Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

Posgrado en Ciencia e Ingeniería de la computación

Un Estudio de Algoritmos para el Problema de Acuerdo de Retícula en un Sistema síncrono

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

M A E S T R O E N C I E N C I A E I N G E N I E R Í A D E L A C O M P U T A C I Ó N

P R E S E N T A:

H E N R Y M A R T Í N E Z B E L L O

Director de tesis:

Dr. Sergio Rajsbaum Gorodezky

Instituto de Matemáticas

Ciudad Universitaria, Cd. Mx., Febrero 2022

Agradecimientos

Agradezco al Dr. Sergio Rajsbaum por permitirme trabajar con él, por su tiempo, sus aportes y enseñanzas para poder concluir este trabajo.

A los doctores Armando Castañeda, David Flores, Jorge Ortega y Manuel Alcántara, por tomarse el tiempo para otorgarme comentarios que enriquecieron este trabajo.

A mis hermanos y padres, que siempre me han apoyado y al mismo tiempo confiado en mí.

A mis amigos y compañeros por el apoyo, en especial a Victor Sánchez por sus comentarios a este trabajo.

A Monse, por apoyarme y estar a mi lado en este trayecto.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico que me permitió realizar estos estudios de posgrado.

Índice general

Un Estudio de Algoritmos para el Problema de Acuerdo de Retícula en un Sistema Síncrono

por

Henry Martínez Bello

Resumen

En el problema de acuerdo de retícula se tienen n procesos, donde cada proceso p_i tiene un valor de entrada x_i que pertenece a una retícula $\mathcal{L}$ y, después de procesar un algoritmo, el proceso decide por un valor de salida y_i que pertenece a la retícula $\mathcal{L}$. Estos valores de salida deben formar una cadena entre ellos, además, el valor de salida debe estar acotado por abajo por el valor de entrada del proceso y por arriba por el *join* de los valores de entrada de los n procesos ($x_i \leq y_i \leq \sqcup X$). En la tesis se va a estudiar el problema de acuerdo de retícula sobre un sistema síncrono con paso de mensajes, donde pueden ocurrir fallas de paro.

En este trabajo se presentan los análisis de tres algoritmos que resuelven el LAP. El primer algoritmo que se presenta es LA_R con un orden de complejidad lineal respecto al número de fallas, donde se demuestra que su complejidad es justa. Un segundo algoritmo es LA_M, el cual es una homologación al sistema presentado en esta tesis del algoritmo de Marios Mavronicolas, este algoritmo tiene la particularidad de ser de terminación temprana, además, cada proceso decide a lo más en mín$\{1+\mathrm{h}(\mathcal{J}(X)), \lceil(1+\sqrt{8f+1})/2\rceil\}$ rondas. Finalmente, el tercer algoritmo estudiado es LA_α, donde cada proceso decide a lo más en $\log(h(\mathcal{L}))$ rondas.

Se hace un estudio sobre las cotas inferiores de los tres algoritmos para demostrar que las cotas propuestas son justas.

Capítulo 1

Introducción

El problema de acuerdo de retícula (*lattice agreement problem*) introducido por Attiya, Herlihy y Rachman en 1995 [1], es un problema de decisión importante para sistemas distribuidos. Dicho problema es equivalente a resolver el problema de instantáneas atómicas (*atomic snapshots*) en memoria compartida. Referente a la demostración se puede consultar con más detalle en [1].

Se iniciará presentando los conceptos asociados al cómputo distribuido. Se verá como se puede enviar un mensaje de un proceso a otro, así como los conceptos asociados a un modelo síncrono y asíncrono. No olvidemos que un sistema no es perfecto, por lo cual se verán los principales tipos de fallas que ocurren o pueden ocurrir. Finalmente, se mostrará un panorama y los trabajos relacionados al problema de acuerdo de retícula.

1.1. Computación Distribuida

Como describen Maurice Herlihy, Dmitry Kozlov y Sergio Rajsbaum en [6] un sistema distribuido es una colección de entidades de computación secuencial llamados *procesos*, los cuales cooperan para resolver un problema o tarea mediante el envío de mensajes o de una memoria compartida. Una característica es que usualmente los procesos en este tipo de sistemas cuentan con una alta capacidad de cómputo y memoria.

- Los sistemas que utilizan *envío de mensajes*, los procesos involucrados en el sistema envían mensajes a los otros procesos mediante canales de comunicación, incluidos a ellos mismos.
- Los sistemas que hacen uso de *memoria compartida*, los procesos involucrados en el sistema pueden acceder a ella, ya sea para comunicarse entre ellos o para evitar copias redundantes. Existen diferentes submodelos por mencionar algunos: *memoria de lectura y escritura* es donde los procesos comparten un arreglo de ubicaciones de memoria; *variable de un solo escritor* es en la cual puede ser escrito únicamente por el dueño pero puede ser leída por todos los procesos del sistema y, de *multiescritura* donde la variable puede ser modificable por todos los procesos.

Un par de características importantes en estos sistemas se refieren a la presencia de fallas y la incertidumbre del tiempo. En todo sistema existen fallas, motivo por el cual se

crean protocolos para tolerarlas e identificarlas. Las *fallas de paro* son en las que un proceso defectuoso se detiene y permanece de esta manera el resto del protocolo. Sin embargo, una falla de tipo *bizantino*, la cual fue propuesta por Lamport para poder abordar el problema de generales Bizantinos en [8], es donde un proceso defectuoso muestra un comportamiento arbitrario o malicioso.

Para manejar la incertidumbre del tiempo se plantean dos modelos: el modelo *síncrono* donde los procesos conocen de antemano el tiempo de recepción de los mensajes de parte de los otros procesos, de tal forma que es posible detectar los procesos que fallan. Por otro lado, en un modelo *asíncrono* no se tiene alguna restricción respecto a la recepción de los mensajes o retrasos en la comunicación.

1.2. Descripción del Problema

En los temas referentes al internet o desarrollo de dispositivos móviles una primera tarea a resolver es la transferencia de información o comunicación. Así mismo, los algoritmos para redes con múltiples procesos, y la incertidumbre del tiempo son otros ejemplos de problemas actuales que aborda el cómputo distribuido. Entre las soluciones propuestas se ha abordado el problema del consenso.

El problema del consenso es uno de los problemas fundamentales en el computo distribuido, en el que dado un conjunto de n procesos sobre un sistema de envío de mensajes, cada proceso propone un valor de entrada y debe decidir un valor de salida tal que todos los procesos correctos decidan el mismo valor. Si se tiene un sistema síncrono con $f < n$ fallas de paro, entonces en al menos $f + 1$ rondas los procesos correctos resuelven consenso. Sin embargo, cuando hablamos de fallas bizantinas en un sistema síncrono el problema puede ser resuelto después de $f + 1$ rondas [2]. Por otro lado, en un sistema asíncrono jamás se llegará a un consenso [4]. Estas peculiaridades hacen que el problema de acuerdo de retícula sea de cierta manera más débil, debido a que, por un lado existen algoritmos que resuelven el problema ya sea en un sistema síncrono o asíncrono con fallas. Además, existen algoritmos con orden de complejidad $O(\log f)$ [14].

El problema de acuerdo de retícula [1], para un conjunto de n procesos se tiene que cada proceso p_i propone un valor $x_i \in \mathcal{L}$ y debe decidir alguna salida $y_i \in \mathcal{L}$. Tanto el valor de entrada como el de salida son elementos de la retícula finita ($\mathcal{L}$) con un orden parcial ($\leq$) y la operación *join* ($\sqcup$). Los conceptos de retícula, orden parcial y la operación *join* se revisarán con más detalle en el capítulo 2.

Un algoritmo resuelve el problema de acuerdo de retícula (de sus siglas en inglés LAP), si satisfacen todas y cada una de las siguientes tres propiedades [1]:

- **Downward-Validity**: Para todo $i \in \{1, ..., n\}$, $x_i \leq y_i$.
- **Upward-Validity**: Para todo $i \in \{1, ..., n\}$, $y_i \leq \sqcup\{x_1, ..., x_n\}$.
- **Comparabilidad**: Para todo $i, j \in \{1, ..., n\}$, se cumple $y_i \leq y_j$ o $y_j \leq y_i$.

[1] En el capítulo 2 se van a explicar las relaciones que existen con los valores de entrada y salida.

1.2.1. Algoritmos que resuelven LAP y toleran fallas de paro

A continuación, se hará un breve resumen de los algoritmos que resuelvan el problema de acuerdo de retícula, tanto en un modelo que tolera fallas de paro (Tabla 1) como uno que admite fallas de tipo bizantino (Tabla 2).

El primer algoritmo propuesto para resolver el problema fue dado por Attiya en [1], el cual trabaja sobre un modelo síncrono con fallas de paro, dicho algoritmo termina en $\log(n) + 1$ rondas. La idea es dividir recursivamente en dos grupos los procesos por cada ronda. Supongamos que en la primera ronda tenemos n procesos, entonces se divide en dos grupos de tal forma que en el primer grupo se encuentran los primeros $\frac{n}{2}$ procesos y los restantes $\frac{n}{2}$ procesos pertenecen al segundo grupo. Ahora, cada proceso que pertenece al primer grupo envía su valor en un mensaje a todos los procesos del segundo grupo y cada proceso del segundo grupo aplica la operación *join* a los valores recibidos. Entonces, el valor de salida de cada proceso correcto del segundo grupo cumple la condición de comparabilidad ($\geq$) con los valores de salida de los procesos correctos del primer grupo. Ahora se aplica este procedimiento sobre los dos grupos generados en la primera ronda, de tal forma que en la segunda ronda existen cuatro grupos: los procesos del primer grupo (los primeros $\frac{n}{4}$ procesos) envían su mensaje a los procesos del segundo grupo (los procesos $\frac{n}{4}+1$ hasta $\frac{n}{2}$) y los procesos del tercer grupo (los procesos $\frac{n}{2}+1$ hasta $\frac{3n}{4}$) envían su mensaje a los procesos del cuarto grupo (los $\frac{n}{4}$ procesos restantes), donde los procesos pertenecientes a los grupos pares aplican la operación *join* sobre los valores recibidos. De tal forma que al final de cada ronda r existen 2^r grupos y cada proceso de cada grupo es comparable con los procesos correctos de los grupos restantes, pero no necesariamente con los procesos que pertenecen a su grupo. Podemos observar la división de los n procesos en las primeras dos rondas en la figura 1.1.

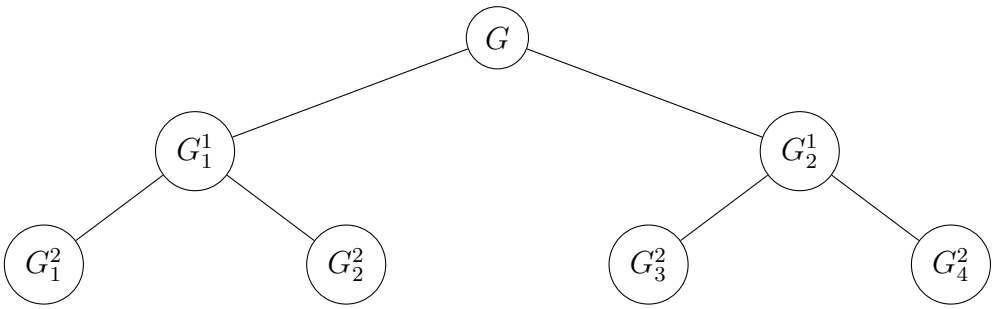

Figura 1.1: División de los procesos de las primeras dos rondas con el algoritmo de Attiya.

Por otro lado, Xiong Zheng, Changyong Hu, y Vijay K. Garg [14] presentan el algoritmo LA_α, el cual trabaja en un modelo síncrono con fallas de paro y tiene un orden $O(\log(h(\mathcal{L})))$, donde $h(\mathcal{L})$ es la altura conocida de la retícula. Esta solución sirve de base para un segundo algoritmo LA_β, el cual con ayuda de LA_α y teniendo el número de fallas tiene un orden $O(\log f)$ rondas. Finalmente, dan un tercer algoritmo LA_γ, el cual tiene un orden mín$\{O(\log^2(h(\mathcal{L}))), O(\log^2 f)\}$. Referente a LA_β, la cota es significativamente mejor que el propuesto por Mavronicolas [11], el cuál es un algoritmo de detención temprana para el acuerdo de retícula y se tiene un orden $min\{O(h(\mathcal{J}(X)^2)), O(\sqrt{f})\}$ rondas.

[2] $\mathcal{J}(X)$ es la sub-retícula superior generada por X.

En la *tabla 1* se muestran algoritmos referentes a los dos modelos. En el modelo asíncrono se tienen dos algoritmos, uno propuesto por Faleiro [3] y el otro es LA_δ [14], en ambos se observa que el número de rondas es de orden lineal, mientras que el de Faleiro es un algoritmo estilo Paxos [7]. El algoritmo de Zheng usa una idea similar a los aplicados en un sistema síncrono en [14], pero con un extra, el cual sirve para solucionar los posibles retrasos arbitrarios de los mensajes.

Referencia	Complejidad	Modelo
[1]	$O(\log n)$	Síncrono
[11]	$min\{O(h(\mathcal{J}(X))), O(\sqrt{f})\}$	
LA_α [14]	$O(\log h(\mathcal{L}))$	
LA_β [14]	$O(\log f)$	
LA_γ [14]	$min\{O(\log^2 h(\mathcal{L})), O(\log^2 f)\}$	
LA_R	$O(f)$	
[3]	$O(n)$	Asíncrono
LA_δ [14]	$min\{O(h(\mathcal{L})), O(f)\}$	

Tabla 1: Algoritmos que toleran fallas de paro.

1.2.2. Algoritmos que resuelven LAP y toleran fallas bizantinas

En cuanto a los algoritmos propuestos para un modelo con fallas de tipo bizantino, Di Luna y otros en [10], presentan el primer algoritmo para un modelo asíncrono donde su algoritmo toma $O(f)$ rondas. La idea general, es que, dado un proceso correcto se construye un conjunto de valores seguros (los únicos valores que posiblemente se entregarán en rondas futuras). Donde cada proceso entregará su mensaje solo si todos los valores que tiene están contenidos en su conjunto de valores seguros.

Xiong Zheng y Vijay Garg proponen tres algoritmos para un sistema asíncrono [12]. El primer algoritmo toma $O(\sqrt{f})$ rondas y tiene la propiedad de detención temprana. Mientras que los otros algoritmos toman $O(\log f)$ y $O(\log n)$ rondas. Donde los tres algoritmos tienen la propiedad de tolerar $f < \frac{n}{3}$ fallas. Por otro lado, agregando un sistema de autentificación se tiene una tolerancia de $f < \frac{n}{2}$ fallas.

En [9] Di Luna y otros muestran un algoritmo para sistemas con fallas bizantinas en un modelo síncrono (mediante canales autentificados), el cual toma $O(\log f)$ rondas y es capaz de tolerar $f < \lceil \frac{n}{4} \rceil$ fallas. En este artículo se muestra que si se tiene un sistema con fallas bizantinas en un modelo síncrono mediante canales autentificados el problema de acuerdo de retícula no puede ser resuelto si $f \geq \lceil \frac{n}{3} \rceil$. Como parte de la solución y tomando un modelo con firmas digitales, ellos muestran un algoritmo que trabaja en un sistema que tolera fallas bizantinas con $f < \lceil \frac{n}{3} \rceil$ en un modelo síncrono y que termina en $O(\log f)$ rondas.

Finalmente, en [13] Xiong y otros muestran dos resultados para el problema de acuerdo de retícula en un sitema con fallas de tipo bizantino en un modelo asíncrono. El primer resultado es un algoritmo que toma $O(\log f)$ rondas y tolera $f < \frac{n}{5}$ fallas, mientras que el segundo resultado asocia un modelo con firmas digitales y es capaz de tolerar $f < \frac{n}{3}$ fallas.

Referencia	Complejidad	DS	Fallas	Modelo
[12]	$O(\log f)$	No	$f < \frac{n}{3}$	Síncrono
[12]	$O(\log f)$	Si	$f < \frac{n}{2}$	
[12]	$O(\log n)$	No	$f < \frac{n}{3}$	
[12]	$O(\log n)$	Si	$f < \frac{n}{2}$	
[12]	$O(\sqrt{f})$	No	$f < \frac{n}{3}$	
[12]	$O(\sqrt{f})$	Si	$f < \frac{n}{2}$	
[9]	$O(\log f)$	No	$f < \frac{n}{4}$	
[9]	$O(\log f)$	Si	$f < \frac{n}{3}$	
[10]	$O(f)$	No	$f < \frac{n}{3}$	Asíncrono
[13]	$O(\log f)$	No	$f < \frac{n}{5}$	
[13]	$O(\log f)$	Si	$f < \frac{n}{3}$	

Tabla 2: Algoritmos que toleran fallas de tipo bizantino.

1.3. Desarrollo de la tesis

Cuando uno habla de algoritmos distribuidos, una de las preguntas principales que se hace es ¿cuál es el orden de complejidad del algoritmo?, pero siendo más ambiciosos uno se puede preguntar ¿cuál es el orden de complejidad del problema a resolver? (complejidad en el número de rondas). Esta pregunta, en el caso de LAP en un sistema síncrono con fallas de paro sigue siendo una incógnita. Sin embargo, existen algoritmos que lo resuelven con un orden de complejidad que podríamos calificar como buenos:

- En [11], Marios Mavronicolas propone su algoritmo. En ese mismo artículo demostró su correctez y a la vez acotó el número de rondas del algoritmo. Al final, Marios Mavronicolas se plantea un par de cuestiones referente al algoritmo, primero saber si la cota propuesta es justa o no, también responder a la pregunta ¿existe un algoritmo de detención temprana que resuelva LAP en un sistema síncrono con un orden de complejidad $\sqrt{f}$?.
- El segundo algoritmo es LA_α de Xiong Zheng, Changyong Hu y Vijay K. Garg presentado en [14], el cual aborda el LAP con la suposición de que los procesos tienen conocimiento previo del tamaño de la retícula de entrada. Además, el algoritmo tiene un orden de complejidad $\log(h(\mathcal{L}))$. Entre las interrogantes que deja este algoritmo es corroborar si la cota propuesta es justa o no.

