Christian Schulte  
KTH – Royal Institute of Technology, Sweden  
schulte@imit.kth.se

Goal of Appetizer
- Underlying principles
- Sketch of how to do in Oz
- Modeling techniques
- Mozart advantages and disadvantages

…no time for hands-on tutorial

Constraint Programming
- Modeling and solving combinatorial problems
- Start with a first toy problem

Send More Money (SMM)
- Find distinct digits for letters, such that

\[
\begin{align*}
\text{SEND} + \text{MORE} &= \text{MONEY} \\
S, E, N, D, M, O, R, Y \in \{0, \ldots, 9\}
\end{align*}
\]

Constraint Model for SMM
- Variables and values
  \( S, E, N, D, M, O, R, Y \in \{0, \ldots, 9\} \)
- Constraints
  \[
  \begin{align*}
  \text{distinct}(S, E, N, D, M, O, R, Y) \\
  1000 \times S + 100 \times E + 10 \times N + D \\
  + 1000 \times M + 100 \times O + 10 \times R + E \\
  = 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y \\
  S \neq 0 \quad M \neq 0
  \end{align*}
  \]

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

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

**Propagation for SMM**

- Results in
  - \( S = 9 \) \( E \in \{4, \ldots, 9\} \) \( N \in \{5, \ldots, 8\} \) \( D \in \{2, \ldots, 8\} \)
  - \( M = 1 \) \( O = 0 \) \( R \in \{2, \ldots, 8\} \) \( Y \in \{2, \ldots, 8\} \)
- Propagation alone not sufficient!
  - create simpler sub-problems
  - distribution and exploration

**Overview**

- Principles
  - constraint propagation
  - search
- Summary of principles and significance
- Modeling techniques
- Oz and Mozart

**Principles: Constraint Propagation**

**Important Concepts**

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

**Constraint Store**

- Stores basic constraints map variables to possible values

- \( x \in \{3,4,5\} \) \( y \in \{3,4,5\} \)
Constraint Store

- Stores finite domain constraints
  - map variables to possible values

Domains: finite sets, real intervals, trees, ...

Propagators

- Implement non-basic constraints

\[
\text{distinct}(x_1, \ldots, x_n)
\]

\[
x + 2xy = z
\]

- Smart algorithmic components

Propagators

- Amplify store by constraint propagation

\[
x \geq y, \quad y > 3
\]

\[
x \in \{3,4,5\}, \quad y \in \{3,4,5\}
\]
Propagators

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

x≥y y≥3

\(x \in \{3,4,5\} \quad y \in \{4,5\}\)

Computation Space

- Store with connected propagators

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 \( x\neq n \)
- Part of model

Search

- Iterate propagation and distribution
- Orthogonal: distribution \( \leftrightarrow \) exploration
- Nodes:
  - Unsolved
  - Failed
  - Succeeded

SMM: Solution

```
SEND
+ MORE
= MONEY
9567
+ 1085
= 10652
```

Solving 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}
  SENDMORY
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}
  SENDMORY
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

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

\[
\begin{align*}
\text{SEND} \\
+ \quad \text{MORE} \\
= \quad \text{MONEY} \\
9567 \\
+ \quad 1085 \\
= \quad 10652
\end{align*}
\]

Send Most Money (SMM++)

- Find distinct digits for letters, such that

\[
\begin{align*}
\text{SEND} \\
+ \quad \text{MOST} \\
= \quad \text{MONEY}
\end{align*}
\]

and \text{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

- Explore with additional constraint
Branch-and-bound Search

- Guarantees better solutions

= Last solution best

= Proof of optimality

Modeling SMM++

- Constraints and branching as before
- Order among solutions with constraints
  - so-far-best solution
  - current node
  - constraint added

\[
10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y < 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y
\]

SMM++: Branch-and-bound

\[
\begin{align*}
\text{SEND} & + \quad \text{MOST} \\
& \quad = \quad \text{MONEY} \\
\hline
9782 & + \quad 1094 \\
\hline
& \quad = \quad 10876
\end{align*}
\]
Modeling and Solving

- **Modeling**
  - variables with domain
  - constraints to state relations
  - branching strategy
  - solution ordering

- **Solving**
  - constraint propagation
  - constraint branching
  - search tree exploration

**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
- ...

**Constraint Programming in Oz**
- **Script**
  - implements constraint model

- **Solution order**
  - defined by binary procedure

- **Exploration**
  - interactive: Oz Explorer
  - search module: plain, best, parallel, ...

**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**

\[
\begin{align*}
\text{SEND} + \text{MORE} &= \text{MONEY} \\
9567 + 1085 &= 10652
\end{align*}
\]

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

**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
- 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, …
Oz and Mozart

Getting Started with Mozart

- Use tutorial shipped with Mozart
- 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

Programmable and Extensible

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

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
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