Abstract Modelling with Conjure

Dr Özgür Akgün (Oz)

School of Computer Science, University of St Andrews, UK

Satisfiability: Theory, Practice, and Beyond Workshop
19 April 2023, Berkeley

Motivation

Example

Opportunities


Motivation

Selected highlights

From today

- John Hooker: A bunch of isomorphic approaches. Why does one work better than another in practice?
- Torsten: importance of modelling for the ubuntu package manager
- Ciaran: fairly large gap between what the user writes and what the solver gets
- Mate, in several occasions, not trusting the user (or tuners)

Some more

- "Are you a proof theory person or a geek who develops solvers?" Karem Sakallah
  - I am a geek who develops modelling tools
- Don Knuth
  - After writing the book on sorting (including 25 algorithms)
  - Didn't know which algorithm to use when faced with a real sorting task
- Karem in his roadmap talk: "Can we recover the high-level structure from CNF?"
  - Well, don't destroy it in the first place?

This talk

- A lot more practical than most talks so far
- Mostly about what happens before solving
- My goal is to convince you that **there is value in capturing the problem without losing the high-level problem structure.**

- **Conjure** and its input language **Essence**
  - github.com/conjure-cp/conjure
  - conjure.readthedocs.io

This talk

- I can only fit so much in 45 minutes
- Out of necessity: a high-level tour of a particular approach
- Not a deep dive
- Several papers on different aspects
- Happy to talk more!

Constraint Programming and friends

- SAT – Boolean satisfiability
- SMT – SAT modulo theories
- CP – Constraint programming
- MIP – Mathematical programming
- ASP – Answer set programming
- Local search methods:
  - LNS – Large neighbourhood search
  - Heuristics like simulated annealing, etc

Observation

- None of these approaches dominate
- i.e. There are classes of problems where one method is better and other classes where another method is better
- Even for different instances of the same problem class

