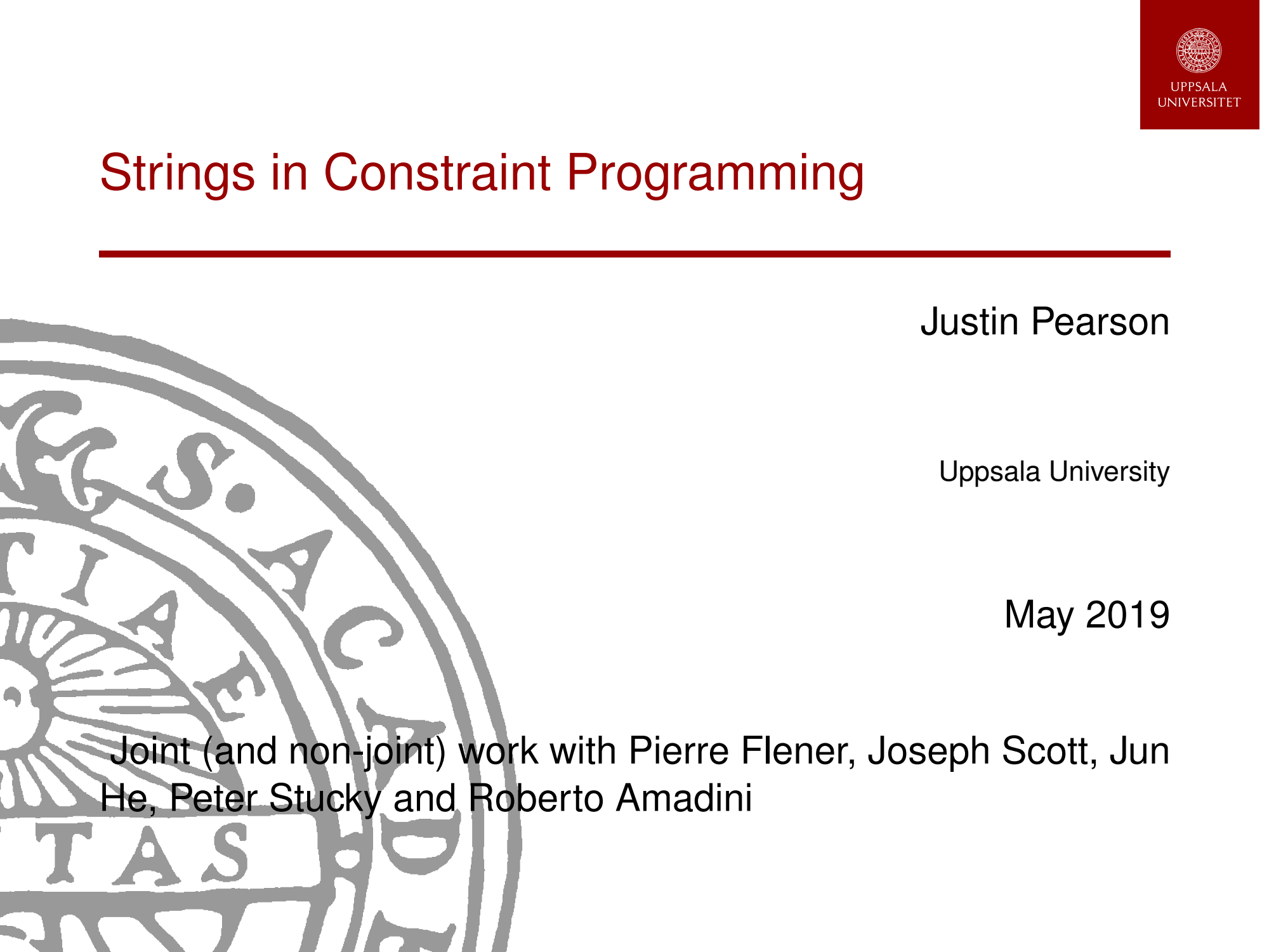

UPPSALA UNIVERSITET

Strings in Constraint Programming

Justin Pearson

Uppsala University

May 2019

Joint (and non-joint) work with Pierre Flener, Joseph Scott, Jun He, Peter Stucky and Roberto Amadini

What do I mean by Constraint Programming(CP)

Apparently we all do constraint solving, but what I mean :-

- Finite Domains
- Intelligent backtracking by assigning domains and propagating the consequences (although there is recent work incorporates clause learning).

CP has been around since the 70s, took off in the 90s with practical and scalable systems: IBM CP Optimiser, Gecode, Chuffed, Google-OR tools plus the MiniZinc[1] tool chain.

