

# Why ESSENCE?

## Frequently Asked Questions about a New Language for Specifying Combinatorial Problems

**Alan M. Frisch** (Univ. of York)

**Matthew Grum** (Univ. of York)

**Christopher Jefferson** (Oxford Univ.)

**Bernadette Martinez Hernandez** (Univ. of York)

**Ian Miguel** (Univ. of St. Andrews)

# ESSENCE FAQ: The dream

*Why is your product so vastly superior in all respects to those of your competitors?*

# ESSENCE FAQ: The reality

*Why the hell are you wasting your time on this?*

# Outline

- What is ESSENCE?
- What motivated you to design ESSENCE?
- What were the objectives in designing ESSENCE?
- Questions from the audience:
  - 11 others from the paper
  - Any of your own

# Question 1

**What is ESSENCE?**

# ESSENCE is ...

- A language for specifying combinatorial *problems*.
- Natural:
  - Enables formal specifications similar to the rigorous ones that people give using a mixture of natural language and discrete mathematics (e.g., Garey and Johnson).
  - Intended to be accessible to anyone with a background in discrete mathematics (not constraint programming).

# ESSENCE is ...

- Not a logical language such as Z or NP-SPEC.
- Similar to OPL, F or ESRA enhanced with features that greatly enhance its level of abstractness
  - Since combinatorial problems require finding a certain type of combinatorial object, ESSENCE provides decision variables whose domains are objects of that type.
  - This enables problems to be stated directly, *without modelling* the decision variable as a collection of simpler decision variables.

# Is ESSENCE Natural?

## ESSENCE

given  $U$  enum(...),  $s, v: U \rightarrow \text{int}(1\dots)$ ,  $B, K: \text{int}(1\dots)$

find  $U'$ : set of  $U$

such that  $\sum_{u \in U'} s(u) \leq B$ ,  $\sum_{u \in U'} v(u) \geq K$

## Garey and Johnson

INSTANCE: Finite set  $U$ , for each  $u$  in  $U$ : a size  $s(u) \in \mathbb{Z}^+$ , a value  $v(u) \in \mathbb{Z}^+$  and positive integers  $B$  and  $K$ .

QUESTION: Is there a subset  $U' \subseteq U$  such that  $\sum_{u \in U'} s(u) \leq B$  and  $\sum_{u \in U'} v(u) \geq K$

# The SONET Problem

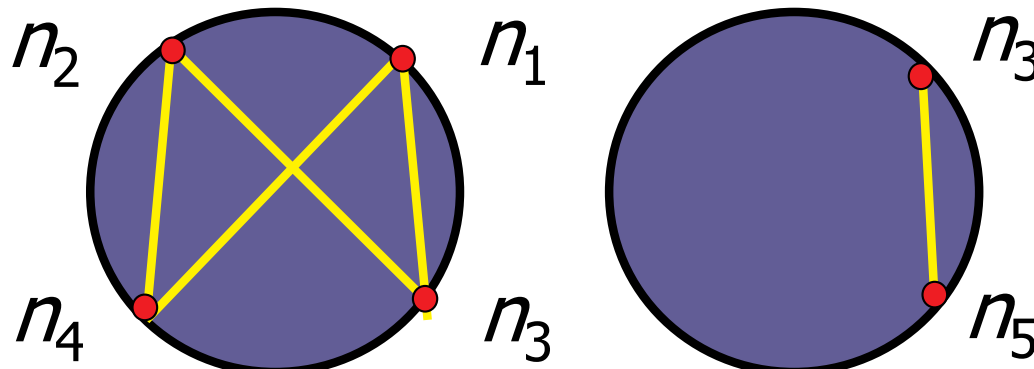
## Specification

- Given *nrings* rings, *nnodes* nodes, a set of pairs of nodes (communication *demand*) and an integer *capacity* (of each ring). Install nodes on rings satisfying demand and capacity constraints. Minimise installations.

