Программное средство *Choco*

Курс «Программные средства для решения задачи удовлетворения ограничений»

НИУ ИТМО, кафедра «Компьютерные технологии»

Choco – обзор

- http://www.emn.fr/z-info/choco-solver/
- Библиотека с открытым исходным кодом для языка *Java* для решения задач удовлетворения ограничений
- Поддержка вещественных переменных и ограничений на них
- Поддержка задач оптимизации

Общая структура программы с использованием *Choco*

```
import choco.Choco;
import choco.cp.solver.CPSolver ;
import choco.cp.model.CPModel ;
import choco.kernel.model.variables.integer.IntegerVariable;
import choco.kernel.model.constraints.Constraint;
public class MagicSquare {
      public static void main(String[] args) {
            // Постановка задачи
            // Построение модели задачи
            // Создание солвера
            // Решение задачи
            // Вывод решения
      }
}
```

Пример задачи

- Магический квадрат
- Нужно заполнить квадрат $n*n$ цифрами от 1 до n^2 так, чтобы в каждом столбце и в каждой строке сумма цифр была одинаковой

1	15	14	4
12	6	7	9
8	10	11	5
13	3	2	16

Постановка задачи

```
// Сторона квадрата:
int n = 3;
// Магическая сумма:
int magicSum = n * (n * n + 1) / 2;
```

Построение модели: задание переменных

```
CPModel m = new CPModel() ;
// Создание массива переменных:
IntegerVariable[][] var = new IntegerVariable[n][n];
// Для каждой переменной задаются имя и диапазон значений:
for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
            var[i][j] = Choco.makeIntVar("var_" + i +
                  "_" +  j, 1, n * n);
            // Добавление переменных в модель:
            m.addVariable(var[i][j]);
      }
}
```

Построение модели: задание ограничений

```
// суммы во всех строках равны волшебной
for (int i = 0; i < n; i++) {
        m.addConstraint(Choco.eq(Choco.sum(var[i]), magicSum));
}
IntegerVariable[][] varCol = new IntegerVariable[n][n];
for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
                varCol[i][j] = var[j][i];
        }
        // суммы во всех столбцах равны волшебной
        m.addConstraint(Choco.eq(Choco.sum(varCol[i]), magicSum));
}
```

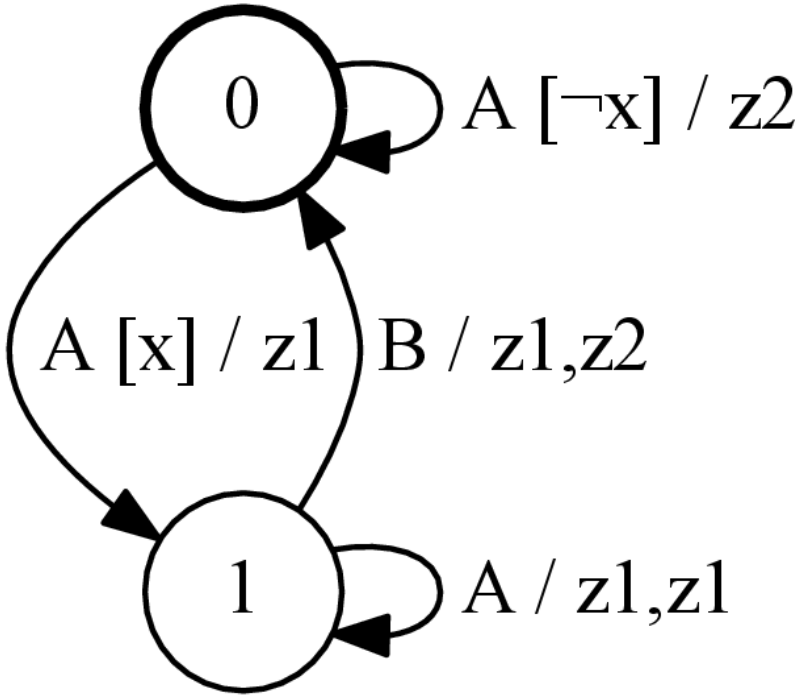
Создание солвера и решение модели

```
// Создание солвера
CPSolver s = new CPSolver();
// Чтение модели солвером
s.read(m);
// Решение модели
s.solve();
```

Вывод ответа

```
for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
            System.out.print(s.getVar(var[i][j]).
                  getVal());
      }
      System.out.println();
}
```

Построение управляющих автоматов при помощи *Choco* (1)



