

# ID2204: Constraint Programming

## Getting Practical...

# Modeling Introduction

Lecture 02, 2010-03-25

---

**Christian Schulte**

[cschulte@kth.se](mailto:cschulte@kth.se)

Software and Computer Systems  
School of Information and Communication Technology  
KTH – Royal Institute of Technology  
Stockholm, Sweden



**KTH Information and  
Communication Technology**

---

# Constraint Programming with Gecode

---

---

# Overview

- Program model as ***script***
  - declare variables
  - post constraints (creates propagators)
  - define branching
  
- Solve script
  - basic search strategy
  - Gist: interactive visual search

---

# Program Model as Script

---

---

# Script: Overview

- Script is class inheriting from class Space
  - members store variables regarded as solution
- Script constructor
  - initialize variables
  - post propagators for constraints
  - define branching
- Copy constructor and copy function
  - copy a Script object during search
- Exploration takes Script object as input
  - returns object representing solution
- Main function
  - invokes search engine

# Script for SMM: Structure

```
#include <gencode/int.hh>
#include <gencode/search.hh>

using namespace Gencode;

class SendMoreMoney : public Space {
protected:
    IntVarArray l; // Digits for the letters
public:
    // Constructor for script
    SendMoreMoney(void) ... { ... }
    // Constructor for cloning
    SendMoreMoney(bool share, SendMoreMoney& s) ... { ... }
    // Perform copying during cloning
    virtual Space* copy(bool share) { ... }
    // Print solution
    void print(void) { ... }
};

...
```

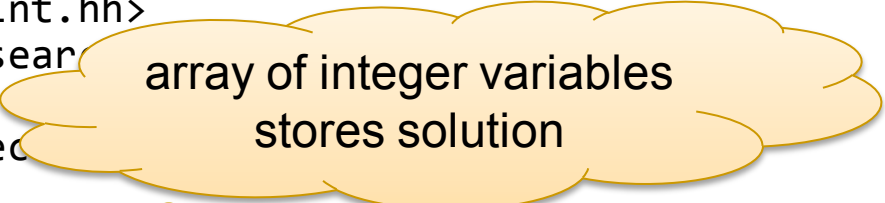
# Script for SMM: Structure

```
#include <gcode/int.hh>
#include <gcode/search>

using namespace Gecode;

class SendMoreMoney : public Space {
protected:
  IntVarArray l; // Digits for the letters
public:
  // Constructor for script
  SendMoreMoney(void) ... { ... }
  // Constructor for cloning
  SendMoreMoney(bool share, SendMoreMoney& s) ... { ... }
  // Perform copying during cloning
  virtual Space* copy(bool share) { ... }
  // Print solution
  void print(void) { ... }
};

...
```



array of integer variables  
stores solution

# Script for SMM: Structure

```
#include <gcode/int.hh>
#include <gcode/search.hh>

using namespace Gecode;

class SendMoreMoney : public Space {
protected:
  IntVarArray l; // Digits for the letters
public:
  // Constructor for script
  SendMoreMoney(void) ... { ... }
  // Constructor for cloning
  SendMoreMoney(bool share, SendMoreMoney& s) ... { ... }
  // Perform copying during cloning
  virtual Space* copy(bool share) { ... }
  // Print solution
  void print(void) { ... }
};

...
```

constructor: initialize  
variables, post  
constraints, define  
branching



# Script for SMM: Structure

```
#include <gecode/int.hh>
#include <gecode/search.hh>
```

```
using namespace Gecode;
```

```
class SendMoreMoney : public Space {
protected:
  IntVarArray l; // Digits for the letters
public:
  // Constructor for script
  SendMoreMoney(void) ... { ... }
  // Constructor for cloning
  SendMoreMoney(bool share, SendMoreMoney& s) ... { ... }
  // Perform copying during cloning
  virtual Space* copy(bool share) { ... }
  // Print solution
  void print(void) { ... }
};
```

