Ten Years of MiniZinc

Zn

Peter J. Stuckey

Overview

- History of MiniZinc
  - How did it come into being
  - Major changes
- The Current State of MiniZinc
  - Features that have been developed
- The Future of MiniZinc
- Conclusion

“Alone we can do so little; together we can do so much.” – Helen Keller

Ignasi Abio, Roberto Amidini, Maria Garcia de la Banda, Ralph Becket, Gustav Bjordal, Sebastian Brand, Geoffrey Chu, Michael Codish, Jip Dekker, Nick Downing, Thibaut Feydy, Pierre Flener, Graeme Gange, Tias Guns, David Hemmi, Kevin Leo, Kim Marriott, Chris Mears, Nick Nethercote, Justin Pearson, Andrea Rendl, Andreas Schutt, Joseph Scott, Guido Tack, Mark Wallace

History of MiniZinc

Genesis of MiniZinc

- CP2006 Nantes:
- Workshop: Next 10 years of CP
  - some speakers went WAY overtime
  - one large question resonated
- **We need a standard for writing CP models**

Genesis of MiniZinc

- The G12 Project
  - commenced in 2004
  - Zinc: a high level modelling language
  - Cadmium: a model transformation language
  - Mercury: a solver compilation language
    - around since 1995, used to build solver
- Monolithic system

Genesis of MiniZinc

- MiniZinc is a simplification of Zinc
  - stripping out complex things
    - constrained types, functions, records, tuples, enums
- Key difference
  - MiniZinc interprets models to FlatZinc
  - Zinc compiles models to Mercury

MiniZinc Key Features

- Separation of model and data
- Expressive enough type system
  - int, float, bool, set of int, arrays
- Predicates
  - crucial to handling global constraints
- Maps to FlatZinc
  - minimal interface to a solver: vars + constraints + objective

Key Aims of MiniZinc

- **Easy for solver writers to support**
  - specialized globals library
  - FlatZinc parser
    - many used the Gecode FlatZinc parser to start
- Easy for modellers to use
- Open source

Hidden History

- MiniZinc paper REJECTED by CPAIOR2007
  - Admittedly we didn't have a full implementation
  - MiniZinc + FlatZinc definition
  - Cadmium translation from MiniZinc to FlatZinc
  - No experiments in the paper
- But the key problem
  - MiniZinc: A **Standard** Language for Modelling CP Problems

ing language

)

Versions of MiniZinc

- 2007: v0.6 and CP paper
- 2008: v0.7, 0.8, first MiniZinc challenge
- 2009: v1.0, BSD license
- 2010: v1.1, rewritten, more efficient mzn2fzn, v1.2, CP-viz, tutorial
- 2011: v1.3, 1.4, improved output
- 2012: v1.5, 1.6
- 2013: v2.0 beta: minizinc.org
- 2014: v2.0
- 2015: MiniZinc IDE, MiniZinc bundle
- 2016: v2.1 MiniZinc and MiniZinc IDE

Significant Advances

- Relational Semantics
- User Defined Functions
- Option Types
- Enumerated Types

Relational Semantics

- What are the solutions of model A
  - `var 0..1: y; constraint 1/y = 2 \/ y < 1`
- What are the solutions of model B
  - `var 0..1: y; constraint not(1/y = 1)`
- Three possibilities
  - **Strict**: A: {}, B: {}
  - **Kleene**: A: {y = 0}, B: {}
  - **Relational**: A: {y = 0}, B: {y= 0}
- MiniZinc implements the relational semantics
  - most modelling languages implement none!

User Defined Functions

- Were in Zinc, but not MiniZinc
- Needed
  - for better common subexpression elimination
  - driven by machine learning examples
- Introduced need for local constraints (not in Zinc)
- Advantages (beyond better CSE)
  - simplify the built ins of MiniZinc
  - improved functional global handling (1/3 globals are functional)
  - better translation to solvers (in particular for MIP)

Option Types

- Representing decisions that are only sometimes relevant
  - like optional interval variables in CP optimizer
- Mainly added to support
  - iteration over variable sets, variable where conditions
- Syntax

```
var set of 1..12: x;
constraint y = sum(i in x)(a[i]);
```

