Gecode

an open constraint solving library

Christian Schulte
KTH – Royal Institute of Technology, Sweden

Gecode

- Generic Constraint Development Environment
  - open source
  - C++ library
  - constraint propagation + complete (parallel) search
  - finite domain and finite set constraints
  - complete documentation (reference, tutorial, papers)
  - thousands of users

Overview

- History and facts
- Modeling (interfacing) & programming
- Openness

History

- 2002
  - development started
- 1.0.0
  - Dec 6, 2005
- 2.0.0
  - Nov 14, 2007
- 3.0.0
  - Mar 13, 2009
- 3.5.0 (current)
  - Feb 1, 2011

24 releases

43 kloc, 21 klod

77 kloc, 41 klod

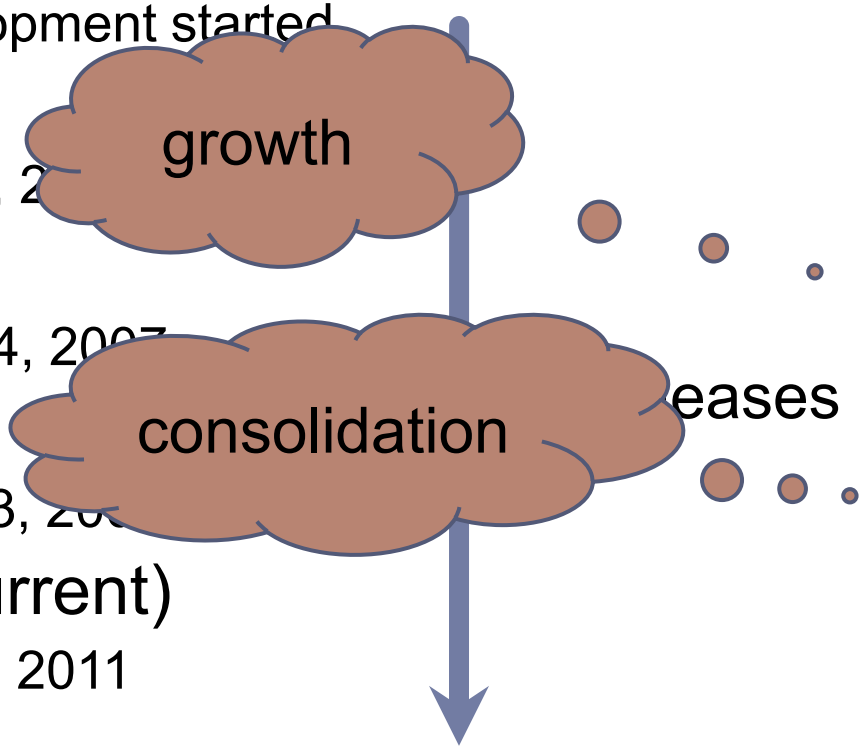
81 kloc, 41 klod

126 kloc, 52 klod

- … 4.0.0 in 2012

History

- 2002
  - development started
- 1.0.0
  - Dec 6, 2
- 2.0.0
  - Nov 14, 200
- 3.0.0
  - Mar 13, 20
- 3.5.0 (current)
  - Feb 1, 2011



