The Design of ESSENCE

A Constraint Language for Specifying Combinatorial Problems

Alan M. Frisch
Artificial Intelligence Group
Dept of Computer Science
University of York

Collaborators: Matthew Grum, Warwick Harvey, Chris Jefferson, Bernadette Martinez Hernandez, Ian Miguel
ESSENCE is....

• A **formal** language for specifying **combinatorial problems**
  - problem classes, not just instances
Motivation
Motivation 1: Automated Constraint Modelling

What is constraint modelling?

- Usually unsystematic – an art
- Social Golfers Problem requires finding a multiset of partitions.
- At least 72 ways to model this with atomic or set variables
Motivation 1: Automated Constraint Modelling

• Input spec must be sufficiently abstract so that no modelling decisions have been made in constructing it

• Thus, spec language must 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

Problem Specification in ESSENCE

CONJURE (Refinement)

Constraint Models In ESSENCE’
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.
Design Objectives
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.

- Examples
  - Syntax ensures that every decision variable has a finite domain.
  - Bounds of a matrix cannot be a decision variable.
ESSENCE by Example
### ESSENCE

<table>
<thead>
<tr>
<th>given</th>
<th>U new type enum, s,v : function (total) U → int (1..), B, K: int(1..)</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td>U′: set of U</td>
</tr>
<tr>
<td>such that</td>
<td>∑ u∈U′. s(u) ≤ B, ∑ u∈U′. v(u) ≥ K</td>
</tr>
</tbody>
</table>
ESSENCE

given  U new type enum, 
      s,v : function (total) U → int (1..), 
      B, K: int(1..)

find  U′: set of U

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


INSTANCE: Finite set U, 
      for each u ∈ 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
      \( \sum_{u \in U′} v(u) \geq K ? \)
**ESSENCE**

**given**
- $U$ new type enum,
- $s, v : \text{function (total)} \ U \rightarrow \text{int (1..)}$,
- $B, K : \text{int(1..)}$

**find**
- $U' : \text{set of } U$

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

---

**Garey & Johnson** [A Guide to the Theory of NP-completeness] 300+ problem specs

**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
- $\sum_{u \in U'} v(u) \geq K$?
### Name the Problem?

### ESSENCE

<table>
<thead>
<tr>
<th>given</th>
<th>U new type enum, s,v : function (total) U → int (1..), B, K: int(1..)</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td>U': set of U</td>
</tr>
<tr>
<td>such that</td>
<td>∑ u∈U'. s(u) ≤ B, ∑ u∈U'. v(u) ≥ K</td>
</tr>
</tbody>
</table>


**INSTANCE:** Finite set U, for each u∈U: a size s(u)∈Z^+, a value v(u)∈Z^+ and positive integers B and K.

**QUESTION:** Is there a subset U'⊆U such that ∑u∈U' s(u) ≤ B, and ∑u∈U' v(u) ≥ K?
**Name the Problem?**

### ESSENCE

<table>
<thead>
<tr>
<th>Given</th>
<th>Find</th>
<th>Such that</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U ) new type enum, ( s, v : \text{function} \ (\text{total}) \ U \rightarrow \text{int} \ (1..), )</td>
<td>( U' : \text{set of} \ U )</td>
<td>( \sum_{u \in U'} s(u) \leq B, )</td>
</tr>
<tr>
<td>( B, K : \text{int}(1..) )</td>
<td></td>
<td>( \sum_{u \in U'} v(u) \geq K )</td>
</tr>
</tbody>
</table>

### Name the Problem?  

**Garey & Johnson** [A Guide to the Theory of NP-completeness] 300+ problem specs

**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 \( \sum_{u \in U'} v(u) \geq K \)?
### Optimisation

**ESSENCE**

<table>
<thead>
<tr>
<th>given</th>
<th>$U$ new type enum, $s,v : function (total) U \rightarrow \text{int (1..)}, B: \text{int(1..)}$</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td>$U'$: set of $U$</td>
<td>vars/domains</td>
</tr>
<tr>
<td>such that</td>
<td>$\sum_{u \in U'} s(u) \leq B$</td>
<td>constraints</td>
</tr>
<tr>
<td>maximizing</td>
<td>$\sum_{u \in U'} v(u)$</td>
<td>objective</td>
</tr>
</tbody>
</table>
### Specifying the Valid Instances