- Actual meaning

```
var set of 1..12: x;
constraint y = sum(i in 1..12)
                  (if i in x then a[i] else <> endif);
```

Option Types

- A new value <> meaning optional
  - in a constraint it is ignored
    - e.g. `alldifferent([<>, 1,4, <>])`
  - acts as an identity in an expression where possible
    - e.g. `<> + 4 = 4`
  - acts as annihilator otherwise
    - e.g `<> - 4 = <>`
- Translated away by default
- Eases modelling, but more work required on globals

Enumerated Types

- Back from Zinc!
- Implemented as type erasure
  - simply a type artifice for integers
- Aim to catch type errors in models

```
array[POS] of var PERSON: order;
enum PERSON = {ann, bob, cal, dan, edna, fred};
constraint order[fred] > order[home];
```

- Enumerated types
  - are ordered as in the definition: `ann < bob`
  - coerce to integers when used as integers: `dan + 1 = 5`

Current State of MiniZinc

Statistics

- 441 citations of the paper
  - 66 from last year, and growing
- 40,000 downloads of MiniZinc package
  - around 50 a day
  - 50% linux, 25% windows, 25% mac

Massive Online Open Courses

- Modeling for Discrete Optimisation
  - 8 week Coursera course on Minizinc
  - launched late 2015, closed mid 2017
  - 8000 students

Massive Online Open Courses

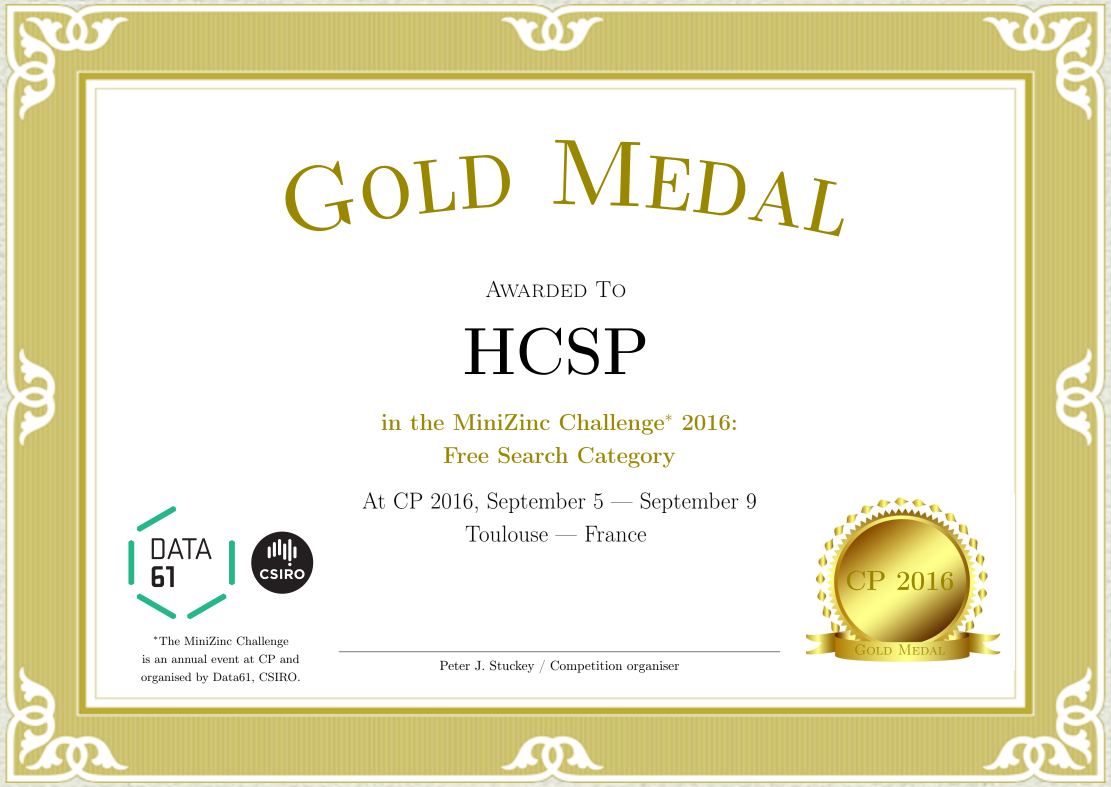
- Two 4 week courses with Jimmy Lee
  - Basic Modeling for Discrete Optimization
  - Advanced Modeling for Discrete Optimization
  - launched Jan 2017
  - Chinese and English
  - 8000+ students

