# NGỮ NGHĨA THỦ TỤC CỦA CHƯƠNG TRÌNH LOGIC CÓ RÀNG BUỘC

*Trương Công Tuấn*
*Trường Đại học Khoa học, Đại học Huế*
*Trần Thị Ngọc Trang*
*Trung tâm Công nghệ thông tin, Đại học Huế*

## TÓM TẮT

*Lập trình logic ràng buộc (CLP) là một hướng tiếp cận mới trong lập trình logic, được ra đời bởi sự kết hợp tính khai báo của lập trình logic với tính hiệu quả của quá trình giải quyết ràng buộc. Trong bài báo này, chúng tôi tập trung trình bày ngữ nghĩa thủ tục của chương trình logic có ràng buộc thông qua các dẫn xuất và cây suy dẫn từ đích, đồng thời chỉ ra những điều kiện trên hàm xử lý ràng buộc để đảm bảo rằng ngữ nghĩa này là độc lập với các quy tắc chọn literal trong đích.*

## I. Mở đầu

Lập trình logic ràng buộc (CLP) là một hướng mở rộng của lập trình logic, đã được nhiều người đầu tư nghiên cứu và có thể tìm thấy trong nhiều công trình [1], [2], [3], [8]. Cơ chế lập trình này đưa ra một khung hình thức với việc tổng quát hóa các hệ phương trình hạng thức trong lập trình logic thành các ràng buộc từ miền tính toán đã được định nghĩa trước. Việc nghiên cứu ngữ nghĩa của các ngôn ngữ CLP đã thực sự hữu dụng trong việc mô hình hóa hệ thống và giải quyết các bài toán tối ưu phức tạp trong cuộc sống.

Trong bài báo này, chúng tôi nghiên cứu ngữ nghĩa thủ tục của chương trình logic có ràng buộc và chỉ ra những điều kiện trên hàm xử lý ràng buộc để đảm bảo rằng ngữ nghĩa này là độc lập với các quy tắc chọn literal trong đích.

## II. Một số định nghĩa và khái niệm cơ sở

Phần này chỉ trình bày tóm tắt một số khái niệm cơ sở của chương trình logic có ràng buộc, chi tiết đầy đủ hơn cũng như một số khái niệm khác của lập trình logic có thể xem trong [10].

### *2.1. Miền ràng buộc*

**Định nghĩa 2.1.** *Bộ ký hiệu* là một tập hữu hạn, khác rỗng các ký hiệu, bao gồm các ký hiệu hàm và ký hiệu vị từ. Mỗi ký hiệu có một số tự nhiên kèm theo, gọi là bậc của ký hiệu. (Bộ ký hiệu thường được ký hiệu là $\sum$).

**Định nghĩa 2.2.** Cho bộ ký hiệu $\sum$, một $\sum$-*cấu trúc*, ký hiệu là $\mathcal{D}$, là một thể hiện của các ký hiệu trong $\sum$ bao gồm:

Một tập $D$ khác rỗng,

Một phép gán mỗi ký hiệu hàm $f$ bậc $n$ trong $\sum$ với một ánh xạ từ $D^n$ vào $D$.

Một phép gán mỗi ký hiệu vị từ $p$ bậc $n$ trong $\sum$ với một ánh xạ từ $D^n$ vào tập {*true, false*}.

Các vị từ trong chương trình logic có ràng buộc được chia thành hai lớp: các *ràng buộc nguyên tố* và các *nguyên tố* do người sử dụng định nghĩa. Các ràng buộc nguyên tố đã được định nghĩa với ngữ nghĩa xác định.

**Định nghĩa 2.3.** Nếu $p$ là ký hiệu vị từ bậc $n$ và $t_1,...,t_n$ là các hạng thức thì $p(t_1,...,t_n)$ được gọi là một *nguyên tố*.

**Định nghĩa 2.4.**

Một *ràng buộc nguyên tố* có dạng $p(t_1,\dots t_n)$, trong đó $t_1,\dots, t_n$ là các hạng thức và $p \in \sum$ là một ký hiệu vị từ.

Một *ràng buộc* là hội của các ràng buộc nguyên tố.

Một *literal* là một nguyên tố hoặc ràng buộc nguyên tố.

**Định nghĩa 2.5.** Một $\sum$-*công thức* là một công thức bậc nhất được xây dựng từ các ràng buộc nguyên tố, các ký hiệu kết nối logic $\wedge, \vee, \neg, \rightarrow$ các ký hiệu lượng tử $\forall$ và $\exists$.