1.3.1. Objetivos

Los objetivos principales de la tesis son los siguientes:

- El objetivo de investigación se centrará en la complejidad respecto al número de rondas de los algoritmos (LA_R, LA_M, LA_α) que resuelven el LAP.
- Se presentará el algoritmo LA_R y se demostrará su correctez así como su complejidad. Además, se estudiará si existen mejoras para reducir su complejidad.

- Se homologará la notación del algoritmo LA_M presentado en [11]. Posteriormente se demostrará la correctez del algoritmo homologado y su complejidad.
- Se encontrará una mejora en la complejidad del algoritmo LA_M.
- Se demostrará que las cotas de los algoritmos LA_R, LA_M y LA_α son justas.
- Finalmente, se plantea el algoritmo LA_τ para la solución al LAP en una retícula completa.

La conclusión de estos objetivos nos permitirá tener un panorama más amplio acerca de estas soluciones abordadas para resolver el LAP. También nos permitirá hacer un estudio acerca del orden de complejidad en el número de rondas para las soluciones descritas del LAP y continuar en la línea de la hipótesis de que el LAP es de orden logarítmico.

1.3.2. Organización de la tesis

La organización de la tesis es la siguiente:

- En el capítulo 2 se mostrarán los conceptos necesarios para abordar un poset y una retícula, de la misma forma se verá con más detalle la definición del Problema de Acuerdo de Retícula. Finalmente, se abordará el modelo topológico de la red sobre la cual se va a trabajar.
- En el capítulo 3, se empezará por exponer el primer algoritmo LA_R. Este algoritmo es una forma natural y sencilla de como resolver el LAP. Se va a demostrar el orden de complejidad en el número de rondas que se requiere para resolver el LAP y se muestra que dicha cota es justa.
- En la línea de estudiar los algoritmos que resuelven el LAP. En el capítulo 4 y 5 se estudiarán dos algoritmos (Algoritmo de Mavronicolas y LA_α) que tienen un orden de complejidad en el número de rondas más óptimo ($O(\sqrt{f})$ y $O(\log(h(\mathcal{L})))$.
- Por último, las conclusiones y el trabajo que se llevará a cabo en el futuro se expondrán en el capítulo 6.

Capítulo 2

Problema de Acuerdo de Retícula (LAP)

En este capitulo se van a estudiar los conceptos de retícula así como la topología del modelo asociado al problema. Una vez en contexto se va a definir el LAP en un modelo síncrono considerando fallas de paro y las propiedades que se deben de satisfacer.

2.1. Conjunto Parcialmente Ordenado (poset)

La teoría de *conjuntos parcialmente ordenados* tiene aplicaciones en diversas áreas de las ciencias de la computación e ingenierías, por ejemplo: en computo distribuido, teoría de concurrencia, minería de datos, entre otros. Por otro lado, se tienen aplicaciones en matemáticas como: combinatoria, teoría de gráficas, teoría de números y teoría de grupos [5].

En esta sección se van a definir formalmente los conceptos de conjunto parcialmente ordenado y abordar unos resultados importantes.

En las matemáticas una disyuntiva es poder definir conceptos primordiales, por ejemplo, en la teoría de conjuntos se puede hacer la pregunta ¿qué es un conjunto?, y responder a esta pregunta se vuelve en un problema de estructura. Motivo por el cual no se va a dar una definición al concepto de *conjunto*, sólo se tomará como un concepto primitivo y se asumirán ciertas características respecto a un *conjunto*.

- Un *conjunto* S está formado por elementos y denotaremos como $a \in S$ si y sólo sí a es un elemento de S. Por otro lado, si a es un elemento y no está en el conjunto S entonces lo denotaremos como $a \notin S$. Un elemento a sólo puede tener alguno de los dos estados.
- El *conjunto vacío* es aquel que no tiene elementos y se denotará por $\emptyset$.
- Sea $P(x)$ una propiedad que caracteriza a los elementos de un conjunto y se denotará como $S = \{x | P(x)\}$.

2.1.1. Relación de equivalencia

En teoría de conjuntos a veces nos gustaría poder clasificar los elementos de un conjunto, por lo cual se establece una *relación de equivalencia.*

Sea $S = \{x | x \in \mathbb{Z}\}$ y nos gustaría agruparlo en dos clases diferentes, tal que $x \in \mathbb{Z}$ entonces $x \in P$ o $x \in I$, pero no en ambos. Donde P es el conjunto de los números pares e I es el conjunto de los impares. Por lo cual establecemos la siguiente relación $x \equiv y \ mod\ 2$ ($\equiv_2$), de tal forma que es posible dividirlos en dos conjuntos P e I independientes.

Ahora definamos formalmente una *relación de equivalencia.*

Definición 2.1.1. *Sea S un conjunto y $\sim$ una relación definida sobre S. Diremos que $\sim$ es una relación de equivalencia si satisface las siguientes propiedades:*

- *Reflexividad: Para cada $x \in S$ $x \sim x$.*
- *Simetría: Si $x \sim y$ entonces $y \sim x$.*
- *Transitividad: Si $x \sim y$ y $y \sim z$ entonces $x \sim z$.*

Dada la relación $\equiv_2$ veamos que en efecto es una relación de equivalencia, para ello tenemos que validar que se satisfacen las tres propiedades ya descritas.

- Reflexividad: Sea $x \in S$ de la definición de congruencia tenemos que $x - x = 0$ y $0 \ mod\ 2 = 0$.
- Simetría: Sean $x, y \in S$ tal que $x \sim y$ implica que $x \equiv y \ mod\ 2$ entonces $2|x - y$. Entonces, $2| - (y - x)$ lo que significa $2| - 1$ o $2|(y - x)$, pero como $2 \not| - 1$ por lo tanto $y \equiv x \ mod\ 2$.
- Transitividad: Sean $x, y, z \in S$, tal que $x \sim y$ y $y \sim z$. Como $2|x - y$ y $2|y - z$ entonces existen $m, l \in \mathbb{Z}$, tal que $x - y = 2m$ y $y - z = 2l$. Por lo tanto, al sumar tenemos $x - z = 2(m + l)$ es decir $2|x - z$.

Por lo tanto, $\equiv_2$ es una relación de equivalencia.

2.1.2. Orden parcial

Definición 2.1.2. *Sea $\mathcal{R}$ una relación binaria sobre un conjunto S, diremos que $(S, \mathcal{R})$ es un conjunto con orden parcial si satisface las siguientes propiedades:*

- *Reflexividad: Para cada $x \in S$ $x\mathcal{R}x$.*
- *Antimetría: Si $x\mathcal{R}y$ y $y\mathcal{R}x$, entonces $x = y$.*
- *Transitividad: Si $x\mathcal{R}y$ y $y\mathcal{R}z$, entonces $x\mathcal{R}z$.*

Un conjunto con un orden parcial se denomina *conjunto parcialmente ordenado (poset)*. La relación $\mathcal{R}$ usualmente se denota como $\leq$ debido a que ya cumple con la propiedad de dicotomía.

Veamos un ejemplo, sea el conjunto $S = \{a, b, c, d, e\}$ y definamos la relación entre cada elemento del conjunto como sigue: $\mathcal{R} := \{(a, b), (b, c), (a, c), (a, d), (d, e), (a, e), (b, e)\}$. Una forma de representar gráficamente a un *conjunto parcialmente ordenado finito* es mediante un *diagrama de Hasse*.

De acuerdo a [5] podemos definir un *diagrama de Hasse* (figura 2.1) de un *poset* como: una gráfica con la propiedad de que existe un arista de x a y si y sólo sí $x <_c y$. Dónde $<_c$ denota a una *cubierta*. Entenderemos por *cubierta* como: sean $x, y \in S$ diremos que y cubre a x si $x < y$ y no existe $z \in S$ tal que $x < z < y$.

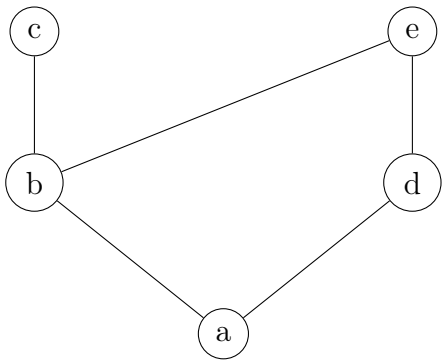

Figura 2.1: Diagrama de Hasse.

2.2. Retícula

Se definirán dos operaciones para operar sobre subconjuntos de un conjunto parcialmente ordenado $(X, \leq)$ [5].

Definición 2.2.1. *Ínfimo (meet). Sea $Y \subseteq X$, donde $(X, \leq)$ es un poset. Para $m \in X$, diremos que $m = \inf Y$ si y sólo sí:*

- $\forall y \in Y : m \leq y$.
- $\forall m' \in X : (\forall y \in Y : m' \leq y) \Rightarrow m' \leq m$.

Definición 2.2.2. *Supremo (join). Sea $Y \subseteq X$, donde $(X, \leq)$ es un poset. Para $m \in X$, diremos que $m = \sup Y$ si y sólo sí:*

- $\forall y \in Y : y \leq m$.
- $\forall m' \in X : (\forall y \in Y : y \leq m') \Rightarrow m \leq m'$.

El *ínfimo* también se conoce como la máxima cota inferior y lo denotaremos como $\sqcap$. Por otro lado, el *supremo* también se conoce como la mínima cota superior y lo denotaremos como $\sqcup$. En la figura 2.1 se puede observar que para el subconjunto $\{c, e\}$ no existe el supremo. Como nos hemos dado cuenta el supremo podría no existir para algún subconjunto Y de $(X, \leq)$, lo mismo puede ocurrir para el ínfimo. Observemos un ejemplo donde existen ambos (figura 2.2), en este *poset* finito podemos ver que tanto el supremo como el ínfimo existen para cualquier subconjunto, al que también se le conoce como *retícula*.

Definición 2.2.3. *Retícula (lattice). Un poset* $(X, \leq)$ *es una retícula* $(\mathcal{L})$ *si y sólo sí* $\forall x, y \in X$ *se tiene la existencia de* $x \sqcup y$ *y* $x \sqcap y$.

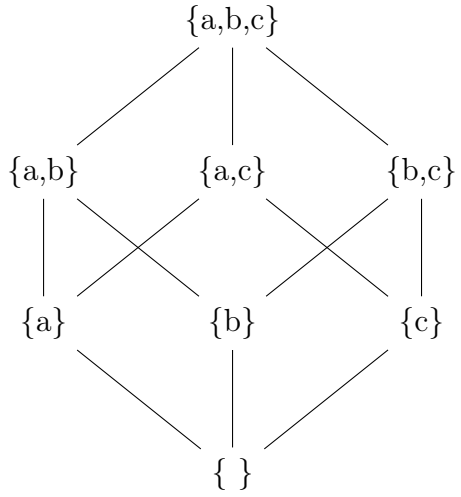

Figura 2.2: Retícula booleana de los subconjuntos de $\{a, b, c\}$.

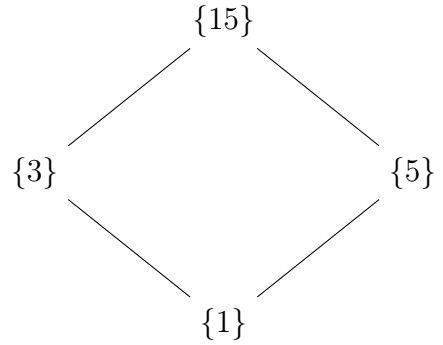

Figura 2.3: Retícula D_{15}.

En la definición de *retícula* las operaciones *join* y *meet* son operaciones binarias. En esta tesis se trabajará con retículas donde ambas operaciones están definidas sobre subconjuntos finitos. Por lo que se van a extender ambas operaciones de la siguiente manera: Sea $\mathcal{L}$ una retícula dada por $(X, \leq)$ y sea $S \subseteq \mathcal{L}$ entonces $\sqcup S \in \mathcal{L}$ y $\sqcap S \in \mathcal{L}$. De tal forma que $\sqcup S$ se obtiene al aplicar la operación *join* inductivamente sobre los elementos de S, de la misma forma para $\sqcap S$. Sean los siguientes ejemplos de retículas:

- Conjunto D_n de divisores naturales de n con la operación de orden parcial usual $x \leq y$ siempre que $x|y$. El supremo viene dado por el mínimo común múltiplo y el ínfimo por el máximo común divisor (figura 2.3).
- Intervalo [0,1] con la operación de orden usual (menor o igual que).
- Conjunto de números enteros con la operación de orden usual (menor o igual que).
- Retícula booleana de los subconjuntos de S. Sea S un conjunto y denotemos como $Pow(S)$ al conjunto potencia del conjunto S, donde la *inclusión* $(\subseteq)$ queda definida como la relación de orden parcial y las operaciones: *join* queda dada por la unión $(\cup)$

y el *meet* por la intersección ($\cap$), a esta retícula se le conoce como *retícula booleana de los subconjuntos de* S (figura 2.2).

Definición 2.2.4. *Sea* $\mathcal{L}$ *una retícula dada por* $(X, \leq)$. *Diremos que* $\mathcal{S} \subseteq \mathcal{L}$ *es una sub-retícula, sí y sólo sí,* $\forall a, b \in \mathcal{S}$ *se tiene que* $a \sqcup b \in \mathcal{S}$ *y* $a \sqcap b \in \mathcal{S}$.

Propiedades básicas de una retícula

Proposición 2.2.5. *Sea* $(X, \leq)$ *una retícula* $\mathcal{L}$. *Sean* $y \in \mathcal{L}$ *y* $S \subseteq \mathcal{L}$ *tal que* $x \leq y \; \forall x \in S$ *entonces* $\sqcup S \leq y$.

Demostración. Tenemos que $x \leq y \; \forall x \in S$, además sea $j = \sqcup S$. De la definición 2.2.2: $\forall x \in S$ se cumple $x \leq j$. Además, se tiene que $\forall j' \in \mathcal{L}$ tal que $\forall x \in S$ se cumple que $x \leq j'$, entonces $j \leq j'$. Por lo tanto, $j \leq y$. ■

Proposición 2.2.6. *Sea* $(X, \leq)$ *una retícula* $\mathcal{L}$. *Sean* $y \in \mathcal{L}$ *y* $S \subseteq \mathcal{L}$ *tal que* $x \leq y$ *o* $x \geq y \; \forall x \in S$ *entonces* $\sqcup S \leq y$ *o* $\sqcup S \geq y$.

Demostración. Si $x \leq y \; \forall x \in S$, por la proposición 2.2.5, entonces $\sqcup S \leq y$. Por otro lado, si existe $x \in S$ tal que $y \leq x$, por la definición de *join* tenemos que $x \leq \sqcup S$, aplicando la transitividad se tiene que $y \leq \sqcup S$. ■

Proposición 2.2.7. *Sea* $(X, \leq)$ *una retícula* $\mathcal{L}$. *Sean* $A, B \subseteq \mathcal{L}$ *tal que* $A \subseteq B$ *entonces* $\sqcup A \leq \sqcup B$.

Demostración. Sea $j_B = \sqcup B$, es decir se tiene que $\forall b \in B \; b \leq j_B$. Por otro lado, sea $j_A = \sqcup A$. De la definición 2.2.2: $\forall a \in A$, $a \leq j_A$. Además, se tiene que $\forall x \in \mathcal{L}$, tal que $\forall a \in A$ se cumple que $a \leq x$, entonces $j_A \leq x$. En particular $a \leq j_B \; \forall a \in A$. Por lo tanto, $j_A \leq j_B$. ■

Al ejemplo proporcionado por el diagrama de Hasse en la figura 2.1 se le conoce como *sub-retícula inferior*. Definiremos una *sub-retícula inferior* $\mathcal{I}$ sobre $(X, \leq)$: si $\forall x, y \in \mathcal{I}$, existe $x \sqcap y \in \mathcal{I}$. Por otro lado, se le conoce como *sub-retícula superior* a $\mathcal{S}$ sobre $(X, \leq)$ si $\forall x, y \in \mathcal{S}$, existe $x \sqcup y \in \mathcal{S}$.

Definición 2.2.8. *Sea* $\mathcal{L}$ *una retícula* $(X, \leq)$. *Sea* $S \subseteq \mathcal{L}$ *se va a definir inductivamente* $\mathcal{L}(S)$ *como la sub-retícula generada por* S, *como sigue [11]:*

- $\forall s \in S$, $s \in \mathcal{L}(S)$.
- *Para cualquier subconjunto numerable* $Y \subseteq \mathcal{L}(S)$ *se tiene:*
  - $\sqcup Y \in \mathcal{L}(S)$.
  - $\sqcap Y \in \mathcal{L}(S)$.
- *Si* $x \in \mathcal{L}$ *y* x *no puede ser generado por las reglas anteriores entonces* $x \notin \mathcal{L}(S)$.