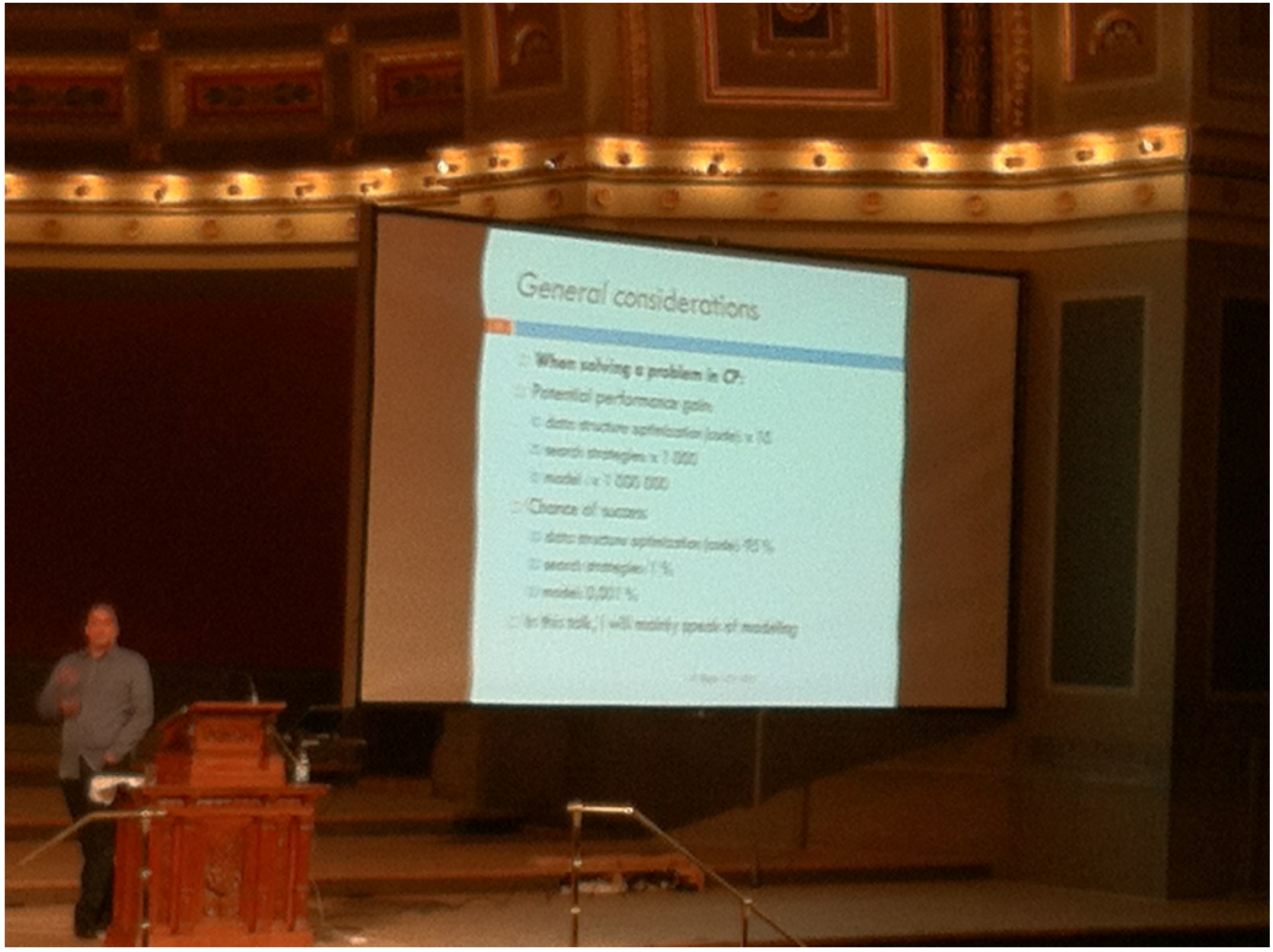
Levels of abstraction

- SAT: boolean decision variables, boolean constraints
- LP/IP/MIP: numeric decision variables, system of inequalities
- CP/SMT
  - typically decision variables with integer domains
  - richer set of constraints
- Essence
  - richer set of domains

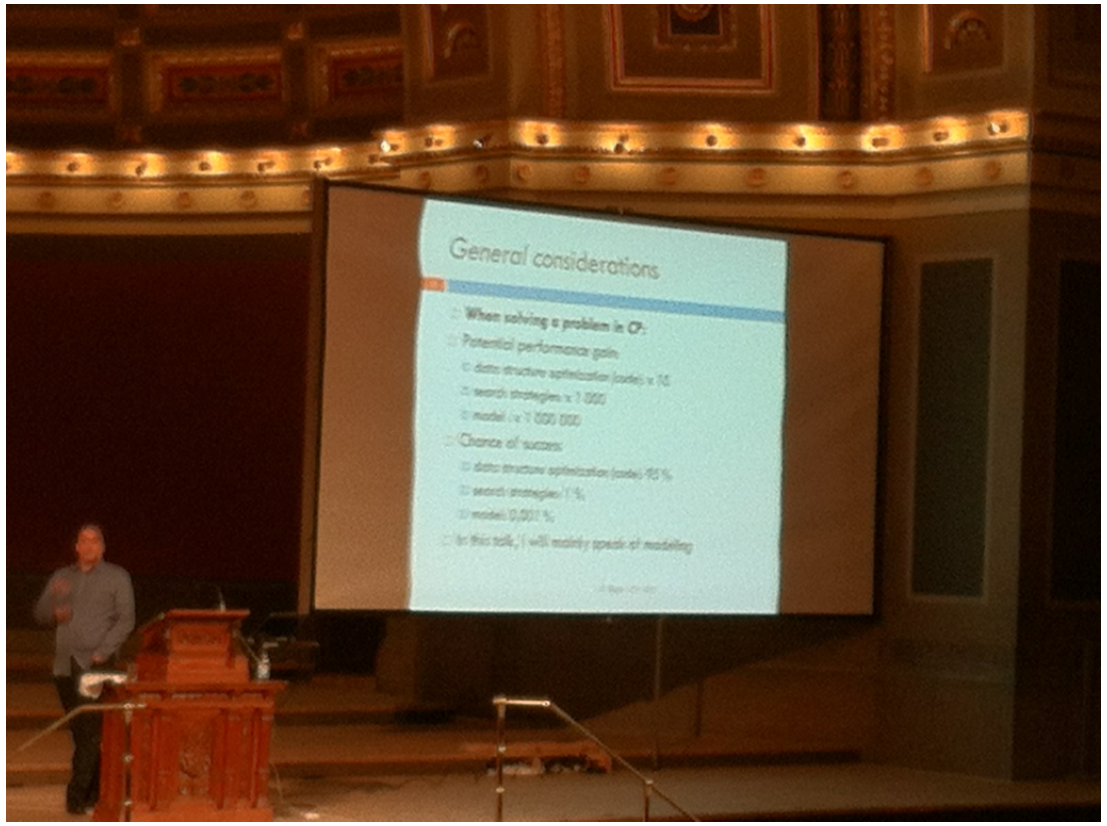
What is significant?

- Which methodology you use is significant
- Which solver you use is significant
- What solver settings you use is significant
- Which model you use is significant

- Our approach: expose options, then choose
  - Decouple the compilation and the decision making


General considerations
When solving a problem in C#:
Potential performance gain:
data structure optimization (code) x 10
search strategies x 1 000
model x 1 000 000
Chance of success:
data structure optimization (code) 95 %
search strategies 5 %
model 0,001 %
In this talk, I will mainly speak of modeling

- When solving a problem in CP
- Potential performance gain
  - Data structures (x10)
  - Search strategies (x1000)
  - Model (x1000000)
- Chance of success
  - Data structures (95%)
  - Search strategies (1%)
  - Model (0.001%)



Jean-Charles Régin

ACP Research Excellence Award

CP 2013

Significance of modelling

- Modelling decisions
  - Multiple ways to model a single problem
  - Huge impact on solving performance
  - *“Modelling is an art, rather than science.”*
  - Can we make it science?
- Holy Grail of computer science
  - the user states the problem
  - the computer solves it

Modelling decisions

Selecting the view point

- Which decision variables to use

Formulating each constraint

- Several ways of stating each constraint

Related

- Multiple view points & Channelling
- Symmetry breaking

Modelling decisions

Compilers are similar

- Multiple ways to compile a code fragment
- They mostly use heuristics

One key difference

- In CP, performance differences can be huge!

