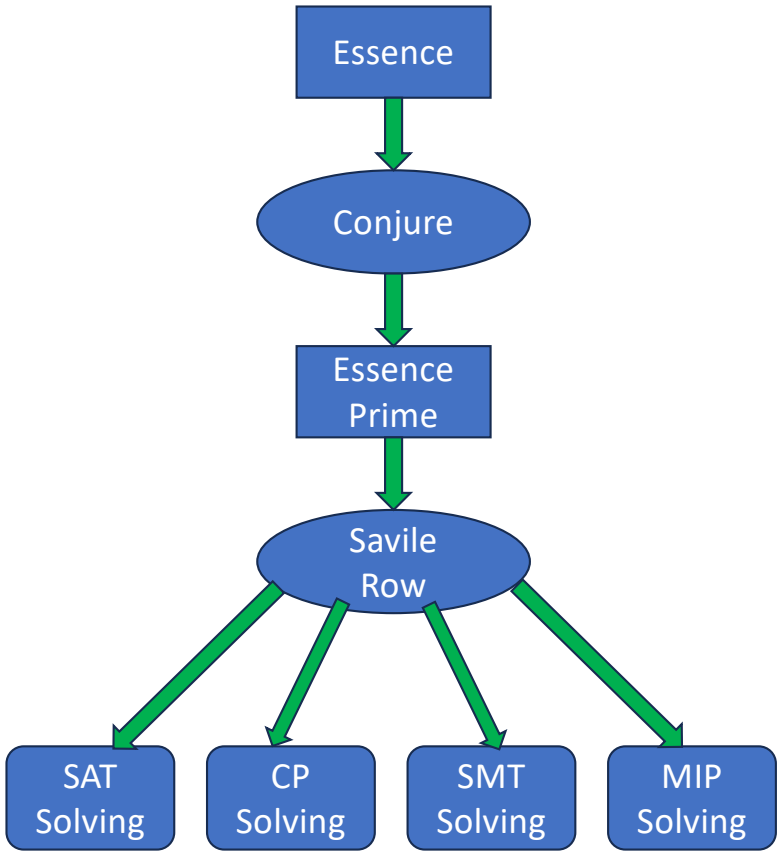
A Constraint Modelling Pipeline

Ian Miguel, Ozgur Akgun

University of St Andrews

ijm@st-andrews.ac.uk ozgur.akgun@st-andrews.ac.uk

Overview: The Pipeline



- **Essence**: an abstract constraint specification language.
- Essence refined by **Conjure** into a solver-independent constraint model in **Essence Prime**.
- Tailored for a particular solving paradigm/solver by **Savile Row**.

- **Automatically improving constraint models in Savile Row**. P Nightingale, Ö Akgün, IP Gent, C Jefferson, I Miguel, P Spracklen. Artificial Intelligence 251, 35-61, 2017.
- **Conjure: Automatic generation of constraint models from problem specifications**. Ö Akgün, AM Frisch, IP Gent, C Jefferson, I Miguel, P Nightingale. Artificial Intelligence 310, 103751, 2022.
- **Essence: A Constraint Language for Specifying Combinatorial Problems.** AM Frisch, W Harvey, C Jefferson, B Martinez-Hernandez, I Miguel. Constraints 13, 268-306, 2008.

Pre-history

Pre-history: Implied Constraints



- 1999: Alan Frisch and Toby Walsh are awarded UK EPSRC grant: Automatic Generation of Implied Constraints (GR/N16129/01) at York.
- Some early work in our thinking about **implied constraints**:
  - **Constraint generation via automated theory formation.** S. Colton, I. Miguel. 7th CP, 575-579, 2001.
  - **Extensions to proof planning for generating implied constraints.** A.M. Frisch, I. Miguel, T. Walsh. Calculemus, 130-141, 2001.
  - **CGRASS: A system for transforming constraint satisfaction problems.** A.M. Frisch, I. Miguel, T. Walsh. ERCIM Workshop on Constraint Solving and Constraint Logic Programming, 15-30, 2002.

Pre-history: Modelling Patterns







- Also started thinking about **modelling patterns**:
  - **Matrix modelling.** P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh. Workshop on Modelling and Problem Formulation, 2001.
  - **Symmetry in matrix models.** P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, T. Walsh. SymCon Workshop, 2001.
  - **Breaking row and column symmetries in matrix models.** P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, T. Walsh. CP, 546-477, 2002.
  - **Symmetry breaking as a prelude to implied constraints: A constraint modelling pattern.** A. M. Frisch, C. Jefferson, I. Miguel. ECAI, 2004.

E.g. 2d 0/1 matrix representing a relation in solving Balanced Incomplete Block Designs

0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	1

Pre-history: Refinement



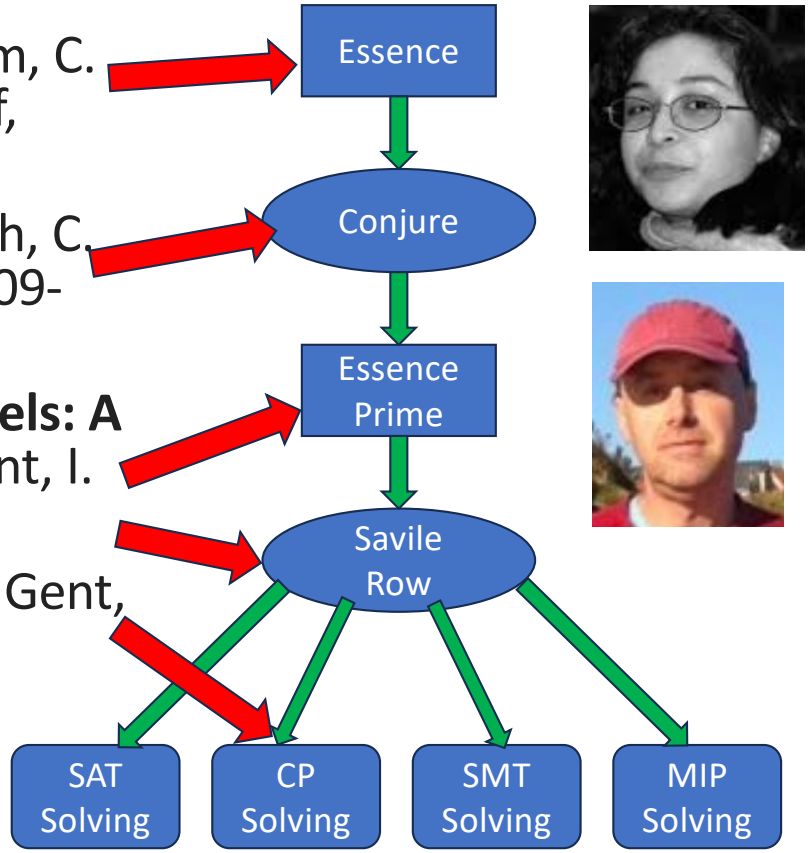


- Work on abstract modelling and refinement:
  - **Towards CSP model reformulation at multiple levels of abstraction**. A. M. Frisch, B. Hnich, I. Miguel, B. M. Smith, T. Walsh. ModRef Workshop, 2002.
  - **Towards automatic modelling of constraint satisfaction problems: A system based on compositional refinement.** A. Bakewell, A. M. Frisch, I. Miguel. ModRef Workshop, 2003.
  - **Function variables for constraint programming.** B. Hnich. *AI Communications*, *16*(2), 131-132, 2003.
  - **Introducing ESRA, a relational language for modelling combinatorial problems.** P. Flener, J. Pearson, M. Ågren. LOPSTR, 214-232, 2003.

Pre-history: First Elements of the Pipeline.