copy constructor and  
copy function

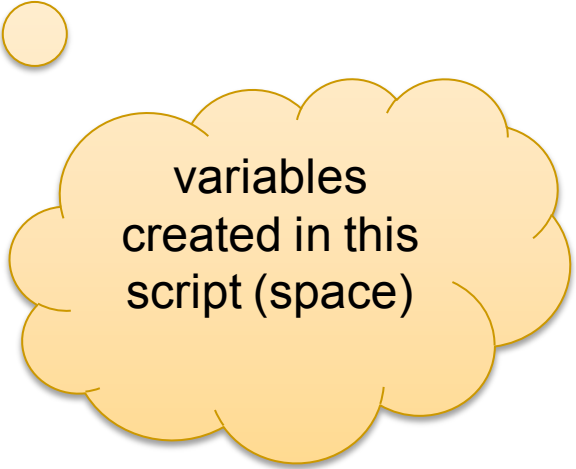
```
...
```

# Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {  
    IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),  
            m(l[4]), o(l[5]), r(l[6]), y(l[7]);  
    // Post constraints  
    ...  
    // Post branchings  
    ...  
}
```

# Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {  
    IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),  
            m(l[4]), o(l[5]), r(l[6]), y(l[7]);  
    // Post constraints  
    ...  
    // Post branchings  
    ...  
}
```



variables  
created in this  
script (space)

# Script for SMM: Constructor

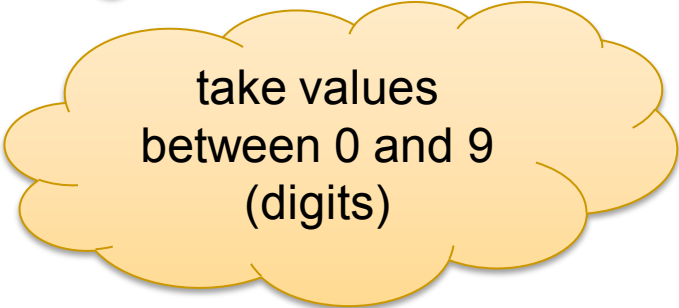
```
SendMoreMoney(void) : l(*this, 8, 0, 9) {  
    IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),  
            m(l[4]), o(l[5]), r(l[6]), y(l[7]);  
    // Post constraints  
    ...  
    // Post branchings  
    ...  
}
```



8 variables

# Script for SMM: Constructor

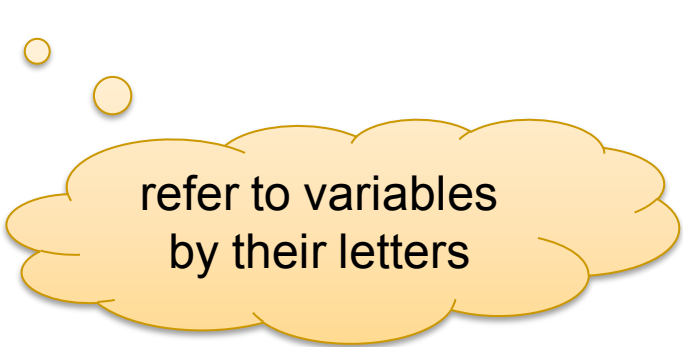
```
SendMoreMoney(void) : l(*this, 8, 0, 9) {  
    IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),  
            m(l[4]), o(l[5]), r(l[6]), y(l[7]);  
    // Post constraints  
    ...  
    // Post branchings  
    ...  
}
```



take values  
between 0 and 9  
(digits)

# Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {  
    IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),  
            m(l[4]), o(l[5]), r(l[6]), y(l[7]);  
    // Post constraints  
    ...  
    // Post branchings  
    ...  
}
```



refer to variables  
by their letters

# Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
    IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),
            m(l[4]), o(l[5]), r(l[6]), y(l[7]);
    // No leading zeros (IRT: integer relation type)
    rel(*this, s, IRT_NQ, 0);
    rel(*this, m, IRT_NQ, 0);
    // All letters must take distinct digits
    distinct(*this, l);
    // The linear equation must hold
    ...
    // Branch over the letters
    ...
}
```

---

# Posting Constraints

- Defined in namespace Gecode
- Check documentation for available constraints
- Take script reference as first argument
  - where is the propagator for the constraint to be posted!
  - script is a subclass of Space (computation space)



# Linear Equations and Linear Constraints

- Equations of the form

$$c_1 \cdot x_1 + \dots + c_n \cdot x_n = d$$

- integer constants:  $c_i$  and  $d$
- integer variables:  $x_i$

- In Gecode specified by arrays

- integers (IntArgs)  $c_i$
- variables (IntVarArray, IntVarArgs)  $x_i$

- Not only equations

- IRT\_EQ, IRT\_NQ, IRT\_LE, IRT\_GR, IRT\_LQ, IRT\_GQ
- equality, disequality, inequality (less, greater, less or equal, greater or equal)

# Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {  
    ...  
    // The linear equation must hold  
    IntArgs c(4+4+5); IntVarArgs x(4+4+5);  
    c[0]=1000; c[1]=100; c[2]=10; c[3]=1;  
    x[0]=s;    x[1]=e;    x[2]=n;    x[3]=d;  
    c[4]=1000; c[5]=100; c[6]=10; c[7]=1;  
    x[4]=m;    x[5]=o;    x[6]=r;    x[7]=e;  
    c[8]=-10000; c[9]=-1000; c[10]=-100; c[11]=-10; c[12]=-1;  
    x[8]=m;    x[9]=o;    x[10]=n;    x[11]=e;    x[12]=y;  
    linear(*this, c, x, IRT_EQ, 0);  
    // Branch over the letters  
    ...  
}
```

---

# Linear Expressions

- Other options for posting linear constraints are available: minimodeling support
  - linear expressions
  - Boolean expressions
  - matrix classes
  - ...
  
- See the examples that come with Gecode

# Script for SMM: Constructor

```
...
#include <gcode/minimodel.hh>
...

SendMoreMoney(void) : l(*this, 8, 0, 9) {
    ...
    // The linear equation must hold
    post(*this,
          1000*s + 100*e + 10*n + d
          + 1000*m + 100*o + 10*r + e
          == 10000*m + 1000*o + 100*n + 10*e + y);
    // Branch over the letters
    ...
}
```

---

# Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {  
    ...  
    // Branch over the letters  
    branch(*this, 1, INT_VAR_SIZE_MIN, INT_VAL_MIN);  
}
```

# Branching

## ■ Which variable to choose

- given order                   INT\_VAR\_NONE
- smallest size                 INT\_VAR\_SIZE\_MIN
- smallest minimum           INT\_VAR\_MIN\_MIN
- ...

## ■ How to branch: which value to choose

- try smallest value           INT\_VAL\_MIN
- split (lower first)         INT\_VAL\_SPLIT\_MIN
- ...


---

# Script for SMM: Copying

```
// Constructor for cloning
SendMoreMoney(bool share, SendMoreMoney& s) : Space(share, s) {
    l.update(*this, share, s.l);
}
// Perform copying during cloning
virtual Space* copy(bool share) {
    return new SendMoreMoney(share,*this);
}
```

# Script for SMM: Copying

```
// Constructor for cloning
SendMoreMoney(bool share, SendMoreMoney& s) : Space(share, s) {
    l.update(*this, share, s.l);
}
// Perform copying during cloning
virtual Space* copy(bool share) {
    return new SendMoreMoney(share, *this);
}
```

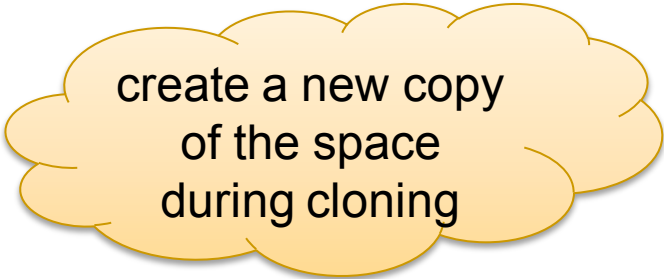


update all  
variables needed  
for solution



# Script for SMM: Copying

```
// Constructor for cloning
SendMoreMoney(bool share, SendMoreMoney& s) : Space(share, s) {
    l.update(*this, share, s.l);
}
// Perform copying during cloning
virtual Space* copy(bool share) {
    return new SendMoreMoney(share,*this);
}
```



create a new copy  
of the space  
during cloning

# Copying

- Required during exploration
  - before starting to guess: make copy
  - when guess is wrong: use copy
  - discussed later
- Copy constructor and copy function needed
  - copy constructor is specific to script
  - updates (copies) variables in particular

# Copy Constructor And Copy Function

- Always same structure
- Important!
  - must update the variables of a script!
  - if you forget: crash, boom, bang, ...

