计算机科学 1994Vol. 21№. 4

约束逻辑程序设计综述

党华锐 郑守淇

(西安交通大学计算机科学与工程系 西安 710049)

摘 要 Constraint Logic Programming(CLP)is a new method of logic programming which has been developed rapidly during 1980's. It combines the declaretive expressions of logic programming and the effectiveness of constraint solver. This paper describes the foundation, syntax and formal semantics of CLP. Then some instances of CLP languages are given and compared. Finally, some trends of researching in CLP are introduced.

关键词 Constraint, Constraint Logic Programming, Semantics

一、引 言

约束逻辑程序设计(Constraint Logic Programming, CLP)是基于人工智能(AI)中约束满足问题(Constraint Satisfaction Problem, CSP)模型的一种程序设计风范。CLP 是逻辑程序设计(LP)的一种推广，是八十年代发展起来的一种新的逻辑程序设计方法。由于它继承了 LP 简单易懂的说明性描述方法并结合了 CSP 在求解问题时的效率，使它在解决很多 AI 问题(如组合问题、资源分配、事务安排等)时有不凡的表现。更由于 AI 领域中绝大多数问题可以用 CLP 来表示，所以这一方法已引起了人们的广泛注意，并在八十年代后期得以迅速发展。

CLP 早期的研究始于 Colmerauer[1]，Jaffar & Lassez[1]等人的工作，此后有许多研究单位和个人都投入到这一领域中来，设计并实现了一些基于不同域的 CLP 语言。其中比较有影响的有 Prolog 的发明人 Colmerauer 主持开发的 Prolog-Ⅲ[1]，IBM 的 Jaffar 等人的 CLP(R)[1,4]，日本的 CAL，新加坡的 ConstraintLisp[4]，欧洲 ECRC 的 CHIP 等。虽然这些系统是在不同的域上采用不同的方法实现的，但它们都有相同的 CLP 语义。

以下本文首先就 CLP 的基础、语法和语义进行讨论，然后列举几个典型系统的实现，最后给出目前这一领域的一些研究方向。

二、CLP 的理论基础

CLP 是基于 AI 中约束满足问题(CSP)模型的，所以本节首先对 CSP 进行简单描述。

1. 约束满足问题[4]

标准的约束满足问题由以下三部分组成：

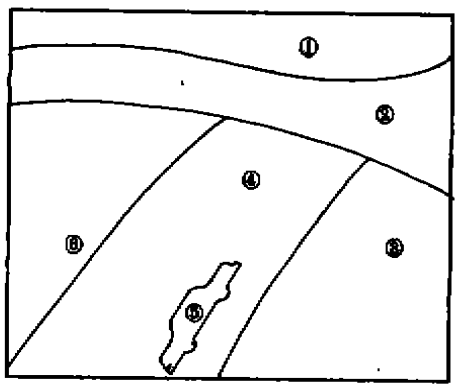
①变量，有一个含有 n 个变元的有限集合：$V=\{v_1, v_2, \cdots, v_n\}$。

②变量的取值。每一个变元 v_i 的取值都有一个与之对应的域 D_i。

③约束。有一个有关变量约束的有限集合：$C=\{c_1, c_2, \cdots, c_m\}$。

求解的目标是找出每个变量所有可能的指派，以使 C 中的每一个约束都得到满足。

下面给出一个具体的例子[1]，称为图象标识问题。设有一幅图象如下：



党华锐 讲师，博士生。郑守淇 教授，博士生导师。主要的研究方向为新型计算机系统结构与人工智能。

其中共有 6 个待标识的区域，记为：$\Omega=\{R_1,R_2,R_3,R_4,R_5,R_6\}$。

待标识的对象共有 5 个，记为：U＝{car，road，trees，grass，sky}。

目标是要将 U 中的元素正确标到图中对应的区域中，显然，此问题的一般解共有 $5^6=15625$ 种。

今按相邻区域的关系(adjacent_to)给其加上一组约束如下：

```
highest(sky)
moving(car)
adjacent_to(car,road)
adjacent_to(road,car)
adjacent_to(road,grass)
adjacent_to(grass,road)
adjacent_to(road,trees)
adjacent_to(trees,road)
adjacent_to(sky,trees)
adjacent_to(trees,sky)
adjacent_to(grass,trees)
adjacent_to(trees,grass)
```