COURSERA - ENG - 16-044 - CUHK-UoM MOOC E - Course 01 Promo.mp4

镇守华容道

MiniZinc Challenge

- Since 2008 (10th running this year)
- Collected over 150 benchmark problems
  - many real world interesting problems
- This year
  - 16 solver (variations) submitted
  - 8 internal submissions
  - 5 categories:
    - Fixed, free, parallel, open, local_search
- Results on Wednesday



GOLD MEDAL

AWARDED TO

HCSP

in the MiniZinc Challenge* 2016:
Free Search Category

At CP 2016, September 5 — September 9
Toulouse — France

DATA 61 CSIRO

*The MiniZinc Challenge
is an annual event at CP and
organised by Data61, CSIRO.

Peter J. Stuckey / Competition organiser

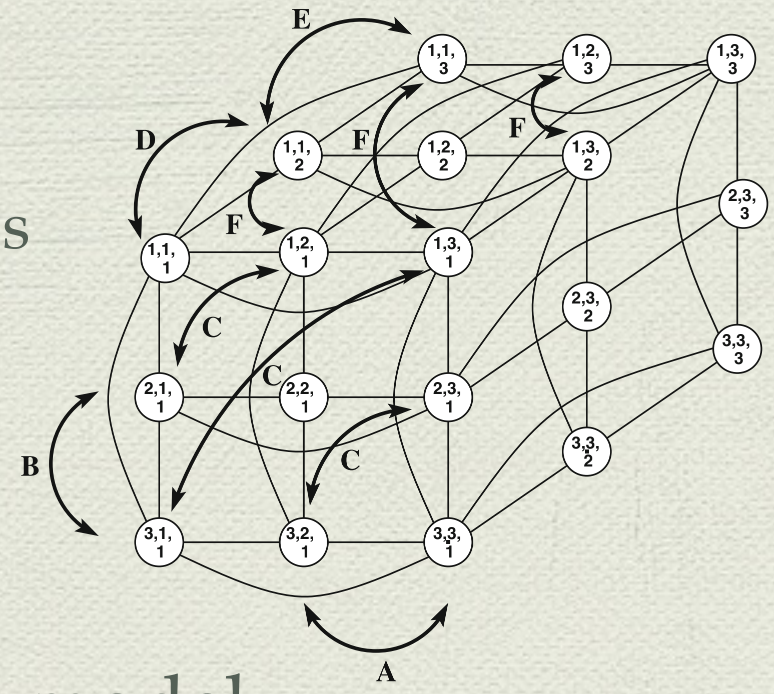
CP 2016
GOLD MEDAL

Long List of Features Coming Soon!

- Symmetry Detection (2009)
    - Dominance Detection (2015)
- Half reification (2011)
- Globals Detection (2013)
- Stochastic MiniZinc (2014)
- Multi-pass compilation (2015)
- MiniSearch (2015)
- Strings (2016)
- Auto tabling (2017)

Symmetry Detection (2009)

- Generate symmetries of small instances
  - find which symmetries generalize across instances
- Generate candidate model symmetries
  - ask the user or use theorem proving
- Add symmetry breaking (dynamic/static) to model
- Extension to dominance
  - separate out objective and/or some constraints
  - generate symmetries
  - convert to dominance constraints



Half Reification (2011)

- `constraint x >= 0 \/ (y < 0 /\ z = x + 1)`
- becomes by reification

```
var bool: b1 = (x >= 0);     b1 -> (x >= 0);
var bool: b2 = (y < 0);      b2 -> (y < 0);
var bool: b3 = (z = x+1);    b2 -> (z = x+1);
var bool: b4 = (b2 /\ b3);
constraint b1 \/ b4;         b1 \/ b2;
```