High-level modelling

Modelling to a solver

- Good old days

Solver independent modelling

- Essence Prime, Minizinc, ...

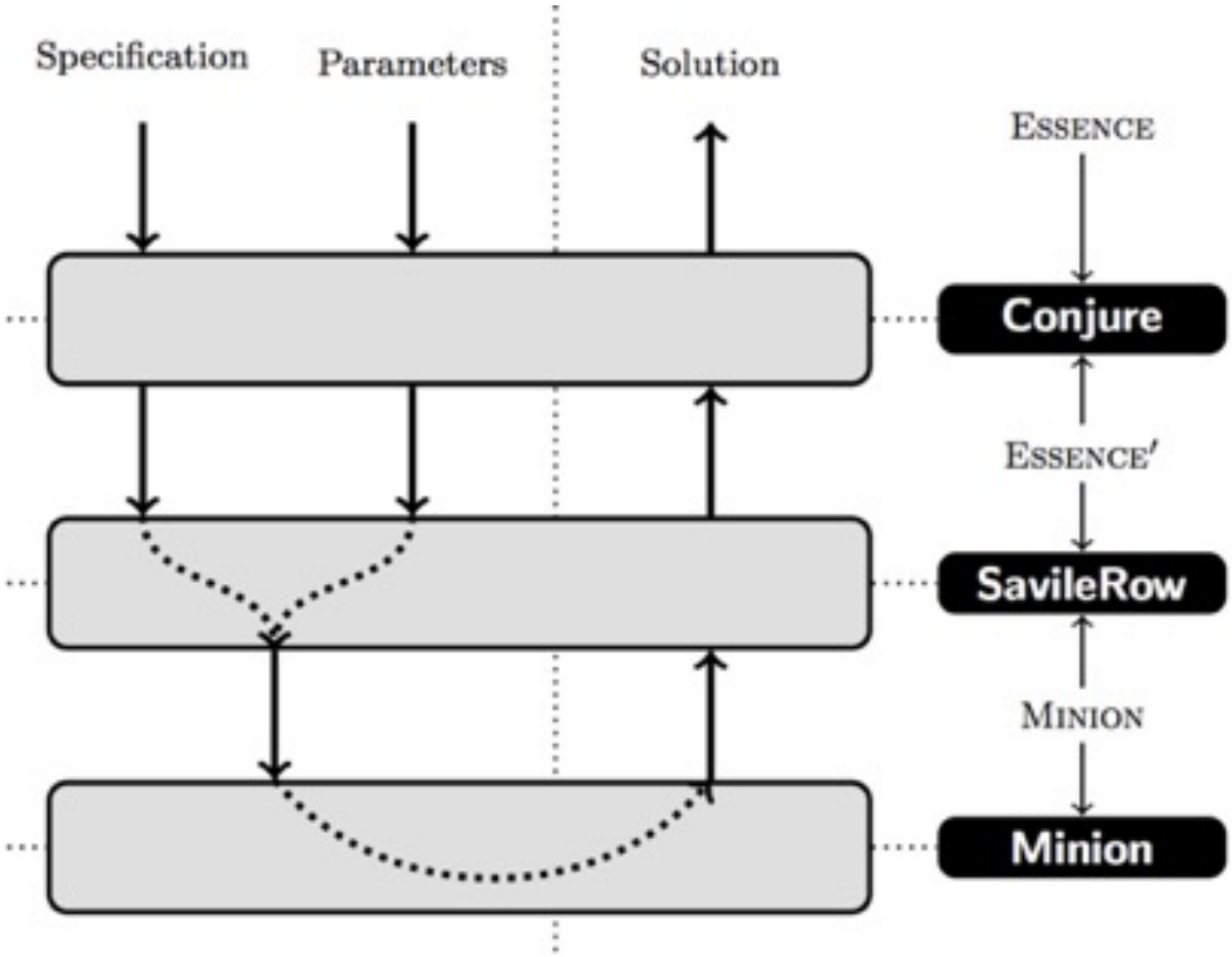
Model independent

- Problem specification
- Vast space of possibilities: Expose
- Progress towards exploiting