文[8]中用 Prolog 模拟给出了一种解法，它将 5^6 减为 6 种，即得到了唯一的合理标识。其结果如下：

R_1 is bound to sky
R_2 is bound to trees
R_3 is bound to grass
R_4 is bound to road
R_5 is bound to car
R_6 is bound to grass

由此可以看出，采用 CSP 的表示不仅程序更接近于自然语言，而且其问题空间也得到了有效的裁减。因此这一方法近年来已引起了越来越多人的关注。

2. CLP 的语法[2,5]

CLP 的语法可采用自顶向下的方法描述为：

①一个 CLP 程序是由子句的序列构成的。子句的形式为：$H\leftarrow c_1,\cdots,c_m\diamond B_1,\cdots,B_n$ 其中：$H,B_1,\cdots,B_n$ 为原子公式；$c_1,\cdots,c_m$ 为约束；$m\geqslant 0,n\geqslant 0$，特别地，当 $m=0$，则子句就是一般的 Horn 子句形式：$H\leftarrow B_1,\cdots,B_n$；而当 $n=0$ 时，子句表示一种事实。

②一个 CLP 程序求解的目标可用不含 H 和 $c_i(i=1,\cdots,m)$ 子句的表示为：$\leftarrow B_1,\cdots,B_n$。这是用反证法表示的[15]，即若由否定的结论推出了空子句(表示矛盾)，则可证明结论的正确性。

③一个原子公式是形如：$p(t_1,\cdots,t_n)$ 的表达式(n 个变元的谓词)。其中 $t_i(i=1,\cdots,n)$ 称为项。

④项定义为一个变量，或一个常量，或一个作用于 n 个项的一个函数：$f(t_1,\cdots,t_n)$。

3. CLP 的语义[2]

一个 CLP 程序的含义可以用说明性描述和其操作语义描述给出。为了使读者能较容易地了解其语义，我们先用自然语言给出一般性说明再给出其语义的形式定义。

3.1 CLP 程序的说明

一个 CLP 程序的含义可以分两部分说明。

①CLP 程序的子句 $H\leftarrow c_1,\cdots,c_m\diamond B_1,\cdots,B_n$，其意为"对变量的任何指派，如果 $c_1,\cdots,c_m$ 和 $B_1,\cdots,B_n$ 都为真，则 H 也为真"。另外，从以上表示可看出存在有若干子句具有相同头部的情况，因此 CLP 的求解算法是非确定性的，即对同一原子可能存在多种解法。

②对目标的求解是一组约束的合取，且使此约束为真的所有指派也是目标的解。

上述描述用一阶语言的表示方法可写成：$P,\mathscr{M}\models(\forall)(\sigma\rightarrow G)$。其中：P 为 CLP 的程序，而 $\mathscr{M}$ 为此约束问题的一种解释；σ 为一组约束的合取；G 为待求的目标。

3.2 CLP 的操作语义

从 CLP 的语法中可以看到，其子句的执行是"要求解 H，则先在 $c_1,\cdots,c_m$ 满足的前提下求解 $B_1,\cdots,B_n$"。于是，CLP 的操作语义可描述如下：

3.2.1 非形式描述

定义 1 ［状态］ 系统的状态可表示为：

①目标部分。待求解的谓词合取。

②约束存贮。程序执行至此时已积累的约束的合取。状态可用$\langle G,\sigma\rangle$表示。其中 G 为目标部分，σ 表示约束存贮部分。

定义 2 ［计算步骤］

1)在 G 中选一个原子公式；

2)在程序上下文中选一子句，使本子句的约束和 G 中所有的约束都得到满足；

3)修改 G 和 σ 得出一个新状态。新状态的构成如下：将上述从程序上下文中选中的子句代替 G 中相应的原子公式，同时将此子句的约束部分加入 σ。

在以上计算中，核心步为步骤 2。因为约束存贮部分已积累了若干满足条件的子句，所以 CLP 的执行不必每一次都重新检验全部的子句是否满足，而只需检验新并入的子句是否满足，即以前积累的结果可以传递给下一步的计算。从此意义上讲约束满足求解是一种可增长的算法。