---

# Script for SMM: Print Function

```
...  
// Print solution  
void print(void) {  
    std::cout << 1 << std::endl;  
}
```

---

# Summary: Script

- Variables
  - declare as members
  - initialize in constructor
  - update in copy constructor
- Posting constraints
- Create branching
- Provide copy constructor and copy function

---

# Solving Scripts

---

---

# Available Search Engines

- Returning solutions one by one for script
  - DFS                    depth-first search
  - BAB                    branch-and-bound
  - Restart, LDS
  
- Interactive, visual search
  - Gist


# Main Method: First Solution

...

```
int main(int argc, char* argv[]) {
    SendMoreMoney* m = new SendMoreMoney;
    DFS<SendMoreMoney> e(m);
    delete m;
    if (SendMoreMoney* s = e.next()) {
        s->print(); delete s;
    }
    return 0;
}
```



# Main Method: First Solution



create root  
space for  
search

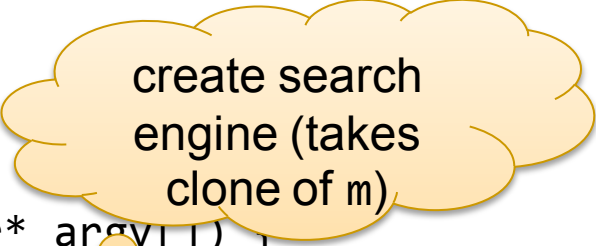
...

```
int main(int argc, char* argv[]) {  
    SendMoreMoney* m = new SendMoreMoney;  
    DFS<SendMoreMoney> e(m);  
    delete m;  
    if (SendMoreMoney* s = e.next()) {  
        s->print(); delete s;  
    }  
    return 0;  
}
```

# Main Method: First Solution

...

```
int main(int argc, char* argv[]) {
    SendMoreMoney* m = new SendMoreMoney;
    DFS<SendMoreMoney> e(m);
    delete m;
    if (SendMoreMoney* s = e.next()) {
        s->print(); delete s;
    }
    return 0;
}
```



create search  
engine (takes  
clone of m)

# Main Method: First Solution

root space not  
any longer  
needed


...

```
int main(int argc, char* argv[]) {
    SendMoreMoney* m = new SendMoreMoney;
    DFS<SendMoreMoney> e(m);
    delete m;
    if (SendMoreMoney* s = e.next()) {
        s->print(); delete s;
    }
    return 0;
}
```

# Main Method: First Solution

...

```
int main(int argc, char* argv[...]  
    SendMoreMoney* m = new SendMoreMoney;  
    DFS<SendMoreMoney> e(m);  
    delete m;  
    if (SendMoreMoney* s = e.next()) {  
        s->print(); delete s;  
    }  
    return 0;  
}
```



search first  
solution and  
print it

# Main Method: All Solutions

...

```
int main(int argc, char* argv[]) {
    SendMoreMoney* m = new SendMoreMoney;
    DFS<SendMoreMoney> e(m);
    delete m;
    while (SendMoreMoney* s = e.next()) {
        s->print(); delete s;
    }
    return 0;
}
```

---

# Gecode Gist

- A graphical tool for exploring the search tree
  - explore tree step by step
  - tree can be scaled
  - double-clicking node prints information: inspection
  - search for next solution, all solutions
  - ...
  
- Best to play a little bit by yourself
  - hide and unhide failed subtrees
  - ...

