PDL使用手册

pdlteam@outlook.com

2020 年 3 月 12 日

目录

第 1 章　PDL简介

PDL（Problem Description Language)是一种用于组合优化问题建模的语言。使用PDL时，用户只需要列举变量之间的约束关系，而无需指明如何求解该问题——PDL附带的生成工具将根据约束的性质自动生成可行的求解算法及相应的源代码。

PDL支持各种常用的运算符与数学函数，允许用户使用接近于数学公式甚至自然语言的形式描述问题。

相比于传统的使用程序设计语言编写程序求解问题，使用PDL生成工具自动生成程序有诸多好处：

- 用户只需具备基本的数学知识即可完成求解工作，无需任何编程基础和算法知识；
- PDL描述可读性强，便于交流及查错；
- 用户只需考虑如何建模，而无需在思考如何求解问题上花费大量精力和时间；
- 使用自动工具生成程序，可以避免在编程过程中因人为失误产生bug；
- 用户可以在自动生成的程序的基础上进行二次开发。

第 2 章　PDL实例

本节呈现了两个经典问题的PDL描述示例，具体的细节将在下一章详细说明。**单词间的空白符（空格、换行、制表符等）可以随意使用。**

2.1　最大公约数

求A和B的最大公约数。

```
#input
        A of int in [1,10^6];
        B of int in [1,10^6];

#required
        G of int in [0,?];
        X of int;
        Y of int;
        A = G * X;
        B = G * Y;

#objective
        maximize G;
```

2.2　0-1背包问题

选取若干物品放入背包，在物品总重量和不超过背包容量的条件下，物品总价值和尽可能大。

```
#input
        N of int in [1,100];
        capacity of int in [1,10000];
        profits of (int in [1,1000])[1~N];
        weights of (int in [1,1000])[1~N];

#required
        knapsack of (int in [1,N]){};
        summation
            [weights[i] : forall i (i in knapsack)]
            <= capacity;

#objective
        maximize summation
            [profits[i] : forall i (i in knapsack)];

#output
        knapsack;
```

第 3 章　PDL语法

3.1　基本框架

一个PDL模型由四部分组成：#input，#required，#objetive和#output（**“#”后不能有空格**）。每个部分均可省略，但**#objetive和#output至少存在一个**。**所有语句均以分号（;）作为结束**。下面将以0-1背包问题的描述来说明PDL的基本框架。

3.1.1　#input部分

定义问题的所有输入变量，包括变量名、类型和取值范围等。**注意：PDL生成工具会按照#input中列出的变量顺序依次生成相应的读入语句，因此务必保证列出顺序与实际输入顺序一致。**

在示例中，#input部分列出了该问题的四个输入变量，分别是两个整数变量N和$capacity$，以及两个整数数组变量$profits$和$weights$；同时，定义了各变量的类型和取值范围，数组变量还需定义下标范围。

3.1.2　#required部分

列举变量之间的关系，由若干陈述（statement）组成。陈述分为两类：

- 定义中间变量，格式与#input部分相同；
- 一个约束关系，用一个bool类型的表达式表示。

在示例中，#required部分首先定义了一个集合变量$knapsack$，即最终选取物品的下标集合。之后，描述了$knapsack$需要满足的一个约束条件——选取物品的总重量和不超过$capacity$。

3.1.3 #objective部分

一个目标函数（objective function）。为最大化（maximize）/最小化（minimize）某个表达式的值。特别地，如果问题仅仅要求找到一组符合所有约束的可行解，则该部分为空。

注意：默认情况下，最终的目标函数值将被生成的程序第一个输出。在maximize/minimize前加“@”可禁止输出该值。

在示例中，求解目标为最大化表达式选取物品的总价值和。

3.1.4 #output部分

若需要输出其他结果，则可以在此部分依次列出相应的表达式。同样的，列出顺序与实际输出顺序必须保持一致。

在示例中，要求额外输出最终选取物品的下标集合。

3.2 变量定义

在PDL中，使用如下语句定义输入变量或中间变量：

```
<标识符> of <类型>
```

3.2.1 标识符

变量的标识符是由一系列字母（区分大小写）、数字及下划线组成的字符串，且必须以字母开头。一些保留字不可以作为标识符，包括:

```
int, real, bool, char, function, of, in, true, false,
and, or, not, xor, mod, if, else, forall, exists,
summation, product, count, min, max, minimize, maximize
```

为避免冲突，**目标程序语言（C/C++/Java等）的保留字亦不可使用**，如`return`，`main`等。

3.2.2 类型

PDL支持三种基本类型和两种复合类型。

基本类型包括：整型int，实型real，布尔型bool。可对基本类型的取值范围做限定：假设基本类型为T，取值范围在lb至ub之间，则可表示为T in [lb,ub]。若取值范围的某一侧没有限制，可以用?表示。

复合类型包括：

- 数组：元素类型为T的数组可表示为T[]，例如：整型数组int[]，二维布尔型数组bool[][]。若需要限定数组下标的范围在li至ui之间，则可表示为T[li～ui]。同样可以用?表示某一侧没有限制。
- 集合：元素类型为T的集合可表示为T{}。集合元素无序且不相同。

3.2.3 变量定义示例

以下是一些常见的变量类型定义：

```
// 整型变量
i of int
// 限定取值范围的整型变量
n of int in [1,10^5]
// 整型数组
a of int[]
// 限定下标范围的整型数组（范围可以是表达式）
b of int[1～2*n]
// 限定下标范围及各元素取值范围的整型数组
c of (int in [1,10^6])[1～100]
// 多维数组，各维下标范围不同
d of (int in [1,100])[0～100][-5～5]
// 整型集合
s of int{}
// 限定元素取值范围的整型集合
t of (int in [1,n]){}
```

