Introducción a Algoritmos en RMES

Centro de Desarrollo de Gestión Empresarial
1 Oriente 1007 – Viña del Mar – Chile
Fono: (56) (32) 2688987 – Fax: (56) (32) 2684079
empresa@cgessa.com

René González Leiva
Ingeniero de Desarrollo
Septiembre 2012

Resumen

El presente documento tiene como objetivo explicar las bases en que se cimientan los principales algoritmos implementados en RMES, el cual servirá de punto inicial para varios otros documentos que los detallarán. Este documento esta enfocado a consultores y no posee lenguaje técnico informático.

Contenido

Tabla de Ilustraciones

1. Árbol lógico-funcional

RMES utiliza un esquema RBD (Reliability Block Diagram) para representar plantas, flotas, etc.

El objetivo de esto es poder extraer indicadores de confiabilidad de la planta/flota en un intervalo de tiempo o en un conjunto de éstos.

Dado que esta representación se basa en una visión sistémica de la planta/flota, la complejidad de un conjunto de sistemas puede ser reducida agrupando este conjunto de sistemas en un solo sistema. Por ejemplo, agrupar un conjunto de bombas en un solo sistema y llamarlo 'sistema de circulación' reduce todo la información de las bombas en este gran sistema.

La representación RBD es muy útil para varios propósitos. Permite visualizar la planta/flota como un árbol de sistemas o **nodos**. Este árbol se conoce como **Árbol lógico-funcional**.

Un **nodo** es simplemente un componente del árbol de sistemas. RMES definen 2 tipos de nodos: el **bloque** y la **configuración.** El **bloque** es el nivel más bajo del árbol sistémico y se caracteriza porque no puede descomponerse en más sistemas. La **configuración** es una agrupación de sistemas. Estos sistemas pueden ser **bloques** o **configuraciones**. Estas últimas, a su vez, pueden agrupar más sistemas, y así sucesivamente.

En base a lo anterior, en un esquema RBD, la planta será un **nodo raíz**, el cual contiene todos los sistemas o nodos que lo constituyen.

Los **nodos directos** de una **configuración** se conocen como **nodos hijos**, y a su vez, la configuración es el **nodo padre** para los **nodos hijos**. Por definición, todo **nodo** posee un **nodo padre**, exceptuando al **nodo raíz.**

