# DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RESTRICCIONES PARA BÚSQUEDA DE PATRONES EN SECUENCIAS DE ADN PARA MOZART

**Ángela Patricia Villota Gómez**

**Escuela de Ingeniería de Sistemas y Computación**

# DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RESTRICCIONE PARA BÚSQUEDA DE PATRONES EN SECUENCIAS DE ADN PARA MOZART

**Ángela Patricia Villota Gómez**

Trabajo presentado para optar al titulo de Ingeniera de Sistemas.

**Escuela de Ingeniería de Sistemas y Computación**

**Facultad de Ingeniería**


Santiago de Cali
27 de febrero de 2006

Esta tesis titulada:
DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RESTRICCIONES PARA BÚSQUEDA DE PATRONES EN SECUENCIAS DE ADN PARA MOZART
escrita por Ángela Patricia Villota Gómez
ha sido aprobada por la Escuela de Ingeniería de Sistemas y Computación

Director: Profesor Juan Francisco Díaz Frıíaz, PhD

Evaluador: Profesor James Jerson Ortiz Vega

Fecha: Marzo 1 de 2006

La copia final de esta tesis ha sido leída por los profesores que firmamos y encontramos que tanto la forma como el contenido se adecuan a los estándares académicos de un trabajo de pregrado en Ingeniería de Sistemas.

*A mi mamá y a mi pequeño hijo.*

# Agradecimientos

Doy gracias a **Juan Francisco Díaz** quien me ha brindado su orientación, apoyo y confianza, a mi esposo **Gerardo Sarria** por motivarme, por ser mi motivo y por conservar su fé en mi, aún en los momentos en los que yo la he perdido.

Quiero Dar gracias también a:

- Mis profesores en la Escuela de Ingeniería de Sistemas y Computación de la Universidad del Valle, en especial a **Martha Millán** y a **Ángel García**.
- Al grupo **AVISPA** y a mis compañeros Andrés Becerra**,** Carlos Martínez y Javier Mena.

Por último pero no menos importante, agradezco a mi familia y a todos aquellos que de una u otra forma ayudaron a la realización de este proyecto.

# Resumen

En las biosecuencias (ADN, ARN, proteínas) existen subsecuencias que se repiten y que pueden ser descritas por patrones; estos patrones pueden describir características comunes en las secuencias que permiten asociarlas a una familia o conjunto de secuencias. La importancia de asociar a una secuencia en una familia está en la posibilidad de predecir características que son comunes para las secuencias que pertenecen a la misma familia. Esto último facilita la identificación de una molécula de ADN y sus características cuando es secuenciada en el laboratorio.

Los patrones a ser encontrados, deben ser buscados en cadenas de ADN de longitud grande. Esta búsqueda es un problema en el cual el espacio de búsqueda aumenta de tamaño en función de la cantidad de caracteres en la secuencia en la que se buscarán dichos patrones. Teniendo en cuenta lo anterior, se realiza una propuesta basada en estudios anteriores ([13] [16] [17] ) en la cual se representan los patrones como restricciones sobre subsecuencias de la secuencia de entrada. La solución del problema es encontrar las subsecuencias de la Secuencia de entrada que sean consistentes con las restricciones impuestas, es decir que puedan ser descritas por medio de los patrones establecidos como restricciones.

El modelo de la solución es una propuesta para resolver este tipo de problemas por medio de la programación con restricciones usando el lenguaje de programación Mozart.

# Índice general

# Introducción

En la actualidad existen muchos proyectos que abordan el estudio del funcionamiento y composición de los organismos de los seres vivos. Estos proyectos tienen que ver con nuevas vacunas, nuevos medicamentos, mejoras en los cultivos y estudios relacionados con genética en general. La información que se obtiene a partir de estas investigaciones es muy extensa, de manera que se vuelve esencial el buen manejo de la información y su análisis. Esta información en la mayoría de los casos está relacionada con cadenas de ADN, ARN y proteínas. La tarea de las personas que trabajan en computación en el área de Bioinformática, consiste entonces en la creación de herramientas que permitan hacer un buen manejo de la información biológica y en el análisis de ésta, con el fin de obtener conocimiento, a partir de dicha información; así como también en el desarrollo de nuevos algoritmos y en la exploración de nuevas técnicas para analizar dicha información. Es por esto que se puede afirmar que el trabajo de los biólogos y los informáticos en esta área es conjunto porque se desea mejorar la calidad de vida, por medio del estudio de la estructura y la actividad de las moléculas esenciales utilizando mecanismos de la ciencia de la computación.

A partir de las investigaciones en biología molecular y genética se ha descubierto una gran variedad de problemas que necesitan ser resueltos computacionalmente por la cantidad de datos que deben ser analizados, en especial porque estos datos son útiles para trabajos futuros. Esto se debe a que biológicamente existen homologías entre individuos y la información relacionada al funcionamiento y estructura de sus organismos. Está información se traduce de la secuencias de las móleculas de ADN, ARN y proteínas. Entre los problemas que se pueden resolver están los problemas relacionados con la información, que generalmente tratan acerca de las secuencias de aminoácidos o nucleótidos que componen las moléculas. Estas secuencias se comparan, se traducen, o se utilizan para la predicción de características físicas como forma y funcionamiento. Visto desde un punto de vista computacional, se hace alineamiento de cadenas, o búsqueda de patrones en el caso de la comparación.

Algunos de los problemas que operan sobre la información biológica, han sido representados como problemas con variables y dominios. Se han definido también restricciones en estos modelos utilizando conjuntos y dominios finitos. De esta forma, se ha demostrado[3, 13, 1] que en estos problemas se puede utilizar el paradigma de restricciones. La naturaleza combinatoria[1] y la posibilidad de trabajar con restricciones en

[1]En general, un problema combinatorio tiene un conjunto de variables con dominios asociados, un conjunto de restricciones que deben cumplirse, un conjunto de restricciones opcionales y un criterio de optima-

este tipo de problemas sugiere una implementación en Mozart debido a que Mozart es un lenguaje de programación multiparadigma que soporta la programación concurrente con restricciones. Para poder implementar soluciones a estos problemas en Mozart sería necesario conocer el modelo que se ha propuesto para cada problema específico. Esto último sería una tarea pesada para un programador que no conozca a fondo la teoría que envuelve el modelo. Lo ideal seria poder programar a un más alto nivel por medio de un sistema de restricciones apropiado a este tipo de problemas.

La investigación en este proyecto estará orientada hacia el diseño e implementación en Mozart de un sistema de restricciones apropiado para encontrar soluciones a problemas de búsqueda de patrones en secuencias de ADN. El proyecto de grado se llevó a cabo en el marco de el proyecto Crisol que realizó el grupo de investigación AVISPA.

El Grupo Avispa (Universidad Javeriana, Universidad del Valle) ha conducido investigación básica por varios años en el área de Programación con Restricciones. Ahora está realizando investigación práctica en el uso de este modelo computacional en problemas industriales por medio de una alianza estratégica con el Parque Tecnológico de Software de Cali (el proyecto fué soportado parcialmente por Colciencias, la agencia colombiana para la financiación de la investigación, Contrato No. 298-2002).

lidad que permite comparar soluciones. [14]

# Parte I

# Objetivos y Estado del Arte

# Capítulo 1

# Objetivos

En el área de Bioinformática hay una amplia gama de problemas a resolver. El objetivo de este proyecto de grado, es definir un sistema de restricciones sobre las secuencias, que permitan modelar varios tipos de problemas existentes en Bioinformática.

## 1.1. Objetivo general

Definir, diseñar e implementar un sistema de restricciones apropiado para resolver problemas de búsqueda de patrones en secuencias de ADN y que opere bajo el modelo de programación con restricciones de Mozart.

## 1.2. Objetivos específicos

- Definir el conjunto de símbolos de constantes, funciones y relaciones que se pueden usar para construir sintácticamente las restricciones sobre secuencias. (Alfabeto de el Sistema $\Sigma$)
- Definir el conjunto de sentencias escritas con el alfabeto $\Sigma$ válidas en el sistema que se está definiendo. (Teoría $\bigtriangledown$)
- Investigar diferentes representaciones y algoritmos que sean útiles para la búsqueda de patrones en secuencias.
- Escoger una representación de las secuencias apropiada para la implementación de los algoritmos, o definir una a partir de las representaciones investigadas, teniendo en cuenta las características de la implementación de los propagadores en C++ para Mozart.
- Comparar los algoritmos investigados y definir los adecuados para la implementación de los propagadores, teniendo como criterio: la cantidad de memoria utilizada y el tiempo de ejecución, como también la capacidad que tiene el algoritmo

para reducir los dominios de las variables y la cantidad de ramas exploradas (en el árbol de búsqueda).

- Implementar estos algoritmos como propagadores en el lenguaje C++
- Realizar la implementación de tal forma que cumpla con las especificaciones necesarias para que los propagadores puedan ser utilizadas como una contribución al lenguaje Mozart.

# Capítulo 2

# Estado del Arte

En el área de bioinformática es muy amplia la gama de problemas que pueden resolverse, computacionalmente hablando. Además los problemas que pueden resolverse, cuando se trata de analizar las biosecuencias son en su mayoría combinatorios.

Los problemas que se resuelven por satisfacción de restricciones tienen características especiales, tales como la posibilidad de expresar dichos problemas en términos de dominios y variables, y en ocasiones de poseer algún criterio de optimalidad.

La búsqueda de patrones en secuencias ha sido modelada utilizando variables lógicas y restricciones sobre estas variables. Los modelos consultados hasta ahora, encierran las características de las secuencias y las operaciones que puedan realizarse con ellas y tienen en cuenta la teoría de lenguajes formales. Entre estas está la solución planteada por David Gilbert[3] y su equipo de trabajo quienes plantearon un lenguaje (teoría de lenguajesa de lenguajes regulares), una gramática y una solución utilizando prolog. Además se propuso una representación de los patrones como restricciones, que permite hacer una búsqueda de subcadenas en una cadena de entrada, en donde la respuesta son las subcadenas que pertenecen a esta cadena de entrada y que cumplen las restricciones.

El algoritmo propuesto realiza la reducción (poda) de los dominios de cada una de las variables teniendo en cuenta las distintas restricciones o características del patrón. En el articulo en que se plantea esta solución[3], se menciona que si la poda se realiza de forma concurrente, la respuesta se puede encontrar de forma mas rápida. Es decir se evidencia la necesidad de que existan procesos que actúen casi autónoma y concurrentemente, verificando la satisfacción de las restricciones y la eliminación correcta de los valores de los dominios.