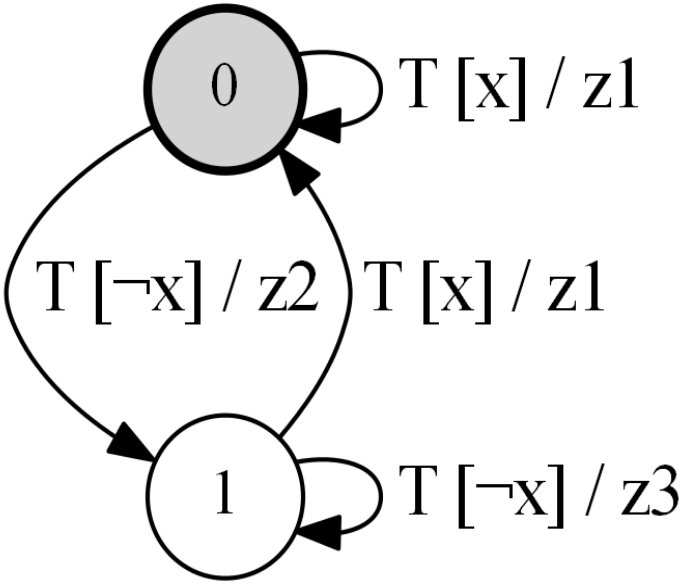
- Управляющий автомат:
  - *e* – входное событие
  - *f* – охранное условие
  - *A* – последовательность выходных воздействий
- Сценарий работы – последовательность троек <*e*, *f*, *A*>

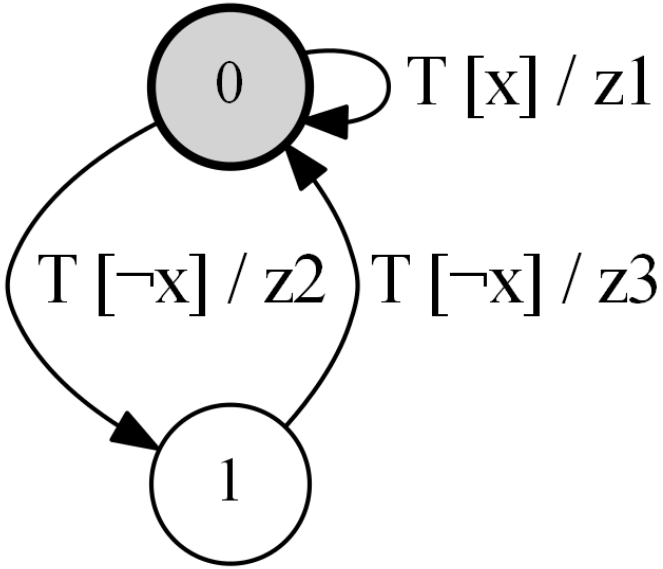
Построение управляющих автоматов при помощи *Choco* (2)

```
public static Automaton build(ScenariosTree tree, int size, boolean isComplete,
                    PrintWriter modelPrintWriter) {
          CPModel model = new CPModel();
          IntegerVariable[] nodesColorsVars = Choco.makeIntVarArray("Color",
                    tree.nodesCount(), 0, size - 1);
          Constraint rootColorConstraint = Choco.eq(nodesColorsVars[0], 0);
          model.addConstraint(rootColorConstraint);
          model.addConstraints(getTransitionsConstraints(size, tree, nodesColorsVars));
          model.addConstraints(getAdjacentConstraints(tree, nodesColorsVars));

          if (isComplete) {
                    model.addConstraints(getCompleteConstraints(size, tree,
                              nodesColorsVars));
          }

          if (modelPrintWriter != null) {
                    modelPrintWriter.println(model.varsToString());
                    modelPrintWriter.println(model.constraintsToString());
                    modelPrintWriter.flush();
          }
          return buildAutomatonFromModel(size, tree, nodesColorsVars, model);
}
```

Напоминание: требование полноты

- Полон только автомат слева





Ограничения, связанные с полнотой автомата (1)

```
private static Constraint[] getCompleteConstraints(int size, ScenariosTree tree,
                    IntegerVariable[] nodesColorsVars) {
          Map<String, Map<String, List<Node>>> eventExprToNodes = new TreeMap<String,
                    Map<String, List<Node>>>();
          Map<String, List<MyBooleanExpression>> pairs =
                    tree.getPairsEventExpression();
          for (String event : pairs.keySet()) {
                    Map<String, List<Node>> exprMap = new TreeMap<String,
                              List<Node>>();
                    eventExprToNodes.put(event, exprMap);
                    for (MyBooleanExpression expr : pairs.get(event)) {
                              exprMap.put(expr.toString(), new ArrayList<Node>());
                    }
          }
          for (Node node : tree.getNodes()) {
                    for (Transition t : node.getTransitions()) {
                              eventExprToNodes.get(t.getEvent()).get(t.getExpr().
                                        toString()).add(node);
                    }
          }

          ...
```

