operaciones de apuntar y hacer clic, toda la funcionalidad de edición se hereda de los Blobs, excepto la existencia de puertos en los agentes, que no es una noción estándar para los grafos dirigidos.
Añadir una arista a una red implica seleccionar sucesivamente los puertos de origen y destino de la arista. También se pueden seleccionar nodos y aristas, un menú contextual muestra las operaciones de edición de cada uno de estos elementos de una red. Los puertos seleccionados se muestran en un color diferente, y los nodos y aristas seleccionados se muestran más gruesos.

3.4 Lógica INblobs

Existe una correspondencia de uno a muchos entre las redes de interacción y las configuraciones del cálculo para redes de interacción. En particular, existe un número mínimo de ecuaciones para representar una red concreta, que viene dado por el número de pares activos de la red, pero pueden obtenerse otras representaciones introduciendo variables adicionales; además, las configuraciones se consideran módulo de conversión alfa (los nombres de las variables siempre pueden sustituirse por otros nombres nuevos).

La herramienta presentada convierte las redes en configuraciones del cálculo mediante el siguiente algoritmo sencillo:

1. Etiqueta cada *borde* de la red con un nombre nuevo.
2. Para cada *agente* de aridad *n* en la red, escribe una ecuación de la forma $y = \alpha(x1, \ldots, xn)$, donde *y* es la etiqueta de la arista conectada al puerto principal de α, y $x1, \ldots, xn$ son las etiquetas de las aristas conectadas a sus puertos auxiliares 1 a *n*. Sea Δ el subconjunto formado por estas ecuaciones.
3. Se toma el subconjunto de nombres apropiado para la interfaz **t** de la red. Se obtiene así una configuración inicial $\mathbf{t}\,\Delta$.
4. Mientras exista en Δ una ecuación $\langle\,|\rangle$ de la forma $x = u$, con *x* a variable y x ($\Delta\, x = u$), aplica la regla de la **Indirección** del cálculo para eliminar esa ecuación.
5. Mientras exista en Δ $\in$ en una ecuación de la forma $x = u$, $\in$ con *x* como variable y *x* (**t**), aplica la regla de la **Colecta** del cálculo para eliminar esa ecuación.

La configuración resultante es mínima en el sentido de que contiene exactamente una ecuación para cada par activo de la red. Como alternativa, la herramienta también puede dar salida a la configuración $\mathbf{t}\,\Delta$ obtenida tras el paso 3 del algoritmo.

El orden en el que se generan las ecuaciones (paso 2) viene dado por los índices de los agentes, que a su $\langle$ vez dependen del orden en el que se colocaron los agentes en la red. Las operaciones geométricas no modifican los índices ni el orden de las ecuaciones.
Como ejemplo, la red de la figura 3 daría lugar, después del paso 3, a la configuración

$$\langle a,\ b \mid y = 0,\ z = S(y),\ z = +(a, x),\ u = 0,\ v = S(u),\ v = +(x, b)\rangle$$

que luego se simplificaría en los pasos 4 y 5 a

$$\langle a,\ b \mid S(0) = +(a, x),\ S(0) = +(x, b)\rangle$$

Se utiliza una versión ligeramente modificada del mismo algoritmo para convertir las reglas de interacción en su descripción textual. En este caso es obligatorio utilizar la versión simplificada que se obtiene aplicando sucesivamente la regla de indirección, ya que las reglas se escriben

como ecuaciones simples.

```
agentes
        Z0;
        S1;
    A2;
                reglas
                    A
                    (
                    x
                    ,
                    x
                    )
                    >
                    <
                    Z
                    ;
                    A
                    (
                    x
                    ,
                    S
                    (
                    y
                    )
                    )
                    >
                    <
                    S
                    (
                    A
                    (
                    x
                    ,
                    y
                    )
                    )
                    ;

                red
                    S(Z) = A(a,x);
                    S(Z) = A(x,b);

interfaz
    a;
    b;

fin
```

Tabla 3.1. Descripción textual generada por INblob

- La utilidad de la conversión de redes a configuraciones textuales radica, en la posibilidad de utilizar la herramienta como editor visual de redes de interacción, pueden exportarse en archivos de texto a otras herramientas, la herramienta utiliza la sintaxis concreta. Ésta es muy parecida a las configuraciones descritas anteriormente, pero permite describir la red de interacción y el sistema de redes de interacción en el mismo archivo. La sintaxis puede modificarse fácilmente para adaptarse a otras representaciones concretas.