43 kloc, 21 klod

77 kloc, 41 klod

eases

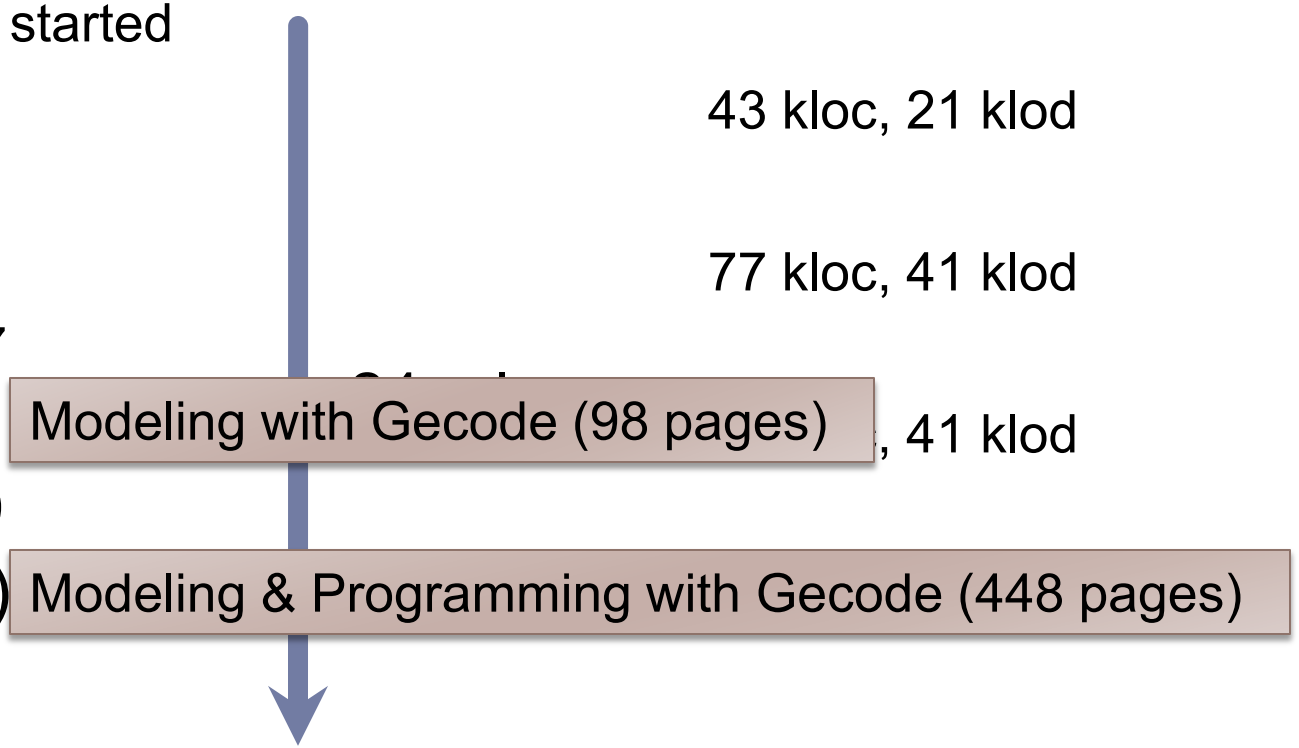
81 kloc, 41 klod

126 kloc, 52 klod

- … 4.0.0 in 2012

History: Tutorial Documentation

- 2002
  - development started
- 1.0.0 — 43 kloc, 21 klod
  - Dec 6, 2005
- 2.0.0 — 77 kloc, 41 klod
  - Nov 14, 2007
- 3.0.0 — , 41 klod
  - Mar 13, 2009
- 3.5.0 (current)
  - Feb 1, 2011
- … 4.0.0 in 2012



People

- Core team
  - Christian Schulte — KTH – Royal Institute of Technology, Sweden
  - Guido Tack — K.U. Leuven, Belgium
  - Mikael Z. Lagerkvist — KTH – Royal Institute of Technology, Sweden

- Code
  - contributions: David Rijsman, Denys Duchier, Filip Konvicka, Gabor Szokoli, Gregory Crosswhite, Håkan Kjellerstrand, Patrick Pekczynski, Raphael Reischuk, Tias Guns.
  - fixes: Alexander Samoilov, David Rijsman, Geoffrey Chu, Grégoire Dooms, Gustavo Gutierrez, Olof Sivertsson.
- Documentation
  - Seyed Hosein Attarzadeh Niaki, Vincent Barichard, Felix Brandt, Markus Böhm, Roberto Castañeda, Gregory Crosswhite, Pierre Flener, Gustavo Gutierrez, Gabriel Hjort Blindell, Sverker Janson, Andreas Karlsson, Håkan Kjellerstrand, Chris Mears, Flutra Osmani, Dan Scott, Kish Shen.

Goals

- Research
  - architecture of constraint programming systems
  - propagation algorithms, search, modeling languages, ...
- Efficiency
  - competitive (winner MiniZinc challenges 2008-2010, all categories)
  - proving architecture right
- Education
  - state-of-the-art, free platform for teaching

Users

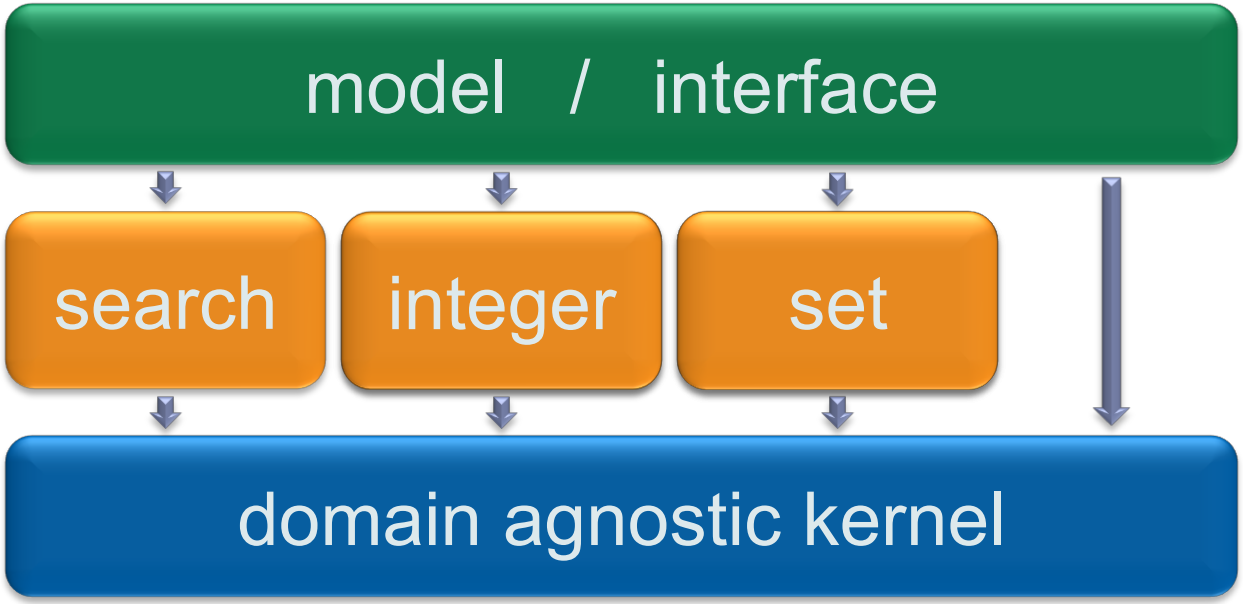
- Research
  - own papers
  - papers by others: experiments and comparison
  - Google scholar: some 500 references to Gecode
- Education: teaching
  - KTH, Uppsala U, U Freiburg, UC Louvain, Saarland U, American U Cairo, U Waterloo, U Javeriana-Cali, …
- Industry
  - several companies have integrated Gecode into products (part of hybrid solvers)