#### ESSENCE

<table>
<thead>
<tr>
<th>given</th>
<th>$U$ new type enum, $s,v : \text{function (total)} \ U \rightarrow \text{int (1..)}$, $B: \text{int(1..)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>where</td>
<td>$\forall u \in U . s(u) \leq B$</td>
</tr>
<tr>
<td>find</td>
<td>$U'$: set of $U$</td>
</tr>
<tr>
<td>such that</td>
<td>$\sum_{u \in U'} s(u) \leq B,$</td>
</tr>
<tr>
<td>maximizing</td>
<td>$\sum_{u \in U'} v(u)$</td>
</tr>
</tbody>
</table>

- **parameters**
- **restrictions**
- **vars/domains**
- **constraints**
- **objective**
Two Other Kinds of Statements

- language ESSENCE version 1.2.0
  - distinguish ESSENCE from ESSENCE’
  - used to check version compatibility

- letting ...
  - used to declare identifiers and user-defined types
  - examples appear later
Instance Data

- Gives values to parameters
- Defined in a separate file
- Not considered in this talk
The SONET Problem

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:** \( nrings=2, \ nnodes=5, \ capacity = 4 \)
\( demand: \ n1 \ & \ n3, \ n1 \ & \ n4, \ n2 \ & \ n3, \ n2 \ & \ n4, \ n3 \ & \ n5 \)

**Solution:**

```
\[ n2 \]
\[ n1 \]
\[ n4 \]
\[ n3 \]
\[ n5 \]
```
SONET Problem in ESSENCE

given    \( nrings, \ nnodes, \ capacity: \ \text{int} \ (1..) \)

letting  \( \text{Nodes} \) be domain \( \text{int} \ (1..\nnodes) \)

given    \( \text{demand} \): set of set (size 2) of \( \text{Nodes} \)

find     \( \text{network} \): mset (size \( nrings \)) of set (maxSize \( capacity \)) of \( \text{Nodes} \)

minimising \( \sum \ \text{ring} \in \text{network}. \ |\text{ring}| \)

such that \( \forall \ \text{pair} \in \text{demand}. \ \exists \ \text{ring} \in \text{network}. \ \text{pair} \subseteq \text{ring} \)
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.

**Instance:** \( s = g = w = 3 \)

**Solution:**

<table>
<thead>
<tr>
<th>weeks</th>
<th>groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,2,3]</td>
<td>[4,5,6]</td>
</tr>
<tr>
<td>[1,4,7]</td>
<td>[2,5,8]</td>
</tr>
<tr>
<td>[1,5,9]</td>
<td>[2,6,7]</td>
</tr>
</tbody>
</table>
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.

\[
\text{given} \quad w, \ g, \ s: \text{int} \\
\text{letting} \quad \text{golfers} \text{ be new type of size } g\ast s \\
\text{find} \quad \text{sched} : \text{set (size w) of partition (numParts g, partSize s) from golfers} \\
\text{such that} \quad \forall \text{week}_1, \text{week}_2 \in \text{sched. } \text{week}_1 \neq \text{week}_2 \rightarrow \\
\quad \forall \text{group}_1 \in \text{parts(week}_1), \text{ group}_2 \in \text{parts(weeks}_2). \\
\quad | \text{group}_1 \cap \text{group}_2 | < 2
\]
Processing Stages
T - I - S

TIS is more general, much harder to implement.
Conjure does TIS (which is uncommon).
ESSENCE enables TIS (hence ITS).

I - T - S

problem spec

Translate

Instance data

Instantiate

Solve

Solutions

problem spec

instance data

Instantiate

Translate

Solve

Solutions
Types vs Domains
Types vs Domains

• Types
  - ESSENCE is strongly typed language.
  - Every expression has a type independent of where it occurs, which can be inferred and checked for correctness.
  - Types are used to determine the denotation of overloaded operators. E.g.: set union vs multiset union.