Problem class level

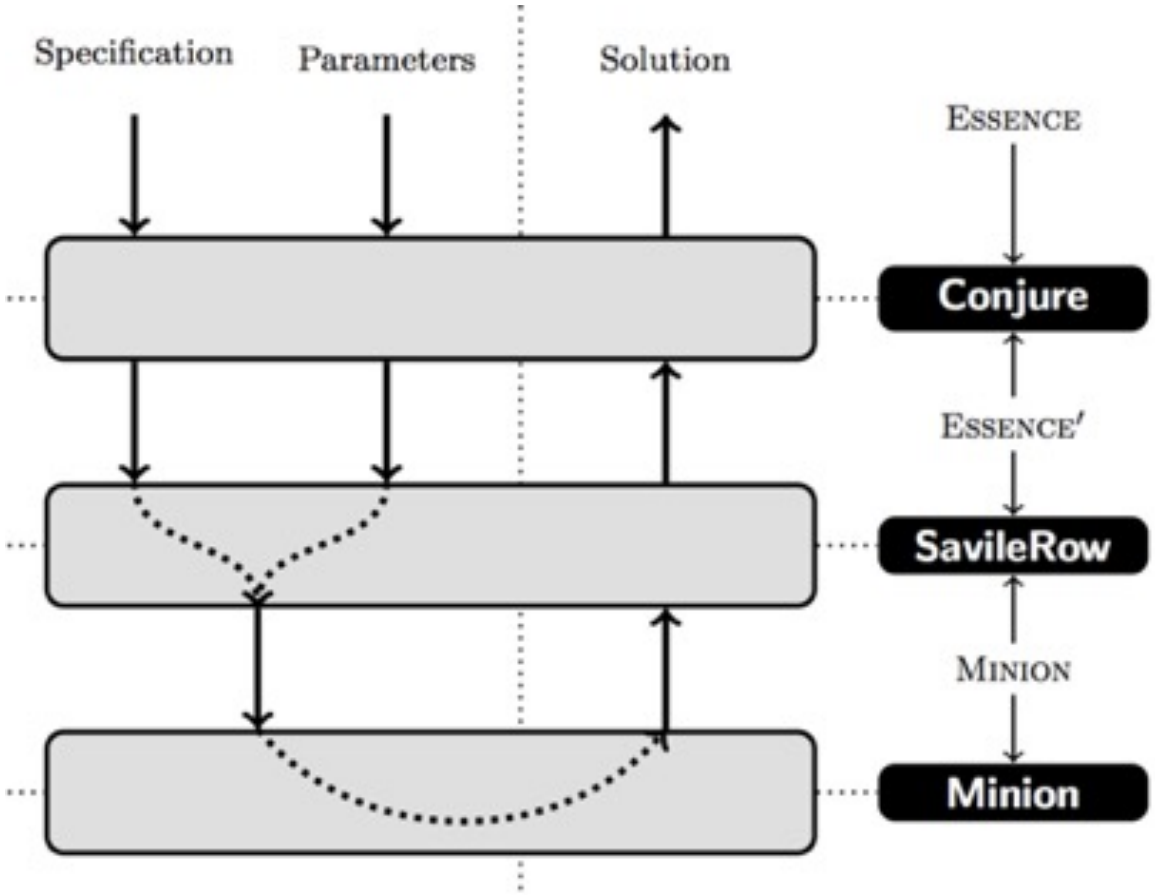
- Most modelling tools work on a single instance
  - Instantiate, reformulate, solve
- Conjure operates at the problem class levels
  - Reformulate, instantiate, solve
  - Parameterised input, parameterised output
- This can be a more challenging engineering task
- But all class level effort is amortised across instances

Conjure: Part of a pipeline



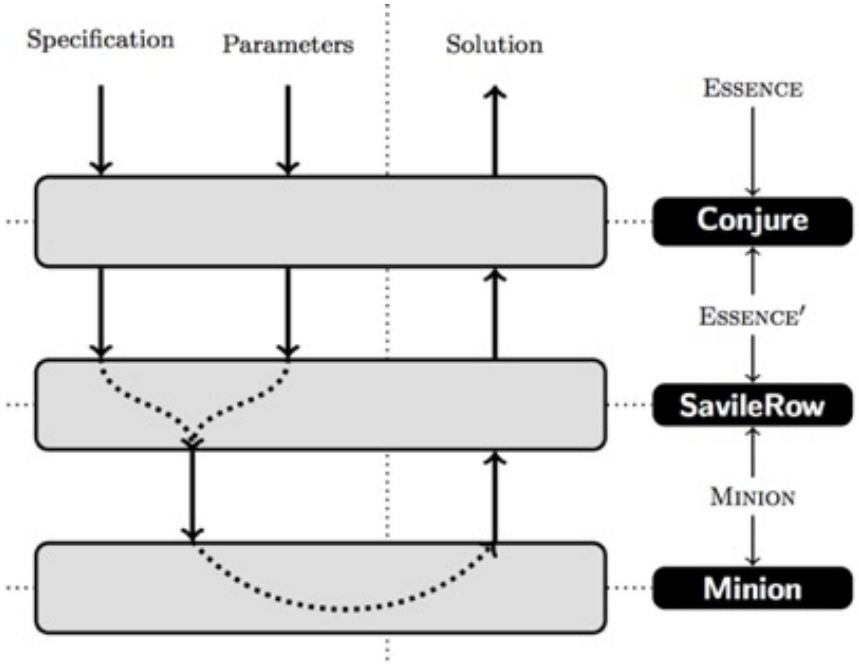
Languages

- Essence:
  - Abstract domains,
  - and operators.
- Essence Prime:
  - Concrete domains. Non-flat.
- Minion:
  - Concrete domains. Flat.
- Other backends:
  - CP solvers: Gecode, Chuffed
  - SAT: Glucose, Lingeling, Open-WBO
  - SMT, MIP, Local search
  - FlatZinc

