2003年度　卒業論文審査会

制約プログラミングを用いたクラス・ライブラリの設計

東京理科大学工学部第二部
経営工学科
5300414　野村紀匡
(沼田研究室)

発表構成

1. はじめに
2. 制約プログラミングのメリット，現状
3. 研究目的・目標
4. 機能設計，クラス設計
5. 設計の評価
6. まとめ

1.1 経営計画立案時のニーズ

素早く，実行可能性のある経営計画を立案したい

様々な制約を考慮しながらの立案は，人手だけでは困難

1.2 従来の経営計画問題解決

- 線形計画問題，非線形計画問題等に定式化できる問題
  - 問題解決支援ツールとして，ソルバーが適用できる
- 日々新たに生じる非定型的な問題
  - ソルバーは適用できない
  - 刻々と変化する経営環境に柔軟に対応できない

1.3 制約プログラミングの登場

オブジェクト指向プログラミング

・変数をオブジェクトとした自立的問題表現

・過去の資産の再利用

制約論理プログラミング

・充足解の列挙[3]

制約充足問題の枠組み

・問題記述の標準化[1]

制約プログラミング

2.1 制約プログラミングのメリット

- 宣言的に記述する
  - 制約充足問題を，変数と変数間の制約の組み合わせとして宣言的に記述する
- 問題の記述と解法を分離する

↓

- 問題の記述に専念できる
- 解法の再利用が可能

2.2 制約プログラミングのメリット～例

- 例題: 覆面算

```
    S E N D
+   M O R E
-----------
  M O N E Y
```

  - 上記を満足する S から Y を求める
  - ただし，各記号は 0 ～ 9 を表し，異なる文字は異なる数となるものとする[3]

2.2 制約プログラミングのメリット～例

- 制約プログラミングを使わずに記述する

```
Private Sub solveSendMoreMoney1()
        ‘ 変数の宣言
        Dim S As Integer = 0
        Dim E As Integer = 0
        Dim N As Integer = 0
        Dim D As Integer = 0
        Dim M As Integer = 0
        Dim O As Integer = 0
        Dim R As Integer = 0
        Dim Y As Integer = 0
        (続く)
```

2.2 制約プログラミングのメリット～例

```
For S = 1 To 9
    For E = 0 To 9
        For N = 0 To 9
            For D = 0 To 9
                For M = 1 To 9
                    For O = 0 To 9
                        For R = 0 To 9
                            For Y = 0 To 9
                                If (S <> E AndAlso S <> N AndAlso S <> D AndAlso _
                                    S <> M AndAlso S <> O AndAlso S <> R AndAlso _
                                    S <> Y AndAlso E <> N AndAlso E <> D AndAlso _
                                    E <> N AndAlso E <> M AndAlso E <> O AndAlso _
                                    E <> R AndAlso E <> Y AndAlso N <> D AndAlso _
                                    N <> M AndAlso N <> O AndAlso N <> R AndAlso _
                                    N <> Y AndAlso M <> O AndAlso M <> R AndAlso _
                                    M <> Y AndAlso O <> R AndAlso O <> Y) AndAlso _
                                    ((1000 * S + 100 * E + 10 * N + D) + _
                                    (1000 * M + 100 * O + 10 * R + E) _
                                    = (10000 * M + 1000 * O + 100 * N + 10 * E + Y))Then
                                    ' 充足解
                                End If
                            Next
                        Next
                    Next
                Next
            Next
        Next
    Next
Next

End Sub
```