**Định nghĩa 2.6.**

$\sum$-công thức được gọi là *đóng* nếu mọi biến xuất hiện trong công thức đều thuộc vào phạm vi của các lượng từ $\forall$, $\exists$.

$\sum$-*lý thuyết* là một tập các $\sum$-công thức đóng.

Cơ chế lập trình logic ràng buộc định nghĩa nên một lớp các ngôn ngữ *CLP*($\mathcal{C}$) trên một miền ràng buộc $\mathcal{C}$. Miền ràng buộc xác định các ràng buộc và tập các ký hiệu hàm, ký hiệu hằng để từ đó các hạng thức trong chương trình có thể được xây dựng. Ta có định nghĩa miền ràng buộc $\mathcal{C}$ như sau:

**Định nghĩa 2.7.** Với bất kỳ bộ ký hiệu $\sum_C$ nào, một *miền ràng buộc* $\mathcal{C}$ sẽ bao gồm 2 thành phần sau:

*Miền tính toán*, ký hiệu là $\mathcal{D}_C$, là $\sum$-*cấu* trúc, nghĩa là thể hiện của các ràng buộc.

*Lớp các ràng buộc*, ký hiệu là $\mathcal{L}_C$ , là tập các $\sum$-công thức.

**Định nghĩa 2.8.** *Hàm xử lý ràng buộc* đối với tập $\mathcal{L}_C$, ký hiệu là $solv_C$ là hàm gán mỗi công thức trong $\mathcal{L}_C$ với một trong các giá trị *đúng*, *sai* hoặc *chưa biết*, chỉ ra

rằng một công thức là *thỏa mãn, không thỏa mãn* hoặc *không xác định*.

Các sự lựa chọn khác nhau về miền ràng buộc và hàm xử lý ràng buộc sẽ phát sinh các ngôn ngữ lập trình khác nhau. Đối với miền ràng buộc $\mathcal{C}$, ta gọi *CLP*($\mathcal{C}$) là ngôn ngữ lập trình ràng buộc dựa trên $\mathcal{C}$.

**Ví dụ 2.1.** Với bộ ký hiệu $\sum_C$ bao gồm 0, 1, $\wedge, \vee, \rightarrow$ và ký hiệu vị từ =, ta có *miền ràng buộc kiểu boolean trên logic hai trị* gồm hai thành phần như sau:

*Miền tính toán* $\mathcal{D}_C$ bao gồm tập *D* là tập các giá trị {*true, false*}. Lúc này $\mathcal{D}_C$ xem các ký hiệu trong $\sum_C$ như là các hàm logic, chẳng hạn $\vee$ là một toán tử logic *OR*, $\wedge$ là toán tử logic *AND*.

*Lớp các ràng buộc* $\mathcal{L}_C$ bao gồm tập các công thức bậc nhất được tạo ra từ các ràng buộc nguyên tố. Chẳng hạn, ta có một ràng buộc trong $\mathcal{L}_C$ như sau: $(x \rightarrow y) \wedge z = 0$.

**Ví dụ 2.2.** Với bộ ký hiệu $\sum_C$ bao gồm $\geq, \leq, >, <, =$ là các ràng buộc nguyên tố, các ký hiệu hàm +, – , * , /, và dãy các số với dấu chấm thập phân là ký hiệu hằng, ta có *miền ràng buộc trên tập các số thực* gồm hai thành phần như sau:

*Miền tính toán* $\mathcal{D}_C$ là tập các số thực, ký hiệu là $\mathcal{R}$.

*Lớp các ràng buộc* $\mathcal{L}_C$ bao gồm các ràng buộc nguyên tố $\geq, \leq, <, >, =$ được thể hiện như các phép toán quan hệ trên $\mathcal{R}$, các ký hiệu hàm +, – , * và / là các phép toán số học trên $\mathcal{R}$. Các ký hiệu hằng được thể hiện như là một biểu diễn thập phân của các thành phần của $\mathcal{R}$.

**Định nghĩa 2.9.** Một ràng buộc nguyên tố *L* được gọi là *nhất quán* với ràng buộc *c* trong hàm xử lý ràng buộc *solv*(*c*) nếu *solv*($c \wedge L$) ≠ *false*, ngược lại ta nói *L không nhất quán* với *solv*(*c*).

