Increasing modeling language convenience with a universal n-dimensional array, CPpy as python-embedded example

Tias Guns, VUB
tias.guns@vub.be

Modref 2019

http://github.com/tias/cppy

What is the purpose of a modeling language?

Lessons learned from Mini**ng**Zinc

Where does the data come from?

```
array [int] of set of int : TDB
        :: query("mydb.sql", "SELECT tid,item FROM purchases");
```

This requires changes to the mzn compiler.

And how to do user-level preprocessing or feature construction?
→ in yet another language...

Modeling: freq. itemset mining with cost

library with itemset mining specific functions and predicates

```
include "lib_itemsetmining.mzn"

int: NrI; int: NrT; int: MinFreq;
array[1..NrT] of set of int: TDB;

var set of 1..NrI: Items;

constraint card(cover(Items, TDB)) >= MinFreq;

array [1..NrI] of int: Cost;
int: MinCost;

constraint sum(i in Items) (Cost[i]) >= MinCost

solve satisfy;
```

Primitive CP usage, 1/2

"Pyconstruct: CP Meets Structured Prediction"
Dragone et al, IJCAI19 demo

- SVM that repeatedly calls CP
- Implemented their own 'mznpy' library
- added a text-based *templating* language over minizinc

Feels like a step back to PHP3 to me...

Weak integration: write strings in python, that are written to file that are sent to mzn command line...

```
{% from 'globals.pmzn' import domain, solve %}
{% from 'linear.pmzn' import linear_model %}
{% from 'chain.pmzn' import
    n_emission_features, emission_features,
    n_transition_features, transition_features
%}
{% from 'metrics.pmzn' import hamming %}

int: MAX_HEIGHT = 9;
int: MAX_WIDTH = 9;
set of int: HEIGHT = 1 .. MAX_HEIGHT;
set of int: WIDTH = 1 .. MAX_WIDTH;

% Set of symbols (labels). Digits are encoded as themselves.
% Assume '+' and '=' are encoded respectively with 10 and 11.
int: PLUS = 10;
int: EQUAL = 11;
int: N_SYMBOLS = 12;
set of int: SYMBOLS = 0 .. N_SYMBOLS - 1;

% Constants
int: N_PIXELS = MAX_HEIGHT * MAX_WIDTH;
set of int: PIXELS = 1 .. N_PIXELS;

{% call domain(problem) %}

    % Input: Length of the sequence and images
    int: length;
    set of int: SEQUENCE = 1 .. length;
    array[SEQUENCE, HEIGHT, WIDTH] of {0, 1}: images;

    % Output: Sequence of symbols
    array[SEQUENCE] of var SYMBOLS: sequence;

    {% if problem == 'loss_aumented_map' %}
        array[SEQUENCE] of int: true_sequence = {{ y_true['sequence']|dzn }};
    {% endif %}

    array[SEQUENCE, PIXELS] of {0, 1}: pixels = array2d(SEQUENCE, PIXELS, [
        images[s, i, j] | s in SEQUENCE, i in HEIGHT, j in WIDTH
    ]);

{% endcall %}
```