Deployment & Distribution

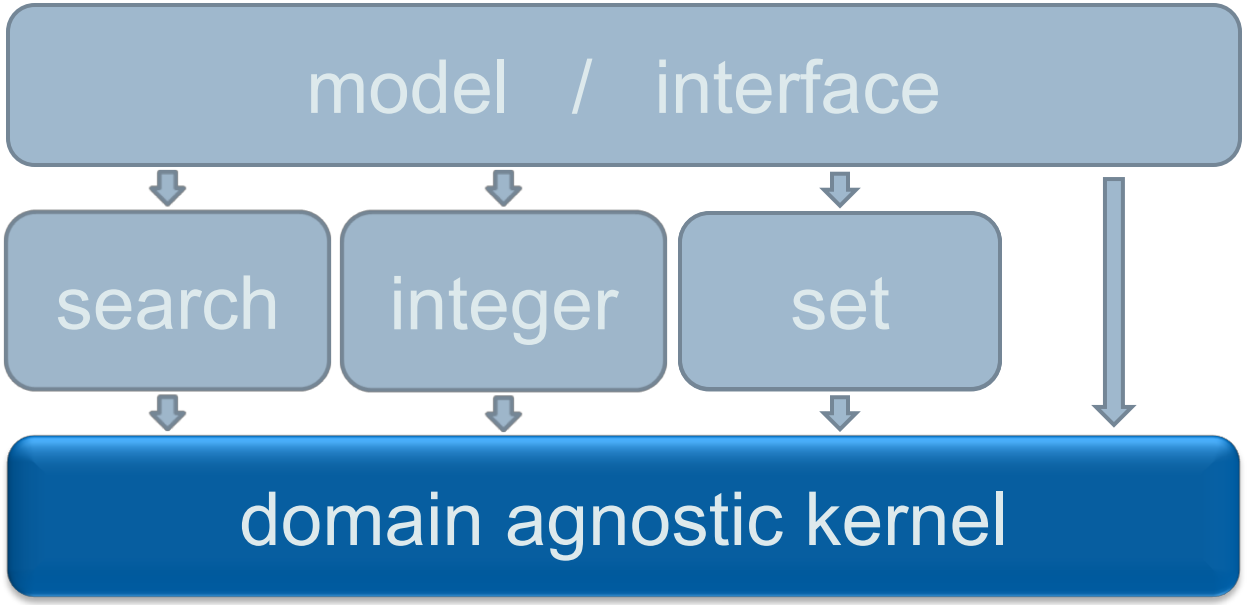
- Open source ≠ Linux only
  - Gecode is native citizen of: Linux, Mac, Windows
- High-quality
  - extensive test infrastructure (around 16% of code base)
- Downloads from Gecode webpage
  - software: between 25 to 125 per day
  - documentation: between 50 to 300 per day
- Included in
  - Debian, Ubuntu, FreeBSD

