Índice General

Introducción **2**

1 Marco teórico **4**

1.1 Fundamentos de la Programación Lógica. 4

1.2 Conjuntos parcialmente ordenados. 8

1.3 Programación de orden parcial 12

1.3.1 Sintaxis 12

1.3.2 Semántica declarativa 13

2 Optimización del espacio de búsqueda **19**

2.1 Semántica operacional 20

2.2 Formalización de la semántica operacional 25

2.3 Extensiones 29

2.3.1 Corte Alpha-Beta 29

2.3.2 Corte por lemas 32

Conclusiones y trabajo a futuro **34**

Introducción

Una de las preocupaciones más tempranas de la computación de los años cincuenta, fue la posibilidad de hacer programas que llevaran a cabo la demostración automática de teoremas. Así empezaron los primeros trabajos de la inteligencia artificial que más de veinte años después dieron lugar al primer lenguaje de programación, que contempla como parte del intérprete los mecanismos de interés necesarios para la demostración automática. Este primer lenguaje está basado en el formalismo matemático de la Lógica de Primer Orden y se ha denominado Prolog.

Actualmente, la programación lógica ha despertado un creciente interés que va mucho más allá del campo de la Inteligencia Artificial y sus aplicaciones. La programación lógica se ha estado usando en diversas áreas de las ciencias de la computación, como es la optimización de problemas. La ventaja de la programación lógica radica en su naturalidad para expresar problemas y su poder de deducción. Sin embargo, Prolog no resuelve todos los problemas inherentes a este paradigma. Un ejemplo claro, son los problemas de optimización con los llamados problemas de satisfacción de restricciones. Ante estas limitantes han surgido diversas extensiones a la programación lógica como son los lenguajes de programación por restricciones. En este tipo de extensiones lo que se hace principalmente es desarrollar sistemas más especializados a una aplicación específica, perdiendo la idea de paradigma de propósito general. Otra alternativa para hacer más robusta a la programación lógica es la combinación con otros paradigmas de programación, por ejemplo; con la programación funcional. Con este tipo de mejoras han surgido lenguajes híbridos que tratan de aprovechar las ventajas que ofrecen cada uno de sus enfoques que lo conforman.

En el caso particular de esta tesis se hace uso de la programación de orden parcial, que principalmente conforma un lenguaje híbrido. Que incorpora aspectos de programación funcional y aspectos de programación lógica.

La sintaxis de la programación de orden parcial se basa en la sintaxis de las cláusulas de orden parcial, las cuales son una generalización de cláusulas de subconjuntos las cuales conforman un lenguaje de programación con conjuntos [3, 4]. Las cláusulas de orden parcial nos permiten una elegante y eficiente especificación en los programas y si además las combinamos con cláusulas lógicas se puede obtener un lenguaje de un alto poder declarativo; un ejemplo de este tipo de lenguajes es el presentado en [14]. Como todo lenguaje formal la programación de orden parcial cuenta con una semántica declarativa y una semántica operacional. La semántica declarativa está basada en el concepto de modelo, la cual sigue la intuición de teoría de punto fijo. La semántica operacional combina los conceptos de reducción de una

consulta de una forma top-dowm con el uso de memo-tabla. El uso de memo-tabla tradicionalmente ha sido usado por lenguajes funcionales tradicionales para determinar de forma dinámica subexpresiones.

En la presente tesis se extienden trabajos recientes en la exploración del uso de corte declarativo [6, 10, 11]. Por corte declarativo debemos entender que el programador puede dar información de forma declarativa a cerca de ciertas funciones en el programa de una forma meramente adicional. Usando esta información, nuestro propósito es proveer una semántica operacional que proporcione una ejecución más eficiente de una programa. Esencialmente requerimos que los dominios de las funciones en los programas sean totalmente ordenados.

El corte declarativo es útil para resolver problemas que requieren una búsqueda sobre un dominio de datos grande que sea totalmente ordenado como son: las operaciones en base de datos, en algoritmos de aproximación, problemas de optimización entre otros.

La estructura de la tesis se divide en sólo dos capítulos. En el primer capítulo se presenta el marco teórico, en donde se definen algunos conceptos básicos de la programación lógica necesarios para la mejor comprensión de la tesis. Además de algunas propiedades de conjuntos parcialmente ordenados. Finalmente, en la última sección se introduce la sintaxis y la semántica declarativa de la programación de orden parcial.

En el segundo capítulo se muestran los resultados de la tesis, que consisten principalmente en la presentación y la formalización de la semántica operacional con corte; la formalización consiste principalmente en probar *Soundness* y *Completeness* de la semántica. En la última sección de este capítulo se muestran algunos tipos de corte que se consideran como posibles extensiones futuras en la semántica operacional.

Capítulo 1

Marco teórico

En este capítulo se presentan las bases conceptuales de la programación lógica además de algunas propiedades de conjuntos parcialmente ordenados y de la programación de orden parcial.

1.1 Fundamentos de la Programación Lógica.

Los programas lógicos son un subconjunto de la lógica de primer orden. Para definir los programas lógicos, se dará un breve panorama sobre la sintaxis y la semántica de la lógica de primer orden. Además se presentarán algunos conceptos básicos, los cuales son necesarios para los fundamentos teóricos de la programación lógica.

La lógica de primer orden tiene dos conceptos fundamentales: la sintaxis y la semántica. El aspecto sintáctico son las fórmulas bien formadas admitidas por la gramática de un lenguaje formal. La semántica se refiere al significado relacionado a las fórmulas bien formadas.

La teoría de primer orden consiste de un alfabeto, un lenguaje de primer orden, un conjunto de axiomas y un conjunto de reglas de inferencia. El lenguaje de primer orden consiste de fórmulas bien formadas. Los axiomas son un subconjunto designado de fórmulas bien formadas. Los axiomas y las reglas de inferencia son usadas para derivar los teoremas de la teoría.

La sintaxis de la lógica de primer orden está basada en un alfabeto, definido de la siguiente manera:

Definición 1.1.1 ([5]) *Un alfabeto consiste de siete clases de símbolos:*

1. *Variables, son denotadas por las letras mayúsculas U, V, W, X, Y y Z.*
2. *Símbolos de funciones, denotadas por una secuencia finita de las letras minúsculas f, g, h.*

3. *Símbolos de predicados, denotados por una secuencia finita de letras minúsculas como por ejemplo p, q, r, path, top.*

4. *Constantes proposicionales, Verdadero y Falso.*

5. *Conectivos lógicos:* $\sim$*(negación),* $\wedge$ *(conjunción),* $\vee$ *(disyunción),* $\rightarrow$ *(implicación) y* $\leftrightarrow$ *(equivalencia).*

6. *Cuantificadores:* $\exists$*(existencia o cuantificador existencial) y* $\forall$ *(para todo o cuantificador universal).*

7. *Símbolos de puntuación, "(", ")" y ",".*

La jerarquía de precedencia es la siguiente:

$$\sim, \forall, \exists$$
$$\vee$$
$$\wedge$$
$$\rightarrow, \leftrightarrow$$

Las clases de símbolos de la 4 a la 7 en la definición anterior son las mismas para cada lenguaje mientras que, los símbolos de funciones y los símbolos de predicados únicamente determinan el alfabeto de cada lenguaje. Hay un número natural asociado con cada símbolo de función y símbolo de predicado, llamado aridad[1] del símbolo. Si el símbolo tiene aridad n, entonces este es llamado un símbolo de aridad n, y si es de aridad 0 es llamado símbolo constante. Las constantes son denotadas por una secuencia finita de letras minúsculas a, b, c, d. Un símbolo de predicado de aridad 0 es llamado símbolo proposicional.

Otras definiciones importantes en la programación lógica son:

Definición 1.1.2 ([5]) *Un término es definido inductivamente por:*

- *Una variable es un término.*
- *Una constante es un término.*
- *Sea f un símbolo de función con aridad n y sean* $t_1,\ldots,t_n$ *términos, entonces* $f(t_1,..,t_n)$ *es un término.*

Definición 1.1.3 ([5]) *Una fórmula bien formada es definida inductivamente por :*

[1]Aridad : Número de argumentos de un símbolo de función o símbolo de predicado.

- *Si p es un símbolo de predicado con aridad n y t_1,..,t_n términos, entonces p(t_1,...,t_n) es una fórmula (llamada fórmula atómica, o simplemente átomo).*
- *Verdadero y falso son fórmulas.*
- *Si F y G son fórmulas, entonces ($\sim$F), (F $\wedge$ G), (F $\vee$ G), (F $\rightarrow$ G) y (F $\leftrightarrow$G) son fórmulas.*
- *Si F es una fórmula y x es una variable, entonces ($\forall$x F) y ($\exists$x F) son fórmulas.*

A menudo será conveniente escribir (F $\rightarrow$G) como (G $\leftarrow$ F).

Un ejemplo de una fórmula bien formada es: $\forall$x $\exists$y (p(x,y) $\rightarrow$q(x))

Definición 1.1.4 ([5]) *El lenguaje de primer orden dado por un alfabeto, consiste de un conjunto de todas las fórmulas construídas de los símbolos del alfabeto.*

Definición 1.1.5 ([5]) *Una literal es un átomo (A) o la negación de un átomo ($\sim$A). Una literal positiva es un átomo. Una literal negativa es la negación de un átomo.*

Definición 1.1.6 ([5]) *Un término ground es un término que no contiene variables. Similarmente, un átomo ground es un átomo que no contiene variables.*

Definición 1.1.7 ([5]) *Una fórmula cerrada es una fórmula que no tiene ocurrencias libres de alguna variable.*

La semántica de una teoría de primer orden provee el significado de la teoría basada en alguna interpretación. Las interpretaciones proveen un significado específico a los símbolos del lenguaje y son usados para proveer significado al conjunto de fórmulas bien formadas.

Algunos conceptos importantes en la teoría de semánticas son pre-interpretación, interpretación y modelo los cuales se definen a continuación.

Definición 1.1.8 ([5]) *Una pre-interpretación de un lenguaje L de primer orden consiste en lo siguiente:*

- *Un conjunto D no-vacío, llamado dominio de la pre-interpretación.*
- *Para cada constante en L, se le asigna un elemento en D.*
- *Para cada símbolo de función con aridad n en L, se le asigna un mapeo de D^n a D.*

Definición 1.1.9 ([5]) *Una interpretación I de un lenguaje de primer orden L, consiste de una pre-interpretación J con dominio D en L junto con lo siguiente: para cada símbolo de predicado de ariadad n en L, se le asocia un mapeo de D^n en {Verdadero, Falso} (o equivalente, a una relación sobre D^n.)*

Definición 1.1.10 ([5]) *Sea I una interpretación para un lenguaje de primer orden L y W una fórmula en L.*

Se dice que W es satisfactible en I si $\exists(W)$ *es verdadero con respecto a I.*

Se dice que W es válida en I si $\forall(W)$ *es verdadero con respecto a I.*

Se dice que W es insatisfactible en I si $\exists$ *(W) es falso con respecto a I.*

Se dice que W es no válida en I si $\forall(W)$ *es falso con respecto a I.*

Definición 1.1.11 ([5]) *Sea I una interpretación de un lenguaje de primer orden L y F una fórmula cerrada de L. Entonces I es un modelo para F, si F evalúa verdadero en I.*

En otras palabras, *un modelo M* para una teoría T (conjunto de fórmulas) es una interpretación I de T tal que, toda formula $f \in T$ evalúa cierto en I.