En términos de búsqueda, existen soluciones como la propuesta por Neuwand and Green[13] quienes definen los patrones de la siguiente forma:

$$P = A_1 - x(d_1) - A_2 - x(d_2) - A_3$$

En donde $A_1$, $A_2$, y $A_3$ representan aminoácidos, y $d_1$, $d_2$ son enteros que tienen una cota superior, es decir, enteros positivos menores que veinte ([1..20]). En pocas palabras se está buscando una subcadena que empiece por $A_1$ así: $A_1$x $A_2$x $A_3$, en donde

las "x" son variables a determinar y están en los dominios $d_1$ y $d_2$ .Las subcadenas que concuerdan con el patrón son determinadas por medio de búsqueda en profundidad (depth first search)

Otra solución es la propuesta de Jonassen[13] en la cual cambia la representación de los patrones otorgándoles mayor expresividad. Los patrones se representan de la siguiente forma:

$$P = A_1 - x(i_1, j_1) - A_2 - x(i_2, j_2)\text{-...}-A_{p-1} - x(i_{p-1}, j_{p-1}) - A_p$$

En donde las $A_k$ son variables, $i_k$, $j_k$ son enteros que determinan el mínimo y el máximo numero de símbolos que puede corresponder a la cadena $x(i_k,j_k)$. Se realiza búsqueda en profundidad; esta estrategia es usada para encontrar patrones en la base de datos PROSITE.

La alternativa propuesta por David Gilbert [3] se puede resumir de la siguiente forma: dada una cadena de entrada (secuencia de ADN), se determina la correspondencia a un patrón si por cada componente del patrón, la cadena de entrada contiene una subcadena con un componente tal que las restricciones se satisfacen. Las restricciones son de 5 tipos:

1. Longitud.
2. Distancia (entre subsecuencias)
3. Contenido (i.e. el segundo elemento es a)
4. Posición.
5. Correlación.

Por el momento no se ha resuelto el problema utilizando el lenguaje de programación Mozart, pero las soluciones pueden naturalmente ser implementadas en este lenguaje. Teniendo en cuenta lo anterior y como se menciona en los objetivos, la meta es poder definir un sistema de restricciones para secuencias, basándose en las soluciones propuestas que estén orientadas a la satisfacción de restricciones, e implementar los propagadores para búsqueda de patrones de tal forma que puedan ser utilizados en el lenguaje de programación Mozart.

# Parte II

# Marco Teórico

# Capítulo 3

# Bioinformática

Existen en la actualidad varias definiciones para el término Bioinformática; según un articulo publicado por David Gilbert [1] las más comunes son:

- Es un subconjunto de la biología, que utiliza la matemática y la computación.
- Es un subconjunto de las matemáticas y las ciencias de la computación y que necesita de la biología(remotamente).
- Es una ciencia interdisciplinaria, motivada por el fin de resolver problemas de la biología desde un punto de vista matemático por medio de las ciencias de la computación.

En resumen, podríamos decir que la Bioinformática tiene que ver con resolver problemas de la biología usando metodologías de la ciencia de la computación. Uno de los aspectos importantes en esta área es la necesidad de herramientas, implementación de algoritmos y de diseño de nuevos algoritmos que permitan analizar los datos Biológicos.

Lo que hace tan particular a los datos Biológicos (biosecuencias) es lo que los científicos llaman "el dogma central", es decir la forma en que se transmite la información biológica entre los individuos, así como también la gran cantidad de datos que además de almacenarse deben analizarse. "La información contenida en el ADN se transcribe en ARN y se traduce en proteínas".

## 3.1. Las Biosecuencias

Entre los conceptos biológicos modelados como secuencias (llamados biosecuencias) encontramos a las proteínas el ADN y el ARN. La importancia de estas moléculas está en la información que contienen y cómo la transmiten.

### 3.1.1. ADN

La molécula de ADN está constituida por dos largas cadenas de nucleótidos unidas entre sí formando una doble hélice. Las dos cadenas de nucleótidos que constituyen una molécula de ADN, se mantienen unidas entre sí porque se forman enlaces entre las bases nitrogenadas de ambas cadenas que quedan enfrentadas.

### 3.1.2. Proteínas

Las proteínas son polímeros constituidos por una cadena o secuencia de monómeros llamados aminoácidos, de los cuales veinte tipos están involucrados en la composición de proteínas.

El ADN tiene la información para hacer las proteínas de la célula. El ARN, en cambio, es la copia de trabajo de la información genética. El ARN que lleva las instrucciones para la síntesis de proteínas se denomina ARN mensajero. Este determina el orden en que se unirán los aminoácidos. Esta información está codificada en forma de tripletas, cada tres bases constituyen un codón que determina un aminoácido. Las reglas de correspondencia entre codones y aminoácidos constituyen el código genético.

## 3.2. Clasificación de los problemas de bioinformática

Los problemas que pueden ser resueltos computacionalmente son:

- Problemas relacionados específicamente con el dogma central: este tipo de problemas se refiere a algún nivel especifico de información, como secuencias, estructura o función.
- Problemas relacionados con datos en general: con el crecimiento exponencial del conocimiento en la biología molecular, aparecen problemas como el almacenamiento y análisis de datos.
- Simulación de procesos biológicos: esto significa en general la predicción de comportamiento dinámico de sistemas biológicos con base de sus componentes.

Entre los problemas relacionados con la información, encontramos los problemas que tienen que ver con secuencias, es decir con la conformación de la molécula. La cantidad de datos disponible para los investigadores en bioinformática crece rápidamente, incrementando también la necesidad de analizarlos. Una alternativa es establecer patrones(por ejemplo restricciones) en estos datos que permitan la caracterización de una molécula o de una familia de éstas.

## 3.3. Problema de Búsqueda de patrones

Un problema importante en biología molecular es la predicción de propiedades biológicas de una molécula (ADN, ARN, proteína) a partir de su secuencia; en particular

predicción de estructura y función. Las proteínas se agrupan en familias, en donde los miembros de cada familia tienen estructuras similares. Las características comunes en las familias de proteínas pueden ser descritas como un patrón en su secuencia, y se pueden formular hipótesis sobre la pertenencia o no de una proteína a una familia si esta cumple con el patrón.

Las cadenas de ADN, ARN y proteínas son cadenas compuestas por moléculas relativamente pequeñas. En el caso de las proteínas existen 20 diferentes aminoácidos que las componen; en el caso del ADN son 4 diferentes bases. De esta forma una biosecuencia puede codificarse como una cadena sobre un alfabeto de 20 ó de 4 elementos. Entonces si se obtiene una nueva biosecuencia y se desea conocer sus características u otro tipo de información (función, estructura), se determina su pertenencia o no a una familia si en esta secuencia se encuentran subcadenas que cumplan con todos los patrones establecidos para la familia. Se puede observar la dificultad de encontrar una subcadena que cumpla con un patrón, puesto que en cada posición de la cadena hay uno de 4, o 20 símbolos distintos según el caso (ADN, ARN o proteína)

# Capítulo 4

# Programación Concurrente por restricciones

## 4.1. Satisfacción de restricciones

### 4.1.1. Restricciones

Las restricciones son una formalización matemática de las relaciones en el mundo real, tales como, “padre de”, “al lado de”, “subcadena de”, entre otras. Lo interesante es que cualquier relación se puede modelar por medio de restricciones.

Un problema resuelto por satisfación de restricciones está compuesto por unas Variables, un Dominio asociado a cada Variable y unas Restricciones para las variables. Para dar una solución para este tipo de problemas se realiza una búsqueda de valores, pertenecientes al Dominio que cumplan con las Restricciones.

Las Variables tienen una característica especial: son Variables lógicas, esto quiere decir que pueden ser utilizadas sin que se sepa explícitamente cómo se obtiene su valor; la información sobre los valores que contienen se obtiene a partir de las relaciones entre las variables (Restricciones).

Una restricción puede considerarse básica si es de la forma:

- X $\in$D, en donde X es una variable y D un dominio
- X = v1, X es una variable y v1 es un valor

Una restricción no básica es un conjunto de restricciones básicas unidas por conjunciones:

$$C = c_1 \wedge c_2 \wedge c_3 \wedge ...c_n$$

Cuando se tiene un problema de restricciones existen tres tipos de operaciones básicas que se pueden realizar: optimización, simplificación y determinar satisfactibilidad.

### 4.1.2. Valuaciones

Una valuación para un conjunto de variables, es una asignación de un valor a cada variable teniendo en cuenta que el valor escogido pertenece al dominio de cada variable. Una solución para un problema de satisfacción de restricciones es una valuación, que cumpla con todas las restricciones del problema. De igual forma, un conjunto de restricciones es satisfactible si tiene por lo menos una solución.

### 4.1.3. Búsqueda y Solver

Un solver es un algoritmo que resuelve un problema de satisfacción de restricciones. El algoritmo determina la satisfactibilidad del problema a medida que construye una solución. La obtención de la solución se hace por medio de búsqueda, es decir, se generan valuaciones hasta encontrar una que satisfaga las restricciones.

En el caso del lenguaje de programación Mozart, se obtiene una solución por medio de un árbol de búsqueda que se construye utilizando Backtraking.

## 4.2. Sistemas de Restricciones

Como se mencionó anteriormente, las restricciones son relaciones que se establecen sobre variables que están asociadas a un dominio. El tipo de dominio de la variable determina "la sintaxis" de las restricciones, o sea las reglas para construir restricciones sobre el dominio; éste también determina los valores que las variables pueden tomar. En el caso de las cadenas de ADN, cada variable puede tomar un valor de entre las 4 posibles bases.

Un sistema de restricciones está compuesto por un alfabeto y una teoría que también están estrechamente relacionados con los dominios.

El alfabeto $\Sigma$ está compuesto por los símbolos de constantes, funciones y relaciones que se pueden usar para construir sintácticamente las restricciones en el sistema que se está definiendo. Las restricciones se construyen de la misma manera que se escriben sentencias en la lógica.

La teoría$\nabla$ es un conjunto de sentencias escritas con el alfabeto $\Sigma$ válidas en el sistema que se está definiendo.

Por ejemplo en las restricciones de dominios finitos:

- las variables pueden tomar valores en conjuntos finitos de valores pertenecientes a los números naturales, estos son los dominios. Ejemplo X en [1...10]
- las operaciones aritmeticas (+,-,*.. etc), las relaciones de igualdad y desigualdad y funciones como potencia, forman parte del alfabeto $\Sigma$.
- los teoremas de la aritmética forman parte de la teoría $\nabla$.
- Una restricción sería:

$$X \geq 3 + Y$$

# Capítulo 5

# Solución de CCPs usando Mozart

Un problema de restricciones es una tripleta $(V, D, R)$ en donde se encuentran las variables del problema, los dominios de las variables y las restricciones. El tipo de valores que pueden tomar las variables caracteriza a los dominios; es por esto que se pueden encontrar Dominios Finitos, Reales, entre otros.

Mozart resuelve los problemas de restricciones por medio de propagación y búsqueda.

## 5.1. Modelo de Solver basado en propagación de restricciones

Un solver basado en propagación de restricciones es aquel que combina dos tecnicas: propagación de restricciones y búsqueda.

La propagación de restricciones es un procedimiento determinista por medio del cual se excluyen o *filtran* los valores incompatibles con una posible solución de los dominios de las variables.