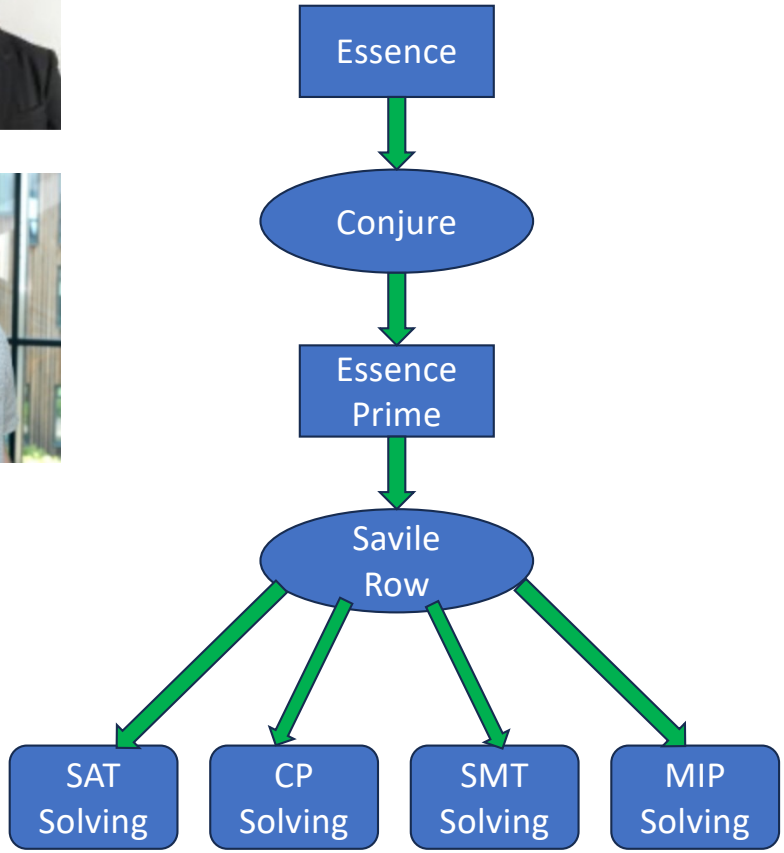


- **The Essence of Essence.** A. M. Frisch, M. Grum, C. Jefferson, B. M. Hernández, I. Miguel. ModRef, 2005.
- **The rules of constraint modelling.** A. M. Frisch, C. Jefferson, B. M. Hernández, I. Miguel. IJCAI, 109-116, 2005.
- **Tailoring solver-independent constraint models: A case study with Essence' and Minion.** I. P. Gent, I. Miguel, A. Rendl. SARA, 18-21, 2007.
- **Minion: A fast scalable constraint solver.** I. P. Gent, C. Jefferson, I. Miguel. ECAI, 98-102, 2006.



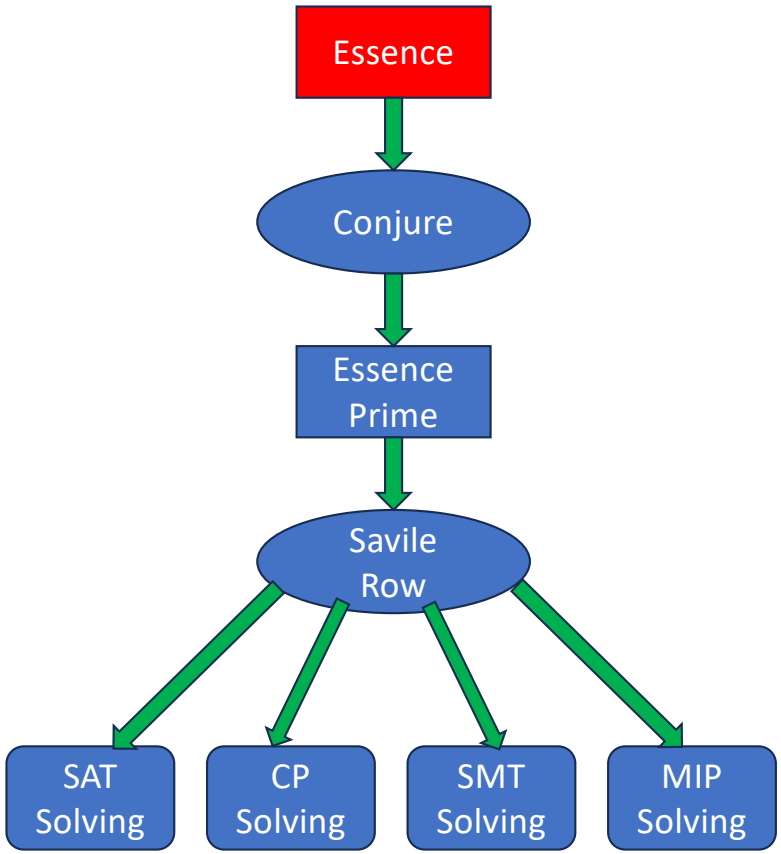
Today

- Enter the current primary developers:
  - Ozgur Akgun (Conjure)
  - Peter Nightingale (Savile Row).



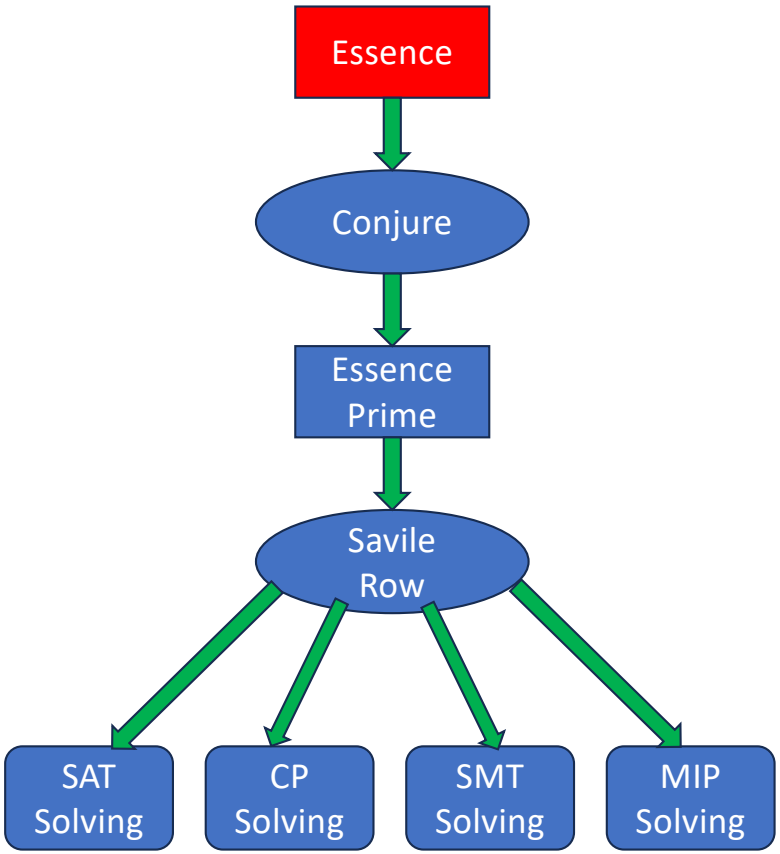
The Pipeline: Essence

Constraint Modelling Pipeline: Essence



- An abstract constraint specification language.
- Domain constructors, such as set, function, sequence, partition, relation, …
  - Arbitrary nesting of these: set of sets, sequence of functions, …
- Attributes of these domains:
  - Injective function, symmetric relation.
- Constraints/Operators on these domains:
  - Projection on relations.
  - Range of function.

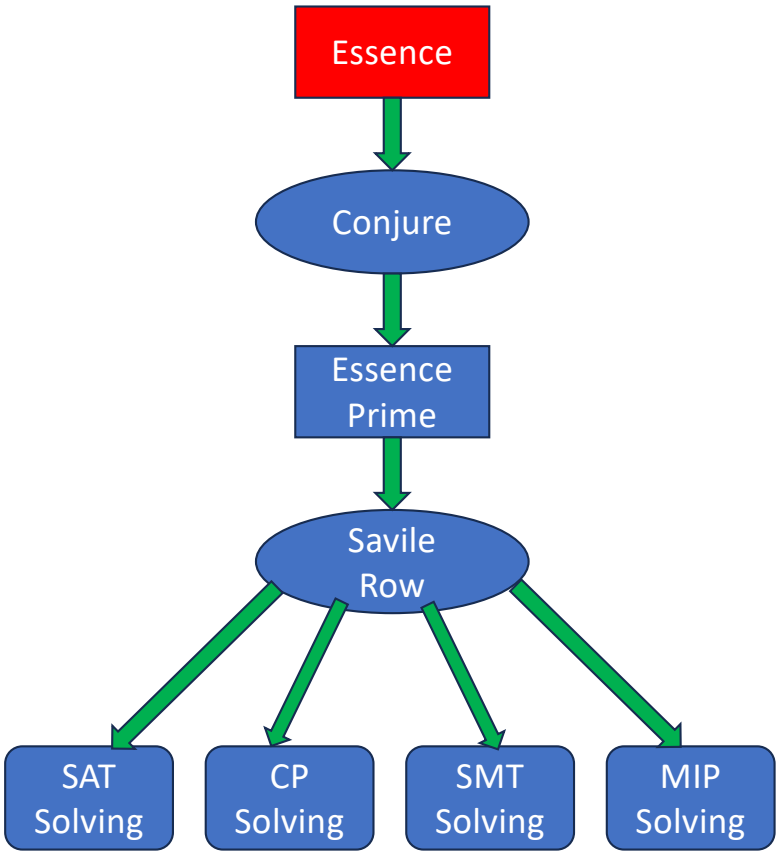
Constraint Modelling Pipeline: Essence



- Example: Social Golfers Problem.
- 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

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5                 partition (regular, numParts g, partSize s)
6                 from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Constraint Modelling Pipeline: Essence