この覆面算に合わせた解法なので，問題が変わると，解法も再記述しなくてはならない．

2.2 制約プログラミングのメリット～例

- 制約プログラミングを使って記述する

```
Private Sub solveSendMoreMoney2()
    ‘ 変数の宣言
    Dim S As New CIntegerVariable(1, 9)
    Dim E As New CIntegerVariable(0, 9)
    Dim N As New CIntegerVariable(0, 9)
    Dim D As New CIntegerVariable(0, 9)
    Dim M As New CIntegerVariable(1, 9)
    Dim O As New CIntegerVariable(0, 9)
    Dim R As New CIntegerVariable(0, 9)
    Dim Y As New CIntegerVariable(0, 9)
    ‘ 制約条件の追加
    Dim alldiffConstraint As New CAllDifferent
    alldiffConstraint.add(S, E, N, D, M, O, R, Y)
    Dim mySolver As New CCPSolver
    mySolver.add(alldifferentConstraint)
    mySolver.add("(1000 * S + 100 * E + 10 * N + D) + (1000 * M + 100 * O + 10 * R + E)
    = (10000 * M + 1000 * O + 100 * N + 10 * E + Y)")
    ‘ 解探索
    While mySolver.nextSolution()
        ‘ 解の出力等
    End While
End  Sub
```

- 問題が変わっても，解法を再記述しなくてよい
- 解法の再利用が可能

2.3 制約プログラミングの現状

- 敷居が高い
  - Prologで記述されたツールがほとんど
    - Prologは使用人口が少ない
    - 開発環境が整備されていない
  - C++で記述されたツールは高価[4]
    - エンジニアが気軽に試すことができる価格ではない
  - Javaで記述されたツールは拡張性が低い[5]

3.1 研究目的･目標

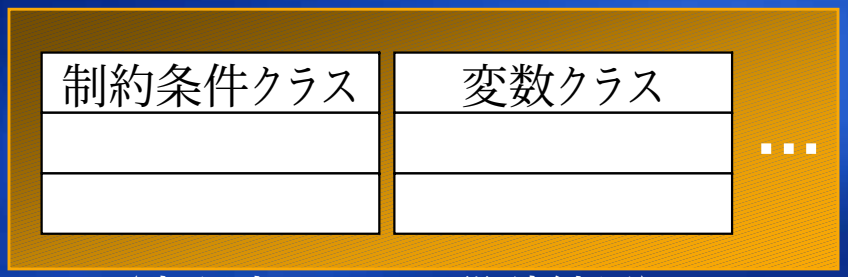
- 制約プログラミングの敷居を低くするような，ツールを設計する
- どのようなツールか？
  - Java，C++等，一般的なオブジェクト指向プログラミング言語で実装するクラスライブラリ[2]

3.2 設計の目標 (a)

- 制約プログラミングの記述に用いる，クラスライブラリの機能設計とクラス設計をする
- どのようなクラスライブラリか？
  - 制約充足問題をコンピュータ･プログラム上に表現することができる
  - 制約充足問題の充足解を探索することができる
  - 拡張性が高い

3.3 設計の目標（b）

- オブジェクト指向プログラミング言語で実装する
- 応用プログラム開発時にインポートして使用する



（本研究における設計結果）

実装

```
package solver;
public class Constraint {
}
```

ユーザが利用時にインポートする

```
import solver;
public class Sample {
    private Constraint SampleConstraint;
}
```

4.1 機能設計

- 問題定義機能
  - 変数定義機能：変数，領域を表現する
  - 制約条件定義機能：制約条件を表現する
- 解探索機能
  - 探索制御機能：解探索を制御する