### 2.2. Chương trình logic có ràng buộc

Chương trình logic có ràng buộc là một mở rộng của chương trình logic bằng cách cho phép các ràng buộc xuất hiện trong thân của các quy tắc và đích. Một chương trình logic có ràng buộc được định nghĩa như sau:

**Định nghĩa 2.10.** Một *chương trình logic có ràng buộc* là một tập hữu hạn các *quy tắc* có dạng:

$A \leftarrow c, B_1, ..., B_n$

trong đó:

*A* là một nguyên tố, được gọi là *đầu* của quy tắc;

$c, B_1, ..., B_n$ là hội của ràng buộc *c* và các literal $B_i$ ($i=1,\ldots, n$), được gọi là *thân* của quy tắc.

Với *P* là chương trình logic có ràng buộc, ta ký hiệu $defn_P(p(t_1, \ldots, t_n))$ là tập các

các quy tắc của $P$ sao cho đầu của mỗi quy tắc có dạng $p(s_1, \ldots, s_n)$.

**Định nghĩa 2.11.** Một *đích* có dạng $c, B_1, ..., B_m$ là hội của ràng buộc $c$ và các literal $B_i$ $(i=1,\ldots, m)$.

**Ví dụ 2.3.** Cho một chương trình logic có ràng buộc và đích trong miền ràng buộc số thực như sau :

$p(X, Y) \leftarrow X > Z, Y \leq 1 + Z, Z \geq 0, q(Z)$

$p(X, Y) \leftarrow X < Z, Y \leq 1 - Z, Z \leq -2, r(Y, Z)$

Đích: $X \leq 0, p(X, Y)$

## III. Ngữ nghĩa thủ tục của chương trình logic có ràng buộc

### *3.1. Mô tả ngữ nghĩa thủ tục*

Ngữ nghĩa thủ tục của chương trình logic có ràng buộc được định nghĩa dưới dạng các *dẫn xuất* từ đích. Ta có định nghĩa sau:

**Định nghĩa 3.1.** Một *dẫn xuất* là dãy các *phép biến đổi* giữa các trạng thái, ở đó mỗi trạng thái là một bộ $\langle G \mid c \rangle$ với $G$ là *đích hiện thời*, và $c$ là *ràng buộc hiện thời* của trạng thái đó.

Tại mỗi bước biến đổi, một literal trong đích được chọn theo một quy tắc chọn cố định nào đó, thường là theo hướng từ *trái sang phải*. Nếu literal là một ràng buộc nguyên tố, và nhất quán với ràng buộc hiện thời, thì nó được thêm vào kho ràng buộc hiện thời. Nếu nó không nhất quán thì dẫn xuất sẽ *thất bại*. Nếu literal là một nguyên tố, thì nó được biến đổi bằng cách sử dụng một trong các quy tắc định nghĩa nguyên tố đó.

Như vậy, một trạng thái $\langle L_1, ..., L_m \mid c \rangle$ có thể được biến đổi như sau: Chọn một literal $L_i$ trong đích và xét các trường hợp sau:

Nếu $L$ là một ràng buộc nguyên tố và $solv(c \wedge L) \neq false$, thì nó được biến đổi thành $\langle L_1, ..., L_{i-1}, L_{i+1}, ..., L_m \mid c \wedge L \rangle$.

Nếu $L$ là một ràng buộc nguyên tố và $solv(c \wedge L) = false$, thì nó được biến đổi thành ‹W| *false*›, trong đó W là ký hiệu cho một đích rỗng.

Nếu $L$ là một nguyên tố, thì nó được biến đổi thành:

$\langle L_1, ..., L_{i-1}, s_1 = t_1, ..., s_n = t_n, B, L_{i+1}, ..., L_m \mid c \rangle$

với $(A \leftarrow B) \in defn_P(L)$, trong đó $L$ có dạng $p(s_1, \ldots, s_n)$ và $A$ có dạng $p(t_1, \ldots, t_n)$.

Nếu $L$ là một nguyên tố và $defn_P(L) = \varnothing$ thì nó được biến đổi thành ‹W| *false*›.

Một dẫn xuất từ đích $G$ trong chương trình $P$ là một dãy các trạng thái $S_0 \Rightarrow S_1$

$\Rightarrow \ldots \Rightarrow S_n$ trong đó $S_0$ là $\langle G \mid true \rangle$ và có một phép biến đổi từ $S_{i-1}$ thành $S_i$ bằng cách sử dụng các quy tắc trong *P*. *Chiều dài* của một dẫn xuất có dạng $S_0 \Rightarrow S_1 \Rightarrow \ldots \Rightarrow S_n$ là *n*.