## Instance

- nrings*=2, *nnodes*=5, *capacity* = 4
- demand*:  $n_1$  &  $n_3$ ,  $n_1$  &  $n_4$ ,  $n_2$  &  $n_3$ ,  $n_2$  &  $n_4$ ,  $n_3$  &  $n_5$

## Solution

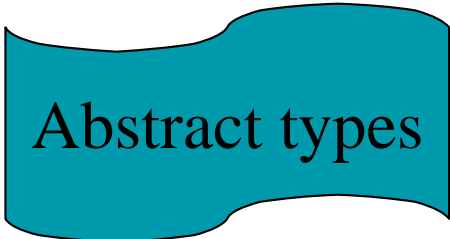




# ESSENCE Provides Abstract Types

Atomic: int, bool, enum, unnamed

Constructors: sets, multisets, partitions, functions, relations, tuples



**given**

*nrings, nnodes, capacity* : int (1...),  
*demand* : set of set (size 2) of int (1..*nnodes*)

**find**

*network* : **mset** (size *nrings*) of **set** (maxsize *capacity*)  
of int (1..*nnodes*)

**minimising**  $\sum_{ring \in network} \cdot |ring|$

**such that**

$\forall_{pair \in demand} \cdot \exists_{ring \in network} \cdot pair \subseteq ring$

# ESSENCE Supports Arbitrarily-Nested Types

Arbitrary nesting  
of types

**given**

$nrings, nnodes, capacity : \text{int } (1\dots),$   
 $demand : \text{set of set (size 2) of int } (1..nnodes)$

**find**

$network : \text{mset (size } nrings) \text{ of set (maxsize } capacity)$   
 $\text{of int } (1..nnodes)$

**minimising**  $\sum_{ring \in network} \cdot |ring|$

**such that**  $\forall_{pair \in demand} \cdot \exists_{ring \in network} \cdot pair \subseteq ring$

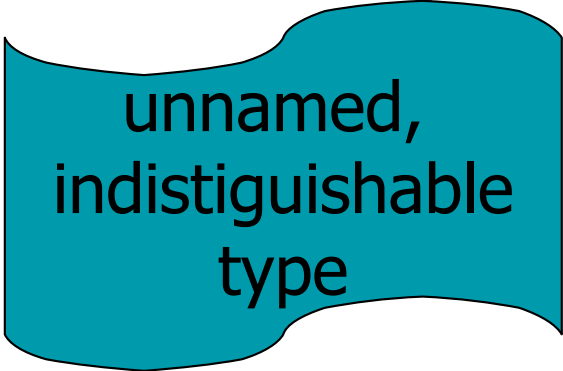
# ESSENCE Unnamed, Indistinguishable Types

**given**  $w, g, s : \text{int } (1\dots)$

**letting** *golfers* be new type of size  $g*s$

**find** *schedule* : mset (size  $w$ ) of rpartition (size  $s$ ) of *golfers*

**such that** .....



unnamed,  
indistinguishable  
type

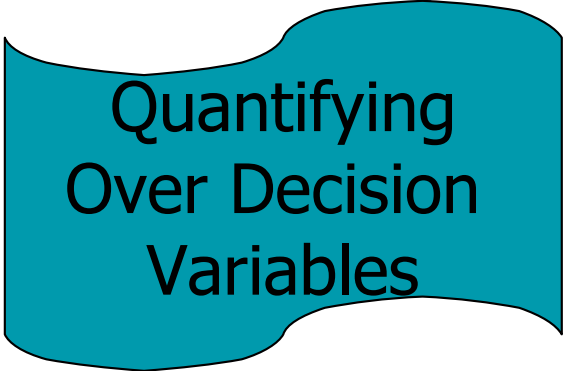
# ESSENCE Supports Quantification over Decision Variables

**given**  $nrings, nnodes, capacity : \text{int } (1\dots),$   
 $demand: \text{set of set (size 2) of int } (1..nnodes)$