Para el caso de esta tesis se estará trabajando sobre sub-retículas superiores, el subconjunto que contiene a todos los elementos que pueden expresarse como el *join* de un subconjunto de $\mathcal{L}(S)$ se va a denominar *sub-retícula superior generada* por S.

Definición 2.2.9. *Sea $\mathcal{L}$ una retícula $(X, \leq)$. Sea $S \subseteq \mathcal{L}$, se va a definir inductivamente $\mathcal{J}(S)$ como la sub-retícula superior generada por S, como sigue [11]:*

- $\forall s \in S$, $s \in \mathcal{J}(S)$.
- *Para cualquier subconjunto numerable $Y \subseteq \mathcal{J}(S)$ se tiene:*
  - $\sqcup Y \in \mathcal{J}(S)$.
- *Si $x \in \mathcal{L}$ y x no puede ser generado por las reglas anteriores entonces $x \notin \mathcal{J}(S)$.*

Teorema 2.2.10 ([11], Proposición 1)**.** *Sea $(X, \leq)$ una retícula $\mathcal{L}$. Sea $A \subseteq \mathcal{L}$ y $\mathcal{J}(A)$ es la sub-retícula superior generada por A si $x \in \mathcal{J}(A)$ entonces $x = \sqcup U$ para algún $U \subseteq A$.*

Para más detalle de la demostración del teorema 2.2.10 abordar [11].

Definición 2.2.11. *Altura de un valor v en una retícula es la longitud del camino más largo de cualquier valor mínimo a v. Se va a denotar como $h(v)$.*

Definición 2.2.12. *Altura de una retícula $\mathcal{L}$ $(X, \leq)$ es el tamaño de su valor más largo:*

$$h(\mathcal{L}) := max\{h(v) | v \in \mathcal{L}\}. \tag{2.1}$$

2.3. LAP en computación distribuida

Se abordará el modelo de cómputo distribuido sobre el cual se va a estar trabajando y se va a definir el problema de acuerdo de retícula.

2.3.1. Topología del modelo.

Se considerará un sistema distribuido, donde la transferencia de información se llevará a cabo mediante un sistema de paso de mensajes. Por otro lado, se va a tener una topología conectada completamente donde existen n procesos, los cuales se van a definir como $p_1, p_2, ..., p_n$.

Como se mencionó anteriormente existen dos tipos de fallas para un proceso. A lo largo de esta tesis se va a estudiar el problema donde un proceso puede tener una falla de paro. Cuando se haga mención de falla se referirá a una falla de paro.

Finalmente, se supondrá que se tiene un canal de comunicación confiable con un modelo síncrono.

2.3.2. Problema de Acuerdo de Retícula

Anteriormente se mencionó cuando un algoritmo resolvía el LAP. De igual manera en la subsección anterior se definió la topología del modelo.

En el contexto de la tesis vamos a trabajar con una sub-retícula finita $\mathcal{L}$ $(X, \leq, \sqcup)$, la cual queda definida tomando un conjunto finito parcialmente ordenado $(X, \leq)$ y la

operación *join* ($\sqcup$). Para fines de esta tesis cuando se haga uso del término retícula se está haciendo referencia a una sub-retícula superior.

Ahora, como se mencionó en la definición de la topología del modelo se tienen n procesos. En el *problema de acuerdo de retícula* [1], cada proceso p_i elige un valor de entrada x_i de la retícula $\mathcal{L}$ (en el caso de la tesis X representa el conjunto de valores de entrada de los procesos). Después de procesar un algoritmo cada proceso p_i termina con un valor y_i de salida que pertenece a $\mathcal{L}$ (figura 2.4).

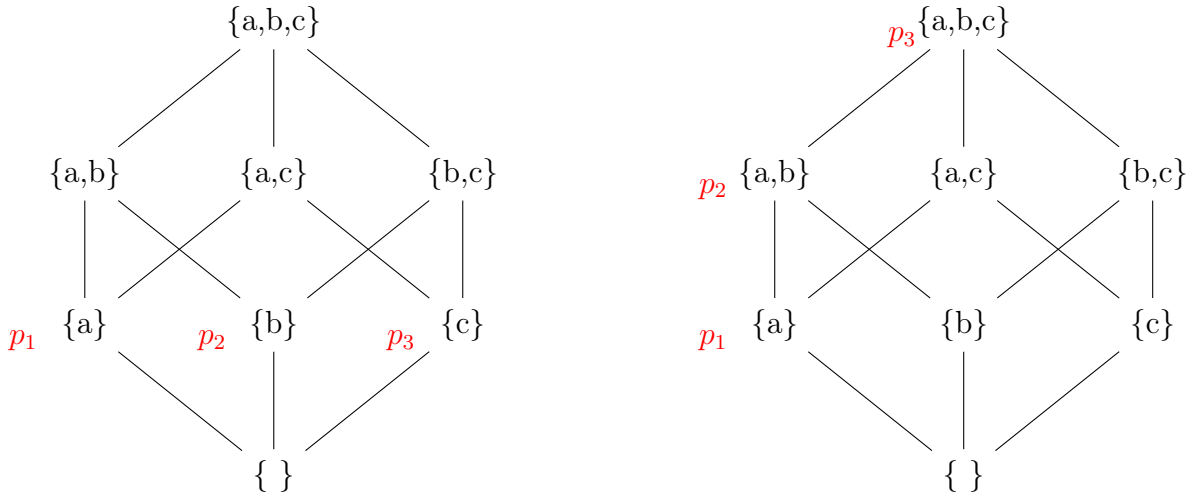

Figura 2.4: a) Valores de entrada para cada uno de los procesos; b)Valores de salida correcta para los tres procesos.

Diremos que un algoritmo resuelve el problema de acuerdo de retícula o es correcto si satisface las siguientes propiedades:

- **Downward-Validity**: Para todo $i \in \{1, ..., n\}$, $x_i \leq y_i$.
- **Upward-Validity**: Para todo $i \in \{1, ..., n\}$, $y_i \leq \sqcup\{x_1, ..., x_n\} = \sqcup X$.
- **Comparabilidad**: Para todo $i, j \in \{1, ..., n\}$, se cumple $y_i \leq y_j$ o $y_j \leq y_i$.

La propiedad de comparabilidad nos pide que los valores de salida de cada proceso correcto sean comparables entre sí. La propiedad Downward-validity requiere que el valor de salida de cada proceso sea comparable y no menor a su entrada. Finalmente, la propiedad Upward-validity requiere que el valor de salida de cada proceso sea comparable y no mayor que el *join* de los valores de entrada ($\sqcup X$). Observemos que estas dos últimas propiedades nos acotan el valor de salida y_i de cada proceso p_i, es decir $x_i \leq y_i \leq \sqcup X$.

Capítulo 3

Algoritmo que tolera fallas de paro

Una vez definida la topología del modelo y abordada la definición del problema de acuerdo de retícula. En este capítulo para entender la metodología de como funcionan las propiedades se va a estudiar un algoritmo que únicamente hace uso de la operación $join$ (lo que quiere decir es que no hay otra operación o comparativa involucrada). Además, cuando se haga mención de que un proceso puede fallar se hará alusión a una falla de paro.

Finalmente, para justificar la cota inferior del algoritmo se buscará una ejecución donde no se cumple alguna de las tres propiedades.

3.1. Algoritmo LA_R

Estudiaremos el primer algoritmo para entender las propiedades que se deben satisfacer para resolver el LAP. Para ello, se considerará un sistema con n procesos y pueden ocurrir a lo más $f < n$ fallas. De manera general se va a dar un algoritmo que termina en a lo más R rondas.

Tomando un proceso p_i, donde $i \in [n]$ y dada una retícula finita ($\mathcal{L}$). Cada proceso p_i propone su valor de entrada $x_i \in \mathcal{L}$ y al final de procesar el algoritmo tiene un valor de salida $y_i \in \mathcal{L}$.

En una primera instancia cada proceso no tiene conocimiento de los valores que han elegido los otros procesos. Para tener conocimiento del sistema los procesos comparten su valor a los otros procesos pero ¿qué pasa si un proceso falla?. Recordemos que cada proceso desconoce que proceso ha fallado o que valor tenía ese proceso. Teniendo este conflicto un algoritmo debe de ayudar a que los procesos decidan su salida y que se satisfagan las propiedades de un LAP.

Una manera sencilla y natural de proponer un algoritmo sería de la siguiente manera:

- Cada proceso recibe su valor de entrada (x_i).
- Cada proceso comparte su valor y suponemos que recibe su propio valor (v_i donde almacena el $join$ de los valores recibidos al momento U_i).
- Cada proceso aplica la operación $join$ a los valores recibidos y almacena el resultado en la variable v_i.

- Repite el proceso un cierto número de rondas R.
- Finalmente produce su valor de salida y_i.

Con la idea anterior planteamos el algoritmo siguiente.

Algoritmo 1: $LA_R(x_i)$ para un proceso p_i

Datos: x_i = valor de entrada
Resultado: y_i = valor de salida

1 $v_i = x_i$
2 **para** $r = 1$ *hasta* R **hacer**
3 | *enviar* a todos (v_i)
4 | $U = \{v_j |$ *recive* v_j *de* $p_j\}$
5 | $v_i = \sqcup\{u | u \in U\}$
6 **fin**
7 $y_i = v_i$

3.1.1. Correctez del algoritmo

Siempre que se analice un algoritmo se va a considerar el valor de las variables al final de cada ronda, además, vamos a denotar con un súper índice la ronda en la que se encuentra (por ejemplo v_i^r hace referencia al valor que tiene el proceso p_i al final de la ronda r).

Una vez establecidas las reglas de operación se va a demostrar que el algoritmo 1 es correcto, es decir, se satisfacen las tres propiedades siguientes:

- Downward-Validity.
- Upward-Validity.
- Comparabilidad.

Primero vamos a ver que se satisface la propiedad de Downward-validity y posteriormente la propiedad Upward-validity:

Lema 3.1.1. *Sea p_i un proceso correcto y y_i su valor de salida, entonces al final de procesar el algoritmo 1, $x_i \leq y_i$ (propiedad Downward-validity).*

Demostración. Se demostrará que la invariante: $x_i \leq v_i^r$ se preserva al término de cada ronda r para un proceso p_i.

- Caso base cuando $r = 1$, sea p_i un proceso correcto.

  De la línea 1, tenemos como valor inicial de $v_i = x_i$.

  De la línea 3, el proceso p_i envía su valor v_i a todos los procesos.

  De la línea 4, se tiene U_i^1 como el conjunto de los valores recibidos por el proceso p_i, el cual incluye v_i. De esta forma, de la línea 5 $v_i^1 = \sqcup U_i^1$, como $x_i = v_i \in U_i^1$. Entonces, al obtener el *join* de U_i^1 se tiene que $x_i \leq \sqcup U_i^1 = v_i^1$ al final de la ronda 1.

- Hipótesis de inducción: Supongamos que la invariante $x_i \leq v_i^r$ se preserva al término de cada ronda r.

- Por demostrar que la invariante se mantiene al final de la ronda $r+1$.

  Al inicio de la ronda $r+1$ se tiene como valor de entrada v_i^r y a su vez, este lo comparte a los demás procesos (línea 3).

  De la línea 4, se tiene U_i^{r+1} como el conjunto de los valores recibidos por el proceso p_i en la ronda $r+1$, el cual incluye v_i^r. De esta forma, de la línea 5 $v_i^{r+1} = \sqcup U_i^{r+1}$, como $v_i^r \in U_i^{r+1}$. Entonces, al obtener el *join* de U_i^{r+1} se tiene que $v_i^r \leq v_i^{r+1}$ al final de la ronda $r+1$, por hipótesis de inducción $x_i \leq v_i^r \leq v_i^{r+1}$.

■

Lema 3.1.2. *Sea p_i un proceso correcto y y_i su valor de salida, entonces al final de procesar el algoritmo 1, $y_i \leq \sqcup X$ (propiedad Upward-validity).*

Demostración. Se demostrará que la invariante $v_i^r \leq \sqcup X$ se preserva al término de cada ronda r para un proceso p_i.

- Caso base cuando r=1, sea p_i un proceso correcto.

  De la línea 1, tenemos como valor inicial de $v_i = x_i$.

  De la línea 3, el proceso p_i envía su valor v_i a todos los procesos.

  De la línea 4, se tiene U_i^1 como el conjunto de los valores recibidos por el proceso p_i, el cual incluye v_i. De esta forma, de la línea 5 $v_i^1 = \sqcup U_i^1$, donde cada valor $v_j \in U_i^1$ se tiene que $v_j \in X$. Por lo tanto, $U_i^1 \subseteq X$ entonces $v_i^1 \leq \sqcup X$.

- Hipótesis de inducción: Supongamos que la invariante $v_i^r \leq \sqcup X$ se preserva al término de cada ronda r.

- Por demostrar que la invariante se mantiene al término de la ronda $r+1$.

  Al inicio de la ronda $r+1$ se tiene como valor de entrada v_i^r y a su vez, este lo comparte a los demás procesos (línea 3).

  De la línea 4, se tiene U_i^{r+1} como el conjunto de los valores recibidos por el proceso p_i en la ronda $r+1$, el cual incluye v_i^r. De esta forma, de la línea 5 $v_i^{r+1} = \sqcup U_i^{r+1}$, por hipótesis de inducción para cada $v_j \in U_i^{r+1}$ se tiene que $v_j \leq \sqcup X$. Por lo tanto, $v_i^{r+1} \leq \sqcup X$.

■

3.2. Complejidad y propiedad de comparabilidad del algoritmo

De la sección anterior nos podemos preguntar si dado un número de procesos n arbitrario ¿cuántas rondas necesita el algoritmo 1 para resolver el LAP con $f < n$?.

Se va a demostrar que cuando $R = \frac{f}{2} + 1$ resuelve el LAP.

Lema 3.2.1. *Sean p_i y p_j procesos correctos, y sea una ronda r, donde $f \leq 1$. Entonces al final de r, p_i y p_j son comparables.*

Demostración. Primero, tomemos la ejecución cuando $f = 0$, es decir, ningún proceso falla en la ronda r. Sea P_C^r el conjunto de procesos que no fallan en la ronda r. Además, sea X^r el conjunto de valores de los procesos del conjunto P_C^r. Sea $p_i \in P_C^r$, como ningún proceso falla, entonces de la línea 4 (algoritmo 1) el conjunto $U_i^r = X^r$. Por lo tanto, $v_i^r = \sqcup X^r$ para todo $p_i \in P_C^r$. Entonces, al final de la ronda r cada proceso es comparable.

Sea la ronda r y $f = 1$, donde p_f es el proceso que falla en la ronda. Sea P_f^r el conjunto de procesos correctos que reciben el valor del proceso p_f en la ronda r. Por otro lado, sea P_c^r el conjunto de procesos correctos en la ronda r que no reciben el valor del proceso p_f. Donde X_f es el conjunto de valores de los procesos en P_f^r. Mientras, X_c es el conjunto de valores de los procesos en el conjunto P_c^r. Por lo tanto, $X_c \subseteq X_f$ y por la proposición 2.2.7 $\sqcup X_c \leq \sqcup X_f$.

■

Teorema 3.2.2. *El algoritmo LA_R (1) resuelve el problema de acuerdo de retícula cuando $R = \frac{f}{2} + 1$.*

Demostración. Se va a demostrar que se satisfacen las tres propiedades:

- Las primeras dos propiedades las obtenemos inmediatamente de los lemas 3.1.1 y 3.1.2.

- COMPARABILIDAD: Existe una ronda $r' \leq \frac{f}{2} + 1$ tal que $f \leq 1$. Aplicando el lema 3.2.1, al final de la ronda r' los procesos son comparables. Si $r' = \frac{f}{2} + 1$ entonces queda demostrado.

  Por otro lado, si $r' \leq \frac{f}{2}$ por el lema 3.2.1 al final de la ronda r' los procesos son comparables, más aún, se observa que existen a lo más dos conjuntos de procesos que tienen diferentes valores entre sí, $P_1^{r'}$ y $P_2^{r'}$. Donde $P_1^{r'}$ tiene como valor v_{P_1} y $P_2^{r'}$ tiene como valor v_{P_2}, de tal forma que v_{P_1} y v_{P_2} son comparables. Se va a demostrar por inducción que al final de la ronda t, $r' + 1 \leq t \leq \frac{f}{2} + 1$, los procesos son comparables.

  Caso base: sea la ronda $r' + 1$, el conjunto de procesos $P_1^{r'}$ es comparable con el conjunto de procesos $P_2^{r'}$. Donde cada proceso $p \in P_1^{r'} \cup P_2^{r'}$ tiene como valor al inicio de la ronda $r' + 1$, v_{P_1} ó v_{P_2}. De esta forma, el proceso p tiene como valor al final de la ronda $r' + 1$, $v_p^{r'+1} = v_{P_1}$ ó $v_p^{r'+1} = v_{P_2}$. Es decir, existen a lo más dos conjuntos $P_1^{r'+1}$ y $P_2^{r'+1}$, donde $P_1^{r'+1}$ tiene como valor v_{P_1} y $P_2^{r'+1}$ tiene como valor v_{P_2}.

  Hipótesis de inducción: supongamos que al final de cada ronda t existen a lo más dos conjuntos de procesos que tienen diferentes valores entre sí, P_1^t y P_2^t. Donde P_1^t tiene como valor v_{P_1} y P_2^t tiene como valor v_{P_2}, de tal forma que v_{P_1} y v_{P_2} son comparables.

