Implementing Tuple Variables in Gecode
Bachelor Thesis
Uppsala University – Autumn 2014

Patrik Broman

February 13, 2015

Abstract

This is the abstract

Contents

1 Acknowledgements

Thanks to Jean-Nol Monette for being a helpful supervisor.
Thanks to Joseph Scott for helping me with a bug.

2 Introduction

2.1 Constraint Programming

Constraint programming (CP) is a method to solve problems by first modeling the problem with different types of variables and then adding constraints that the variables must fulfill. The variables have an initial domain of possible values. The CP solver then uses the constraints to remove impossible values. A very typical problem suitable for constraint programming is sudoku. The most intuitive model is a 9x9 matrix of integer variables. The general constraints for all standard sudokus is that every cell has a domain of 1..9, and that all cells in a row, column and box must be different. For a specific sudoku constraints are added to require specific cells to have specific values.

The available data types limits the options to model a specific problem. With more data types a specific problem can be modeled in different ways. There is a famous problem called Queens. The problem is to place eight queens on a chess board in such a way that no queen threatens another. If only boolean variables were available we would have to model the problem with 64 boolean variables. If we instead use integer variables we can use the fact that all rows will have exactly one queen when we model the problem. This could be solved with booleans by redundant constraints, but it is more efficient to implement the constraints directly in the model.

2.2 Terminology

- Constraint - A condition that may be hard or soft. Hard conditions must be satisfied, but soft conditions are optional.
- Pruning - Removing values from the domain of a Variable with a Propagator.
- Propagator - In its simplest form it just
- Brancher -

- Variable -
- Searching -

2.3 Gecode

Gecode is an open source CP solver programmed in C++ and is licensed under the MIT license. It is flexible and supports customization of most parts, including data types, branchers and propagators. It is suitable for both education and professional use.

For more information, look at the homepage. `http://www.gecode.org`

3 Background

3.1 The need for tuple variables

When using a CP solver, one wants to avoid the brute force searching. This is done by intelligently choosing model and constraints. By doing this the variable domains can be pruned to smaller sizes. For example, if two variables x and y have domains $x = \{1, 2, 3\}$ $y = \{2, 3, 4\}$ and the constraint $y < x$ is added we can prune the domains to $x = \{3\}$ and $y = \{2\}$. If we instead have the same variables but add the constraint $x \neq y$ nothing would be pruned, since for all values in both domains there exists solutions. In this example the problem was modeled with two integer variables, but if the problem instead would be modeled with one tuple variable pruning would be possible. Then we would have a tuple variable t with domain $t = \{< 1, 2 >, < 1, 3 >, < 1, 4 >, < 2, 2 >, < 2, 3 >, < 2, 4 >, < 3, 2 >, < 3, 3 >, < 3, 4 >\}$ and pruning with the distinct condition would yield $t = \{< 1, 2 >, < 1, 3 >, < 1, 4 >, < 2, 3 >, < 2, 4 >, < 3, 2 >, < 3, 4 >\}$. This shrinks the size of the search tree from 9 to 7.

If we instead look at the case where $x, y = \{1..1000\}$ and the constraint $x = y$ the benefits is more obvious. Using integer variables pruning can not remove anything. If tuple variables were used, the size of the domain would shrink from 10^6 to 10^3. This indicates that tuple variables can be quite useful.

4 Tuple Variable

4.1 Creating a new Variable

4.1.1 Specification file

Creating the specification file is the first thing that needs to be done when designing a new variable. Caution should be taken, because this step may be very hard to modify later. A simple specification file could look like this.

```
[General]
Name:          IntTuple
Namespace:     MPG::IntTuple
[ModEvent]
Name:          FAILED=FAILED
[ModEvent]
Name:          VAL=ASSIGNED
[ModEvent]
Name:          NONE=NONE
[PropCond]
```

```
Name:           NONE=NONE
[PropCond]
Name:           VAL=ASSIGNED
ScheduledBy:    VAL
[End]
```

As can be seen, there are three sections. The specification file must start with the [General] section and end with [End]. There are also sections for modification events and propagation conditions. When the specification file is written a configuration script is used to generate base classes from the specification file.

The [General] section is pretty straight forward. This is where the name and namespace of the variable is specified. Both name and namespace is arbitrary. In Gecode the standard variables have namespace Gecode::VarName and in the examples in the documentation they instead have the namespace MPG::VarName.