• Domains
  - ESSENCE is a finite-domain language.
  - Every decision variable has a finite domain of values.
  - Domains can be quite intricate sets. E.g.: any finite set of integers
Types ≠ Domains

• It is tempting to view types and domains as the same things
  - both prescribe a range of values a variable can take

• Consequence: the intricate sets defined as domains would have to be handled by the type system. E.g. every finite set of integers would be a type.

• Design goal: keep the type system simple. (Domains must be rich).

• Therefore: Types ≠ Domains
Types vs Domains

- Each **type** denotes a non-empty sets that contain all elements that have a similar structure.

- Each **domain** denotes a possibly-empty set whose elements are drawn from the same type.
  - Thus, each domain is associated with an underlying type.
    - E.g.: domain comprising 1..10 has underlying type `int`.
    - E.g.: domain comprising all sets of two integers between 1 and 10 has underlying type `set of int`.

- Type checking, type inference and operator overloading are based on types, not domains.
Types vs Domains

-Domains can (and often do) contain parameters
  - E.g., in SONET:
    find *network*: mset (size *nrings*) of set (maxSize *capacity*) of *Nodes*
  - hence their values are determined at instantiation time

-Types can’t contain parameters
  - thus they can be reasoned with at translation time (prior to instantiation) to determine grammaticallity.
Types vs Domains
Different Roles in the Grammar

• The grammar is specified by a typed BNF.

• “Domain” is a non-terminal in this grammar.
  - It generates the domains of the language
  - There is no “type” non-terminal, types do not appear in ESSENCE specs (though every expression has a unique type).
Types
## Types

- **Atomic types**

<table>
<thead>
<tr>
<th>Atomic types</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>all user-defined enumerated types</td>
<td>letting players be new type enum {alan, berna, chris, ian}</td>
</tr>
<tr>
<td></td>
<td>given U new type enum</td>
</tr>
<tr>
<td>all user-defined unnamed types</td>
<td>letting golfers be new type of size g*s</td>
</tr>
</tbody>
</table>
Types

- Type constructors
  - where $\tau$ is any type and $\omega$ is any ordered type (bool, int, any enumerated type)

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Denotation (approximate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>set of $\tau$</td>
<td>every finite subset of $\tau$</td>
</tr>
<tr>
<td>mset of $\tau$</td>
<td>every finite multiset drawn from $\tau$</td>
</tr>
<tr>
<td>function $\tau_1 \rightarrow \tau_2$</td>
<td>every finite partial function with domain $\tau_1$ and codomain $\tau_2$</td>
</tr>
<tr>
<td>tuple $(\tau_1, \ldots, \tau_n)$</td>
<td>$\tau_1 \times \cdots \times \tau_n$</td>
</tr>
<tr>
<td>rel of $(\tau_1 \times \cdots \times \tau_n)$</td>
<td>every finite subset of $\tau_1 \times \cdots \times \tau_n$</td>
</tr>
<tr>
<td>partition from $\tau$</td>
<td>every partition of every finite subset of $\tau$</td>
</tr>
<tr>
<td>matrix indexed by $[\omega_1, \ldots, \omega_n]$ of $\tau$</td>
<td>every $n$-dimensional matrix such that each dimension $i$ is indexed by some finite range of values of type $\omega_i$ and each matrix entry is a member of $\tau$.</td>
</tr>
</tbody>
</table>

- Others could easily be added.
Domains
Domains

- A set of values all of the same type.
- Formed by annotating a type name with restrictions that select particular values of the type.
  - `int (1..10)` (underlying type: int)
  - `set (size 2) of int (1..10)` (underlying type: set of int)
- Every type is a domain (with no restrictions).
- Annotations can contain parameter expressions, but not unbound quantified variables or decision variables.
Annotating Atomic Types

- players \((\text{alan..chris})\)
- int \((1..3, 5..10)\)
- int \((1..)\)
- int \((..1, 1..)\)
- int \((\text{lower..upper})\)
- int \((S U \{0,1}\))
Annotating Type Constructors

- set, mset, partition, relation, function can be annotated with size, maxSize, minSize

- mset can be annotated with maxOccur, minOccur

- Examples
  - mset (maxOccur 5) of int
  - set (minSize m, maxSize n) of int
  - partition (size n+2) from mset (maxSize 4) of int (1..100)

- Syntax designed to assure non-ambiguity
Annotating Type Constructors

- function can be annotated with one of partial and total and one of surjection, injection, bijection.