Một dẫn xuất từ *G* được gọi là *kết thúc* nếu đích cuối cùng không thể biến đổi được nữa. Trạng thái cuối cùng trong một dẫn xuất được kết thúc từ *G* phải có dạng ‹W |*c*›. Nếu *c* = *false* thì dẫn xuất được gọi là *thất bại*. Ngược lại dẫn xuất đó là *thành công*. Những câu trả lời của một đích *G* cho chương trình *P* là các ràng buộc $\exists_{vars(G)} c$ trong đó có một dẫn xuất thành công từ *G* đến trạng thái cuối cùng với ràng buộc *c*.

**Ví dụ 3.1.** Xét một chương trình logic có ràng buộc để tính giai thừa của một số như sau:

($R_1$) *fac*(0, 1)

($R_2$) *fac*(*N*, *N* * *F*) ← N ≥ 1, fac(*N* – *1*, *F*)

Một dẫn xuất thành công từ đích *fac*(*1*, *X*) là:

$$\langle fac(1, X) \mid true \rangle$$

$$\Downarrow R_2$$

$$\langle \underline{1 = N}, X = N \times F, N \geq 1, fac(N-1, F) \mid true \rangle$$

$$\Downarrow$$

$$\langle \underline{X = N \times F}, N \geq 1, fac(N-1, F) \mid 1 = N \rangle$$

$$\Downarrow$$

$$\langle \underline{N \geq 1}, fac(N-1, F) \mid 1 = N \wedge X = N \times F \rangle$$

$$\Downarrow$$

$$\langle \underline{fac(N-1, F)} \mid 1 = N \wedge X = N \times F \wedge N \geq 1 \rangle$$

$$\Downarrow R_1$$

$$\langle \underline{N - 1 = 0}, F = 1 \mid 1 = N \wedge X = N \times F \wedge N \geq 1 \rangle$$

$$\Downarrow$$

$$\langle \underline{F = 1} \mid 1 = N \wedge X = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \rangle$$

$$\Downarrow$$

$$\langle W \mid 1 = N \wedge X = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \wedge F = 1 \rangle$$

Do các *biến trung gian* không được chú ý đến nên chúng được lượng hóa để đưa ra câu trả lời như sau: $\exists N \exists F(1 = N \wedge X = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \wedge F = 1)$, tương đương logic với *X = 1*.

### 3.2. Tính độc lập đối với quy tắc chọn literal

Trong phần này chúng ta sẽ chỉ ra rằng việc định giá một truy vấn theo kiểu trên xuống đối với chương trình logic có ràng buộc là độc lập với các quy tắc chọn literal.

**Định nghĩa 3.2.** Một *quy tắc chọn literal* $\mathcal{S}$ là một hàm mà với một dẫn xuất đã cho sẽ trả về một literal $L$ trong đích cuối cùng của dẫn xuất đó.

**Định nghĩa 3.3.** Một dẫn xuất được gọi là *thực hiện theo một quy tắc chọn* $\mathcal{S}$ nếu tất cả những lựa chọn của các nguyên tố chọn trong dẫn xuất đó đều được thực hiện theo $\mathcal{S}$. Nghĩa là, nếu ta có một dẫn xuất:

$$\langle G_1 \mid c_1 \rangle \Rightarrow \langle G_2 \mid c_2 \rangle \Rightarrow \ldots \Rightarrow \langle G_n \mid c_n \rangle$$

thì với mỗi $i = 1, \ldots, n$, literal được chọn từ trạng thái $\langle G_i \mid c_i \rangle$ sẽ là:

$$\mathcal{S}(\langle G_1 \mid c_1 \rangle \Rightarrow \ldots \Rightarrow \langle G_i \mid c_i \rangle)$$

**Định nghĩa 3.4.** Một hàm xử lý ràng buộc *solv* cho miền ràng buộc $\mathcal{C}$ là *hiệu quả* nếu với bất kỳ ràng buộc $c_1$ và $c_2$ nào từ $\mathcal{C}$ phải thỏa mãn 2 tính chất sau:

**Logic**: *N*ếu $c_1$ và $c_2$ tương đương logic với nhau thì những kết quả trả về cho hàm xử lý ràng buộc phải giống nhau đối với cả $c_1$ và $c_2$.

**Đơn điệu**: Nếu hàm xử lý ràng buộc thất bại với $c_1$ thì với bất kỳ $c_2$ nào chứa nhiều ràng buộc hơn so với $c_1$, hàm xử lý ràng buộc cũng sẽ thất bại với $c_2$.

