Constraint Programming…
…and Oz

Christian Schulte
School of Information and Communication Technology
KTH – Royal Institute of Technology, Sweden
`web.it.kth.se/~schulte/`

KTH Information and Communication Technology

Lecture Goals

- What is constraint programming?
  - reminder of basic aspects
  - discussion of advanced aspects
- How to use constraint programming?
  - solving principles
  - modelling applications
- Constraint programming and Oz
  - setup
  - unique aspects: programmable constraint services
  - advantages and disadvantages

Approach

- Baseline
  - modelling principles
  - solving principles
  - overview character (complement book)
- Excursions
  - how to do it in Oz
  - strong propagation
  - programming constraint services

Constraint Programming

Basic aspects

Constraint Programming

- Modelling and solving combinatorial problems

start with a first toy problem

Send More Money (SMM)

- Find distinct digits for letters, such that

```
    SEND
+   MORE
= MONEY
```

Constraint Model for SMM

- Variables and values
  `S,E,N,D,M,O,R,Y ∈ {0,…,9}`
- Constraints

```
  distinct(S,E,N,D,M,O,R,Y)
               1000×S+100×E+10×N+D
 +             1000×M+100×O+10×R+E
 = 10000×M+1000×O+100×N+10×E+Y
  S≠0      M≠0
```

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 7

Solving SMM

- Find values for variables
  such that
  all constraints satisfied
- Enumerate values, test constraints…
  …poor: we can do better than that!

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 8

Constraint Programming

- Compute with set of **possible** values
  - as opposed to assignments
- Prune impossible values
  - constraint propagation
- Search
  - distribute — search tree of simpler subproblems
  - explore — find solution in tree

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 9

Propagation for SMM

- Results in

```
S=9  E∈{4,…,7} N∈{5,…,8} D∈{2,…,8}
M=1  O=0       R∈{2,…,8} Y∈{2,…,8}
```

- Propagation **alone** not sufficient!
  - create simpler sub-problems
  - distribution and exploration

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 10

Principles: Constraint Propagation

Important Concepts

- Constraint store
- Basic constraint
- Propagator
- Non-basic constraint
- Constraint propagation

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 12

Constraint Store

x∈{3,4,5} y∈{3,4,5}

- Stores basic constraints
  - map variables to possible values

Constraint Store

finite domain constraints

x∈{3,4,5} y∈{3,4,5}

- Stores basic constraints
  - map variables to possible values

Constraint Store

x∈{3,4,5} y∈{3,4,5}

- Stores basic constraints
  - map variables to possible values
- Domains: finite sets, real intervals, trees, …

Propagators

- Implement non-basic constraints

  `distinct(`x_1`,…,`x_n`)`

  `x + 2×y = z`

  - smart algorithmic components

Propagators

x≥y   y>3

x∈{3,4,5} y∈{3,4,5}

- Amplify store by constraint propagation

Propagators

x≥y   y>3

x∈{3,4,5} y∈{3,4,5}

- Amplify store by constraint propagation

Propagators

$x \geq y$ $y>3$

$x \in \{3,4,5\}$ $y \in \{4,5\}$

- Amplify store by constraint propagation

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 19

Propagators

$x \geq y$ $y>3$

$x \in \{3,4,5\}$ $\mathbf{y \in \{4,5\}}$

- Amplify store by constraint propagation

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 20

Propagators

$\mathbf{x \geq y}$ $y>3$

$x \in \{4,5\}$ $y \in \{4,5\}$

- Amplify store by constraint propagation

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 21

Propagators

$x \geq y$ $y>3$

$x \in \{4,5\}$ $y \in \{4,5\}$

- Amplify store by constraint propagation
- Disappear when done (entailed)
  - no more propagation possible

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 22

Propagators

$x \geq y$

$x \in \{4,5\}$ $y \in \{4,5\}$

- Amplify store by constraint propagation
- Disappear when done (entailed)
  - no more propagation possible

2006-04-25 Constraint Programming... ...and Oz, C. Schulte 23

Computation Space

$x \geq y$ $y>3$

$x \in \{4,5\}$ $y \in \{4,5\}$

- Store with connected propagators

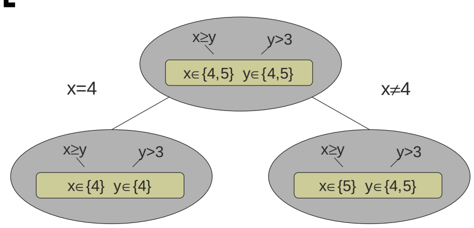
2006-04-25 Constraint Programming... ...and Oz, C. Schulte 24