Definición 1.1.12 ([5]) *Sea T una teoría y* α *una fórmula cerrada, entonces definimos que* α *es una consecuencia lógica de T, si para toda interpretación I tal que I es modelo de T, entonces I también es modelo de* α. *Se representa como:* $T \models \alpha$.

Ejemplo 1.1 *Considere los siguientes predicados:*

P(x) : donde x es hombre.

Q(x) : donde x es mortal.

a : Sócrates.

Sea T el siguiente conjunto de fórmulas:

$P(a)$

$\forall\, x(P(x) \rightarrow Q(x))$

En este modelo es cierto que Sócrates es mortal, por lo tanto $T \models Q(a)$.

La sustitución de variables por términos en una fórmula, juega un papel importante en la teoría de programas lógicos.

Definición 1.1.13 ([5]) *Una sustitución* θ *es un conjunto finito de la forma* $\{v_1 \leftarrow t_1, ..., v_n \leftarrow t_n\}$, *donde cada* v_i *es una variable, cada* t_i *es un término distinto de* v_i *y las variables* $v_1,...,v_n$ *son distintas, donde* θ *es llamada sustitución ground si las* t_i *son términos ground.*

El proceso de encontrar una sustitución es llamado unificación para los átomos. La unificación está basada en la sustitución, además de tener una gran importancia en la teoría de la programación lógica. Algunos conceptos importantes de la unificación son:

Definición 1.1.14 ([5]) *Dos átomos (o expresiones) A y B son unificables si existe una sustitución θ tal que $A\theta = B\theta$. La sustitución θ es llamada unificador para A y para B. Este es llamado un unificador más general (mgu) si para cada unificador η para A y para B existe una sustitución γ tales que $\eta = \theta\gamma$.*

Definición 1.1.15 *Match es un caso especial de la unificación, donde uno de los átomos es ground (no contiene variable) y el otro no.*

Un ejemplo de *match* es el siguiente:

Ejemplo 1.2 *Consideremos los siguientes átomos P(a, b) y P(x, y), es claro que el primer átomo es ground y el otro no, entonces para que unifiquen se tiene el unificador siguiente:*

$$\theta \leftarrow \{x \leftarrow a, y \leftarrow b\}$$

1.2 Conjuntos parcialmente ordenados.

La teoría de relaciones es fundamental en la comprensión de la programación de orden parcial presentada mas adelante. En esta sección se presentarán los conceptos fundamentales de la teoría de relaciones y retículos[2].

Frecuentemente queremos comparar o contrastar varios elementos de un conjunto, quizá para acomodarlos en un orden apropiado para agrupar aquéllos con propiedades similares. La estructura matemática para describir este tipo de organización de conjuntos es la teoría de relaciones.

Para ilustrar lo anterior, supongamos que hay una manera de comparar los elementos de un conjunto, de modo que siempre que se den dos elementos del conjunto sabemos como compararlos, veamos el siguiente ejemplo.

Ejemplo 1.3

1. *Estamos acostumbrados a comparar números reales. Por ejemplo: 3 es menor que 5, -1 es menor que 4 y -1 es mayor que -3. Comparamos dos números diciendo cual es menor y cuál es mayor.*
2. *Si el conjunto S tiene índices en P o N de manera que elementos distintos tienen índices distintos, entonces podemos comparar dos elementos de S observando cuál de ellos tiene menor índice.*

Si los elementos del conjunto se pueden comparar como en los incisos 1) y 2), se dice que el conjunto está *ordenado*, y a la especificación de cómo comparar un elemento de otro se le llama *relación de orden* en el conjunto. Para decir algo útil sobre conjuntos ordenados, debemos dar una definición más precisa. Primero hay que notar que, en muchos conjuntos que surgen naturalmente sabemos como comparar algunos elementos con otros, pero también hay parejas que no son comparables veáse el siguiente ejemplo.

[2]Del inglés lattices.

Ejemplo 1.4 *1. Podemos comparar dos números en {1,2,3,...,73} en caso de que uno sea factor de otro. Entonces podemos comparar 6 y 72 y también 3 y 6 sin embargo, no podemos comparar 6 y 8 ya que ni 8 es factor de 6 ni 6 es factor de 8.*

2. Podemos comparar dos subconjuntos de un conjunto S, si uno está contenido en otro. Si S tiene más de un elemento, entonces tiene subconjuntos incomparables. Por ejemplo, si s_1 y s_2 están en S y $s_1 \neq s_2$, entonces los conjuntos $\{s_1\}$ y $\{s_2\}$ no son comparables.

Los conjuntos donde las relaciones de comparación permite la posibilidad de que dos elementos no sean comparables como los del ejemplo 1.4, se llaman *conjuntos parcialmente ordenados*, y forman una clase muy importante que definiremos ahora.

Definición 1.2.1 ([15]) *Supongamos que nos dan una relación en un conjunto S de manera que para cada par $\langle x, y\rangle$ en $S \times S$ sabemos cuando x está relacionada con y. Escribimos $x \preceq y$ si y sólo si x está relacionada con y. La relación se llama un orden parcial en S si satisface lo siguiente:*

(R) $s \preceq s$ para toda s en S;

(AS) $s \preceq t$ y $t \preceq s$ implica $s = t$;

(T) $s \preceq t$ y $t \preceq u$ implica $s \preceq u$.

Las condiciones (R), (AS) y (T) son las propiedades *reflexiva, antisimétrica* y *transitiva* respectivamente, y se dice que $\preceq$ es reflexiva, antisimétrica o transitiva si las satisface. Si $\preceq$ es un orden parcial en S, se dice que $(S, \preceq)$ es un conjunto parcialmente ordenado, o para abreviar, un *copo*. Se utilizará la notación " $\preceq$ "como símbolo para un orden parcial en general.

En el ejemplo 1.4 las relaciones que se presentan se sobreentiende que son "*es un factor de*" y "*es subconjunto de*". También podríamos haber considerado las relaciones "*es un múltiplo de*" y "*contiene*", ya que estas relaciones dan la misma información comparativa que las elegidas.

Definición 1.2.2 ([15]) *Dado un orden parcial $\preceq$ en un conjunto S, podemos definir la relación $\prec$ en S como :*

$x \prec y$ si y sólo si $x \preceq y$ y $x \neq y$

Por ejemplo: si $\preceq$ es la inclusión $\subseteq$, entonces $A \prec B$ significa que A es un subconjunto propio de B, es decir $A \subset B$. La relación $\prec$ satisface:

$(AR) s \prec s$ es falso para toda s en S;

(T) $s \prec t$ y $t \prec u$ implica $s \prec u$.

Una relación que satisface (AR) es *antireflexiva*. Una relación que satisface (AR) y (T) es un *orden parcial estricto*. Cada orden parcial en S genera un orden parcial estricto y recíprocamente si $\prec$ es un orden parcial estricto en S, entonces la relación $\preceq$ definida por:

$x \preceq y$ si y sólo si $x \prec y$ o $x = y$

es un orden parcial en S. Dependiendo del problema en particular de que se trate, se trabaja con el orden parcial o con el orden parcial estricto asociado. Generalmente se trabajará con orden parcial.

Definición 1.2.3 ([15]) *Si $(P, \preceq)$ es un copo, decimos que $x \in P$ es máximal si no existen $y \in P$ tal que $x < y$ y llamamos a x minimal si no existe $y \in P$ tal que $y < x$.*

Si S es un subconjunto de un copo $(P, \preceq)$, puede suceder que S tenga un elemento M tal que $s \preceq M$ para toda s en S. Un elemento M con esta propiedad se llama *el mayor elemento* del S o *el máximo* de S y se denota $máx(S)$. De manera similar, si S tiene un elemento m tal que $m \preceq s$ para toda s en S, entonces m es *el emento menor* del conjunto o el *mínimo* de S y se denota por $mín(S)$.

A pesar de que un subconjunto S de un copo $(P, \preceq)$ no tenga elemento máximo, puede existir un elemento x en P tal que $s \preceq x$ para toda s en S. Un elemento que satisfaga esto se llama *cota superior* para S en P. Si x es una cota superior de S en P tal que $x \preceq y$ para toda y cota superior de S en P, entonces x es llamada *mínima cota superior* (least upper bound) o *supremo* de S en P y la denotamos $lub(S) = x$. De manera similar un elemento z en P tal que $z \preceq s$ para toda s en S es una *cota inferior* para S en P. Una cota inferior z tal que $w \preceq z$ para toda cota inferior w de S en P se llama *máxima cota inferior* (greatest lower bound) o *ínfimo* de S en P y la denotamos $glb(S) = z$. Por la propiedad *antisimétria* (AS), un subconjunto de P no puede tener dos mínimas cotas superiores distintas o dos máximas cotas inferiores distintas.

Definición 1.2.4 ([2]) *Sea P un conjunto parcialmente ordenado no vacío.*

(i) *Si para todo $x, y \in P$ existen el $lub\{x, y\}$ y $glb\{x, y\}$ entonces P es llamado retículo.*

(ii) *Si para todo $S \subset P$ existen el lub S y glb S entonces P es llamado retículo completo.*

Algunas propiedades del *ínfimo* (glb) y *supremo* (lub) de un conjunto ordenado y de un *retículo* son las siguientes.

Lema 1.2.1 ([2]) *Sea P un conjunto ordenado y $S, T \subseteq P$, asumimos que lub(S), lub(T), glb(S) y glb(T) existen en P. Entonces:*

(i) Para todo $s \in S$, $s \leq lub(S)$ y $s \geq glb(S)$.

(ii) Sea $x \in P$; entonces $x \leq glb(S)$ si y sólo si $x \leq s$ para toda $s \in S$.

(iii) Sea $x \in P$; entonces $x \geq lub(S)$ si y sólo si $x \geq s$ para toda $s \in S$.

(iv) $lub(S) \leq glb(T)$ si y sólo si $s \leq t$ pata toda $s \in S$ y toda $t \in T$.

(v) Si $S \subseteq T$, entonces $lub(S) \leq lub(T)$ y $glb(S) \geq glb(T)$.