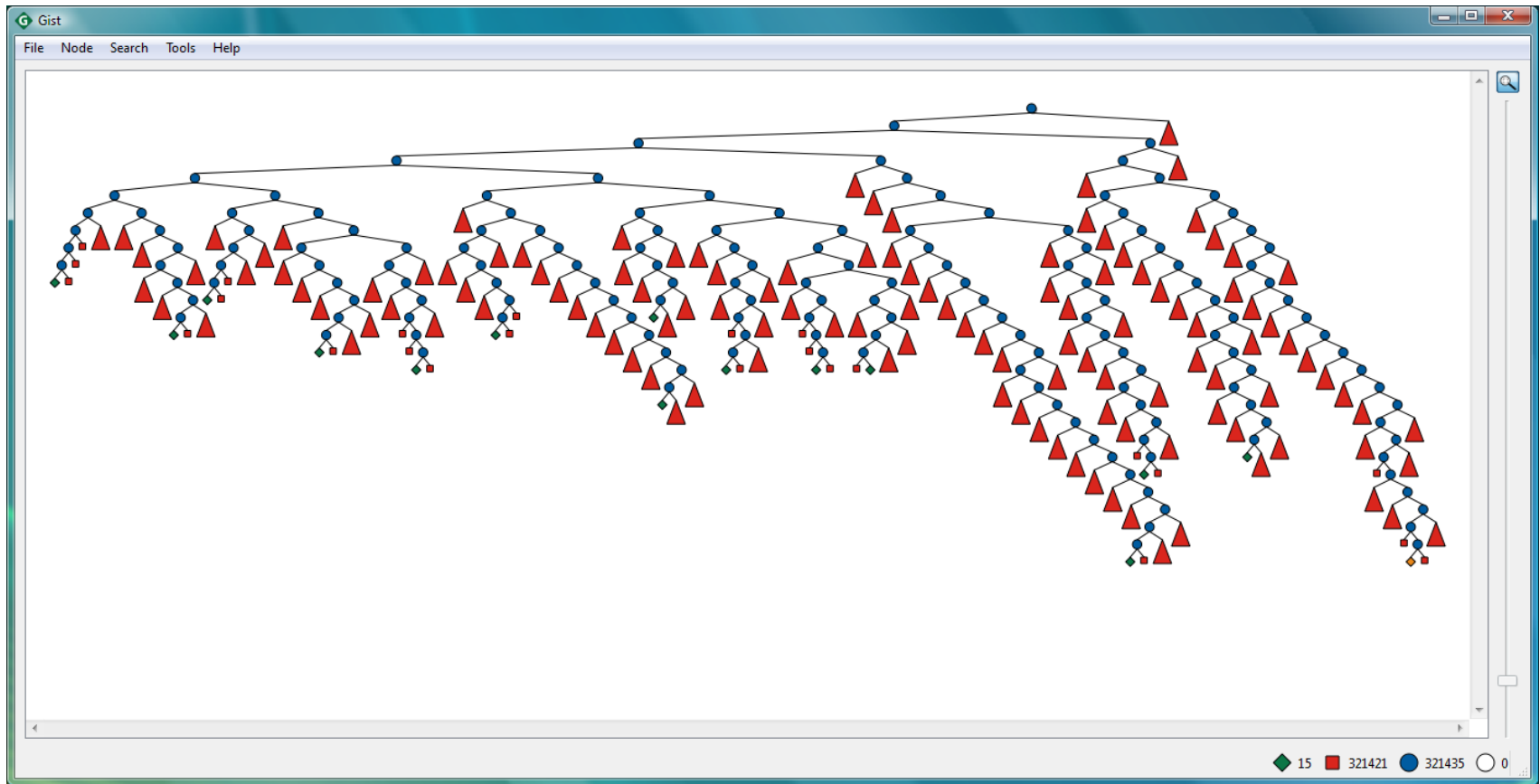
---

# Main Function: Gist

```
#include <gecode/gist.hh>

int main(int argc, char* argv[]) {
    SendMoreMoney* m = new SendMoreMoney;
    Gist::dfs(m);
    delete m;
    return 0;
}
```

# Gist Screenshot





---

# Best Solution Search

---

# Reminder: SMM++

- Find distinct digits for letters, such that

$$\begin{array}{r} \text{SEND} \\ + \text{MOST} \\ \hline = \text{MONEY} \end{array}$$

and **MONEY** maximal

---

# Script for SMM++

- Similar, please try it yourself at home
- In the following, referred to by `SendMostMoney`

# Solving SMM++: Order

- Principle

- for each solution found, constrain remaining search for better solution

- Implemented as additional method

```
virtual void constrain(const Space& b) {  
    ...  
}
```

- Argument b refers to so far best solution

- only take values from b
- never mix variables!

- Invoked on object to be constrained

# Order for SMM++

```
virtual void constrain(const Space& _b) {
    const SendMostMoney& b =
        static_cast<const SendMostMoney&>(_b);

    IntVar e(1[1]), n(1[2]), m(1[4]), o(1[5]), y(1[8]);

    IntVar b_e(b.l[1]), b_n(b.l[2]), b_m(b.l[4]),
        b_o(b.l[5]), b_y(b.l[8]);

    int money = (10000*b_m.val()+1000*b_o.val()+100*b_n.val()+
        10*b_e.val()+b_y.val());

    post(*this, 10000*m+1000*o+100*n+10*e+y > money);
}
```

# Main Method: All Solutions

...

```
int main(int argc, char* argv[]) {
    SendMostMoney* m = new SendMostMoney;
    BAB<SendMostMoney> e(m);
    delete m;
    while (SendMostMoney* s = e.next()) {
        s->print(); delete s;
    }
    return 0;
}
```

---

# Main Function: Gist

```
#include <gecode/gist.hh>

int main(int argc, char* argv[]) {
    SendMostMoney* m = new SendMostMoney;
    Gist::bab(m);
    delete m;
    return 0;
}
```

---

# Summary: Solving

- Result-only search engines
  - DFS, BAB
- Interactive search engine
  - Gist
  
- Best solution search uses constrain-method for posting constraint
- Search engine independent of script and constrain-method