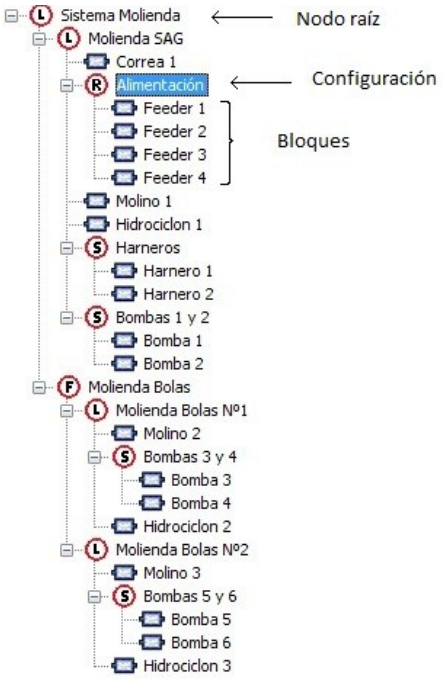

Ilustración 1. Vista en RMES de árbol lógico-funcional

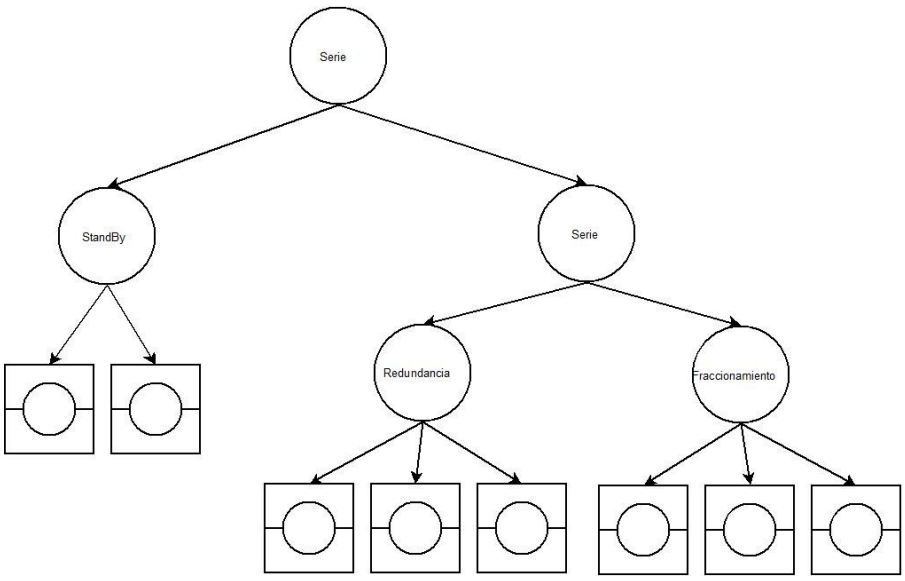

Ilustración 2. Vista como esquema RBD de árbol lógico-funcional

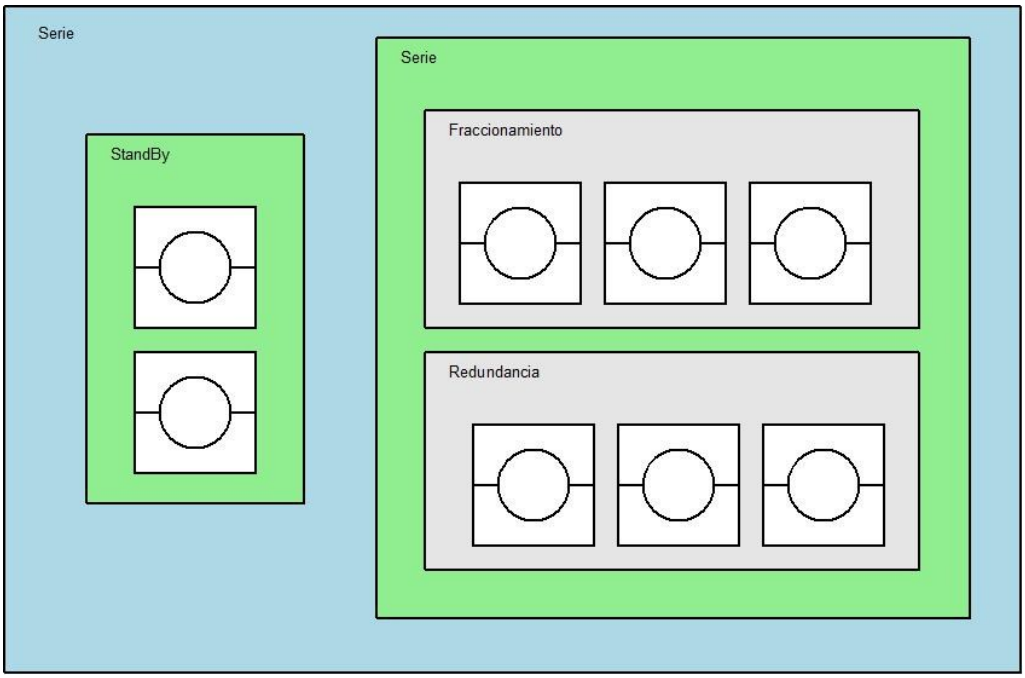

Ilustración 3. Vista como diagrama VISIO de árbol lógico-funcional

2. Tupla

Para efectos de que se puedan realizar estudios de confiabilidad sobre uno o varios **nodos** del **Árbol lógico-funcional** se debe definir el concepto de **tupla**.

Una **tupla** define el intervalo de tiempo en el cual el **nodo** estuvo detenido (cesó su funcionamiento), en cuánto la afectó y la razón de la falla, entre otras propiedades. Comúnmente, esta información es conocida por el consultor como la **detención cargada sobre un nodo**, o sea, la información pura que se ingresa a RMES del no funcionamiento de un nodo. Generalmente, esta información se carga vía repositorios de archivos .csv o también existe la posibilidad de hacerlo manualmente. Por efectos de nomenclatura y sobrecarga de significado de términos, recomendamos referirse a este concepto como **tupla**.

El término de tupla nace desde el mundo de la informática y las matemáticas. Una tupla se refiere a un vector de n-dimensional. El concepto se utiliza simplemente porque, en su generalidad, representa algo que posee varias características.

En definitiva, en RMES, la tupla es el elemento básico de información y contiene todos los datos necesarios para realizar todo tipo de estudios. La tupla posee las siguientes propiedades:

- Fecha de Inicio
- Duración (en horas)
- Costo
- Tipo de Falla
- Detiene el sistema (si o no)
- Síntoma
- Modo de Falla
- Causa
- Orden

La **Fecha de Inicio** y la **Duración** definen el intervalo de tiempo en que el nodo estuvo detenido.

