Simulación de modelos

Erick Hernandez Navarrete

16 de julio de 2020

Índice general

Capítulo 1

Decidibilidad en los modelos

1.1. Introducción

En este documento expondremos las nociones de decibilidad en el mundo de maquinas de Turing y en el mundo distribuido, así como formalizaremos la noción de simulación de modelos computacionales, demostrando que el modelo A de la maquina de Turing es equivalente en poder al modelo distribuido B, en particular al modelo **LOCAL**

1.2. Decidibilidad en el modelo de maquinas de Turing

1.2.1. Elementos del modelo

Primero enunciaremos los elementos del modelo de maquina de Turing,

Definition 1. *Una maquina de Turing is una 7-tupla,* $(Q, \Sigma, \Lambda, \delta, q_0, q_{accept}, q_{reject})$ *donde* Q, Σ, Λ *son conjuntos finitos.*

1. Q *es el conjunto de estados,*
2. Σ *es el alfabeto de entrada que no contiene el simbolo blanco* $\sqcup$
3. Λ *es el alfabeto de la cinta, donde* $\sqcup \in \Lambda$ *y ademas* $\Sigma \subseteq \Lambda$,
4. $\delta : Q \times \Lambda \to Q \times \Lambda \times \{L, R, S\}$ *es la función de transición,*
5. q_0 *es el estado de iniciación,*
6. q_{accept} *es el estado de aceptacion*
7. q_{reject} *es el estado de rechazo, donde* $q_{accept} \neq q_{reject}$,

Esta definición, esta encapsulando los elementos del modelo, así que una vez que tenemos los elementos, podremos definir la semántica de que un problema es soluble en este clasico modelo, definiendo la noción de decidible.

1.2.2. Decidibilidad

Podemos observar que en este modelo, las estructuras de datos que están consumiendo son cadenas a priori finitas, entonces una vez que le damos como entrada a una máquina de Turing una cadena w, podemos decir que esa cadena es aceptada o rechazada a priori. Entonces definimos lo siguiente:

Definition 2. *Sean x, y dos cadenas y q_j con $j \in \mathbb{N}$ un estado en el conjunto de estados Q, estos elementos del modelo Turing conformaran una semántica: xq_jy, donde la posición del estado representa la localidad del cabezal en ese momento del computo. Para esa triada x, q_j, y definirá, la noción de configuración, que la denotaremos de la siguiente manera C_j.*

Una vez que tenemos la definición formal de **configuración**, nos movemos a conectar esta idea con la noción de computo.

Definition 3. *Sean C_1, C_2 configuraciones:*
vamos a decir que C_1 produce a C_2 legalmente en un solo paso si, semánticamente pasá lo siguiente:
Sea $a, b, c \in \Lambda$, además tomamos también u, v cadenas y tenemos el siguiente contexto:
uaq_ibv produce uq_jacv si $\delta(q_i, b) = (q_j, c, L)$, donde R, L, S para denotar la semántica del movimiento del cabezal, derecha, izquierda o en su defecto no moverse.
En el caso de un movimiento a la derecha es: uaq_ibv produce $uacq_jv$ si $\delta(q_i, b) = (q_j, c, R)$.

La definición anterior, formalizara la noción de la secuencias de configuraciones, que en esencia esta formalizando la noción de computo.

Definition 4. *Decimos que tenemos una configuración inicial $C0$, para M, una máquina de Turing y w una cadena, que se escribirá de la siguiente manera: q_0w. El cual indica que la máquina esta en su estado inicial q_0, con su cabezal en la última localidad del lado izquierdo.*

Definition 5. *Tenemos el siguiente contexto en términos de **configuración**:*

1. *En una configuración de **aceptación**, tenemos el estado q_{accept}*
2. *En una configuracioń de **rechazo**, tenemos el estado q_{reject}*

Remark. *Podemos clasificar las configuraciones en dos grupos:*

- *Configuraciones de paro.*
- *Y las que no son de paro.*

Dentro de ese grupo, las configuraciones de paro tienen la propiedad de no producir otra configuración C_k, en otras palabras son configuraciones finales.
*Las configuraciones que son de dicha naturelaza, son especificamente las configuraciones de **aceptacioń** y la configuración de **rechazo**.*
Mas aún, podriamos complicar la función δ tomando $Q^* = Q - \{q_{accept}, q_{reject}\}$ y redefinimos a delta como:

$$\delta : Q^* \times \Lambda \to Q \times \Lambda \times \{L, R, S\} \tag{1.1}$$