The [ModEvent] section describes the modification events. For all variables, modification events for NONE, FAILED and ASSIGNED are required.

The [PropCond] section describes how the propagators are scheduled. For all variables, propagator conditions for NONE and ASSIGNED are required.

4.1.2 Variable implementation

The variable implementation class would with the specification file in the previous section be called IntTupleImp and inherit from the generated class IntTupleVarBase. It has to implement a function called assigned that returns true if the variable is assigned.

4.1.3 Variable class

The variable class is the class that is used when modeling the actual problem and inherits from the variable implementation. It is a read only user interface, and a programmer that does not program any custom branchers or propagators will not use anything else than this. This class would be called IntTupleVar.

4.1.4 Deltas

4.1.5 Views

The variable class does not offer any methods for removing values. These methods are instead implemented in a view.

4.1.6 Propagators

A propagator implements constraints and prunes variables. They interact with the views and not variable class and have four possible return values.

- ES_FAILED - There are no values in the domain that satisfies the constraint.
- ES_FIX - It is impossible for the propagator to prune more values before the domain has changed by either a brancher or another propagator.
- ES_SUBSUMED - The propagator is done. Regardless of any changes made to the variable this particular propagator will not be able to do anything more.
- ES_NOFIX - None of the others are applicable.

The propagator must implement ES_FAILED and at least one of the others. Furthermore, the propagator needs the following methods.

- post
- disposal - This works as a destructor. The only reason to not have a regular destructor is that destructors in C++ can't take arguments.
- copy - A method to copy the constructor.
- cost - Estimates the cost to run the propagator.
- propagate

4.1.7 Branchers

When there are no propagators able to prune any variables the CP solver must start searching. The branchers determines how this is done. The simplest brancher just picks the first value in the first branch and the rest of the values in a second branch. Another way is to pick a random value in the domain, but more sophisticated ways exist.

A brancher must implement the following methods:

- status
- choice
- commit
- print
- ngl

4.2 Implementation of exact IntPair

The exact version does not approximate the domain, that is, the implementation is a long list of all integer pairs that's currently in the domain.

The specification is as follows.

```
[General]
Name:          IntPair
Namespace:     Gecode::IntPair

[ModEvent]
Name:          FAILED=FAILED
[ModEvent]
Name:          NONE=NONE
[ModEvent]
Name:          VAL=ASSIGNED

[PropCond]
Name:          NONE=NONE
[PropCond]
Name:          VAL=ASSIGNED
ScheduledBy:   VAL
```

```
[PropCond]
Name:          BND
ScheduledBy:   VAL, BND, XBND, YBND, XMIN, YMIN, XMAX, YMAX
[PropCond]
Name:          XBND
ScheduledBy:   VAL, BND, XMIN, XMAX
[PropCond]
Name:          YBND
ScheduledBy:   VAL, BND, YMIN, YMAX
[PropCond]
Name:          XMIN
ScheduledBy:   VAL, BND, XMIN, XBND
[PropCond]
Name:          XMAX
ScheduledBy:   VAL, BND, XMAX, XBND
[PropCond]
Name:          YMIN
ScheduledBy:   VAL, BND, YMIN, YBND
[PropCond]
Name:          YMAX
ScheduledBy:   VAL, BND, YMAX, YBND
[End]
```

4.3 Restrictions

To keep things simple only two-dimensional integer tuples are considered, so called Pairs.

4.4 Implementation of rectangle IntPair

This version of IntPair approximates the domain by just storing four pairs. The benefit is vastly less use of memory, but the drawback is that it can't be pruned as efficient as the exact version. Another benefit for future work is that an approximation is possible to extend to float numbers.

4.5 Performance test

5 Related work

5.1 Implementation of bit-vector variables in a CP solver

5.2 A propagator design framework for constraints over sequences

6 Conclusions and future work

6.1 Write the implementation in such a way that it may be accepted in a new Gecode release

6.2 Test different implementations with different domain representations

6.3 Extend the tuples to arbitrary dimensions and not just pairs

6.4 Extend the tuples to floats

6.5 Reach better performance for a specific problem with an array of pair variables than with two arrays of integer variables

7 Git repository

`http://github.com/patwotrik/gecode-tuples`

8 Legal

References

[1] Calculus, Adams

[2] Operating systems and its concepts, Galvin