**find**  $network: \text{mset (size } nrings) \text{ of set (maxsize } capacity)$   
 $\text{of int } (1..nnodes)$

**minimising**  $\sum_{ring \in network} |r|$

**such that**  $\forall_{pair \in demand} \cdot \exists_{ring \in network} \cdot pair \subseteq ring$



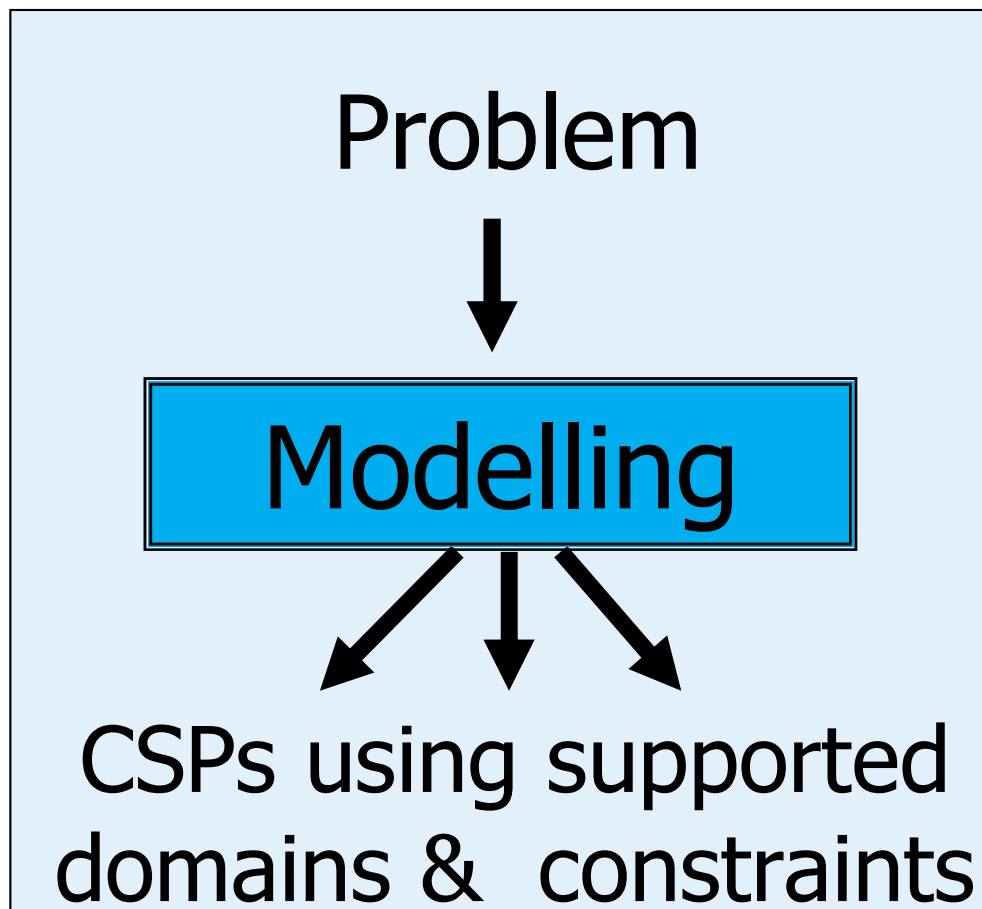
Quantifying  
Over Decision  
Variables

# Question 2

**What Motivated You to  
Design ESSENCE?**

# Motivation 1: Automated Modelling

## What is modelling?

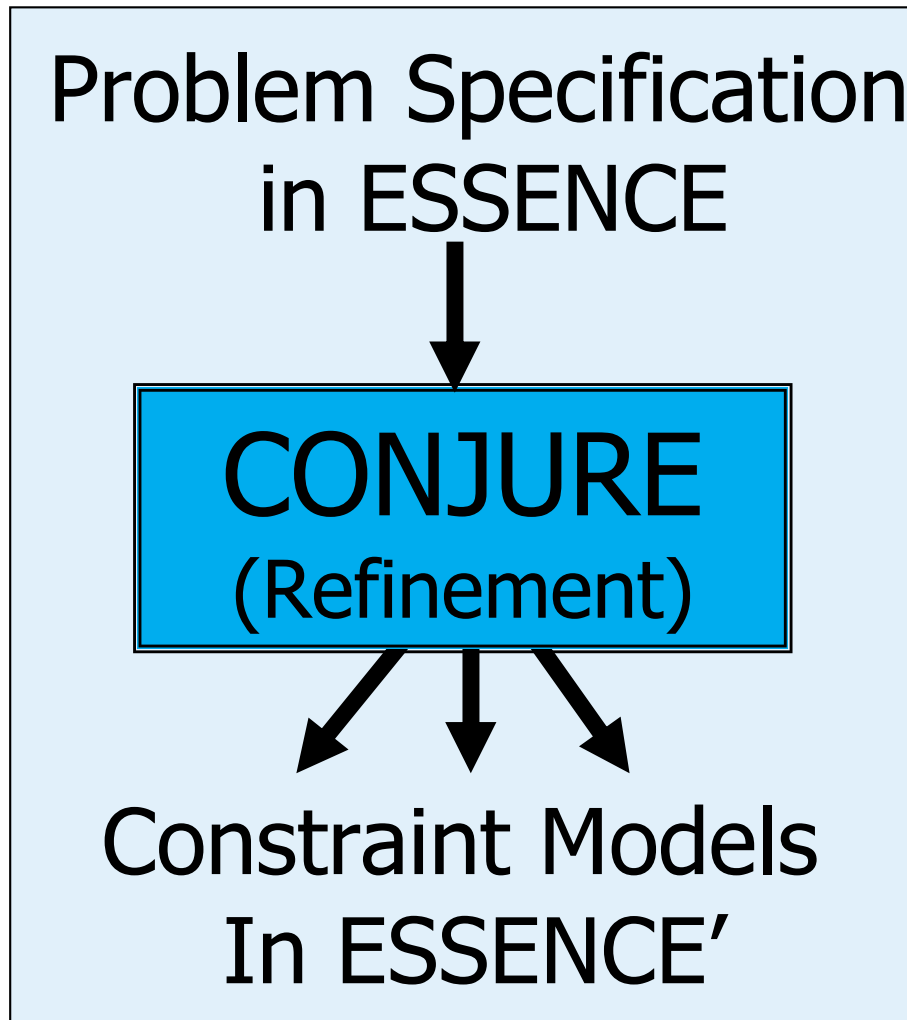


Usually unsystematic –  
an art

Social Golfers Problem  
requires finding a multiset  
of partitions.

At least 72 ways to model  
this with a set of atomic or  
atomic set variables

# Motivation 1: Automated Modelling



- Input spec must be sufficiently abstract so that no modelling decisions have been made in constructing it
- Thus, spec language provide level of abstraction above that at which modelling decisions are made
- Having such a *problem specification language* is a prerequisite to studying automated modelling

# Motivation 2: Human Communication

- Formal problem specifications could facilitate communication between humans better than the informal ones currently used.
  - Example: Could be used in CSPLib.
  - Requires availability of problem specification language

# Question 3

**What Were the Objectives in  
Designing ESSENCE?**

# Objective 1: Naturalness

- Necessary for human communication
- Necessary for input to automated modelling system
  - One cannot claim to have an automated modelling system if using it requires a major translation into the system's input language.

# Objective 2: Abstractness

- Necessary for input of automated modelling system
- Necessary to obtain naturalness

# Objective 3: Capture CSP

- All problems specified in the language must be reducible to finite-domain CSP.
  - Example: syntax ensures that every decision variable has a *finite* domain. Bounds of a matrix cannot be a decision variable.

# Further Questions

- What evidence is there that you have met the design objectives?
- There already exist many specification languages, most notably Z. Do we really need another one?
- ESSENCE appears to provide some redundant type constructors. Why?
- ESSENCE appears to be missing type constructors for some important combinatorial objects. It also appears to be missing some of my favourite operators. Why?
- Why does ESSENCE provide a **maxsize** annotation but no **minsize** annotation?