Definition 6. *Decimos que una máquina de Turing* ***acepta*** *una entrada w si existe una secuencia de configuraciones $C_0, \ldots, C_k$ tal que:*

1. *C_0 es la configuración inicial*
2. *Cada C_i produce C_{i+1} y*
3. *C_k es la configuración de* ***aceptación****.*

Definition 7. *Sea M una máquina de Turing.*
La colección de cadenas tales que M las acepta, es un ***lenguaje*** *que acepta ó que es reconocible por M, y lo denotamos de la siguiente manera: $L(M)$.*

Mas aún, podemos definir la siguiente noción:

Definition 8. *Llamamos a un lenguaje* ***Turing-Reconocible*** *si existe una máquina de Turing que lo reconozca.*

Remark. *Cuando iniciamos una máquina de Turing con entrada w, pueden pasar tres escenarios:*

- *acepta,*
- *rechaza,*
- *nunca se detiene.*

En este caso, tomaremos solamente máquinas de Turing que se detienen para todas sus entradas, por lo tanto dichas máquinas siempre se dentendrán en algún momento de la ejecución de M, para $\forall w$ entrada. Formalizaremos lo anterior con una definición.

Definition 9. *Sea M máquina de Turing tal que $\forall w$ entrada, esta se detiene en algún momento de la ejecución, entonces decimos que dicha máquina de Turing tiene el atributo de ser* ***decidible*** *Es decir, siempre entrán en su estado de* ***aceptación*** *o en su estado de* ***rechazo****, en algún momento de la ejecución.*

Y por el lado de la noción del lenguaje definimos lo siguiente:

Definition 10. *Decimos que un lenguaje es* ***decidible*** *si existe un máquina de Turing que lo* ***decide****.*

Ejemplo de lenguajes decidibles

A continuación describiremos un ejemplo de una máquina de Turing, describiendo cáda una de sus partes que la definen, así como el principio de funcionamiento de la misma.

Example 1. *Tomamos el siguiente lenguaje:*

$$A = \{0^{2n} : n >= 0\} \tag{1.2}$$

Es el lenguaje que consta de cadenas de 0's tal que su longitud es una potencia de 2.

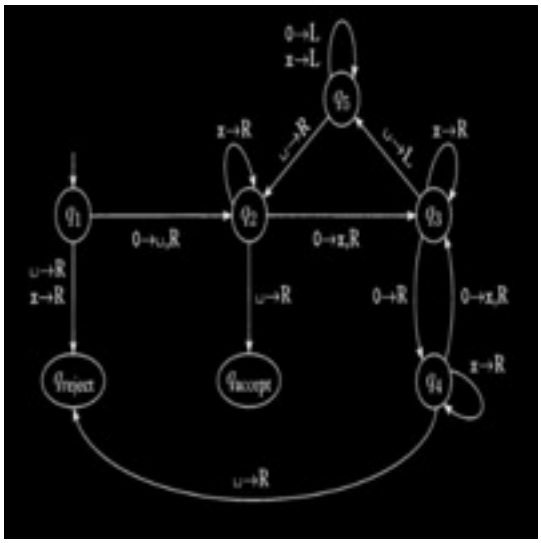
Entonces afirmamos que M_2 es un lenguaje decidible. Entonces podemos hacer las instrucciones (alto nivel) para M_2 como sigue:

1. recorrer de izquierda a derecha a través de la cinta, tachando uno de cada dos ceros.
2. Si en la etapa 1, la cinta contenia un solo 0, entonces **acepta**.
3. Si en la etapa 1, la cinta contenía mas de un 0 y ademas contenía un numero impar, mayor a 1, **rechaza**.
4. Regresa el cabezal al lado izquierdo al final de la cinta.
5. Ve a la etapa 1.

Una vez que tenemos la descripción de la máquina de Turing a un alto nivel, podemos hacer un puente a un bajo nivel, que es formalmente describir la máquina con sus elementos que la definen para una entrada w, el cual nuestro puente será en describir la función δ vía un diagrama de estados.