Ограничения, связанные с полнотой автомата (2)

```
int varsCount = tree.getVariablesCount();
Map<String, Integer> exprSetsCount = new TreeMap<String, Integer>();
for (MyBooleanExpression expr : tree.getExpressions()) {
          int cnt = expr.getSatisfiabilitySetsCount() * (1 << (varsCount -
                    expr.getVariablesCount()));
          exprSetsCount.put(expr.toString(), cnt);
          //System.out.println(expr.toString() + " " + cnt);
}
Map<String, Map<String, IntegerVariable[]>> used = new TreeMap<String, Map<String,
          IntegerVariable[]>>();
for (String event : pairs.keySet()) {
          Map<String, IntegerVariable[]> map = new TreeMap<String,
                    IntegerVariable[]>();
          for (MyBooleanExpression expr : pairs.get(event)) {
                    String arrayName = "used_" + event + "_" + expr.toString();
                    IntegerVariable[] vars = Choco.makeBooleanVarArray(arrayName,
                              size);
                    map.put(expr.toString(), vars);
          }
          used.put(event, map);
}
ArrayList<Constraint> ans = new ArrayList<Constraint>();
```

Ограничения, связанные с полнотой автомата (3)

```
for (String event : pairs.keySet()) {
        for (MyBooleanExpression expr : pairs.get(event)) {
                List<Node> nodes =
                        eventExprToNodes.get(event).get(expr.toString());
                IntegerVariable[] vars = used.get(event).get(expr.toString());
                for (int color = 0; color < size; color++) {
                        List<Constraint> orClauses = new ArrayList<Constraint>();
                        for (Node node : nodes) {
                                Constraint clause = Choco.eq(nodesColorsVars
                                        [node.getNumber()], color);
                                orClauses.add(clause);
                        }
                        Constraint orConstraint = Choco.or(orClauses.toArray(
                                new Constraint[0]));
                        Constraint eqConstraint = Choco.eq(vars[color], 1);
                        Constraint res = Choco.ifOnlyIf(orConstraint,
                                eqConstraint);
                        ans.add(res);
                }
        }
}
int totalSetsCount = 1 << varsCount;
```

Ограничения, связанные с полнотой автомата (4)

```
for (String event : pairs.keySet()) {
        for (int color = 0; color < size; color++) {
                List<IntegerVariable> vars = new
                        ArrayList<IntegerVariable>();
                int[] countsArray = new int[pairs.get(event).size()];
                int i = 0;
                for (MyBooleanExpression expr : pairs.get(event)) {
                        IntegerVariable var = used.get(event).
                                get(expr.toString())[color];
                        vars.add(var);
                        countsArray[i++] = exprSetsCount.get(
                                expr.toString());
                }
                IntegerVariable[] varsArray = vars.toArray(new
                        IntegerVariable[0]);
                Constraint equationFull = Choco.equation(totalSetsCount,
                        varsArray, countsArray);
                Constraint equationZero = Choco.equation(0, varsArray,
                         countsArray);
                ans.add(Choco.or(equationFull, equationZero));
        }
}
return ans.toArray(new Constraint[0]);
}
```

Ограничения, связанные со смежностью

```
private static Constraint[] getAdjacentConstraints(ScenariosTree tree,
                    IntegerVariable[] nodesColorsVars) {
          ArrayList<Constraint> ans = new ArrayList<Constraint>();
          Map<Node, Set<Node>> adjacent = AdjacentCalculator.getAdjacent(tree);
          for (Node node : tree.getNodes()) {
                    for (Node other : adjacent.get(node)) {
                              if (other.getNumber() < node.getNumber()) {
                                        IntegerVariable nodeColor =
                                                  nodesColorsVars[node.getNumber()];
                                        IntegerVariable otherColor =
                                                  nodesColorsVars[other.getNumber()];
                                        ans.add(Choco.neq(nodeColor, otherColor));
                              }
                    }
          }
          return ans.toArray(new Constraint[0]);
}
```