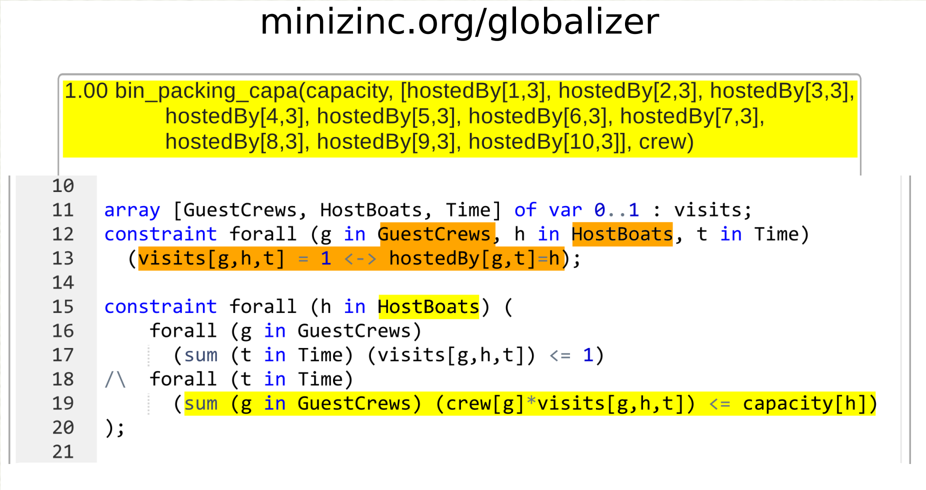
- Better translation

Half Reification (2011)

- Benefits
  - all globals can be half reified:
    - separate failure from propagation
  - less propagation (faster)
  - simplifies implementation of relational semantics
- Some solvers internally perform half-reification
  - CPX
  - LCG-glucose

Globals Detection (2013)

- Find global constraints which are implied by the model
    - Use structure of model to find sub-problems
    - Generate candidate global constraints
    - Rank the global candidates by
        - coverage by solutions, size of global
    - Present the globals to the user in ranked order



minizinc.org/globalizer

```
1.00 bin_packing_capa(capacity, [hostedBy[1,3], hostedBy[2,3], hostedBy[3,3],
          hostedBy[4,3], hostedBy[5,3], hostedBy[6,3], hostedBy[7,3],
          hostedBy[8,3], hostedBy[9,3], hostedBy[10,3]], crew)
```

```

array [GuestCrews, HostBoats, Time] of var 0..1 : visits;
constraint forall (g in GuestCrews, h in HostBoats, t in Time)
  (visits[g,h,t] = 1 <-> hostedBy[g,t]=h);

constraint forall (h in HostBoats) (
    forall (g in GuestCrews)
      (sum (t in Time) (visits[g,h,t]) <= 1)
/\  forall (t in Time)
      (sum (g in GuestCrews) (crew[g]*visits[g,h,t]) <= capacity[h])
);

```

- Was available as a web tool: minizinc.org/globalizer
- Highly important approach for naive modellers
    - gives a way to “lookup” the globals you need for your problems

Stochastic MiniZinc (2014)

- Extend MiniZinc to express stochastic problems
  - `:: stage(n)` annotation for pars and vars
  - `:: expected` value objectives
  - scenarios and scenario weights
- Three approaches: transformation + solving hybrid
  - deterministic equivalence (transformation only)
  - policy-based search
  - progressive hedging
- Was available: `minizinc.org/stochastic/`
  - new approach by Guido Tack and David Hemmi to be integrated

Multi Pass Compilation (2015)

- MiniZinc flattens to FlatZinc
  - many decisions made during flattening, e.g

```
var {2,4}: x; var {2,4}: y; var {2,4,5}: z;
constraint all_different([x,y,z]);
constraint x+y+z=12 -> y=max([x,y,z]);
```

  - becomes

```
var {2,4}: x; var {2,4}: y; var {2,4,5}: z;
constraint all_different([x,y,z]);
var 2..5: i0 = max([x,y,z])
var bool: b0 = (y = i0)
var bool: b1 = (x+y+z != 12)
constraint or(b0,b1);
```

MiniSearch (2015)

- Meta-search language for MiniZinc
- Principles
  - no new interaction with solver
    - post constraints, get next solution, stack of scopes
- A procedural language for solver control
  - an interpreter in C++
  - natively interacts with MiniZinc variables
  - manages solutions
- Expresses searches such as
  - lexicographic B&B, large neighbourhood search, and/or search, interactive optimization
- Available at `minizinc.org/minisearch/`

