Towards Solver-Independent Propagators[1]

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

CP 2012

[1] supported by grant 2011-6133 of VR, the Swedish Research Council

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

```
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) }
}
```

Introduction
Language and Properties
Compilation and Evaluation
Conclusion

The PLUS Constraint

```
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) }
}
```

Introduction
Language and Properties
Compilation and Evaluation
Conclusion

The PLUS Constraint

```
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 PLUS Constraint


```
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 PLUS Constraint

```
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

```
def SUM(vint[] X,vint N){
  propagator(v1){
    N in sum(i in rng(X))(dom(X[i]));
    forall(i in rng(X))
      X[i] in dom(N) - sum(j in {k in rng(X):k!=i})(dom(X[j]));
  }
  propagator(v2){
    N in sum(i in rng(X))(min(X[i])) ..
                     sum(i in rng(X))(max(X[i]));
    forall(i in rng(X))
      X[i] in min(N) - sum(j in {k in rng(X):k!=i})(max(X[j])) ..
                     max(N) - sum(j in {k in rng(X):k!=i})(min(X[j]));
  }
  checker{ val(N) = sum(i in rng(X))(val(X[i])) }
}
```

The SUM Global Constraint

```
def SUM(vint[] X,vint N){
  propagator(v1){
    N in sum(i in rng(X))(dom(X[i]));
    forall(i in rng(X))
      X[i] in dom(N) - sum(j in {k in rng(X):k!=i})(dom(X[j]));
  }
  propagator(v2){
    N in sum(i in rng(X))(min(X[i])) ..
                  sum(i in rng(X))(max(X[i]));
    forall(i in rng(X))
      X[i] in min(N) - sum(j in {k in rng(X):k!=i})(max(X[j])) ..
                  max(N) - sum(j in {k in rng(X):k!=i})(min(X[j]));
  }
  checker{ val(N) = sum(i in rng(X))(val(X[i])) }
}
```

The SUM Global Constraint

```
def SUM(vint[] X,vint N){
  propagator(v1){
    N in sum(i in rng(X))(dom(X[i]));
    forall(i in rng(X))
      X[i] in dom(N) - sum(j in {k in rng(X):k!=i})(dom(X[j]));
  }
  propagator(v2){
    N in sum(i in rng(X))(min(X[i])) ..
                  sum(i in rng(X))(max(X[i]));
    forall(i in rng(X))
      X[i] in min(N) - sum(j in {k in rng(X):k!=i})(max(X[j])) ..
                  max(N) - sum(j in {k in rng(X):k!=i})(min(X[j]));
  }
  checker{ val(N) = sum(i in rng(X))(val(X[i])) }
}
```

The EXACTLY Global Constraint

```
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 {j in rng(X):i!=j})(b2i(entailed(EQ(X[j], v))))){
          post(NEQ(X[i], v));
      }
      once(val(N) >
       sum(j in {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)) }
}
```

The EXACTLY Global Constraint

```
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 {j in rng(X):i!=j})(b2i(entailed(EQ(X[j], v))))){
          post(NEQ(X[i], v));
      }
      once(val(N) >
       sum(j in {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)) }
}
```

UPPSALA UNIVERSITET

The EXACTLY Global Constraint

```
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 {j in rng(X):i!=j})(b2i(entailed(EQ(X[j], v))))){
          post(NEQ(X[i], v));
      }
      once(val(N) >
       sum(j in {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

Syntactic Analysis and Tools

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, . . .
- 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.
  - . . .

UPPSALA UNIVERSITET

Introduction
Language and Properties
Compilation and Evaluation
Conclusion

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.

UPPSALA UNIVERSITET

Results

Relative running times:

	MAXIMUM	SUM	EXACTLY	ELEMENT
Built-in	1.0	1.0	1.0	1.0
Indexicals	1.3	2.7	2.5	1.2
Decomposition	1.9	3.0	3.1	2.0
Automaton	6.7	n/a	n/a	4.9

Runtime increase with the number of variables:

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

UPPSALA UNIVERSITET

Introduction

Language and Properties

Compilation and Evaluation

Conclusion

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.

UPPSALA UNIVERSITET

Future Work

- Improve the language.
- Improve the compilation.
- Target more solvers (including non FD solvers).
- Improve the support for analysis/transformations.
- Automatically test the consistency level.
- Synthesise propagator descriptions.

Compiler available from `http://user.it.uu.se/~jeamo371/indexicals/`

Appendix

List of implemented constraints

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