La búsqueda se realiza debido a que usualmente la propagación de restricciones no es suficiente para encontrar soluciones a un problema, es decir, no se puede filtrar todos los valores incompatibles para dejar un único valor solución en los dominios. Tan pronto la propagación de restricciones cesa (porque no hay más valores incompatibles que filtrar), se activa la búsqueda. Por medio de la búsqueda se elige una alternativa que filtra valores de los dominios de algunas variables de forma especulativa; este proceso de filtro activa de nuevo la propagación de restricciones. Este proceso se repite hasta que se encuentra una solución o todas las alternativas posibles son exploradas.

Un solver basado en propagación de restricciones está compuesto por: propagación de restricciones y búsqueda, los cuales son procesos distintos e independientes, que a su vez son realizados por mecanismos diferentes: la *máquina de propagación* (propagation engine) y la *máquina de búsqueda* (search engine).

### 5.1.1. Máquina de propagación

La máquina de propagación es la encargada de realizar la propagación de restricciones. Según el modelo usado por Mozart, está constituida por unas *Herramientas de propagación* que son genéricas y unos *Dominios del solver* los cuales proveen las variables y propagadores que son inherentes a cierto tipo de dominio (ej: dominios Finitos, reales, conjuntos, entre otros). Un Sistema de Restricciones es representado por un Dominio del Solver.

Una de las ventajas del modelo fig 2.1[1] de la máquina de propagación es que las *Herramientas de propagación* se encargan de controlar y administrar las variables y propagadores del *Dominio del solver* independientemente del contenido de este, además de permitir la interacción de varios Dominios del solver, incluso la cooperación entre estos.

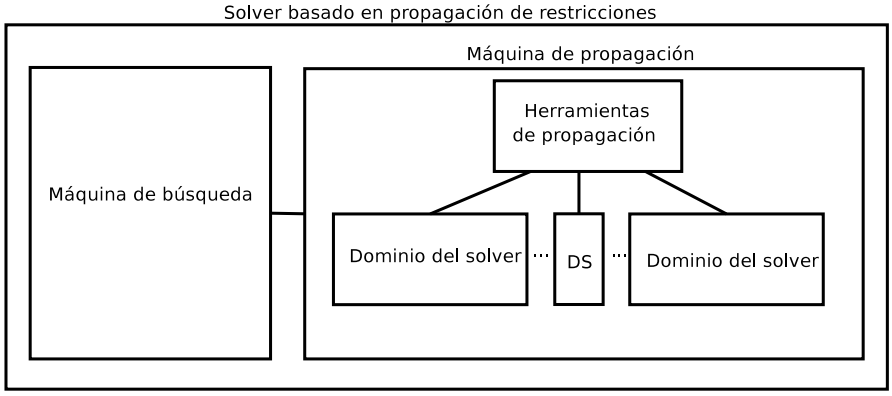


Figura 2.1

## 5.2. Restricciones

Según el modelo de programación concurrente por restricciones (CCP) visto en el capítulo anterior existen dos tipos de restricciones: *restricciones básicas y no básicas.* En el modelo utilizado por Mozart (basado en propagación de restricciones) estos conceptos son modelados haciendo uso del *"Store"* o almacén de restricciones y de los propagadores.

"El *Store"* o almacén de restricciones contiene la conjunción de cada una de las restricciones básicas, además, el *Store* representa el estado del problema en cierto punto del proceso de propagación y búsqueda debido a que los dominios que en él se encuentran tienen valores que han sido filtrados por el proceso de propagación y las decisiones tomadas por la búsqueda.

Por otra parte, las restricciones no básicas (que tienen que ver con relaciones entre variables) son representadas por un propagador.

[1] Figura modificada del documento *Constraints propagation in Mozart[18]*

### 5.2.1. Propagadores

Un propagador es un agente computacional concurrente que se impone entre un conjunto de variables. El propagador hace cumplir una restricción ejecutando una *Función Filtro* (inherente al propagador) usando como parámetros el conjunto de variables pertenecientes a la restricción. Como resultado algunos valores de los dominios de las variables inconsistentes con la restricción son filtrados. En la mayoría de los casos una variable puede estar asociada a más de un propagador, por lo tanto el proceso de filtro de dominios de un propagador, activa el proceso de filtro de los demás propagadores asociados a dicha variable.

Un propagador está constituido por una *cabeza* y un *cuerpo* los cuales encapsula y son accesibles por el propagador mismo.

El cuerpo del propagador contiene la función filtro y referencias a sus parámetros; esta información esta directamente relacionada con el *Dominio del Solver.*

La *cabeza* del propagador es la parte que hace interfaz entre las *Herramientas de propagación* (genéricas) y el cuerpo del propagador (inherente al *Dominio del Solver*) así como también el estado del propagador.

### 5.2.2. Estado de un propagador

Entre las *Herramientas de propagación* encontramos un proceso encargado de enviar a ejecución los propagadores, este proceso se sirve del estado de cada propagador para saber si lo envía o no a ejecución. Los Estados de un propagador fig 2.2 son: *running* (ejecutandose)*, runnable* (para ejecutar)*, sleeping* (suspendido)*, entailed* (implicado) *y failed* (fallido).

En el momento en que un propagador es creado se ejecuta inmediatamente su función filtro, por lo tanto el primer estado que recibe un propagador es *running*. Después de ejecutada la función filtro del propagador se conoce el nuevo estado del propagador. Un propagador tiene como estado *failed* (fallido) si la restricción es inconsistente con el *Store*, esto significa que no será ejecutado de nuevo. Si el nuevo estado es *entailed (implicado)* significa que no agrega información al Store, al igual que un propagador fallido, el propagador implicado no será ejecutado de nuevo.

En el caso en que no se pueda agregar más información al store y tampoco hay incosistencias después de correr la función filtro de un propagador, el estado del propagador será *sleeping (suspendido),* esto quiere decir que se espera que el propagador agregue mayor información más adelante. Siempre que el dominio de alguna de las variables parámetro de un propagador es podado, el propagador se activa de nuevo y su estado cambia de *sleeping* a *runnable*, esto quiere decir que el proceso de Mozart encargado de planear las tareas a realizar (scheduler) pone al propagador en lista de ejecución. En la figura 2.2 [2] se muestra un diagrama de los posibles estados de un propagador:

[2]Figura modificada del documento *Constraints propagation in Mozart[18]*

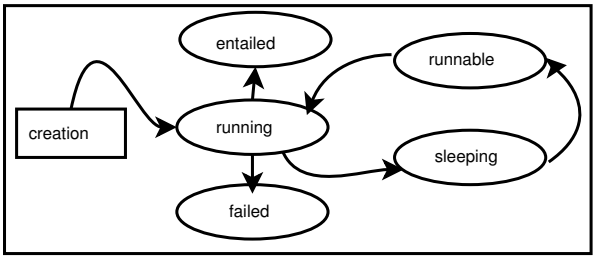


Figura 2.2

## 5.3. Nuevos Sistemas de Restricciones

Así como las restricciones no básicas son representadas por propagadores, los Sistemas de restricciones son representados por Dominios el Solver. Una de las ventajas de la arquitectura de un solver basado en propagación de restricciones, en especial la de la máquina de propagación, es la posibilidad de crear nuevos dominios del Solver (Sistemas de Restricciones). así como la conexión de estos a las Herramientas de propagación.

### 5.3.1. La *CPI,* interfaz de propagación de restricciones de Mozart

La interfaz de propagación de restricciones de Mozart *CPI,* es una interfaz en C++ que provee los mecanismos necesarios para construir Dominios del Solver que se puedan conectar a las herramientas de propagación. La CPI hace posible implementar nuevos dominios del Solver y extender los dominios del solver existentes por medio de la creación de nuevos propagadores.

# Parte III

# Modelo e Implementación

# Capítulo 6

# Modelo propuesto

El modelo de un sistema de restricciones para secuencias, implementando propagadores para búsqueda de patrones en secuencias de ADN, se basa en el modelo encontrado en el artículo de David Gilbert [3] en donde se proponen algoritmos que permiten resolver el problema usando programacion lógica con restricciones, y en donde una cadena o secuencia de ADN es modelada como una secuencia finita de números enteros $entre\{1,2,3,4\}$, dado que una cadena de ADN es una secuencia finita de nucleótidos conocidos :adenina, guanina, citosina y timina.

Como se había mencionado anteriormente, la naturaleza de los dominios y de las variables es *determinante* en la naturaleza del sistema de restricciones; en este caso el sistema de restricciones para secuencias de ADN no fué modelado desde cero, sino, que se creó utilizando las herramientas existentes en el sistema de restricciones de Dominios finitos teniendo en cuenta el modelo de las variables y los dominios.

Un patrón es modelado como una restricción sobre una secuencia de ADN; la ocurrencia de un patrón en una cadena de ADN implica la satisfacción de dicha restricción. Una restricción es una relación entre variables del sistema de restricciones. Todas las relaciones tienen que ver con la cadena de ADN de entrada, esta cadena es en la cual que se desea establecer la presencia de un patrón.

A continuación está el modelo adoptado para variables, dominios y restricciones.

## 6.1. Variables

Una variable representa una cadena, es denominada variable de cadena *"sv"* según el modelo original, en donde una variable de cadena es modelada por medio de una tupla que contiene dos variables de dominios finitos, las cuales corresponden a la posición inicial y final en la cadena de entrada, de la cadena que representan. El modelo original de una variable de cadena fué modificado, dado que para la implementación de una de las restricciones (Correlación), se requería información adicional (la longitud de la cadena). Una variable de cadena es entonces una tupla que contiene tres variables de dominio finito, que representan: las posiciones inicial y final en la cadena de entrada y la longitud de la cadena. De forma general una variable de cadena $\alpha$ se representa de

la siguiente forma:

$$\alpha = (\alpha_L, \alpha_U, \alpha_{long}), \text{ en donde}$$

$\alpha_L$ es el nombre que toma el primer campo de la tupla, la **L** quiere decir ***L****ower.*

$\alpha_U$ es el nombre que toma el segundo campo de la tupla, la **U** quiere decir ***U****pper*

$\alpha_{long}$ es el nombre que toma el tercer campo de la tupla , que representa la longitud de la cadena.

## 6.2. Dominios

Teniendo en cuenta que una variable es una tupla de tres campos en donde cada campo es una variable de dominio finito, es claro que los dominios de las variables sean dominios finitos. Siguiendo el modelo el dominio inicial de cada una de las variables estará entre $[1...longitud(cadena\_de\_entrada)]$.

$$\alpha_L :: [1...longitud(cadena\_de\_entrada)]$$

$$\alpha_U :: [1...longitud(cadena\_de\_entrada)]$$

$$\alpha_{Long} :: [1...longitud(cadena\_de\_entrada)]$$

## 6.3. Restricciones

Una restricción es una relación sobre una o más variables; en un sistema de restricciones, existen algunas restricciones implicítas que tienen que ver con el modelo adoptado.