[1] `https://www.minizinc.org/`

Constraint Programming in a Nutshell

UPPSALA UNIVERSITET

Slogan of CP

Constraint Program = Model [+ Search]

CP provides:

- high level declarative modelling abstractions,
- a framework to separate search from from modelling.

We often spend a lot of time thinking about search procedures.

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

Example (Sudoku model)

```
1  array[1..9,1..9] of var 1..9: Sudoku;
2  ...
3  solve satisfy;
4  forall(r in 1..9)
5     (ALLDIFFERENT([Sudoku[r,c] | c in 1..9]));
6  forall(c in 1..9)
7     (ALLDIFFERENT([Sudoku[r,c] | r in 1..9]));
8  forall(i,j in {1,4,7})
9     (ALLDIFFERENT([Sudoku[r,c] | r in i..i+2, c in j..j+2]));
```

Global Constraints

UPPSALA UNIVERSITET

Global constraints such as AllDifferent and Sum enable the preservation of combinatorial sub-structures of a constraint problem, both while modelling it and while solving it. Many *n*-ary constraints (Catalogue) have been identified and encapsulate complex propagation algorithms declaratively., including

- n-ary linear and non-linear arithmetic
- Rostering under balancing & coverage constraints
- Scheduling under resource & precedence constraints
- Geometrical constraints between points, segments, ...

There are many more.

CP Solving = Search + Propagation

UPPSALA UNIVERSITET

A CP solver conducts search interleaved with propagation:

- Familiar idea as in SAT solvers.
  - Propagate until fix point.
  - Make a choice.
  - Backtrack on failure.
- Because we have global constraints we can often do more propagation than unit-propagation of clauses at each step.

The AllDifferent constraint

Consider the n-ary constraint AllDifferent, with $n = 4$:

$$\text{AllDifferent}([a, b, c, d]) \tag{1}$$

The AllDifferent constraint

Consider the n-ary constraint AllDifferent, with $n = 4$:

$$\text{AllDifferent}([a, b, c, d]) \tag{1}$$

Modelling: (1) is equivalent to $\frac{n(n-1)}{2}$ binary constraints:

$$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d \tag{2}$$

The AllDifferent constraint

Consider the n-ary constraint AllDifferent, with $n = 4$:

$$\text{AllDifferent}([a, b, c, d]) \tag{1}$$

Modelling: (1) is equivalent to $\frac{n(n-1)}{2}$ binary constraints:

$$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d \tag{2}$$

Inference: (1) propagates much better than (2). Example:

$$a \in \{4, 5\},\ b \in \{4, 5\},\ c \in \{3, 4\},\ d \in \{1, 2, 3, 4, 5\}$$

No domain pruning by (2).

The AllDifferent constraint

Consider the n-ary constraint AllDifferent, with $n = 4$:

$$\text{AllDifferent}([a, b, c, d]) \tag{1}$$

Modelling: (1) is equivalent to $\frac{n(n-1)}{2}$ binary constraints:

$$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d \tag{2}$$

Inference: (1) propagates much better than (2). Example:

$$a \in \{4, 5\},\ b \in \{4, 5\},\ c \in \{3, \cancel{4}\},\ d \in \{1, 2, \cancel{3}, \cancel{4}, \cancel{5}\}$$

No domain pruning by (2). But perfect propagation by (1)

Bounded-length Strings in CP

bounded-length sequence representation

A b-length sequence over-approximates a set of strings of length $\leq b$.

$$\langle\langle \mathcal{A}[1], \ldots, \mathcal{A}[b]\rangle, \mathcal{N}\rangle$$

Each $\mathcal{A}[i]$ is a set of characters, which can become empty and $\mathcal{N}$ in interval giving the lower and upper bound of the string length.
That the implementation comes with some invariants relating the length and the non-emptyness of sets. With a clever implementation you can generate the sets $\mathcal{A}[i]$ on the fly.

Bounded-length Strings

A bounded length is a string of a (possibly)-unknown that is bounded from above by some implementation specific constant. Possible implementations

- Decompose are arrays of characters with a length variable and a padding character (padded).
- Implement special propagators to work with the padding approach approach (aggregate)
- Implement a bespoke variable type inside a constraint solver (native).

We need padding characters because when a domain becomes empty a CP solver will fail at that node and backtrack.

New data-types in CP

UPPSALA UNIVERSITET

Implement a new datatype as a first class citizen in the constraint solver. A classic example is set variables.