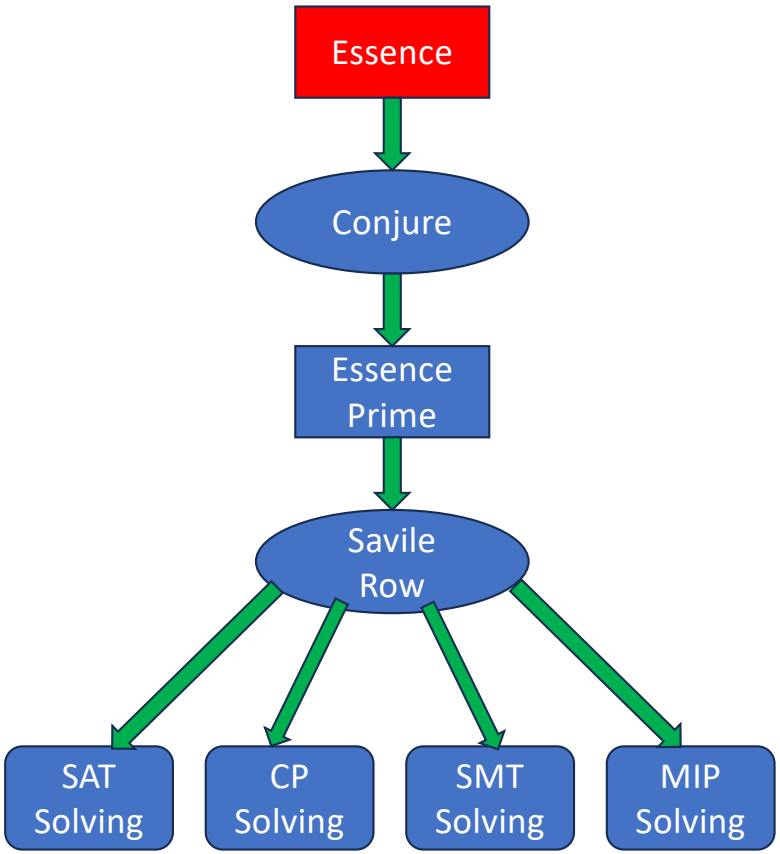


- Example: Social Golfers Problem.
- In a golf club there are a number of golfers who wish to play together in **g** groups of size **s**. Find a schedule of play for **w** weeks such that no pair of golfers play together more than once

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5                 partition (regular, numParts g, partSize s)
6                 from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Integer parameters

Constraint Modelling Pipeline: Essence

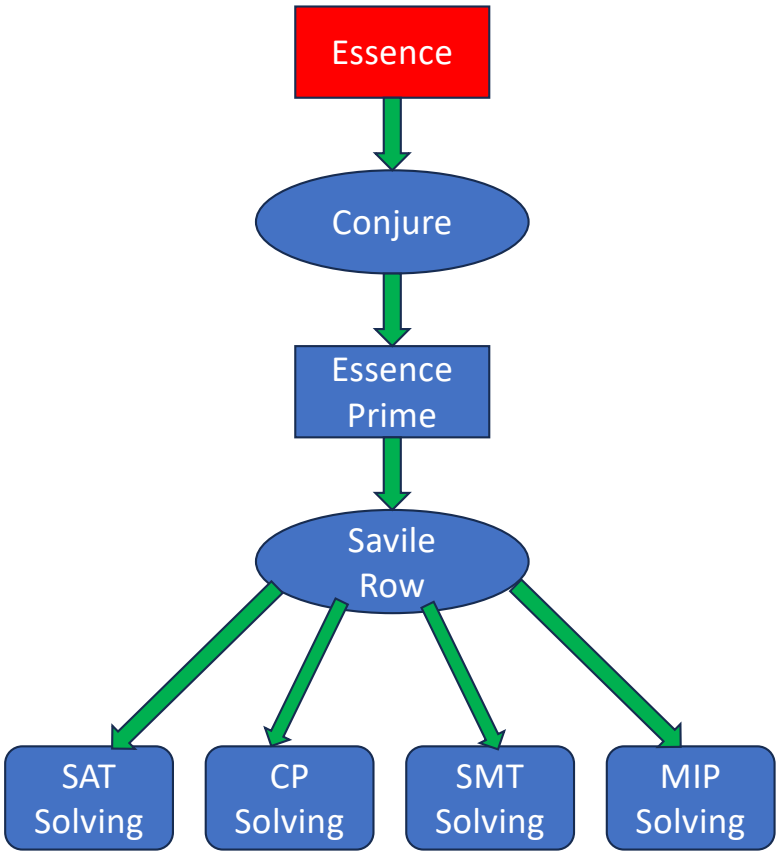


- Example: Social Golfers Problem.
- In a golf club there are a number of golfers who wish to play together in **g** groups of size **s**. Find a schedule of play for **w** weeks such that no pair of golfers play together more than once

Individual golfers don't need to be identified.
Symmetry avoided.

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5                 partition (regular, numParts g, partSize s)
6                 from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Constraint Modelling Pipeline: Essence

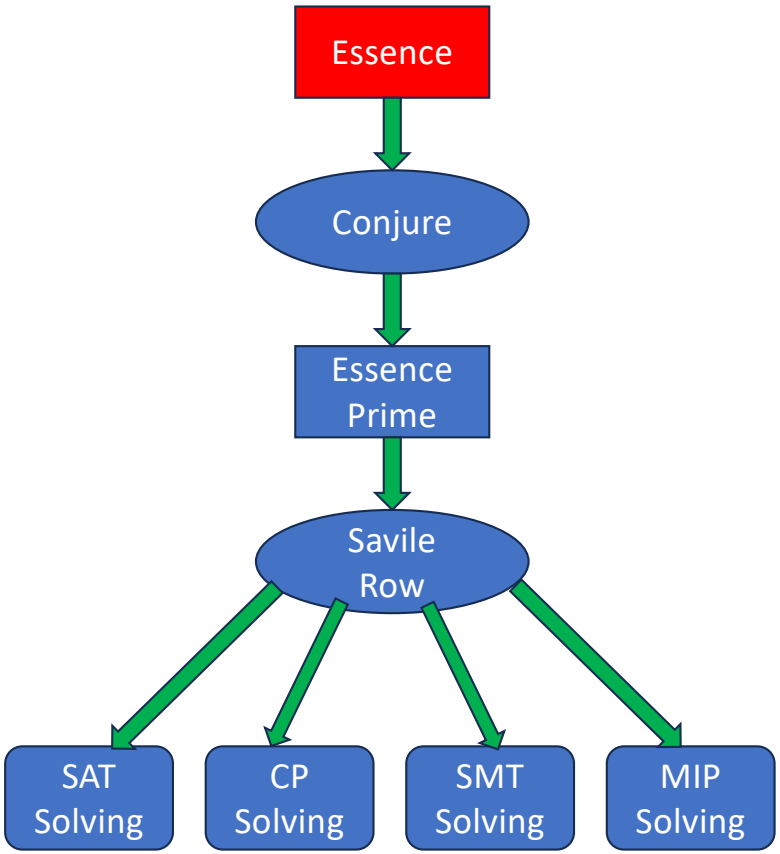


- Example: Social Golfers Problem.
- In a golf club there are a number of golfers who wish to play together in **g** groups of size **s**. Find a schedule of play for **w** weeks such that no pair of golfers play together more than once

One highly-structured decision variable.

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5               partition (regular, numParts g, partSize s)
6               from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Constraint Modelling Pipeline: Essence



- Example: Social Golfers Problem.
- 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

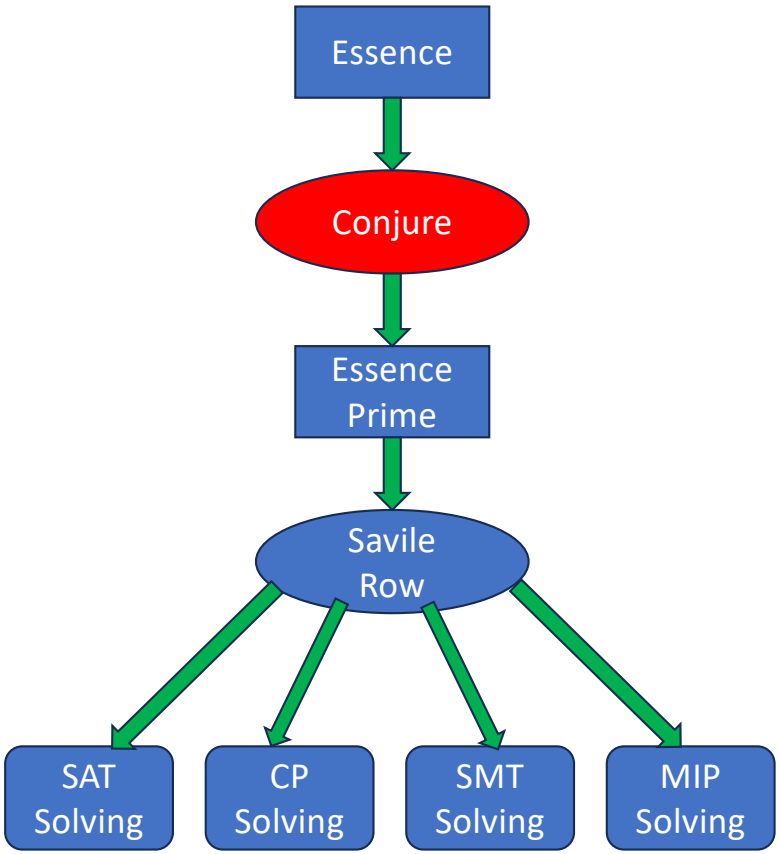
The **socialisation** constraint

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5                 partition (regular, numParts g, partSize s)
6                 from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

NB Having described the combinatorial structure to be found using Essence's types this is the only constraint left to be stated.