**Bổ đề 3.1.** Cho $S$ là một trạng thái và $L$, $L'$ là các literal trong đích của $S$. Cho *solv* là một hàm xử lý ràng buộc hiệu quả và cho $S \Rightarrow S_1 \Rightarrow S'$ là một dẫn xuất không thất bại được xây dựng bằng cách sử dụng *solv* với $L$ được chọn đầu tiên, tiếp đến là $L'$. Lúc đó, ta sẽ có một dẫn xuất $S \Rightarrow S_2 \Rightarrow S''$ cũng được xây dựng từ *solv* với $L'$ được chọn đầu tiên, tiếp đến là $L$, sao cho $S'$ và $S''$ là đồng nhất với nhau bằng cách sắp xếp lại thứ tự các thành phần trong ràng buộc của chúng.

**Chứng minh.** Giả sử rằng $S$ là trạng thái $\langle L, L' \mid c \rangle$. Có bốn cách để $S$ có thể được biến đổi thành $S'$:

1. Nếu cả $L$ và $L'$ đều là các ràng buộc. Trong trường hợp này, trạng thái $S_1$ là $\langle L' \mid c \wedge L \rangle$ và trạng thái $S'$ là ‹W | $c \wedge L \wedge L'$›. Nếu chọn $S_2$ là $\langle L \mid c \wedge L' \rangle$ và $S''$ là ‹W | $c \wedge L' \wedge L$› thì $S \Rightarrow S_2 \Rightarrow S''$ là một dẫn xuất hợp lệ vì chúng ta biết rằng $solv(c \wedge L \wedge L') \neq false$. Ngoài ra do tính hiệu quả của hàm xử lý ràng buộc *solv*, nên $solv(c \wedge L') \neq false$ và $solv(c \wedge L' \wedge L) \neq false$.
2. Nếu cả $L$ và $L'$ là đều là các nguyên tố. Giả sử rằng $L$ có dạng $p(t_1, \ldots, t_n)$ và được biến đổi bằng cách sử dụng quy tắc đổi tên có dạng $p(s_1, \ldots, s_m) :- B$ và $L'$ có dạng $q(t'_1, \ldots, t'_{m'})$ và được biến đổi bằng cách sử dụng quy tắc đổi tên có dạng $q(s'_1, \ldots, s'_{m'})$. Lúc đó ta sẽ có $S_1$ là $\langle t_1 = s_1, \ldots, t_m = s_m, B, L' \mid c \rangle$ và $S'$ là

$\langle t_1 = s_1, ..., t_m = s_m, B, t_1^{'} = s_1^{'}, ..., t_{m'}^{'} = s_{m'}^{'} , B' | c \rangle$. Trong trường hợp này chúng ta chọn $S_2$ là:

$$\langle L, t_1^{'} = s_1^{'}, ..., t_{m'}^{'} = s_{m'}^{'}, B' | c \rangle$$

và *S"* sẽ là *S'*. Rõ ràng rằng $S \Rightarrow S_2 \Rightarrow S'$ là một dẫn xuất hợp lệ do các quy tắc đổi tên vẫn còn tách biệt với nhau.

3. Nếu *L* là một ràng buộc và *L'* là một nguyên tố: Kết hợp hai trường hợp 1 và 2 để chứng minh.
4. Nếu *L'* là một ràng buộc và *L* là một nguyên tố: Kết hợp hai trường hợp 1 và 2 để chứng minh. □

**Định lý 3.1.** (*Tính độc lập đối với quy tắc chọn literal*) Giả sử ta có một hàm xử lý ràng buộc hiệu quả, *P* là một chương trình logic có ràng buộc và *G* là một đích. Nếu có một dẫn xuất từ *G* với câu trả lời là *c*, thì với bất kỳ quy tắc chọn literal $\mathcal{S}$ nào, sẽ có một dẫn xuất có cùng chiều dài từ *G* thông qua $\mathcal{S}$ với câu trả lời là quá trình sắp xếp lại của *c*.

**Chứng minh.** *(Sử dụng phương pháp quy nạp)*

Ta có giả thuyết quy nạp: Nếu có một dẫn xuất thành công *D* của chiều dài *N* từ một trạng thái *S* đến trạng thái ‹W | *c*› thì bằng cách sử dụng một quy tắc chọn literal $\mathcal{S}$ sẽ có một dẫn xuất có cùng chiều dài từ *S* đến ‹W | *c*› , trong đó *c'* là một quá trình sắp xếp lại của *c*.

Việc chứng minh được thực hiện bằng phương pháp quy nạp trên chiều dài của *D*. Trong trường hợp cơ sở khi chiều dài của *D* bằng *0* thì trạng thái *S* sẽ là ‹W | *c*› , từ đó dễ dàng suy ra điều cần chứng minh.