Lema 1.2.2 ([2]) *Sea P un retículo y $S, T \in P$, asumimos que $lub(S)$, $lub(T)$, $glb(S)$ y $glb(T)$ existen en P. Entonces $lub(S \cup T) = lub\{lub(S),\ lub(T)\}$ y $glb(S \cup T) = \{glb(S), glb(T)\}$*

Lema 1.2.3 ([2]) *Sea P un retículo y $a, b \in P$. Entonces las siguientes relaciones son equivalentes:*

(i) $a \leq b$.

(ii)$lub(a, b) = b$.

(iii) $glb(a, b) = a$.

Un caso especial de los copos es la existencia de un orden total o lineal definido acontinuación.

Definición 1.2.5 ([15]) *Sea S un conjunto, un orden parcial se llama orden total o lineal si para cada elección de s y t en S, $s \preceq t$ o $t \preceq s$. Entonces llamaremos a S un conjunto totalmente ordenado o conjunto linealmente ordenado.*

Ejemplo 1.5 *Ejemplos de conjuntos totalmente ordenados:*

a) El conjunto $\mathbb{R}$ (números reales) con el orden usual $\leq$ es un conjunto totalmente ordenado.

b) Las listas de nombres en un directorio telefónico o de palabras en un diccionario, son conjuntos totalmente ordenados si definimos $w_1 \leq w_2$ como $w_1 = w_2$ o w_1 aparece antes de w_2.

Todo subcopo de un conjunto totalmente ordenado a su vez es un conjunto totalmente ordenado. Por ejemplo los copos $(\mathbb{Z}, \leq)$ y $(\mathbb{Q}, \leq)$ son subcopos de $(\mathbb{R}, \leq)$ y están totalmente ordenados por el orden que heredan de $\mathbb{R}$.

Definición 1.2.6 *Una función f es monótona si $X \leq Y$ implica que $f(X) \leq f(Y)$. Una función f es inflationary si $X \leq f(X)$. Una función f es deflationary si $f(x) \leq x$.*

1.3 Programación de orden parcial

En esta sección se presentarán algunos conceptos importantes de la programación de orden parcial como es la sintaxis de los programas y algunos conceptos de la semántica declarativa.

1.3.1 Sintaxis

Un programa P es una pareja $< PO, PA >$ donde PO es un conjunto de *cláusulas de orden parcial* y DA es un conjunto de *cláusulas datalog*. Las cláusualas de orden parcial son de la forma:

$$f(terms) \geq expression \text{ :- } p_1(terms), \ldots, p_k(terms)$$
$$f(terms) \leq expression \text{ :- } p_1(terms), \ldots, p_k(terms)$$

donde cada variable en *expression* también ocurre en el cuerpo de las cláusulas, $p_1(terms)$, ..., $p_k(terms)$ donde $k \geq 0$ y cada predicado p_i es definido por un *programa datalog* o en otra palabras por un conjunto de *cláusulas datalog*. Cuando $k = 0$, tendremos clásulas incondicionales de la forma:

$$f(terms) \geq expression$$
$$f(terms) \leq expression$$

La sintaxis de *terms* y *expression* está dada por la siguiente gramática:

term ::= *variable* | *constant* | *c*(*terms*)
terms ::= *term* | *term* , *terms*
expression ::= *term* | *c*(*exprs*) | *f*(*exprs*)
exprs ::= *expression* | *expression* , *exprs*

El símbolo c es un estándar para cualquier constructor, mientras que f es un estándar para símbolos no constructores también llamados símbolos definidos por el usuario. La sintaxis de las *cláusulas datalog* está determinada por la siguiente gramática:

rule ::= *atom* | *atom* :- *body*
body ::= *literal* | *literal*, *body*
literal ::= ∼ *atom* | *atom*
atom ::= *p*(*terms*)

En la programación de orden parcial se tiene la convención de iniciar los símbolos de constantes con letras minúsculas y los símbolos de las variables con letras mayúsculas. Cabe destacar que las cláusulas de orden parcial costituyen un paradigma de programación funcional y por consiguiente todas las llamadas a las funciones se realizan con argumentos ground. También se hace uso del término *consulta*, con esto nos referimos a un llamado a una función. La sintaxis de una consulta es de la siguiente forma:

$$f(ground - term)$$

En el caso general, múltiples cláusulas de orden parcial son usadas para definir a una función. Para ilustrar un poco la forma en que se modela un programa en este paradigma veamos los siguientes ejemplos:

Ejemplo 1.6 (Distancia más corta en un grafo) *La formulación del problema de encontrar la distancia más corta en un grafo es una de las más elegantes e ilustrativas del uso de cláusulas de orden parcial.*

```
distancia(X,Y) ≤ C1 :- arco(X,Y,C1).
distancia(X,Y) ≤ C2 + distancia(Z,Y) :- arco(X,Z,C2).
```

La relación `arco(X,Y,C1)` *significa que existe un arco del vértice* `X` *al vértice* `Y` *con una distancia* `C1`. *Observe que la función* `distancia(X,Y)` *está definida mediante la relación* $\leq$. *Esto quiere decir informalmente que la* `distancia(X,Y) = mínimo{C1, C2 + distancia(Z,Y)}`.

Otro interesante ejemplo, es la formulación del problema de la mochila.

Ejemplo 1.7 (Problema de la mochila) *El planteamiento del problema de la mochila es el siguiente: Se tienen n objetos con las ganancias* p_i *y los costos* w_i, *para* $1 \leq i \leq n$, *y una capacidad de almacenamiento de* M. *Se quiere maximizar* $\sum x_i p_i$, *sujeto a* $\sum x_i w_i \leq M$, *y* $x_i \in \{0,1\}$, $1 \leq i \leq n$. *El valor de uno en* x_i *indica que el objeto i se mete a la mochila teniendo una ganancia de* p_i, *y la capacidad de la mochila sera* $M - w_i$ *y el valor de cero en* x_i *que no se mete a la mochila el objeto i. La formulación de este problema con cláuslas de orden parcial es la siguiente:*

$\texttt{kn}_{01}(\texttt{0},\texttt{M}) \geq \texttt{0}.$
$\texttt{kn}_{01}(\texttt{I},\texttt{M}) \geq \texttt{kn}_{01}(\texttt{I}-1,\texttt{M}) :- \texttt{I} \geq 1.$
$\texttt{kn}_{01}(\texttt{I},\texttt{M}) \geq \texttt{kn}_{01}(\texttt{I}-1,\texttt{M}-\texttt{c(I)}) + \texttt{p(I)}$ `:- I` $\geq$ `1, c(I)` $\leq$ `M.`

La función `c(I)` *define el costo del objeto* `I`, *la función* `p(I)` *define la ganacia del objeto* `I`. *Note que la función* $\texttt{kn}_{01}$ *está definida mediante la relación* $\geq$ *que informalmente nos indica que* $\texttt{kn}_{01}$`(I,M) = máximo { 0,` $\texttt{kn}_{01}$`(I-1,M),` $\texttt{kn}_{01}$`(I-1, M - c(I)) + p(I) }`.

1.3.2 Semántica declarativa

Una parte importante en todo lenguaje formal es su semántica declarativa, es por esto que se presentarán algunos conceptos de ésta.

Para la presentación de la semántica declarativa, primero definiremos la noción de *reducir un programa con respecto a su sección de cláusulas datalog.* La reducción consta de dos pasos. El primero, consiste en reducir el programa datalog en una base de datos extencional (EDB)[3]. Para esto es necesario que el programa datalog sea estratificado para asegurar que el proceso de reducción termina en una EDB. En este primer paso, se hace uso del algoritmo

[3]Una *base de datos extencional (EDB)* es un conjunto de átomos ground.

presentado en [1]. Este proceso tiene un comportamiento polinomial con respecto al tamaño de la EDB. Otro posible algoritmo para reducir el programa datalog es el presentado en [9]. En el segundo paso, se reduce el programa original con respecto a la EDB obtenida en el primer paso.

Cabe hacer mención que los algoritmos a utilizar en el primer paso se basan principalmente en reglas de trasformación de programas lógicos como las que se definen en [13]; esto es gracias a que su tiempo de cálculo es polinomial. Una descripción de cómo utilizar reglas de transformación en programas datalog es presentada en [8]. De hecho existe una gran variedad de aplicaciones alrededor de la trasformación de programas lógicos, como las que se discuten en [12].

Definición 1.3.1 *La reducción de un programa P con respecto a una EDB E, es denotada por R(P) y definida como sigue:*

$R(P) = \{(g(t) \leq e)\theta \mid g(t) \leq e$ `:-` $e_1, .., e_n \in P, \theta$ *es una sustitución tal que cada* $(e_i\theta) \in E\}$

Ejemplo 1.8 *Consideremos nuevamente el programa del ejemplo 1.6, con la siguiente EDB:*

```
arco(a,b,2).
arco(a,c,3).
```

Entonces reduciendo la función `distancia` *con respecto al predicado* `arco` *obtenemos el siguiente programa:*

```
distancia(a,b) ≤ 2.
distancia(a,c) ≤ 3.
distancia(a,Y) ≤ 2 + distancia(b,Y).
distancia(a,Y) ≤ 3 + distancia(c,Y).
```

Ahora se definirá la clase de programas legales para la semántica. En la siguiente definición se asume que el programa ya fue reducido con respecto a su EDB.

Definición 1.3.2 *Un programa P es generalmente estratificado si existe una función, nivel* : $F \rightarrow N$, *que mapea desde F que es el conjunto de símbolos de las funciones definidas por el usuario (no constructores) en P a (subconjunto finito de) los números naturales N tal que:*

(i) Toda cláusula de la forma $f(term_1) \geq term_2$ es válida. Toda cláusula de esta forma es llamada cláusula S-S (S-S es una abreviación de stronly-stratified).

(ii) Para cláusulas de la forma

$f(term) \geq g(expre)$

donde f y g son funciones definidas por el usuario, $nivel(f)$ es mayor o igual al $nivel(g)$ y $nivel(f)$ es mayor que $nivel(h)$, donde h es un símbolo de función definida por el usuario que ocurre en expre. También este tipo de cláusulas son llamadas cláusulas S-S.

(iii) Para cláusulas de la forma

$f(term) \geq m(g(expre))$

donde m es una función monótona, nivel(f) es mayor que nivel(m), nivel(f) es mayor o igual a nivel(g) y nivel(f) es mayor que nivel(h), donde h es un símbolo de función definida por el usuario que ocurre en expre. Las cláusulas de esta forma son llamadas cláusulas G-S (G-S es una abreviación de general-stratified).

(iv) Toda cláusula de otra forma es no válida.

Aunque un programa puede tener diferentes niveles de mapeo, se selecciona un conjunto imagen que parta del número 1. En la anterior definición, note que las funciones f y g no son necesariamente diferentes. También la dependencia de funciones no monótonas ocurre sólo con funciones de bajo nivel. Observe que se puede tener mayor libertad en la definición anterior. Partiendo de la idea que la composición de funciones monótonas es monótona entonces la función m puede ser reemplazada por una composición de funciones monótonas.