The Pipeline: Conjure

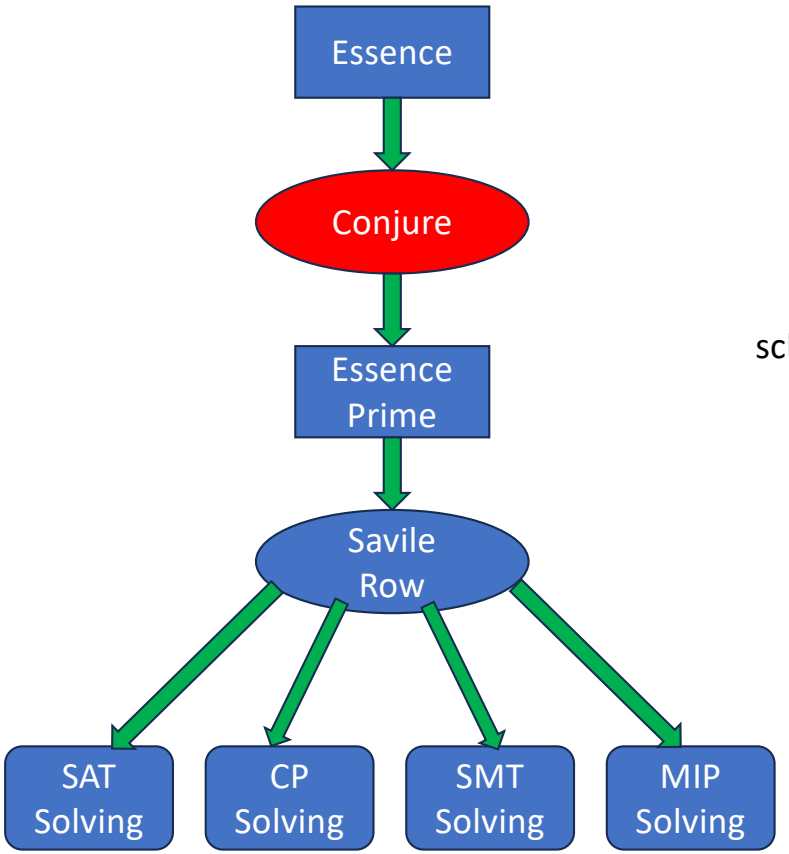
Constraint Modelling Pipeline: Conjure



- We can't typically solve an Essence specification directly.
- We use the **Conjure** system to **refine** an Essence specification into **Essence Prime**.
  - A subset of Essence with facilities common in constraint modelling languages.
  - (Matrices of) Integer, Boolean variables.
  - Logical, Arithmetic, Global Constraints.

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5                 partition (regular, numParts g, partSize s)
6                 from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Constraint Modelling Pipeline: Conjure



- Refinement proceeds from the choice of representation of the decision variables.
- The outer structure of sched here is a **fixed-cardinality set**.
- A natural model is via a matrix:

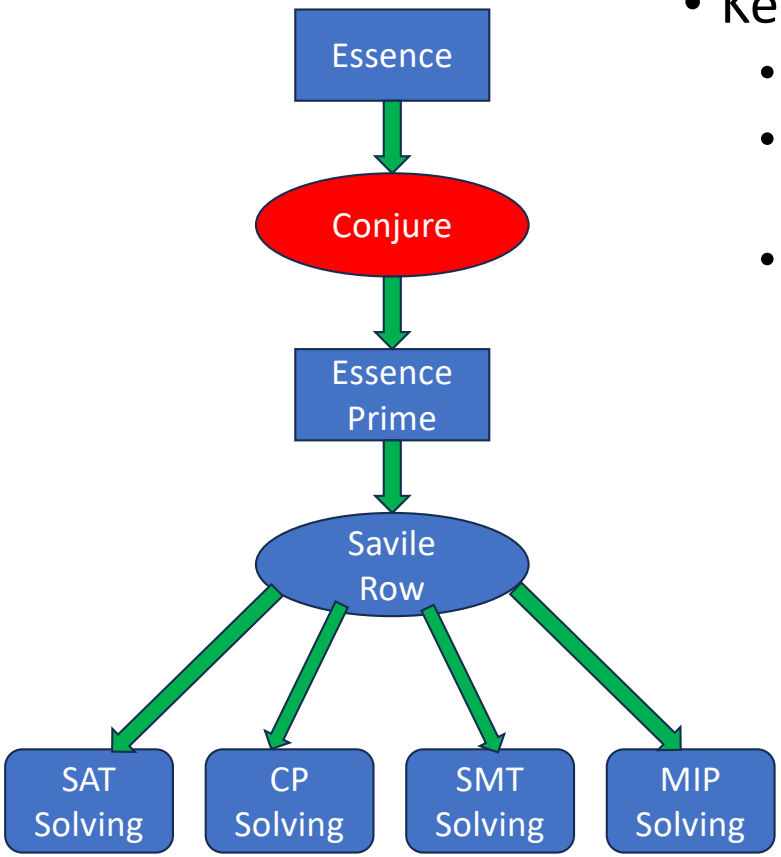
sched

1	2	…	w-1	w
<partition>	<partition>	…	<partition>	<partition>

Structural constraint: AllDifferent(sched)

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5               partition (regular, numParts g, partSize s)
6               from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Constraint Modelling Pipeline: Conjure



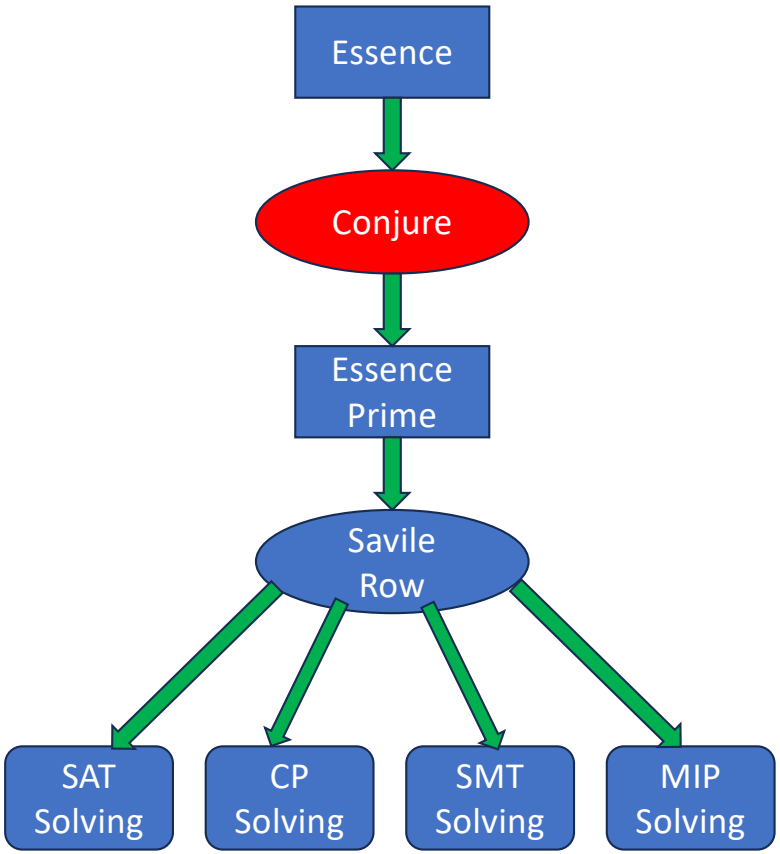
- Key advantage of refinement-based approach:
  - Recognise and break **symmetry** as it enters the model.
  - By refining a set to an indexed matrix we introduce symmetry: permuting the weeks is solution-preserving.
  - Conjure knows this and adds constraints to break this symmetry:

1	<	2	<	...	<	w-1	<	w
<partition>		<partition>		...		<partition>		<partition>

(and the AllDifferent is automatically removed)

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5               partition (regular, numParts g, partSize s)
6               from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Constraint Modelling Pipeline: Conjure



- We can think of a partition as a constrained set of sets:

Parts (i.e. groups)