Bây giờ chúng ta chứng minh bước quy nạp. Xét một dẫn xuất *D* có chiều dài *N + 1* như sau:

$$S \Rightarrow S_1 \Rightarrow \ldots \Rightarrow S_N \Rightarrow \text{‹W | } c\text{›}$$

Giả sử rằng chiến lược $\mathcal{S}$ chọn literal *L* trong dẫn xuất *S*. Khi *D* là một dẫn xuất thành công, thì mỗi literal trong *D* phải được chọn theo một cách nào đó. Vì vậy, khi *L* được chọn tại một thời điểm nào đó thì lúc đó sẽ có một phép biến đổi từ trạng thái $S_i$ đến $S_{i+1}$. Bằng cách áp dụng bổ đề 3.1 *i lần*, chúng ta có thể sắp xếp lại *D* để thu được một dẫn xuất *E* có dạng:

$$S \Rightarrow S_1^{'} \Rightarrow \ldots \Rightarrow S_N^{'} \Rightarrow \text{‹W | } c\text{"›}$$

trong đó *L* được chọn trong trạng thái *S* và *c"* là một quá trình sắp xếp lại của *c*. Từ giả thuyết quy nạp, có một dẫn xuất *E'* chiều dài *N* sử dụng $\mathcal{S}'$ từ $S_1^{'}$ đến ‹W | *c'*› , trong đó $\mathcal{S}'$ là một chiến lược lựa chọn literal mà nó chọn ra cùng literal trong E' như khi đã thực

hiện bởi $\mathcal{S}$ trong $S \Rightarrow E'$ và $c'$ là một quá trình sắp xếp lại của $c$" và vì vậy nó cũng là một quá trình sắp xếp lại của $c$. Vậy dẫn xuất $S \Rightarrow E'$ là dẫn xuất được yêu cầu. □

### 3.3. Cây suy dẫn và lỗi hữu hạn

Việc lựa chọn một quy tắc chọn literal có thể hình thành nên một "*cây suy dẫn*" được định nghĩa như sau:

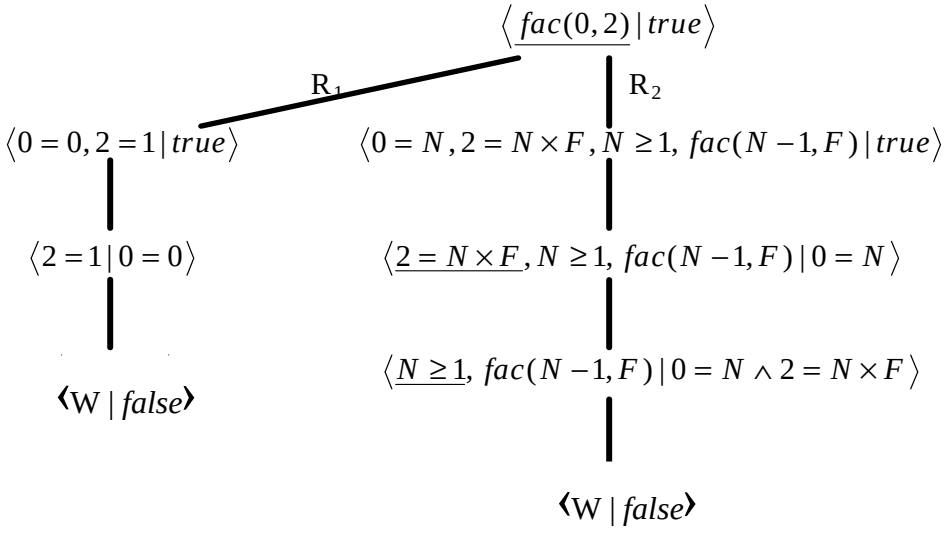
**Định nghĩa 3.5.** Cho $P$ chương trình logic, *cây suy dẫn* cho một đích $G$ với quy tắc chọn literal $\mathcal{S}$ là một cây với các trạng thái là các nút và được xây dựng như sau: *Gốc của cây* là một trạng thái $\langle G \mid true \rangle$, và *con của mỗi nút* trong cây là các trạng thái mà nó có thể biến đổi tại nơi mà literal chọn được chọn ra với một quy tắc chọn $\mathcal{S}$ nào đó.