Multi Pass Compilation (2015)

- More information = better decisions

```
var {2,4}: x; var {2,4}: y; var {~~2,4,~~5}: z;
constraint all_different([x,y,z]);
var 2..5: i0 = ~~max([x,y,z])~~  5
var bool: b0 = ~~(y = i0)~~      false
var bool: b1 = (x+y+~~z~~5 != 12) true
constraint ~~or(b0,b1);~~
```

- finally

```
var {2,4}: x; var {2,4}: y; var {5}: z;
constraint x != y;
constraint x+y != 7;
```

Multi Pass Compilation (2015)

- Multi pass compilation
  - Gecode first pass: Other solver second pass
  - reduces model size: around 5%
  - reduces run time for MIP solvers: around 50%
  - can improve compile time, no worse than double

(Bounded) Strings (2016)

- We have extended MiniZinc with
- `var list of $T`: a sequence of type T
  - `$T` could be `int`, or an enumerated type `enum DNA = { A, C, G, T };`
- string constraints
  - (lex)order, concatenation, reverse, length, regular, gcc
- coercion: `array[int] of var int` coerces to `var list of int`
- Default: translated to integer constraints
- But: Gecode+S (native definition)

(Bounded) Strings (2016)

- Strings in MiniZinc provide
  - a standard way of writing string problems
  - a new challenge for CP solver implementors
  - see our paper on Friday

Auto Tabling (2017)

- Annotate a predicate as: `:: presolve(autotable);`

```
predicate rank_apart(var 1..52: a, var 1..52: b)
  = table(abs([a,b](a|-1b,2 mod 1,133)|in {1,522}51 |]);
```

- Solution are computed
    - predicate replaced by a table constraint
- Variations
    - call-based, and instance independent
- Benefits
    - improved solving time
    - automatic reformulation of poor models
- Not done in Australia

Other Stuff

- linearization library
  - MIP solvers now usable through MiniZinc
- C++ interface for MiniZinc
- JSON input/output for MiniZinc
- MiningZinc:
  - a special version for itemset mining

Future of MiniZinc

Upcoming Stuff

- New library
  - changes to FlatZinc
- Automatic checking/grading
  - see talk later in this workshop
- New Coursera course: Solving Technologies