En este caso existe una restricción implicíta entre los dos primeros campos de una variable de cadena: $\alpha_L y \alpha_U$ esto es que $\alpha_U \geq \alpha_L$. La característica particular de este modelo es que aunque las restricciones se imponen entre una o dos variables de cadena, las relaciones se *traducen* a relaciones entre las variables $\alpha_L, \alpha_U$ *y* $\alpha_{Long}$ de las variables de cadena. Lo anterior es lo que permite se pueda utilizar como base el sistema de restricciones de dominios finitos porque los campos $\alpha_L, \alpha_U$ *y* $\alpha_{Long}$ de una variable de cadena $\alpha$ son variables de dominio finito.

### 6.3.1. Relaciones sobre una sola variable de cadena

- Longitud: Establece un rango L de enteros el cual contiene valores posibles para la longitud de la variable de cadena. Parámetros: cadena de entrada y L rango
- Contenido: Determina los posibles valores que contiene una cadena en una de sus posiciones, los parámetros son: la variable de cadena, una posición en esta cadena y un conjunto no vacío de caracteres.
- Posición: permite restringir los dominios de los dos primeros campos de una variable de cadena (variables *Lower y Upper*) a un rango. Parámetros: variable de cadena y rango. Hay dos restricciones de este tipo, la que restringe la posición inicial y la que restringe la posición final.

### 6.3.2. Relaciones sobre dos variables de cadena

- Distancia: esta relación se refiere a la distancia contada en posiciones en la cadena de entrada, que debe existir entre dos variables de cadena $\alpha, \beta$ dada por un rango, existen 4 relaciones de este tipo, estas son:
  - Inicio_inicio: establece los posibles valores que puede tomar la distancia entre la posición inicial de una cadena $\alpha$ y la posición inicial de $\beta$.
  - Fin_inicio: establece los posibles valores que puede tomar la distancia entre la posición final de una cadena $\alpha$ y la posición inicial de $\beta$.
  - Inicio_fin: establece los posibles valores que puede tomar la distancia entre la posición inicial de una cadena $\alpha$ y la posición final de $\beta$.
  - Fin_fin: establece los posibles valores que puede tomar la distancia entre la posición final de una cadena $\alpha$ y la posición final de $\beta$.
- Correlación: es una relación entre los contenidos y las longitudes de dos variables de cadena. Una correlación tiene las siguientes propiedades:
  - Se impone sobre dos variables de cadena $\alpha$ y $\beta$, en donde $\alpha$ se llama *source* y $\beta$ *target.*
  - La longitud de las variables de cadena es la misma.
  - Una correlación además necesita dos relaciones: una de dirección y una relación sobre los símbolos de el alfabeto. El programador puede usar la restricción de correlación así:
    - Definiendo sus propias correlaciones pasando como parámetro las relaciones de dirección y de símbolo, ó
    - Usar una correlación previamente definida.

# Capítulo 7

# Definición del Sistema de Restricciones

## 7.1. Variables

En el modelo presentado en el capítulo anterior, se presenta una variable de cadena como una tupla de tres campos, en donde cada uno de los campos es una variable de dominio finito. Lo anterior en el lenguaje Mozart es representado por una estructura de tipo *Vector;* esto quiere decir que una variable de cadena puede ser declarada como una tupla, una lista o un registro de 3 campos en donde cada uno de los campos es una variable de dominio finito.

Para declarar una variable de cadena se utiliza un procedimiento del *Sistema de Restricciones* que encapsula operaciones de el sistema de restricciones de dominios finitos. A continuación se presenta un ejemplo de la declaración de dos variables de cadena: Sv1 y Sv2 en donde Sv1 es una tupla y Sv2 es una lista.

```
local
   Cadena  Sv1  Sv2  in
   Cadena="acgtacgtacgt"   %cadena  de  12  caracteres
    {Sq.sistema _inicio  Cadena }
    {Sq.tupla  Sv1 }
    {Sq.lista  Sv2 }
    {Browser.browse  ["esta  es  Sv1:"  Sv1] }
    {Browser.browse  ["esta  es  Sv2:"  Sv2] }
end
```

En la figura 7.1[1] se encuentra el resultado de ejecutar el anterior codigo.

[1] screen shot del browser de Mozart

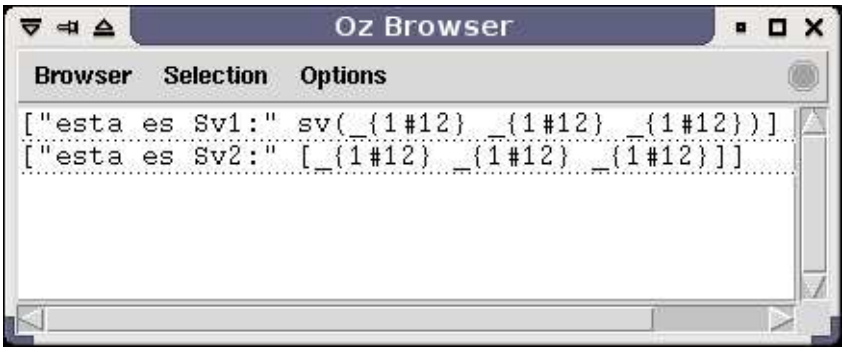


Figura 7.1

## 7.2. Dominios

Anteriormente se había mencionado que el dominio de las variables que conforman los tres campos de una variable de cadena es un dominio finito que tiene como característica principal que inicialmente sea: $[1...longitud(cadena\_de\_entrada)]$ Para declarar el dominio de una variable de cadena, se utilizan las operaciones de el sistema de restricciones para declarar dominios finitos, pero de forma transparente para el usuario.

## 7.3. Restricciones

### 7.3.1. Generalidades

Cada una de las 5 restricciones es una relación entre variables de dominios finitos, ya sea que relacione una o más variables cadena, por lo tanto para la implementación de cada una de estas restricciones se creó un nuevo propagador. El llamado de cada uno de los propagadores se realiza desde Mozart respetando la sintaxis de este lenguaje. Como se mencionó anteriormente cada uno de los propagadores tiene una cabeza y un cuerpo, así como también una parte que comparte con otros propagadores que son las variables de cadena sobre las cuales está impuesto. Cada nuevo propagador es un objeto que pertenece a la clase virtual "Propagator" en la CPI de Mozart, es por esto que tiene unos métodos que deben implementarse y que tienen que ver con el manejo de los parámetros y el tipo de estos. Cada nuevo propagador tiene un metodo qué es el encargado de hacer el filtro de los valores inconsistentes con la restricción que representa de los dominios de las variables de cadena. La arquitectura de la máquina de propagación de Mozart provee diferentes mecanismos que facilitan la implementación de los métodos de un nuevo propagador, en especial de aquellos que tienen que ver con el manejo de parámetros y los tipos de las variables a las que está asociado un propagador. Se debe prestar especial atención en el metodo encargado de filtrar los valores de los dominios de las variables parámetro; en este método es en donde se implementa en lenguaje C++ la función filtro asociada a un propagador. A continuación

se muestra el modelo de la fúnción filtro de cada uno de los 5 nuevos propagadores y algunos detalles de implementación.

## 7.3.2. Propagadores

### 7.3.2.1. Propagador de Longitud

La restricción de Longitud como aparece en el capítulo anterior, es una relación sobre una variable de cadena que restringe la longitud de una variable de cadena a un conjunto de enteros. Los parámetros de esta restricción son:

- Una variable de cadena $\alpha$
- Un entero que pertenece al rango $[1...Longitud\_de\_cadena\_de\_entrada]$ y
- Un literal que pertenezca al conjunto $\{'<:',':>',':>=',':<=',':='\}$

En lenguaje de programación Mozart una restricción de longitud se escribe por ejemplo de la siguiente forma:

*{Fsq.long A 2 ':='}*

la sentencia anterior impone la restricción: *La longitud de la variable de cadena A es igual a 2.*

#### Modelo de la función filtro

Esta restricción se traduce en una relación entre las variables que corresponden a los dos primeros campos de una variable de cadena $\alpha$ (*Lower y Upper*) teniendo en cuenta la restricción interna $\alpha_U \geq \alpha_L$ para cualquier variable de cadena $\alpha$. Teniendo en cuenta lo anterior, la función filtro de este propagador consiste en quitar de los dominios de las variables *Lower y Upper* asociadas a la variable de cadena $\alpha$ los valores que NO cumplan lo siguiente:

$$\alpha_U - \alpha_L + 1, \in Rango$$

Rango es un conjunto de números enteros resultado de aplicar la relación matemática denotada por los parámetros de la restricción (el literal y el numero entero). En el ejemplo anterior:

$$\textit{\{Fsq.long A 2 ':='\} es equivalente a } A_U - A_L + 1, \in \{2\}$$

Así, solo se dejan en los dominios de las variables que cumplan la condición. En cuanto a la variable $A_{Long}$ los valores que quedan en su dominio son únicamente aquellos que pertenecen al Rango denotado por la relación matemática que representan los parámetros de la restricción.

#### 7.3.2.2. Propagador de Posición

Al igual que la restricción de Longitud, la restricción de Posición es una restricción que se impone sobre una variable de cadena. El objetivo de esta restricción es acotar la posición inicial o final de una variable de cadena a un rango de enteros. Los parámetros de esta restricción son:

- Una variable de cadena $\alpha$
- Un Rango que pertenece al conjunto $[1...Longitud\_de\_cadena\_de\_entrada]$ y
- Un literal que puede ser $\{'inicio','fin'\}$

Un ejemplo de esta restricción escrita en Mozart es:

*{FSq.pos A [1 3 4 5 6 7] 'inicio'}*

La sentencia anterior denota la restricción "la posición inicial de la variable de cadena A $\in [1,3,4,5,6,7]$

**Modelo de la función Filtro:**

Esta restricción se traduce en una relación entre las variables que corresponden a los dos primeros campos de una variable de cadena $\alpha$ (*Lower ó Upper* según sea el caso) teniendo en cuenta la restricción interna $\alpha_U \geq \alpha_L$ para cualquier variable de cadena $\alpha$. En el caso de la *Función filtro* asociada a esta restricción se podría decir que es muy sencilla, porque la exclusión de valores de los dominios de la variable *(Lower ó Upper*) se hace de forma directa, dejando en su dominio únicamente los valores que corresponden al conjunto de enteros *Rango*. En el caso que el usuario ingrese como parámetro el literal *'inicio'* se impone la siguiente restricción básica sobre la Variable de cadena $\alpha$ *:*

$$\alpha_L :: Rango$$

En el caso en que el parámetro de entrada sea *'fin'* se impone:

$$\alpha_U :: Rango$$

#### 7.3.2.3. Propagador de contenido

La restricción de Contenido es una relación sobre una variable de cadena $\alpha$ que permite al usuario establecer la aparición de un *caracter* (o conjunto de caracteres) *especifíco* en una *posición deseada* de la variable de cadena. El llamado de esta restricción se hace teniendo en cuenta los siguientes parámetros:

- Una variable de Cadena $\alpha$
- una posición que es un entero que pertenece al conjunto $[-longitud(\alpha)...longitud(\alpha)]$ (longitud de la variable de cadena)

- Un conjunto de Caracteres C $\subseteq \Sigma$

Un ejemplo de el llamado a esta restricción es:

*{FSq.cont A ~2 "t" }*

La sentencia anterior quiere decir en lenguaje natural: en la penúltima posición de la variable de cadena A, está presente el caracter *"t"*.