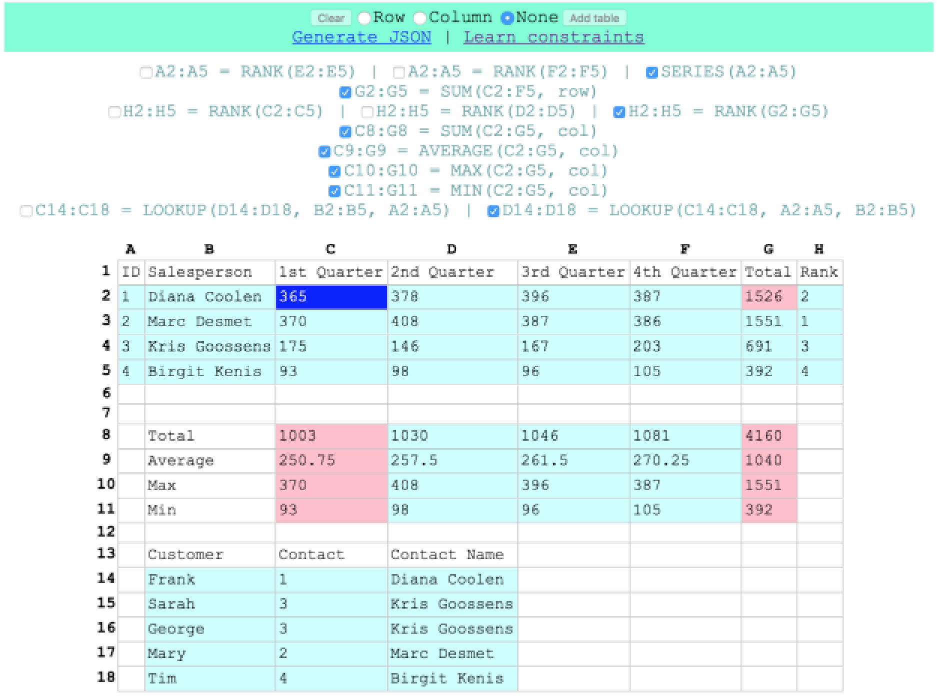
Primitive CP usage, 2/2

“Tacle: learning constraints in tabular data” Kolb et al, 2017

- learns formula's in sheets
- uses CP for efficient candidate generation
- uses 'python-constraints'

a 1200 sloc forward checker...

but native python, trivial integration



	A	B	C	D	E	F	G	H
1	ID	Salesperson	1st Quarter	2nd Quarter	3rd Quarter	4th Quarter	Total	Rank
2	1	Diana Coolen	365	378	396	387	1526	2
3	2	Marc Desmet	370	408	387	386	1551	1
4	3	Kris Goossens	175	146	167	203	691	3
5	4	Birgit Kenis	93	98	96	105	392	4
6								
7								
8		Total	1003	1030	1046	1081	4160	
9		Average	250.75	257.5	261.5	270.25	1040	
10		Max	370	408	396	387	1551	
11		Min	93	98	96	105	392	
12								
13		Customer	Contact	Contact Name				
14		Frank	1	Diana Coolen				
15		Sarah	3	Kris Goossens				
16		George	3	Kris Goossens				
17		Mary	2	Marc Desmet				
18		Tim	4	Birgit Kenis				

Popular data-driven (AI) frameworks:

- scikit-learn (and pandas) :: ML
- pytorch :: deep learning
- cvxpy :: convex optimisation

Popular data-driven (AI) frameworks:

- scikit-learn (and pandas) :: ML
- pytorch :: deep learning
- cvxpy :: convex optimisation

- Why?
  - ease of use and documentation: quick start
  - ease of integration with existing code
  - solid technology underneath
- What do they have in common?
  - Python-based library
  - Numpy's ndarray as basic data structure
  - Operator overloading and as convenient as the standard library

Example: CVXpy

Stephen Boyd's framework 'disciplined convex programm'

Can you spot the difference between the use of built-in functions, numpy functions and cvx functions?

I wish this existed for CP!

```python
import cvxpy as cp
import numpy as np

# Problem data.
m = 30
n = 20
np.random.seed(1)
A = np.random.randn(m, n)
b = np.random.randn(m)

# Construct the problem.
x = cp.Variable(n)
objective = cp.Minimize(cp.sum_squares(A*x - b))
constraints = [0 <= x, x <= 1]
prob = cp.Problem(objective, constraints)

# The optimal objective value is returned by `prob.solve()`.
result = prob.solve()
# The optimal value for x is stored in `x.value`.
print(x.value)
# The optimal Lagrange multiplier for a constraint is stored in
# `constraint.dual_value`.
print(constraints[0].dual_value)
```

Purpose of modeling language?