El **Costo** es el valor monetario referente al no funcionamiento del nodo.

Los **Tipos de Falla** son categorizaciones del tipo de no funcionamiento. Por ejemplo, puede deberse a causas internas al nodo (reparación o mantención), a causas externas (detención del proceso por hora de descaso) u otras tantas más. Actualmente, en RMES existen varias (MC, MP, DO, DONP, etc), pero no las enumeraremos todas debido a que cada cierto tiempo se agregan nuevos tipo y este documento pretende ser lo más estático posible.

El campo **Detiene el sistema** dice si la tupla efectivamente representa un no funcionamiento. Generalmente, este campo es **Si**.

El **Síntoma**, **Modo de Falla**, **Causa** son propiedades referentes al **Modo de Falla** de la tupla, concepto que no será explicado en este documento.

El campo **Orden** es muy poco usado, pero se refiere a un identificador de la tupla.

3. Bloques

Como ya se dijo anteriormente, un bloque es la unidad básica del árbol lógico-funcional. Generalmente representa un equipo dentro de una planta, pero más específicamente, representa al objeto más primordial de una planta del cual se puede obtener información.

Por ejemplo, es común que existan equipos de gran envergadura, los cuales se dividen internamente en varios sistemas, estos últimos se dividen en subsistemas y así sucesivamente (difícilmente existirán más de 3 niveles de granularidad en la industria). De esta forma, estos equipos en realidad serán configuraciones. El último bloque finalmente será en donde el usuario puede obtener información directa (correas, rodamientos, etc).

4. Configuraciones

En RMES se definen 5 tipos de **configuraciones lógico-funcionales**. Estos se diferencian en como afecta el no funcionamiento de los nodos hijos sobre el nodo padre. Estas son:

1. **Línea:** La falla de un nodo hijo provoca la detención total del nodo padre.
2. **Paralelo**: Esta configuración falla sólo si todos los nodos hijos lo hacen.
3. **StandBy:** Es una configuración especial que contiene sólo 2 nodos, ni más, ni menos. Si uno falla, el otro funciona inmediatamente, por ende, esta configuración falla cuando ambos nodos lo hacen.
4. **Redundancia Parcial**: Esta configuración define una nomenclatura de tipo *(n:m),* donde *n* es la cantidad total de nodos hijos y *m* es la cantidad mínima de nodos hijos necesarios en funcionamiento para que el nodo padre no falle. Entonces, si fallan más de *(n-m)* nodos hijos, esta configuración falla. Ej: (4:3), se necesitan mínimo 3 nodos en funcionamiento, por ende, si más de (4-3) = 1 nodos hijos fallan, el nodo padre falla.
5. **Fraccionamiento:** Configuración que permite que los nodos hijos funcionen a una capacidad fraccionada de su total, provocando fallas fraccionadas sobre el padre. El nodo padre fallará completamente si todos los nodos hijos fallan.

4.1.Configuraciones en fraccionamiento

Es necesario realizar un apartado para este tipo de configuración dado que posee ciertas particularidades que lo hacen diferente del resto de los **configuraciones lógico funcionales** y que afectan profundamente al algoritmo de propagación.

Las configuraciones en fraccionamiento poseen hijos "impactados", es decir, sus hijos poseen un impacto, que es la tasa de influencia del nodo hijo sobre el nodo padre en fraccionamiento. El impacto es un valor entre 0 y 1 y corrige el impacto de las fallas que el nodo hijo contenga (por ejemplo, si una falla tiene impacto de 100 %, y está presente en un nodo con impacto 30 %, el impacto de la falla se corrige a 30 %, que es simplemente la multiplicación de los 2 impactos).

El impacto sólo tiene sentido para esta configuración. Para el resto de las configuraciones, el impacto no se utiliza, pero para efectos matemáticos y de uniformidad, todo nodo que no tenga un padre directo en fraccionamiento posee un impacto 1 (100%).

4.1.1. Fraccionamiento Simple

La sumatoria de los impactos de los hijos suma exactamente 1, o en términos porcentuales y más usado, 100 %.

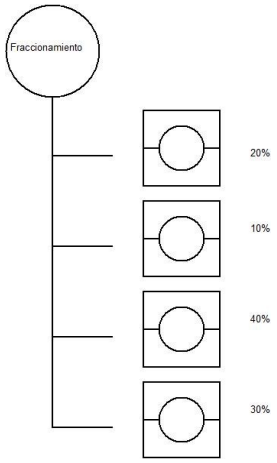

Ilustración 4. Fraccionamiento Simple

4.1.2. Fraccionamiento con Capacidad Ociosa

La sumatoria de los impactos de los hijos puede superar el 100%, pero si falla un nodo, la sumatoria de impacto de los nodos que aún siguen en funcionamiento no debe superar el 100%. Esto se puede chequear fácilmente restando a la sumatoria de impactos total el menor de los impactos.