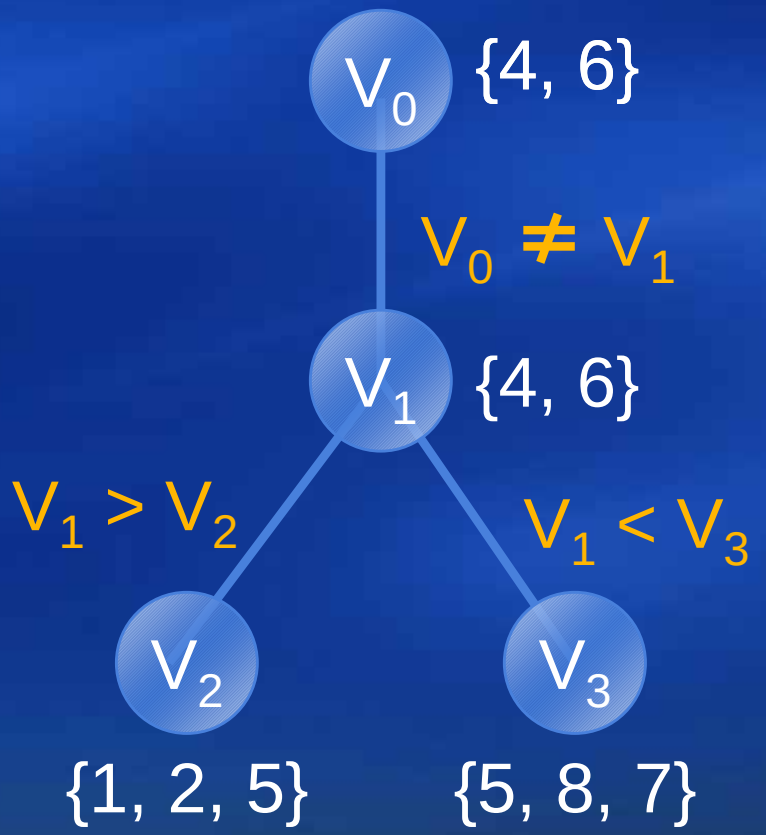
4.2 覆面算を例に機能を考える

- **(問題定義機能)** SEND + MORE = MONEY **を表現する**
  - **変数**: S, E, N, D, M, O, R, Y
  - **領域**: **変数** S **と** M **は**{1,…,9}**，他の変数は**{0,…,9}**．**
  - **制約条件**:
    - **変数に割り当てられる値はすべて異なる．**
    - (1000*S + 100*E + 10*N + D)+ (1000*M + 100*O + 10*R + E)= 10000*M + 1000*O + 100*N + 10*E + Y．
- **(解探索機能) 解を探索する**
  - S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2

4.3 クラス設計 (a)

- 各機能を実装するクラスを設計する
  - 問題定義機能を実装するクラス
    - 変数クラス
    - 値クラス
    - 領域クラス
    - 制約条件クラス
  - 解探索機能を実装するクラス
    - ソルバークラス
    - 解探索クラス

(例)



4.4 クラス設計 (b)