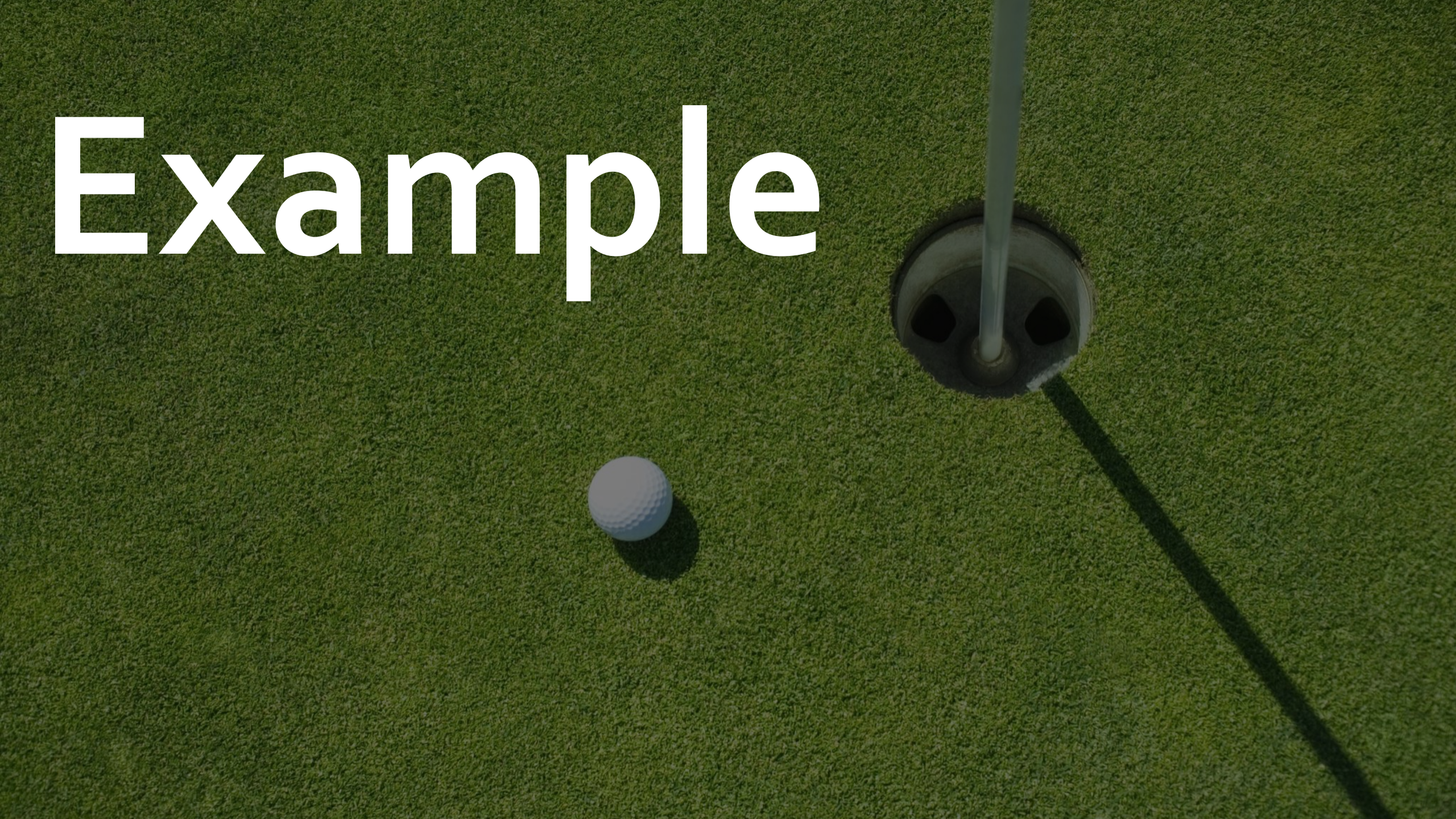


Tools

- Conjure: Class level
  - Main act: Type refinement
    - Structural & Symmetry breaking constraints
    - Channelling
  - Operator refinement
- Savile Row: Instance level
  - Instance level optimisations
  - Targets several solvers



Example



Social Golfers

- The coordinator of a local golf club has come to you with the following problem.
- In their club, there are 32 social golfers, each of whom play golf once a week, and always in groups of 4.
- They would like you to come up with a schedule of play for these golfers, to last as many weeks as possible, such that no golfer plays in the same group as any other golfer on more than one occasion.

Social Golfers

- The problem can easily be generalized to that of scheduling **g** groups of **s** golfers over **w** weeks, such that no golfer plays in the same group as any other golfer twice (i.e. maximum socialisation is achieved).