Se debe tener completamente en claro que ningún nodo, sin importar el tipo de configuración, puede funcionar a más de un 100%. Lo que sucede es que existe una sobrecapacidad que se consume a medida que vayan surgiendo fallas. Cuando esta capacidad se consume completamente, la configuración en fraccionamiento comienza a presentar fallas.

Este razonamiento se puede aplicar a Sistemas Redundantes, sólo que cuando la sobrecapacidad se consume, el sistema falla completamente, y no de forma fraccionada.

La fórmula matemática que rige la configuración de impactos para los hijos de una configuración Fraccionada es:

$$\sum_j I_j - I_i \leq 1 \qquad \forall i = \{1, \dots, n\} \tag{1}$$

Dónde $\sum_j I_j$ es la sumatoria total de impactos, I_i es el impacto del nodo a chequear y n es la cantidad de nodos hijos. Todos los nodos hijos deben cumplir esta regla.

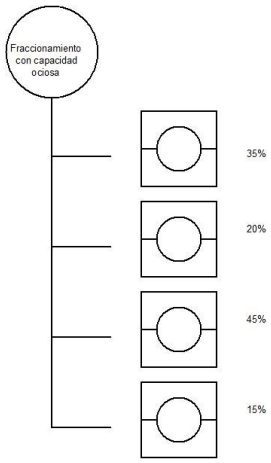

Ilustración 5. Fraccionamiento con Capacidad Ociosa

4.1.3. Fraccionamiento con Redundancia

El fraccionamiento con redundancia es un fraccionamiento con capacidad ociosa, pero que no se rige por la regla anteriormente descrita. La sumatoria de impactos puede superar el 100% y posee una nomenclatura especial (n:m) en donde *n* es la cantidad de nodos hijos y *m* es la cantidad mínima de nodos hijos en funcionamiento para que la configuración completa funcione a una capacidad del 100 %. Además, todos los nodos hijos poseen el mismo impacto, el cual es 1/m.

Una visión más simple, pero incompleta de este tipo de fraccionamiento es que éste se comporta como una configuración en Redundancia Parcial hasta que el mínimo de nodos en funcionamiento no se cumple. Luego, la configuración funciona como un fraccionamiento simple. Este enfoque es incompleto dado que no se puede aplicar si los hijos que posee son sistemas fraccionados, dado que los impactos generalmente son menores, por ende, es posible que la sumatoria de impactos de los *m* nodos hijos no sea suficiente para impactar una detención sobre el nodo padre.

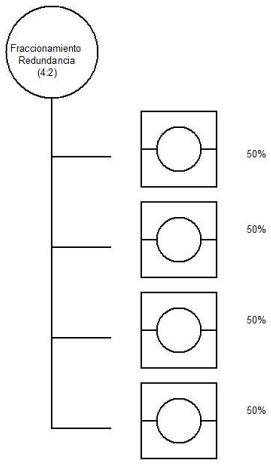

Ilustración 6. Fraccionamiento con Redundancia (4:2)

SÓLO PARA USO INTERNO DE CGS

5. Filtro de intersección

Para que los algoritmos internos del software trabajen correctamente es necesario que las tuplas contenidas en cada nodo cumplan los siguientes requisitos:

1. Deben estar ordenadas por fecha (de menor a mayor)
2. No debe estar traslapadas

Entendiéndose por traslape que 2 detenciones compartan un mismo espacio de tiempo. Esto en la práctica es imposible (?), o al menos es un comportamiento no deseado. En vista de que no se puede confiar en que el usuario se preocupe detenidamente de estos detalles al momento de ingresar información al sistema, dado que es una tarea muy difícil, RMES implementa un algoritmo denominado **Filtro de Intersección** el cual ordena las tuplas ingresadas y resuelve los traslapes en base a ciertos criterios.

Este algoritmo se realiza automáticamente cada vez que se agregan, modifican o eliminan tuplas de un nodo, de forma que en todo momento, dentro de RMES, exista siempre información consistente.

El filtro posee varios pasos y chequeos, por lo que a continuación se explicarán los más relevantes. Al inicio, para agilizar el algoritmo, se chequea la situación más probable de carga de tuplas, que es un ingreso de tuplas no traslapadas. Por ello, se realiza un chequeo "todos con todos" de las tuplas ingresadas buscando si alguna traslapa con otra. Si esto no ocurre, el filtro retorna las mismas tuplas dado que no es necesario procesarlas.