Ограничения на переходы

```
private static Constraint[] getTransitionsConstraints(int size, ScenariosTree tree,
                IntegerVariable[] nodesColorsVars) {
        List<Constraint> ans = new ArrayList<Constraint>();
        Map<String, IntegerVariable[]> transitionsVars = new HashMap<String,
                IntegerVariable[]>();
        for (Node node : tree.getNodes()) {
                for (Transition t : node.getTransitions()) {
                        String key = t.getEvent() + "[" +
                                t.getExpr().toString() + "]";
                        if (!transitionsVars.containsKey(key)) {
                                IntegerVariable[] vars = Choco.makeIntVarArray(
                                        key, size, 0, size - 1);
                                transitionsVars.put(key, vars);
                        }
                        IntegerVariable[] transitionVars = transitionsVars.
                                get(key);
                        for (int i = 0; i < size; i++) {
                                Constraint c1 = Choco.eq(nodesColorsVars[
                                        node.getNumber()], i);
                                Constraint c2 = Choco.eq(nodesColorsVars[
                                t.getDst().getNumber()], transitionVars[i]);
                                Constraint c = Choco.implies(c1, c2);
                                ans.add(c);
                        }
                }
        }
        return ans.toArray(new Constraint[0]);
}
```

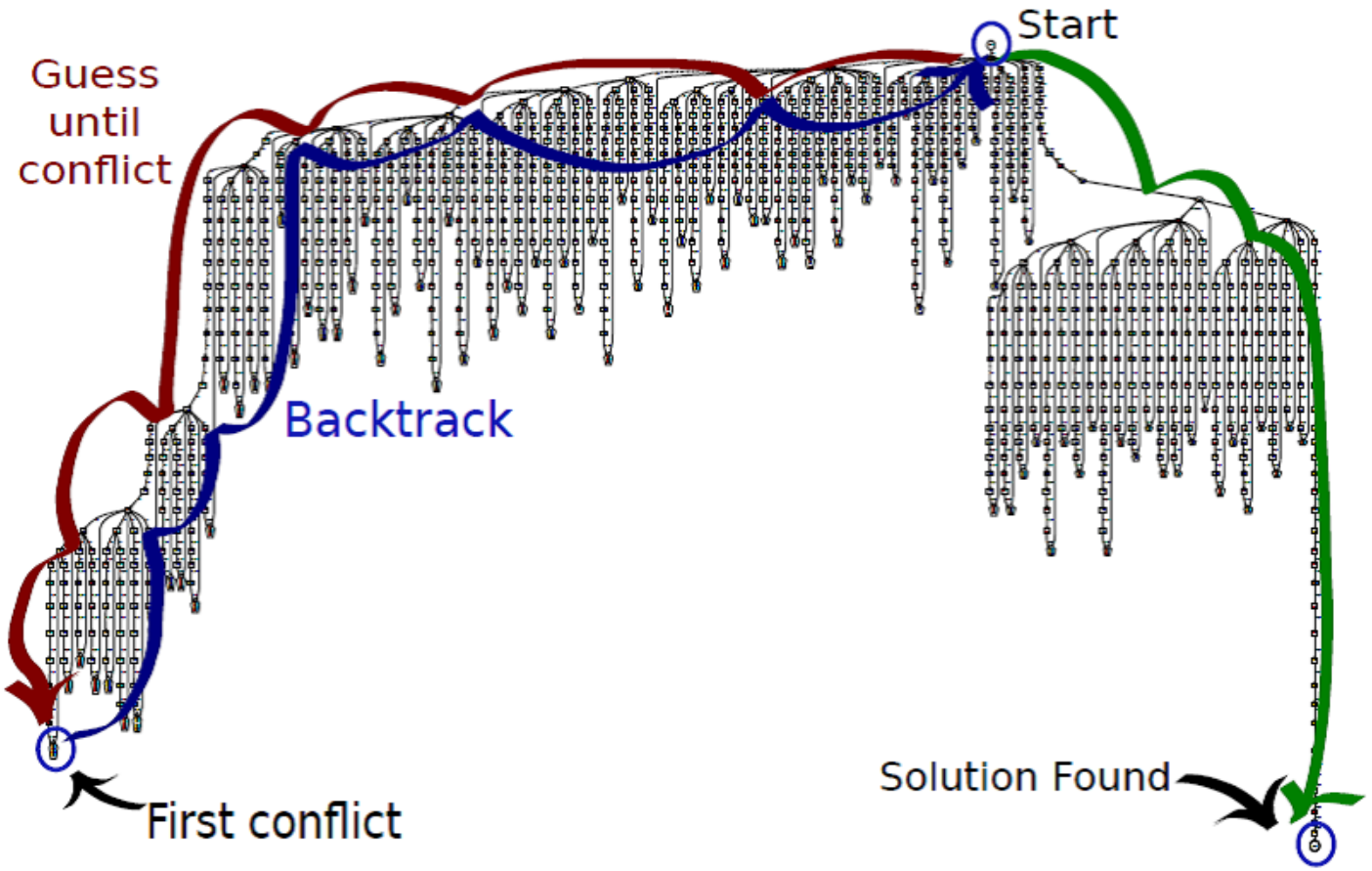
Принцип работы *Choco* и некоторых других программных средств

