Towards Solver-Independent Propagators\footnote{supported by grant 2011-6133 of VR, the Swedish Research Council}

Jean-Noël Monette, Pierre Flener, and Justin Pearson

ASTRA Research Group on Constraint Programming
Department of Information Technology
Uppsala University
Sweden

http://www.it.uu.se/research/group/astra

Journées Francophones de Programmation par Contraintes
2013
Propagators

- Propagator tailored especially for a global constraint.
- Tedious to make it correct, efficient, compliant with the solver interface, in various programming languages.

Aim: Solver-independent language to describe propagators.
- Ease the implementation and sharing of propagators.
- Ease the proof of propagator properties.
Indexicals

X in min(Y)+min(Z)..max(Y)+max(Z);

- An indexical defines a restriction on the domain of a decision variable
- Used e.g. in SICStus Prolog.
- Deal with constraints of fixed arity.

Our contribution: deal with constraints of non-fixed arity (i.e., global constraints).
The PLUS Constraint

```python
1 def PLUS(vint X, vint Y, vint Z){
2     propagator(DR){
3         X in dom(Y)+dom(Z);
4         Y in dom(X)-dom(Z);
5         Z in dom(X)-dom(Y);
6     }
7     propagator(BR){
8         X in (min(Y)+min(Z)) .. (max(Y)+max(Z)) ;
9         Y in (min(X)-max(Z)) .. (max(X)-min(Z)) ;
10        Z in (min(X)-max(Y)) .. (max(X)-min(Y)) ;
11     }
12     propagator(VR){
13         X in {val(Y)+val(Z)};
14         Y in {val(X)-val(Z)};
15         Z in {val(X)-val(Y)};
16     }
17     checker{ val(X) == val(Y) + val(Z) }
18 }
```
The PLUS Constraint

```python
1 def PLUS(vint X, vint Y, vint Z){
2     propagator(DR){
3         X in dom(Y)+dom(Z);
4         Y in dom(X)-dom(Z);
5         Z in dom(X)-dom(Y);
6     }
7     propagator(BR){
8         X in (min(Y)+min(Z)) .. (max(Y)+max(Z)) ;
9         Y in (min(X)-max(Z)) .. (max(X)-min(Z)) ;
10        Z in (min(X)-max(Y)) .. (max(X)-min(Y)) ;
11     }
12     propagator(VR){
13         X in {val(Y)+val(Z)};
14         Y in {val(X)-val(Z)};
15         Z in {val(X)-val(Y)};
16     }
17     checker{ val(X) == val(Y) + val(Z) }
18 }```

JFPC 2013
The PLUS Constraint

```python
1 def PLUS(vint X, vint Y, vint Z){
2     propagator(DR){
3         X in dom(Y)+dom(Z);
4         Y in dom(X)-dom(Z);
5         Z in dom(X)-dom(Y);
6     }
7     propagator(BR){
8         X in (min(Y)+min(Z)) .. (max(Y)+max(Z)) ;
9         Y in (min(X)-max(Z)) .. (max(X)-min(Z)) ;
10        Z in (min(X)-max(Y)) .. (max(X)-min(Y)) ;
11     }
12     propagator(VR){
13         X in {val(Y)+val(Z)};
14         Y in {val(X)-val(Z)};
15         Z in {val(X)-val(Y)};
16     }
17     checker{ val(X) == val(Y) + val(Z) }
18 }
```
The PLUS Constraint

```c
1 def PLUS(vint X, vint Y, vint Z) {
2     propagator(DR) {
3         X in dom(Y) + dom(Z);
4         Y in dom(X) - dom(Z);
5         Z in dom(X) - dom(Y);
6     }
7     propagator(BR) {
8         X in (min(Y) + min(Z)) .. (max(Y) + max(Z)) ;
9         Y in (min(X) - max(Z)) .. (max(X) - min(Z)) ;
10        Z in (min(X) - max(Y)) .. (max(X) - min(Y)) ;
11    }
12     propagator(VR) {
13         X in {val(Y) + val(Z)} ;
14         Y in {val(X) - val(Z)} ;
15         Z in {val(X) - val(Y)} ;
16    }
17     checker{ val(X) == val(Y) + val(Z) }
18 }
```
The PLUS Constraint