En la etapa 1 de M_2, corta el número de ceros a la mitad. Mientras la máquina barre tráves de la cinta en la etapa 1, lleva la cuenta de 0's vistos si es par o impar.
Si el número es impar mayor a 1, entonces el número original de 0's en la entrada no puede ser una potencia de 2. Entonces la máquina **rechaza** en esa instancia. Como sea, si el número de 0's visto es 1, entonces el número original tiené que ser una potencia de 2, entonces en ese caso la máquina **acepta**. Sea $M_2 = (Q, \Sigma, \Lambda, \delta, q_1, q_{accept}, q_{reject})$:

1. $Q = \{q_1, q_2, \ldots, q_5, q_{accept}, q_{reject}$
2. $\Sigma = \{0\}$ and
3. $\Lambda = \{0, x, \sqcup\}$.
4. Describirimos a δ con un diagrama de estado.
5. Los estados de inicio, aceptación y rechazo son $q_1, q_{accept}, q_{reject}$ respectivamente.



En este diagrama, la etiqueta $0 \to \sqcup, R$ que aparecé en la transición del estado q_1 al estado q_2, tiene la semántica de la definición de la función δ, que en otras palabras se describe como si:
en el estado q_1 con el cabezal leyendo 0, la máquina va al estado q_2, escribe $\sqcup$ y mueve el cabezal a la derecha.
Lo cúal en términos formales significa que: $\delta(q_1, 0) = (q_2, \sqcup, R)$. Otra escritura que usaremos para hacer la notación en este ejemplo mas compacta, es denotar: $0 \leftarrow R$, que la semántica es que se hace la transición del estado q_3 al estado q_4, que es la etiqueta que se aprecia en el diagrama de estados, y ademas significa que la máquina se mueve a la derecha en el momento en el que lee un 0 en el estado q_3, pero que no altera la cinta(hace una operación escritura), entonces $\delta(q_3), 0) = (q_4, 0, R)$ en términos del mapeo que se esta generando.
Esta máquina inicia escribiendo un simbolo blanco sobre el último 0 de lado izquierdo. En particular esto sirve para delimitar el final de la cinta con el simbolo blanco $\sqcup$, en esta particular máquina de Turing.
Ahora vamos a hacer una ejecución de esta maquina, M_2 con la entrada $w = 0000$, en el cual escribiremos la función transición δ con la semántica de las configuraciones.

La siguiente secuencia es la ejecución de M_2 con la entrada en particular $w = 0000$, Se lee hacía abajo de las columnas de izquierda a derecha.

$q_1 0000$	$\sqcup q_5 x 0 x \sqcup$	$\sqcup x q_5 x x \sqcup$
$\sqcup q_2 000$	$q_5 \sqcup x 0 x \sqcup$	$\sqcup q_5 x x x \sqcup$
$\sqcup x q_3 00$	$\sqcup q_2 x 0 x \sqcup$	$q_5 \sqcup x x x \sqcup$
$\sqcup x 0 q_4 0$	$\sqcup x q_2 0 x \sqcup$	$\sqcup q_2 x x x \sqcup$
$\sqcup x 0 x q_3 \sqcup$	$\sqcup x x q_3 x \sqcup$	$\sqcup x q_2 x x \sqcup$
$\sqcup x 0 q_5 x \sqcup$	$\sqcup x x x q_3 \sqcup$	$\sqcup x x q_2 x \sqcup$
$\sqcup x q_5 0 x \sqcup$	$\sqcup x x q_5 x \sqcup$	$\sqcup x x x q_2 \sqcup$
		$\sqcup x x x \sqcup q_{accept}$

1.3. Decidibilidad en el modelo de computo distribuido LOCAL

Ahora lo que haremos es definir el modelo de computo distribuido de manera formal, y tomar un modelo en particular en la que enfocaremos nuestro estudio, para posteriormente estudiar la noción de decibilidad en dicho modelo.

1.3.1. Presentación del modelo de computo distribuido

Presentación del protocolo de comunicación

Este modelo tendrá una capa de abstracción de comunicación, asi como su respectiva capa de computación de la siguiente manera.