Para la presentación de la semántica operacional en el siguiente capítulo se asume que el programa ya fue reducido con respecto a su sección de cláusulas datalog y su EDB y además que toda cláusula está en un *formato plano*. Para la presentación del formato plano veamos primero un ejemplo y posteriormente daremos la definición formal.

Ejemplo 1.9 *Asumimos que f, g, h y k son funciones definidas por el usuario y las restantes funciones son constructores, el formato plano de la cláusula.*

$$f(c(X,Y)) \leq c1(c2(g(c3(X)), k(d1(h(Y,1)))))$$

es el siguiente:

$$f(c(X,Y)) \leq c1(c2(Y1,Y3)) : -g(c3(X)) = Y1, h(Y,1) = Y2, k(d1(Y2)) = Y3.$$

En la cláusula anterior, seguimos la convención de Prolog usando $: -$ para $'si'$ y comas para la conjunción.

Definición 1.3.3 *La forma general de una cláusula en forma plana es:*

$$Cabeza : -Cuerpo$$

donde la Cabeza es de la forma $f(t) \leq u$, y t y u son términos, y el Cuerpo es de la forma $E_1, ..., E_n$. Cada E_i es de la forma $f_i(t_i) = y_i$, donde cada f_i es un símbolo de una función definida por el usuario, cada y_i es una nueva variable no presentada en la Cabeza, y cada t_i es un término que es equivalente al argumento de f_i, en la cláusula original sin formato plano. Cada fórmula $f_i(t_i) = y_i$ en el Cuerpo es llamada meta básica, y una lista de metas básicas es llamada una secuencia de metas básicas.

El orden en que las metas básicas son listadas en el lado derecho de una cláusula en formato plano es: el más a la izquierda es el más interno $(leftmost - innermost)$.

Semántica basada en modelos

La semántica declarativa de la programación de orden parcial se basa en el concepto de *modelo*, es por esto, la necesidad de esta sección. Para la presentación se utilizará la interpretación de Herbrand, donde el universo de Herbrand de un programa P sólo consiste de términos ground, y es representado como U_p. La base de Herbrand B_p de un programa P consiste de igualdades ground de la forma $f(t) = u$, donde f es una función definida por el usuario, t es un término ground y u es un término ground perteneciente a algún dominio(retículo completo). Dados dos términos ground t_1 y t_2, se escribe $t_1 \leq t_2$ para denotar que t_1 es menor o igual a t_1 en algun retículo.

La presentación de la semántica es sin referencia a un retículo específico tal como conjuntos, números, etc. Esto permite una presentación enfocada a nuestro objetivo principal, que son las cláusulas de orden parcial sin perder objetividad en la presentación.

Asumimos que toda interpretación I siempre incluye la igualdad y desigualdad de átomos de la forma $t_1 = t_2$ y $t_1 \leq t_2$, de acuerdo con la interpretación fija de estos en el programa. También, en toda interpretación I, f es interpretada como una función total, es decir:

(i) $(\forall t \in U_p)(\ \exists u \in U_p)\ f(t) = u$ *; y*

(ii) $f(t) = t_1 \in I$ *y* $f(t) = t_2 \in I \Rightarrow t_1 = t_2$.

Definición 1.3.4 ([7]) *Sea P un programa. Una interpretación M es un modelo de P, denotado por $M \vDash P$, si para toda instancia ground, $f(t) \leq t_1 : -E_1, ..., E_k$, de una cláusula $\leq$ en P y si $\{E_1, ..., E_k\} \subseteq M$ entonces existe un átomo $f(t) = u \in M$ y $u \leq t_1$.*

Se motivará brevemente el enfoque de semánticas basadas en modelos. Básicamente, se definirá la semántica de una función llamada $f(t)$, donde t es un término ground, para ser el *lub* de todos los términos definidos para $f(t)$ en los diferentes modelos de Herbrand para f. Para ver la necesidad de tomar los *lubs*, considere el siguiente programa trivial P:

$f(x) \leq 1$

Asumamos que el dominio resultante para f es el retículo de números totalmente ordenados, $N : 0 \leq 1 \leq 2...\top$(donde $\top$ denota el elemeto mayor del retículo). Cada modelo de P interpreta a f como una función constante.

f(x) = 0, para toda $x \in U_p$

f(x) = 1, para toda $x \in U_p$

El modelo asignado a la función f, es $f(x) = 1$ para toda $x \in U_p$, note que este modelo no es obtenido por la intersección clásica de conjuntos ($\cap$) de todos los modelos, pero en vez se usa el *lub* de los términos definidos para $f(x)$ en los diferentes modelos. En el ejemplo anterior, el *lub* es el *máximo*. Una importante propiedad en la programación de orden parcial es garantizada por el teorema siguente:

Teorema 1.3.1 ([7]) *Todo programa de orden parcial es consistente.*

Pero sintácticamente no todo programa bien formado tiene un significado bien definido. La circularidad en funciones definidas es permisible, a lo largo de su ocurrencia mediante funciones monótonas. Consideremos el siguiente programa (tomado de [7]) donde *not* es el operador de negación común, y como sabemos *not* no es una función monótona con respecto al retículo boleano $false \leq true$ y $not(false) = true$ y $not(true) = false$.

$a \geq not(b)$

$b \geq not(a)$

Este programa tiene tres modelos:

$\{a = true, b = true\}$

$\{a = true, b = false\}$

$\{a = false, b = true\}$

Sin embargo, tomando el glb[4] de los términos definidos para a y b respectivamente en los tres modelos obtenemos:

$\{a = false, b = false\}$

que claramente no es un modelo. Se puede observar que las funciones no monótonas no son permisibles cuando estas son definiciones circulares mediante funciones. Esto motiva a considerar programas de orden parcial estratificados.

Definición 1.3.5 ([7]) *Sea P un programa generalmente estratificado. Se define P_k como aquellas cláusulas de P para las cuales el símbolo de la función del lado izquierdo de la cláusula tiene un nivel $\leq k$.*

Definición 1.3.6 ([7]) *Dadas dos interpretaciones I y J para un programa P, se define $I \sqsubseteq J$ si $\forall f(t) = t_1 \in I$ existe $f(t) = t_2 \in J$ tal que $t_1 \leq t_2$. Se dice que $I = J$ si $I \sqsubseteq J$ y $J \sqsubseteq I$*

La construcción de la semántica basada en modelos de un programa generalmente estratificado se realizará nivel por nivel. Por este motivo se define una interpretación limitada por algún nivel.

Definición 1.3.7 ([7]) *Para cada interpretación I, $I_k := \{A : A \in I \wedge nivel(A) \leq k\}$*

Definición 1.3.8 ([7]) *Dadas dos interpretaciones I y J de un programa P, $I \sqcup J := \{f(t) = lub(u, u') \quad : \quad f(t) = u \in I, \quad f(t) = u' \in J, \; f$ un símbolo de función de P, $t \in U_P \}$*

[4]En este caso calculamos el *glb* de los términos porque el programa está definido por cláusulas $\geq$ que es el caso dual de las cláusulas $\leq$.

Definición 1.3.9 ([7]) *Para cualquier conjunto X de interpretaciones, $\sqcup X$ es la natural generalización de la anterior definición.*

Proposición 1.3.1 ([7]) *Sea X un conjunto de modelos para un programa P con nivel j tal que para cualquier $I \in X$ y $J \in X$, $I_{j-1} = J_{j-1}$. Entonces $\sqcup X$ es también un modelo.*

Definición 1.3.10 ([7]) *Sea P un programa con j niveles, se define la semántica basada en modelos de P como:*

for $j = 1$, $\mathcal{M}(P_1) := \sqcup\{M : M \models P_1\}$, *y*

for $j > 1$, $\mathcal{M}(P_j) := \sqcup\{M : M_{j-1} = \mathcal{M}(P_{j-1})$ *y* $M \models P_j\}$.

Definición 1.3.11 ([7]) *Sea P un programa con j niveles y G un conjunto de secuencia de metas, se dice que la sustitución θ es una respuesta correcta para G si*

$$\mathcal{M}(P_j) \models \bigwedge_{g \text{ es una meta basica en } G\theta} g$$

La noción de sustitución máxima es necesaria en este enfoque como se verá más adelante. En la siguiente definición, se usa la función $Vars(\theta)$ para hacer referencia al dominio de la sustitución θ. Por ejemplo $Vars(\{X \leftarrow 1, Y \leftarrow 2\}) = \{X, Y\}$.

Definición 1.3.12 (Sustitución Máxima [7]) *Sea θ_1 y θ_2 sustituciones. Se define $\theta_1 \leq \theta_2$ como $Vars(\theta_1) = Vars(\theta_1)$ y $\forall X$, si $X \leftarrow s_1 \in \theta$ entonces existe un s_2 tal que $X \leftarrow s_2 \in \theta \wedge s_1 \leq s_2$. Dada una sustitución, decimos que una sustitución es máximal si es el elemento máximo en el conjunto ordenado por $\leq$ como se definió anteriormente.*

Capítulo 2

Optimización del espacio de búsqueda

En la programación de orden parcial se puede hacer uso del conocimiento que los operadores *lub/glb* definen sobre el dominio de datos, al determinar un orden total para:

1. *Dar dirección a la búsqueda de una solución.*
2. *Realizar cortes al espacio de búsqueda.*

Esto se realiza, aplicando los operadores *lub/glb* a varias instancias de un problema para determinar qué ramificaciones en el espacio de búsqueda, nos llevan a soluciones no óptimas y podamos aplicar cortes al espacio de búsqueda, y qué ramificaciones son más adecuadas para que nos conduzcan a una solución óptima. Mas aún, la existencia de un orden total sobre el dominio de datos nos permite aplicar los operadores *lub/glb* cuando sólo parte del cómputo tenga lugar. En otras palabras, que podamos algunas veces eliminar una expresión cuando partes de esta aún no tienen solución, de esta forma eliminamos los cálculos de términos sin resolver.

Un importante aspecto de este enfoque es la habilidad de comparar expresiones con términos sin resolver; mientras estos problemas son en general complejos, la estructura de la mayoría de los programas es suficiente para hacerlo factible. Primero requerimos que el dominio de datos de las expresiones sean comparables para que sean totalmente ordenados; segundo, que los términos comprendidos en la expresión sean combinados con operadores monótonos. Dos variantes de esto son necesarias:

1. *La primera es el criterio de best-first para clasificar expresiones para que sean resueltas.*
2. *Un criterio de estricta comparación para eliminar expresiones que claramente son menos óptimas que otras.*

El criterio de *best-first* se aplica cuando una expresión contiene operadores monótonos que combinan valores ya calculados con subproblemas sin resolver. Idealmente, quisiéramos

comparar términos con alguna combinación de los valores ya calculados, y considerando el mejor término para ser el primero en evaluar con el valor más óptimo (desatendiendo la porción sin resolver). Este criterio nos dirige la búsqueda para una solución. En el segundo caso, consideramos términos donde las partes sin resolver son idénticas, comparamos los valores calculados y eliminamos los términos con los valores menos óptimos. Esta comparación nos permite realizar cortes al espacio de búsqueda, este tipo de corte es llamado corte monónoto. Otro tipo de corte con la misma idea es llamado corte inflationary, en este corte se aprovecha el conportamiento de las funciones inflationarias.