Một cây suy dẫn biểu diễn tất cả các dẫn xuất từ một đích cho một quy tắc chọn literal cố định. *Một dẫn xuất thành công* được biểu diễn trong cây suy dẫn bởi một đường đi từ nút gốc đến một nút lá với *đích rỗng* và ràng buộc *khác false*. *Một dẫn xuất thất bại* được biểu diễn trong cây suy dẫn bởi một đường đi từ nút gốc đến nút lá với một *đích rỗng* và ràng buộc *bằng false*.

Ngoại trừ việc trả về các câu trả lời cho một đích, việc xử lý một chương trình logic có ràng buộc cũng sẽ trả về một câu trả lời đặc biệt "*no*" chỉ ra rằng đích "*bị thất bại*", nghĩa là với một quy tắc chọn literal cụ thể nào đó thì tất cả các dẫn xuất của đích đều thất bại.

**Định nghĩa 3.6.** Nếu một trạng thái hoặc một đích $G$ có một cây suy dẫn hữu hạn đối với một quy tắc chọn literal $\mathcal{S}$ và tất cả các dẫn xuất trong cây đều thất bại thì $G$ được gọi là một *lỗi hữu hạn* đối với $\mathcal{S}$.

**Ví dụ 3.2.** Xét ví dụ tính giai thừa ở trên. Ta có cây suy dẫn cho đích *fac*(*0, 2*) được xây dựng với một quy tắc chọn literal từ *trái sang phải* như bên dưới. Từ cây suy dẫn, ta thấy rằng, với quy tắc chọn literal này thì đích *fac*(*0, 2*) sẽ dẫn đến một lỗi hữu hạn.



Như như đã xét ở phần 3.1, chỉ với điều kiện hàm xử lý ràng buộc là hiệu quả thì những câu trả lời thu được từ một đích là độc lập với quy tắc chọn literal. Vậy, với trường hợp lỗi hữu hạn thì câu hỏi đặt ra là: "*Khi nào lỗi hữu hạn sẽ là độc lập với quy tắc chọn literal?*".

**Ví dụ 3.3.** Xét chương trình sau:

$p \leftarrow p$

và đích ($p$,1 = 2). Với một quy tắc chọn *từ trái sang phải* thì đích này có một dẫn xuất vô hạn, trong đó $p$ được viết lặp lại cho chính nó. Tuy nhiên, với quy tắc chọn *từ phải sang trái* thì đích có một dẫn xuất thất bại, nghĩa là đích cũng được xem là gặp lỗi hữu hạn.

Trong ví dụ trên, tính độc lập không thỏa mãn đối với lỗi hữu hạn bởi vì với một dẫn xuất vô hạn, một literal là nguyên nhân gây ra lỗi sẽ không bao giờ được chọn. Để khắc phục hạn chế trên, chúng ta cần một quy tắc chọn literal thỏa tính chất "*bình đẳng*".

**Định nghĩa 3.7.** Một quy tắc chọn literal $\mathcal{S}$ là *bình đẳng* nếu với mỗi dẫn xuất vô hạn được thực hiện theo quy tắc $\mathcal{S}$ thì bất kỳ literal nào trong dẫn xuất đó đều được chọn.

Như vậy, với một hàm xử lý ràng buộc hiệu quả và các quy tắc chọn literal thỏa tính bình đẳng, thì lỗi hữu hạn sẽ độc lập với quy tắc chọn literal. Định lý dưới đây sẽ chỉ rõ điều này.

**Định lý 3.2.** Cho một hàm xử lý ràng buộc hiệu quả, $P$ là một chương trình và $G$ là một đích. Giả sử rằng $G$ có một dẫn xuất với chiều dài vô hạn thông qua một quy tắc chọn literal $\mathcal{S}$. Lúc đó, $G$ cũng có một dẫn xuất với chiều dài vô hạn thông qua bất kỳ quy tắc chọn literal $\mathcal{S}'$ nào.

**Chứng minh.** Cho $D$ là một dẫn xuất có chiều dài vô hạn thông qua quy tắc chọn $\mathcal{S}$. Chúng ta định nghĩa một dãy các dẫn xuất bình đẳng vô hạn $D_0$, $D_1$, $D_2$, … sao cho với mỗi $N$, nếu $D_N$ là:

$S_0 \Rightarrow S_1 \Rightarrow \ldots \Rightarrow S_N \Rightarrow \ldots$

thì phần đầu của suy dẫn

$S_0 \Rightarrow S_1 \Rightarrow \ldots \Rightarrow S_N$

là một dẫn xuất từ G thông qua $\mathcal{S}'$. Giới hạn của dãy này là một dẫn xuất vô hạn từ G thông qua $\mathcal{S}'$.