Principles: Search

Important Concepts

- Distribution
- Exploration
- Heuristics
- Best solution search

Distribution (Branching)



- Yields spaces with additional constraints
- Enables further constraint propagation

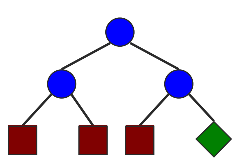
Distribution Strategy

- Pick variable x with at least two values
- Pick value n from domain of x
- Distribute with

  x=n and x≠n

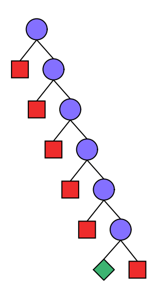
- Part of model

Search



- Iterate propagation and distribution
- Orthogonal: distribution ⇆ exploration
- Nodes:
  - **Unsolved** • **Failed** • **Succeeded**

SMM: Solution



```
    SEND
+   MORE
= MONEY

    9567
+   1085
= 10652
```

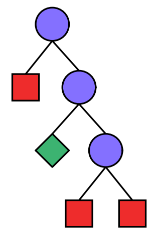
Heuristics for Distribution

- Which variable
  - least possible values (first-fail)
  - application dependent heuristic
- Which value
  - minimum, median, maximum

    $x=m$ or $x \neq m$
  - split with median m

    $x<m$ or $x \geq m$
- Problem specific

SMM: Solution With First-fail



```
  SEND
+ MORE
= MONEY

  9567
+ 1085
= 10652
```


Send Most Money (SMM++)

- Find distinct digits for letters, such that

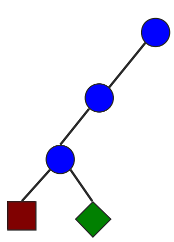
```
  SEND
+ MOST
= MONEY
```

and `MONEY` maximal

Best Solution Search

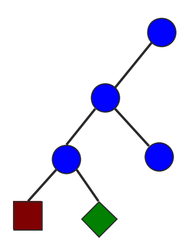
- Naïve approach:
  - compute all solutions
  - choose best
- Branch-and-bound approach:
  - compute first solution
  - add “betterness” constraint to open nodes
  - next solution will be “better”
  - prunes search space

Branch-and-bound Search



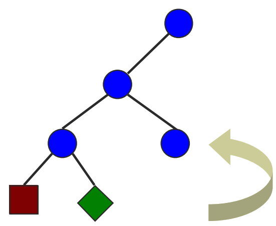
- Find first solution

Branch-and-bound Search



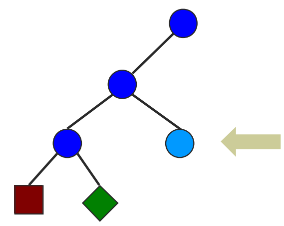
- Explore with additional constraint

Branch-and-bound Search



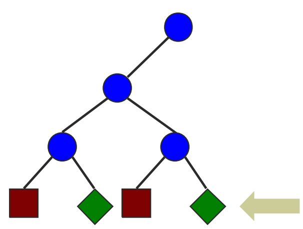
- Explore with additional constraint

Branch-and-bound Search



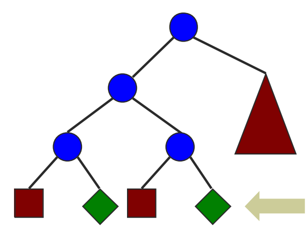
- Guarantees better solutions

Branch-and-bound Search



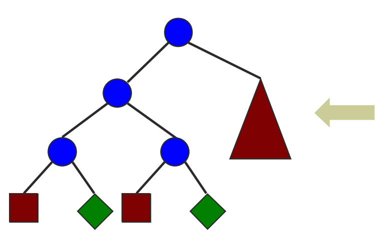
- Guarantees better solutions

Branch-and-bound Search



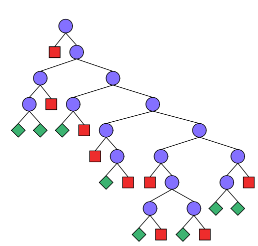
- Last solution best

Branch-and-bound Search



- Proof of optimality

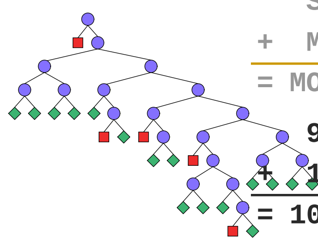
SMM++: Branch-and-bound



```
   SEND
+  MOST
= MONEY

   9782
+  1094
= 10876
```