  Al inicio de la ronda $t + 1$, el conjunto de procesos P_1^t es comparable con el conjunto de procesos P_2^t. Donde cada proceso $p \in P_1^t \cup P_2^t$ tiene como valor al inicio de la ronda $t + 1$, v_{P_1} ó v_{P_2}. De esta forma, el proceso p tiene como valor al final de la

ronda $t+1$, $v_p^{t+1} = v_{P_1}$ ó $v_p^{t+1} = v_{P_2}$. Es decir, existen a lo más dos conjuntos P_1^{t+1} y P_2^{t+1}, donde P_1^{t+1} tiene como valor v_{P_1} y P_2^{t+1} tiene como valor v_{P_2}, los cuales son comparables.

■

3.3. Cota inferior

Los algoritmos descritos en la bibliografía son eficientes. Motivo por el cual nos llegamos a preguntar ¿qué tan eficiente llega a ser el algoritmo 1?. A continuación se estudiará si para $R <= \frac{f}{2}$ existe una ejecución tal que el algoritmo no resuelva el LAP.

De esta forma se va a trabajar sobre la retícula 3.1 inciso a. Además, sean $f < n$ fallas, donde en cada ronda ocurren 2 fallas.

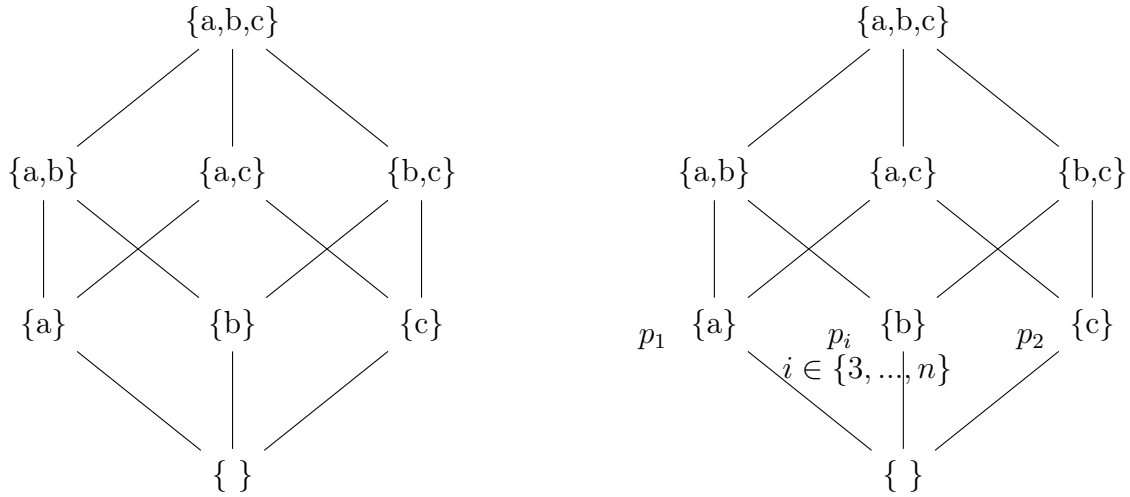

Figura 3.1: a) Retícula booleana de los subconjuntos de $\{a, b, c\}$, b) Valores de entrada para los procesos.

Una vez definida la retícula sobre la cual vamos a trabajar y el número de fallas por ronda. Se procede a definir la ejecución y como se va a llevar a cabo:

- Sin pérdida de generalidad, sea F_1 el conjunto de procesos que fallan en la ronda 1.
- El conjunto F_1 está conformado por el proceso p_1, el cual tiene como valor de entrada $\{a\}$, y el proceso p_2 con valor de entrada $\{c\}$ en la ronda 1.
- El conjunto A_1 está conformado por los procesos restantes, los cuales tienen como valor de entrada $\{b\}$ en la ronda 1.
- El envío de mensajes en la primera ronda se va a llevar a cabo de la siguiente manera (figura 3.2):

- El proceso $p_1^{F_1}$ envía un mensaje únicamente al primer proceso del conjunto A_1 y el proceso $p_2^{F_1}$ envía un mensaje únicamente al segundo proceso del conjunto A_1.
- El conjunto F_2 está conformado por los primeros dos procesos pertenecientes al conjunto A_1.
- Los procesos $p_k \in A_1$ envían su mensaje a todos los procesos.
- Sea A_2 el conjunto $A_1 \setminus F_2$.

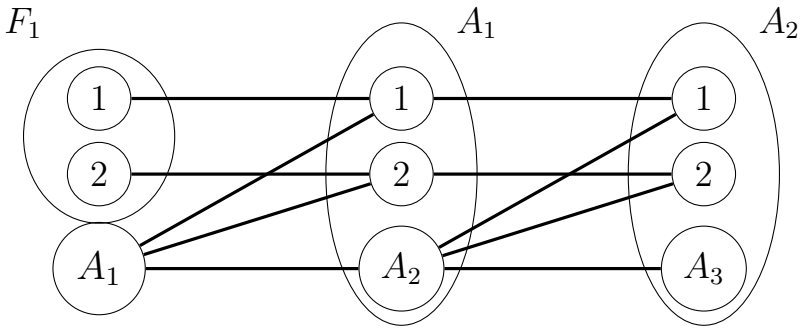

Figura 3.2: Ejecución de la primera y segunda ronda.

- En las rondas subsecuentes se continúa de la siguiente manera: Sea $2 \leq r \leq \frac{f}{2}$, donde F_r es el conjunto de procesos que fallan en la ronda r.
- Por otro lado, A_r es el conjunto de procesos que no fallan en la ronda r.
- El envío de mensajes continúa de la siguiente manera (figura 3.3):
  - El primer proceso del conjunto F_r envía un mensaje únicamente al primer proceso del conjunto A_r y el segundo proceso del conjunto F_r envía un mensaje únicamente al segundo proceso del conjunto A_r.
  - El conjunto F_{r+1} está conformado por los primeros dos procesos pertenecientes al conjunto A_r.
  - Los procesos $p_k \in A_r$ envían su mensaje a todos los procesos.
  - Sea A_{r+1} el conjunto $A_r \setminus F_{r+1}$

Se tienen las siguientes observaciones: En cada ronda fallan dos procesos, de esta forma tenemos que $f-1 \leq \sum_1^{\lfloor \frac{f}{2} \rfloor} 2 \leq f$. Para denotar al primer y segundo proceso que pertenecen a un conjunto A se usara la siguiente notación p_1^A y p_2^A o $p_1, p_2 \in A$.

Se va a demostrar por inducción que al final de cada ronda $1 \leq r \leq \lfloor \frac{f}{2} \rfloor$ existen dos procesos que no son comparables.

- Caso base: cuando $r = 1$. De acuerdo con la ejecución, $p_1, p_2 \in P$ tienen como valor de entrada $\{a\}$ y $\{c\}$ respectivamente. Mientras que los procesos restantes $p \in A_1$ tienen como valor de entrada $\{b\}$. Además, $p_1^{A_1}$ recibe los valores $\{a\}$ y $\{b\}$, y $p_2^{A_1}$

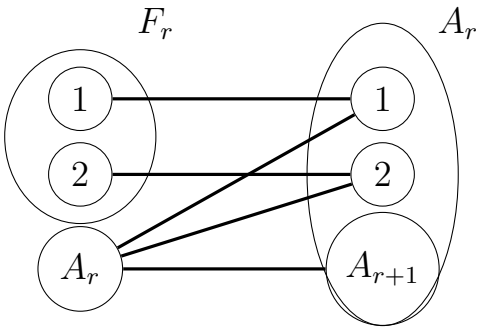

Figura 3.3: Ejecución de la ronda r.

recibe los valores $\{b\}$ y $\{c\}$. Por lo tanto, al final de la ronda 1 $v^1_{p_1^{A_1}} = \{a, b\}$ y $v^1_{p_2^{A_1}} = \{b, c\}$, los cuales no son comparables entre sí. Por otro lado, los procesos $p_a \in A_2$ tienen como valor al final de ronda 1, $v^1_a = \{b\}$. De acuerdo con la ejecución, $p_1^{A_1}, p_2^{A_1} \in F_2$, y A_2 es el conjunto $A_1 \setminus F_2$.

- Hipótesis de inducción: Sea $1 \leq r < \frac{f}{2}$, al final de cada ronda r los procesos que pertenecen a F_{r+1} no son comparables entre sí, es decir, $p_1, p_2 \in F_{r+1}$ $v^r_1 = \{a, b\}$ y $v^r_2 = \{b, c\}$. Además, los procesos que pertenecen al conjunto A_{r+1} al final de la ronda r tienen como valor a $\{b\}$.

- Por demostrar que la hipótesis se mantiene al final de la ronda $r + 1$.

  Por hipótesis, $p_1, p_2 \in F_{r+1}$ tienen como valores de entrada $v^r_1 = \{a, b\}$ y $v^r_2 = \{b, c\}$. Mientras que para todo proceso $p_a \in A_{r+1}$ tiene como valor de entrada $v^r_a = \{b\}$.

  De acuerdo con la ejecución, $p_1 \in A_{r+1}$ recibe los valores $\{a, b\}$ y $\{b\}$, mientras que $p_2 \in A_{r+1}$ recibe los valores $\{b, c\}$ y $\{b\}$. Por otro lado, los procesos $p_a \in A_{r+2}$ tienen como valor al final de la ronda $r + 1$, $v^{r+1}_a = \{b\}$. Por lo tanto, el proceso $p_1^{A_{r+1}}$ tiene como valor $v^{r+1}_1 = \{a, b\}$, y el proceso $p_2^{A_{r+1}}$ tiene como valor $v^{r+1}_2 = \{b, c\}$, los cuales no son comparables entre sí. Finalmente, los procesos $p_a \in A_{r+2}$ tienen como valor al final de la ronda $r + 1$, $v^{r+1}_a = \{b\}$. De acuerdo con la ejecución, $p_1^{A_{r+1}}, p_2^{A_{r+1}} \in F_{r+2}$ y A_{r+2} es el conjunto $A_{r+1} \setminus F_{r+2}$.

Por lo tanto, al final de cada ronda $1 \leq r \leq \lfloor \frac{f}{2} \rfloor$ existen dos procesos que no son comparables, es decir, forman una anticadena [1]. Como $f - 1 \leq \sum_1^{\lfloor \frac{f}{2} \rfloor} 2 \leq f$, entonces en la ronda $\frac{f}{2} + 1$ a lo más puede ocurrir 1 falla. Por la proposición 3.2.1, al final de la ronda $\frac{f}{2} + 1$ todos los procesos correctos son comparables entre sí.

Teorema 3.3.1. *Existe una ejecución tal que el algoritmo LA_R requiere $\frac{f}{2} + 1$ rondas para resolver el LAP.*

[1] Una anticadena es un subconjunto S de X (conjunto parcialmente ordenado) en la que entre cada par de elementos que pertenecen a S no existe alguna relación

Capítulo 4

Algoritmo LA_M

En [11] Mavronicolas aborda un algoritmo, el cual es una modificación del algoritmo 1 presentado en el capítulo anterior. Hasta ese momento era el algoritmo con mejor tiempo para encontrar una solución al problema. Mavronicolas demuestra que el orden de complejidad de rondas está acotado por mín$\{O(h(\mathcal{J}(X))), O(\sqrt{f})\}$. En este capítulo se estudiará el algoritmo LA_M y se va a demostrar que la cota bajo la cual está acotado el algoritmo es justa.

4.1. Algoritmo propuesto por Mavronicolas

En esta sección se estudiará el algoritmo propuesto por Mavronicolas [11]. El algoritmo propuesto es capaz de tolerar $f < n$ fallas de paro, además tiene la propiedad de detención temprana ($early - stopping$) y fue el primero en donde la complejidad de rondas depende del tamaño ($height$) de la retícula de entrada. Uno de nuestros objetivos principales es corroborar la cota propuesta por Mavronicolas y al mismo tiempo demostrar si es una cota justa.

Primero se va a demostrar que el algoritmo es correcto. Antes de entrar en contexto se va a describir el comportamiento del algoritmo.

- El proceso recibe su valor de entrada (x_i).
- Se usa una variable auxiliar para almacenar el valor de entrada ($v_i = x_i$).
- Se instancía una variable $bandera = true$.
- Se vá a ejecutar la estructura de control $while$ hasta que la variable $bandera$ sea $falsa$.
- El proceso comparte su valor (v_i) con todos los demás procesos.
- El proceso almacena los valores recibidos en la ronda (U), suponemos que recibe su propio valor.
- Se ejecuta la condición $v_i \leq u$ o $v_i \geq u$, $\forall u \in U$.

- Si es verdadera la variable *bandera* se le asigna el valor falso y regresa v_i.
- En caso contrario, el proceso aplica la operación *join* a los valores recibidos y almacena el resultado en la variable v_i.

- Finalmente produce su valor de salida y_i.

Algoritmo 2: $LA_M(x_i)$ para un proceso p_i

Datos: x_i = valor de entrada
Resultado: y_i = valor de salida

```
1  v_i = x_i
2  bandera = true
3  mientras bandera hacer
4  |  enviar a todos (v_i)
5  |  U = {v_j| recive v_j de p_j}
6  |  si v_i ≤ u o v_i ≥ u, ∀u ∈ U entonces
7  |  |  bandera = false
8  |  |  devolver v_i
9  |  en otro caso
10 |  |  v_i = ⊔{u|u ∈ U}
11 |  fin
12 fin
13 y_i = v_i
```

4.1.1. Correctez del algoritmo

Se demostrará que la decisión de un proceso satisface las tres propiedades del problema de acuerdo de retícula.

- Downward-Validity.
- Upward-Validity.
- Comparabilidad.

Primero se va a demostrar que se satisface la propiedad Downward-validity y posteriormente Upward-validity:

Lema 4.1.1. *Sea p_i un proceso correcto y y_i su valor de salida, entonces al final de procesar el algoritmo 2, $x_i \leq y_i$ (propiedad Downward − validity).*

Demostración. Se demostrará que la invariante: $x_i \leq v_i^r$ se preserva al término de cada ronda r para un proceso p_i.

- Caso base cuando $r = 1$, sea p_i un proceso correcto.

  De la línea 1, tenemos como valor inicial de $v_i = x_i$.

De la línea 4, el proceso p_i envía su valor v_i a todos los procesos.

De la línea 5, se tiene U_i^1 como el conjunto de los valores recibidos por el proceso p_i, el cual incluye v_i.

De la línea 6, se tienen dos casos: si v_i es comparable con todos los valores $u \in U_i^1$, entonces termina y $y_i = x_i$.

Por otro lado, de la línea 10, $v_i^1 = \sqcup U_i^1$. Como $x_i = v_i \in U_i^1$, entonces al obtener el *join* de U_i^1, se tiene que $x_i \leq \sqcup U_i^1 = v_i^1$ al final de la ronda 1.

- Hipótesis de inducción: Supongamos que la invariante $x_i \leq v_i^r$ se preserva al término de cada ronda r.

- Por demostrar que la invariante se mantiene al término de la ronda $r + 1$.

  Al inicio de la ronda $r + 1$ se tiene como valor de entrada v_i^r y a su vez, este lo comparte a los demás procesos (línea 4).

  U_i^{r+1} (línea 5) el conjunto de los valores recibidos por el proceso p_i en la ronda $r+1$, el cual incluye v_i^r.

  De la línea 6, se tienen dos casos: Si v_i^r es comparable para todo $u \in U_i^{r+1}$, entonces termina y $y_i = v_i^r$, por hipótesis de inducción $x_i \leq v_i^r$.

  Por otro lado, de la línea 10 $v_i^{r+1} = \sqcup U_i^{r+1}$, como $v_i^r \in U_i^{r+1}$, entonces al obtener el *join* de U_i^{r+1} se tiene que $v_i^r \leq v_i^{r+1}$ al final de la ronda $r + 1$, por hipótesis de inducción $x_i \leq v_i^r \leq v_i^{r+1}$.

■

Lema 4.1.2. *Sea p_i un proceso correcto y y_i su valor de salida, entonces al final de procesar el algoritmo 2, $y_i \leq \sqcup X$ (propiedad Upward − validity).*

Demostración. Se demostrará que la invariante $v_i^r \leq \sqcup X$ se preserva al término de cada ronda r para un proceso p_i.

- Caso base cuando r=1, sea p_i un proceso correcto.

  De la línea 1, tenemos como valor inicial de $v_i = x_i$.

  De la línea 4, el proceso p_i envía su valor v_i a todos los procesos.

  De la línea 5, se tiene U_i^1 como el conjunto de los valores recibidos por el proceso p_i, el cual incluye v_i.

  De esta forma de la línea 6 se tienen dos casos: si v_i es comparable con todos los valores $u \in U_i^1$, entonces termina y $y_i = x_i \leq \sqcup X$.

  En caso contrario, de la línea 10, $v_i^1 = \sqcup U_i^1$, donde cada valor $u \in U_i^1$ se tiene que $u \in X$. Por lo tanto, $U_i^1 \subseteq X$ entonces $v_i^1 \leq \sqcup X$.

- Hipótesis de inducción: Supongamos que la invariante $v_i^r \leq \sqcup X$ se preserva al término de cada ronda r.

- Por demostrar que la invariante se mantiene al término de la ronda $r+1$.

  Al inicio de la ronda $r+1$ se tiene como valor de entrada v_i^r y a su vez, este lo comparte a los demás procesos (línea 4).

  U_i^{r+1} (línea 5) el conjunto de los valores recibidos por el proceso p_i en la ronda $r+1$, el cual incluye v_i^r.

  De la línea 6, se tienen dos casos: si v_i^r es comparable con todo $u \in U_i^{r+1}$, entonces termina y $y_i = x_i^r \leq \sqcup X$.