Capa de comunicación

El modelo de comunicación consiste en una red de comunicación 1-1 que será descrita en términos formales por una gráfica conexa, no dirigida $G = (V, E)$, donde los vertices $V = \{v_1, \dots v_n\}$, operando entre ellos. Inicialmente, consideraremos identificadores unicos asignados a los procesos

de la gráfica G. Más concretamente consideraremos a estos identificadores de un conjunto ordenado de enteros: de la siguiente manera:

$$S = \{s_1, \ldots, s_n\} \; donde: \; s_i < s_{i+1} \; \forall i \geq 1 \tag{1.3}$$

Entonces con esta notación, una ID-asignación es un mapeo: $ID : V \to S$, entonces nos referiremos a su identificador como: $ID(v)$.
Entonces la comunicación se llevara acabo de la siguiente manera:
Cada vertice tendra asociado el numero de puertos, como el $deg_G(v)$, entonces en este sentido, el conjunto de aristas adyacentes al vertice contiene exactamente $deg_G(v)$, donde cada arista esta conectado en un puerto de v.
Podemos denotar que a cada arista (u, v) le corresponde la pareja $((u, i), (v, j))$, donde $1 \leq i \leq deg_G(u)$ y $1 \leq j \leq deg_G(v)$, con semántica:
un canal de comunicación conectadose en el puerto i de u con el puerto j de v.
El vertice u envía(ejecuta la operación $send()$) un mensaje a sus vecinos v, cargando el mensaje en un puerto apropiado, digamos i. Este mensaje es recibido($deliver()$) por v, a tráves del puerto j.

Capa de Computación

Una vez que tenemos la capa de comunicación, presentaremos la capa formal del modelo de computo.
El modelo estara governado por un algoritmo Π, que estará compuesto de protocolos $\Pi_1, \ldots, \Pi_n$, donde cada Π_i residirá en su correspondiente v_i.

Remark. *Hasta ahora hemos hablado a secas de los vertices, pero podemos darle una semántica de computo en términos de* ***procesos****, el cual significa que es una entidad de computo, y entonces a cada v_i lo nombraremos como el proceso p_i.*
Con esta convención podemos decir que cada protocolo(algoritmo local) Π_i residirá en su respectivo proceso: p_i.

Remark. *Podemos observar que podemos modelar a cada Π_i como una maquina de estado para $\forall i$ con su correspodiente conjunto de estados estado Q_i conteniendo en particular a su estado inicial q_{0i}, asi como sus estados de aceptación y rechazo: $q_{accept}, q_{reject} \in Q_i$, respectivamente tal que en cualquier momento dado el proceso p_i esta en el estado q_i de Q_i. Mas aún, podemos pensar en cada Π_i como en una máquina de Turing, equipada con operaciones de envío y recepción de* ***mensajes****.*

Por otro lado en la capa de comunicación, tendremos el siguiente esquema:

Definition 11. *Vámos a definir un mensaje MSG como la información local que sera enviada del proceso v al proceso u, por medio del canal $((v, i), (u, j))$, donde el proceso v tiene el atributo de la operacion $send(MSG)$, y el vertice v hace una operación $deliver(MSG)$, para que finalmente el proceso v, hagá una operación $compute()$.*
Podemos decir que el tamaño de la información del mensaje MSG, es $O(\log n)$ bits.

En cualquier momento dado y en cualquier canal de comunicación $e_i = (u, v)$ está en algun estado $\overline{q}_i$ del conjunto de estados $\overline{Q}_i$ el estado $\overline{q}_i$ esta compuesto de dos componentes denotadas

de la siguiente manera: $\overline{q}_{u\leftarrow v}$ y $\overline{q}_{v\leftarrow v}$ una por cada direccion del canal de comunicación. Vamos a denotar como M a la colección de todos los posibles mensajes que se pueden enviar de un proceso a otro en toda ejecucion del algoritmo, cada uno de los dos componentes $\overline{q}_{u\leftarrow v}$ es un elemento de $M \cup \lambda$, $\overline{q}_{u\leftarrow v} = MSG \in M$ significa que ahora el mensaje MSG esta en transicion de u a v, y denotaremos que $\overline{q}_{u\leftarrow v} = \lambda$ para representar una semantica de que el canal actual esta vacio en esa dirección. En el inicio del computo, todos los procesos estan en el estado inicial $q_{0,i}$ $\forall i$ y todos los canales de comunicación estan vacios. Es decir que sintacticamente: $\overline{q}_{i,0} =< \lambda, \lambda >$

Ejecución de un algoritmo en este modelo

La ejecución del algoritmo en este ambiente consistira de **Eventos**, ocurriendo en diversos lugares de la red y afectando a los procesos involucrados. Diremos que un **paso computacional** es una operación como máquina de Turing del proceso p. Los eventos puede ser del tipo:

- Computacional: Representando un paso en un procesador
- Comunicación: Representando la entrega o la recepción de un mensaje.