```python
def PLUS(vint X, vint Y, vint Z):
    propagator(DR) {
        X in dom(Y)+dom(Z);  
        Y in dom(X)-dom(Z);  
        Z in dom(X)-dom(Y);
    }
    propagator(BR) {
        X in (min(Y)+min(Z)) .. (max(Y)+max(Z));  
        Y in (min(X)-max(Z)) .. (max(X)-min(Z));  
        Z in (min(X)-max(Y)) .. (max(X)-min(Y));
    }
    propagator(VR) {
        X in {val(Y)+val(Z)};
        Y in {val(X)-val(Z)};
        Z in {val(X)-val(Y)};
    }
    checker{ val(X) == val(Y) + val(Z) }
```
The SUM Global Constraint

1 def SUM(vint[] X, vint N){
2     propagator(v1){
3         N in sum(i in rng(X))(dom(X[i]));
4         forall(i in rng(X))
5             X[i] in dom(N) - sum(j in rng(X):j!=i)(dom(X[j]));
6     }
7     propagator(v2){
8         N in sum(i in rng(X))(min(X[i])) .. sum(i in rng(X))(max(X[i]));
9         forall(i in rng(X))
10            X[i] in min(N) - sum(j in rng(X):j!=i)(max(X[j])) ..
11                max(N) - sum(j in rng(X):j!=i)(min(X[j]));
12     }
13     checker{ val(N) = sum(i in rng(X))(val(X[i])) }
14 }
The SUM Global Constraint

```python
1 def SUM(vint[] X, vint N){
2    propagator(v1){
3        N in sum(i in rng(X))(dom(X[i]));
4        forall(i in rng(X))
5            X[i] in dom(N) - sum(j in rng(X):j!=i)(dom(X[j]));
6    }
7    propagator(v2){
8        N in sum(i in rng(X))(min(X[i])) ..
9            sum(i in rng(X))(max(X[i]));
10       forall(i in rng(X))
11           X[i] in min(N) - sum(j in rng(X):j!=i)(max(X[j])) ..
12              max(N) - sum(j in rng(X):j!=i)(min(X[j]));
13    }
14    checker{ val(N) = sum(i in rng(X))(val(X[i])) }
15 }
```
The SUM Global Constraint

1 def SUM(vint[] X, vint N){
2   propagator(v1){
3       N in sum(i in rng(X))(dom(X[i]));
4       forall(i in rng(X))
5           X[i] in dom(N) - sum(j in rng(X): j!=i)(dom(X[j]));
6   }
7   propagator(v2){
8       N in sum(i in rng(X))(min(X[i])) ..
9           sum(i in rng(X))(max(X[i]));
10      forall(i in rng(X))
11          X[i] in min(N) - sum(j in rng(X): j!=i)(max(X[j])) ..
12             max(N) - sum(j in rng(X): j!=i)(min(X[j]));
13    }
14   checker{ val(N) = sum(i in rng(X))(val(X[i])) }
The EXACTLY Global Constraint

```python
1  def EXACTLY(vint[] X, vint N, int v){
2      propagator{
3          N in sum(i in rng(X))(b2i(entailed(EQ(X[i], v)))) ..
4              sum(i in rng(X))(b2i(satisfiable(EQ(X[i], v)))));
5          forall(i in rng(X)){
6              once(val(N) <=
7                  sum(j in rng(X):i!=j)(b2i(entailed(EQ(X[j], v))))){
8                  post(NEQ(X[i], v));
9              }
10             once(val(N) >
11                 sum(j in rng(X):i!=j)(b2i(satisfiable(EQ(X[j],v))))){
12                 post(EQ(X[i], v));
13             }
14          }
15      }
16     checker{ val(N) = sum(i in rng(X))(b2i(val(X[i]) == v)) }
17  }
```
The EXACTLY Global Constraint