El objetivo de este capítulo es presentar la semántica operacional de la programación de orden parcial, la cual es extendida para mejorar su eficiencia de cálculo de una consulta al lenguaje mediante la incorporación de cortes. Además se presentan las pruebas de *Soundness* y *Completeness* de la semántica para asegurar que la extensión mantiene el buen comportamiento de la semántica. Y finalmente, en la última sección se presentan algunas ideas intuitivas de otras posibles extensiones a la semántica operacional que requieren un estudio cuidadoso.

2.1 Semántica operacional

La siguiente definición da un fundamento de la noción de corte declarativo, donde se sabe cierto conocimiento del comportamiento de la función. Se puede concluir fácilmente si dadas dos expresiones e_1 y e_2; si cumplen con algúna relación de orden por ejemplo: $e_1 \leq e_2$ la cual se escribe como $e_1 \preceq e_2$. Si además necesitamos calcular $lub\{e_1, e_2\}$ obtenemos e_2 sin calcular el valor de e_1.

Definición 2.1.1 *Dado un retículo L con el orden parcial $\preceq$, y dos expresiones e_1 y e_2, se escribe $e_1 \preceq e_2$ si y sólo si los siguientes casos se cumplen:*

1. *$e_1 = f(\overline{t}_1, a, \overline{t}_2)$, $e_2 = f(\overline{t}_1, b, \overline{t}_2)$, a y b son términos ground y $a \leq b$ es verdadero, donde f es una función monótona y $\overline{t}_1$, $\overline{t}_2$ son vectores de términos.*
2. *$e_1 = f(\overline{t}_1, a, \overline{t}_2)$, $e_2 = b$, a y b son términos ground y $a \leq b$ es verdadero, donde f es una función deflationary y $\overline{t}_1$, $\overline{t}_2$ son vectores de términos.*
3. *$e_1 = a$, $e_2 = f(\overline{t}_1, b, \overline{t}_2)$, a y b son términos ground y $a \leq b$ es verdadero, donde f es una función inflationary y $\overline{t}_1$, $\overline{t}_2$ son vectores de términos.*

Para ilustrar la anterior definición veamos el ejemplo siguiente:

Ejemplo 2.1 *Sea $f(X, Y) = X + Y$ y considere el retículo de los números naturales. Es claro que la función f es monótona (en ambos argumentos). Consideremos la instancia siguiente:*

$$f(3, X) \leq f(5, X)$$

note que no importando el valor de la variable X la relación se cumple. Además si consideramos el retículo de los números naturales entonces f es una función inflationary. Por ejemplo se cumple la relación

$$5 \leq f(5, X)$$

no importando el valor de la variable X.

Lema 2.1.1 *Dado un retículo y dos expresiones tal que* $e_1 \preceq e_2$ *entonces* $lub\{e_1, e_2\} = e_2$ *y* $glb\{e_1, e_2\} = e_1$.

Prueba. Es directa por lema 1.2.3. ■

Antes de presentar la definición formal de la semántica operacional, primero definiremos algunos conceptos como es la *reducción-glb* de una consulta ground con respecto a un programa generalmente estratificado P partiendo de que el programa P ya está en un formato plano.

Las siguientes definiciones son adaptaciones de algunas definiciones de [7].

Definición 2.1.2 *Dado un programa generalmente estratificado* P *y una consulta ground* $f(t_1)$, *donde* t_1 *es un término ground, se define la reducción-glb de* $f(t_1)$ *con respecto a* P *como la triple* $< G, V, s >$, *donde* G, V *y* s *se definen como sigue: (se asume que las variables en distintas cláusulas son diferentes).*

Sea $P_1 := \{f(t\theta) \leq Y \text{ :- } B\theta \mid f(t_1) \leq Y \text{ :- } B \in P \wedge \theta$ *es un match*[1] *de* t *y* $t_1\}$.

Entonces

$G := \{[B] \mid A \text{ :- } B \in P_1\}$
$V := \{Y \mid f(t) \leq Y \text{ :- } B \in P_1\}$
$s := glb(\{u \mid f(t_1) \leq u$ *es una instancia ground de una única cláusula* $\})$

Si no existen cláusulas con la cabeza $f(t) \leq u$ *tal que* $t\theta = t_1$ *para algún* θ, *entonces* $s = \top$, G *y* V *son vacíos.*

Ejemplo 2.2 *Sea* P *el siguiente programa (note que las cláusulas están en un formato plano):*

$h(X) \leq \{10, 40\}$
$h(X) \leq \{20, 40\}$
$h(X) \leq Z_1 \text{ :- } h(X) = Y_1, p(Y_1) = Z_1$
$h(X) \leq Z_2 \text{ :- } g(X) = Z_2$
$p(\{X \backslash \_\}) \leq \{X, 30\}$

Entonces la reducción-glb de $h(100)$ *con respecto a* P *es* $< G, V, s >$, *donde:*

$G := \{[h(100) = Y_1, p(Y_1) = Z_1], [g(100) = Z_2]\}$
$V := \{Z_1, Z_2\}$
$s := \{40\}$

[1]Note que Y no es afectada θ.

Definición 2.1.3 *Una memo-tabla es un conjunto de fórmulas de la forma $f(t) = u$, donde f es una función definida por el usuario, t es un argumento ground y u es un término.*

Definición 2.1.4 *Un functional-constraint es un conjunto de metas básicas de la forma $\{f_1(u_1) = X_1, \cdots, f_n(u_n) = X_n\}$, donde cada f_i es una función monótona y cada u_i es un término de la forma $glb\{t_i, X_{j1}, ..., X_{jm}\}$ de algunas varibles o posibles términos ground. Además, si toda u_i, donde $\{X_{j1}, \ldots, X_{jm}\} \subseteq \{X_1, \ldots, X_n\}$ y t_i un término ground, y toda X_i ocurre en algún u_j entonces diremos que el funtional-costraint es simple. Si $\{X_{j1}, \ldots, X_{jm}\}$ es vacío entonces u_i es sólo t_i. Un elemento $f_i(u_i) = X_i$ de un functional-constraint es llamado fácil constraint si u_i es ground.*

Definición 2.1.5 *Dado un functional-constraint $C := \{f_1(u_1) = X_1, \cdots, f_n(u_n) = X_n\}$, se define nivel(C)=$min\{i \mid P_i$ para toda $f_j,\ j = 1, ..., n\}$.*

Definición 2.1.6 *Una extensión de meta G^e es una cuádrupla de la forma $< G, C, T, L >$, donde G es un conjunto de secuencias de metas, C es un functional-constraint, T es una memo-tabla, y L es un número natural. Una extensión de meta inicial tiene la forma $< \{[f(t) = X]\}, \phi, \phi, L >$, donde f es una función definida por el usuario de nivel L, t es un término ground y X es una variable. Una extensión de meta final tiene la forma $< \phi, \phi, T, L >$, note que el conjunto de secuencias de metas y el functional-constraint son vacíos.*

Observe que no hay pérdida de generalidad en asumir que la extensión de meta inicial consiste de una simple llamada a una función $f(t)$.

Proposición 2.1.1 ([7]) *Todo functional-constraint tiene un respuesta correcta.*

En la proposición anterior, la respuesta correcta es evaluada por un procedimiento iterativo de mínimo punto fijo, presentado en la siguiente definición. Es claro que este procedimiento puede caer en un ciclo infinito. Se retomará este tema más tarde.

Definición 2.1.7 ([7]) *Para una functional-constraint $C := \{f_1(u_1) = X_1, ..., f_n(u_n) = X_n\}$ la respuesta, θ, para C es determinada por el siguiente procedimiento:*

$\sigma :=$ $\{X_1 \leftarrow \top, ..., X_n \leftarrow \top\}$
Repeat
$\theta := \sigma$.
$\sigma := \cup_{i=1,n}\{X_i \leftarrow s\}$ *donde s es la respuesta para* $f_i(u_i\theta)$
Until $\theta = \sigma$.
Return θ

En la definición anterior, se necesita obtener la respuesta para cada meta básica $f_1(u_1\theta)$, la cual es obtenida mediante el procedimiento presentado en la definición 2.2.1. Note que la definición 2.2.1 y la definición 2.1.7 son mutuamente recursivas, pero la recursión tiene un buen comportamiento porque si recordamos un functional constraint es un conjunto de funciones monótonas; además de ser estrictamente de un nivel menor.

Ahora definiremos la noción de *sustitución par* que es ligeramente diferente que la definición estándar de sustitución. De hecho esta sustitución simplemente elimina una variable en un contexto dado.

Definición 2.1.8 *Una sustitución par θ es una pareja de la forma $t_i, t_j \leftarrow t_i$, donde t_i es un término y t_j es una variable. Sea E una lista de metas básicas o un conjunto de secuencias de metas, se define $E\theta$ como sigue: Borra toda ocurrencia del término t_j en una expresión glb donde t_i ocurra también.*

Definición 2.1.9 *Sea S una expresión o un conjunto de metas básicas o una secuencia de metas. Se define el conjunto Variables(S) como sigue:*

$$Variables(S) := \{x \mid x \text{ es una variable que aparece en } S\}$$

Definición 2.1.10 *Sea G un conjunto de secuencias de metas y sea X una variable tal que X aparece en L y $L \in G$. Entonces se define la sustitución de X con respecto a G (denotada por θ_X) recursivamente como sigue:*

Caso base: Si $(e = X) \in L$ y e no tiene variables entonces $\theta_X := \{X \leftarrow e\}$ y $V_X := \{X\}$

Caso recursivo: Sea $(e = X) \in L$, si $V := Variables(e)$ entonces $\theta_X := \{X \leftarrow e\theta_V\}$ y $V_X := \{X\} \cup \bigcup_{Y \in V} V_Y$ donde $\theta_V := \{\theta_Y \mid Y \in V\}$.