  Por otro lado, de la línea 10 $v_i^{r+1} = \sqcup U_i^{r+1}$, por hipótesis de inducción para cada $u \in U_i^{r+1}$ se tiene que $u \leq \sqcup X$. Por lo tanto, $v_i^{r+1} \leq \sqcup X$.

■

Lema 4.1.3. *Sea p_i un proceso correcto entonces se cumple $v_i < v_i^1 < v_i^2 < ... < v_i^{r^d-1} = y_i$, donde r^d es la ronda en la que el proceso p_i decide y termina.*

Demostración. Se demostrará que la invariante $v_i^{r-1} < v_i^r$ se preserva al término de cada ronda r para un proceso p_i correcto.

- Caso base cuando $r = 1$, sea p_i un proceso correcto:

  De la línea 1, tenemos como valor inicial de $v_i = x_i$.

  Procesadas las líneas 4 y 5, se tiene el conjunto U_i^1 como el conjunto de los valores recibidos por el proceso p_i.

  De la línea 6 se tienen dos casos:

  - Si v_i es comparable con todos los valores $u \in U_i^1$, entonces termina y $y_i = v_i$.
  - En caso contrario, de la línea 10 $v_i^1 = \sqcup U_i^1$. Por lo tanto, $v_i \leq v_i^1$. Como v_i se está actualizando implica que existe al menos $u \in U_i^1$, tal que no es comparable con v_i. Por lo tanto, $v_i < v_i^1$.

  Cuando $r = 2$:

  Se tiene como valor de entrada en la ronda 2, v_i^1.

  Procesadas las líneas 4 y 5, se tiene el conjunto U_i^2 como el conjunto de los valores recibidos por el proceso p_i.

  De la línea 6 se tienen dos casos:

  - Si v_i^1 es comparable con todos los valores $u \in U_i^2$, entonces termina y $y_i = v_i^1$. Por lo tanto, $v_i < v_i^1 = y_i$.
  - En caso contrario, de la línea 10 $v_i^2 = \sqcup U_i^2$. Por lo tanto, $v_i^1 \leq v_i^2$. Como v_i^1 se está actualizando implica que existe al menos $u \in U_i^2$, tal que no es comparable con v_i^1. Por lo tanto, $v_i^1 < v_i^2$.

- Hipótesis de inducción: Supongamos que la invariante $v_i^{r-1} < v_i^r$ se preserva al término de cada ronda r.

- Por demostrar que la invariante se mantiene al final de la ronda $r+1$.

  Se tiene como valor de entrada en la ronda $r+1$, v_i^r.

  Procesadas las líneas 4 y 5, se tiene el conjunto U_i^{r+1} como el conjunto de los valores recibidos por el proceso p_i.

  De la línea 6 se tienen dos casos:

  - Si v_i^r es comparable con todos los valores $u \in U_i^{r+1}$, entonces termina y $y_i = v_i^r$. Por lo tanto, $v_i^{r-1} < v_i^r = y_i$.
  - En caso contrario, de la línea 10 $v_i^{r+1} = \sqcup U_i^{r+1}$. Por lo tanto, $v_i^r \leq v_i^{r+1}$. Como v_i^r se está actualizando implica que existe al menos $u \in U_i^{r+1}$, tal que no es comparable con v_i^r. Por lo tanto, $v_i^r < v_i^{r+1}$.

■

El valor v_i^r para cada proceso correcto p_i al final de la ronda es no decreciente. Además, se encuentra acotado por $\sqcup X$. Por último se va a demostrar que entre cada par de procesos correctos al terminar el algoritmo sus valores que deciden son comparables entre sí.

Lema 4.1.4. *Sea p_i un proceso tal que decide en la ronda r su valor y_i, entonces y_i es comparable con v_j^r de un proceso correcto p_j.*

Demostración. Sea p_i un proceso correcto que decide en la ronda r, se tienen dos casos a analizar:

- Sea p_j un proceso correcto tal que decide en la ronda r. Como p_j es un proceso correcto, entonces p_i recibió el valor v_j^r . De esta forma, dado que p_i decidió en la ronda r (línea 6) y_i es comparable con $y_j = v_j^r$.
- Sea p_j tal que no decide en la ronda r. Como p_i es un proceso correcto, entonces p_j recibió v_i^r y lo almacena en U_j^r (línea 5). Por lo tanto, $y_i \leq \sqcup U_j^r = v_j^r$

■

Lema 4.1.5. *Sean p_i y p_j procesos correctos, entonces al final de procesar el algoritmo sus valores y_i y y_j son comparables.*

Demostración. Sea p_i y p_j dos procesos correctos que terminan en las rondas r_i y r_j respectivamente. Sin pérdida de generalidad, supongamos que $r_i \leq r_j$. De 4.1.4, y_i es comparable con $v_k^{r_i}$ para cualquier proceso correcto en la ronda r_i. $V = \{v_k^{r_i} |$ donde p_k no decidió en la ronda $r_i\}$.

Como $r_j \geq r_i$, entonces $U_j^{r_j} \subseteq V$ y como y_i es comparable para todo $u \in V$. Por lo tanto, y_i es comparable con y_j.

■

Los lemas 4.1.1, 4.1.2 y 4.1.5 implican el siguiente teorema.

Teorema 4.1.6. *El algoritmo LA_M (2) resuelve el problema de acuerdo de retícula.*

4.1.2. Complejidad computacional

En [11] demuestran que la cota superior para el cual el algoritmo 2 resuelve el problema de acuerdo de retícula se tiene en el siguiente teorema.

Teorema 4.1.7 ([11],Teorema 2)**.** *El algoritmo LA_M (2) resuelve el problema de acuerdo de retícula en* $\min\{1+h(\mathcal{J}(X)), \lfloor(3+\sqrt{8f+1})/2\rfloor\}$ *rondas, para cada ejecución en la que los procesos $p_1, p_2, ..., p_n$ inician con valores de entrada $x_1, x_2, ..., x_n$ y f procesos fallan.*

En el presente trabajo se logró ajustar la cota para el orden de complejidad del algoritmo LA_M en $\min\{1+\mathrm{h}(\mathcal{J}(X)), \lceil(1+\sqrt{8f+1})/2\rceil\}$. La idea de la demostración se basa en la presentada por Mavronicolas en [11] para el teorema 4.1.7.

Lema 4.1.8 ([11],Lema 3)**.** *Sea p_i un proceso correcto y $1 \leq r$ entonces $v_i^r \in \mathcal{J}(X)$.*

Demostración. Se demostrará que la invariante $v_i^r \in \mathcal{J}(X)$ se preserva al término de cada ronda r para un proceso p_i correcto.

- Caso base cuando $r = 1$, sea p_i un proceso correcto. De la línea 1, tenemos como valor inicial de $v_i = x_i$.

  Procesadas las líneas 4 y 5, se tiene el conjunto U_i^1 como el conjunto de los valores recibidos por el proceso p_i.

  De la línea 6 se tienen dos casos:

  - Si v_i es comparable con todos los valores $u \in U_i^1$, entonces termina y $y_i = v_i = x_i \in \mathcal{J}(X)$.
  - En caso contrario, de la línea 10 $v_i^1 = \sqcup U_i^1$. Como $\forall u \in U_i^1$ $u \in \mathcal{J}(X)$ y de la definición 2.2.9 en el punto 2 entonces $v_i^1 \in \mathcal{J}(X)$.

- Hipótesis de inducción: Supongamos que la invariante $v_i^r \in \mathcal{J}(X)$ se preserva al término de cada ronda r.

- Por demostrar que la invariante se mantiene al final de la ronda $r+1$.

  Se tiene como valor de entrada en la ronda $r+1$, v_i^r.

  Procesadas las líneas 4 y 5, se tiene el conjunto U_i^{r+1} como el conjunto de valores recibidos por el proceso p_i.

  De la línea 6 se tienen dos casos:

  - Si v_i^r es comparable con todos los valores $u \in U_i^{r+1}$, entonces termina y $y_i = v_i^r$. Por hipótesis $y_i \in \mathcal{J}(X)$.
  - En caso contrario, de la línea 10 $v_i^{r+1} = \sqcup U_i^{r+1}$. Por hipótesis $\forall u \in U_i^{r+1}$, $u \in \mathcal{J}(X)$. Por la definición 2.2.9 en el punto 2, entonces $v_i^{r+1} \in \mathcal{J}(X)$.

■

Proposición 4.1.9. *Sea r una ronda, donde ocurren $f_r < f$ fallas, y sean p_i y p_j procesos correctos que no deciden. Si p_i recibe únicamente los valores de los procesos que no fallan en r entonces v_i^r es comparable con v_j^r.*

Demostración. Sea V^r el conjunto de valores de los procesos que no fallan en la ronda r y S_f un subconjunto de los valores de los procesos que fallan en r. De la línea 5 del algoritmo 2, $U_i^r = V^r$ y $U_j^r = V^r \cup S_f$. $U_i^r \subseteq U_j^r$ entonces $v_i^r \leq v_j^r$. ■

Lema 4.1.10 ([11],Afirmación 8)**.** *Sea $r_0 \geq 1$ donde ocurren $f_{r_0} < f$ fallas entonces cualquier proceso correcto decide en $f_{r_0} + r_0$ rondas.*

Demostración. Sin pérdida de generalidad, supongamos que los procesos que fallan en la ronda r_0 son $p_1, p_2, ..., p_{f_{r_0}}$. Mientras que el conjunto de procesos $p_{f_{r_0+1}}, ..., p_n$ son los que comparten su mensaje a todos los procesos.

Por la proposición 4.1.8, sea p_i un proceso correcto y $r \geq r_0$ entonces $v_i^r \in \mathcal{J}(X)$. Además, por el teorema 2.2.10, $v_i^r = \sqcup\{\{x_{f_{r_0}+1}, .., x_n\} \cup S_k\}$, donde $S_k \subseteq \{x_1, x_2, ..., x_{f_{r_0}}\}$ con $k \leq f_{r_0}$.

Por otro lado, del lema 4.1.3, sea p_i un proceso correcto entonces $v_i^{r_0} < v_i^{r_0+1} < ... < v_i^{r^d-1} = y_i$. Además, por la proposición 4.1.9 en la ronda r_0, p_i no solo recibe los valores de los procesos que no fallan. Por estas dos observaciones la longitud de esta sucesión va a deberse a que p_i permanezca en cada ronda conociendo un valor nuevo cada vez, es decir conozca $u \in \{x_1, x_2, ..., x_{f_{r_0}}\}$ en cada ronda. La longitud máxima que puede tener la sucesión es f_{r_0}. Si dicha cota fuera mayor implicaría que existen dos rondas tal que conoce al mismo valor.

De esta forma, la sucesión de valores para un proceso p_i correcto al final de cada ronda se puede visualizar de la manera siguiente:

- $v_i^{r_0} = \sqcup\{\{x_{f_{r_0}+1}, .., x_n\} \cup S_1\}$.
- $v_i^{r_0+1} = \sqcup\{\{x_{f_{r_0}+1}, .., x_n\} \cup S_2\}$.
- ...
- $v_i^{r_0+f_{r_0}-1} = \sqcup\{\{x_{f_{r_0}+1}, .., x_n\} \cup S_{f_{r_0}}\}$.

Donde $S_1 \subset S_2 \subset S_3 \subset ... \subset S_{f_{r_0}} = \{x_1, x_2, ..., x_{f_{r_0}}\}$ y la cardinalidad está dada por $|S_k| = k$, $k \in \{1, 2, 3, ..., f_{r_0}\}$.

Por otro lado, en el algoritmo línea 8 se requiere que para terminar debe de ser comparable con los valores recibidos. El valor de la ronda $r_0 + f_{r_0} - 1$ es comparable con cualquier valor recibido previamente y ya no existe un valor que pueda conocer. Por lo tanto, en la ronda $r_0 + f_{r_0}$ puede decidir y terminar.

Por lo cual el proceso p_i decide en a lo más $r_0 + f_{r_0}$ rondas.

■

Sea l el número de rondas necesarias para resolver el LAP en el algoritmo 2. Por lo que $1 \leq r_0 < l$, además por el lema 4.1.10, $l \leq f_{r_0} + r_0$. Así tenemos que $l - r_0 \leq f_{r_0}$. Aplicando la suma sobre $l - 1$, debido a que $r_0 < l$ se tiene:

$$\sum_{r_0=1}^{l-1}(l-r_0) \leq \sum_{r_0=1}^{l-1}(f_{r_0}) \tag{4.1}$$

En el lado izquierdo de la ecuación se tiene una suma sobre un elemento que en cada índice va decreciendo:

$$\sum_{r_0=1}^{l-1}(l-r_0) = (l-1)+(l-2)+...+1 = \sum_{r_0=1}^{l-1} r_0. \tag{4.2}$$

Mientras que en el lado derecho al estar operando sobre las fallas que ocurren en cada ronda $r_0 < l$ se tiene:

$$\sum_{r_0=1}^{l-1} f_{r_0} = f_{r_1} + f_{r_2} + ... + f_{r_{l-1}} \leq f. \tag{4.3}$$

Por 4.2 y 4.3 se tiene la siguiente desigualdad:

$$\begin{aligned} \sum_{r_0=1}^{l-1} r_0 &\leq f \\ \frac{(l-1)l}{2} &\leq f \\ l^2 - l &\leq 2f \\ l^2 - l - 2f &\leq 0 \end{aligned} \tag{4.4}$$

De la ecuación 4.4, se va a resolver $l^2 - l - 2f = 0$

$$l = \frac{1 \pm \sqrt{1+4(2f)}}{2} = \frac{1 \pm \sqrt{8f+1}}{2} \tag{4.5}$$

Como el número de rondas es un entero positivo nos quedamos con la parte positiva de la ecuación, es decir, $l = \frac{1+\sqrt{8f+1}}{2} \leq \lceil \frac{1+\sqrt{8f+1}}{2} \rceil$.

Lema 4.1.11 ([11],Lema 5)**.** *El algoritmo 2 resuelve el problema de acuerdo de retícula en* $\lceil 0.5 + \sqrt{2f + 0.25} \rceil$ *rondas, donde ocurren f fallas.*

Con lo cual se tiene una cota superior para el algoritmo propuesto (algoritmo 2). Por otro lado, nos gustaría corroborar si dicha cota es justa. Por lo tanto, se va a construir una ejecución de tal forma que nos permita determinar lo anterior.

Cota inferior

Tomemos un conjunto (P) de n procesos sobre una retícula $\mathcal{L}$ con a lo más f fallas (sin pérdida de generalidad $p_1, p_2, ..., p_f$). Denotaremos a P_C como el conjunto de los $n-f$ procesos correctos.

Vamos a trabajar sobre una retícula booleana de subconjuntos, la cual queda definida de la siguiente manera: Sea $S = \{a_1, a_2, ..., a_{f_1}, a\}$ tal que $|S| = f_1 + 1$, donde f_1 es el número de procesos que fallan en la primera ronda, entonces vamos a trabajar la retícula sobre $Pow(S)$.

Por otro lado, se va a determinar cuanto vale f_1 además de determinar cuantos procesos fallan en cada ronda. En primera instancia requerimos que en cada ronda falle un proceso menos que en la ronda anterior, es decir si en la ronda 1 fallan f_1 procesos entonces en la ronda 2 fallan $f_1 - 1$ procesos y así consecutivamente. Por lo tanto:

$$f = \sum_{i=1}^{f_1} i = \frac{f_1(f_1 + 1)}{2} \tag{4.6}$$

es decir, necesitamos resolver $f_1^2 + f_1 - 2f = 0$ por lo tanto:

$$f_1 = \frac{-1 \pm \sqrt{1 + 8f}}{2} \tag{4.7}$$

debido a que $f_1 > 0$ nos tomamos la raíz positiva, es decir, $f_1 = \lfloor -0.5 + \sqrt{0.25 + 2f} \rfloor$.

Una vez definida la retícula sobre la que vamos a trabajar y f_1. Procedemos a definir la ejecución y como se va a llevar a cabo:

- F_1 es el conjunto de procesos que fallan en la ronda 1 y consta de $f_1 = \lfloor -0.5 + \sqrt{2 * f + 1/4} \rfloor$ procesos.
- El valor entrada (x_i) para cada proceso $p_i \in F_1$ es $x_i = a_i$, $i \in \{1, 2, .., f_1\}$.
- Sea A_1 el conjunto $P \setminus F_1$.
- Cada proceso $p_j \in A_1$ tiene como valor de entrada $x_j = a$.
- Obsérvese que los valores de entrada de los procesos en F_1 y el valor de entrada de los procesos pertenecientes a A_1 forman una anticadena.
- El envío de mensajes en la primera ronda se va a llevar a cabo de la siguiente manera (figura 4.1):
  - El conjunto F_2 está conformado por los primeros $f_2 = f_1 - 1$ procesos del conjunto A_1.
  - El proceso $p_i \in F_1$ envía un mensaje únicamente al proceso $q_i \in F_2$, $i \in \{1, 2, .., f_1 - 1\}$.
  - Sea A_2 el conjunto $A_1 \setminus F_2$.
  - El proceso $p_{f_1} \in F_1$ envía un mensaje únicamente a cada uno de los procesos que pertenecen al conjunto A_2.
  - Finalmente, los procesos $p \in A_1$ envían su mensaje a todos los procesos.