3.3 基本运算

PDL支持基本类型上的各种数学运算符，按照优先级从高到低为：

1. 括号()
2. 幂运算^
3. 乘*，除/，取余mod
4. 加+，减-
5. 类型判断of，取值范围判断in
6. 非not，与and
7. 或or，异或xor
8. 比较符号：=，!=，>，<，>=，<=

3.4 复合运算

针对复合类型，PDL内置如下运算：

- 元素选择：数组元素选择[]。优先级仅次于括号。示例：a[5]，a[i+1]，其中a是一个数组或元组，元组中元素编号从1开始。
- 集合运算：交*，并+，差-，优先级与相同符号的基本运算一致。
- 集合比较：相等=，不等!=，被真包含>，真包含<，被包含>=，包含<=，优先级同比较符号。
- 属于运算in，优先级同比较符号。表达式e in C返回一个bool值，表示集合类型变量C中是否存在元素e。
- 聚集运算：连加summation()，连乘product()，统计数量count()，最大值max()，最小值min()。这些运算的参数可以是数组或集合。例如：summation(a)计算数组a所有元素之和，max(s)得到集合s中的最大元素。

3.5 量词

PDL支持一阶谓词逻辑中的两个量词∃和∀，分别表示为exists和forall。

3.5.1 存在量词∃

存在量词表达式语法：

```
exists (<变量列表v1,v2,...>) (<bool表达式>)
```

该表达式返回一个bool值，表示能否为变量列表中的变量赋值，使得后面bool表达式为真。

注意：一个约束本身就暗含“存在”关系，因此exists表达式不能作为单独的约束，只能以子表达式的形式出现在更为复杂的表达式中。

示例：判断数组a是否存在逆序对。

```
exists (i,j) ((i<j) and (a[i]>a[j]))
```

3.5.2 全称量词∀

全称量词表达式语法：

```
[<表达式> : forall (<变量列表v1,v2,...>) (<bool表达式>)]
```

该语句枚举所有满足后面bool表达式的变量取值组合，分别带入前面的表达式得到每个元素值，所有元素构成最终的数组。如果变量列表只有一个变量，可以省略括号。

示例：选取数组a中所有下标属于集合s的元素。

```
[ a[i] : forall i (i in s) ]
```

3.6 特殊约束关系

在#required部分，PDL提供了两种特殊的约束语句来简化描述，并为其提供了针对性的优化。

3.6.1 forall约束

对于一系列形式相似的约束，可用forall约束来描述。语法：

```
<bool表达式> : forall (<变量列表v1,v2,...>) (<bool表达式>)
```

该语句枚举所有满足后面bool表达式的变量取值组合，分别带入前面的bool表达式得到一组约束。**注意：与全称量词生成数组的语法基本相似，区别是最外层没有方括号“[ ]”且最前面的表达式是bool类型的。**

示例1：要求数组a中元素单调递增。

```
a[i] < a[i+1] : forall i ( 1 <= i and i < N )
```

注意：若变量列表中的变量取值范围可直接由上下文推断而无须额外约束，forall后面的bool表达式可以省略。上例可简写成：

```
a[i] < a[i+1] : forall i
```

示例2：要求数组a中元素两两不同。

```
a[i] != a[j] : forall (i, j) (i != j)
```

3.6.2 alldiff约束

若要求一个数组内元素互不相同，可使用alldiff约束描述，语法：

```
alldiff <数组表达式>
```

其中数组表达式可以是一个数组变量，或是一个由全称量词生成的数组。

示例：

```
alldiff a
alldiff [ a[i] : forall i (i in s) ]
```

第一个约束是数组a中元素互不相同，第二个约束是数组a中下标属于集合s的元素互不相同。

第 4 章　常见错误

本章收集了以往用户在使用过程中遇到的一些典型错误、可能的原因以及解决办法，供参考。

4.1　编译错误 Compile Error

编译错误可分为四类：语法错误、输出错误、类型错误和语义错误。

4.1.1　语法错误

编译信息为：

```
Parse Error!
Encountered ... at line X, column Y.
...
```

遇到此类错误，请仔细检查X行Y列附近的语句是否符合语法。

4.1.2　输出错误

编译信息为：

```
No Output Error!
```

这由于同时缺失了#objective和#output部分，导致程序无输出结果。

4.1.3　类型错误

编译信息为：

```
Type Error!
...
```

造成此类错误的原因是子表达式的类型不符合操作符的要求。请检查各表达式是否正确。

4.1.4 语义错误

编译信息为：

```
Cannot be enumerated!
Unbounded variables: ...
```

导致此类错误的原因一般是缺少了对列出变量取值范围的必要限制，或取值范围设置错误。可通过加强/修正变量取值范围来解决。**注意：有时变量名错误也有可能产生此类错误。**

4.2 其他错误

其他错误包括答案错误（Wrong Answer），超时（Time Limit Exceeded）等，错误原因一般是PDL描述与题意不完全相符。建议解决步骤：

1. 检查对题意的理解是否正确；
2. 检查是否遗漏某些重要约束条件；
3. 检查PDL描述细节是否有误。

以下列举了PDL描述细节的常见错误，在检查第三步时可以重点关注：

- 变量取值范围缺失或错误
- 变量名不一致
- 比较符号错误，例如“<=”写成“<”
- 输入/输出顺序与题目要求不符

此外，还可以**将生成的程序拷贝到本地运行，利用样例输入输出来定位错误**。