- Examples
  - function (total, injection) players → players
  - function (total) int → int (empty)
  - function (total) int(1..10) → int (infinite)
  - function (total) int(1..10) → int(1..10) (finite)
Annotating Type Constructors

• partition can be annotated with
  - partSize, maxPartSize, minPartSize,
  - numParts, maxNumParts, minNum
  - regular, complete

• Examples
  - partition from int (1..32)
  - partition (size 32) from int (1..32)
  - partition (complete) from int (1..32)
  - partition (partSize 4, complete) from int (1..32)
  - partition (partSize 6, complete) from int (1..32)
Three Ways to Use Domains

• Every decision variable must have a domain
  - find $M$: matrix indexed by $[\text{int}(1..k)]$ of $\text{int} (1..n)$

• Every parameter must have a domain
  - given $M$: matrix indexed by $[\text{int}(1..10)]$ of $\text{int}$
  - given $N$: matrix indexed by $[\text{int}(1..)]$ of $\text{int}$

• Quantified variables can be drawn from a domain
  - $\forall x: \text{int}(1..10)$ . ....
  - $\forall s: \text{set (size 2) of int (1..10)}$ . ....
Three Ways to Use Domains

• Every decision variable must have a domain
  - find $M$: matrix indexed by $\{\text{int}(1..k)\}$ of int $(1..n)$
  - **must be finite**.

• Every parameter must have a domain
  - given $M$: matrix indexed by $\{\text{int}(1..10)\}$ of int
  - given $N$: matrix indexed by $\{\text{int}(1..)\}$ of int
  - **can be infinite**.

• Quantified variables can be drawn from a domain
  - $\forall x: \text{int}(1..10)$ . ....
  - $\forall s: \text{set (size 2)}$ of int $(1..10)$ . ....
  - **must be finite**.
Three Ways to Use Domains

• Every decision variable must have a domain
  - find $M$: matrix indexed by $\text{int}(1..k)$ of int $(1..n)$
  - must be finite. **must be definite.**

• Every parameter must have a domain
  - given $M$: matrix indexed by $\text{int}(1..10)$ of int
  - given $N$: matrix indexed by $\text{int}(1..)$ of int
  - can be infinite. **can be indefinite.**

• Quantified variables can be drawn from a domain
  - $\forall x: \text{int}(1..10)$ . ....
  - $\forall s: \text{set (size 2)}$ of int $(1..10)$ . ....
  - must be finite. **must be definite.**
Expressions
Expressions

- Denote objects in the universe of computation
- Every object in the universe of computation has a name in the language. Achieved by a set of value constructors.
### Expressions