Definición 2.1.11 *Sea T una memo-tabla y X una variable. Se define el conjunto de entradas de T dependientes de X como sigue:*

$$E_{X,T} := \{f(t) \mid (f(t) = w) \in T \text{ y } X \text{ aparece en } w\}$$

Definición 2.1.12 *Dado un programa generalmente estratificado P, se define la relación de reducción $G_1^e \rightarrow G_2^e$ como sigue: Sea $G_1^e =< G_1, C_1, T_1, L >$, donde $G_1 = \{S_1, \ldots, S_i, \ldots, S_n\}$ y S_i es de la forma $[E|R_1]$ y sea $R := G_1 \setminus \{S_i\}$. Entonces $G_2^e :=< G_2, C_2, T_2, L >$ es definido como sigue:*

P-m) **Corte monótono:** *Suponga que $e \in T_1$ donde e es de la forma $f(t) = glb\{t_1, ..., t_n\}$ y sea $\{t_i \leftarrow n_i\}$ la sustitución de t_i con respecto a G_1 y sea $\{t_j \leftarrow n_j\}$ la sustitución de t_j con respecto a G_1 (estas sustituciones son de acuerdo a la definición 2.1.10). Si*

$n_i \preceq n_j$ es verdadero (primer caso de la definición 2.1.1) entonces sea θ la sustitución par $t_i, t_j \leftarrow t_i$ y

$$T_2 := (T_1\theta \setminus \{\ g(t) = w \mid (g(t) = w) \in T_1\theta \text{ y } Variables(w) \cap V_{t_j} \neq \emptyset\})$$

donde V_{t_j} es el conjunto de variables como se define en 2.1.10 y

$$G_2 := (G_1 \setminus \{L \mid L \in G_1 \text{ y } (e' = X) \text{ aparece en } L \text{ y } X \in V_{t_j} y\ E_{X,T_2} = \emptyset\})\theta$$

donde $E_{X,T_1\theta}$ se presenta en la definición 2.1.11, y

$$C_2 := (C_1 \setminus \{e = X \mid (e = X) \in C_1 \text{ y } X \in V_{t_j} \text{ y } E_{X,T_2} = \emptyset\})\theta$$

P-i) **Corte inflationary:** *Suponga que $e \in T_1$ donde e es de la forma $f(t) = glb\{t_1, ..., t_n\}$ y existe un término ground t_i. Si existe $S \in G$ tal que $(g(t) = t_j) \in S$ donde g es una función inflationary y además $t_i \preceq g(t)$ es vedadero, entonces sea θ la sustitución par $t_i, t_j \leftarrow t_i$ y*

$$G_2 := (G_1 \setminus \{S\})\theta \quad si \quad E_{t_j,T_1\theta} = \emptyset \quad en\ otro\ caso \quad G_2 := G_1\theta$$

$$T_2 := T_1\theta \quad y \quad C_2 := C_1\theta$$

M-c) **Mueve Constraint**: *Si E es una función monótona y los argumentos de E no son ground, entonces $G_2 = R$, $C_2 := C_1 \cup \{E\}$ y $T_2 := T_1$. Note que en este caso $R_1 = \emptyset$*

C-e) **Solución de constraint fácil**: *Si $e \in C_1$ es un constraint fácil entonces sea θ la respuesta de $< \{[e]\}, \phi, \phi, nivel(e) >$. Además, se asume que $< \phi, \phi, T_3, nivel(e) >$ es la extensión de meta obtenida al calcular θ. Entonces $G_2 := G_1\theta$, $C_2 := (C_1 \setminus \{e\})\theta$, y $T_2 := (T_1\theta \cup T_3)$.*

Rec) **Recursión**: *Si E es de la forma $g(t_1) = X_1$, $nivel(g) < L$, t_1 es un término ground y la extensión de meta $< \{[g(t_1) = X_1]\}, \phi, \phi, nivel(g) >$ tiene una respuesta θ donde T_3 es la memo-tabla en la extensión de meta final, entonces $G_2 := (R_1 \cup R)\theta$, $C_2 := C_1\theta$, $T_2 := T_1\theta \cup T_3$.*

TLu) **Table Lookup**: *Si E es de la forma $g(t_1) = X_1$ y $g(t_1) = w \in T_1$ para algún w, entonces $G_2 := (R \cup R_1)\theta$, $C_2 := C_1\theta$, y $T_2 := T_1\theta$, donde $\theta := \{X_1 \leftarrow w\}$ si X_1 no ocurre en w; en otro caso, $\theta := \{X_1 \leftarrow glb\{s, Y_1, ..., Y_{i-1}, Y_{i+1}, ..., Y_m\}\}$, asumimos que w es de la forma $glb\{s, Y_1, ..., Y_{i-1}, X_1, Y_{i+1}, ..., Y_m\}$.*

Red) **Reducción**: *Si E es de la forma $g(t_1) = X_1$, $nivel(g) = L$, y $g(t_1)$ no pertenece a T_1, entonces sea la reducción-glb de E con respecto a P igual a $< G, \{Y_1, \cdots, Y_n\}, s >$. Entonces se define $\theta := \{X_1 \leftarrow glb\{s, Y_1, ..., Y_n\}\}$, y $G_2 := (G \cup R_1 \cup R)\theta$, y $C_2 := C_1\theta$, y $T_2 := (T_1 \cup \{g(t_1) = X_1\})\theta$.*

C-s) **Solución de un constraint simple**: *Si C_1 es un simple functional-costraint, entonces $G_2 := G_1\theta$, $C_2 := \phi$ y $T_2 := (T_1 \cup C_1)\theta$, donde θ es la respuesta calculada para C_1 (definición 2.1.7)*

El orden de selección no afecta el buen comportamiento (soundness) de la semántica operacional, pero sí la eficiencia. La experiencia nos sugiere el orden siguiente: el paso de reducción toma lugar sólo cuando no se puede encontrar alguna secuencia de metas donde podamos aplicar corte, solución de constraint fácil, recursión o table lookup. Además si varias secuencias de metas son posibles reducir, se puede seleccionar alguna con el criterio de *best-first* explicado anteriormente.

2.2 Formalización de la semántica operacional

El objetivo principal de esta sección es presentar las pruebas de dos teoremas importantes *Soundness* y *Completeness*. Estos dos teoremas nos garantizan el buen comportamiento del nuestra semántica operacional.

Definición 2.2.1 *Sea $\rightarrow^*$ para denotar la cerradura transitiva y reflexiva de la relación $\rightarrow$. Sea G_1^e una extensión de meta inicial que termina con la extensión de meta $G_2^e :=< \phi, \phi, T_2, L >$, es decir, $G_1^e \rightarrow^* G_2^e$. Se define la respuesta calculada para G_1^e como sigue: Si $G_1^e :=< \{[f(t) = X]\}, \phi, \phi, L >$, donde t es un término ground, entonces la respuesta calculada para G_1^e es $\{X \leftarrow s\}$, donde $f(t) = s \in T_2$ para algún s.*

Definición 2.2.2 *Una respuesta correcta para un conjunto de metas básicas S y/o secuencia de metas es una sustitución θ tal que θ es una respuesta correcta para toda $e \in S$.*

Definición 2.2.3 *Sea S un conjunto de metas o secuencia de metas. Se define RHS(S) como sigue:*

$$RHS(S) := \{X | f(t) = X \in S\}$$

Definición 2.2.4 *Sea S un conjunto de metas o secuencia de metas. Se define LHS(S) como sigue:*

$$LHS(S) := \{Variables(t) | f(t) = X \in S\}$$

Definición 2.2.5 *Sea P un programa generalmente estratificado y $G^e :=< G, C, T, L >$ una extensión de meta. Entonces G^e es llamada una tabla invariante si las siguientes condiciones se cumplen:*

1. *Toda entrada E de T es de la forma $f(t) = glb\{s_1, ..., s_m, X_1, ..., X_n\}$ donde $n \geq 0$, t y toda s_j son términos ground, toda X_i ocurre en G o en C.*

2. *Si $g(t) = u \in C$ entonces $Variables(u) \subseteq Variables(T)$.*

3. *Si $g(t) = u \in C$ y $Variables(t) \subseteq Variables(RHS(C) \cup (RHS(G)))$.*

4. *Si $g \in G$ y g es de la forma $[e_1, \ldots, e_n]$ entonces $Variables(LHS(e_i)) \subseteq Variables([e_1, \ldots, e_{i-1}])$, donde $1 \leq i \leq n$ y e_i no es una función monótona.*

5. *Si $g \in G$ y g es de la forma $[e_1, \ldots, e_n]$ entonces $Variables(LHS(e_i)) \subseteq Variables([e_1, \ldots, e_{i-1}] \cup RHS(C) \cup RHS(G \setminus g))$, donde $1 \leq i \leq n$ y e_i es una función monótona.*

6. *Si θ es la mayor respuesta correcta de $G \cup C$ entonces θ es la respuesta correcta de T.*

Teorema 2.2.1 (Soundness de un programa generalmente estratificado)
Sea P un programa generalmente estratificado y $G^e := < \{[f(t) = X]\}, \phi, \phi, nivel(f) >$ es una extensión de meta inicial que termina con la extensión de meta final $G_2^e :=< \phi, \phi, T_2, nivel(f) >$, formalmente $G^e \rightarrow^* G_2^e$. Entonces la respuesta calculda de G^e es correcta.

Prueba. La prueba es por una doble inducción sobre l (nivel) y n (longitud de la derivación). La idea básica es probar por inducción el buen comportamiento de la semántica impuesto por el nivel del programa, en otras palabras que toda extensión de meta G^e tal que $G^e \rightarrow^* G_1^e$ es invariante sobre toda la derivación.

Caso base l = 1: Ahora se prodecerá por inducción sobre n.

Caso base n = 1: Como la memo-tabla es vacía entonces se cumple la invariante.

Paso inductivo sobre n: Se necesitan verificar dos casos: *Table Look-up*, y *Reducción*. Estos casos ya fueron considerados en [7].

Paso inductivo sobre l: Nuevamente se procede por inducción sobre n.

Caso base n = 1: Como la memo-tabla es vacía entonces se cumple la invariante.

Paso inductivo sobre n: Necesitamos probar la invariante sobre todos los casos de la relación $\rightarrow$

P-m) Como $n_i \preceq n_j$ entonces por el lema 2.1.1, $glb\{n_i, n_j\} = n_i$. Entonces $glb\{t_1, ..., t_n\} = glb\{t_1, ..., t_{j-1}, t_{j+1}, ..., t_n\}$ y por consiguiente la sustitución es correcta. Esto implica que la última condición de la definición de la invariante es válida. La primera propiedad se mantiene por la construcción de la extensión de meta G^e en la condición $Variables(w) \cap Vt_j \neq \emptyset$. Las otras condiciones se mantienen de igual forma por construcción.

P-i) Como $t_i \preceq r$ entonces por lema 2.1.1, $glb\{t_i, r\} = t_i$. Entonces, $glb\{t_1, ..., t_n\}\} = glb\{t_1, ..., t_{j-1}, t_{j+1}, ..., t_n\}\}$ y por lo tanto la sustitución es correcta. Esto implica que la última condición de la definición de la invariante se satisface. El resto de las propiedades se cumplen debido a que la sustitución θ se aplica por todas las extensiones de metas para obtener la nueva.

M-C) Es directa.

C-e) En caso de solución de fácil constraint la justificación es la siguiente. Si e es un fácil constraint entonces es de un nivel menor que el nivel de la extensión de meta actual. Entonces por hipótesis de inducción asumimos que su respuesta calculada θ es correcta. Como θ es aplicada en todas la extensiones de metas se obtiene una nueva, donde la invariante se preserva.