- Why is it so important to avoid introducing symmetry into problem specs?
- What is the current status of ESSENCE?
- What do ESSENCE and ESSENCE' have to do with automated modelling?
- What progress have you made towards the automation of modelling?
- What are your plans for future work

# Further, Further Questions

- Though ESSENCE has been shown to be useful in specifying a wide range of problems, how do we know that you haven't selected the problems based on their ease of specification?

## Question 10

**Why is it so important to avoid introducing symmetry into problem specifications?**

# Sources of Symmetry in Constraint Models

- Inherent in problem
  - E.g. rotation of chess board in n-queens
- In modelling
  - Since modelling languages often force a spec to
    - Introduce unnecessary objects
    - Unnecessarily distinguish between objects
  - Example: Social Golfers Problem

# Sufficient Facilities for Abstraction?

- Never force introduction of unnecessary objects or distinctions
- Guiding principle in design of ESSENCE
- Elimination of symmetry has been used to evaluate ESSENCE
  - Every problem we have considered has an ESSENCE spec that contains no symmetries other than those inherent in problem.
  - No other language meets this test

# Important Consequence

- All non-inherent symmetries in a model have been introduced by modelling process
- Hypothesis: Modelling is systematic and symmetries are introduced in a systematic way.
  - An automated modelling system ought to be able to identify the symmetries it has introduced
  - CONJURE prototype does it!

# Question 11

**What is the current status of  
ESSENCE?**

# Status of ESSENCE

- Full definition of Version 1.1.0
  - Syntax and semantics
- Haskell implementation of parser complete
  - Type checking, type inference, category checking
- Java implementation of parser nearing completion
- ESSENCE' 1.b.a (subset of Essence 1.1.0)
  - Solver-independent modelling language (OPL-like)
  - Translator to Eclipse is implemented
  - Translator to Minion under development

# How Usable is ESSENCE?

- Specifications of  $\sim 50$  problems found in the CSP literature written by an undergraduate with **no** background in constraint programming.
- URL: <http://www.cs.york.ac.uk/aig/constraints/>

# Further Information

- <http://www.cs.york.ac.uk/aig/constraints/>
  - Our papers
  - Catalogue of CONJURE rules
  - Syntax and semantics of ESSENCE version 1
  - Catalogue of ~50 problems specified in ESSENCE and other constraint languages