<table>
<thead>
<tr>
<th></th>
<th>( \tau ) is set of ( \tau' )</th>
<th>( \tau ) is mset of ( \tau' )</th>
<th>( \tau ) is partition from ( \tau' )</th>
<th>( \tau ) is rel of ( (\tau_1 \times \cdots \times \tau_n) )</th>
<th>( \tau ) is function from ( \tau_1 \rightarrow \tau_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( e_1:\tau \cup e_2:\tau )</td>
<td>( \tau )</td>
<td>( \tau )</td>
<td>( \tau )</td>
<td>( \tau )</td>
</tr>
<tr>
<td>2</td>
<td>( e_1:\tau \cap e_2:\tau )</td>
<td>( \tau )</td>
<td>( \tau )</td>
<td>( \tau )</td>
<td>( \tau )</td>
</tr>
<tr>
<td></td>
<td>( e_1:\tau \subseteq e_2:\tau )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( e_1:\tau \supset e_2:\tau )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( e_1:\tau \subset e_2:\tau )</td>
<td>bool</td>
<td>bool</td>
<td>bool</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>( e_1:\tau \subseteq e_2:\tau )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( e_1:\tau \supset e_2:\tau )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(</td>
<td>e:\tau</td>
<td>)</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>5</td>
<td>( e':\tau' \in e:\tau )</td>
<td>bool</td>
<td>bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>( \text{max}(e:\tau) )</td>
<td>( \tau' \uparrow )</td>
<td>( \tau' \uparrow )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \text{min}(e:\tau) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>( \text{toSet}(e:\tau) )</td>
<td>set</td>
<td>set</td>
<td>set of</td>
<td>set of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of ( \tau' )</td>
<td>of ( \tau' )</td>
<td>tuple ( \langle \tau_1, \ldots, \tau_n \rangle )</td>
<td>tuple ( \langle \tau_1, \tau_2 \rangle )</td>
</tr>
<tr>
<td>8</td>
<td>( \text{toMset}(e:\tau) )</td>
<td>mset</td>
<td>mset</td>
<td>mset of</td>
<td>mset of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of ( \tau' )</td>
<td>of tuple ( \langle \tau_1, \ldots, \tau_n \rangle )</td>
<td>tuple ( \langle \tau_1, \tau_2 \rangle )</td>
<td>tuple ( \langle \tau_1, \tau_2 \rangle )</td>
</tr>
<tr>
<td>9</td>
<td>( \text{toRel}(e:\tau) )</td>
<td></td>
<td></td>
<td>rel of</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( (\tau_1 \times \tau_2) )</td>
<td></td>
</tr>
</tbody>
</table>
## Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>( \tau ) is</th>
<th>( \tau ) is</th>
<th>( \tau ) is</th>
<th>( \tau ) is</th>
<th>( \tau ) is</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>set of ( \tau' )</td>
<td>mset of ( \tau' )</td>
<td>partition from ( \tau' )</td>
<td>rel of ( (\tau_1 \times \cdots \times \tau_n) )</td>
<td>function ( \tau_1 \to \tau_2 )</td>
</tr>
<tr>
<td>defined(e:( \tau ))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>set of ( \tau_1 )</td>
</tr>
<tr>
<td>range(e:( \tau ))</td>
<td></td>
<td></td>
<td></td>
<td>set of ( \tau_2 )</td>
<td></td>
</tr>
<tr>
<td>e:( \tau(e_1:\tau_1) )</td>
<td></td>
<td></td>
<td>( \tau_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>image(e:( \tau ), e_1:( \tau_1 ))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>image(e:( \tau ), e_1:set of ( \tau_1 ))</td>
<td></td>
<td>set of ( \tau_2 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>preimage(e:( \tau ), e_2:( \tau_2 ))</td>
<td></td>
<td></td>
<td>set of ( \tau_1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inverse(e:( \tau ), e':function ( \tau_2 \to \tau_1 ))</td>
<td></td>
<td></td>
<td></td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>e:( \tau(e_1:\tau_1, \ldots, e_n:\tau_n) )</td>
<td></td>
<td></td>
<td>bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e:( \tau(\ldots, _, \ldots) )</td>
<td></td>
<td></td>
<td>rel of ( (\tau_{i_1}, \ldots, \tau_{i_k}) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>together(e_1:( \tau'_1 ), e_2:( \tau'_2 ), e:( \tau ))</td>
<td></td>
<td></td>
<td></td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>apart(e_1:( \tau'_1 ), e_2:( \tau'_2 ), e:( \tau ))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>party(e':( \tau' ), e:( \tau ))</td>
<td></td>
<td></td>
<td>set of ( \tau' )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>participants(e:( \tau ))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>set of ( \tau' )</td>
</tr>
<tr>
<td>parts(e:( \tau ))</td>
<td></td>
<td></td>
<td>set of ( \tau' )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>freq(e:( \tau ), e':( \tau' ))</td>
<td></td>
<td></td>
<td></td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>hist(e:( \tau ), e':matrix indexed by [( \omega ) of ( \tau' )])</td>
<td></td>
<td></td>
<td>matrix indexed by [( \omega ) of int]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abstraction in ESSENCE
by providing several powerful features that enable abstraction.