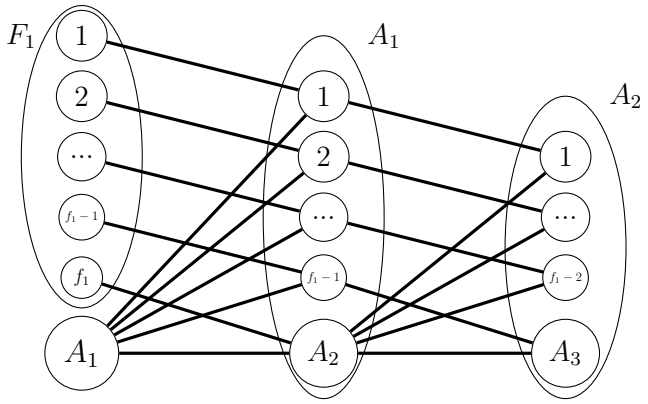

Figura 4.1: Ejecución de la primera y segunda ronda.

- En las rondas subsecuentes se continúa de la siguiente manera: Sea $2 \leq r \leq f_1$ tal que F_r es el conjunto de procesos que fallan en la ronda r y el número de procesos es $f_r = f_1 - r + 1$.
- Por otro lado, A_r es el conjunto de procesos que no fallan en la ronda r.
- El envío de mensajes continúa de la siguiente manera (figura 4.2) :
  - Sea F_{r+1} el conjunto de los primeros $f_{r+1} = f_r - 1$ procesos del conjunto A_r.
  - El proceso $p_i \in F_r$ envía un mensaje únicamente al proceso $q_i \in F_{r+1}$, $i \in \{1, .., f_r - 1\}$.
  - Sea A_{r+1} el conjunto $A_r \setminus F_{r+1}$.
  - El proceso p_{f_r} envía un mensaje únicamente a cada uno de los procesos que pertenecen al conjunto A_{r+1}.
  - Finalmente, los procesos $p \in A_r$ envían su mensaje a todos los procesos.

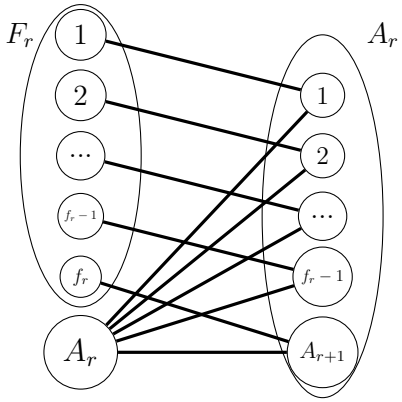

Figura 4.2: Ejecución de la ronda r.

De acuerdo con la ejecución enunciada anteriormente, se tiene que en cada ronda ocurre una falla menos que en la ronda anterior, $f_1 > f_2 > ... > 2 > 1$. Para denotar que un proceso p_i pertenece a un conjunto A se usara la siguiente notación p_i^A o $p_i \in A$.

Se va a demostrar por inducción que al final de cada ronda $1 \leq r \leq f_1 - 1$ existe al menos un proceso que no ha decidido.

- Caso base: Cuando r=1. De acuerdo con la ejecución, el conjunto F_1 contiene a los primeros f_1 procesos de P, donde para cada $p_i \in F_1$ tiene como valor de entrada $v_i = x_i$. Mientras los procesos restantes $p_j \in A_1$ tienen como valor de entrada $v_a = a$. Además, cada proceso $p_i \in F_2$ recibe los valores $\{x_i\}$ y $\{a\}$, donde F_2 es el conjunto formado por los primeros $f_1 - 1$ procesos del conjunto A_1. Finalmente, los procesos $p_j \in A_2$ reciben los valores $\{x_{f_1}\}$ y $\{a\}$. De acuerdo con la ejecución, se tiene que $A_1 = A_2 \cup F_2$.

  Por lo tanto, cada proceso $p_i \in A_1$ no decide y aplica la operación *join* sobre los valores recibidos, quedando de la siguiente manera: cada procesos $p_i \in F_2$ tiene como valor al final de la ronda 1, $v_i^1 = \{x_i, a\}$. Por otro lado, cada proceso $p_j \in A_2$ tiene como valor al final de la ronda 1, $v_j^1 = \{v_f, a\}$. Se observa que el conjunto de los valores de los procesos forman una anticadena, es decir, $X_1 = \{\{x_1, a\}, \{x_2, a\}, ..., \{x_{f_1}, a\}\}$ es el conjunto de valores recibidos en la ronda 1. Por lo tanto, los procesos no deciden.

- Hipótesis de inducción: Se $1 \leq r < f_1 - 1$. Al final de cada ronda r, existe al menos un proceso que no ha decidido. Además, el conjunto de los valores de los procesos que pertenecen al conjunto A_r forman una anticadena, es decir, $X_r = \{\{x_1, x_{f_1-r+2}, ..., a\}, \{x_2, x_{f_1-r+2}, ..., a\}, ..., \{x_{f_1-r+1}, x_{f_1-r+2}, .., x_f, a\}\}$, con $x_{f_1+1} = a$. Donde, $A_r = A_{r+1} \cup F_{r+1}$, y se tiene que cada proceso $p_i \in F_{r+1}$ tiene como valor al final de la ronda r, $v_i^r = \{x_i, x_{f_1-r+2}, ..., a\}$. Por otro lado, cada proceso $p_j \in A_{r+1}$ tiene como valor al final de la ronda r, $v_j^r = \{x_{f_1-r+1}, x_{f_1-r+2}, ..., a\}$.

- Por demostrar que la hipótesis se mantiene al final de la ronda $r + 1$.

  Por hipótesis, $p_i \in F_{r+1}$ tiene como valor al inicio de la ronda $r+1$, $v_i^r = \{x_i, x_{f_1-r+2}, ..., a\}$. Mientras para cada proceso $p_j \in A_{r+1}$ tiene como valor al inicio de la ronda $r + 1$, $v_j^r = \{x_{f_1-r+1}, x_{f_1-r+2}, ..., a\}$. De acuerdo con la ejecución, cada proceso $p_i \in F_{r+2}$ recibe los valores $\{x_i, x_{f_1-r+2}, ..., a\}$ y $\{x_{f_1-r+1}, x_{f_1-r+2}, ..., a\}$, donde F_{r+2} es el conjunto de los primeros $f_{r+1} - 1$ procesos del conjunto A_{r+1}. Finalmente, los procesos $p_j \in A_{r+2}$ reciben los valores $\{x_{f_1-r}, x_{f_1-r+2}, ..., a\}$ y $\{x_{f_1-r+1}, x_{f_1-r+2}, ..., a\}$.Donde, $A_{r+1} = F_{r+2} \cup A_{r+2}$.

  Por lo tanto, cada proceso $p_i \in A_{r+1}$ no decide y aplica la operación *join* sobre los valores recibidos, quedando de la siguiente manera: cada proceso $p_i \in F_{r+2}$ tiene como valor al final de la ronda $r + 1$, $v_i^{r+1} = \{x_i, x_{f_1-r+1}, x_{f_1-r+2}, ..., a\}$. Por otro lado, cada proceso $p_j \in A_{r+2}$ tiene como valor al final de la ronda $r + 1$,$v_j^{r+1} = \{x_{f_1-r}, x_{f_1-r+1}, x_{f_1-r+2}, ..., a\}$. Se observa que el conjunto de los valores de los procesos que pertenecen al conjunto A_{r+1} forman una anticadena, es decir, $X_{r+1} = \{\{x_1, x_{f_1-r+1}, x_{f_1-r+2}, ..., a\}, \{x_2, x_{f_1-r+1}, x_{f_1-r+2}, ..., a\}, ..., \{x_{f_1-r}, x_{f_1-r+1}, x_{f_1-r+2}, .., x_f, a\}\}$. Por lo tanto, los procesos no deciden.

Por lo tanto, al final de cada ronda $1 \leq r \leq f_1 - 1$ los procesos que pertenecen al conjunto A_r no deciden, es decir, el conjunto de los valores de los procesos que pertenecen al conjunto A_r forman una anticadena:

$X_r = \{\{x_1, x_{f_1-r+2}, ..., a\}, \{x_2, x_{f_1-r+2}, ..., a\}, ..., \{x_{f_1-r+1}, x_{f_1-r+2}, .., x_f, a\}\}$, donde $|X_r| = f_1 - r + 1$.

En la ronda $f_1 - 1$, el conjunto X_{f_1-1} tiene dos elementos, $|X_{f_1-1}| = 2$. Por lo tanto, en la ronda f_1 (figura 4.3) se tienen dos conjuntos: el primero formado por un proceso perteneciente al conjunto F_{f_1} y el segundo por el conjunto A_{f_1}, los cuales envían sus mensajes al conjunto $A_{f_1+1} = P_C$. Al inicio de la ronda f_1 cada proceso $p_i \in A_{f_1}$ tiene como valor $v_i^{f_1-1} = \{x_2, x_3, .., x_f, a\}$, mientras que $p_1^{F_{f_1}}$ tiene como valor $v_1^{f_1-1} = \{x_1, x_3, x_4, .., x_f, a\}$. De la ejecución, cada proceso $p_i \in A_{f_1+1}$ recibe los valores $\{x_2, x_3, .., x_f, a\}$ y $\{x_1, x_3, x_4, .., x_f, a\}$. Por lo tanto, cada proceso $p_i \in A_{f_1+1}$ no decide y aplica la operación *join* sobre el conjunto de valores recibidos, donde al final de la ronda f_1, $v_i^{f_1} = \{x_1, x_2, x_3, .., x_f, a\}$ $\forall p_i \in A_{f_1+1} = P_C$. Con lo cual cada proceso en el conjunto P_C es comparable entre sí.

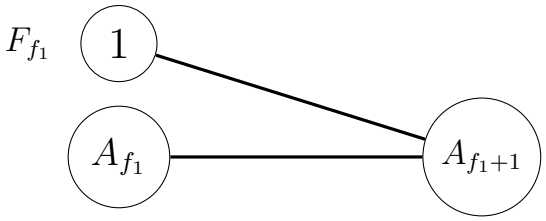

Figura 4.3: Ejecución de la ronda f_1.

Para la ejecución definida anteriormente, el algoritmo LA_M requiere f_1 rondas para resolver el LAP. Por otro lado, observemos que en la ejecución ocurren f fallas, es decir, si $R < \lfloor\sqrt{2 * f + 0.25} - 0.5\rfloor$ rondas, entonces no se satisface la propiedad de comparabilidad. Observación, si $f = 0$ entonces $\lfloor\sqrt{2 * f + 0.25} - 0.5\rfloor = 0$, por lo tanto se requieren $\lfloor\sqrt{2 * f + 0.25} - 0.5\rfloor + 1$ rondas. Denotemos a l como el número de rondas requeridas en el algoritmo 2 para resolver el LAP, entonces $\lfloor\sqrt{2 * f + 0.25} + 0.5\rfloor \leq l \leq \lceil 0.5 + \sqrt{2f + 0.25}\rceil$ rondas. De los argumentos anteriores podemos concluir el siguiente teorema respecto al algoritmo LA_M (2).

Teorema 4.1.12. *Existe una ejecución tal que el algoritmo LA_M requiere $\lfloor 0.5 + \sqrt{0.25 + 2f}\rfloor$ rondas para resolver el LAP.*

Lema 4.1.13 ([11], Lema 4)**.** *El algoritmo 2 resuelve el problema de acuerdo de retícula en $h(\mathcal{J}(X)) + 1$ rondas, donde ocurren f fallas.*

Los lemas 4.1.13 y 4.1.11 implican el siguiente teorema.

Teorema 4.1.14. *El algoritmo LA_M (2) resuelve el problema de acuerdo de retícula en* mín$\{1 + h(\mathcal{J}(X)), \lceil(1 + \sqrt{8f + 1})/2\rceil\}$ *rondas, para cada ejecución en la que los procesos $p_1, p_2, ..., p_n$ inician con valores de entrada $x_1, x_2, ..., x_n$ y f procesos fallan.*

Capítulo 5

Algoritmo LA_α

En [14] Xiong Zheng presenta un primer algoritmo de orden logarítmico que bajo ciertos parámetros resuelve el LAP. En este capítulo se estudiará el algoritmo LA_α y al final se corroborara que la cota bajo la cual está acotado el algoritmo es justa.

5.1. Algoritmo LA_α propuesto por Xiong Zheng

Hasta el momento se han visto dos algoritmos y con modificaciones simples, tales como meter la comparabilidad dentro del algoritmo reduce su complejidad drásticamente. Sin embargo, cuando se trabaja en algoritmos y el problema que resuelven una de las preguntas habituales es ¿cuál es el orden de complejidad del problema?. Esta respuesta no es algo fácil de abordar, sin embargo la forma en que se puede estudiar es ver si se puede proponer un algoritmo que resuelva una versión simplificada del problema.

Ahora bien, en esta sección se va a estudiar el algoritmo LA_α [14], con la finalidad de corroborar y demostrar la correctez y si la cota dada por Xiong Zheng es justa.

Como se mencionó anteriormente, el orden de complejidad del algoritmo 2 está en función al tamaño de la retícula de entrada. De esta forma Xiong Zheng en [14] aborda de primera mano el LAP con una variedad, esta variedad va referida a la suposición de que se tiene el conocimiento del *tamaño* de la retícula de entrada por cada proceso.

5.1.1. Procedimiento del clasificador síncrono

El algoritmo LA_α tiene una complejidad de orden $O(log(H))$, dónde H es la altura de la retícula de entrada. Una de las características principales es que $H \leq n$ y tolera $f < n$ fallas.

La parte que se encarga de llevar a cabo la clasificación y de esta manera dividir el conjunto de procesos en dos subgrupos, recursivamente, es el clasificador síncrono (algoritmo 3). El primer grupo es *dominantes* (*Master*) y el segundo grupo *dominados* (*Slave*). El clasificador nos permite que si un proceso pertenece al grupo de *slave* entonces su valor es a lo más el valor de un proceso que pertenezca al grupo *master* (comparable).

El procedimiento 3 es instanciado por cada proceso p_i, donde pasa como parámetros v_i el valor que actualmente tiene, y k_i el parámetro de umbral. El parámetro de umbral

ayuda a identificar la iteración o grupo al que pertenece p_i.

Descripción del algoritmo del clasificador síncrono:

- El proceso envía un mensaje a todos los procesos, el mensaje incluye el valor de entrada v_i y el parámetro k_i.
- El proceso recibe un conjunto de mensajes con la estructura (u, k) y almacena los valores u en el conjunto U, aquellos que comparten el mismo parámetro k. Se asume que recibe su propio mensaje.
- Condición de *terminación temprana*: si el valor v_i es comparable con todo valor $u \in U$, entonces termina el procedimiento y devuelve el valor de entrada como valor de salida y verdadero como el estado actual.
- Se aplica la operación $join$ al conjunto U y se almacena en una variable w.
- Condición para la clasificación en subgrupos:
  - *Master*: Se determina la altura de w en la retícula $\mathcal{L}$, si esta es mayor que el parámetro k_i, entonces el procedimiento devuelve w como valor de salida, asigna *master* como el grupo en el cual se ha clasificado al proceso y *false* como el estado actual de decisión para p_i $(w, master, false)$.
  - *Slave*: En caso contrario, el procedimiento devuelve como valor de salida v_i, *slave* como el grupo en el cual se ha clasificado el proceso y *false* como el resultado actual de decisión para p_i $(w, slave, false)$.

Algoritmo 3: Clasificador-síncrono(v_i, k_i) para un proceso p_i

Datos: v_i = valor de entrada, k_i = parámetro de umbral
1 *enviar* a todos (v_i, k_i)
2 $U = \{v_j |\ (v_j, k_j)$ recibida de p_j que comparten el mismo parámetro de umbral$\}$
3 **si** $|U| = 0$ *o* $v_i \leq u$ *o* $v_i \geq u$, $\forall u \in U$ **entonces**
4 | **devolver** $(v_i, -, true)$
5 **fin**
6 $w = \sqcup\{u | u \in U\}$
7 **si** $h(w) > k_i$ **entonces**
8 | **devolver** $(w, master, false)$
9 **en otro caso**
10 | **devolver** $(v_i, slave, false)$
11 **fin**

Observación 5.1.1. *Si dos procesos p_i y p_j tienen el mismo parámetro de umbral de tal forma que p_i ha sido clasificado en el grupo slave y p_j en el grupo master, entonces el valor $w_j \geq v_i$. Esta observación es fácil de ver debido a que el proceso p_j ha aplicado la operación join sobre U.*

5.1.2. Algoritmo LA_α

El algoritmo principal (LA_α) se ejecuta en $\log(H)+1$ rondas, el cual hace una llamada al clasificador 3 que tiene como parámetros de entrada $H = h(\mathcal{L})$ y x_i. Una vez que se ha ejecutado el clasificador y dependiendo de la etiqueta devuelta (*master* o *slave*) se actualiza el parámetro k_i.

Definición 5.1.2. *Etiqueta (label): Cada proceso tiene una etiqueta (label), la cual sirve para conocer el umbral y se pasa como parámetro de umbral k cuando un proceso llama al procedimiento clasificador-síncrono.*

Definición 5.1.3. *Grupo: Un grupo es un conjunto de procesos que tiene la misma etiqueta. La etiqueta de un grupo es la etiqueta de los procesos en ese grupo.*

Descripción del algoritmo LA_α:

- Se usa una variable auxiliar para almacenar el valor de entrada ($v_i = x_i$).
- Se instancía la variable de *etiqueta* como $l = \frac{H}{2}$. Todo proceso inicia en el mismo umbral k.
- Se instancía una variable $bandera = false$ para determinar el estado que tiene un proceso. Todo proceso inicia como $false$ debido a que aún no han decidido.
- Se vá a ejecutar la estructura de control for hasta que r sea mayor a $\log(H)+1$.
    - Se llama al procedimiento *clasificador-síncrono* con los parámetros v_i y l. El conjunto de valores devueltos sirven para actualizar los valores en las variables v_i, cls y $bandera$.
    - Condición de parada temprana:
        - Si la $bandera$ es $true$ entonces el proceso devuelve v_i como valor decidido.
        - En caso contrario vamos asignar una nueva etiqueta dependiendo del grupo que ha regresado el clasficador (cls).
            - Si el grupo al que pertence el proceso es *master* entonces aumentamos la etiqueta en $\frac{H}{2^{r+1}}$.
            - Si el grupo al que pertence el proceso es *slave* entonces decrecemos la etiqueta en $\frac{H}{2^{r+1}}$.
- El proceso asigna como valor de salida $y_i = v_i$.