- Choice of representation.
- How to interact with the propagation loop
  - Changes in domains are signalled by events that form a lattice.
  - A propagator subscribes to events to control how much information and how often the propagator is woken up.
- What exactly should we propagate?
  - A representation is an approximation of the mathematical reality.
  - We have a galois-based framework for specifying propagators and deriving what propagation should and can be done in different representations.

String Constraints

Some of the constraints that we have considered, s_j are string variables, c_j are character variables and i_j are integer variables.

- EQUAL(s_1, s_2) if s_1 and s_2 are equal, that is $s_1 = s_2$
- REVERSE(s_1, s_2) if $s_1 = c_1 c_2 \cdots c_n$ and $s_2 = c_n \cdots c_2 c_1$
- CONCAT(s_1, s_2, s) if $s_1 \oplus s_2 = s$, with concatenation $\oplus$
- SUBSTRING(s_1, i_1, i_2, s) if $s_1[i_1 : i_2] = s$
- CHARACTERAT(s, i, c) if SUBSTRING$(s, i, i, \text{“}c\text{”})$
- LENGTH(s, i) if s has i characters, that is $|s| = i$
- REGULAR$(s, \mathcal{R})$ if s is a word of a regular language $\mathcal{R}$, given by a regular expression or a finite automaton
- CONTEXTFREE$(s, \mathcal{F})$ if s is a word of a context-free language $\mathcal{F}$, given by a context-free grammar
- COUNT$(s, [c_1, ..., c_n], [i_1, ..., i_n])$ if in s all c_j occur i_j times

Constraints and Decision Variables

Instead of communicating theories we communicate via propagation.

- $\textsc{Length}(s, l) \wedge l + m = c$

Propagation on l,m,c will propagate information to the length of s as well as the other direction.

Native Strings

Three tightly related choices:

- data structure
  - candidate lengths $\subset \mathbb{N}$: range sequence, bitset, interval
  - candidate characters $\subset \mathbb{N}$: range sequence, bitset, interval
  - sequence: array, list, list of arrays, etc
- restriction operations must consider undefinedness
  - work by removing values from components
  - result is to remove strings from the domain
- propagation events
  - representation invariant: many promising looking event systems are not monotonic

Native Strings

UPPSALA UNIVERSITET

Three tightly related choices:

- data structure
  - candidate lengths $\subset \mathbb{N}$: range sequence, bitset, interval
  - candidate characters $\subset \mathbb{N}$: range sequence, bitset, interval
  - sequence: array, list, list of arrays, etc
- restriction operations must consider undefinedness
  - work by removing values from components
  - result is to remove strings from the domain
- propagation events
  - representation invariant: many promising looking event systems are not monotonic

Another Approach — Dashed Strings

- A dashed string (**D.S.**) is a concatenation $S_1^{l_1,u_1} S_2^{l_2,u_2} \cdots S_k^{l_k,u_k}$ of **blocks** $S_i^{l_i,u_i}$ such that:

$$0 < k \leq b \qquad S_i \subseteq \Sigma \qquad 0 \leq l_i \leq u_i \leq b \qquad \Sigma_{i=1}^{k} l_i \leq b$$

- Each block $S_i^{l_i,u_i}$ represents the set of strings of S_i^* having length in $[l_i, u_i]$.
- Graphical interpretation: continuous segments of length l_i are the **mandatory** part (characters that *must* appear), dashed segments of length $u_i - l_i$ are the **optional** part (characters that *may* appear)
  - e.g., graphical representation of D.S. $\{\texttt{B,b}\}^{1,1}\{\texttt{o}\}^{2,4}\{\texttt{m}\}^{1,1}\{\texttt{!}\}^{0,3}$

Conclusions

- Lots of experiments, we are competitive. Implementations exists, but they are not exactly off the shelf at the moment.
- Dashed strings often work better as more information can be propagated about the length, but this makes the propagators more complicated.

Thank you

- Questions?

References

J. Scott. Other Things Besides Number: Abstraction, Constraint Propagation, and String Variable Types. PhD thesis, Department of Information Technology, Uppsala University, Sweden, March 2016.
`http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-273311`

Roberto Amadini, Pierre Flener, Justin Pearson, Joseph D. Scott, Peter J. Stuckey, Guido Tack: MiniZinc with Strings. LOPSTR 2016: 59-75

UPPSALA
UNIVERSITET

Roberto Amadini, Graeme Gange, Peter J. Stuckey: Sweep-Based Propagation for String Constraint Solving. AAAI 2018: 6557-6564

Roberto Amadini, Graeme Gange, Peter J. Stuckey: Propagating Regular Membership with Dashed Strings. CP 2018: 13-29