Trong trường hợp cơ sở ($N$ = 0), lúc này dẫn xuất chính là $D$.

Bây giờ, chúng ta giả sử rằng $D_N$ là:

$S_0 \Rightarrow S_1 \Rightarrow \ldots \Rightarrow S_N \Rightarrow S_{N+1} \Rightarrow S_{N+2} \Rightarrow \ldots$

Cho literal $L$ được chọn bởi $\mathcal{S}'$ trong $S_N$. Do $D_N$ là fair, nên $L$ cũng được chọn tại một đoạn nào đó trong $D_N$, giả sử tại trạng thái $S_{N+I}$, trong đó $i \geq 0$. Bằng cách áp dụng bổ đề 3.1 *i lần*, chúng ta có thể sắp xếp lại $D_N$ để thu được một dẫn xuất $D_{N+1}$ có dạng:

$$S_0 \Rightarrow S_1 \Rightarrow \ldots \Rightarrow S_N \Rightarrow S'_{N+1} \Rightarrow S'_{N+2} \Rightarrow \ldots$$

trong đó $L$ được chọn trong trạng thái $S_N$. Lúc đó ta có

$$S_0 \Rightarrow S_1 \Rightarrow \ldots \Rightarrow S_N \Rightarrow S'_{N+1}$$

là một dẫn xuất từ G thông qua $\mathcal{S}'$. Cũng do $D_{N+1}$ là fair nên nó sắp xếp lại một chọn literal trong một dẫn xuất fair $D_N$. □

**IV. Kết luận**

Ngữ nghĩa thủ tục của CLP là sự mở rộng về ngữ nghĩa của chương trình logic khi tích hợp thêm ràng buộc vào chương trình. Cách tiếp cận ngữ nghĩa này được thực hiện thông qua các dẫn xuất từ đích và cây dẫn xuất. Chúng tôi cũng đã chỉ ra việc định giá một truy vấn theo kiểu trên xuống đối với chương trình logic có ràng buộc là độc lập với các quy tắc chọn literal. Trong lĩnh vực nghiên cứu ngữ nghĩa của CLP, lớp các miền ràng buộc khác nhau sẽ phát sinh các ngôn ngữ CLP khác nhau. Tuy nhiên, trong khuôn khổ bài báo chúng tôi không đề cập đến vấn đề này.

**TÀI LIỆU THAM KHẢO**


1. Francois Fages. *Constraint logic programming*, published in French by Ellipse, 1996.
2. J. Jaffar & J – L. Lassez. *Constraint logic programming, in Proc*. Fourteenth Ann. ACM Symp. Principles of Programming Languages, (1987), 111-119.
3. J. Cohen. *Constraint logic programming Languages*, CACM, 33, (1990), 52-68.
4. Joxan Jaffar, Michael J. Maher, Kim Marriott, Peter J. Stuckey. *The Semantics of Constraint Logic Programs*, J. Log. Program, 37(1-3), (1998), 1 - 46.
5. Henk Vandecasteele. *Constraint Logic Programming An Informal Introduction*, Department of Computer Science, K.U.Leuven Celestijnenlaan 200A, B-3001 Heverlee, Belgium, 1993.
6. M. Gabbrielli, M.G. Dore and G. Levi. *Observable semantics for Constraint Logic Programs*, Journal of Logic and Computation, 5(2), (1995), 133-171.
7. M. Gabbrielli, G.Levi. *Modeling answer constraints in Constraint Logic Programming*, Proc. 18th International Conference on Logic Programming, (1991), 238-252.
8. P.van Hentenryck & Y. Deville. *Constraint logic programming, Journal of Logic programming*, 16, 3&4, 1991.

9. R. Giacobazzi, S. Debray, G. Levy. *A generalized semantics for constraint logic programs*, In Internatzonal Conference on Fifth Generation Computing Systems, 1992.
10. Vladimir Lifschitz. *Foundations of logic programming, Principles of knowledge representation*, CSLI Publications, 1996.

## OPERATIONAL SEMANTICS OF CONSTRAINT LOGIC PROGRAMS

*Truong Cong Tuan*
*College of Sciences, Hue University*
*Tran Thi Ngoc Trang*
*Centre of Information and Technology, Hue University*

### SUMMARY

*Constraint Logic Programming (CLP) is a new approach in logic programming which was discovered by combining the declarativity of logic programming with the efficiency of constraint solving. In this paper, we mainly discuss operational semantics of Constraint Logic Programs via derivations and derivation trees from goal, and mention some criteria of constraint solver to ensure that this semantics is independent of literal selection strategies in goal.*