- Convenience
- High-level abstractions
- Possible to reuse/extend the backend

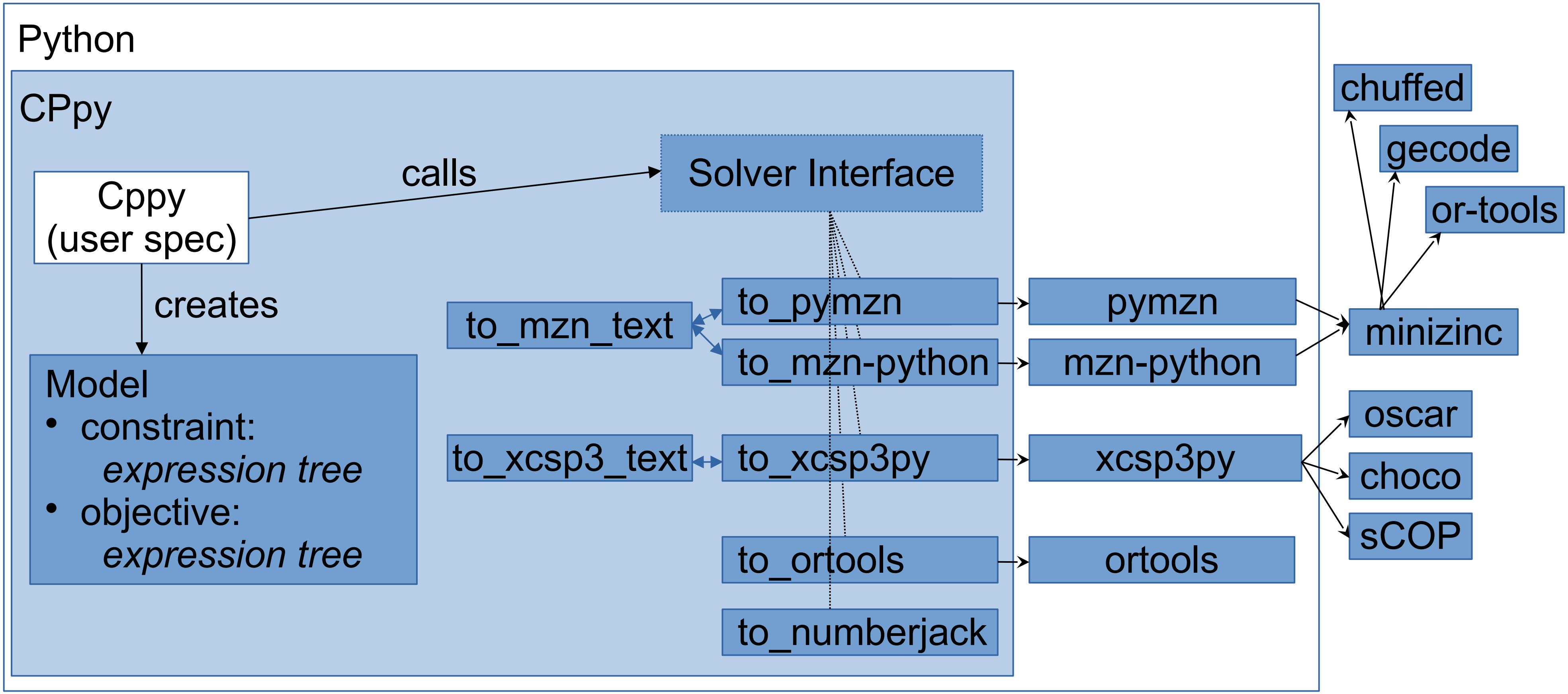
CPpy design principles

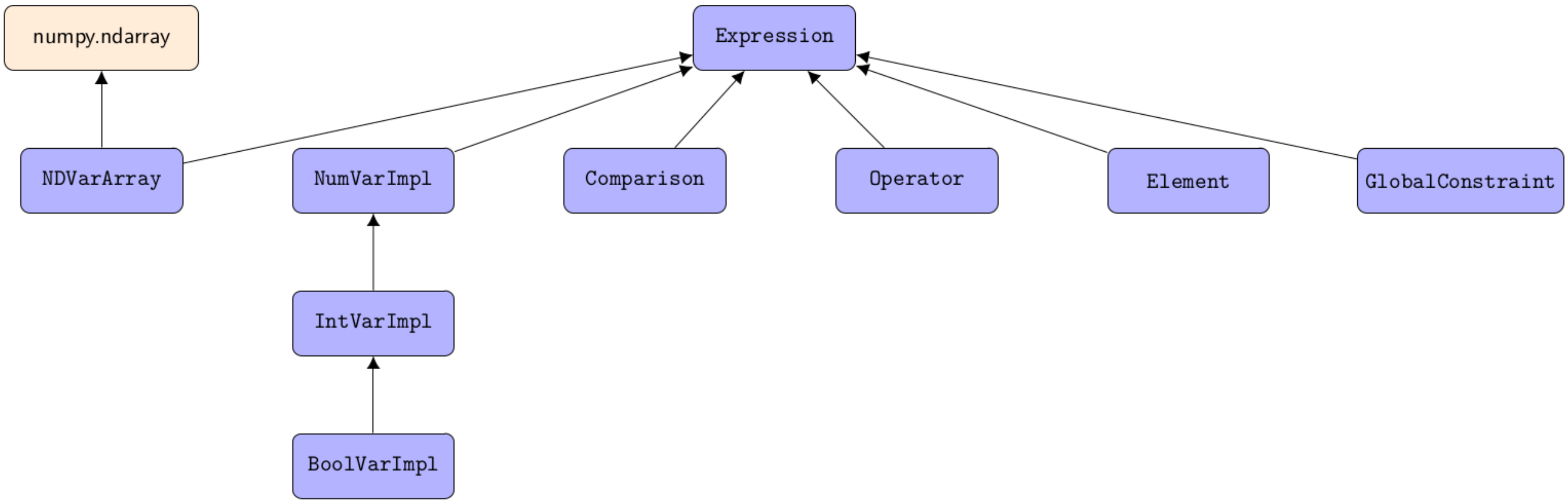
1) solver independent
2) n-dimensional array as basic datastructure (Numpy's)
3) operator overloading, few custom constructs
4) light-weight abstract syntax tree: no logic inside
5) variable objects give direct access to the solution

```python
from cppy import *
import numpy as np

# Construct the model
s,e,n,d,m,o,r,y = IntVar(0,9, 8)

constraint = []
constraint += [ alldifferent([s,e,n,d,m,o,r,y]) ]
constraint += [     sum(    [s,e,n,d] * np.flip(10**np.arange(4)) )
                  + sum(    [m,o,r,e] * np.flip(10**np.arange(4)) )
                 == sum(  [m,o,n,e,y] * np.flip(10**np.arange(5)) ) ]
constraint += [ s > 0, m > 0 ]

model = Model(constraint)
stats = model.solve()
print("  S,E,N,D =  ", [x.value for x in [s,e,n,d]])
print("  M,O,R,E =  ", [x.value for x in [m,o,r,e]])
print("M,O,N,E,Y =", [x.value for x in [m,o,n,e,y]])
```

```python
x = 0 # cells whose value we seek
n = 9 # matrix size
given = numpy.array([
    [x, x, x,  2, x, 5,  x, x, x],
    [x, 9, x,  x, x, x,  7, 3, x],
    [x, x, 2,  x, x, 9,  x, 6, x],

    [2, x, x,  x, x, x,  4, x, 9],
    [x, x, x,  x, 7, x,  x, x, x],
    [6, x, 9,  x, x, x,  x, x, 1],

    [x, 8, x,  4, x, x,  1, x, x],