SMM++: All Solution Search



```
  SEND
+ MOST
= MONEY

  9782
+ 1094
= 10876
```


Excursion: Constraint Programming in Oz

Basic setup

SMM in Oz

- Program script
  - script implements model
  - unary procedure: argument (root variable) is solution
- Script
  - introduce variables
  - basic constraints
  - post constraints
  - create branching

Oz Script for SMM: Solution and Basic Constraints

```
proc {SMM Sol}
   S E N D M O R Y
in
   Sol=smm(s:S e:E n:N d:D m:M o:O R:r y:Y)
   Sol ::: 0#9
   …
end
```


Oz Script for SMM: Post Propagators

```
proc {SMM Sol}
   …
   {FD.distinct Sol}
   S\=:0 M\=:0
              1000*S+100*E+10*N+D
            + 1000*M+100*O+10*R+E
   =: 10000*M+1000*O+100*N+10*E+Y
   …
end
```


Oz Script for SMM: Distribution Strategy

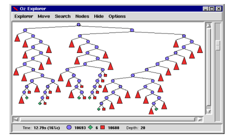
```
proc {SMM Sol}
   …
   {FD.distribute naive Sol}
end
```

Complete Oz Script for SMM

```
proc {SMM Sol}
   S E N D M O R Y
in
   Sol=smm(s:S e:E n:N d:D m:M o:O R:r y:Y)
   Sol ::: 0#9
   {FD.distinct Sol}
   S\=:0 M\=:0
              1000*S+100*E+10*N+D
             +1000*M+100*O+10*R+E
   =: 10000*M+1000*O+100*N+10*E+Y
   {FD.distribute naive Sol}
end
```


Solving SMM in Oz

`{ExploreOne SMM}`



- Use Oz Explorer
  - interactive, visual search
  - allows access to nodes in search tree
  - gain insight into propagation and distribution
- Other engines available

Modeling SMM++

- Constraints and branching as before
- Order among solutions with constraints
  - so-far-best solution `S,E,N,D,M,O,T,Y`
  - current node **`S,E,N,D,M,O,T,Y`**
  - constraint added

    `10000×M+1000×O+100×N+10×E+Y`

    `<`

    `10000×`**`M`**`+1000×`**`O`**`+100×`**`N`**`+10×`**`E`**`+`**`Y`**

Summary

What is constraint programming?

Modeling and Solving

applications

- Modeling
  - variables with domain
  - constraints to state relations
  - branching strategy
  - solution ordering
- Solving
  - constraint propagation
  - constraint branching
  - search tree exploration

principles

Application Areas

- Timetabling
- Scheduling
- Crew rostering
- Resource allocation
- Workflow planning and optimization
- Gate allocation at airports
- Sports-event scheduling
- Railroad: track allocation, train allocation, schedules
- Automatic composition of music
- Genome sequencing
- Frequency allocation
- …

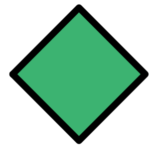
Why Does CP Matter?

- Middleware for combining smart algorithmic components (propagators)
  - scheduling
  - graphs
  - flows
  - …

  …for strong propagation
- Essential extra constraints…

  …for flexibility

SMM: Strong Propagation



```
  SEND
+ MORE
= MONEY

  9567
+ 1085
= 10652
```


Excursion: Strong Propagation

Example: Distinct Propagator

- Naive distinct propagator
  - wait until variable becomes assigned
  - remove value from all other variables
- Domain-consistent distinct propagator
  - only keep values appearing in a solution to constraint
  - essential for many problems

Distinct Propagator: Hall Sets

- Direct approach: Hall sets
  - Van Beek, Quimper, et. al. [CP 2004]
- Set $\{x_1, \ldots, x_n\}$ of variables Hall set, iff set of values $D(x_1) \cup \ldots \cup D(x_n)$ has cardinality n
- Pruning
  - find Hall set H
  - prune values in H from all other variables