```python
1  def EXACTLY(vint[] X, vint N, int v){
2      propagator{
3          N in sum(i in rng(X))(b2i(entailed(EQ(X[i], v)))) ..
4              sum(i in rng(X))(b2i(satisfiable(EQ(X[i], v)))));
5          forall(i in rng(X)){
6              once(val(N) <=
7                  sum(j in rng(X):i!=j)(b2i(entailed(EQ(X[j], v))))){
8                  post(NEQ(X[i], v));
9              }
10             once(val(N) >
11                sum(j in rng(X):i!=j)(b2i(satisfiable(EQ(X[j],v))))){
12                  post(EQ(X[i], v));
13             }
14         }
15     }
16  checker{ val(N) = sum(i in rng(X))(b2i(val(X[i]) == v)) }
17 }
```
def EXACTLY(vint[] X, vint N, int v){
    propagator{
        N in sum(i in rng(X)) (b2i(entailed(EQ(X[i], v)))) ..
        sum(i in rng(X)) (b2i(satisfiable(EQ(X[i], v))))
        forall(i in rng(X)){
            once(val(N) <=
            sum(j in rng(X):i!=j)(b2i(entailed(EQ(X[j], v))))){
                post(NEQ(X[i], v));
            }
            once(val(N) >
            sum(j in rng(X):i!=j)(b2i(satisfiable(EQ(X[j], v))))){
                post(EQ(X[i], v));
            }
        }
    }
    checker{ val(N) = sum(i in rng(X))(b2i(val(X[i]) == v)) }
Language Design Decisions

- Based on indexicals: close to the human reasoning.
- Stateless: cannot describe e.g. DC ALLDIFFERENT.
- Strongly typed: int, vint, bool, set, cstr
- Introduces arrays and n-ary operators.
- Meta-constraints, constraint invocation, and local variables.
- Limited number of new constructs, for simplicity of use.
- No solver-specific hooks: only domain access and narrowing.
Desired Properties of a Propagator

- Correct
- Checking, singleton correctness, and singleton completeness
- Contracting
- Monotonic
- Domain consistent or other consistency level
- Idempotent
- Low time (and space) complexity
- Avoid useless execution:
  - Entailment detection
  - Subscription to relevant events
Most properties are difficult to prove for a given propagator but...

- Indexicals satisfy contraction by definition.
- Possible to check the monotonicity [Carlson et al, ICLP’94]
- Correctness and checking can sometimes be proven.

In addition to analysis, we can also make algorithmic transformations:

- Changing the level of reasoning (e.g., use bounds instead of the whole domain).
- Grounding some decision variables.
Compiler

- Compilation, not interpretation.
- Written in Java.
- Backends for Comet, Gecode (and MiniZinc Binding), OscaR, and JaCoP.
- Compiled propagators are stateless.
- Compiled propagators use coarse-grained wake-up events.
- Some code optimisations are nevertheless performed.
  - Dynamic programming pre-computation of arrays (typically gets linear complexity, instead of quadratic).
  - Factorisation of repeated expressions.
  - ...
Experimental Evaluation

Comparing time to solve problems with

- Indexical-based generated propagators
- Gecode built-in propagators
- Decompositions
- Regular constraints modeled as automata.