    [x, 6, 3,  x, x, x,  x, 8, x],
    [x, x, x,  6, x, 8,  x, x, x]])


# Variables
puzzle = IntVar(1, n, shape=given.shape)

constraint = []
# constraints on rows and columns
constraint += [ alldifferent(row) for row in puzzle ]
constraint += [ alldifferent(col) for col in puzzle.T ]

# constraint on blocks
for i in range(0,n,3):
    for j in range(0,n,3):
        constraint += [ alldifferent(puzzle[i:i+3, j:j+3]) ]

# constraints on values
constraint += [ puzzle[given>0] == given[given>0] ]

model = Model(constraint)
stats = model.solve()
```

Toolchain (not fully implemented)





- Minimal but meaningful class diagram
  - automatically constructed through operator overloading
  - example: X + Y → Operator("sum",[X,Y])
- Goal: easy to add rewrite rules in backend
  → foster more research and use of *modref* principles!

Discussion (last slide)

- You just propose syntax you are used too, and all syntax takes getting used too
- This is just NumberJack, and that has not taken off (already has matrix variable)

- I believe the purpose of modeling languages instead is …
- Text-based languages in a programming language are a hack (e.g. minizinc-python)
- Is CPpy a modeling language or not?

- Our current modeling languages are modern enough already
- Nobody wants to add own rewrite rules or change the back-end,
  we should aim for push-button software