Algoritmo 4: $LA_\alpha(H, x_i)$ para un proceso p_i

Datos: H = tamaño de la retícula de entrada, v_i = valor de entrada

```
1  v_i = x
2  l = H/2
3  bandera = false
4  para r=1 hasta log(H) + 1 hacer
5  |  (v_i, cls, bandera)=clasificador-sincrono(v_i, l)
6  |  si bandera entonces
7  |  |  devolver v_i
8  |  en otro caso
9  |  |  si cls = master entonces
10 |  |  |  l = l + H/2^{r+1}
11 |  |  en otro caso
12 |  |  |  l = l - H/2^{r+1}
13 |  |  fin
14 |  fin
15 fin
16 y_i = v_i
```

5.1.3. Correctez del algoritmo

Como se mencionó en el **capítulo 3**, siempre que analicemos el algoritmo 4 se va a considerar el valor de las variables al final de la ronda, además vamos a denotar con un súper índice la ronda en la que se encuentra (por ejemplo v_i^r hace referencia al valor que tiene el proceso p_i en la ronda r). Por otro lado, en esta sección cuando hagamos referencia al valor de una variable al inicio de la ronda r, nos referimos al valor al final de la ronda anterior (por ejemplo v_i^{r-1} hace referencia al valor al final de la ronda $r-1$ y el valor al inicio de la ronda r).

Vamos a denotar como G el *grupo* de procesos en la ronda r. Además, cuando hablemos de $M(G)$ nos vamos a referir al grupo *master* de G y si tenemos $S(G)$ hacemos referencia al grupo *slave* de G. Como dato adicional diremos que G es el conjunto padre de $M(G)$ y $S(G)$ o que estos últimos son subgrupos de G.

Lema 5.1.4. *Sea G^r, tal que comparten la misma etiqueta (nodo) y $1 \leq r$ una ronda.*

1. *Sean $p_m \in M_{G^r}$ y $p_s \in S_{G^r}$ procesos correctos, entonces $v_s^r \leq v_m^r$.*
2. *Sea $p_m \in M_{G^r}$, entonces $v_m^r \geq \sqcup\{v_s^r | p_s \in S_{G^r}\}$*

Demostración. .

1. Como p_m y p_s son procesos correctos, entonces $v_s^{r-1} \in U_m^r$. Por otro lado, como $p_s \in S_{G^r}$ entonces $v_s^r = v_s^{r-1}$. Además, $v_m^r = \sqcup U_m^r$. Por lo tanto, $v_s^r \leq v_m^r$.
2. Del inciso anterior, sea $p_m \in M_{G^r}$ $\forall p_s \in S_{G^r}$, $v_s^r \leq v_m^r$. Por lo tanto, $\sqcup\{v_s^r | p_s \in S_{G^r}\} \leq v_m^r$

■

Lema 5.1.5. *Sea p_i un proceso correcto y y_i su valor de salida, entonces al final de procesar el algoritmo 4, $x_i \leq y_i$ (propiedad Downward − validity).*

Demostración. Se demostrará que la invariante $x_i \leq v_i^r$ se preserva al término de cada ronda r para un proceso p_i correcto.

- Caso base cuando $r = 1$, sea p_i un proceso correcto.

  Línea 1, tenemos como valor al inicio de la ronda 1 $v_i = x_i$.

  Línea 5, se hace llamada al procedimiento clasificador-sincrono con parámetros v_i y l.

  En el procedimiento $k_i = l$.

  Una vez ejecutadas la líneas 1 y 2 del procedimientos se tienen tres casos:

  - Si v_i es comparable con todo valor $u \in U_i^1$ entonces se devuelve $v_i = x_i$. Por lo tanto, el algoritmo devuelve como $v_i^1 = v_i = x_i$ y termina.
  - En caso de que no se cumpla la condición, se calcula la altura de $w = \sqcup U_i^1$ $(h(w))$. Si $h(w) > k_i$ entonces se devuelve $w = \sqcup U_i^1$, es decir $v_i^1 = w$. Por otro lado, como $v_i \in U_i^1$ entonces $v_i \leq w = v_i^1$ y como v_i forma una anticadena con al menos un $u \in U_i^1$ entonces $v_i < v_i^1$.
  - Si $h(w) \leq l$, entonces se devuelve $v_i = x_i$, es decir $v_i^1 = v_i = x_i$.

  Por lo tanto, $x_i \leq v_i^1$.

- Hipótesis de inducción: sea G^r un conjunto de procesos en la ronda r que comparten la misma etiqueta. Entonces $\forall p_i \in G^r$, $x_i \leq v_i^r$ al final de cada ronda r.

- Por demostrar que la invariante se mantiene al final de la ronda $r + 1$.

  Sea p_i un proceso correcto en la ronda $r + 1$ tal que $x_i \leq v_i^r$.

  De la línea 5, llama al procedimiento *clasificador síncrono* con su valor v_i^r y k_i^r.

  Una vez ejecutadas las líneas 1 y 2 del procedimiento, se tienen tres casos:

  - Si v_i^r es comparable con todo valor $u \in U_i^{r+1}$ entonces se devuelve $v_i^{r+1} = v_i^r \geq x_i$.
  - Si $p_i \in M_{G^r}$: es decir si $h(w) > k_i^{r+1}$ $(w = \sqcup U_i^{r+1})$, entonces se devuelve $v_i^{r+1} = w$, tal que, $v_i^r \in U_i^{r+1}$ y $v_i^r \geq x_i$. Por lo tanto, $v_i^{r+1} \geq x_i$.
  - Si $p_i \in S_{G^r}$ entonces $v_i^{r+1} = v_i^r \geq x_i$.

  Por lo tanto, al final de la ronda $r + 1$ se tiene $v_i^{r+1} \geq x_i$.

■

Lema 5.1.6. *Sea p_i un proceso correcto y y_i su valor de salida, entonces al final de procesar el algoritmo 4, $y_i \leq \sqcup X$ (propiedad Upward − validity).*

Demostración. Se demostrará que la invariante $v_i^r \leq \sqcup X$ se preserva al término de cada ronda r para un proceso p_i correcto.

- Caso base $r = 1$, sea p_i un proceso correcto. Línea 1, tenemos como valor al incio de la ronda 1 $v_i = x_i$.

  Línea 5, se hace llamada al procedimiento clasificador-síncrono con parámetros v_i y l.

  En el procedimiento $k_i = l$.

  Una vez ejecutas las líneas 1 y 2, se tienen tres casos:

  - Si v_i es comparable con todo valor $u \in U_i^1$ entonces se devuelve $v_i = x_i$. Por lo tanto, el algoritmo devuelve $v_i^1 = x_i \leq \sqcup X$.
  - En caso de que no se cumpla la condición se calcula la altura de $w = \sqcup U_i^1$ $(h(w))$. Si $h(w) > k_i$ entonces se devuelve w, es decir $v_i^1 = w$. Además, $U_i^1 \subseteq X$ entonces $v_i^1 \leq \sqcup X$.
  - Si $h(w) \leq k_i$ entonces se devuelve $v_i = x_i$. Por lo tanto, $v_i^1 = x_i \leq \sqcup X$.

  Por lo tanto, $v_i^1 \leq \sqcup X$.

- Hipótesis de inducción: sea G^r, un conjunto de procesos en la ronda r que comparten la misma etiqueta. Entonces, $\forall p_i \in G^r$, $v_i^r \leq \sqcup X$ al final de cada ronda r.

- Por demostrar que la invariante se mantiene al final de la ronda $r + 1$.

  Sea p_i un proceso correcto en la ronda $r + 1$ tal que $v_i^r \leq \sqcup X$.

  De la línea 5, llama al procedimiento clasificador-síncrono con sus valores v_i^r y k_i^r.

  Una vez ejecutadas las líneas 1 y 2 del procedimiento, se tienen tres casos:

  - Si v_i^r es comparable con todo valor $u \in U_i^{r+1}$, entonces se devuelve $v_i^{r+1} = v_i^r \leq \sqcup X$.
  - Si $p_i \in M_{G^r}$: es decir si $h(w) > k_i^{r+1}$ $(w = \sqcup U_i^{r+1})$, entonces devuelve $v_i^{r+1} = w$, donde $\forall u \in U_i^{r+1}$ $u \leq \sqcup X$ por lo que $v_i^{r+1} = w \leq \sqcup X$.
  - Si $p_i \in S_{G^r}$: entonces $v_i^{r+1} = v_i^r \leq \sqcup X$.

  Por lo tanto, al final de la ronda r+1, $v_i^{r+1} \leq \sqcup X$.

■

Para la demostración de la propiedad de comparabilidad, la idea de la demostración se basa en la presentada por Xiong Zheng en [14].

Lema 5.1.7. *Sea p_i tal que decide en la ronda r su valor y_i, entonces y_i es comparable con v_j^r de un proceso correcto p_j.*

Demostración. Sea $p_i \in G^r$ que termina en la ronda r. Sea p_j tal que no decide en la ronda r y es correcto. Existen dos casos:

- Sea $p_j \in G^r$: Como p_j es un proceso correcto, entonces de la línea 2, el proceso p_i recibió el valor v_j^{r-1} en la ronda r. De esta manera y como p_i decidió en la ronda r entonces v_j^{r-1} es comparable con $y_i = v_j^r$ (línea 3 del procedimiento).
- Sea $p_j \notin G^r$: tomamos G^t, tal que $p_i, p_j \in G^t$ y al término de la ronda t, $p_i \in M_{G^t}$ y $p_j \in S_{G^t}$ o $p_j \in M_{G^t}$ y $p_i \in S_{G^t}$.
  - Supongamos $p_i \in M_{G^t}$ y $p_j \in S_{G^t}$. Por el lema 5.1.4 inciso 1 y 2, al término de la ronda t se tiene $v_j^t \leq \sqcup\{u_s^t | p_s \in S_{G^t}\} \leq v_i^t$. Además, $v_i^t \leq y_i$ por lo que $\sqcup\{u_s^t | p_s \in S_{G^t}\} \leq y_i$. Por otro lado, $v_j^r \leq \sqcup\{u_s^t | p_s \in S_{G^t}\}$, Así, $v_j^r \leq y_i$.
  - Supongamos $p_j \in M_{G^t}$ y $p_i \in S_{G^t}$. Por el lema 5.1.4 inciso 1 y 2, al término de la ronda t se tiene $v_i^t \leq \sqcup\{u_s^t | p_s \in S_{G^t}\} \leq v_j^t$. Además, $v_j^t \leq v_j^r$ por lo que $\sqcup\{u_s^t | p_s \in S_{G^t}\} \leq v_j^r$. Por otro lado, $y_i = v_i^r \leq \sqcup\{u_s^t | p_s \in S_{G^t}\}$, Así, $y_i \leq v_j^r$.

■

Lema 5.1.8. *El algoritmo 4 satisface la propiedad de comparabilidad.*

Demostración. Sean p_i y p_j dos procesos correctos que terminan en las rondas r_i y r_j respectivamente. Además, sean y_i y y_j los valores que deciden p_i y p_j respectivamente.

Sin pérdida de generalidad, supongamos que $r_i \leq r_j$. Por el lema 5.1.7 se tiene que y_i es comparable con $v_k^{r_i}$ para cualquier proceso correcto p_k. Por otro lado, sea $V^{r_i} = \{v_k^{r_i} | p_k$ es un proceso correcto que no decide en la ronda $r_i\}$. De tal forma que p_j puede tomar a lo más el $join$ de $U_j \subseteq V^{r_i}$. Tenemos dos casos:

- Si $\forall u \in U_j$ se tiene que $u \leq y_i$ entonces $y_j = \sqcup U_j \leq y_i$.
- Si $\exists v \in U_j$ tal que $y_i \leq v$, por un lado tenemos que $v \leq \sqcup U_j$. Por lo tanto, $y_i \leq v \leq \sqcup U_j = y_j$, es decir $y_i \leq \sqcup U_j = y_j$.

Por lo tanto, se tiene que y_i es comparable con y_j.

■

Los lemas 5.1.5, 5.1.6 y 5.1.8 implican el siguiente teorema.

Teorema 5.1.9. *El algoritmo LA_α (4) resuelve el problema de acuerdo de retícula.*

5.1.4. Complejidad computacional

En [14] demuestran el orden de complejidad del algoritmo 4, quedando enunciada en el siguiente teorema.

Teorema 5.1.10 ([14], Teorema 8)**.** *El algoritmo LA_α (4) resuelve el problema de acuerdo de retícula en* $\log(H)$ *+1 rondas, donde ocurren $f < n$ fallas.*

Haciendo uso de los siguiente lemas:

Lema 5.1.11 ([14], Lema 5)**.** *Sea G^r un grupo de procesos al inicio de la ronda r con etiqueta k, entonces:*

- $\forall p_i \in G^r$, $k - \frac{H}{2^r} < h(v_i^{r-1}) \leq k + \frac{H}{2^r}$.
- $h(\sqcup\{v_i^{r-1} : p_i \in G^r\}) \leq k + \frac{H}{2^r}$.

Lema 5.1.12 ([14], Lema 6)**.** *Sean $p_i, p_j \in G^r$ dos procesos correctos, al inicio de la ronda $r = \log(H) + 1$. Entonces, $v_i^{r-1} = v_j^{r-1}$.*

Cota inferior

Sea un conjunto (P) de n procesos sobre una retícula $\mathcal{L}$ y a lo más $f < n$ fallas (sin pérdida de generalidad $p_1, p_2, .., p_f$). Denotaremos a P_C como el conjunto de los $n - f$ procesos correctos.

Se va a tomar una retícula booleana de subconjuntos, la cual queda definida de la siguiente manera: sea $S = \{a_1, a_2, ..., a_n\}$ tal que $|S| = n$ entonces vamos a tomar la retícula $(\mathcal{L})$ sobre $Pow(S)$.

Por otro lado, se determinara cuantos procesos fallan por ronda. En primera instancia requerimos que en la primera ronda fallen $\lfloor \frac{n}{2} \rfloor$ procesos y en cada ronda subsecuente van a fallar $\lceil \frac{n}{2^{r-1}} \rceil - \lceil \frac{n}{2^r} \rceil$ procesos. Por lo tanto:

$$f = \lfloor \frac{n}{2} \rfloor + \sum_{i=2}^{\log n} (\lceil \frac{n}{2^{r-1}} \rceil - \lceil \frac{n}{2^r} \rceil). \tag{5.1}$$

Una vez definida la retícula sobre la que se va a trabajar y el número de procesos que fallan en cada ronda. Procedemos a definir la ejecución y como se va a llevar a cabo:

- El valor de entrada (x_i) para cada proceso p_i es $x_i = a_i$, $i \in \{1, 2, ..., n\}$.
- F_1 es el conjunto de procesos que fallan en la ronda 1 y consta de los primeros $f_1 = \lfloor \frac{n}{2} \rfloor$ procesos.
- Sea A_1 el conjunto $P \setminus F_1$, el conjunto de procesos que no fallan en la ronda 1 con $\lceil \frac{n}{2} \rceil$ procesos.
- Observemos el conjunto de los valores de entrada de los procesos forman una anti-cadena. Además, $X = S$.
- La retícula con la cual se va a trabajar tiene como altura $H = h(\mathcal{L}) = n$.
- El envío de mensajes en la primera ronda se va a llevar a cabo de la siguiente manera (figura 5.1):
  - El proceso $p_i \in F_1$ envía un mensaje únicamente al proceso $q_i \in A_1$, $i \in \{1, 2, .., f_1 - 1\}$.

- A_{f_1} es el conjunto A_1 menos sus primeros $f_1 - 1$ procesos.
- El proceso $p_{f_1} \in F_1$ envía un mensaje únicamente a cada uno de los procesos que pertenecen al conjunto A_{f_1}.
- Los procesos $p \in A_1$ envían su mensaje a todos los procesos.

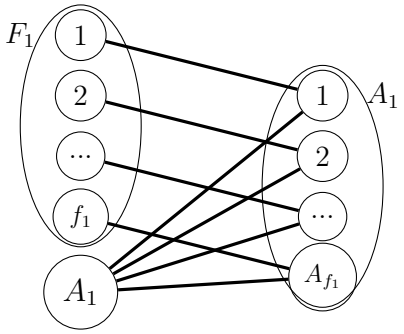

Figura 5.1: Ejecución de la primera ronda.

- En las rondas subsecuentes se continúa de la siguiente manera: Sea $2 \leq r \leq \log(H)$:
- F_r es el conjunto de los primeros $f_r = \lceil \frac{n}{2^{r-1}} \rceil - \lceil \frac{n}{2^r} \rceil$ procesos del conjunto A_{r-1}, los cuales fallan en la ronda r.
- A_r es el conjunto de los procesos que no fallan en la ronda r, además, A_r consta de $\lceil \frac{n}{2^r} \rceil$ procesos.
- El envío de mensajes continúa de la siguiente manera (figura 5.2):
  - El proceso $p_i \in F_r$ envía un mensaje únicamente al proceso $q_i \in A_r$, $i \in \{1, 2, ..., f_r - 1\}$.
  - A_{f_r} es el conjunto A_r menos sus primeros $f_r - 1$ procesos.
  - El proceso $p_{f_r} \in F_r$ envía un mensaje únicamente a cada uno de los procesos que pertenecen al conjunto A_{f_r}.
  - Los procesos $p \in A_r$ envían su mensaje a todos los procesos.