**Modelo de la Función Filtro**

La *Función Filtro* de esta restricción se ciñe a la siguiente idea: "en la posición p de la variable de Cadena $\alpha$ debe haber uno de los símbolos del conjunto de caracteres C". Dado que la posición es un entero positivo o negativo entonces tenemos dos casos posibles para la función filtro:

si la posición es >=1 se halla el siguiente conjunto:

$PL : \{j | CadenaEntrada_{j+p-1} \in C\}$ en donde PL es un conjunto de enteros

y se filtran valores de el dominio de la variable $\alpha_L$ así:

$$\alpha_L :: Dom_{\alpha_L} \cap PL$$

Es decir se excluyen de el dominio de $\alpha_L$ los valores que no pertenezcan al conjunto *PL.*

En el caso en que la posición es <=-1 se halla el conjunto

$PU : \{j | CadenaEntrada_{j-p+1}\}$en donde PU es un conjunto de enteros

y se filtran valores de el dominio de la variable $\alpha_U$ así:

$$\alpha_U :: Dom_{\alpha_U} \cap PU$$

Es decir se excluyen de el dominio de $\alpha_U$ los valores que no pertenezcan al conjunto *PU.*

#### 7.3.2.4. Propagador de Distancia

La restricción de distancia se impone entre dos variables de cadena $\alpha, \beta$ y un rango de enteros D que es subconjunto del conjunto de enteros $\{1..longitud(Cadena\_Entrada)\}$. Esta restricción permite al usuario restringir la cantidad de posiciones que separan una variable de cadena de otra en la Cadena de entrada. La restricción de distancia tiene los siguientes parámetros:

- Dos variables de Cadena $\alpha, \beta$
- Un literal que pertenezca al conjunto $\{'inicio\_inicio', 'inicio\_fin', 'fin\_inicio', 'fin\_fin'\}$ y
- Un Rango D $\subseteq \{1..longitud(Cadena\_Entrada)\}$

En lenguaje de programación Mozart una restricción de distancia se escribe por ejemplo de la siguiente forma:

*{FSq.dist A B 'inicio_inicio' [1 2 3 4 5 6]}*

la sentencia anterior impone la restricción: *La distancia entre el inicio de la variable de cadena A y el inicio de la variable de cadena B es un número entero* $\in \{1,2,3,4,5,6\}$.

**Modelo de la Función Filtro**

Para poder filtrar los valores de los dominios de las variables de cadena $\alpha, \beta$, se debe tener en cuenta el literal que entra como parámetro, porque es dicho literal el que determina los campos de las variables de cadena que se utilizarán al realizar la relación que corresponde a la restricción de distancia. Como aparece en el capitulo anterior, existen 4 casos posibles: 'inicio_inicio', 'inicio_fin', 'fin_inicio' y 'fin_fin'. En cada uno de estos casos la *Función Filtro* opera con pequeñas variaciones:

- En el caso en que el literal sea 'inicio_inicio' la relación se establece entre las variables $\alpha_L, \beta_L$ y la función filtro excluye de los dominios de las variables los valores que NO cumplen con lo siguiente: $\beta_L - \alpha_L + 1, \in D$
- En el caso en que el literal sea 'inicio_fin' la relación se establece entre las variables $\alpha_L, \beta_U$ y la función filtro excluye de los dominios de las variables los valores que NO cumplen con lo siguiente: $\beta_U - \alpha_L + 1, \in D$
- En el caso en que el literal sea 'fin_inicio' la relación se establece entre las variables $\alpha_U, \beta_L$ y la función filtro excluye de los dominios de las variables los valores que NO cumplen con lo siguiente: $\beta_L - \alpha_U + 1, \in D$
- En el caso en que el literal sea 'fin_fin' la relación se establece entre las variables $\alpha_U, \beta_U$ y la función filtro excluye de los dominios de las variables los valores que NO cumplen con lo siguiente: $\beta_U - \alpha_U + 1, \in D$

En el ejemplo anterior *:*

*{FSq.dist A B 'inicio_inicio' [1 2 3 4 5 6]} es equivalente a*
$\beta_L - \alpha_L + 1, \in \{1,2,3,4,5,6\}$

#### 7.3.2.5. Propagador de Correlación

Cada variable de cadena representa una subcadena de la *Cadena de entrada*. Una restricción de correlación entre dos variables de cadena $\alpha$ y $\beta$ permite espresar restricciones del tipo:

- La subcadena $\beta$ es igual a la subcadena $\alpha$.
- La subcadena $\beta$ es el complemento de la subcadena $\alpha$.

- La subcadena $\beta$ es igual a la subcadena $\alpha$ al revés.

Cuando se impone una restricción de correlación entre dos variables de cadena $\alpha$ y $\beta$ se debe determinar cual de ellas es llamada ***source*** y cual ***target***[2]; en las restricciones anteriores $\alpha$ es llamada source y $\beta$ target porque el contenido de la subcadena $\beta$ depende directamente de el contenido de la subcadena $\alpha$.

Para expresar este tipo de restricciones se debe tener en cuenta que la longitud de las subcadenas debe ser la misma, que la relación se puede establecer entre la dirección de las subcadenas y el contenido de estas. A continuación se muestran las relaciones de contenido y dirección en los ejemplos anteriores:

- La subcadena $\beta$ es igual a la subcadena $\alpha$: tienen la misma dirección y el mismo contenido.
- La subcadena $\beta$ es el complemento de la subcadena $\alpha$: tienen la misma dirección y existe una función sobre los caracteres del alfabeto llamada complemento en donde $complemento(\alpha_i) = \beta_i$.
- La subcadena $\beta$ es igual a la subcadena $\alpha$ al revés: tiene el mismo contenido pero distinta dirección.

Entonces, una correlación es una restricción que relaciona dos variables de cadena $\alpha, \beta$ en cada uno de sus campos, así como también establece una relación de contenido y dirección entre cada uno de los caracteres que forman parte de $\alpha, \beta$. La restricción de correlación permite también expresar patrones estructurales. Para poder imponer una restricción de correlación se deben tener en cuenta los dos componentes adicionales:

* Dirección, $d \in \{1, -1\}$ y
* Contenido, $f(\varepsilon)$ en donde $\varepsilon$, $f(\varepsilon) \in \Sigma$.

Una correlación se escribe: $Cr(\alpha, \beta, d, f(\varepsilon))$ en donde $\alpha$ es llamada source y $\beta$ target. La restricción de correlación tiene como restricción interna que la longitud de las variables de cadena $\alpha, \beta$ debe ser la misma. Una correlación recibe como parámetros de entrada:

- Dos variables de cadena $\alpha, \beta$
- Un entero $d \in \{1, -1\}$ que denota la dirección y
- Una función $f(\varepsilon)$ en donde $\varepsilon$, $f(\varepsilon) \in \Sigma$

Una correlación en Mozart se escribe por ejemplo:

*{FSq.corr A C 1 "acgt" }*

[2]**Source:** fuente, origen.
**Target:** objetivo, meta.

**Modelo de la Función Filtro**

La *Función Filtro* de este propagador fué la más interesante y a la vez la más complicada de diseñar e implementar puesto que el modelo de ésta estaba enunciado en el artículo de Gilbert[3], más no era explicita la forma en que se llenaba la *matriz de correlación*, además que la exclusión de valores del dominio era mínima ya que no se tenía en cuenta la información adquirida por medio de las *Restricciones de Longitud*.

Una Matriz de correlación es una estructura que permite ubicar facilmente las posiciones en la cadena de entrada en donde existe la relación de contenido entre las dos variables de cadena que entran como parámetro de entrada. Para filtrar valores de los dominios de las variables de cadena se debe llenar la Matriz de correlación y hacer una búsqueda de diagonales dependiendo de el componente de dirección de una correlación. Para poder llenar una Matriz de correlación se debe tener en cuenta los siguientes planteamientos:

Sean $\alpha, \beta$ dos variables de cadena y C la cadena de entrada. $Cr(\alpha, \beta, d, f(\varepsilon))$ es una correlación entre $\alpha$ y $\beta$ en donde $\alpha$ es la variable *source*, $\beta$ la variable *target*, d es la dirección y $f(\varepsilon)$ una función sobre los caracteres del alfabeto.

Existe una matriz de correlación $M_{m\times n}$, en donde:

$$m = max(D_{\alpha_U}) - min(D_{\alpha_L}) + 1,$$

$$n = max(D_{\beta_U}) - min(D_{\beta_L}) + 1,$$

$$M_{i,j} \in [0,1],$$

$$i \in [min(D_{\alpha_L}) \ldots max(D_{\alpha_U})]\text{y}$$

$$j \in [min(D_{\beta_L}) \ldots max(D_{\beta_L})]$$

Cada uno en la matriz representa un valor que satisface la restricción de contenido entre los caracteres que pertenecen a las cadenas $\alpha$ y $\beta$ teniendo en cuenta la función $f(\varepsilon)$ de la siguiente forma:

$$M_{i,j} \begin{cases} 1, \text{ si y solo si } C_j = f(C_i) \\ 0, \text{ de lo contrario} \end{cases}$$

A continuación se muestra de que forma se llena una matriz de correlación:

Sea *C= "'actgcctacttact"'* una cadena de entrada y dos variables de cadena $\alpha$ y $\beta$.

$\alpha$ con longitud igual a 3 y $Dom(\alpha_L) = [1 \ldots 11], Dom(\alpha_U) = [4 \ldots 14], Dom(\alpha_{Long}) = [4]$.

$\beta$ con $Dom(\beta_L) = [5 \ldots 11], Dom(\beta_U) = [9 \ldots 14], Dom(\beta_{Long}) = [4]$.

Se impone la restricción de correlación $Cr(\alpha, \beta, 1, Id)$ , en donde *Id* es la función identidad.