Domain: {1, …, g x s}
Where g x s is the number of golfers

A week	1	2	..	g
1		<	<	<
2	<			
..	<			
s	<			

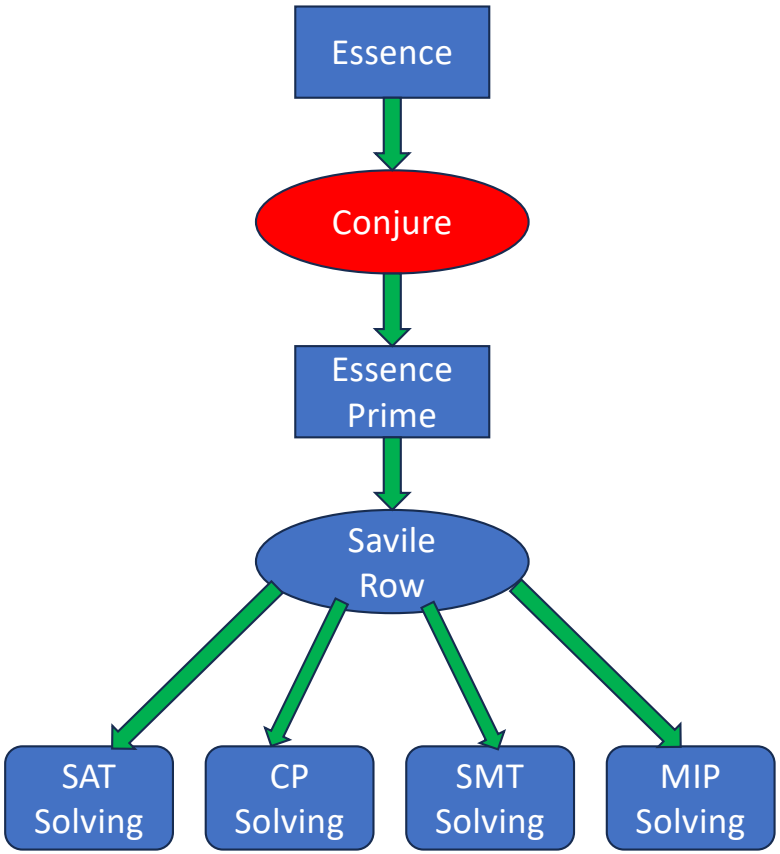
Elements of each part (i.e. golfers)

Again, breaking **symmetry**

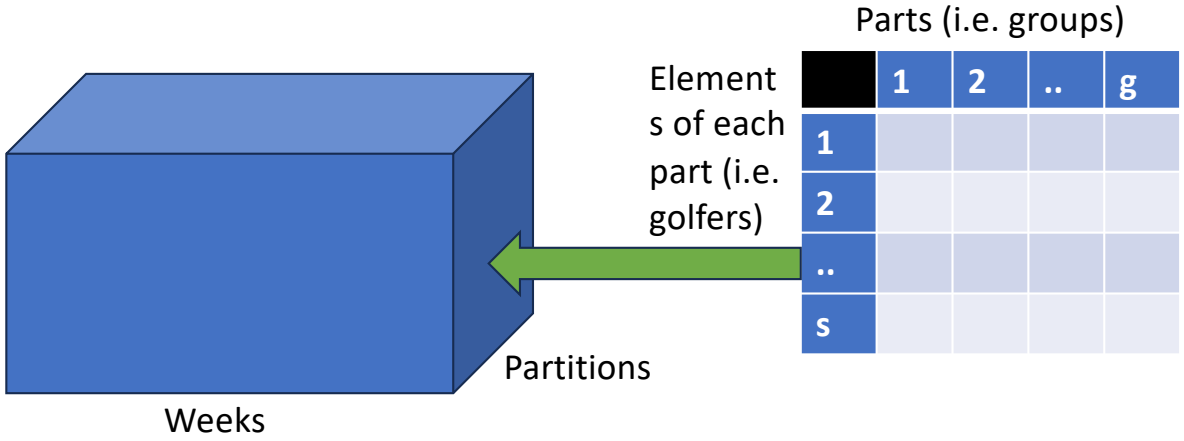
Structural constraint: AllDifferent

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5               partition (regular, numParts g, partSize s)
6               from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Constraint Modelling Pipeline: Conjure

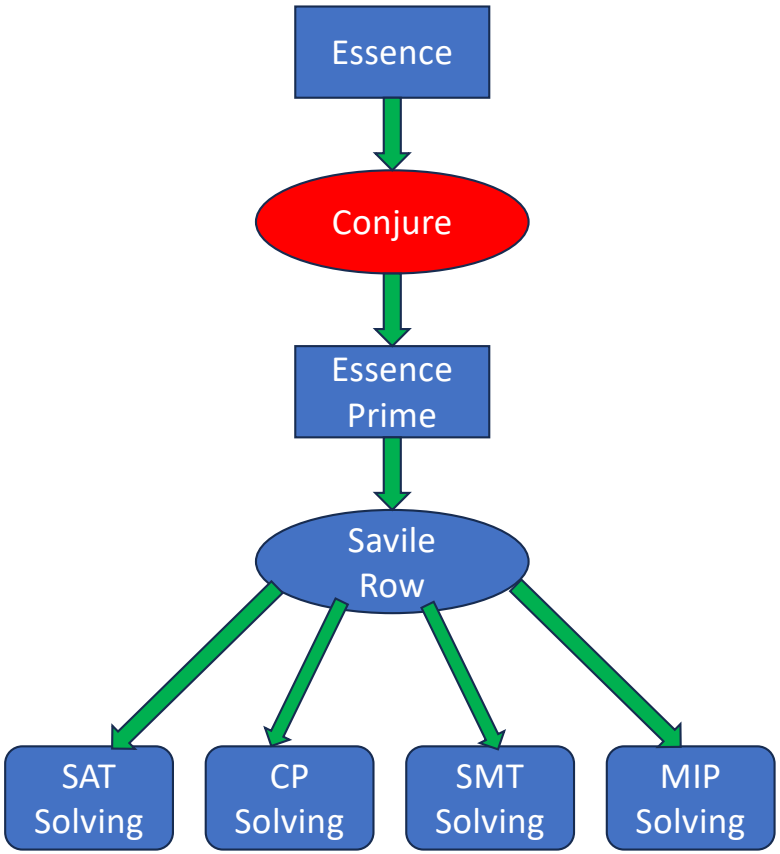


- Giving (a) representation of sched:




```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5               partition (regular, numParts g, partSize s)
6               from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Constraint Modelling Pipeline: Conjure



- Conjure then refines the constraints to suit the representation chosen:

Parts (i.e. groups)

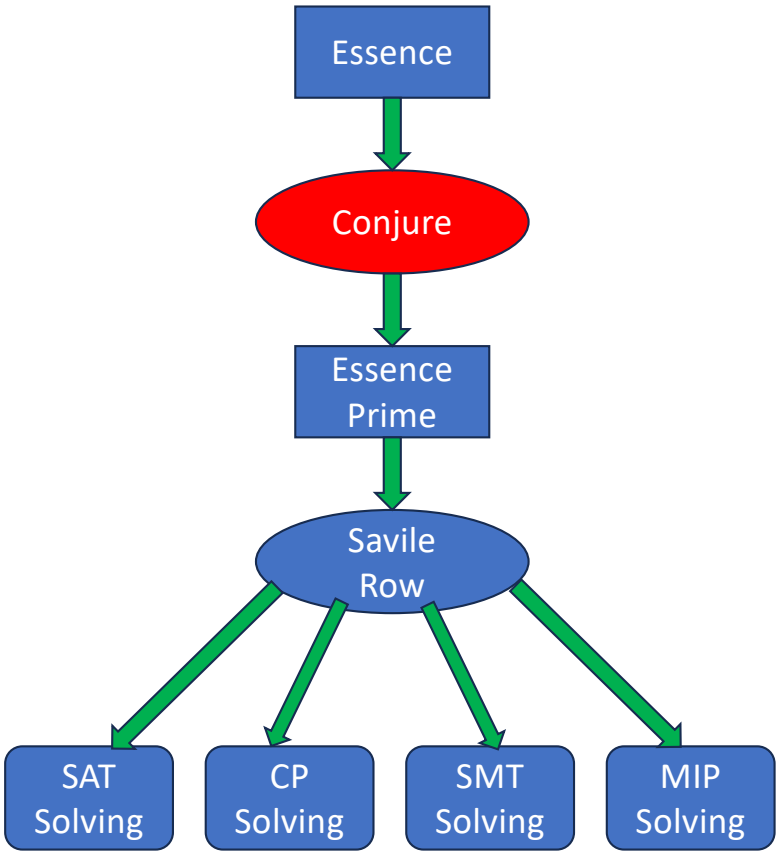
Elements of each part (i.e. golfers)

	1	2	..	g
1	4			
2	5			
..				
s				

- **Disallow 4, 5 in the same group in any other week**
- **How:**
  - **Represent the intersection between parts in different weeks.**
  - **Ensure size at most 1.**

```
1  language Essence 1.3
2  given w, g, s : int(1..)
3  letting Golfers be new type of size g * s
4  find sched : set (size w) of
5                 partition (regular, numParts g, partSize s)
6                 from Golfers
7  such that
8    forAll g1, g2 : Golfers, g1 < g2 .
9      (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Constraint Modelling Pipeline: Conjure



- Conjure has alternative refinement rules for both decision variable and constraint representation.
- Allows us to explore the **space of models**.
- Heuristics to select models likely to be effective.