定义 3 ［初始状态］ G 为待求目标，σ 为空。

定义 4 ［终结状态］ 以下的状态称为终结状态：①G 为空；②没有新的可选子句以产生新状态。

定义 5 ［计算］ 一个 CLP 的执行(也称计算)是由一系列计算步组成的。计算是由初始状态开始，

最后达到终结状态或发散；若一计算能在有限的计算步内使 G 为空，则称计算成功，否则称为失败。

例　一个处理不等式的 CLP 如下：

1)P(X,Y)←X≥Z+3,Y≤Z◇q(X,Y,Z).

2)q(X,Y,Z)←r(X,Y).

3)q(X,Y,Z)←Z≥Y+2◇.

4)r(X,Y)←X≤Y+2◇.

其计算过程如下：

①(初始)　〈P(X,Y),ε〉。

②(选 clause 1)　〈q(X,Y,Z),X≥Z+3&Y≤Z〉。

③(选 clause 3)(注：◇后无符号，表示空原子)　〈ε,X≥Z+3&Y≤Z&Z≥Y+2〉。

因为 G=ε，所以计算结束，其结果用更直接的表示可表示为：X≥Y+5。

3.2.2　CLP 类语言操作语义的形式描述　CLP 操作语义的形式描述是基于变换系统[7]模型的。

定义 6　一个变换系统是一个三元组(Γ,T,↦)。其中，Γ 是构形(Configurations)的集合，T⊆Γ 是终结构形的集合；↦⊆Γ×Γ 是 Γ 上的二元关系，且满足：∀γ∈T,∀γ∈Γ,γ↦γ'。在 CLP 中，构形被用来代表计算状态〈G,σ〉，而 γ↦γ' 意为"γ 非确定性地约简为 γ'"。

变换的规则可表示如下：

〈condition 1〉

⋮

〈condition n〉

γ↦γ'

意为当条件 1 至 n 都成立时，状态可从 γ 变换为 γ'。

在定义 CLP 的操作语义时，只需定义一条变换规则。

[变换规则]

1. $p(u_1,\cdots,u_n)\leftarrow\sigma' B_1,\cdots,B_m\in P$

2. $\mathscr{B}\models(\exists)(\sigma\wedge\sigma'\wedge t_1=u_1\wedge\cdots\wedge t_n=u_n)$

$\langle p(t_1,\cdots,t_n)\&G,\sigma\rangle\mapsto(B_1\&\cdots\&B_m\&G,\sigma\&\sigma'\&t_1=u_1\&\cdots\&t_n=u_n\rangle$

其含义为：

$p(u_1,\cdots,u_n)$ 为 G 中被选中的原子公式；

p 为程序的上下文；

(∃)(Ψ)表示在 P 中存在一个子句 Ψ，(它使本子句的约束 σ' 和约束存贮中的 σ 都满足)，并将原子公式中的变元 u_i 换为 t 以使变元不同名；

当以上条件成立时，一个计算步便可进行，其结果为将 G 中的 $p(t_1,\cdots,t_n)$ 用 $B_1,\cdots,B_m$ 代替，并将约束 σ' 加入约束存贮中。

CLP 具体的操作语义可用成功、发散和失败集合给出。为此引入如下记号：

P⊢ 表示在程序 P 上下文中进行的变换；

$\overset{*}{\mapsto}$表示 ↦的传递闭包。在 P 中，一个 γ 称为发散的，指存在一个变换的无限序列，使 $P\vdash\gamma\mapsto\gamma_1\mapsto\cdots\mapsto\gamma_i\mapsto\cdots$。