Ahora bien, si lo anterior no fuese el caso, se procede con creación de un conjunto de **IITBlock**, denotado así por su denominación informática. Un **IITBlock** es una agrupación de tuplas intersectadas en donde cada una de ellas esta traslapada con al menos otra tupla. Por definición, un **IITBlock** al menos contiene una tupla y dos **IITBlock** no comparten tuplas traslapadas, o sea, nunca existirá traslape a nivel de **IITBlock**. La Ilustración 7 representa un conjunto de tuplas a ser cargadas sobre un nodo. Como se observa, existen traslapes por lo que el filtro debe ser aplicado.

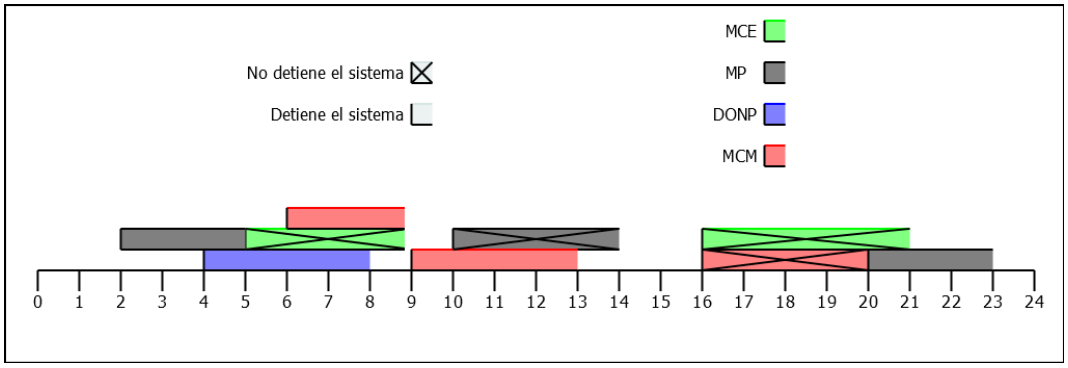

Ilustración 7. Conjunto de tuplas traslapadas en un nodo

La creación de un **IITBlock** simplemente une aquellas que estén traslapadas. Existe además un valor conocido como **Tolerancia al Traslape** (conocido informáticamente como **overlapTolerance**), variable que define la distancia mínima de separación entre dos tuplas para ser consideradas como no traslapadas, o sea, es un TBF restrictivo.

El caso A de la Ilustración 8 presenta dos tuplas no contiguas, pero con una separación menor a la Tolerancia, por lo que el algoritmo de filtrado las considerara traslapadas. En el caso B, la distancia entre ambas tuplas supera la Tolerancia por lo que no se consideran traslapadas.

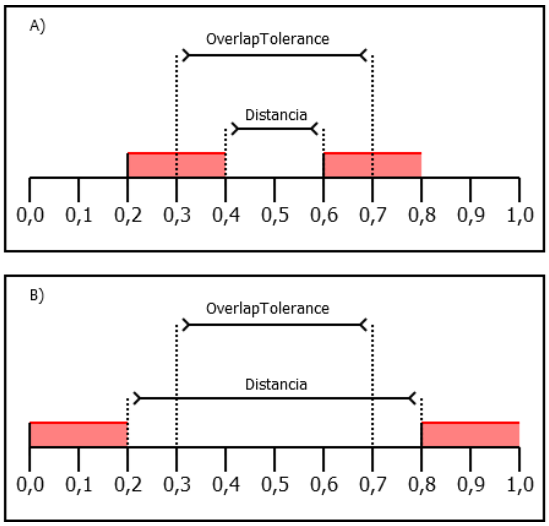

Ilustración 8. Tolerancia al Traslape

Teniendo claro lo anterior, se puede continuar con la creación de las Agrupaciones compactadas de tuplas. La Ilustración 9 representa las agrupaciones creadas. Luego, se continúa con el procesamiento del conjunto de **IITBlock**, uno por uno.