Modeling & Programming

Architecture

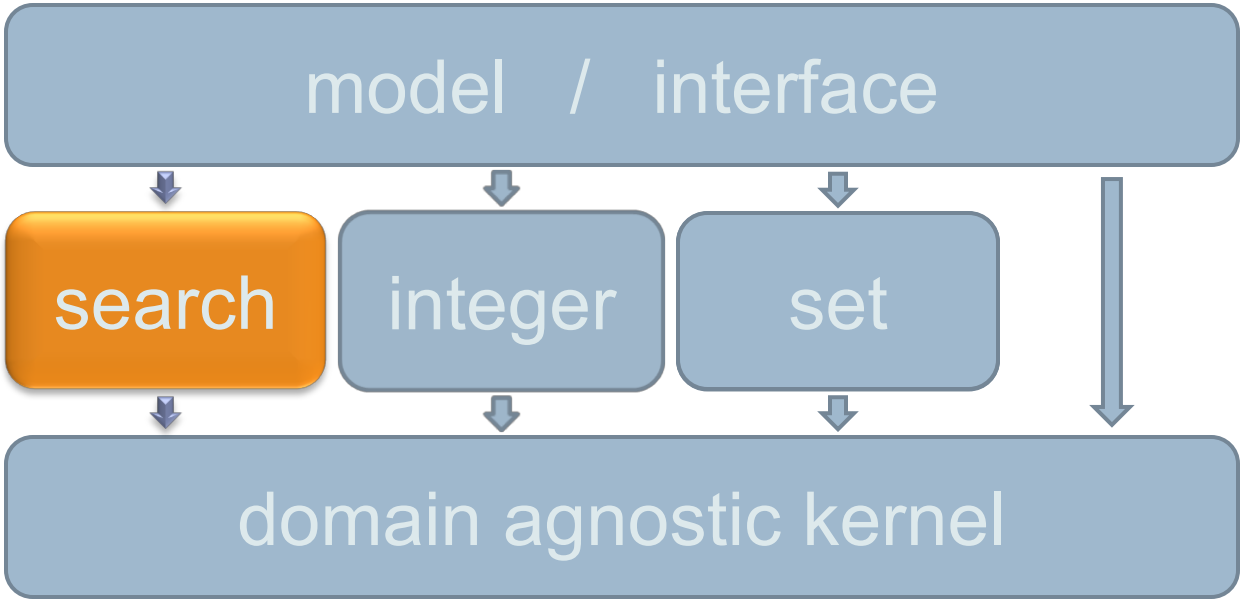


Architecture



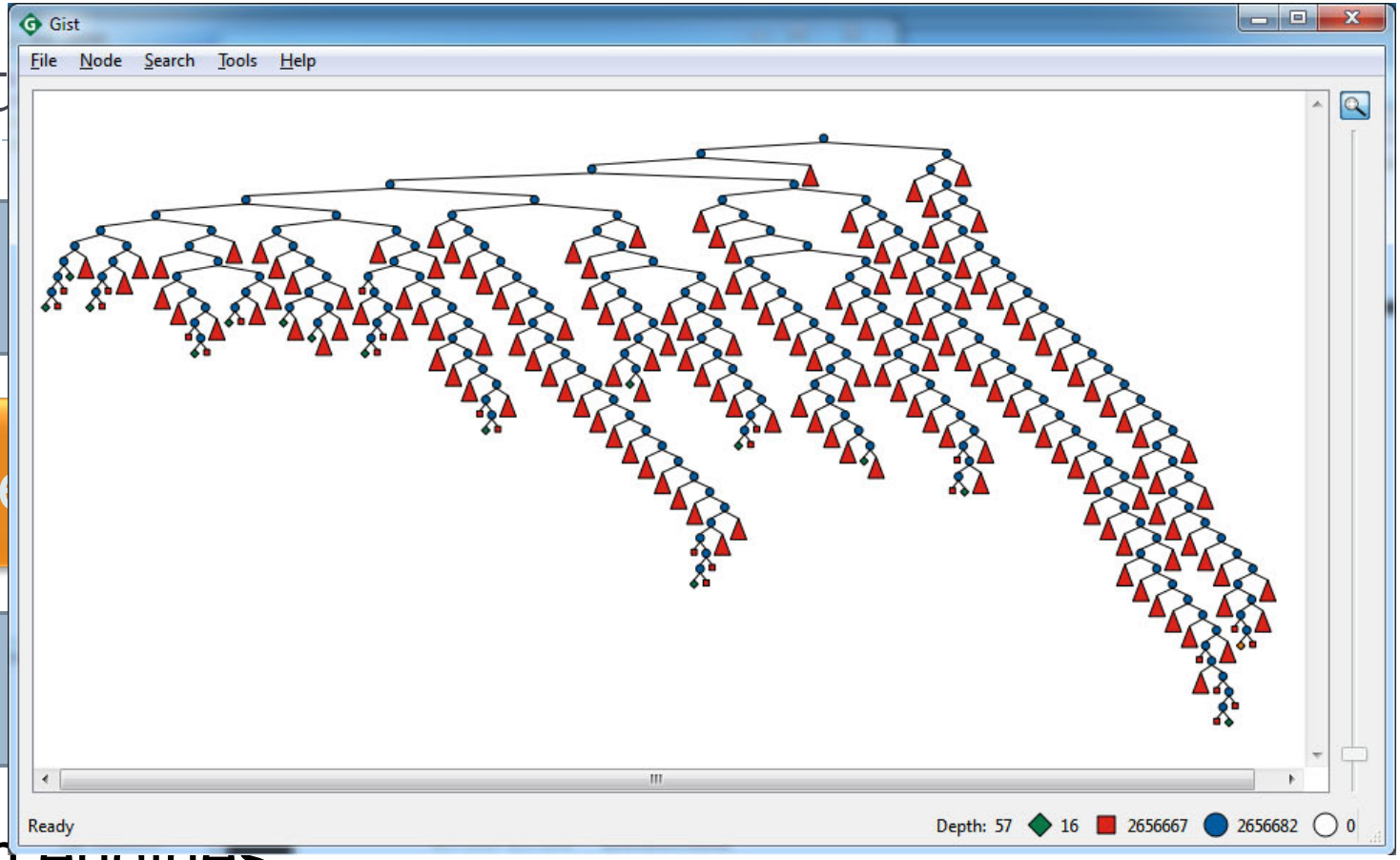
- propagation loop
- backtracking for search
- memory management