Donde cada evento de comunicación tiene una semántica de:
$SEND(i, j, MSG)$ o $DELIVER(i, j, MSG)$ para algún mensaje MSG. Entonces a manera de reportorio de eventos tenemos:

1. Evento $COMPUTE(i)$: El proceso v_i ejecuta una operación interna, basado en su estado local, y posiblemente cambie su estado local.
2. Evento $SEND(i, j, MSG)$: El proceso v_i envia de salida un mensaje MSG en algún canal de comunicación link e_l con destino al proceso v_j
3. Evento $DELIVER(i, j, MSG)$: El mensaje MSG originado de un proceso v_i que es enviado por el canal de comunicación e_l es entregado en la entrada del destino v_j

Entonces la computación en un sistema distribuido lo podemos pensar de la siguiente manera:
Como una secuencia de configuraciones, capturando el estado actual de los procesos y los canales de comunicación.
Cada evento cambia de estado para algun procesador v_i, y posiblemente también para un canal de comunicación y eso cambiara la configuracion del sistema. En términos formales lo podemos pensar de la siguiente manera:

Definition 12. *Una **configuración** es una tupla $(q_1, \ldots, q_n, \overline{q}_1, \ldots, \overline{q}_m)$, donde $q_i, \overline{q}_j$ es el estado del procesador p_i y del canal de comunicación e_j respectivamente y la configuración inicial es:*

$$q_{0,1}, \ldots, q_{0,n}, \overline{q}_{0,1}, \ldots \overline{q}_{0,m} \tag{1.4}$$

Remark. *Como estamos pensando de manera intuitiva a cada uno de los procesos como una máquina de Turing, entonces una vez que uno de los procesos entra en alguno de los estados: q_{accept}, q_{reject}, el proceso ya no cambía de estado, pero sus operaciones de envio y recepción de mensajes siguen activos, i.e quedan activos las operaciones de $send(), receive()$.*

Entonces modelaremos la computación del algoritmo como una (posible) infinita secuencia de configuraciones alternadamente con eventos.

Definition 13. *La ejecución de un algoritmo* Π *en una grafica con cierta topología* G*, con una entrada inicial* I *en los procesos es denotado como* $\kappa_{\Pi(G,I)}$*. Formalmente, una* ***ejecución*** *es una secuencia de la forma:*

$$\kappa = (C_0, \rho_1, C_1, \rho_2, C_2, \ldots) \tag{1.5}$$

donde cada C_k *es una configuración y cada* ρ_j *es un evento, y en particular* C_0 *denota la configuración inicial.*

Podemos imponer ciertas restricciones en las ejecuciones del algoritmo, pero por el momento podemos definir formalmente el concepto en terminos de ejecuciónes para algun algoritmo Π. Entonces lo anterior nos permite definir lo siguiente:

Definition 14. *Diremos que un* ***modelo*** *es un subconjunto de esas posibles ejecuciones*

Con la definición anterior, nos basaremos en un modelo en particular con una propiedad en particular, a saber el que tiene una estructura de rondas.

Definition 15. *Diremos que un* ***modelo*** *tiene el atributo de* ***rondas****: Si las ejecuciones tienen estructura de* ***rondas****, es decir:*

1. *Cada proceso p ejecuta* ***send*** *a todos sus vecinos,* ***deliver*** *de todos sus vecinos y finalmente* ***compute****,*
2. *Cada proceso ejecuta su r-ésima ronda si todos los procesos ejecutaron su* $r-1$ *ronda.*

La definición anterior nos permitira definir el siguiente modelo:

Definition 16. *Llamaremos* ***LOCAL*** *al modelo que tenga el atributo de* ***rondas***

Y podemos observar que el punto 2 de la definición de rondas, es la que da el cáracter de síncronia en la ejecución.