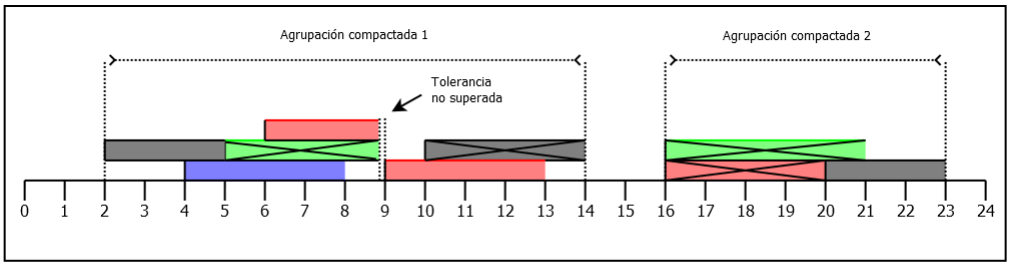

Ilustración 9. Agrupaciones compactadas de fallas.

Cada **IITBlock** pasa por 3 procesos:

1. **Corte**
2. **Jerarquización**
3. **Juntado**

El proceso de **Corte** guillotina todas las tuplas en base a todas las fechas de inicio y fin presentes en una agrupación compactada creando **subagrupaciones** de fallas las cuales comparten la misma fecha de inicio y fin. La fecha inicio original y la duración original de cada tupla cortada no se modifica.

El proceso de **Jerarquizado** implementa los criterios de "subida" de tuplas, es decir, dentro del conjunto de tuplas presentes en una **subagrupación** creada anteriormente, elegir cual de ellas prevalecerá por sobre las otras. Además, la tupla que sea seleccionada "absorbe" las ordenes de trabajo del resto de las tuplas, conteniendo todas aquellas que hayan existido en ese intervalo de tiempo.

Los criterios de jerarquización a seguir dentro de una **subagrupación** son los siguientes:

1. Se elige la tupla que detenga a la planta. En caso de que varias detengan la planta o ninguna la detenga, se continúa con el siguiente criterio.
2. De las que han sido seleccionadas del criterio anterior, se selecciona aquella que tenga la fecha original más anterior. Si existen varias tuplas que cumplan con la condición, se continúa con el siguiente criterio.
3. De las que han sido seleccionadas del criterio anterior, se elige aquella que tenga la duración original mayor. Si existen varias tuplas que cumplan con la condición, se continúa con el siguiente criterio.
4. De las que han sido seleccionadas del criterio anterior, se ordenan por **Jerarquía de Tipo Tuplas** y se elige la primera.

La **Jerarquía de Tipo de Tuplas** es la siguiente:

Tipo	Jerarquía
MCM	1
MCE	2
MCI	3
MP	4
DONP	5
DO	6
ODNP	7
OD	8
AONP	9

Tabla 1. Jerarquía de Tipo de Tuplas

El proceso de **Juntado** une las tuplas contiguas que sean iguales. Dos tuplas son iguales si:

1. Poseen el mismo tipo

2. Tienen el mismo impacto
3. Ambas detienen o no la planta
4. Sus modos de falla son iguales

Si dos tuplas son iguales, sus órdenes de trabajo se unirán sin repetición.

La Ilustración 10 muestra gráficamente el proceso completo.