Architecture



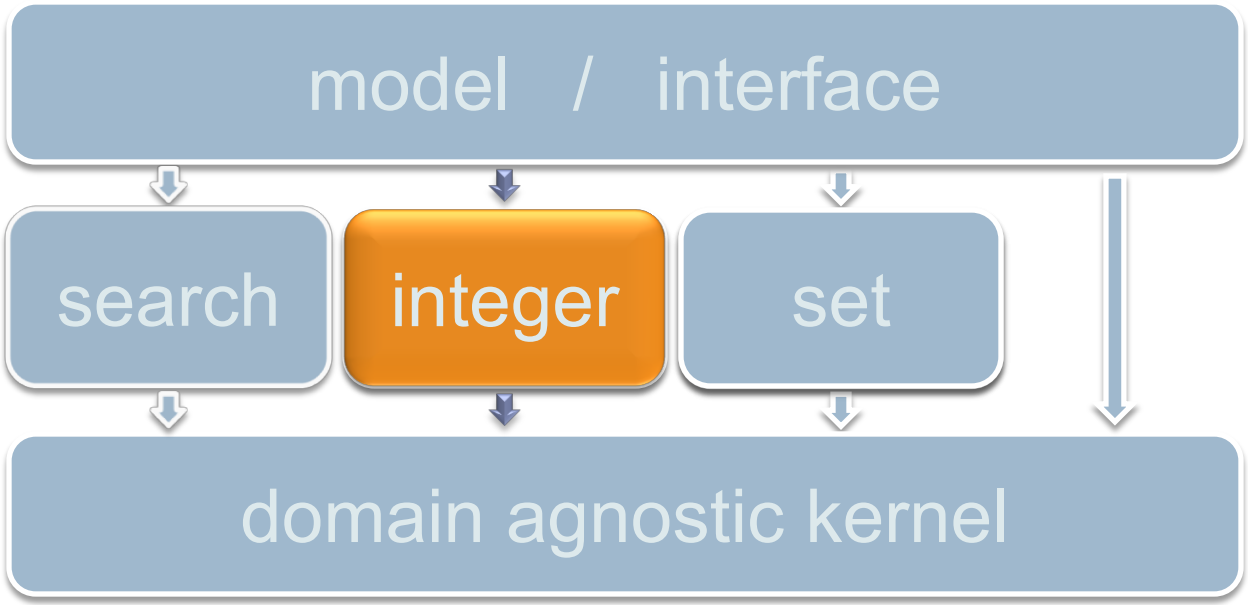
- search engines
  - depth-first (DFS) and branch-and-bound (BAB)
  - parallel search