- La herramienta genera descripciones textuales directamente en la ventana del editor, o bien las escribe en un archivo especificado por el usuario. Ambas opciones son accesibles desde el menú "Operaciones". Se ofrece la opción de generar una descripción conjunta de la red y el sistema, o bien una descripción del sistema, o sólo de la configuración de la red. La tabla 1 muestra un fichero generado por la herramienta, que contiene nuestra configuración de red de ejemplo, junto con la descripción del sistema de red de interacción para la adición de números naturales, esto ilustra la sintaxis concreta generada por INblobs.

3.5 Reducción de las redes de interacción

La forma más obvia de implementar la reducción de la red de interacción es mantener una lista de pares activos. Cada paso de reducción corresponde a la elección de un par de esta lista y a la búsqueda de una regla coincidente en la lista de reglas de interacción. Otra posibilidad es mantener una lista de ecuaciones, como en el cálculo de las redes de interacción, lo que permite un control más estrecho de las operaciones jerárquicas de "recableado". En cualquier caso, la estructura de datos adecuada para representar una red es una lista de pares de árboles, donde cada árbol corresponde a un término del cálculo.
En INblobs, la red de interacción actual puede reducirse dentro de la herramienta. Cabe destacar que, en aras de la flexibilidad, es posible alternar los pasos de reducción con los de edición, tanto en el sistema de redes de interacción como en la propia red que se está reduciendo). Una consecuencia importante de este hecho es que la reducción debe funcionar en la representación subyacente de la red utilizada por el editor visual.
Suponemos que el par activo ha sido seleccionado por el usuario o aleatoriamente por la herramienta; se identifica por un índice entero n, correspondiente a la arista que conecta el par de agentes.

1. Obtener del intmap de la arista (con argumento n) la información relativa a los agentes x, y donde se conecta la arista n, y también los puertos de x, y donde se conecta.
2. Consultar el nodo intmap (con argumentos x e y) para comprobar que los dos puertos son principales; obtener los símbolos correspondientes de los agentes.
3. Busca en la lista de reglas de interacción la regla que coincida con los dos símbolos relevantes. Recordemos que cada lado de la regla está representado por un par de int maps para sus nodos y aristas.

4. Integrar en la red actual una copia R del lado derecho de la regla. Esto se hace añadiendo a los dos intmaps la información de los intmaps de R, después de actualizar los índices en R (todos los índices deben ser frescos con respecto a la red actual). Esta copia está por ahora desconectada de la red.
5. Sustituir el par activo por R: para cada arista que estaba conectada al par activo en la red (excepto n), conectarla en su lugar al puerto correspondiente en R. Se trata de una serie de operaciones sobre los intmaps,que consiste en hacer coincidir los agentes de la interfaz a la izquierda y a la derecha de la regla.
6. Limpiar: eliminar de los intmaps la información relativa al par activo que se acaba de reducir, los agentes de interfaz en R y las aristas conectadas a ellos.

Hay tres modos de reducción disponibles como botones en la ventana principal. El usuario puede reducir el par activo seleccionado en ese momento o bien reducir un par activo seleccionado al azar; la tercera posibilidad es hacer un bucle con este último paso hasta que se alcance una red de forma normal, esto no puede interrumpirse, por lo que el usuario es responsable de garantizar la terminación de las reducciones de la red.

Después de la reducción, la herramienta no intenta dibujar la red de forma inteligente, cada paso de reducción da lugar a una red en la que los nodos pueden estar superpuestos y las aristas pueden cruzarse, el usuario es completamente responsable de "ordenar" la salida después de cada paso de reducción o una serie de pasos.
Dado que los pasos de interacción son locales, la cantidad de trabajo que supone cada paso es mínima nos parece que esta opción es apropiada ya que la red inicial también es dibujada por el usuario, y normalmente el trazado corresponde a alguna interpretación de la red dependiente de la aplicación, que sólo el usuario es capaz de restaurar.