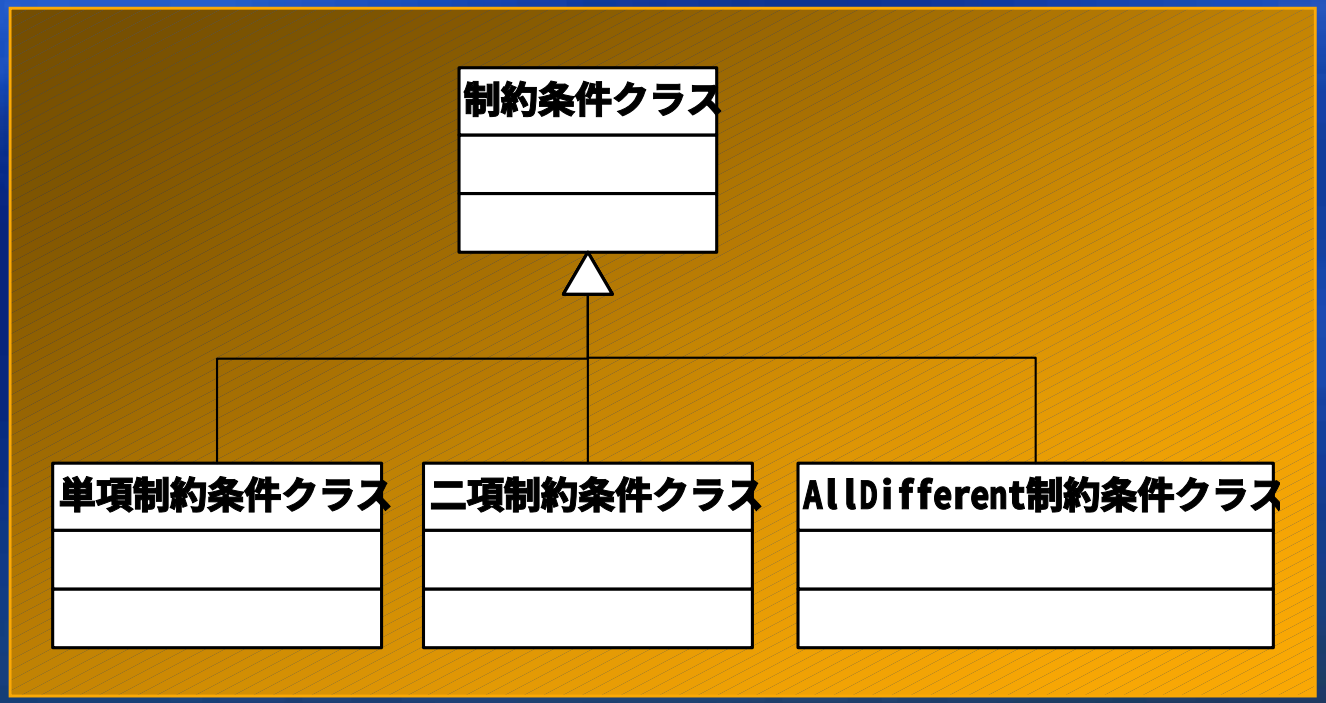
- 抽出したクラスの属性と操作を設計する
- (例) 制約変数クラス

- 属性
  - 名前
  - 領域
  - 値
- 操作
  - 名前を設定する
  - 領域をセットする
  - 値をセットする

制約変数クラス
− 名前 − 領域 − 値
＋ 名前を設定する ＋ 領域をセットする ＋ 値をセットする

4.5 クラス設計 (c)

- クラス階層を設計する
  - (例) 制約条件クラス
    - 単項制約条件クラス
    - 二項制約条件クラス
    - All Different制約条件クラス



4.6 クラス設計 (d)

- クラス階層を設計する
  - 実装例