Archit



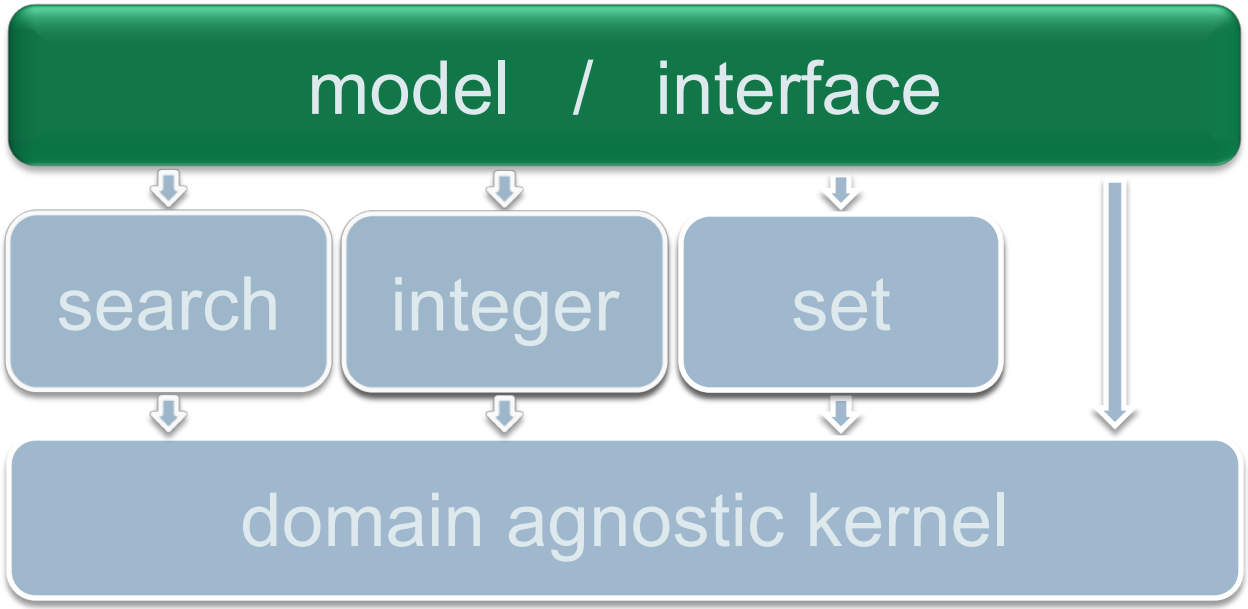
- search engines
  - depth-first (DFS) and branch-and-bound (BAB)
  - parallel search
- search tool: Gist (millions of nodes)

Architecture



- variables
- propagators (constraints)
- branchers (search heuristics)

Architecture



- direct C++ modeling or interfacing
- language interfaces: MiniZinc (see Guido's talk later), Java, JavaScript, Lisp, Ruby, Eclipse Prolog, …

Modeling (interfacing)

- Use modeling layer in C++
  - matrices, operators for arithmetical and logical expressions, …
- Use predefined
  - constraints
  - search heuristics and engines
- Documentation
  - getting started 30 pages
  - concepts and functionality 96 pages
  - case studies 76 pages

Modeling (interfacing)

- Constraint families
  - arithmetics, Boolean, ordering, ….
  - alldifferent, count (global cardinality, …), element, scheduling, table and regular, sorted, sequence, circuit, channel, bin-packing, lex, geometrical packing
- Families
  - different variants
  - different propagation strength

More Programming

- Programming propagators and branchings
  - straightforward object-oriented interfaces
- Programming new variable types
  - system-specific aspects generated from simple specification
  - only domain implementation required
- Programming search engines (exploration)
  - based on spaces, similar to Oz [Schulte, CP 1997]

Programming

- Interfaces for programming
  - propagators (for constraints)
  - branchers (for search heuristics)
  - variables
  - search engines

Documentation	intro	advanced
propagators	40 pages	58 pages
branchers	22 pages	
variables		44 pages
search engines	12 pages	26 pages

Openness

Open Source

- MIT license
  - permits commercial, closed-source use
  - disclaims all liabilities (as far as possible)
- License motivation
  - public funding
  - focus on research
- Not a reason
  - attitude, politics, dogmatism

Open Architecture

- More than a license
  - **license** restricts what users **may do**
  - **code and documentation** restrict what users **can do**
- Modular, structured, documented, readable
  - complete tutorial and reference documentation
  - ideas based on scientific publications
- Equal rights: clients are first-class citizens
  - you can do what we can do: APIs
  - you can know what we know: documentation
  - on every level of abstraction

Open Development

- We encourage contributions
  - direct, small contributions
    → we take over maintenance and distribution
  - larger modules on top of Gecode
    → you maintain the code, we distribute it

- Prerequisites
  - MIT license
  - compiles and runs on platforms we support

Summary

- Open source libraries require open architecture
  - users need good code to build on
- Open architecture promotes equality
  - client code is first-class citizen
  - encourages code contributions
- Open development fosters research
  - collaboration
  - experiments are reproducible

Constraint Programming with Gecode

Modeling in Gecode

- Model structure
  - subclass from class Space (node in search tree)
  - constructor: create variables, post constraints & branchings
  - two additional methods for copying (trivial)

- Solving model
  - create instance of model
  - pass to search engine, or apply search strategy

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 <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) { … }
};