Para denotar que un proceso p_i pertenece a un conjunto A se usara la siguiente notación p_i^A o $p_i \in A$.

Se va a demostrar por inducción que al final de cada ronda $1 \leq r \leq \log(H) - 1$ existe al menos un proceso que no ha decidido.

- Caso base: Cuando $r = 1$, $l = \frac{n}{2}$. De acuerdo con la ejecución, el conjunto F_1 contiene a los primeros $f_1 = \lfloor \frac{n}{2} \rfloor$ procesos de P, donde para cada $p_i \in F_1$ tiene como valor de entrada $v_i = x_i$. Por otro lado, los procesos restantes $p_j \in A_1$ tienen como valor $v_j = x_j$, donde $|A_1| = \lceil \frac{n}{2} \rceil$. Como se mencionó en la ejecución, $H = h(X) = n$. Finalmente, cada proceso $p_i \in A_1$ recibe los valores de los procesos en A_1, además recibe el valor v_i del proceso $q_i \in F_1$, donde $i \in \{1, 2, .., f_1 - 1\}$, $U_{p_i}^1 = \{\{x_i\}, \{x_{\lfloor \frac{n}{2} \rfloor + 1}\}, ..., \{x_n\}\}$.

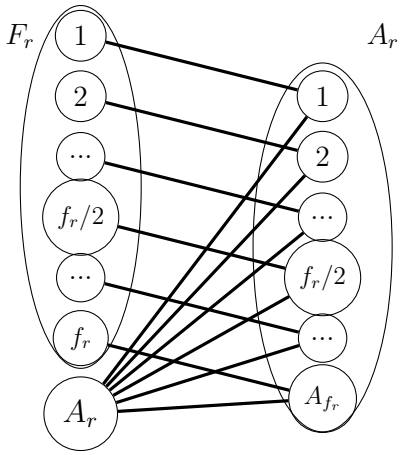

Figura 5.2: Ejecución de la ronda r.

Mientras, los procesos $p_k \in A_{f_1}$ reciben los valores de los procesos en A_1 y el valor del proceso $q_{f_1} \in F_1$, $U^1_{p_k} = \{\{x_{\lfloor \frac{n}{2} \rfloor}\}, \{x_{\lfloor \frac{n}{2} \rfloor + 1}\}, ..., \{x_n\}\}$.

Por lo tanto, cada proceso $p_i \in A_1$ no decide y aplica la operación *join* sobre los valores recibidos, además $h(\sqcup U^1_i) = \lceil \frac{n}{2} \rceil + 1 > l$. Entonces, cada proceso es clasificado como *master* y tiene como valor al final de la ronda 1 $v^1_i = \sqcup U^1_i$, y $l = \frac{3n}{4}$. Se observa que el conjunto de los valores de los procesos en A_1 al final de la ronda 1 forman una anticadena, es decir, $X_1 = \{\{x_1, x_{\lfloor \frac{n}{2} \rfloor + 1}, ..., x_n\}, \{x_2, x_{\lfloor \frac{n}{2} \rfloor + 1}, ..., x_n\}, ..., \{x_{\lfloor \frac{n}{2} \rfloor}, x_{\lfloor \frac{n}{2} \rfloor + 1}, ..., x_n\}\}$ es una anticadena.

- Hipótesis de inducción: Sea $1 \leq r < \log(H) - 1$. Al final de cada ronda r, existe al menos un proceso que no ha decidido. Además, al final de la ronda r el conjunto de los valores de los procesos que pertenecen al conjunto A_r forman una anticadena, es decir, $X_r = \{\{x_1, x_{f_r+1}, ..., x_n\}, \{x_2, x_{f_r+1}, ..., x_n\}, ..., \{x_{f_r}, x_{f_r+1}, ..., x_n\}\}$ es una anticadena, donde $|X_r| = f_r$, y los primeros $f_r - 1$ procesos del conjunto A_r tienen como valor al final de la ronda r, $v^r_i = \{x_i, x_{f_r+1}, ..., x_n\}$. Por otro lado, cada proceso $p_i \in A_{f_r}$ tiene como valor al final de la ronda r, $v^r_i = \{x_{f_r}, x_{f_r+1}, ..., x_n\}$. Finalmente, $h(v^r_i) = 1 + \lceil \frac{n}{2} \rceil + \sum_{k=2}^{r} (\lceil \frac{n}{2^{k-1}} \rceil - \lceil \frac{n}{2^k} \rceil) > l$ y cada proceso es clasificado como *master*, $\forall p_i \in A_r$. Teniendo al final de la ronda r, $l = \sum_{k=1}^{r+1} \frac{n}{2^k}$.

- Por demostrar que la hipótesis se mantiene al final de la ronda $r + 1$.

  Por hipótesis, al inicio de la ronda $r + 1$, $l = \sum_{k=1}^{r+1} \frac{n}{2^k}$. Cada proceso $p_i \in A_r \setminus A_{f_r}$ tiene como valor al inicio de la ronda $r + 1$, $v^r_i = \{x_i, x_{f_r+1}, ..., x_n\}$. Y cada proceso $p_j \in A_{f_r}$ tiene como valor al inicio de la ronda $r + 1$, $v^r_j = \{x_{f_r}, x_{f_r+1}, ..., x_n\}$. De acuerdo en la ejecución, F_{r+1} es el conjunto de los primeros $f_{r+1} = \lceil \frac{n}{2^r} \rceil - \lceil \frac{n}{2^{r+1}} \rceil$ procesos del conjunto A_r. Mientras, A_{r+1} es el conjunto de los procesos que no fallan en la ronda y consta de los $\lceil \frac{n}{2^{r+1}} \rceil$ procesos restantes.

  Finalmente, cada proceso $p_i \in A_{r+1}$ recibe los valores de los procesos en A_{r+1}, además, recibe el valor v_i del proceso $q_i \in F_{r+1}$, para $i \in \{1, 2, ..., f_{r+1} - 1\}$, $U^{r+1}_{p_i} = \{\{x_i, x_{f_r+1}, ..., x_n\}, \{x_{f_{r+1}+1}, x_{f_r+1}, ..., x_n\}, ..., \{x_{f_r}, x_{f_r+1}, ...x_n\}\}$. Mientras, los procesos $p_k \in A_{f_{r+1}}$ reciben los valores de los procesos en A_{r+1} y el valor de $q_{f_{r+1}} \in F_{r+1}$,

$U_{p_i}^{r+1} = \{\{x_{f_{r+1}}, x_{f_r+1}, ..., x_n\}, \{x_{f_{r+1}+1}, x_{f_r+1}, ..., x_n\}, ..., \{x_{f_r}, x_{f_r+1}, ...x_n\}\}$.

Por lo tanto, cada proceso $p_i \in A_{r+1}$ no decide y aplica la operación *join* sobre los valores recibidos. De esta forma, como cada proceso $p_i \in A_{r+1}$ recibió $(\lceil \frac{n}{2^r} \rceil - \lceil \frac{n}{2^{r+1}} \rceil)$ de valores, entonces la altura de su valor v_i^{r+1} es $h(v_i^{r+1}) = 1 + \lceil \frac{n}{2} \rceil + \sum_{k=2}^{r} (\lceil \frac{n}{2^{k-1}} \rceil - \lceil \frac{n}{2^k} \rceil) + (\lceil \frac{n}{2^r} \rceil - \lceil \frac{n}{2^{r+1}} \rceil) = 1 + \lceil \frac{n}{2} \rceil + \sum_{k=2}^{r+1} (\lceil \frac{n}{2^{k-1}} \rceil - \lceil \frac{n}{2^k} \rceil) = 1 + 2\lceil \frac{n}{2} \rceil - \lceil \frac{n}{2^{r+1}} \rceil > l$. Por lo tanto, cada $p_i \in A_{r+1}$ no decide y es clasificado como *master*. Teniendo al final de la ronda $r+1$, $l = \sum_{k=1}^{r+2} \frac{n}{2^k}$. Se observa que el conjunto de los valores de los procesos en A_{r+1} al final de la ronda 1 forman una anticadena, es decir $X_{r+1} = \{\{x_1, x_{f_{r+1}+1}, ..., x_n\}, ..., \{x_{f_{r+1}}, x_{f_{r+1}+1}, ..., x_n\}\}$.

Por lo tanto, al final de cada ronda $1 \leq r \leq \log(H) - 1$ los procesos que pertenecen al conjunto A_r no deciden, es decir, el conjunto de los valores de los procesos que pertenecen al conjunto A_r forman una anticadena:

$X_r = \{\{x_1, x_{f_r+1}, ..., x_n\}, \{x_2, x_{f_r+1}, ..., x_n\}, ..., \{x_{f_r}, x_{f_r+1}, ..., x_n\}\}$, donde $|X_r| = f_r$.

En la ronda $\log(H) - 1$, el conjunto $X_{log(H)-1}$ puede ser de dos maneras: $|X_{log(H)-1}| = 2$ o $|X_{log(H)-1}| = 3$. Por lo tanto, al inicio de la ronda $\log(H)$ (figura 5.3) $l = \sum_{k=1}^{log(H)} \frac{n}{2^k}$ y se tienen dos conjuntos: el conjunto $F_{\log(H)}$ formado por un proceso y el conjunto $A_{\log(H)}$, donde los procesos de ambos conjuntos envían sus mensajes a los procesos del conjunto $A_{\log(H)}$. Entonces, cada proceso $p_i \in A_{\log(H)}$ no decide y aplica la operación *join* sobre los valores recibidos, con lo cual $h(v_i^{log(H)}) = n > n - 1$. Por lo tanto, cada $p_i \in A_{log(H)}$ no decide y es clasificado como master, y $v_i^{\log(H)} = \{x_1, ..., x_n\}$. Entonces, cada proceso $p_i \in A_{\log(H)}$ es comparable al final de la ronda $\log(H)$.

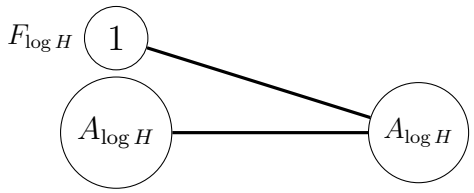

Figura 5.3: Ejecución de la ronda $\log(H)$.

De este argumento se puede concluir el siguiente teorema respecto al algoritmo LA_α.

Teorema 5.1.13. *Existe una ejecución tal que el algoritmo LA_α requiere* $\log(H)$ *rondas para resolver el LAP.*

Capítulo 6

Conclusiones y trabajo futuro

En el trabajo se estudió el *problema de acuerdo de retícula* para un sistema síncrono y donde pueden ocurrir $f < n$ fallas de paro. Para poder abordar el problema se estudiaron los conceptos de retícula, así como un par de resultados importantes sobre el mismo. En este capítulo se van a abordar las conclusiones y el trabajo futuro.

6.1. Conclusiones

La estructura de la tesis sirvió para destacar el mejoramiento en orden de complejidad de los algoritmos mostrados. Primero se decidió estudiar el algoritmo LA_R, el cual tiene un orden de complejidad $O(f)$ en el número de rondas. Entre las observaciones que se encontraron fue que al final de cada ronda el valor de salida v_i^r es no decreciente y que se encuentra acotado por x_i y $\sqcup X$. Debido a que no se revisaban como eran los mensajes recibidos con respecto al valor del proceso al inicio de cada ronda, se tuvo un algoritmo de orden lineal.

Con las observaciones finales se decidió abordar LA_M propuesto por Mavronicolas el cual tiene un orden de complejidad $O(\sqrt{f})$ respecto al número de rondas. Se demostró que la cota podía mejorarse dando como resultado el siguiente teorema:

Teorema 6.1.1. *El algoritmo LA_M (2) resuelve el problema de acuerdo de retícula en* $\text{mín}\{1 + h(\mathcal{J}(X)), \lceil (1 + \sqrt{8f+1})/2 \rceil\}$ *rondas, para cada ejecución en la que los procesos $p_1, p_2, ..., p_n$ inician con valores de entrada $x_1, x_2, ..., x_n$ y f procesos fallan.*

Se llevo a cabo una construcción para demostrar que la cota era justa. Al igual que LA_R se observó que al final de cada ronda de LA_M el valor de salida v_i^r es no decreciente. Una de las mejoras fue la terminación temprana y esto hizo que la cota mejorara de un orden lineal a uno de raíz cuadrada.

Finalmente se estudió el algoritmo LA_α propuesto por Xiong Zheng y equipo. El algoritmo LA_α consiste en ir partiendo un grupo G^r en dos (M_{G^r} y S_{G^r}) a partir del criterio de la altura de la retícula, tal que cada proceso comparte la misma etiqueta. De esta forma se crea un árbol binario cuya altura es $\log(H)$ y por lo cual se requiere $\log(H) + 1$ rondas para encontrar la solución. Continuando bajo la misma metodología de los dos algoritmos anteriores, se demostró que la cota es justa mediante la construcción de una ejecución que requiere $\log(H)$ rondas.

6.2. Trabajo futuro

El estudio de los tres algoritmos descritos en la tesis pretenden servir de base en la investigación de la existencia de mejoras en los algoritmos para el LAP. Para hacer este estudio se está abordando el problema en una *retícula completa*, donde se tiene la existencia tanto del *meet* como del *join*. Estamos analizando el algoritmo LA_τ, el cuál engloba a los algoritmos estudiados en este trabajo. Como nota, *prop* de la línea 15 será explicado más adelante.

Algoritmo 5: $LA_\tau(x_i)$ para un proceso p_i

Datos: x_i = valor de entrada
Resultado: y_i = valor de salida

```
1  v_i = x_i
2  bandera = true
3  nodo = ∅
4  r = 1
5  mientras bandera hacer
6  |  enviar a todos (v_i, nodo)
7  |  U = {v_j | recive v_j de p_j sí nodo = nodo_j}
8  |  si v_i ≤ u o v_i ≥ u, ∀u ∈ U entonces
9  |  |  bandera = false
10 |  |  devolver v_i
11 |  fin
12 |  si r = 1 entonces
13 |  |  v_i = ⊔{u|u ∈ U}
14 |  en otro caso
15 |  |  si prop entonces
16 |  |  |  v_i = ⊔{u|u ∈ U}
17 |  |  |  nodo = nodo + 0
18 |  |  en otro caso
19 |  |  |  v_i = ⊓{u|u ∈ U}
20 |  |  |  nodo = nodo + 1
21 |  |  fin
22 |  fin
23 |  r = r + 1
24 fin
25 y_i = v_i
```

Como resultado principal sobre este algoritmo se tiene que es correcto. Por otro lado, actualmente nos encontramos estudiando la complejidad, esto se está haciendo sobre la proposición *prop* (línea 15).

1. *prop* depende de la altura de la retícula.

2. *prop* depende del número de fallas.

Para el punto uno, dado que este algoritmo está rescatando la condición del algoritmo LA_α, se espera que se tenga un orden de complejidad $O(\log H)$. Mientras que en el punto dos, se espera encontrar un orden de complejidad $O(\log f)$.

Bibliografía

[1] H. Attiya, M. Herlihy, and O. Rachman. Atomic Snapshots Using Lattice Agreement. *Distributed Computing*, 8(2):121–132, 1995.

[2] D. Dolev and H. R. Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing*, (12(4)):656–666, 1983.

[3] J. M. Faleiro, S. Rajamani, K. Rajan, G. Ramalingam, and K. Vaswani. Generalized Lattice Agreement. *In Proceedings of the 2012 ACM symposium on Principles of distributed computing*, pages 125–134, 2012.

[4] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of Distributed Consensus With One Faulty Process. *Journal of the ACM*, (32(2)):374–386, 1985.

[5] V. K. Garg. *Introduction to Lattice Theory With Computer Science Applications*. Wiley, 2015.

[6] M. Herlihy, D. Kozlov, and S. Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Elsevier-Morgan Kaufmann, 2013.

[7] L. Lamport. Paxos Made Simple. *ACM Sigact News*, (32(4)):18–25, 2001.

[8] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, (4(3)):382–401, 1982.

[9] G. A. D. Luna, E. Anceaume, S. Bonomi, and L. Querzoni. Synchronous Byzantine Lattice Agreement in O(log(f)) Rounds. *40th IEEE International Conference on Distributed Computing Systems (ICDCS 2020)*, pages 146–156, 2020.

[10] G. A. D. Luna, E. Anceaume, and L. Querzoni. Byzantine Generalized Lattice Agreement. *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 674–683, 2020.

[11] M. Mavronicolas. A Bound on the Rounds to Reach Lattice Agreement. *http://www.cs.ucy.ac.cy/ mavronic/pdf/lattice.pdf*, 2005.

[12] X. Zheng and V. Garg. Byzantine Lattice Agreement in Synchronous Message Passing Systems. *34th International Symposium on Distributed Computing (DISC 2020)*, 179:32:1–32:16, 2020.

[13] X. Zheng and V. K. Garg. Byzantine Lattice Agreement in Asynchronous Systems. *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, 184:4:1–4:16, 2020.

[14] X. Zheng, C. Hu, and V. K. Garg. Lattice Agreement in Message Passing Systems. *32nd International Symposium on Distributed Computing (DISC 2018)*, 121:41:1–41:17, 2018.