Rec) De igual forma que el caso de C-e).

Rest) *Resto de casos:* Los demás casos son considerados en [7].

Entonces G_2^e es invariante, y toda entrada de T_2 es ground y verdadera.

■

Ahora probaremos la propiedad de *completeness* de nuestra semántica. Debemos asumir que el retículo es finito. Primero se mostrará que nuestra semántica operacional es completa sin corte para un retículo finito. Hay que recordar que todo retículo finito es completo. Para motivar el interés en retículos finitos veamos el siguiente ejemplo:

Ejemplo 2.3 *Sea N el conjunto de los números naturales. Se puede completar el retículo N agregando el elemento* $\top$. *Sea la función* $+$ *extendida de N a* $N \cup \{\top\}$ *como sigue*

$$\begin{array}{ll} X + Y = Z & \textit{where } X, Y, Z \in N \\ X + \top = \top & \textit{where } X \in N \\ \top + Y = \top & \textit{where } Y \in N \end{array}$$

Note que la función $+$ *es monótona sobre* $N \cup \{\top\}$. *Sea* `f` *la función* $\{X\} \rightarrow N \cup \{\top\}$. *Se define la función* `f` *como sigue:*

```
f(X) ≥ f(X) + 1.
```

Entonces `f(X)` $= \top$ *por la semántica declarativa, pero en nuestra semántica operacional no tiene una respuesta. Note que el programa anterior es un programa generalmente estratificado. Sin embargo la propiedad de buen comportamiento de la semántica operacional (completeness) se pierde aún en programas fuertemente estratificados (programas formados por sólo cláusulas S-S). Para ilustrar lo anterior consideremos el siguiente ejemplo:*

```
f(X) ≥ f(X+1).
```

Entonces `f(0)= 0` *por la semántica declarativa, pero nuestra semántica operacional no puede calcular una respuesta.*

Definición 2.2.6 *Sea $f : \mathcal{G} \to N^3$ (donde $\mathcal{G}$ es el conjunto de extensiones de metas y N el conjunto de los números naturales), se define como sigue:*

$$f(< G, C, T, L >) :=< n_1, n_2, n_3 >$$

donde $n_1 = \overline{T}$,[2], n_2 es el número de metas básicas que ocurren en G y $n_3 = |C|$.

Definición 2.2.7 *Se denota por $< n_1, n_2, n_3 >$ después de $< n'_1, n'_2, n'_3 >$ en N^3 el bien conocido orden lexicográfico.*

Teorema 2.2.2 (Completeness de programas generalmente estratificados sin corte) *Sea P un programa definido sobre un retículo completo. Supongase que la semántica operacional no tiene los casos de corte. Si θ es la respuesta correcta para una extensión de meta inicial G^e entonces θ es la respuesta calculada para G^e*

Prueba. Note que la propiedad de invariante también se cumple en una derivación sin cortes. Para la prueba, primero se mostrará que hay una extensión de meta final G_1^e tal que $G^e \to^* G_1^e$. Principalmente se necesitan verificar las siguientes condiciones:

1. La relación $\to$ siempre termina.
2. Si G_1^e no es una extensión de meta final entonces exite G_2^e tal que $G_1^e \to G_2^e$.

Prueba del primer caso: Note que si $G_8^e \to^+ G_9^e$ entonces (verificando que cada caso en $\to$), $\to^+$ es no reflexiva. Suponga ahora que existe una derivación infinita con todas las extensiones de metas diferentes entre ellas. Esta derivación no puede aplicar un número infinito de veces Red porque la base de Herbrand es finita y por lo tanto estaría en la memotabla. Entonces hay una extensión de meta i tal que después del paso i alcanzamos una extensión de meta E_i^e y de esta extensión de meta ya no se puede aplicar más Red. Sea $f(E_i^e) =< n_1, n_2, n_3 >$. Considere j pasos tal que $j > i$. Entonces es fácil verificar que $f(E_j^e) =< n_1, m_2, m_3 >$ y $< n_1, m_2, m_3 >$ *después de* $< n_1, n_2, n_3 >$. Por consiguiente existe k tal que k $(k \geq i)$ pasos obtenemos $f(E_k^e) =< n_1, 0, 0 >$ y entonces E_k^e es una extensión de meta final, obteniendo una contradicción.

Prueba del segundo caso: Suponga que G_1 no es vacía. Entonces sea g la meta inicial de alguna lista G_1. Si g es una función monótona entonces podemos aplicar constraint o recursión (dependiendo de los argumentos de la función). En otro caso g es una función estándar (no monótona). Entonces es importante puntualizar que los argumentos de g deben ser ground (por la propiedad de la invariante) y por consiguiente siempre se puede aplicar recursión y table lookup o reducción (dependiendo del nivel de g).

[2]Donde $\overline{T} = MAX - |T|$ y MAX es la cardinalidad de la base de Herbrand B_P.

Suponga que G_1 es vacía, entonces C debe ser una simple constraint por la propiedad de invarianza y por lo tanto se puede aplicar *C-s*.

Como sabemos que hay una extensión de meta final G_1^e tal que $G^e \rightarrow^* G_1^e$. Por soundness, la respuesta calculada por la semántica es correcta. ■

Note, que sin embargo algunos retículos infinitos permiten calcular una respuesta por la semántica operacional.

Teorema 2.2.3 (Completeness de programas generalmente estratificados)
Sea P un programa definido sobre un retículo completo. Suponga que la semántica operacional incluye los casos de corte. Si θ es una respuesta correcta para una extensión de meta G^e entonces θ es una respuesta calculada para G^e.

Prueba. Lo único que hay que destacar es que el corte monótono puede borrar entradas de la memo-tabla. Sin embargo por soundness sabemos que no necesitamos tales entradas en la tabla para obtener la respuesta correcta, por lo tanto podemos borrar tales entradas sin ningun riesgo. ■

2.3 Extensiones

En esta sección se consideran dos simples extensiones a la semántica operacional las cuales pueden aplicar dos tipos de corte al espacio de búsqueda sin afectar el teorema de soundness.

2.3.1 Corte Alpha-Beta

Una posible extensión a considerar en el paradigma de programación de orden parcial es la combinación de los dos tipos de desigualdades (cláusulas con $\geq$ y con $\leq$). En este contexto podemos hacer uso del bien conocido corte *Alpha-Beta*. Para mostrar la idea veamos el siguiente ejemplo:

Ejemplo 2.4 (Min-Max)

```
p(X)  ≤  X1  :-  final(X,X1).
p(X)  ≤  X2  :-  arco2(X,Y), q(Y) = X2.
q(X)  ≥  X3  :-  final(X,X3).
q(X)  ≥  X4  :-  arco1(X,Y), p(Y) = X4.
```

Este programa modela el juego entre dos jugadores mediante un grafo bipartite, el cual es representado por los predicados `arco1` y `arco2`. La función `p` obtiene el valor mínimo de que gane un jugador A, mientras que `q` obtiene al valor máximo de que gane el jugador B

(donde B es el jugador contrincante del jugador A). El programa es considerado no monótono puesto que existe una dependencia entre cláusulas con la desigualdad $\geq$ y las cláusulas con la desigualdad $\leq$. Ahora consideremos la siguiente EDB:

```
final(d,0).     arco1(a,b).
final(e,1).     arco1(a,c).
final(f,-1).    arco2(b,d).
final(g,0).     arco2(b,e).
                arco2(c,f).
                arco2(c,g).
```

Como veremos en la derivación más adelante la semántica operacional tiene un comportamiento muy semejante al bien conocido corte *alpha-beta*. En este ejemplo se asume que el retículo en cuestión tiene tres elementos: `-1, 0, 1` con la tradicional relación de orden en los números naturales.

Si reducimos el programa con su respectiva EDB se obtiene el siguiente programa:

```
p(d) ≤ 0.   p(b) ≤ X2 :- q(d) = X2.
p(e) ≤ 1.   p(b) ≤ X2 :- q(e) = X2.
p(f) ≤ -1.  p(c) ≤ X2 :- q(f) = X2.
p(g) ≤ 0.   p(c) ≤ X2 :- q(g) = X2.

q(d)≥ 0.   q(a)≥ X4 :- p(b) = X4.
q(e)≥ 1.   q(a)≥ X4 :- p(c) = X4.
q(f)≥ -1.
q(g)≥ 0.
```

Ahora consideremos la consulta `q(a)=Ans`. La siguiente derivación muestra que `Ans = 0` y muestra una nueva forma de corte en nuestra semántica operacional.

Goal Sequence	Func -Const	θ	Memo Tabla
{[q(a)=Ans]}	ϕ		ϕ
{ [p(b)=N1)][p(c)=N2]}	ϕ	Ans←lub{N1,N2}	{ q(a)=lub{N1,N2}}
{ [q(d)=N3][q(e)=N4] [p(c)=N2]}	ϕ	N1←glb{N3,N4}	{q(a)=lub{glb{N3,N4},N2} p(b)=glb{N3,N4}}
{ [q(e)=N4] [p(c)=N2]}	ϕ	N3←0	{q(a)=lub{glb{0,N4},N2} p(b)=glb{0,N4},q(d)=0}
{ [p(c)=N2]}	ϕ	N4←1	{q(a)=lub{glb{0,1},N2} p(b)=glb{0,1} q(d)=0, q(e)=1}
{ [q(f)=N5][q(g)=N6]}	ϕ	N2←glb{N5,N6}	{q(a)=lub{0,glb{N5, N6}} p(b)=0,q(d)=0, q(e)=1, p(c)=glb{N5, N6}}
{ [q(g)=N6]}	ϕ	N5←-1	{q(a)=lub{0,glb{-1,N6}} p(b)=0,q(d)=0, q(e)=1, p(c)=glb{-1, N6}}
ϕ	ϕ	pruning	{q(a)=0 p(b)=0,q(d)=0, q(e)=1, p(c)=-1}

Note que en el último paso de la derivación se hace un corte a la búsqueda porque sí observamos `q(a) = lub{0,glb{-1,N6}} = 0` menospresiando el valor de `N6`. Observe que otro tipo de corte es posible, puesto que `glb{-1,N6} = -1` y como `-1` es el elemento menor del retículo.