- https://www.csplib.org/Problems/prob010/

```
1 language Essence 1.3
```

```
language Essence 1.3
$ prob010.essence: Social Golfers Problem
$ Problem details available at http://www.csplib.org/Problems/prob010/
$
$ In a golf club there are a number of golfers who wish to play together in g
$ groups of size s. Find a schedule of play for w weeks such that no pair of
$ golfers play together more than once.
```

```
language Essence 1.3
$ prob010.essence: Social Golfers Problem
$ Problem details available at http://www.csplib.org/Problems/prob010/
$
$ In a golf club there are a number of golfers who wish to play together in g
$ groups of size s. Find a schedule of play for w weeks such that no pair of
$ golfers play together more than once.

given w, g, s : int(1..)
```

```
language Essence 1.3
$ prob010.essence: Social Golfers Problem
$ Problem details available at http://www.csplib.org/Problems/prob010/
$
$ In a golf club there are a number of golfers who wish to play together in g
$ groups of size s. Find a schedule of play for w weeks such that no pair of
$ golfers play together more than once.

given w, g, s : int(1..)

$ No need to identify individual golfers
$ Symmetry breaking
letting Golfers be new type of size g * s
```

```
language Essence 1.3
$ prob010.essence: Social Golfers Problem
$ Problem details available at http://www.csplib.org/Problems/prob010/
$
$ In a golf club there are a number of golfers who wish to play together in g
$ groups of size s. Find a schedule of play for w weeks such that no pair of
$ golfers play together more than once.

given w, g, s : int(1..)

$ No need to identify individual golfers
$ Symmetry breaking
letting Golfers be new type of size g * s

find sched : set (size w) of partition (numParts g, partSize s) from Golfers
```

```
language Essence 1.3
$ prob010.essence: Social Golfers Problem
$ Problem details available at http://www.csplib.org/Problems/prob010/
$
$ In a golf club there are a number of golfers who wish to play together in g
$ groups of size s. Find a schedule of play for w weeks such that no pair of
$ golfers play together more than once.

given w, g, s : int(1..)

$ No need to identify individual golfers
$ Symmetry breaking
letting Golfers be new type of size g * s

find sched : set (size w) of partition (numParts g, partSize s) from Golfers

$ maximum socialisation: g1 & g2 are together at most once
such that
  forAll g1, g2 : Golfers, g1 != g2 .
    (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```


Essence

Essence: A problem specification

- A problem specification is made up of a number of top level statements.
- A statement is one of:
  - A declaration: decision variables, problem parameters, aliases, ...
  - A constraint (such that)
  - An objective statement (maximising/minimising)
  - Conditions on valid instances (where)
  - Search directives (branching on)

Essence: Declaration statements

- Decision variables
  - find X : [DOMAIN]
- Problem parameters
  - given P : [DOMAIN]
- Aliases to expressions or domains
  - letting K be [EXPRESSION]
  - letting D be domain [DOMAIN]
- (We can also declare unnamed and enumerated types, but won't cover those just yet.)

Essence: Constraints, Objectives

- Constraints
  - such that BOOL-EXPRESSION
  - Typically involving decision variables
- Objectives
  - minimising/maximising INT-EXPRESSION
  - Typically involving decision variables
- Conditions on valid instances
  - where BOOL-EXPRESSION
  - In terms of the problem parameters

```
language Essence 1.3
$ prob026.essence: Round Robin Tournament Scheduling
$ Problem details available at http://www.csplib.org/Problems/prob026/

given n_teams : int(1..)

where n_teams % 2 = 0

letting Team   be new type of size n_teams,
        Week   be new type of size n_teams-1,
        Period be new type of size n_teams/2

find sched :
    relation (size ((n_teams-1)*n_teams)/2) of
        (Week * Period * set (size 2) of Team)

such that
    forAll t : Team . forAll w : Week . exists (p,ts) in toSet(sched(w,_,_)) . t in ts,
    forAll t : Team . forAll p : Period .
        (sum (w,ts) in toSet(sched(_,p,_)) . toInt(t in ts)) <= 2,
    forAll t1,t2 : Team , t1 != t2 . |toSet(sched(_,_,{t1,t2}))| = 1
```

```
language Essence 1.3

$ CSPLib 45: The Covering Array Problem

given k,b,t: int(1..), g: int(2..)
where k>=t, b>=g**t

letting alphabet be new type of size g
letting switches be new type of size k

find CoverTest: mset (size b) of function (total) switches --> alphabet

such that
    $ every test case must be covered by some test in CoverTest
    forAll testcase: function switches --> alphabet
        , |toSet(testcase)| = t
        . exists test in CoverTest . testcase subsetEq test
```

Essence: Domains

- Boolean, integer, matrix
- Enumerated, unnamed
- Tuple, record, variant
- Set, multi-set, sequence, function, relation, partition
- Graph (work in progress)

Domains: Set

- Set
  - Variable size, no duplicates, unordered
  - Cardinality attributes (minSize, maxSize, size)
- Representation options
  - Fixed cardinality explicit
  - Occurrence
  - Variable cardinality explicit (with markers)

```
language Essence 1.3
$ prob013.essence: Progressive Party Problem
$ Problem details available at http://www.csplib.org/Problems/prob013/

given n_boats, n_periods : int(1..)

letting Boat be domain int(1..n_boats)

given capacity, crew : function (total) Boat --> int(1..)

find hosts : set of Boat,
     sched : set (size n_periods) of function (total) Boat --> Boat

minimising |hosts|
```

Domains: Multi-set

- Multi-set
  - Variable size, may contain duplicates, unordered
  - Cardinality attributes (minSize, maxSize, size)
  - Occurrence attributes (minOccur, maxOccur)
- Representation options
  - Explicit with repetition
  - Explicit with counts
  - Occurrence

```
language ESSENCE 1.2.0
$ prob116.essence: Vellino's Problem
$ Problem details available in:
$   The OPL Optimization Programming Language
$   Pascal Van Hentenryck
$   MIT Press, January 1999.
$
$ 27 July 2007

...