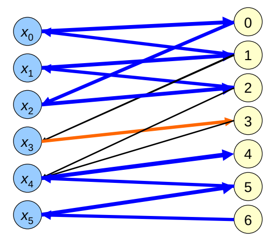
Domain-consistent Distinct Propagator

- Can be propagated efficiently
  - $O(n^{2.5})$ is efficient
  - breakthrough: Régin, A filtering algorithm for constraints of difference in CSPs, AAAI 1994.
- Uses graph algorithms
  - insight on problem structure
  - relation between solutions of constraint and properties of graph

Approach

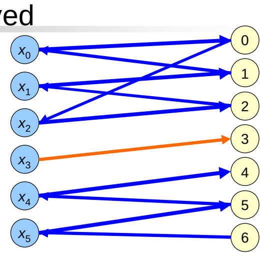
- Construct a variable-value graph
  - bipartite graph
    - variable nodes → value nodes
- Characterize solutions in graph
  - maximal matchings
- Use matching theory
  - one matching can describe all matchings
- Remove edges not representing solutions

All Marked Edges



Edges Removed

- Remove
  - $1 \rightarrow x_3$
  - $2 \rightarrow x_4$
  - $3 \rightarrow x_4$
- Keep
  - $x_3 \rightarrow 3$
  - matched!



Significance

- Constraint programming identified as a strategic direction in computer science research
  - [ACM Computing Surveys, December 1996]
- Applications are ubiquitous

Modeling

Wish and reality

Modeling Strategy

- Understand problem
  - identify variables
  - identify constraints
  - identify optimality criterion
- Attempt initial model simple
  - try on examples to assess correctness
- Improve model much harder
  - scale up to real problem size

Modeling Techniques

- Find variables and values
  - decrease symmetries
  - dual models: change values and variables
  - combine models: channeling
- Increase propagation
  - strong methods
  - redundant (implied) constraints but non-redundant propagation

Modeling Techniques

- Remove useless solutions
  - symmetrical: symmetry breaking
  - same cost: dominance constraints
- Good heuristic for distribution
  - which variable: size, degree, regret, …
  - how to split domains: single value, bisection, …
  - in which order to split: minimum, median, maximum, …

Modelling: Wish and Reality

- Wish: entirely declarative
  - only state model
- Reality: solving specific
  - good constraints
  - good propagators
  - good heuristics
  - good exploration

Why is Oz Good?

Getting Started with Mozart

- Use tutorial shipped with Mozart

  Schulte, Smolka. Finite Domain Constraint Programming in Oz. A Tutorial.
- Little knowledge on Oz required
  - scripts are unary procedures
  - orders are binary procedures
  - introducing variables
  - conditional statements
  - calling functions and procedures
  - tuples (records) for solutions
  - loops for iterating over tuples

Mozart Features

- Finite domain integers
  - general purpose: arithmetic, …
  - scheduling
- Finite sets
- Search: orthogonal exploration
  - basic + interactive + parallel + …
- Tools
  - OPI, Explorer, Browser, Inspector, …

Mozart Advantages

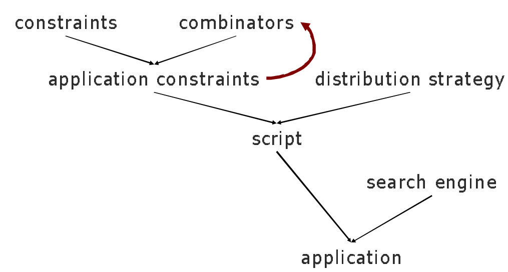
- Incremental and interactive development
  - understand problem and refine model
  - rich tool support
- Integration with concurrency and distribution
  - multi agent applications
- Well documented
- Freely available
- Programmable and Extensible

Excursion: Programming Constraint Services

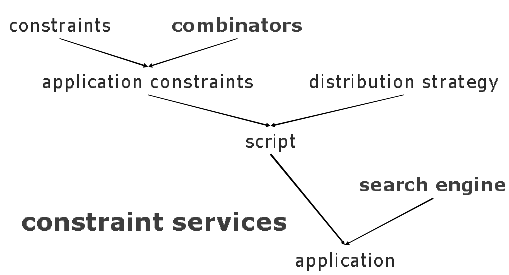
Mozart Unique Features

- Constraint services can be programmed at high-level
  - search engines
  - combinators

Application Architecture



Application Architecture



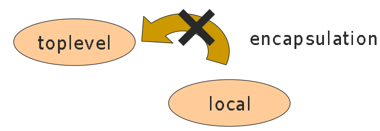
State of the Art: Search

- Not programmable or difficult
  - Limited to few strategies (first, all, best)
  - Systems unable to keep pace
    - New strategies: LDS [Harvey&Ginsberg 1995], …
- Based on trailing
  - Incompatible with concurrency
  - Fixed to depth-first