Un interesante punto resultante de esta clase de programas son los programas que tienen diferentes modelos. En otras palabras que la respuesta a una consulta en el programa no es única. Para ver esto consideremos la siguiente EDB:

```
final(e,3).   arco2(g,i).    arco1(j,h).
final(f,2).   arco2(g,f).    arco1(i,g).
final(k,2).   arco2(g,e).    arco1(i,h).
final(m,1).   arco2(h,e).    arco1(i,m).
              arco2(h,i).    arco1(j,m).
              arco2(h,j).    arco1(j,k).
```

Y se puede verificar que el programa tiene tres posibles modelos, los cuales son:

i: `q(e) = 3, q(j) = 2, q(f) = 2, q(i) =1, p(e)= 3, p(k)= 2, p(f) =2, p(m)=1, p(h)=1, p(g)=1, q(k) = 2, q(m) = 1.`

ii: `q(e) =3, q(j) = 2, q(i) = 2, q(f) = 2, p(e) =3, p(k) = 2, p(h) = 2, p(g) = 2, p(f) = 2, p(m) =1, q(k)= 2, q(m) =1.`

iii: `q(j) = 3, q(i) = 3, q(e) = 3, q(f) = 2, p(h) = 3, p(e) = 3, p(k) =2 , p(g) = 2, p(f)= 2, p(m) =1, q(k) = 2, q(m) = 1.`

Este tipo de programas se dejan como un problema abierto para estudio de una amplia clase de programas.

2.3.2 Corte por lemas

Otra posible extensión al paradigma de programación de orden parcial es el uso de desigualdades tipo lema que es conocimiento que tiene el programador del problema. Este tipo de desigualdades se puede usar para realizar cortes al espacio de búsqueda. Para presentar la idea veamos el siguiente ejemplo.

Ejemplo 2.5 *Consideremos nuevamente el ejemplo 1.7. Sea* $\mathtt{kn}_{01}$ *la función asociada al problema de la mochila booleana y sea* $\mathtt{kn}_{rat}$ *la función asociada al problema de la mochila racional. Es bien sabido que la siguiente desigualdad siempre se cumple:*

$$\mathtt{kn}_{01}(\mathtt{X},\mathtt{Y}) \leq \mathtt{kn}_{\mathtt{rat}}(\mathtt{X},\mathtt{Y})$$

para todo valor de `X` *y* `Y`*. Esto es una desigualdad de tipo lema para el problema de mochila. Por consiguiente podemos usar esta información para reducir el espacio de búsqueda. Consideremos el siguiente programa que modela el problema de mochila y además integra el lema antes presentado.*

```
c(1) ≤ 9.     g(1) ≤ 16.
c(2) ≤ 7.     g(2) ≤ 13.
c(3) ≤ 8.     g(3) ≤ 15.
c(4) ≤ 12.    g(4) ≤ 24.
c(5) ≤ 11.    g(5) ≤ 23.
kn01(0,M) ≥ 0.
kn01(I,M) ≥ Y :- I ≥ 1,   I − 1 = J,   kn01(J,M) = Y.
kn01(I,M) ≥ Z :- I ≥ 1,   c(I) = D,   D ≤ M,   g(I) = G,   M − D = M1,
                 I - 1 = L,  kn01(L,M1) = R,  R + G = Z
kn01(I,M) ≤ knrat(I,M).
```

Se omite la definición de la función $\mathtt{kn}_{rat}$ *(problema de la mochila racional) porque es no relevante. Ahora veamos la derivación donde se hace uso del lema.*

S-T	Goal Sequence	Inequality-constr.	Memo Table
	$\{[\mathtt{kn}_{01}(5,26)=\mathtt{Ans}]\}$	$\mathtt{Ans} \leq \mathtt{kn}_{\mathtt{rat}}(5,26)$	ϕ
I-C	$\{[\mathtt{kn}_{01}(5,26)=\mathtt{Ans}]\}$		ϕ
	$\{[\mathtt{kn}_{01}(5,26)=\mathtt{Ans}]\}$	$\mathtt{Ans} \leq 52.62$	$\{\mathtt{kn}_{\mathtt{rat}}(5,26)=52.62\}$
	$\{[5 \geq 1, 5-1=\mathtt{J}, \mathtt{kn}_{01}(\mathtt{J},26)=\mathtt{Y}]$ $[5 \geq 1, \mathtt{c}(5)=\mathtt{D}, \mathtt{D} \leq 26, \mathtt{g}(5)=\mathtt{G},$ $26-\mathtt{D}=\mathtt{M1}, 5-1=\mathtt{L},$ $\mathtt{kn}_{01}(\mathtt{L},\mathtt{M1})=\mathtt{R}, \mathtt{R}+\mathtt{G}=\mathtt{Z}]\}$		$\{\mathtt{kn}_{01}(5,26)=lub\{\mathtt{Y},\mathtt{Z}\},$ $\mathtt{kn}_{\mathtt{rat}}(5,26)=52.62\}$
	...(several steps)		...
	$\{[\mathtt{kn}_{01}(4,26)=\mathtt{Y}]$ $[\mathtt{kn}_{01}(4,15)=\mathtt{R}, \mathtt{R}+23=\mathtt{Z}]\}$		$\{\mathtt{kn}_{01}(5,26)=lub\{\mathtt{Y},\mathtt{Z}\},$ $\mathtt{kn}_{\mathtt{rat}}(5,26)=52.62\}$
Red	...(several steps)		...
	$\{[\mathtt{kn}_{01}(4,26)=\mathtt{Y}]\}$		$\{\mathtt{kn}_{01}(5,26)=lub\{\mathtt{Y},51\},...,$ $\mathtt{kn}_{01}(4,15)=28,$ $\mathtt{kn}_{\mathtt{rat}}(5,26)=52.62\}$
I-C	$\{[\mathtt{kn}_{01}(4,26)=\mathtt{Y}]\}$	$\mathtt{Y} \leq \mathtt{kn}_{\mathtt{rat}}(4,26)$	$\{\mathtt{kn}_{01}(5,26)=lub\{\mathtt{Y},51\},...,$ $\mathtt{kn}_{01}(4,15)=28,$ $\mathtt{kn}_{\mathtt{rat}}(5,26)=52.62\}$
P-l	$\{[\mathtt{kn}_{01}(4,26)=\mathtt{Y}]\}$	$\mathtt{Y} \leq 50.14$	$\{\mathtt{kn}_{01}(5,26)=lub\{\mathtt{Y},51\},...,$ $\mathtt{kn}_{01}(4,15)=28,$ $\mathtt{kn}_{\mathtt{rat}}(5,26)=52.62,$ $\mathtt{kn}_{\mathtt{rat}}(4,26)=50.14\}$
	ϕ		$\{\mathtt{kn}_{01}(5,26)=51,...,$ $\mathtt{kn}_{01}(4,15)=28,$ $\mathtt{kn}_{\mathtt{rat}}(5,26)=52.62,$ $\mathtt{kn}_{\mathtt{rat}}(4,26)=50.14\}$

En el primer paso el código I-C significa que la meta básica tiene asociada una desigualdad de tipo lema la cual tiene que ser resuelta primero. Después de varios pasos se aplica una reducción. Por el criterio del best-first se selecciona la segunda secuencia de meta para ser resuelta. Pero más adelante nuevamente encontramos un I-C. Finalmenet encontramos un P-l que es el código de corte por lema el cual nos permite ignorar la meta $\mathtt{kn}_{01}$`(4,26)=Y`.

Conclusiones y trabajo a futuro

La principal aportación de esta tesis es mostrar como la eficiencia de un lenguaje funcional de consultas puede sacar provecho de la noción de corte declarativo. Se desarrolla un enfoque partiendo del paradigma de programación de orden parcial, el cual provee un buen soporte para un lenguaje funcional de consultas. La idea primordial de la tesis es mostrar que es posible hacer más eficiente el cálculo de una respuesta en la semántica operacional cuando el dominio de las funciones es totalmente ordenado. Esta información hace más rápida la búsqueda de una solución, esto es gracias a que se hace corte de ramas en el espacio de búsqueda para las cuales no existe una solución óptima. Este enfoque ofrece una significante mejora en la estrategia top-down en la que se eliminan ciclos infinitos, y la solución de submetas iguales sólo requiriendo de una memo-tabla.

Ahora el siguiente paso en esta línea de investigación sera la implementación de esta semántica operacional. Además del estudio detallado de las extensiones presentadas en la última sección del capítulo 2. Y por último determinar que otras estrategias de búsqueda pueden ser usadas, y cómo el usuario podría especificarlas.

Bibliografía

[1] Stefan Brass, Ulrich Zukowski, and Burkhardt Freitag. Transformation Based Bottom-Up Computation of the Well-Founded Model. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 1216, pages 171–201. Springer, Berlin, 1997.

[2] B.A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, New York, USA., 1990.

[3] Bharat Jayaraman. Implementation of subset-equational programs. *Journal of Logic Programming*, 11(4):299–324, 1992.

[4] Bharat Jayaraman and K. Moon. Subset logic programs and their implementation. *Journal of Logic Programming*, 41(2):71–110, 2000.

[5] John W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1987. 2nd edition.

[6] Bharat Jayaraman Mauricio Osorio and J. C. Nieves. Declarative pruning in a functional query language. In Danny De Schreye, editor, *Proceedings of the International Conference on Logic Programming*, pages 588–602. MIT Press, 1999.

[7] Bharat Jayaraman Mauricio Osorio and David Plaisted. Theory of partial-order programming. *Science of Computer Programming*, 34(3):207–238, 1999.

[8] J. C. Nieves, Mauricio Osorio, Fernando Zacarias, and Eleazar Oropeza. Partial-order relational programming. In *Proccedings of the First International Worschop on Rule-Based Programming (RULE'2000)*, pages ?–?, Montreal, Canada, 2000.

[9] Juan Carlos Nieves and Gabriel Cervantes. Is the class of well-behaved semantics so small? In *Proceedings of 12th European Summer School in Logic, Language and Information*, pages 189–198, Birmingham, U.K., 2000.

[10] Mauricio Osorio and J. C. Nieves. Extended partial-order logic programming. In Gianfranco Rossi and Bharat Jayaraman, editors, *Proceedings of the Workshop on Declarative Programming with Sets*, pages 19–26, Paris, France, 1999.

[11] Mauricio Osorio and J. C. Nieves. Partial-order functional-logic programming. In *Proceedings of the Simposium Internacional de Computacion CIC '99*, pages ?–? Mexico, 1999.

[12] Mauricio Osorio, J. C. Nieves, and Gabriel Cervantes. Application of simplification theories. In Hans Jurgen Ohlbach, Ulrich Endriss, Odinaldo Rodrigues, and Stefan Schlobach, editors, *Proceeding of Seventh Workshop onAutomated Reasoning*, pages ?–? King's College London, U.K., 2000.

[13] Mauricio Osorio, J. C. Nieves, and Chis Giannella. Useful transformations in answer set programming. In *Proceedings of the AAAI 2001 Spring Symposium Series*, page to appear. AAAI press, Stanford, E.U., 2001.

[14] Mauricio Osorio, J. C. Nieves, Fernando Zacarias, and E. Saucedo. Knowledge representation using high-level non-monotonic reasoning. LNAI 1793, pages 13–24, Mexico, 2000. Springer Verlag.

[15] K.A. Ross and Charles R. B. Wright. *Matematicas discretas.* Prentice - Hall, 1990.