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`.