$ colour: each bin is assigned a colour
$ contents: the contents of each bin is a multiset of materials
find colour : function Bin --> Colour,
     $ The size of each mset returned by contents is <= some value returned by capacity (given)
     contents : function Bin --> mset (maxOccur maxMaterial, maxSize max(range(capacity))) of Material

$ minimise the number of bins that have colours & materials assigned to them
minimising |defined(colour)|

...

```

Domains: Function

- Function
  - Between two domains
  - May be partial
  - Cardinality attributes
  - Function attributes (injective, surjective, bijective)
- Representation options
  - For total functions: matrices
  - For partial functions: matrices with boolean markers to indicate function definedness
  - Via-relation for sparse partial functions

```
language Essence 1.3

$ The Ramsey number R(k,l) is the smallest number such that
$ if we two-colour the edges of complete graph of this size,
$ there always exists a monochromatic subgraph of either k or l nodes.

...

$ we are two-colouring
letting Colour be new type enum {red, blue}

letting Vertex be domain int(1..num_vertices)

find graph : function (size num_edges) (Vertex, Vertex) --> Colour

...
```

Domains: Sequence

- Sequence
  - Variable size, may contain duplicates, ordered
  - Cardinality attributes (minSize, maxSize, size)
  - Function-like attributes (injective, surjective, bijective)
- Representations
  - A finite array with a size marker

```
language Essence 1.3

$ You are given an 8 pint bucket of water, and
$ two empty buckets which can contain 5 and 3 pints respectively.
$ You are required to divide the water into two by pouring water between buckets
$ (that is, to end up with 4 pints in the 8 pint bucket, and
$                           4 pints in the 5 pint bucket).

...

given HORIZON : int

find actions : sequence (maxSize HORIZON) of (Buckets, Buckets, int(1..maxCapacity))
find states  : sequence (maxSize HORIZON) of function (total) Buckets --> int(0..buckets)

...
```

Domains: Relation

- Relation
  - On an arbitrary number of domains
  - Cardinality constraints
  - A number of binary relations attributes (reflexive, symmetric, Euclidean, ...)
- Representations
  - An n-dimensional boolean matrix
  - A set-of-tuples representations (good for sparse relations)

```
language Essence 1.3
$ prob028.essence: Balanced Incomplete Block Design
$ Problem details available at http://www.csplib.org/Problems/prob028/

given v, b, r, k, lambda : int(1..)

letting Obj   be new type of size v,
        Block be new type of size b

find bibd : relation of (Obj * Block)

such that
    forAll o  : Obj   . |toSet(bibd(o,_ ))| = r,
    forAll bl : Block . |toSet(bibd(_,bl))| = k,
    forAll o1, o2 : Obj
        , o2 > o1
        . |toSet(bibd(o1,_)) intersect toSet(bibd(o2,_))| = lambda
```

Domains: Partition

- Partition
  - On any domain
  - Attributes on size and number of parts + regularity
- Representations
  - Via set-of-sets: which in turn means a bunch of different representations
  - A direct partition occurrence representation

```
language ESSENCE 1.2.0
$ prob022.essence: Bus Driver Scheduling
$ Problem details available at http://www.csplib.org/Problems/prob022/
$ 05 September 2007

given Tasks new type enum,
      shifts : set of set of Tasks

find sched : partition from Tasks

minimising |parts(sched)|

such that forAll s in parts(sched) . s in shifts

```

Concrete domains

- bool, int, matrix (very much like a multi-dimensional array)
- Matrices are sometimes natural for a problem
- But often they are a “code smell” – try to use abstract domains where possible!

- Also available in Essence Prime

```
language ESSENCE 1.2.0
$ prob019.essence: Magic Squares
$ Problem details available at http://www.csplib.org/Problems/prob019/
$ 05 September 2007

given n : int(1..)

letting Index be domain int(1..n),
        Value be domain int(1..n**2)

find square : matrix indexed by [Index,Index] of Value,
     s : int(1..sum i : int(n**2+1-n..n**2) . i)

such that
    allDiff(flatten(square)),
    forAll r : Index . (sum c : Index . square[r,c]) = s,
    forAll c : Index . (sum r : Index . square[r,c]) = s,
    (sum d : Index . square[d,d]) = s,
    (sum d : Index . square[d,n+1-d]) = s
```

Domains: Tuple/Record

- Tuples
  - find x : (DOMAIN_1, DOMAIN_2, ...)
  - Arbitrary arity, accessed by square brackets
  - Indexing starting at 1
- Records
  - Similar to tuples, but fields are named
  - find x : record { foo : DOMAIN_1, bar : DOMAIN_2 }
  - Access by square brackets
  - x[foo] + x[bar] = ...

Domains: Variants

- Similar syntax to records
- But only one *active* at a time
- Called tagged unions in some programming languages
- Syntax
  - find x : variant { foo : DOMAIN_1, bar : DOMAIN_2 }
  - x[foo]
  - active(x, foo) – returns a Boolean

Domains: enumerated & unnamed types

- Enumerated
  - letting Colours be new type enum {Red, Yellow, Green}
  - given modes new type enum
- Unnamed
  - Avoids naming values of a domain
  - Aids type-directed symmetry breaking