```
Public MustInherit Class Constraint

End Class
```

```
Public Class BinaryConstraint
        Inherits Constraint

End Class
```

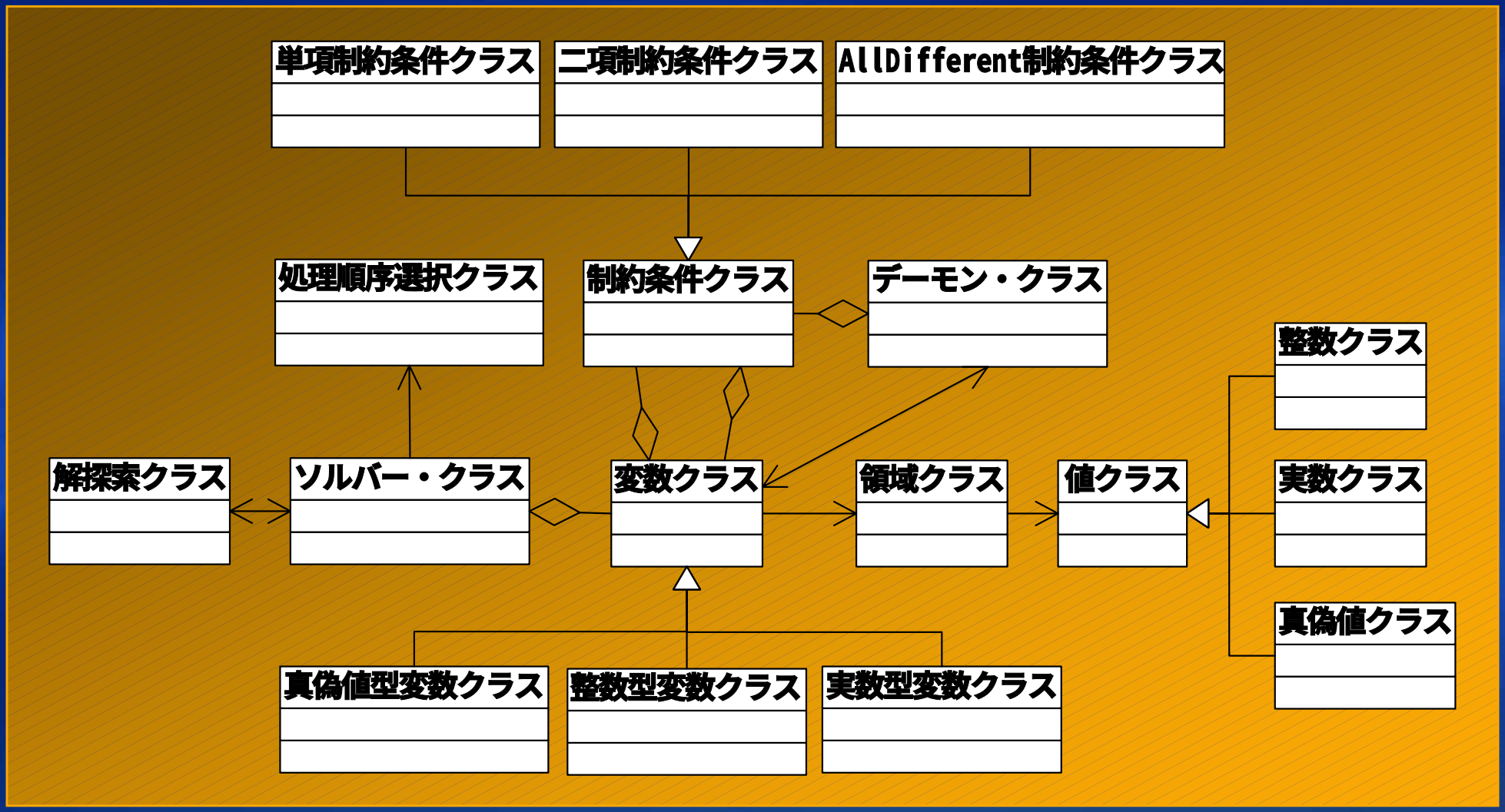
```
Public Class UnaryConstraint
        Inherits Constraint

End Class
```

```
Public Class AllDifferentConstraint
        Inherits Constraint

End Class
```