…
```

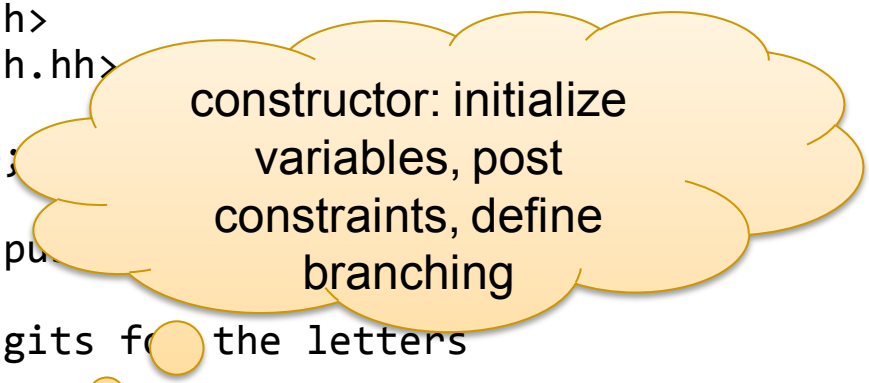
Script for SMM: Structure

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

using namespace Gec

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

using namespace Gecode;

class SendMoreMoney : pu
protected:
  IntVarArray l; // Digits fo 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

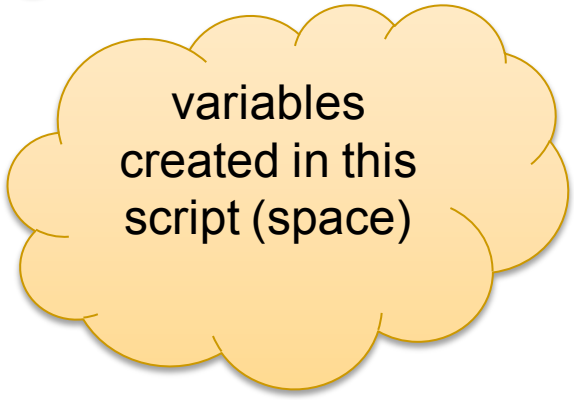
```
#include <gecode/int.hh>
#include <gecode/search.hh>

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) { … }
};

…
```

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
  …
}
```

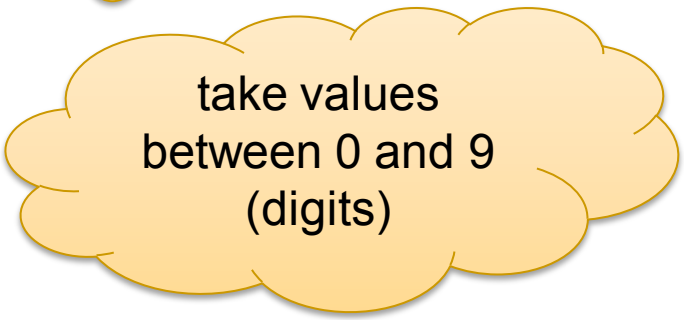


Script for SMM: Constructor

```
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
  …
}
```

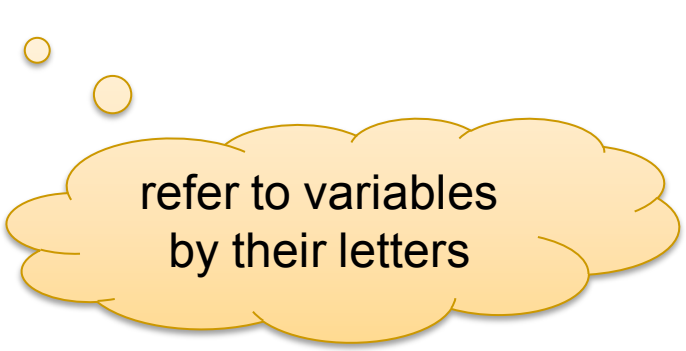
Script for SMM: Constructor

```
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
  …
}
```



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]);
  // 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

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

  - 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 <gecode/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, l, 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,*th
}
```

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);
}
```

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 << l << 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

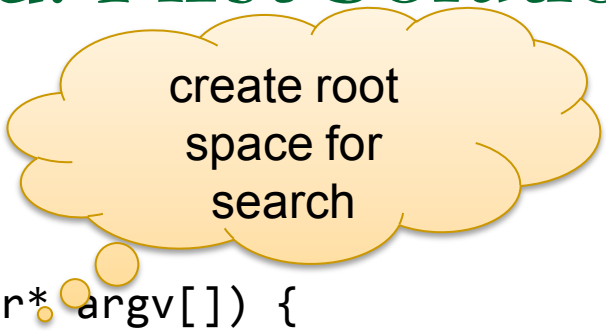
- Returning solutions one by one for script
  - `DFS` depth-first search
  - `BAB` branch-and-bound
  - `Restart, LDS`
- Interactive, visual search
  - `Gist`

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;
}
```

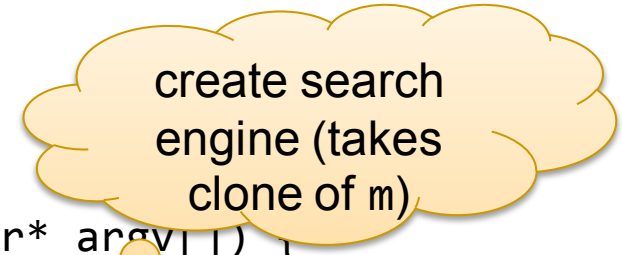
Main Method: First Solution



…

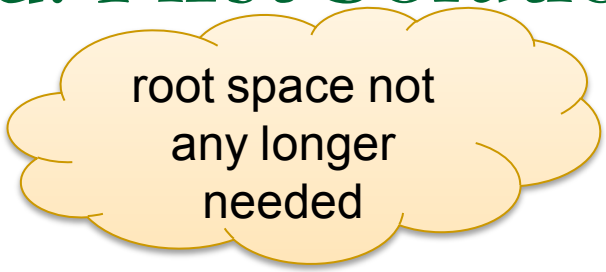
```
int main(int argc, char* argv[]) {
  SendMoreMoney* m = new SendMoreMoney;
  DFS<SendMoreMoney> e(m);
  delete m;
  if (SendMoreMoney* s = e.next()) {
    s->print(); delete s;
  }
  return 0;
}
```

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;
}
```