定义 7　程序 P 的成功集 SS，发散集 DS 和失败集 FS 定义为：

$SS[P]=\{G\mid\vdash G\overset{*}{\mapsto}\sigma\}$

DS[P]={G|G 在 P 中是发散的}

$FS[P]=\{G\mid G\notin SS[P]\cup DS[P]\}$。

计算的结果可定义为：

$RES[P,G]=\{\sigma'\mid P\vdash G\overset{*}{\mapsto}\sigma\}$

为了达到上述语义 RES，则 CLP 中应存在一个约束求解算法，使得：

若 $\mathscr{B}\models(\exists)(\sigma)$

则算法返回为 true(求解成功)，

否则返回 false。

与操作语义有关的两个定理为：

定理 1　[可靠性(soundness)定理]　设 P 为程序，G 是目标，σ 为约束，则由 σ∈RES[P,G]可推出 $\mathscr{P},\mathscr{B}\models(\forall)(\sigma\rightarrow G)$；由 G∈FS[P]可推出 $\mathscr{P},\mathscr{B}\models\neg(\exists)(G)$。

定理 2　[完备性(strong completeness)定理]　设 P 为程序，G 是目标，σ 是约束，若 $\mathscr{P},\mathscr{B}\models(\forall)(\sigma\rightarrow G)$ 且 $\mathscr{B}\models(\exists)(\sigma)$，则存在一组约束 $\sigma_1,\cdots,\sigma_n\in RES[P,G]$ 使 $\mathscr{B}\models(\forall)(\sigma\rightarrow(\exists y_1,\cdots,y_m)(\sigma_1\vee\cdots\vee\sigma_n))$。其中 $y_1,\cdots,y_m$ 是出现在 $\sigma_1,\cdots,\sigma_n$ 中，但不出现在 σ 中的一组变元。

三、CLP 语言的实现

以上就一般的 CLP 进行了讨论，而真正具体实现的 CLP 语言是基于不同域上的约束满足求解。从实现方法上这种语言可分为两类，一是直接基于 CLP 语义而设计的"纯"CLP 语言，如 CHIP(Dincbas et al.,1988; Van Hentenryck,1989)，CLP(R)(Jaffar et al.,1990)，CAL(Aiba et al.,1988)等；另一类是将 CSP 与其它 AI 语言相结合而成的 CLP 语言，是在保留原语言的基础上加上了 CSP 的解释系统，所以这类语言既保留了原语言的特色，又为其增加了求解问题的新手段。其中有代表性的如在 Prolog 基础上实现的 Prolog-Ⅱ(Colmerauer,1990)和在 Lisp 基础上实现的 ConstraintLisp(Liu et al.,1991)等。

虽然以上系统的问题域各不相同，但在具体实现时有两点是一样的，一是力求 CSP 算法有较好的平均复杂度，二是将该算法设计成可增长的，即新并入的约束将不会导致对所有原已积累的约束重新进行满足性检验。

以下简要介绍其中几个系统的实现情况，更多的信息请参考有关的文献。

1. Prolog-Ⅱ[1] 是由法国马赛人工智能中心的 Alain Colmerauer 教授主持开发的，他是著名的 Prolog 语言的发明人，Colmerauer 是最早提出 CLP 概念的人之一，他主持的小组首先在 Prolog 的基础上将 CSP 引入，在 1982 年左右完成了 Prolog-Ⅰ的设计，其约束域主要是无穷树，其运算是等式和不等式；在此基础上进一步实现的 Prolog-Ⅱ将问题的域进一步扩展到有理数、字符串和布尔值。

Prolog-Ⅱ的语法形式如下：

$$t_0 \leftarrow t_1, \cdots, t_n.S$$

其中，$t_0, t_1, \cdots, t_n$ 为项；S 为一组约束组成的表；上述公式中所有变量的量词为全称量词；其意为“当 $t_1, \cdots, t_n$ 和 S 中所有约束都满足时 t_0 为真”。