Parts (i.e. groups)

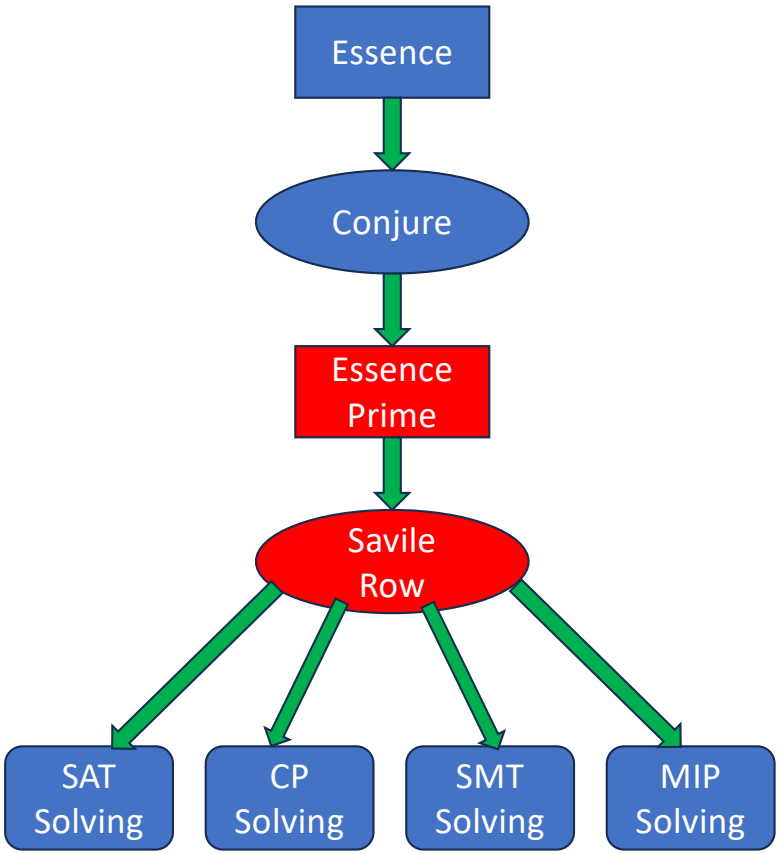
Elements of each part (i.e. golfers)

	1	2	..	g
1	0/1			
2	0/1			
..				
gxs	0/1			

Structural: each column sums to s.

The Pipeline: Savile Row

Constraint Modelling Pipeline: Savile Row

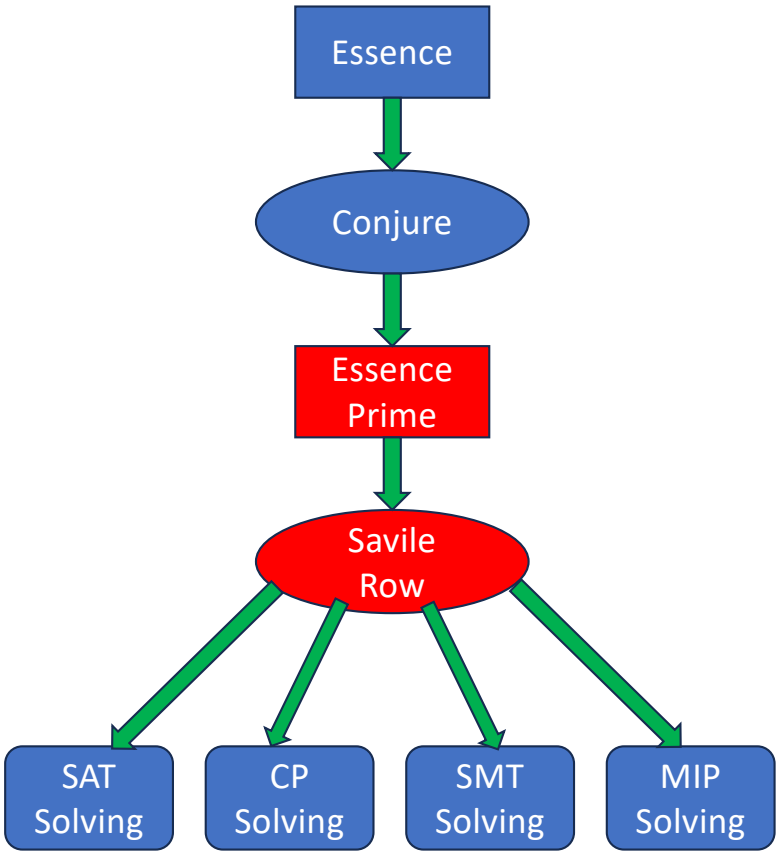


Social Golfers	3 weeks		
3 groups, size 3	[1, 2, 3]	[1, 4, 7]	[1, 5, 9]
	[4,5,6]	[2,5,8]	[2,6,7]
	[7,8,9]	[3,6,9]	[3,4,8]

Solution to an instance of Social Golfers

- The Essence Prime model is close to the input of a constraint solver.
- **Savile Row** is responsible for:
  - Tailoring this model to a particular solver
  - Or encoding to a different formalism.
- While further enhancing the model.
  - E.g. **Common subexpression elimination**, **tabulation**.

Constraint Modelling Pipeline: Savile Row

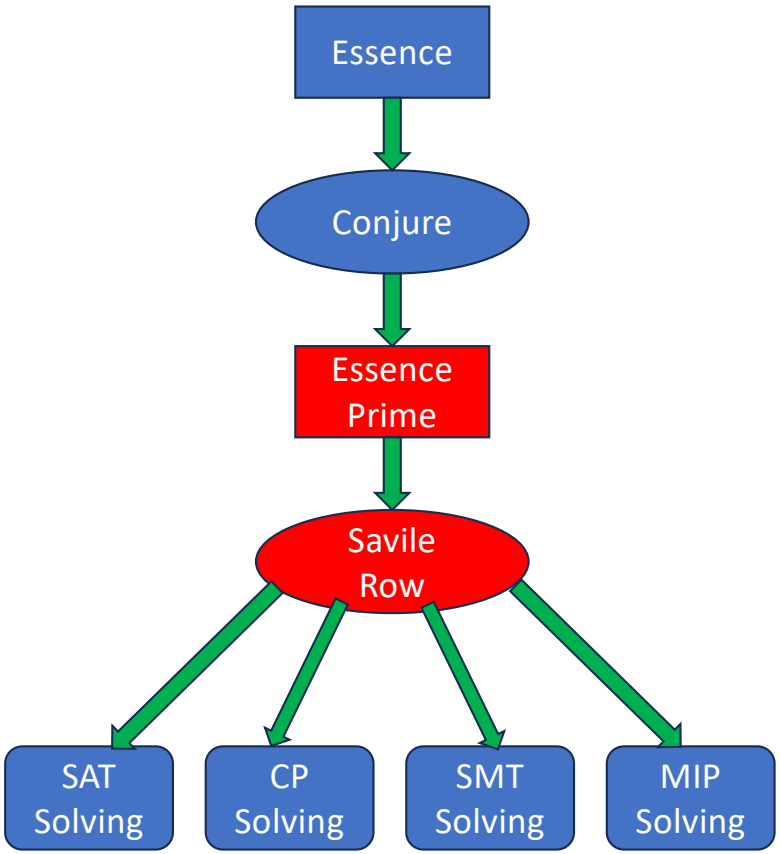


- Common Subexpression Elimination Example:
- Two constraints over four variables, each with domain {0, ..., 10}.
  - w + x + y + z = 6
  - z + y + w = 5

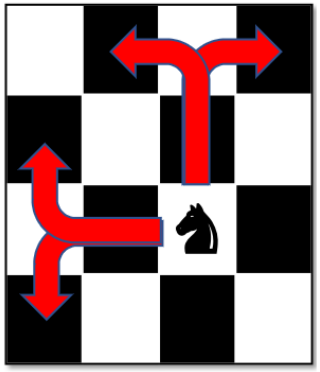
  Largest contiguous subexpression visible through normalisation: y + z
- Making both constraints consistent individually does not reveal that x = 1.
- Savile Row extracts the common subexpression w + y + z and replaces it with a variable a:
  - a = w + y + z
  - a = 5
  - x + a = 6

  Simple Bounds consistency reveals x = 1

Constraint Modelling Pipeline: Savile Row



- Tabulation Example:



```
given n: int
given startCol,startRow : int(0..n-1)
find tour : matrix indexed by [int(0..n*n-1)] of int(0..n*n-1)
such that
allDiff(tour),
tour[0] = startCol + (startRow)*n,
forAll i : int(0..n*n-2) .
((|tour[i]%n - tour[i+1]%n| = 1) /\ (|tour[i]/n - tour[i+1]/n| = 2)) \/
((|tour[i]%n - tour[i+1]%n| = 2) /\ (|tour[i]/n - tour[i+1]/n| = 1))
```

- These complex expressions constraining the moves **propagate poorly**.
- Instead tabulate their allowed values.
- GAC on the table.