```
language Essence 1.3
$ prob010.essence: Social Golfer Problem
$ Problem details available at http://www.csplib.org/Problems/prob010/
$
$ In a golf club there are a number of golfers who wish to play together in g
$ groups of size s. Find a schedule of play for w weeks such that no pair of
$ golfers play together more than once.

given w, g, s : int(1..)

letting Golfers be new type of size g * s

find sched : set (size w) of
                $ regular is implied by numParts g and partSize s
                partition (regular, numParts g, partSize s) from Golfers

such that
forAll g1, g2 : Golfers, g1 != g2 .
    (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```


Opportunities

Selected highlights

- Symmetry breaking
- Model strengthening
- Savile Row optimisations
- Model selection (heuristics, racing, tuning)
- Streamlining
- Local search (SNS, Athanor)
- Instance generation (for training, evaluation)
- Constraint Dominance Programming

Symmetry breaking

- Automated symmetry breaking
- 2Conjure does not introduce symmetry
  - i.e. breaks all symmetry it introduces

- Automated Symmetry Breaking and Model Selection in Conjure (CP 2013)
- Breaking Conditional Symmetry in Automated Constraint Modelling with Conjure (ECAI 2014)

- Great when enumerating combinatorial objects

Streamlining constraints

- Adding uninferred constraints
- Directed by the domains
  - Half of the functions in this set of functions are non-decreasing
  - A sequence is surjective on the even subset of values
  - Etc...
- Selection: Monte Carlo Search on a Model Lattice
- "Automated streamliner portfolios for constraint satisfaction problems." Artificial Intelligence (2023)

Constraint Dominance Programming

- Adding constraints among solutions
- If X is a solution, Y shouldn't be
- Very expressive
- Applications in multi-objective optimisation and data mining

- Exploiting incomparability in solution dominance: Improving general purpose constraint-based mining (ECAI 2020)

Structure guided local search

- Generate declarative neighbourhoods
- Again, directed by the domains
  - For a partition, move an item between parts
  - Instead of removing the item and not putting it back
- A Framework for Constraint Based Local Search using Essence (IJCAI 18)
- Athanor: high-level local search over abstract constraint specifications in Essence (IJCAI 19)

Channelled models

- There are typically several ways of representing an abstract decision variable
- Every occurrence of an abstract decision variable is an opportunity for a new representation
- Conjure automatically produces channelling constraints
- May help with propagation
- May act as a good search order
  - Modelling Langford's Problem: A Viewpoint for Search (ModRef 18)

Multiple models

- Conjure can generate a large number of models (1000s in some cases)
- What to do with these?
- Model selection, model portfolios, racing
- Heuristics
  - There are a few built in
  - “Compact” seems to work reasonably well

Savile Row optimisations

- Instance level, for a particular solver
- Common subexpression elimination
  - A way of connecting constraints
- Automatic tabulation
  - Explicit representation of allowed tuples
  - For subsets of constraints
  - Another way of connecting constraints
  - Choosing when to do this: to be presented at IJCAI 2023

Model & solver selection

- Applying algorithm tuning and algorithm selection methods
- They work reasonably well within a problem class
- They can help explain **why** something works

- Caveat:
  - I share some of Mate's Skepticism about the existing tools

Generating benchmark instances

- Can automatically generate benchmark instances that differentiate between options (model or solver)
- Works well for approach A, but not for approach B
- "Automated cherry-picking"
- Useful for identifying shortcomings of particular options

- Instance generation via generator instances (CP 2019)
- Discriminating instance generation from abstract specifications: A case study with CP and MIP (CPAIOR 2020)

Summary

- I believe **capturing the problem without losing the high-level problem structure is very valuable.**
  - Allows efficient translation to multiple models and solvers
  - Expose options, choose depending on data
  - "Conjure: Automatic generation of constraint models from problem specifications" (Artificial Intelligence 2022)
- Next steps
  - Better translations & better choosing
  - Better understanding of what works and why
  - So we can focus on what needs improving

Some advertisement

- Get Conjure
  - github.com/conjure-cp/conjure
  - conjure.readthedocs.io

- Savile Row savilerow.cs.st-andrews.ac.uk
- Essence Catalog github.com/conjure-cp/EssenceCatalog
- CSPLib CSPLib.org

University of St Andrews

Abstract Modelling with Conjure

Dr Özgür Akgün

School of Computer Science, University of St Andrews

ozgur.akgun@st-andrews.ac.uk

Berkeley, 19 April 2023