Prolog-Ⅱ的操作语义可由 Prolog 给出。

作为说明，一个短的 Prolog-Ⅱ程序形式如下，它处理的是低热配餐问题。

```
LightMeal(a,m,d)→
    Appetizer(a,i)Main(m,j)
    Dessert(d,k).
    {i≥0,j≥0,k≥0,i+j+k≤10};
Main(m,i)→Meat(m,i);
Main(m,i)→Fish(m,i);
Appetizer(radishes,1)→;
Appetizer(salad,6)→;
Meat(beef,5)→;
Meat(pork,7)→;
Fish(sole,2)→;
Fish(tuna,4)→;
Dessert(fruit,2)→;
Dessert(icecream,6)→.
```

2. CLP(R)[4] 是由 IBM T. J. Watson 研究中心的 Jaffer 和 Michaylov 等人开发的一个 CLP 语言，其基础是 Jaffer 和 Lassez 定义的基于规则约束的程序设计理论[3]。

CLP(R)选择的域是实数和有穷树。其主要的特点是表示的一致性，即 CLP(R)的输入、输出和程序都是用约束来表示的。另外，它还可以处理非线性的 CSP，而 Prolog-Ⅱ只能处理线性的 CSP。在实现技术上，CLP(R)采用了“延迟/唤醒”机制使约束求解过程成为动态的。

在 CLP(R)中，程序是由如下形式的规则构成的：

$$A_0 :- a_1, a_2, \cdots, a_n$$

其中：A_0 称为规则的头部，其余部分称为规则体；$a_i (1 \leqslant i \leqslant n)$ 是原子公式或是约束条件。

例如，一个求斐波纳奇数的 CLP(R)程序如下：

```
fib(0,1).
fib(1,1).
fib(N,X1+X2) :-
            N>1,
            fib(N-1,X1),
            fib(N-2,X2).
```

3. ConstraintLisp[8] 是新加坡国家计算机委员会下属的信息技术研究所知识系统实验室的 Bing Liu 等人研制的。它是在 Lisp 的基础上实现的一个面向对象的约束逻辑程序设计语言。由于它将面向对象的技术引入 CLP，因而兼备了 CSP 的解题效率的 OO 的问题抽象和描述能力。在实现上它利用 CLOS 来描述对象，同时 ConstraintLisp 也允许用户用 CLOS 和 CommonLisp 来书写自己的程序。所以可以认为它是一种面向对象 Lisp 的推广。从语法上讲 ConstraintLisp 的语句是用 Lisp 形式给出的，其操作语义则可用 CLOS 描述。

四、当前 CLP 的研究方向

CLP 作为一种新的逻辑程序设计方法已在世界范围内引起了人们广泛的注意。当前，与 CLP 相关的主要研究方向包括：

1. 新的 CLP 语言的设计与实现；
2. 不同域上的约束求解算法及其复杂性分析；
3. CLP 与其它 AI 语言的关系；
4. CLP 并行求解算法的研究；
5. 面向对象技术在 CLP 设计与实现中的应用；
6. CLP 语言的应用等。

参考文献

[1] Colmerauer, A., Opening the Prolog-Ⅱ Universe. BYTE Mag. 12(9). 1987

[2] Hentenryck, P. V., Constraint Logic Programming. The Knowledge Engin. Review, Vol. 6, No. 3, Sept. 1991

[3] Jaffer & Lassez, Constraint Logic Programming, POPL-87, Munich Germany, 1987

[4] Jaffer et al., The CLP(R) Language and System, ACM Trans. On Prog. Lang. & System, Vol. 14, No. 3 (July, 1992)

[5] Kowalski, R., Logic for Problem Solving, Elsevier North Holland, Inc., 1979

[6] Lin, B., ConstraintLisp: A Object-oriented Constraint Programming Language, SIGPLAN Not, Vol. 27, No. 11, 1992

[7] Plotkin, G. D., A Structural Approach to Operational Semantics, Tech. Report DAIMI FN-19, 1981, CS Department

[8] Schalkoff, R. J., Artificial Intelligence: An Engineering Approach, McGraw-Hill, Inc., 1990