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

```
…

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: 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;
}
```

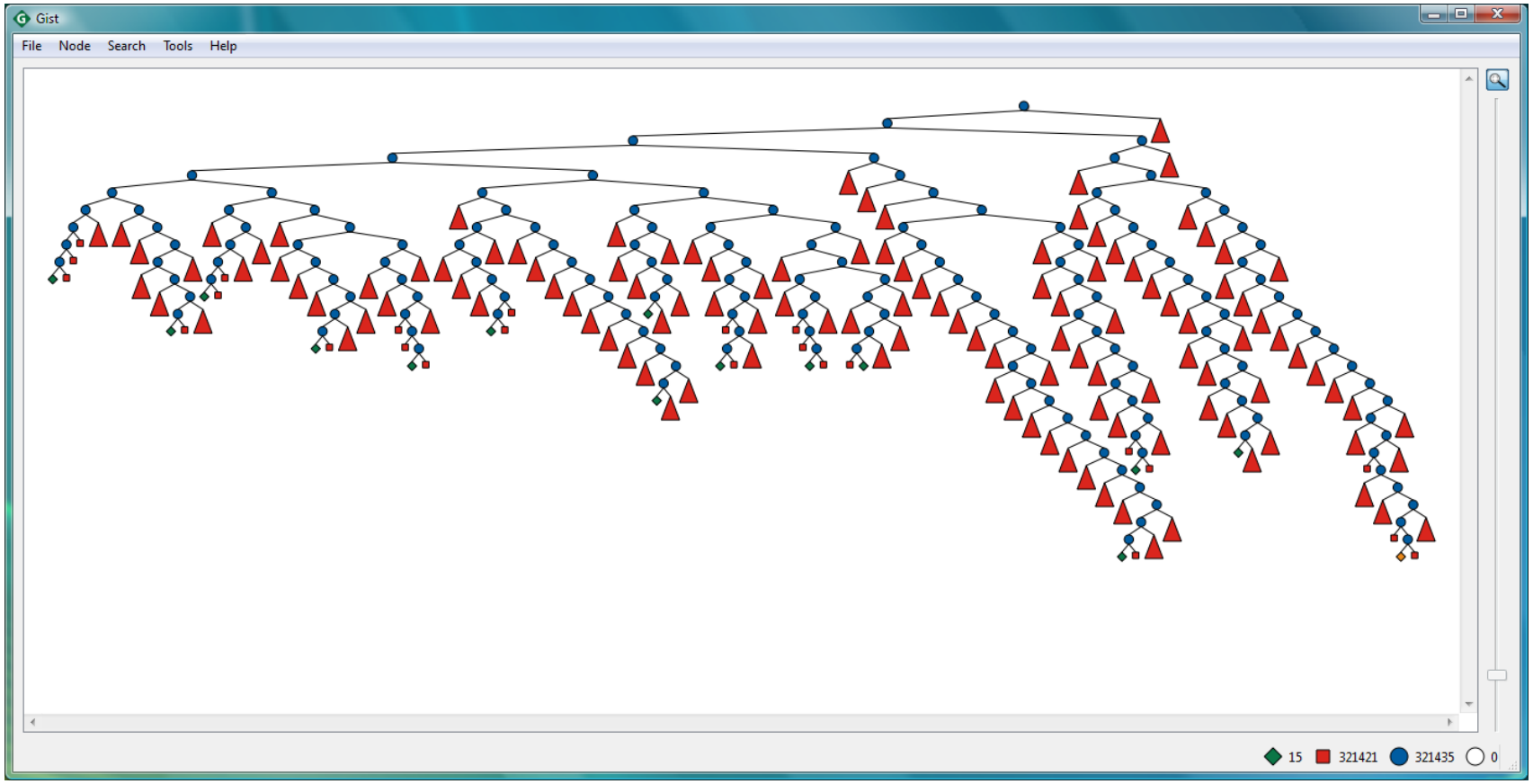
Gecode Gist

- 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

```
  SEND
+ MOST
-------
= MONEY
```

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(l[1]), n(l[2]), m(l[4]), o(l[5]), y(l[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);
}
```

value of any next solution

value of current best solution b

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