Ejemplos

1.3.2. Decidibilidad

Una vez que tenemos los elementos del modelo distribuido, podemos introducir la noción de **decibilidad**, en este modelo de computo formal.

Definition 17. *Sea w una cadena, podemos escribir a la cadena como $w_0, \ldots, w_n$, la cual sera la entrada al algoritmo $\Pi(w)$, donde de manera distribuida, tendremos como inicialización, que cada proceso p tendra como entrada un caracter de la cadena w, digamos $p_j(w_k)$.*
Sea $\kappa_{\Pi(w,G)}$ una ejecución del algoritmo Π con entrada w en la grafica G, entonces diremos que la entrada w es aceptada, si existe una configuración en la ejecución $\kappa_{\Pi(w,G)}$, digamos C_k tal que existe un estado q_a en C_k, de su correspondiente proceso p_a, tal que $q_a = q_{accept}$.
Es decir, que el estado en esa configuracion es exactamente el estado q_{accept}.
En simbolos:

$$\forall \kappa_{\Pi(w,G)},\ \exists C_K\ \mid \exists p_a\ \mid q_{a,k} = q_{accept}. \tag{1.6}$$

Lo anterior, nos permitira definir lo siguiente:

Definition 18. *Al conjunto de cadenas que acepta un algoritmo distribuido Π es el lenguaje de Π, o el lenguaje que decide Π, y lo denotaremos como $L(\Pi)$.*

Capítulo 2

Simulación de modelos Maquina de Turing y LOCAL

2.1. Noción de simulación en modelos

Una vez que tenemos estos dos modelos de computo formal, a nivel logico podemos decir que:

Definition 19. *Decimos que un modelo de computo formal T simula a un modelo de computo formal S si:*

$$\forall x \in L(S) \text{ entonces } x \in L(T) \tag{2.1}$$

Mas aún decimos que son modelos equivalentes (computacionalmente) si:

$$\forall x \in L(T) \iff x \in L(S) \tag{2.2}$$

Entonces enunciaremos nuestro teorema de la siguiente manera:

Theorem 1. *Sea TM una maquina de Turing, entonces existe un Π algoritmo distribuido que simula a TM, con la semántica de **simulación**, con base a la definición anterior.*

Una vez que tenemos enunciado este teorema, nos adentraremos al diseño del algoritmo distribuido, digamos Π, tal que para toda ejecucion $\Theta_{\Pi(w,G)}$ con ambiente el modelo **LOCAL**, con la entrada w, para una grafica G con una cierta topología, es tal que **acepta**.

2.2. Diseño del algoritmo

Remark. *Trivialmente, podemos pensar que al darle la entrada la cadena que es aceptada por una maquina de Turing TM, es consumida tal que cada proceso la tiene enteramente como entrada, i.e $\Pi(w)$ entonces de manera local se da que $p_j(w)$, $\forall v_j$, entonces este proceso en particular es tal que en algún momento de la ejecución(para alguna ejecució $\Theta_{\Pi(w,G)}$), exista una configuración C_k, y en esta exista un estado $q_{r,k}$, tal que $q_{r,k} = q_{accept}$, pues su respectivo proceso p_r es una máquina de Turing, por lo tanto de manera global, $w \in L(\Pi)$, pero podemos observar que la forma de la entrada en el algoritmo Π es de manera no distribuida, pues de manera intuitiva todos los procesos saben toda la información.*

Entonces no es verdaderamente un diseño de un algoritmo distribuido, por lo tanto procederemos a diseñar de manera distribuida el siguiente algoritmo:
Daremos la distribución de la información a manera de contricante de la siguente manera:
Sea $w \in L(TM)$, para una maquina de Turing TM arbitraria, entonces decimos que una rebanada de la cadena $w[i]$ o con notación de indice w_i tiene localidad i.
Esta semántica nos va a permitir definir lo siguiente:

Definition 20. *Sea p_k un proceso de una grafica G, con cierta topología, entonces diremos que tendrá una familia f_k de posibles entradas, formada por caracteres w_t, con t localidad para un algoritmo distribuido Π, con ambientación en modelo* ***LOCAL***

Esto nos da la noción del control externo de las entradas, que por el momento esto nos esta generando una cierta familiaridad del papel a un alto nivel de la visión del contrincante, como podremos observar esta es la contraparte del algoritmo que esta gobernando computacionalmente (o por la capa de computo) del algoritmo. Entonces nos propondremos el siguiente diseño del algoritmo que nos dará a priori la solución del problema que estamos atacando.