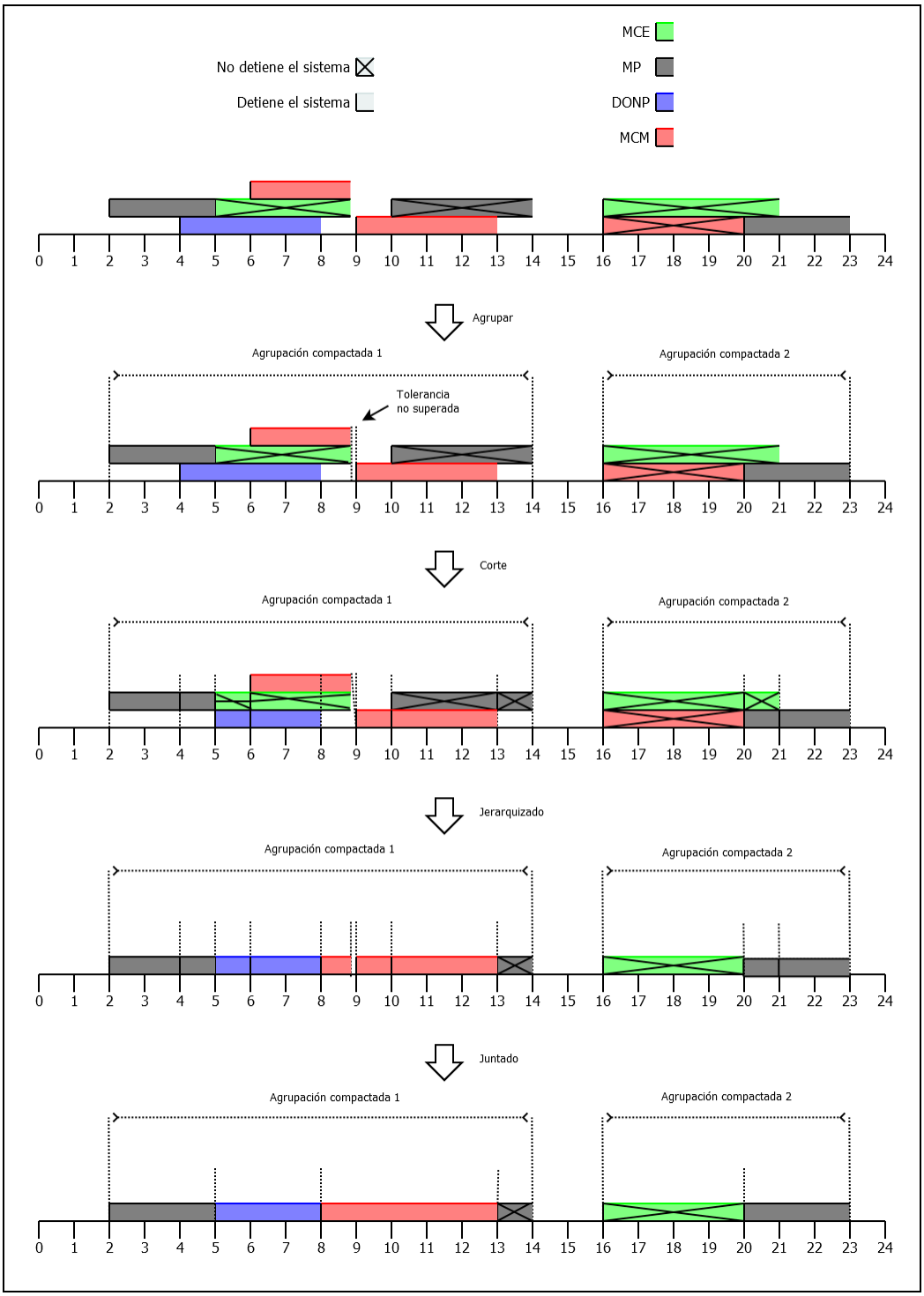

Ilustración 10. Resumen del algoritmo Filtro de Intersección

6. Recorrido Post-Orden

Recapitulando un poco, se necesita comprender claramente que la planta/flota, para el sistema informático es una estructura de tipo árbol. Por ejemplo, la estructura de directorios de cualquier sistema operativo es un árbol. Como toda estructura de datos, ésta existe para almacenar datos que deben ser accesibles en algún momento. En el caso de un árbol, el acceso a los datos se realiza con un proceso llamado **recorrido**.

Imaginemos, que estamos parados sobre algún directorio de nuestro disco duro y queremos ir a un subdirectorio que éste contiene. Comúnmente hacer doble-click sobre el subdirectorio permite ingresar a él. Podremos llegar tan profundo como queramos mientras directorios existan (niveles), pero siempre llegaremos a un punto en que encontraremos documentos (o a veces simplemente el directorio no los tiene). Si llegamos a un punto en que el directorio solo contiene documentos, habremos recorrido una rama específica del árbol, pero aún existen otras.

Supongamos que quisiéramos los nombres de todos los documentos de nuestro disco duro. Para ello, deberíamos recorrer todas las ramas del árbol de directorios, rama por rama. Seria muy poco eficiente si al terminar de recorrer una rama (o sea, encontramos un documento), volviéramos a la raíz de nuestro disco duro y llegáramos al documento contiguo que estaba en el mismo directorio que vimos en la rama anterior, suponiendo que existe otro. Lo mejor es que, al llegar a un directorio, miráramos lo nombres de todos los documentos que el directorio contiene, para luego continuar con los directorios que el directorio actual contiene, y así sucesivamente. Finalmente esta es la idea recursiva que maneja un recorrido de árbol.

En el caso anterior, podríamos anotar los nombres de los documentos al momento de encontrarlos en el recorrido, lo cual sería lo mejor para nuestro problema. Pero existen casos en que es mejor llegar al nivel más bajo del árbol, anotar el nombre de todos los documentos y luego continuar con los documentos de directorio superior, y así sucesivamente. El primer caso se conoce como recorrido Pre-Orden, el segundo, como recorrido **Post-Orden**.

De la anterior se pueden desprender 2 conceptos:

- Visita: que es simplemente llegar al nodo.
- Procesamiento: realizar alguna tarea sobre el nodo.