New Library

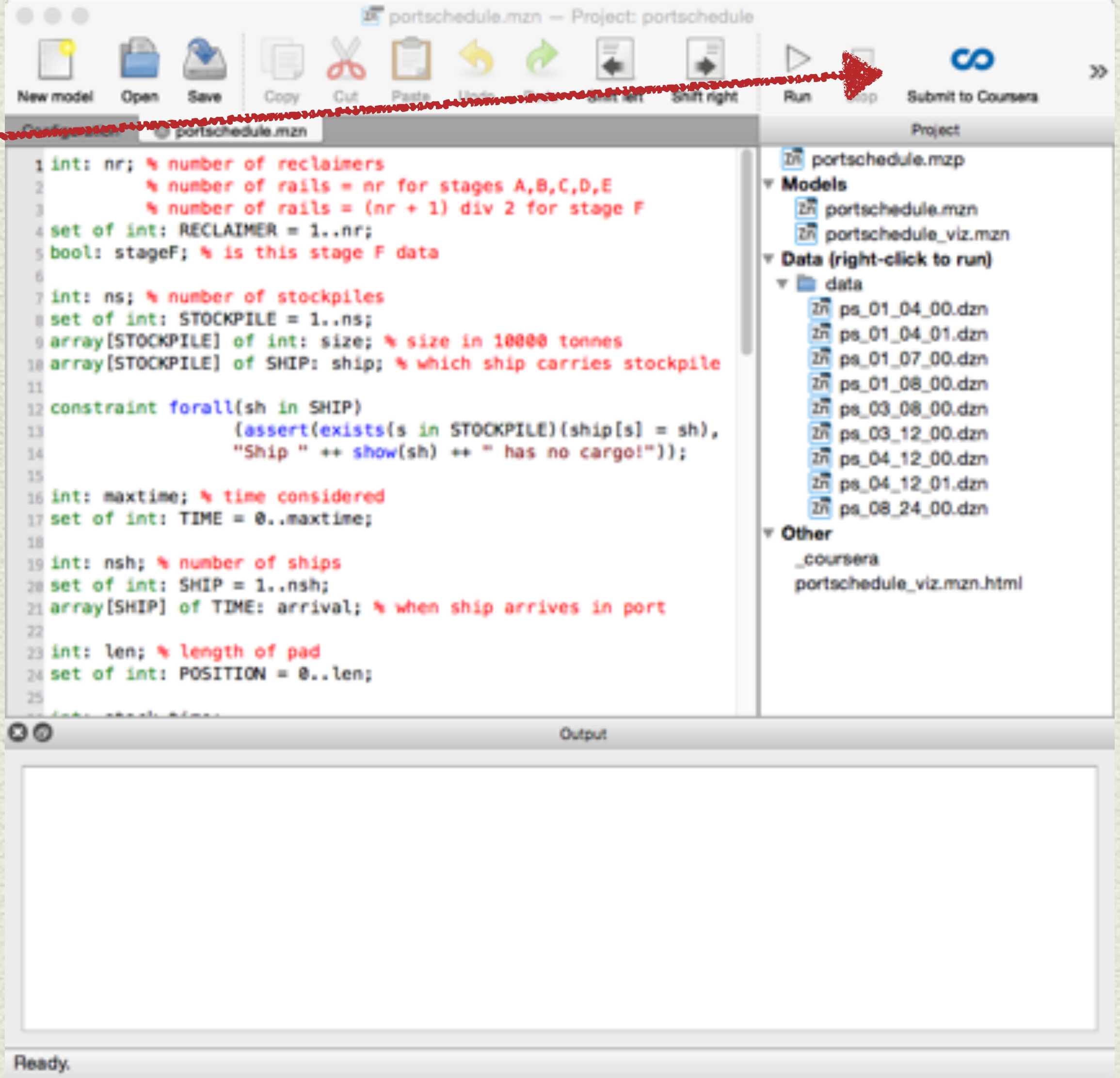
- Naming convention changing
    - `alldifferent` (MiniZinc level)
    - `fzn_alldifferent` (FlatZinc level)
        - solver implementors adjust this definition
- Enables better presolving
- Allows more reuse

New Library

- New graphs globals included
  - `bounded_path`, `connected`, `dag`, `path`, `reachable`, `steiner`, `tree`, `wst` (weighted spanning tree)
  - directed and undirected versions
- Some others: `cost_mdd`

Auto Checking Grading

- Infrastructure used for Coursera
- Build a standalone project with detailed feedback
- Checking in IDE
- Checking as web service
- More details later in MODREF



```
int: nr; % number of reclaimers
         % number of rails = nr for stages A,B,C,D,E
         % number of rails = (nr + 1) div 2 for stage F
set of int: RECLAIMER = 1..nr;
bool: stageF; % is this stage F data

int: ns; % number of stockpiles
set of int: STOCKPILE = 1..ns;
array[STOCKPILE] of int: size; % size in 10000 tonnes
array[STOCKPILE] of SHIP: ship; % which ship carries stockpile

constraint forall(sh in SHIP)
                 (assert(exists(s in STOCKPILE)(ship[s] = sh),
                 "Ship " ++ show(sh) ++ " has no cargo!"));

int: maxtime; % time considered
set of int: TIME = 0..maxtime;

int: nsh; % number of ships
set of int: SHIP = 1..nsh;
array[SHIP] of TIME: arrival; % when ship arrives in port

int: len; % length of pad
set of int: POSITION = 0..len;
```

Coursera Course: Solving

- Looking at solving technologies
  - Constraint programming
  - Mixed integer programming
  - Local search
- Using MiniZinc
- Fable based learning

The Further Future

- Better Search Annotations
  - complete search + LNS
  - local search
- Nested Constraint Programming
  - extending Stochastic MiniZinc
- ??????

Conclusion

- MiniZinc is a successful modelling language
    - ease of use, ease of learning
    - ease of solver support
    - expressiveness (except search expressiveness)
- A suite of standard benchmarks for CP
- Still lacking
    - ease of integration in applications
    - resources to integrate/maintain features

Questions

- What do you want from MiniZinc?
- What can you do for MiniZinc?