At the time ESSENCE was designed, these were unique or rare among constraint languages.
A Wide Range of Types

given \( nrings, \) \( nnodes, \) \textit{capacity}: int (1..)

letting \( \text{Nodes} \) be domain int (1..\( nnodes \))

given set of set (size 2) of \( \text{Nodes} \)

find \( \text{network}: \text{mset} \) (size \( nrings \)) of \( \text{set} \) (maxSize \( \text{capacity} \)) of \( \text{Nodes} \)

maximising \( \sum \text{ring} \in \text{network}. \text{|ring|} \)

such that \( \forall \text{pair} \in \text{demand}. \exists \text{ring} \in \text{network}. \text{pair} \subseteq \text{ring} \)
Nested Types

given \( nrings, \ nnodes, \ capacity: \ int (1..) \)

letting \( Nodes \) be domain \( int (1..\nnodes) \)

given \( \) set of set (size 2) of \( Nodes \)

find \( network: \textbf{mset} \) (size \( nrings \)) of \( \textbf{set} \) (maxSize \( capacity \)) of \( Nodes \)

maximising \( \sum \ ring \in \ network. \ |ring| \)

such that \( \forall \ pair \in \text{demand}. \ \exists \ ring \in \ network. \ pair \subseteq ring \)
given \( w, g, s \)

letting \( \text{golfers} \) be new type of size \( g*s \)

find \( \text{sched}: \text{set (size } w \text{)} \) of partition (numParts \( g \), partSize \( s \)) from \( \text{golfers} \)

such that \( \forall \text{week}_1, \text{week}_2 \in \text{sched}. \ \text{week}_1 \neq \text{week}_2 \rightarrow \)

\( \forall \text{group}_1 \in \text{parts(week}_1\text{)}, \text{group}_2 \in \text{parts(week}_2\text{)}. \)

\( | \text{group}_1 \cap \text{group}_2 | < 2 \)
Quantification over Variables

given \( n_{rings}, n_{nodes}, capacity: \text{int (1..)} \)

letting \( \text{Nodes} \) be domain int (1..\( n_{nodes} \))

given set of set (size 2) of \( \text{Nodes} \)

find \( \text{network}: \text{mset (size } n_{rings} \text{) of set (maxSize } capacity \text{) of } \text{Nodes} \)

maximising \( \sum \text{ring} \in \text{network}. |\text{ring}| \)

such that \( \forall \text{pair} \in \text{demand}. \exists \text{ring} \in \text{network}. \text{pair} \subseteq \text{ring} \)
Analysis, Reflection, Evaluation
Analysis: Expressiveness of ESSENCE

- Using descriptive complexity theory, Mitchell and Ternovska [2008]
  - Prove: simple, first-order fragment captures NP
  - Prove: adding nested types leads to poly-time hierarchy
  - Prove: adding succinct domains leads to NEXP-time
  - Conjecture: complexity of any problem specified in ESSENCE is $\text{NTIME}[n \text{ raised to the power } n, k \text{ times}]$ for some $k$. 
Reflection on Design Process

• Usual order of design
  - Implementation
  - Design language
  - Write problem specifications

• We worked in the reverse order!
Evaluation: Naturalness

• Specifications of 70+ problems found in the CSP literature written by two computer science undergraduates with no background in constraint programming.

• http://www.cs.york.ac.uk/aig/constraints/AutoModel/Essence/specs120/

• paper contains comparisons to other languages
Evaluation: Sufficient Abstraction

• Insufficient facilities for abstraction can force the introduction of unnecessary objects or distinctions, which introduces symmetry into the specification.

• 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
Thank you!

Further information on ESSENCE and CONJURE

www.cs.york.ac.uk/aig/constraints/AutoModel