Teniendo en cuenta las formulas anteriores:

$$m = max(D_{\alpha_U}) - min(D_{\alpha_L}) + 1,$$

$$n = max(D_{\beta_U}) - min(D_{\beta_L}) + 1,$$

en nuestro ejemplo $m = 14$ y $n = 10$, $i \in [1 \dots 14]$ y $j \in [5 \dots 14]$

En la tabla que aparece a continuación (Ver Tabla 5.1) se representa una Matriz de correlación. Los ceros aparecen en negro y los números que están en color azul representan los unos que cumplen con la condición $C_j = f(C_i)$. Aparecen también unos en color rojo, los índices de estos unos quedarán almacenados en los dominios, todo lo correspondiente a la poda de dominios está en la siguiente sección. En la primera fila y primera columna están los caracteres asociados a cada una de las posiciones $C_k$ en la cadena de entrada. Por ejemplo, en el caso de la posición $M_{2,1}$ (en nuestra tabla $T_{4,3}$) aparece un ***uno*** debido a que se cumple que $C_5 = f(C_2)$.

| | | c | c | t | a | c | t | t | a | c | t |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $C_5$ | $C_6$ | $C_7$ | $C_8$ | $C_9$ | $C_{10}$ | $C_{11}$ | $C_{12}$ | $C_{13}$ | $C_{14}$ |
| *a* | $C_1$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| *c* | $C_2$ | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| *t* | $C_3$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| *g* | $C_4$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *c* | $C$ | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| *c* | $C_6$ | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| *t* | $C_7$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| *a* | $C_8$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| *c* | $C_9$ | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| *t* | $C_{10}$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| *t* | $C_{11}$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| *a* | $C_{12}$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| *c* | $C_{13}$ | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| *t* | $C_{14}$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Tabla 5.1. Matriz de correlación

**Poda de Dominios**

Para podar los dominios de las variables $\alpha_L, \alpha_U, \beta_L, \beta_U$ se debe tener en cuenta la dirección de la correlación.

Si $d = 1$, se deben búscar unos en las diagonales de $M_{m \times n}$, de izquierda a derecha, y de arriba a abajo.

Si $d = -1$ la búsqueda de unos en las diagonales de la matriz se hará de izquierda a derecha, y de abajo a arriba. Los valores que quedan en los dominios de las variables de cadena son: En el dominio de $\alpha_L$ quedarán los valores que pertenezcan al conjunto $D_{\alpha_L} \bigcap M_{id}$ y en el dominio de $\alpha_U$ los valores que pertenezcan al conjunto $D_{\alpha_U} \bigcap M_{id}$, en donde $M_{id}$ son los índices de las filas que forman parte de las diagonales de unos que tengan una longitud mayor o igual al menor valor del Dominio de longitud de la variable $\alpha_{Long}$.

En el dominio de $\beta_L$ quedarán los valores que pertenezcan al conjunto $D_{\beta_L} \bigcap M_{jd}$ y en el dominio de $\beta_U$ los valores que pertenezcan al conjunto $D_{\beta_U} \bigcap M_{jd}$, en donde $M_{jd}$ son los índices de las columnas que forman parte de las diagonales de unos que tengan una longitud mayor o igual al menor valor del Dominio de longitud de la variable $\beta_{Long}$.

De esta forma en una solución si $d = 1, L_{\alpha+k} = L_{\beta+k}$ en donde $k \in [0... \, longitud(\alpha)]$. Si $d = -1, L_{\alpha+k} = L_{\beta-k}$en donde $k \in [0... \, longitud(\alpha)]$.

Retomando el ejemplo anterior, para hallar los valores de $M_{id}$y $M_{jd}$se tuvo en cuenta únicamente los índices correspondientes a *la Cadena de entrada* de los unos que están en color rojo en la tabla 5.1. Estos son los resultados:

$$M_{id} = [5,14] \text{ y } M_{jd} = [5,14]$$
$$Dom(\alpha_L) = [5,11] \text{ y } Dom(\alpha_U) = [5,14]$$
$$Dom(\beta_L) = [5,11] \text{ y } Dom(\beta_U) = [9,14]$$

# Capítulo 8

# Detalles de implementación

## 8.1. Propagadores

Cada uno de los nuevos propagadores fué implementado en C++ utilizando la *CPI de Mozart* (Constraints Propagation Interface) y siguiendo el tutorial de *Extensión de restricciones* [19].

En la ***CPI de Mozart****,* cada propagador es representado como un objeto de la clase abstracta *Propagator,* de esta forma todos los nuevos propagadores tienen los métodos necesarios para su interacción con las *herramientas de propagación* de Mozart. Las funciones filtro anteriormente descritas se implementaron en el método *propagate* de los propagadores.

### 8.1.1. Extensiones

El modelo adoptado para las variables de cadena del nuevo sistema de restricciones, permite que los 5 propagadores implementados utilicen las *herramientas de propagación* del sistema de restricciones de dominios finitos. Debido a que cada variable de cadena es un vector con 3 campos, se creó un nuevo objeto ***ExtendedExpect*** que permite utilizar vectores de variables de dominios finitos y vectores de enteros.

Se utiliza también el objeto ***Iterator*** que aparece en el tutorial de *Extensión de restricciones* [19], con el fin de facilitar el manejo de los vectores de variables de dominio finito, ya que se encarga de ejecutar alguna intrucción a cada uno de los campos de un vector.

## 8.2. Sistema de Restricciones

El nuevo Sistema de Restricciones se implemento como un *functor de OZ* que utiliza otro functor encargado de realizar el enlace entre los nuevos propagadores implementados en C++ y Mozart. Dentro del functor principal (Sistema de Restricciones) se hacen los llamados a los nuevos propagadores y se declaran 3 correlaciones que se utilizan con frecuencia en los ejemplos del articulo de David Gilbert [3] con el fin de

realizar comparaciones entre los resultados obtenidos por el Sistema de Restricciones y los resultados obtenidos por los propagadores implementados por David Gilbert en CLP.

El enlace entre los propagadores implementados en C++ y Mozart se realizo siguiendo el tutorial de *Extensión de restricciones* de Tobias Müller[19].

### 8.2.1. Documentación

La documentación del Sistema de Restricciones se presenta como páginas html navegables, siguiendo el formato de la documentación de Mozart. Estas páginas html se generaron desde el código por medio de la herramienta **Ozh.**

## 8.3. Herramientas

### 8.3.1. Ozmake

Ozmake es una herramienta creada por Denys Duchier que provee al programador capacidades de compilación automática similares a la herramienta **make** en los sistemas Linux. Para la compilación de las aplicaciones (propagadores y Sistema de Restricciones) se utilizó ozmake. Cabe mencionar que los propagadores que están codificados en C++ deben ser recompilados en la máquina del usuario (esto también se logra usando ozmake).

### 8.3.2. Ozh

Ozh es una herramienta creada por Nils Franzén y Andreas Sundström la cual es una herramienta que permite que los programadores creen automáticamente una interfaz que describa los functors que han creado. Esta herramienta toma un archivo como argumento, toma en cuenta los functor que se importan en dicho archivo y genera la documentación de los módulos como paginas html. De esta forma se puede navegar entre las páginas html que describen los functor con un simple click del *mouse*.

### 8.3.3. MOGUL

Este es el acrónimo para MOzArt Global User Library, un repositorio de contribuciones Oz que puede accederse a través de la web[1] ó a través de ozmake, para bajar o actualizar paquetes. La herramienta **Ozh** se encuentra en Mogul. Se espera dejar las herramientas desarrolladas en una sección de MOGUL una vez que se hayan robustecido adecuadamente.

[1] `http://www.mozart-oz.org/mogul`

# Capítulo 9

# Pruebas y Resultados

## 9.1. Pruebas internas

Se realizaron pruebas a cada uno de los nuevos propagadores teniendo en cuenta los diferentes parámetros que pueden recibir así:

- Propagador de Longitud: se verificó que funcionara correctamente cuando se pasa como parámetro cualquier literal que pertenezca al conjunto {'<:','>:','>=:','<=:','=:'}.
- Propagador de posición: se verificó que funcionara correctamente cuando se le pasa como parámetro cualquier lista de números enteros entre [1...Longitud_de_cadena_de_entrada] y literales que pertenezcan al conjunto {'inicio','fin'}.
- Propagador de contenido: para este propagador se verificó que la poda de los dominios se realizara correctamente usando posiciones negativas y positivas.
- Propagador de distancia: se verificó que funcionara correctamente cuando se le pasaban como parámetro cualquier literal que pertenezca al conjunto {'inicio_inicio','inicio_fin','fin_inicio','fin_fin'}.
- Propagador de correlación: para este propagador se verficó que la poda de los dominios se realizara correctamente usando las funciones identidad y complemento asi como direccion 1 y -1.

## 9.2. Pruebas externas

### 9.2.1. Enlace y Ejemplos

Cada uno de los archivos de pruebas se realizó como un functor de Mozart que importa en la variable Sq los procedimientos del functor secuencias.oz. En el functor secuencias.oz se hace el llamado a la libreria nativa **'prop_sq/seq.so{native}'** que es la que enlaza los propagadores implementados en lenguaje C++. Se implementó el ejemplo del artículo de David Gilbert [3] con el fin de hacr comparaciones.

#### 9.2.1.1. Ejemplo No. 1.

En el ejemplo que aparece a continuación, se imponen varias retricciones de longitud sobre la misma variable de cadena (llamada SV), la cadena de entrada consta de 12 caracteres, las 4 primeras restricciones no causan poda del dominio, la quinta impone la restricción: “la longitud de Sv es igual a 3”.

```
local

   Cadena  Sv  in
   Cadena="acgtacgtacgt"   %cadena  de  12  caracteres
    {Sq.sistema _inicio  Cadena }
   proc   {Script  Sv }
       {Sq.tupla  Sv }

       {Sq.longitud  Sv  13  «<:« }
       {Sq.longitud  Sv  12  «<=:« }
       {Sq.longitud  Sv  0  «>:« }
       {Sq.longitud  Sv  1  «>=:« }
       {Sq.longitud  Sv  3  «=:« }
       {FD.distribute  ff  Sv }
   end

    {Browser.browse  Sv }
    {Explorer.all  Script }
end
```

En la Figura 9.1 se encuentra el árbol de búsqueda de las soluciones, en la fig.9.2 están cada una de las soluciones encontradas por el motor de búsqueda de Mozart.