4.7 設計したクラスの全体図



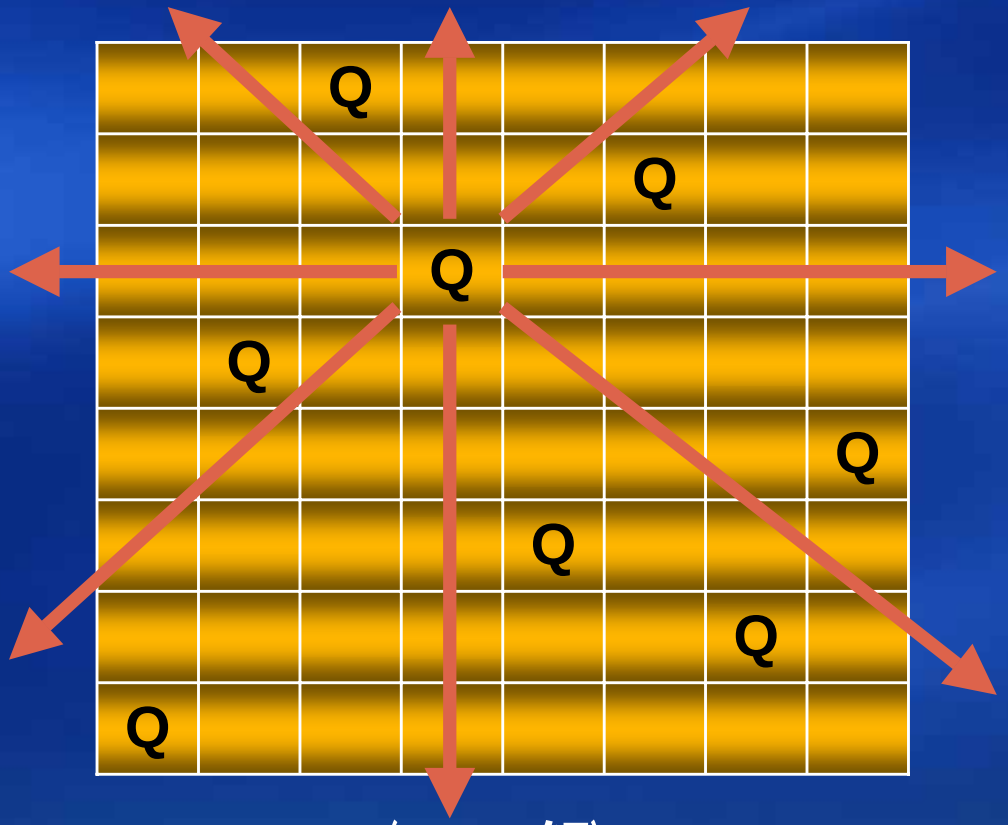
5.1 設計を評価する

- 一般的なオブジェクト指向プログラミング言語で実装するクラスライブラリを設計した
  - (後述) サンプル問題を記述できるか評価する
- 既存ライブラリよりも拡張性を高めた
  - (後述) 既存ライブラリと拡張性を比較して評価する

5.2 サンプル問題を記述する (a)

- サンプル問題：n-クイーン問題

“*n* × *n* のチェスの盤上に，*n* 個のクイーンを互いに獲られない場所に置け[1] ”



(1つの解)

5.2 サンプル問題を記述する (b)

```
‘ 制約変数を生成する
Dim VarArray(aQueenCount - 1) As CIntegerVariable

‘ AllDifferent制約条件を生成する
Dim CtAllDiff As New CAllDifferent

‘ 利き筋制約条件を生成する
Dim varCt As CIntSubtractNEConstraint
For j As Integer = VarArray.GetLowerBound(0) To VarArray.GetUpperBound(0)
    For k As Integer = (j + 1) To VarArray.GetUpperBound(0)
        varCt = New CIntSubtractNEConstraint(VarArray(j), VarArray(k), k - j)
        MySolver.add(CType(varCt, CConstraint))
        varCt = New CIntSubtractNEConstraint(VarArray(j), VarArray(k), j - k)
        MySolver.add(CType(varCt, CConstraint))
    Next
Next
```

(Visual Basic.NETによる例)

5.3 既存ライブラリと拡張性を比較する (a)

- (例) 制約条件を追加したい
  - 既存ライブラリの場合，同様の処理を再記述しなくてはならない

既存の制約条件

```
public class BinaryConstraint{
    public String getType() {
        …;
    }
    public boolean isSatisfied(){
        …;
    }
    …
}
```

追加する制約条件

```
public class AllDifferentConstraint {
    public String getType() {
        …;
    }
    public boolean isSatisfied() {
        …;
    }
    …
}
```

5.4 既存ライブラリと拡張性を比較する (b)

- 本研究で設計したクラスライブラリによる記述
  - 差分だけ記述すればよい

制約条件のひな型

```
public abstract class Constraint {
    public String getType() {
        …;
    }
    public abstract boolean isSatisfied() ;
    …
}
```

ひな型を派生

```
public class AllDifferentConstraint {
    …
    public boolean isSatisfied() {
        …;
    }
}
```

6. まとめ

- 制約プログラミングの敷居が高いという現状を踏まえ，それを低くするクラスライブラリを設計した．
- 今後の目標
  - 設計したクラスライブラリの実装
  - スケジューリング問題等，実際問題への適用

参考文献

[1]石塚満. 知識の表現と高速推論. 丸善, 1996.

[2]落水浩一郎, 東田雅宏. オブジェクトモデリング [新版] . 1998.

[3]溝口文雄他. 制約論理プログラミング. 共立出版, 1989.

[4]ILOG, Inc. http://www.ilog.co.jp/. 2004.

[5]Marc Torrens. http://www.marctorrens.com/. 2003.

[6]Pierre Roy, Anne Liret and Francois Pachet. A Framework approach for constraint satisfaction. ACM Computing Surveys, Vol. 32, No. 1es, pp. 13-16, 2000.

[7] Pierre Roy and Francois Pachet. A Framework approach for expressing knowledge about constraint satisfaction problems, 1997.