Algorithm 1 $Simula\_Algo\_TM(w)$

```
for all round ← 1 do
  v_j(w_i){Código para v_j}
  read(w_i)
  while true do
    call δ(q_j, w_i)
    (q_r, w_r, P) ← δ(q_j, w_i)
  end while
  if q_r == q_accept then
    return q_r
  end if
else
  MSG ←< q_r, w_r >
  send(MSG) to Neighbours(j) {vecinos de j}
end for
```

2.3. Descripción del algoritmo

Observando el pseudocódigo del algoritmo, podemos observar que vamos a hacer la iteracion por rondas, *round*, en virtud del ambiente **local**, entonces hacemos el código para cada proceso v_k, luego hacemo la lectura de la entrada w_i, que es la que es por la inicializacion o despues de una operación $deliver(MSG)$, luego hacemos una iteración, en la cual haremos llamadas de $\delta()$, y actualizaremos la salida de dicho llamado, para repetir este proceso hasta que la localidad de la cadena de salida w_r, sea de localidad no asignada a la familia de caracteres de la cadena, asignada a el proceso actual, el cual es asignado por el lado del contrincate, que es el agente externo del sistema distribuido. Finalmente tomamos la decisión de regresar el estado de q_{accept}, si el estado $q_r == q_{accept}$; en otro caso hacemos una operación $send(MSG)$ a los vecinos del proceso actual v_k. Lo que sigue, es demostrar que este algoritmo es correcto, lo cual se enunciará en el siguiente teorema.

2.4. Demostración del procedimiento $Simula\_Algo\_TM$

Theorem 2. *El algoritmo $Simula\_Algo\_TM$ es correcto.*

Demostración. Sea r una ronda de la ejecución del algoritmo $\Pi = Simula\_Algo\_TM$, al incio de esa ronda se estará iniciando un evento del tipo $COMPUTE(k)$, de nuestro repertorio de eventos para el proceso v_k, por la naturaleza de la distrubución de la información llegara un momento de la iteración en la que se de una estructura de dato $msg \leftarrow < q_r, w_r, P >$, arrojada por el llamado iterativo de δ, ya que la localidad de w_r no esta asignada a v_k, sin perdida de generalidad. Entonces, siguiendo el codigo, observamos que tenemos una lógica para la estructura de dato: si $q_r == q_{accept}$, entonces $w \in L(\Pi)$, y se acabaria la ejecución en dicha ronda. Si no, entonces hacemos la operación $Send(t, MSG)$, donde sin perdida de generalidad t representa el indice de uno de los vecinos del proceso v_k, por otro lado como $w \in L(TM)$ entonces $\exists v_l$ en la ronda $r+1$ tal que al final dicha ronda $\exists q_l$ estado tal que $q_l == q_{accept}$.
$\therefore w \in L(\pi)$, por lo tanto el algoritmo es correcto.

□

Una vez que tenemos la corrección del algoritmo se desprende a manera de corolario la simulación de TM en $LOCAL$.

Corrollary 1. *Sea TM una maquina de Turing, entonces:*

$$\forall w \ \in L(TM) \ \exists \Pi \text{ algoritmo con ambientacin } LOCAL \text{ t.q } w \in L(\Pi) \tag{2.3}$$

Demostración. Sean $w \in L(TM)$ para una maquina de Turing y $\Pi = Simula\_Algo\_TM$, $\Pi(w)$ como dicho algoritmo es correcto por el teorema 2, entonces ya tenemos un algoritmo en $LOCAL$ que hace que $w \in L(w)$ $\forall w \in L(TM)$, que semánticamente se reduce a que Π **simula** a TM, con TM una **máquina de Turing** abstracta. □

Entonces una vez que tenemos un algoritmo que es correcto a nivel semantico, la siguiente pregunta es la complejidad asociada a la ejecución de $\Pi \leftarrow Simula\_Algo\_TM$ tanto espacial,de comunicación asi como temporal.

2.4.1. Complejidad del algoritmo