State of the Art: Combinators

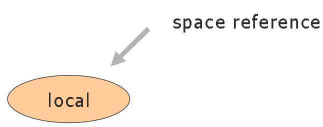
- Only mechanism: reification
  - Reflect validity of constraint to 0/1 variable
  - Combine 0/1 variables with combinators
- Problematic: All-or-nothing approach
  - All constraints must offer reification
  - Incompatible with programming
  - No propagation in common situations

Local Computation Spaces



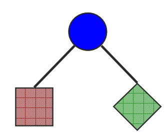
- Delegate speculative computation to local space
  - Failure is encapsulated
  - Execution as in toplevel space

First-class Spaces



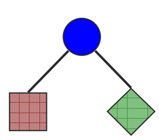
- Promote spaces to first-class status
  - Enable programming
- Constraint services
  - Programmed from operations on spaces

Space Creation



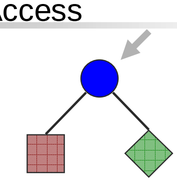
- NewSpace : Script $\longrightarrow$ Space
  - Script (procedure): problem definition
  - Root variable R: entry point to solution
  - Runs {Script R} in space

Status Access



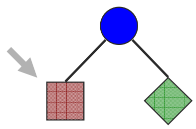
- Ask : Space $\longrightarrow$ Status
  - Synchronizes on stability
  - Returns status
    - alternatives(2)
    - failed
    - succeeded

Status Access



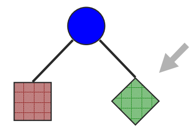
- Ask : Space $\longrightarrow$ Status
  - Synchronizes on stability
  - Returns status
    - alternatives(2)
    - failed
    - succeeded

Status Access



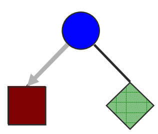
- Ask : Space $\rightarrow$ Status
  - Synchronizes on stability
  - Returns status
    - alternatives(2)
    - failed
    - succeeded

Status Access



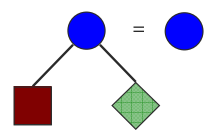
- Ask : Space $\rightarrow$ Status
  - Synchronizes on stability
  - Returns status
    - alternatives(2)
    - failed
    - succeeded

Committing to Alternatives



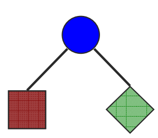
- Commit : Space $\times$ Int
  - Triggers reduction of selected alternative

Cloning Spaces

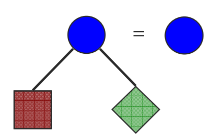


- Clone : Space $\rightarrow$ Space
  - Synchronizes on stability
  - Returns clone of space

Exploring Both Alternatives

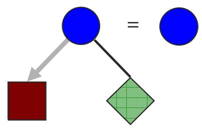


Exploring Both Alternatives



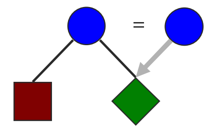
1. Create clone

Exploring Both Alternatives



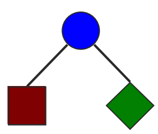
1. Create clone
2. Commit original to first alternative

Exploring Both Alternatives



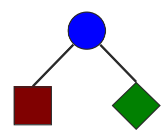
1. Create clone
2. Commit original to first alternative
3. Commit clone to second alternative

Exploring Both Alternatives



1. Create clone
2. Commit original to first alternative
3. Commit clone to second alternative

Solution Access



- Merge : Space $\longrightarrow$ Solution
  - Merges space with toplevel space
  - Returns root variable

All Solution Exploration

```
fun {DFE S}
   case {Ask S}
   of failed    then nil
   [] succeeded then [{Merge S}]
   [] alternatives(2) then C={Clone S} in
      {Commit S 1} {Commit C 2}
      {Append {DFE S} {DFE C}}
   end
end
```


Recomputation

- Recompute node on demand
  - Store path as list of integers
  - Memory
    - Depends on depth only!
    - Independent of problem size
- Recomputation alone is infeasible
  - Tree depth N: N/2 times exploration steps