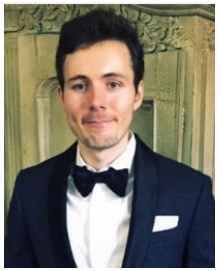
Fruitful Branches

Branches

- The Constraint Modelling Pipeline project has had several offshoots, some of which themselves have been developing for several years.
- Examples:
  - Automated Streamlining.
  - Athanor: Local search on Essence.
  - AutoIG: Automatic Instance Generation.

Streamlining





- Our pipeline infrastructure also supports **Streamlining**:
- As first proposed by Gomes and Sellmann in:
  - Gomes, C., Sellmann, M.: Streamlined constraint reasoning. In: Principles and Practice of Constraint Programming - CP 2004, pp. 274–289. Springer (2004)
- Streamlining is an attempt to focus the search onto a promising area of the search space.
- Streamliners are:
  - **Uninferred** constraints (i.e. not guaranteed to be sound), added to a constraint model.
  - Will reduce the search space, sometimes drastically.
  - The intention is to retain **at least one solution**.

Automated streamliner portfolios for constraint satisfaction problems. P. Spracklen, N. Dang, Ö. Akgün, I. Miguel. *Artificial Intelligence*, *319*, 2023.

Automating Streamlining

- As originally conceived, streamlining is a **manual** process.
- For a parameterised problem class of interest:
  - Solve small instances with an initial model.
  - Observe solutions to these instances, looking for patterns.
  - Use these patterns to conjecture candidate streamliners.
  - Test these streamliners on larger and more difficult instances.
- When successful, streamlining can lead to a huge reduction in search effort.
- An automated approach is therefore desirable.

First Order Streamlining Rules

- We have a set of **rules** that automate the generation of streamliners from the structures in an Essence specification.
- Examples:
  - **Integer**:
    - Take an odd (or similarly, even) value.
    - Restrict to upper (or lower) half of values.
  - **Function**:
    - Monotonically increasing (or decreasing).
    - Insist that a binary function is commutative (or non-commutative), or associative.
  - **Relation**:
    - Insist that it is reflexive, irreflexive, symmetric, …

Higher Order Streamlining Rules

- Lift first-order and higher-order streamlining rules to work on nested domain constructors of Essence.
- Examples: **all**, **half**, **at most one**.
- So now if we have a set of integers, we can say:
  - Half must be even.
- Or if we have a set of multisets of integers:
  - For half of the multisets restrict the elements to the upper half of their domain.

Candidate Streamliners

Problem	#Candidate Streamliners
BACP	108
BIBD	200
CoveringArray	64
Car Sequencing	36
EFPA	312
FLECC	144
Transshipment	68
Tail Assignment	336
Social Golfers	260
Vessel Loading	208

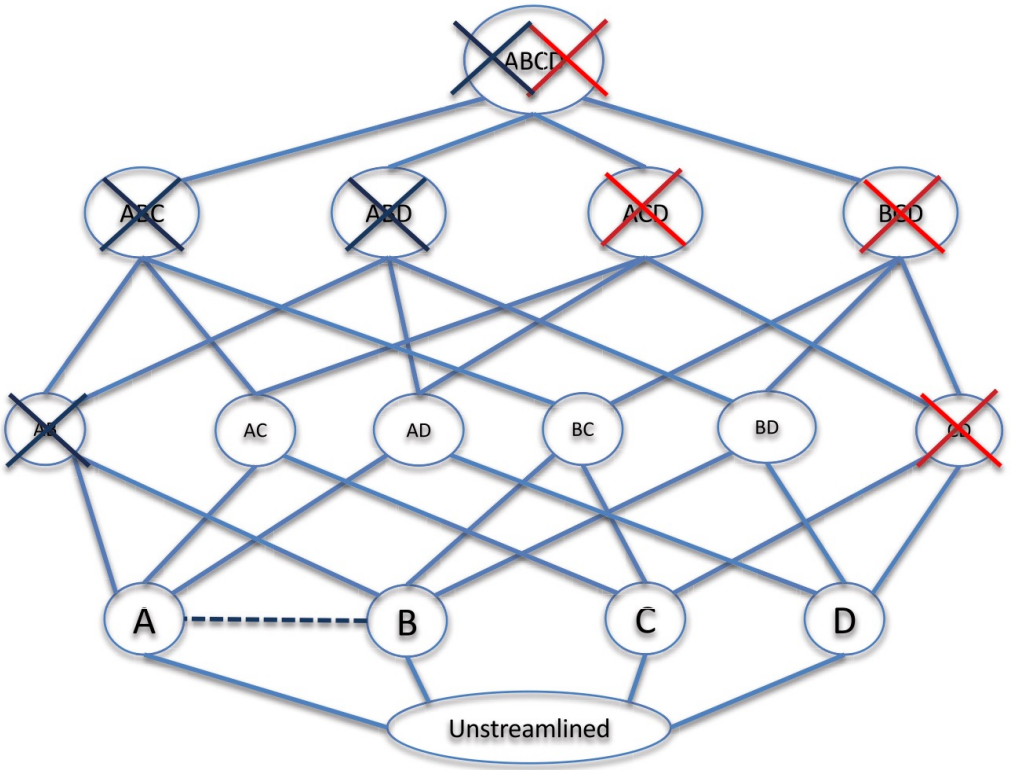
- Streamliners can also be **combined**.
- Simple example:
  - This integer variable should be both odd and restricted to the upper half of its domain.
- So, typically a large number of candidate streamliners for a problem class.
- Problem:
  - We don't know which will be **effective**.
    - Might remove all solutions, or not help.
  - We don't know how performance varies between instances.

Identifying Effective Combinations of Streamliners

- Two objectives:
  - **Applicability**: The proportion of training instances for which the streamlined model admits a solution.
  - **Reduction**: The mean search reduction in solving time achieved by the streamliner on the satisfiable instances.
- We want to search for a portfolio of Pareto-optimal streamlined models.
  - i.e. Streamliners whose <application, reduction> pair measure is not dominated.

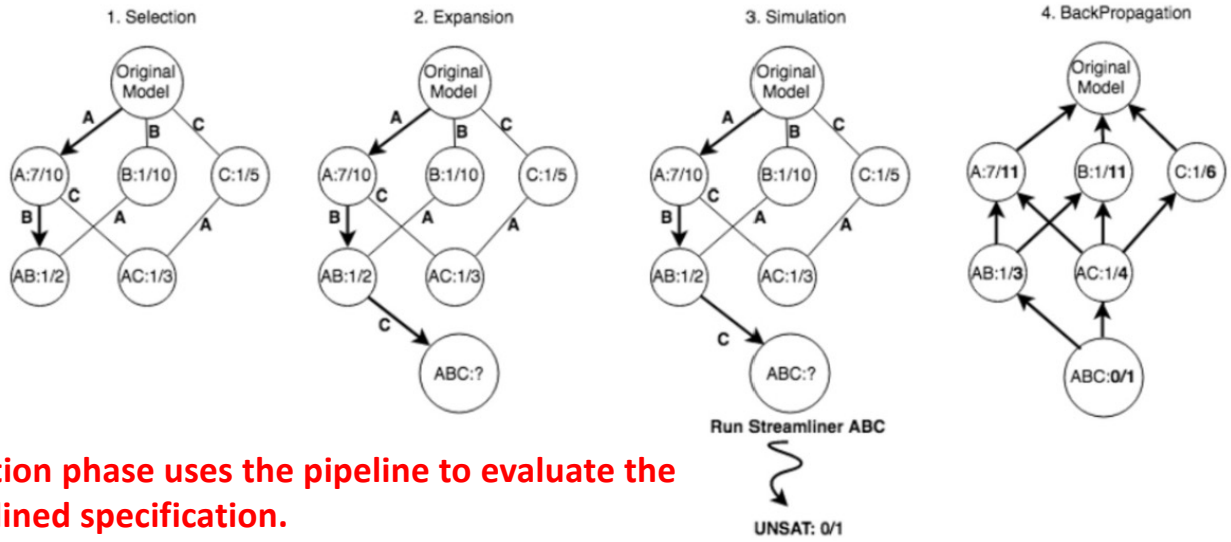
Identifying Effective Combinations of Streamliners

- Combining streamliners can result in larger performance gains.
- Ideally, we would consider the power set of streamliner combinations.
- The space can be **pruned**:
  - If a streamliner combination has zero applicability, its supersets are not explored.
  - Combinations of streamliners that are tagged as being mutually exclusive are not explored.