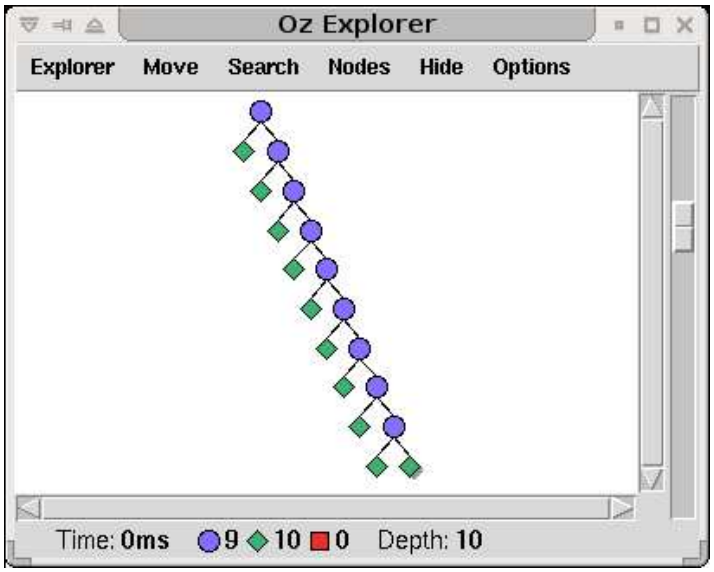


Figura 9.1

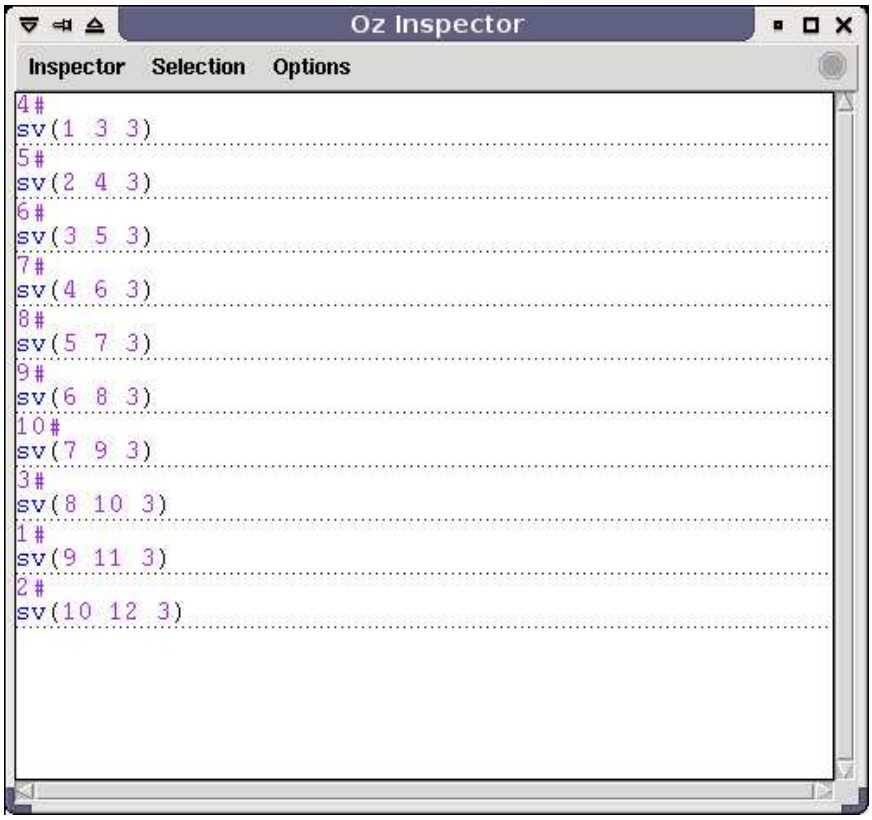


Figura 9.2

#### 9.2.1.2. Ejemplo No. 2.

El objetivo de este ejemplo es mostrar el desempeño de la restriccion de contenido, la cadena de entrada consta de 12 caracteres, se impuso las siguientes restricciones:

- El caracter de la segunda posición de la variable de cadena Sv es una “a”.
- El caracter de la última posición de la variable de cadena Sv en una “t”.
- La longitud de la cadena Sv es igual a 5.

```
local

   Cadena  Sv  in
   Cadena="acgtacgtacgt"   %cadena  de  12  caracteres
    {Sq.sistema _inicio  Cadena }
   proc   {Script  Sv }
       {Sq.tupla  Sv }
```

```
      {Sq.contenido  Sv  2  "a" }
      {Sq.contenido  Sv  ~1  "t" }
      {Sq.longitud  Sv  5  «=:« }
      {FD.distribute  ff  Sv }
end


{Explorer.all  Script }
```

En la Figura 9.3 se encuentra el árbol de búsqueda de las soluciones, en la fig.9.4 están cada una de las soluciones encontradas por el motor de búsqueda de Mozart.

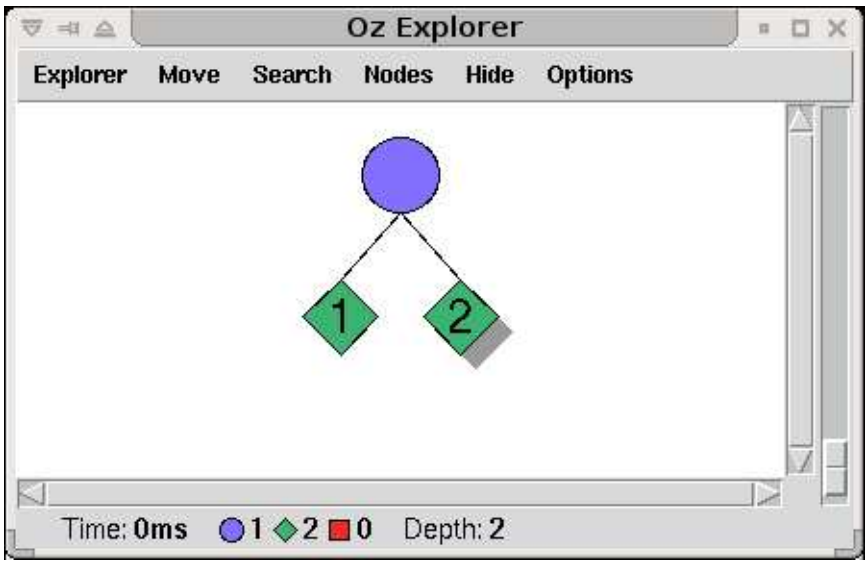


Figura 9.3


Figura 9.4

#### 9.2.1.3. Ejemplo No. 3.

En el siguiente ejemplo se imponen las restricciones anteriores junto con la restricción de distancia, la cadena de entrada consta de 12 caracteres y aparecen en esta

ocasión restricciones sobre dos variables de cadena, se impuso las siguientes restricciones:

- La posición inicial de la cadena Sv1 está en el conjunto [1,2].
- La distancia entre la posición final de la cadena Sv1 y la posición final de la caedna Sv2 es 5.
- La longitud de la cadena Sv1 es igual a 3.
- La longitud de la cadena Sv2 es igual a 3.

```
local

   Cadena  Sv1  Sv2   in
   Cadena="acgtacgtacgt"   %cadena  de  12  caracteres
    {Sq.sistema _inicio  Cadena }
    {Sq.tupla  Sv1 }
    {Sq.tupla  Sv2 }
    {Browser.browse  ["Sv1:"  Sv1  "Sv2:"  Sv2] }
    {Sq.posicion  Sv1  «inicio«  [1  2] }
    {Sq.distancia  Sv1  Sv2  «fin _inicio«  [5] }
    {Sq.longitud  Sv1  3  «=:« }
    {Sq.longitud  Sv2  3  «=:« }
    {Browser.browse  ["cadena  de  Entrada:"  Cadena] }
end
```

En la Figura 9.5 se encuentra el browser de Mozart en donde se puede ver el resultado de la propagación al imponer las anteriores restricciones.

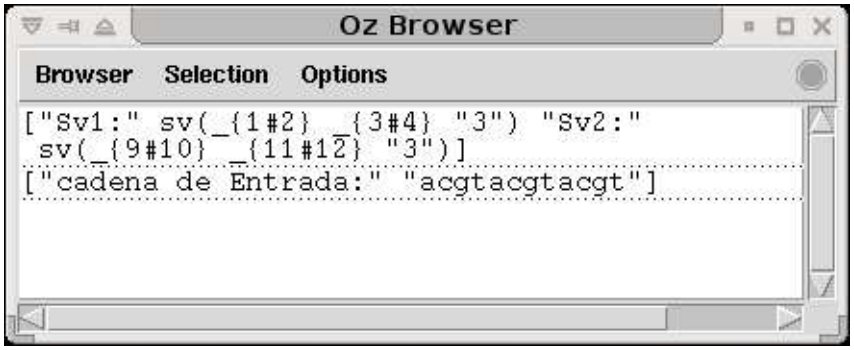


Figura 9.5

#### 9.2.1.4. Ejemplo No. 4.

El siguiente ejemplo tiene como objetivo mostrar cómo se escribe el ejemplo encontrado en el artículo [3].

```
   local

      Cadena   A  B  C  D   in
      Cadena="attagtactatagctagctaactagcgcgctata"   %cadena  de  34   caracteres
       {Sq.sistema _inicio  Cadena }

       %  3<=longitud(A)<=5
       {Sq.longitud  A  3  «>=:«  }
       {Sq.longitud  A  5  «<=:«  }

       %  3<=longitud(B)<=5   se  entiende  que  tiene  la  misma
longitud  de  A  como  consecuencia  de  una  correlación
       {Sq.longitud  B  3  «>=:«  }
       {Sq.longitud  B  5  «<=:«  }

       %  3<=longitud(C)<=5    se  entiende  que  tiene  la  misma
longitud  de  A  como  consecuencia  de  una  correlación
       {Sq.longitud  C  3  «>=:«  }
       {Sq.longitud  C  5  «<=:«  }

       %  3<=longitud(D)<=5     se  entiende  que  tiene  la  misma
longitud  de  B  como  consecuencia  de  una  correlación
       {Sq.longitud  D  3  «>=:«  }
       {Sq.longitud  D  5  «<=:«  }

       %  Restricciones  de  distancia
       {Sq.distancia  A  B  «fin _inicio«  [1  2  3  4] }

       {Sq.distancia  B  C  «fin _inicio«  [1  2  3  4  5  6] }

%                 {Sq.distancia  C  D  'fin _inicio'   {List.number
1  34  1 } }

       %Restriccion  de  contenido
       {Sq.contenido  B   ~2  "t" }

       %restricciones  de  correlación
       {Sq.correlacion  A  C  1  "acgt" }
       {Sq.correlacion  B  A   ~1  "tgca" }
       {Sq.correlacion  D  B  1  "acgt" }
       {Browser.browse  ["cadena  de  Entrada:"  Cadena] }
       {Browser.browse  ["A:"  A  "B"  B  "C"  c  "D"  D] }
   end
```

## 9.3. Resultados

El sistema de restricciones para búsqueda de patrones en cadenas de ADN para Mozart funciona correctamente, fué implementado como un functor de Mozart llamado

secuencias.oz. El functor secuencias.oz contiene procedimientos para declarar variables de cadena, tiene cinco propagadores que fueron implementados en C++ y que son llamados desde Mozart usando como enlace la libreria **'prop_sq/seq.so{native}'** . Se encuentran también implementadas las diferentes restricciones de distancia y algunas correlaciones definidas como las más útiles en el artículo [3].

El sistema de restricciones para búsqueda de patrones en cadenas de ADN para Mozart se puede ampliar facilmente dado que fué creado usando el sistema d restricciones de dominios finitos *FD*; si se encuentra una restricción que pueda modelarse teniendo en cuenta el tipo de variables y dominios, se puede implementar su propagador asociado en C++ y enlazarlo en la libreria, así como es posible también ampliar el tipo de problemas que puede resolver.

En los ejemplos anteriores se muestran distintos problemas en los cuales se muestra que es muy sencillo escribirlos usando el sistema de restricciones para búsqueda de patrones en cadenas de ADN para Mozart; también se puede notar que una de sus características principales es la sencilles y la expresividad para modelar las soluciones de los distintos problemas.