Según como se realice el recorrido, se llamará Pre-Orden si antes de visitar los hijos procesamos al nodo actual, y **Post-Orden** si primero visitamos los hijos (todos los descendientes) y luego procesamos al nodo actual.

Como se dijo anteriormente, **el recorrido Post-Orden primero procesa los hijos de un nodo y luego al nodo mismo**, tal como se muestra en la Ilustración 11. Esta de definición es recursiva, de modo que si los hijos de un nodo a su vez poseen hijos, primero se procesaran los hijos de los hijos del nodo, luego los hijos del nodo y finalmente al nodo.

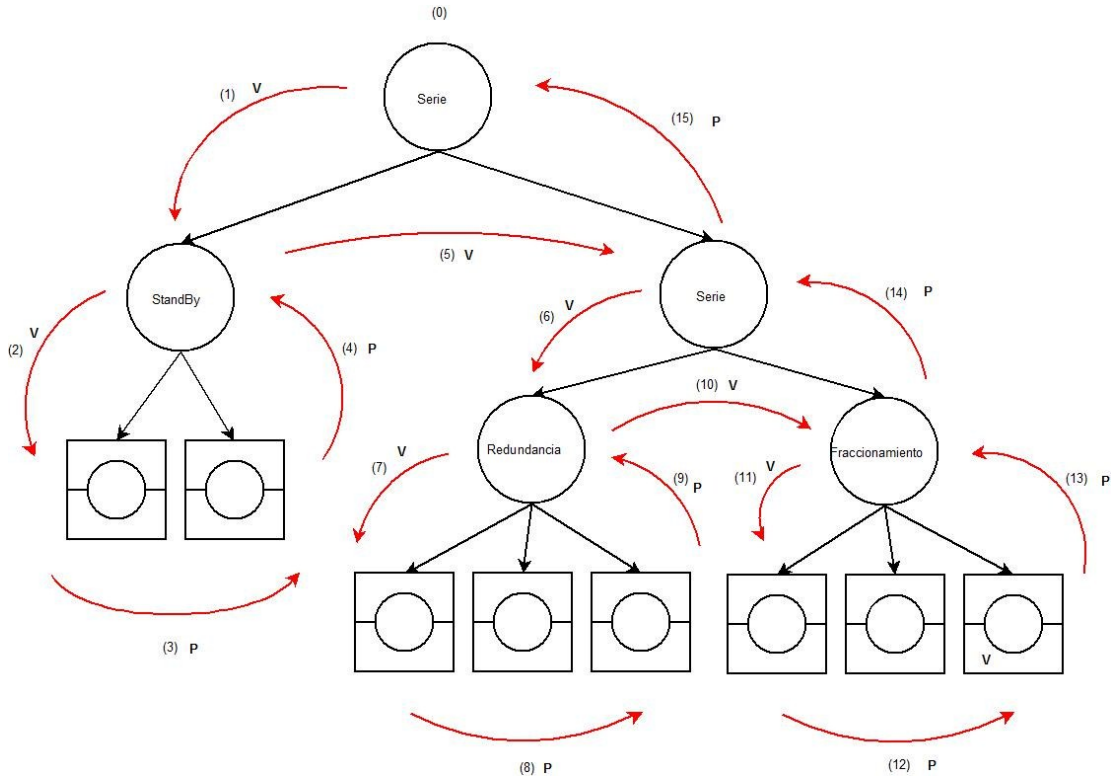

Ilustración 11. Recorrido Post-Orden donde V es visita y P es procesamiento

Observando la Ilustración 11:

- En el punto (0) comienza el recorrido. En ese punto, el nodo contiene 2 hijos, por ende primero visita a los hijos.
- En el punto (1) se visita al primer hijo de la raíz. Dado que este contiene 2 hijos más, se visitan éstos primero.
- En el punto (2) se visita al primer hijo de la configuración en StandBy.
- Sólo en el punto (3) se realiza el procesamiento de los hijos de la configuración en StandBy.
- En el punto (4), y sólo después de haber procesado todos los hijos del nodo en StandBy, se procesa la configuración.

Los siguientes puntos siguen la misma estructura explicada anteriormente. En resumen, las flechas que apuntan hacia abajo implican visita, y las que apunta hacia arriba, implican procesamiento.

Lo importante de comprender de este concepto es que **en todo momento, al procesar un nodo padre, los hijos (y toda su descendencia) ya han sido procesados y se tiene toda la información lista para ser usada por el padre.**

Es importante tener claro este algoritmo dado que es ampliamente utilizado por RMES en muchos procesos, principalmente en los referidos a cálculos históricos y probabilísticos, e incluso en procesos estocásticos.