Monte Carlo Tree Search

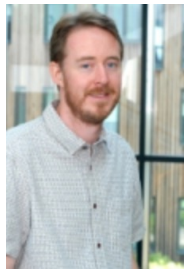
- Even after pruning, the number of combinations to consider is still typically too large to allow exhaustive enumeration.
- To search the lattice structure for a portfolio of Pareto optimal streamlined models we employ a Monte Carlo Tree Search style algorithm:



Simulation phase uses the pipeline to evaluate the streamlined specification.

Instance Generation via AutoIG



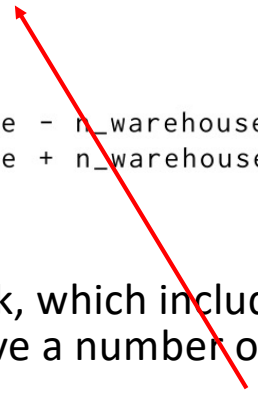



- Often the case that training instances required to integrate ML with CP.
- E.g. in Automated Streamlining we generate a large number of candidate training instances for each pair of problem class and solver via **AutoIG**.
- Allows users to describe the generation of instances for a given problem class declaratively as an **instance generation model**.
  - Parameters to the Essence specification of the original model, which describe instance data that define an individual instance, are transformed into decision variables.
  - Constraints are added to capture bounds or relationships between the instance data.
- Solutions to the instance generation model are **instances of the original problem**.

A framework for generating informative benchmark instances. N Dang, O Akgun, J Espasa, I Miguel and P Nightingale. CP 2022.

Fragment of an Instance Generation Model

```
given n_warehouses_middle: int(1..100)
given n_warehouses_delta: int(0..49)
find n_warehouses: int(1..100)
such that
  n_warehouses >= n_warehouses_middle - n_warehouses_delta,
  n_warehouses <= n_warehouses_middle + n_warehouses_delta
```



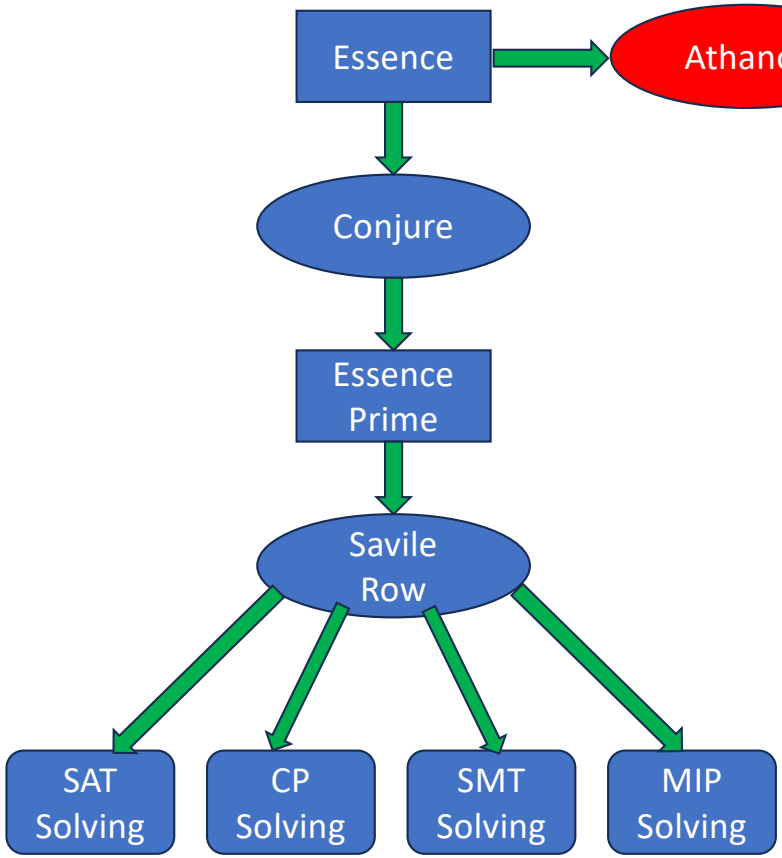
- From the transshipment problem.
  - Considers the design of a distribution network, which includes a **number of warehouses** and transshipment points to serve a number of customers.
- The instance generation model is itself parameterised.
  - This is to allow the algorithm configuration tool Irace to control the instance generation process.
- We search for satisfiable instances that are solvable by a chosen solver within the solving time range of, e.g., [10,300] seconds.
  - Lower bound of 10 seconds avoids trivially solvable instances, as the gain when applying streamliners on such instances are often negligible.

Searching for Instances

```
given n_warehouses_middle: int(1..100)
given n_warehouses_delta: int(0..49)
find n_warehouses: int(1..100)
such that
  n_warehouses >= n_warehouses_middle - n_warehouses_delta,
  n_warehouses <= n_warehouses_middle + n_warehouses_delta
```

- Irace decides on an instance of the generator model.
  - I.e. values for these parameters.
- Solved to produce an instance of the original model.
- This in turn is solved to evaluate its suitability for training.
  - I.e. within [10,300] seconds.
- The results fed back to Irace to guide its future choices.
- Instance generation stops once a given tuning budget is exhausted.
  - All instances satisfying the required properties are returned.
- Clustering can then be used to to select a smaller representative subset of training instances.

Solving Essence Specifications Directly.

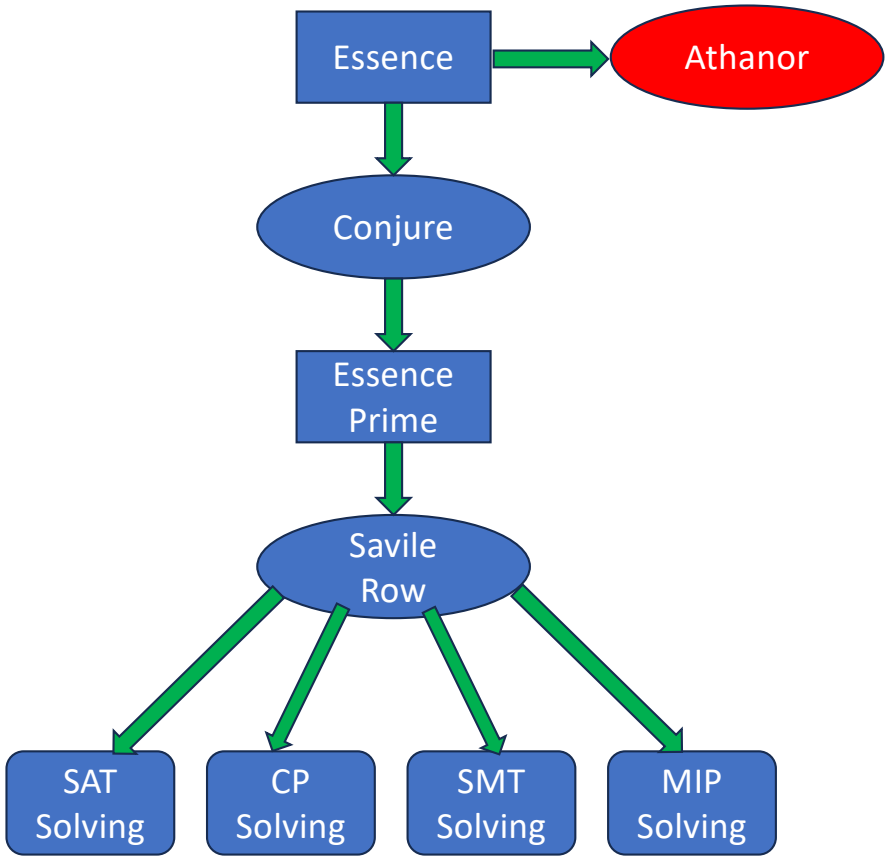





- Rather than the refinement approach of the pipeline, we can attempt to solve an Essence specification directly.
- **Athanor** takes a **constraint-based local search** approach.
  - Generating an initial assignment.
  - Iteratively modify this assignment to improve an objective through a sequence of moves.
  - Selected from a **neighbourhood** of assignments reachable from the current assignment.

Athanor: high-level local search over abstract constraint specifications in Essence. S Attieh, N Dang, C Jefferson, I Miguel, P Nightingale. IJCAI 2019.

Solving Essence Specifications Directly.



- Advantage of proceeding from Essence:
  - structure apparent in an abstract specification of a problem can be exploited to generate high quality **neighbourhoods**.
- Proceeds from the Essence types:
  - Set, multiset, sequence, function, relation, and partition.
- Neighbourhoods preserve this structure:
  - Select a set and remove an element.
  - Select a set and add an element.
  - Select two sets, move an element from one to the other: moves connections to where they may be better used, i.e. to connect to more nodes.

Similar neighbourhoods for other types.

Summary

Conclusions

- The Constraint Modelling Pipeline has been in development for over two decades.
- A main branch, and fruitful side-projects.
- Sustained by a combination of continuity and collaboration.
- Actively interested in interfacing with your work/ideas.