# Parte IV

# Conclusiones

# Capítulo 10

# Conclusiones

- Se cumplieron en su mayoría los objetivos propuestos para el proyecto de grado; se diseñó e implementó un Sistema de Restricciones para Mozart que permite hacer búsqueda de patrones en secuencias de ADN. Solo fueron implementados los algoritmos del artículo *"A Constraint Based Structure Description Language for Biosequences"*[3], por lo tanto no se realizó la comparación de los diferentes algoritmos investigados teniendo en cuenta la cantidad de memoria utilizada, o el tiempo de ejecución. El sistema de restricciones para búsqueda de patrones en secuencias de ADN cuenta con 5 nuevos propagadores que fueron implementados en C++ siguiendo las especificaciones del tutorial de extensión de restricciones [19].

- El sistema de restricciones propuesto permite modelar en Mozart problemas de búsqueda de patrones en secuencias de ADN de forma sencilla en programas que son cortos mostrando así, la *ganancia en expresividad* que aporta el sistema de restricciones. Las restricciones modeladas para el sistema de restricciones, excepto la restricción de correlación, pueden ser modeladas completamente usando el sistema de restricciones de dominios finitos *FD*, se decidió hacer un nuevo sistema de restricciones para que se puedan implementar problemas de búsqueda de patrones en secuecias de ADN en forma más clara, sencilla y corta, además en donde se pueda hacer uso de la restricción de correlación.

- El documento *Constraints propagation in Mozart* de Tobias Müller es apropiado para entender cómo el lenguaje de programación Mozart puede resolver un *CCP*. Se considera de gran importancia conocer a fondo la forma en que fueron modelados y el mecanismo de los procesos de propagación de restricciones y búsqueda, para desarrollar un nuevo sistema de restricciones para Mozart.

- El tutorial de *Extensión de Restricciones* se divide en dos partes, en la primera se encuentra paso a paso el proceso de creación de un nuevo propagador, en la segunda se explica al usuario qué necesita hacer para poder crear un nuevo Sistema de Restricciones. La primera parte del tutorial es muy sencilla y permite que el usuario cree su propagador siguiendo una receta. Para la segunda parte

es esencial tener conocimiento del modelo de *solver basado en propagación de restricciones* de Mozart.

- Existen dos alternativas para crear un sistema de restricciones: crear un sistema de restricciones desde la nada, o utilizar sistemas de restricciones existentes. Al momento de decidir cual de las dos alternativas tomar se debe tener en cuenta el tipo de problema, o problemas que se quieren solucionar. Para esto hay que preguntarse qué tipo de datos va a contener el Dominio de una variable y la forma en que un usuario del sistema de restricciones modelará la solución de un problema específico; qué restricciones son necesarias para resolver un problema, y en especial: si el sistema de restricciones permite resolver distintos tipos de problemas (ie: el sistema de restricciones para dominios finitos *FD*). Adicionalmente se debe tener en cuenta si se desea dejar abierto el sistema de restricciones para futuras contribuciones.

- En el caso del sistema de restricciones propuesto en este trabajo de grado, se consideró adecuado utilizar al sistema de restricciones para dominios finitos teniendo en cuenta que las cadenas en Mozart se modelan como listas de enteros entre [0..,255] y que el conjunto de todos los posibles caracteres presentes en una cadena de ADN se reduce a cuatro letras minusculas [a, c, g, t].

- Mozart es un lenguaje muy adecuado para resolver problemas de tipo combinatorio usando programación con restricciones, ya que permite que el usuario agregue nuevos propagadores o sistemas de restricciones cuando considere necesario.

- En cuanto al área de Bioinformática, en la actualidad existe una amplia gama de problemas relacionados con información biológica y con información útil extraida de la información biológica. Es poco probable encontrar una rama de la ciencia de la computación adecuada para todos los tipos de problemas relacionados con la información biológica. Por esto se sugiere explorar la utilización de *solvers hibridos,* es decir que combinen diferentes técnicas.

- Así como existe una amplia gama de problemas relacionados con información biológica y con información útil extraida de la información biológica, existe una gran cantidad de articulos relacionados con alternativas para solucionar estos problemas. La mayoría de las soluciones propuestas utilizan técnicas de Inteligencia Artificial, y dividen el artículo en dos partes: un componente biológico y uno relacionado con las ciencias de la computación. Se encontraron dos tendencias: una en que el significado biológico de las respuestas obtenidas por los algoritmos es más importante que la técnica utilizada y otra en la que la importancia se centra en el algoritmo en sí, en que sea más eficiente en cuanto a tiempo de ejecución, memoria y cantidad de respuestas obtenidas. Es importante encontrar cierto grado de equilibrio entre ambas tendencias para que los resultados de las investigaciones sean tomados en cuenta por la comunidad científica tanto en el área de biología molecular como en la de las ciencias de la computación.

Personalmente considero que es necesario que un estudiante o profesinal de las ciencias de la computación tenga conocimiento de genética y biológia molecular si desea trabajar en el área de bioinformática, asi como también sea sensible a la importancia del significado biológico de las respuestas de el algoritmo que está desarrollando.

## 10.1. Trabajo Futuro

- Revisar los algorítmos de cada una de las funciones filtro de los propagadores propuestos con el fín de hacer análisis y pruebas de eficiencia; teniendo en cuenta el tiempo de respuesta y la cantidad de memoria ocupada. Es importante poner especial atención en el propagador de correlación y la forma en que se representa una matriz de correlación y por el mecanismo usado para encontrar diagonales en la matriz de correlación, como resultado se espera obtener una nueva propuesta para representar la matriz de correlación.
- Traducir al inglés la documentación del sistema de restricciones.
- Escribir un artículo sobre el diseño e implementación del sistema de restricciones para búsqueda de patrones en secuencias de ADN para Mozart.

# Parte V

# Anexos

# Capítulo 11

# Manual de instalación del software

## Resumen

A continuación se encuentra una descripción de los archivos que forman parte del Sistema de Restricciones para Búsqueda de patrones en cadenas de ADN para Mozart, así como también un instructivo para ejecutar los archivos de pruebas y ejemplos.

## 11.1. Archivos contenidos

### 11.1.1. Documento

En la carpeta Documento se encuentran cada uno de los archivos que conforman el documento del proyecto de grado. Se utilizó lyx como editor. El archivo documento.ps contiene completo el documento del proyecto de grado. Existe una carpeta llamada oz en donde se encuentran los archivos necesarios para que el código Mozart que aparece en el documento conserve su formato.

### 11.1.2. Documentación del Sistema de Restricciones

La documentación del sistema de restricciones se hizo usando la herramienta Ozh, los archivos de extención html se encuentran en la carpeta out.

### 11.1.3. Archivos de Pruebas y ejemplos

En la carpeta pruebas se encuentran los siguientes archivos:

1. longitud.oz
2. contenido.oz

3. posicion.oz
4. distancia.oz
5. ejemplo_gilbert.oz

Estos archivos corresponden a las pruebas individuales que se realizo a cada uno de los propagadores.

El archivo ejemplo_Gilbert.oz contiene el ejemplo que aparece en el artículo [**?**]

### 11.1.4. Sistema de restricciones

En la carpeta prop_sq se encuentran los siguientes archivos:

1. seq.cc
2. content.cc
3. correla.cc
4. distancia.cc
5. long2.cc
6. posicion.cc

Los archivos enumerados son los que comonen los propagadores del nuevo sistema de restricciones.

En la carpeta principal está el archivo secuencias.oz, este es el functor que implementa el nuevo sistema de restricciones en Mozart.

## 11.2. Ejecución de pruebas y ejemplos

Para ejecutar las pruebas o el ejemplo debe primero compilar todo el software. Para compilar el software debe:

- Verificar que tiene instalado Mozart en su pc
- Ejecutar el makefile de la carpeta pruebas y el makefile de la carpeta principal.

Si no aparecen mensajes de error, puede proceder a ejecutar las pruebas y el ejemplo.

Ejemplo de ejecución de un archivo de prueba:

angievig@cactus:~/Tesis/pruebas$ ./longitud

A continuación encontrará el siguiente mensaje en la consola:

angievig@cactus:~/Tesis/pruebas$ ./contenido cargando propagadores: longitud, posición, distancia, contenido y correlacion

y aparecen dos ventanas: el Browser y el Explorer de OZ en las cuales podrá encontrar los resultados.

# Bibliografía


[1] Rolf Backofen, David Gilbert. Bioinformatics and Constraints, *Journal of Constraints*, 2001

[2] DOE (U.S. Department of Energy) Human Genome Program. *DOE Human Genome* 1991-92 *Program Report*, Junio 1992.

[3] Ingvar Eidhammer, David Gilbert, Inge Jonassen y Madu Ratnayake, A Constraint Based Structure Description Language for Biosequences, *Constraints Journal: especial issue on constraints and Bioinformatics*, 1999

[4] Alviz Brazma, David Gilbert, *A pattern language for molecular biology*, 1995

[5] David Gilbert, David Westhead, Juris Viksna, Janet Thornton, *Topology-based prot ein struvture comparision using a pattern discovery technique*, 2000

[6] David Gilbert, David Westhead, Juris Viksna, Janet Thornton, *A computer system to perform structure comparison using TOPS representations of protein structure*, 2001

[7] David Gilbert, Ingvar Eidhammer, Inge Jonassen, *StructWeb: biosequence structure searching on the web using clp(FD)*, 1997

[8] David Gilbert, David Westhead, Nozomi Nagano, Janet Thornton, *Motif-based search ing in TOPS protein topology data bases*

[9] Jaques van Helden, David Gilbert, *Interactive visualisation and exploration of biological data*.

[10] Juris Viksna, David Gilbert, *Pattern discovery methods forprotein topology diagrams*.

[11] http://www.infor.uva.es/~cesargf/proyectos/memorias/reconocimiento/tema9.pdf, *Modelos de markov.*

[12] http://www.umass.edu/molvis/workshop/homolmod.htm, *Homology modeling for beginners*, 2001

[13] http://www.ii.uib.no/~inge/patterns.html, Patterns in biosequences, Inge Jonassen.

[14] GRUPO Avispa, Metodología de desarrollo - CRISOL, 2003

[15] Gerardo M. Sarria M., Restricciones, Modelando con restricciones de dominio finito, notas del curso Programación con Restricciones, http:// malpelo.univalle.edu.co/~cursos/restricciones/notes, 2002

[16] Ingvar Eidhammer, David Gilbert, Inge Jonassen y Madu Ratnayake, A Constraint Based Structure Description Language for Biosequences, *Constraints Journal: especial issue on constraints and Bioinformatics*, 1999

[17] Alviz Brazma, David Gilbert, *A pattern language for molecular biology*, 1995

[18] Müller Tobias, Constraint Propagation in Mozart, 2001

[19] Müller Tobias, The Mozart Constraint Extensions Tutorial, 2004