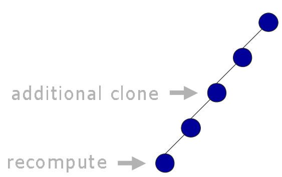
Fixed Recomputation

- Hybrid: clone from time to time
- Maximal recomputation distance (MRD):
  - Limit recomputation steps to MRD
  - Decreases memory by factor of MRD
- Optimistic attitude
  - Assumes that search does not go wrong
  - Controlled by MRD

Why Recomputation Matters

- Memory
  - Independent of problem size
- Little search with deep search trees
  - Exponential number of nodes: smaller fraction
  - Optimistic attitude fits well
- Clustered failures
  - Likely: not only last decision wrong
  - Fail once: fail soon again
  - Requires more pessimistic attitude

Adaptive Recomputation

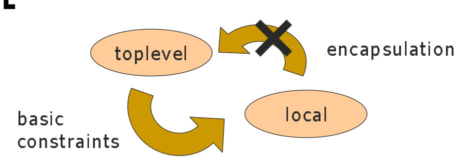


- Additional clone while recomputing
  - Strategy: in between
- Next failure: 50% exploration steps

Combinators

- Spaces for combinators
- Control and status
- Programming: Negation combinator
- Discussion

Nested Propagation

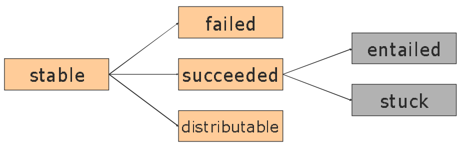


- Computation with global information
  - Still obeys encapsulation
- Leads to tree of spaces: space tree

Stability

- Space is blocked
  - May become runnable later
  - Due to constraints in superordinated space
- Space is stable
  - Remains blocked forever
  - No constraints on global variables
  - No synchronization on global variables
- Pioneered by AKL [Haridi&Janson,1992]

Space Status



- Entailed
  - No computations left
- Stuck
  - Computations remain: typically programming error

Negation Combinator: Idea

- Create space S executing statement
- Synchronize until
  - S is failed: okay
  - S is entailed: fail
  - S is stuck: programming error
- Must execute concurrently

Negation Combinator: Idea

- Create space S executing statement
- **Synchronize** until
  - S is failed: okay
  - S is entailed: fail
  - S is stuck: programming error
- Must execute **concurrently**

Negation Combinator

```
proc {Not P}
   S={NewSpace P}
in thread
      case {Status S}
      of failed    then skip
      [] entailed then fail
      [] stuck     then {Error}
      end
   end
end
```


Other Combinators

- Generalized reification
- Disjunction
  - Failure-based reduction
  - Entailment-based reduction
  - Andorra-style
    - Reduction by propagation
    - Reduction by search
- Committed-choice conditional

Discussion

- Combinators are composable
  - Applicable to arbitrary constructions
  - Including: obtained by combination
  - Combination compatible with programming
  - Known as deep-guard combinators
- Combinators are concurrent
  - Concurrency for propagation
  - Synchronization

Programmable and Extensible

- Programming [Oz]
  - scripts
  - distribution
  - exploration (Explorer, parallel search, ...)
  - combination mechanisms
- Extending [CPI in C++]
  - propagators
  - variables

Why is Oz Bad?

Mozart Disadvantages

- Small set of good propagators
  - "global constraints"
  - will worsen due to lack of contributors
- Inflexible interface for propagators
  - unrealistic assumptions
- Initial burden to learn Oz
- Not easy to embed

Mozart Disadvantages

- Constraint propagation slow...
- Recomputation too eager
  - excludes batch recomputation
  - save propagation during recomputation
- Implementation way too complex
  - too compositional: search, encapsulation, local variables, ...
  - hard to maintain

Mozart Disadvantages

- My opinion (suspicion?): who will further develop and maintain
  - systems need contineous care!

Summary

Constraint Programming with Mozart

- Powerful technology for combinatorial optimization
- Mozart free, programmable, and accessible system for constraint programming
  - requires more propagators
- Most effort is in modeling (understanding)
  - not dependent on Oz and Mozart

My Interest: Gecode

www.gecode.org

- freely available C++ library
  - simple, efficient, open
- used in research and education
  - KTH, UU, Louvain

My Focus on Interest: Gecode

- How to construct generic and minimal kernel for constraint programming
- How to better coordinate propagation
  - variables: simple, but too limited?
- How to assist in modelling?
  - find right propagators
- Efficiency as consequence of simplicity
  - beats Mozart by one order of magnitude