Setting

- Four contraints: **SUM, MAXIMUM, EXACTLY, and ELEMENT**.
- Gecode 3.7.3
- Consider each constraint in isolation.
- Search for all its solutions.
- Repeat with different search heuristics.
- 9 variables in arrays with 9 values in domains.
- Use the fact that constraints are total functions.
Results

Relative running times:

<table>
<thead>
<tr>
<th></th>
<th>Maximum</th>
<th>Sum</th>
<th>Exactly</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Indexicals</td>
<td>1.3</td>
<td>2.7</td>
<td>2.5</td>
<td>1.2</td>
</tr>
<tr>
<td>Decomposition</td>
<td>1.9</td>
<td>3.0</td>
<td>3.1</td>
<td>2.0</td>
</tr>
<tr>
<td>Automaton</td>
<td>6.7</td>
<td>n/a</td>
<td>n/a</td>
<td>4.9</td>
</tr>
</tbody>
</table>

Runtime increase with the number of variables:

- indexicals: linear
- Gecode built-ins: sub-linear, due to dynamic variable elimination.
Use Cases

- Development of a new solver.
- Eased maintenance and testing of a solver.
- Prototyping of propagators for new constraints (e.g. string constraints).
- Replace a decomposition by a propagator (e.g. $X = Y \Rightarrow Z = W$).
Conclusion

- A solver-independent language to describe propagators.
- Extends indexicals for global constraints.
- Eases the writing and sharing of propagators.
- Eases the proving of their properties.

Compiler available from http://user.it.uu.se/~jeamo371/indexicals/
First Workshop on Domain Specific Languages for Combinatorial Optimization

- In conjunction with CP 2013 (Sept. 16, Uppsala)
- Submission of paper or abstract before June 28
- [http://cp2013.a4cp.org/workshops/cospel](http://cp2013.a4cp.org/workshops/cospel)
Appendix
# List of implemented constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>AND</td>
</tr>
<tr>
<td>DIST</td>
<td>CHANGE</td>
</tr>
<tr>
<td>EQ</td>
<td>COUNT</td>
</tr>
<tr>
<td>GEQ</td>
<td>EXACTLY_IND</td>
</tr>
<tr>
<td>GT</td>
<td>EXACTLYSEQ</td>
</tr>
<tr>
<td>INSET</td>
<td>FIRST</td>
</tr>
<tr>
<td>LEQ</td>
<td>IMPLY</td>
</tr>
<tr>
<td>LT</td>
<td>IsTransition</td>
</tr>
<tr>
<td>MAX</td>
<td>ITH</td>
</tr>
<tr>
<td>NEQ</td>
<td>NOT</td>
</tr>
<tr>
<td>NOTINSET</td>
<td>NotTransition</td>
</tr>
<tr>
<td>PLUS</td>
<td>OR</td>
</tr>
<tr>
<td>PLUSLEQ</td>
<td>REIFY</td>
</tr>
<tr>
<td>Reif_EQ</td>
<td>SEQ_BIN</td>
</tr>
<tr>
<td>TIMES</td>
<td>Transition</td>
</tr>
<tr>
<td></td>
<td>AMONG</td>
</tr>
<tr>
<td></td>
<td>ELEMENT</td>
</tr>
<tr>
<td></td>
<td>EXACTLY</td>
</tr>
<tr>
<td></td>
<td>Global_Contiguity</td>
</tr>
<tr>
<td></td>
<td>INCR_NVALUE</td>
</tr>
<tr>
<td></td>
<td>INCREASING</td>
</tr>
<tr>
<td></td>
<td>Ith_POS_DIFF_ZERO</td>
</tr>
<tr>
<td></td>
<td>Lex_Less</td>
</tr>
<tr>
<td></td>
<td>Lex_Lesseq</td>
</tr>
<tr>
<td></td>
<td>MAXIMUM</td>
</tr>
<tr>
<td></td>
<td>NON_DECREASING</td>
</tr>
<tr>
<td></td>
<td>PLATEAU</td>
</tr>
<tr>
<td></td>
<td>SOME_EQ</td>
</tr>
<tr>
<td></td>
<td>STRICTLY_INCR_SEQ</td>
</tr>
<tr>
<td></td>
<td>SUM</td>
</tr>
</tbody>
</table>