- Поиск с возвратом (backtracking)
- Распространение ограничений (constraint propagation)

Поиск с возвратом (1)

- Рекурсивный перебор всех возможных значений переменной
- Переменной присваивается значение, после этого упрощаются ограничения
- Если ограничения оказываются неудовлетворимы, происходит возврат из рекурсии

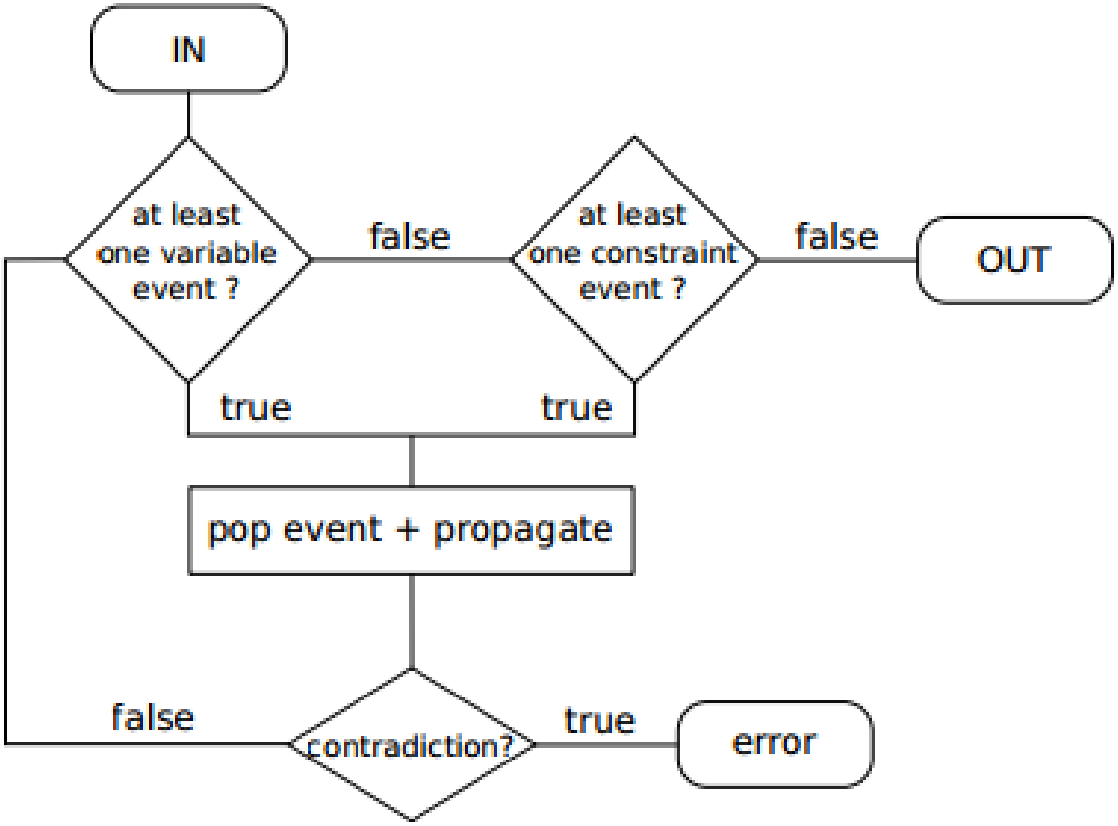
Поиск с возвратом (2)



Распространение ограничений (1)

- Способ отсечь заведомо недопустимые значения переменных на основе анализа ограничений
- В *Choco* используется событийный подход, а именно поддерживается очередь событий двух типов
- Событие первого типа связано с переменной и при «исполнении» сужает область ее значений
- «Исполнение» события может повлечь за собой новые события
- События второго типа связаны с ограничениями и являются вызовами методов их обработки

Распространение ограничений (2)



(Взято из http://choco.svn.sourceforge.net/viewvc/choco/tags/choco-2.1.5/src/site/resources/tex/documentation/choco-doc.pdf)

Спасибо за внимание!
Вопросы?