Conclusión

Al principio resultaba difícil de comprender las redes de interacción pues este modelo computacional recurrente, quiere decir que el modelo define una serie de eventos que desencadenan transiciones de un estado a otro para cada una de las actividades, acciones o tareas, este modelo es muy peculiar ya que involucra teoría de otros modelos computacionales, que aunque todos comparten la misma teoría, sus implementaciones son muy variadas y aplicables a otras áreas.

Lo que se intento en este documento fue la comprensión de las redes de interacción como modelo computacional, un modelo que recibe parámetros, esta regulado por reglas de interacción y que se obtiene uno o multiples resultados de ella, en este caso usamos el editor para redes para representar las reglas, transiciones y agentes involucrados y como estos en conjunto trabajan.

Bibliografía

Jos´e Bacelar Almeida, Jorge Sousa Pinto & Miguel Vilac¸a (2007): A Tool for Programming with Interaction Nets. In: Joost Visser & Victor Winter, editors: Proceedings of the Eighth International Workshop on Rule-Based Programming. Elsevier. To appear in Electronic Notes in Theoretical Computer Science.

Richard Barker: Case*Method: Entity Relationship Modelling. Addison-Wesley.

Peter Borovansky, Claude Kirchner, H´el`ene Kirchner, Pierre-Etienne Moreau & Christophe Ringeissen (1998): An Overview of ELAN. In: H´el`ene Kirchner, Claude & Kirchner, editor: Second Workshop on Rewriting Logic and its Applications - WRLA'98 Electronic Notes in Theoretical Computer Science, Electronic Notes in Theoretical Computer Science 15. Elsevier Science B. V., Pont- `a-Mousson,France, p. 16 p. Available de http://hal.inria.fr/inria-00098518/en/. Colloque avec actes etcomit´e de lecture.

Maribel Fern´andez & Ian Mackie (1999): A Calculus for Interaction Nets. In: Proceedings of PPDP'99,Paris, number 1702 in Lecture Notes in Computer Science. Springer.

[5] G. Gentzen (1969): Investigations into Logical Deduction. In: M. E. Szabo, editor: The Collected Papers of Gerhard Gentzen. North-Holland.

Jean-Yves Girard (1987): Linear Logic. Theoretical Computer Science 50(1), pp. 1–102.

T. Hammond (2007). LADDER: A Perceptually-Based Language to Simplify Sketch Recognition User Interfaces Development. MIT PhD Thesis.

Abubakar Hassan, Ian Mackie & Jorge Sousa Pinto (2008): Visual Programming with Interaction Nets. In: Gem Stapleton, John Howse & John Lee, editors: Diagrammatic Representation and Inference, 5th International Conference, Diagrams

H´el`ene Kirchner & Oana Andrei (2007): A Rewriting Calculus for Multigraphs with Ports. Proc.8th Int. Workshop on Rule-Based Programming (RULE07) .

Yves Lafont (1990): Interaction Nets. In: Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90). ACM Press, pp. 95–108.

Sylan Lippi (2002): in2: A Graphical Interpreter for Interaction Nets. In: RTA '02: Proceedings of the 13th International Conference on Rewriting Techniques and Applications. Springer-Verlag,, UK, pp. 380–386.

J.R. Ullman (1976): An Algorithm for Subgraph Isomorphism. Journal of the ACM 23(1), pp. 31–42.

Eelco Visser (2001): Stratego: A Language for Program Transformation based on Rewriting Strategies. System Description of Stratego 0.5. In: A. Middeldorp, editor

https://www.researchgate.net/publication/222033040_Compilation_of_Interaction_Nets

Wikipedia contributors. (2019, 13 diciembre). *Interaction nets*. Wikipedia. https://en.wikipedia.org/wiki/Interaction_nets

El Modelo Del Agente Inteligente Sistemas Multiagentes Sistemas Ubicuos - Noticias Modelo

Kslolgan. (2016, 23 noviembre). *Logica Lineal* [Vídeo]. YouTube. https://www.youtube.com/watch?v=PdWlLlpkuck

Introducción a los agentes y sistemas multiagentes - Departamento de informática Universidad de Valladolid. MAS.PDF (uva.es)

Arleinginformatica, V. T. L. E. (2016, 25 julio). *Paralelismo Multiprocesamiento*. wikiparalelismomultiprocesador. https://wikiparalelismomultiprocesador.wordpress.com/2016/07/25/paralelismo-multiprocesamiento/