Integer Interval/Domain Constraints

Frank Kaufer

Department of Computer Science
Saarland University
kaufer@web.de

Abstract. This paper deals with integer constraints on a introductory level. Nevertheless the reader should be familar with the basic notions of Constraint Programming [1] [3]. After some simple examples we provide some formalisations to define integer constraints. Then we discuss propagation on a general level, before we look at some particular propagators.

1 Introduction

Integer constraints are a very elementary and well researched constraint domain. Often they are referred to as Finite Domain constraints [3][4], where variables ranging only over a finite set of nonnegative integers, i.e. a finite subset of the natural numbers, instead of the whole infinite set of integers. We will also do so in this paper. Further, we will only use nonnegative constants within the constraints. Nevertheless we well call constraints with such variables and constants integer constraints, because negative polarity can be introduced by an unary minus function.
At first we introduce two very simple examples (Fig. 1, 2), which show what integer constraints basically are about.

$$\begin{array}{r} \textit{SEND} \\ \underline{+\ \ \textit{MORE}} \\ \textit{MONEY} \end{array}$$

$$S, E, N, D, M, O, R, Y \in \{0, \ldots, 9\}$$

$$S \neq 0,\ M \neq 0$$
$$S \neq E,\ S \neq N,\ \ldots,\ E \neq N,\ \ldots \quad \text{(all\_different)}$$

$$\begin{array}{rrrrrrrrrr} & & & 1000\cdot S & + & 100\cdot E & + & 10\cdot N & + & D \\ & & + & 1000\cdot M & + & 100\cdot O & + & 10\cdot R & + & E \\ = & 10000\cdot M & + & 1000\cdot O & + & 100\cdot N & + & 10\cdot E & + & Y \end{array}$$

Fig. 1. example: Send More Money

```
declare
proc {Safe C}
   {FD.tuple code 9 1#9 C}
   {FD.distinct C}
   C.4 - C.6 =: C.7
   C.1 * C.2 * C.3 =: C.8 + C.9
   C.2 + C.3 + C.6 <: C.8
   C.9 <: C.8
   {For 1 9 1 proc {$ I} C.I \=: I end}
   {FD.distribute ff C}
end

{ExploreAll Safe}
```

Fig. 2. example: safe with 9 nonzero digits satisfying some (in/dis)equations (Mozart code [2])

As one can see, integer constraints are somehow relations about different expressions containing integer constants and variables which are declared to be values of an integer domain. In the next chapter we will formalise this more exactly, before we will discuss how the method of *constraint propagation* (that is narrowing down the domains for the declared variables) works for integer constraints in general. Finally we will look at some particular classes of propagators.

2 Constraints

We now introduce integer constraints more formally. Therefore we first have to define the *integer expressions* we are dealing with.

Definition 1 (expression). *Let a_i be some* ***constants****, x_j some* ***variables*** *with respective domains D_j and F a set of* ***functions*** *in the form of $f(y_1 : D_1, ..., y_k : D_k) : D_f$, where f is the function symbol, k the arity of the function, $y_l (l \in [1, k])$ the parameter values with respective domains D_l, and D_f the domain of the return value.*
We then call terms build of constants $a_i \in \mathbb{N}$, variables x_j with domains $D_j \subseteq \mathbb{N}$, and functions of F with domains $D \subseteq \mathbb{N}$ ***integer expressions****.*
Integer expressions without constants, with $\mathbb{B} = \{0, 1\}$, $\forall x_j : D_j \subseteq \mathbb{B}$ and

$$F = \{\wedge(y_1 : \mathbb{B}, y_2 : \mathbb{B}) : \mathbb{B}, \vee(y_1 : \mathbb{B}, y_2 : \mathbb{B}) : \mathbb{B}, \neg(y_1 : \mathbb{B}) : \mathbb{B}\}$$

we call ***boolean expressions****.*
Integer expressions with

$$F = \{+(y_1 : \mathbb{N}, y_2 : \mathbb{N}) : \mathbb{N}, \cdot(y_1 : \mathbb{N}, y_2 : \mathbb{N}) : \mathbb{N},$$
$$-(y_1 : \mathbb{N}, y_2 : \mathbb{N}) : \mathbb{N}, -(y_1 : \mathbb{N}) : \mathbb{N}\}$$

we call ***arithmetic expressions****.*
Arithmetic expressions, where all multiplications have at least one constant as

*argument are called **linear** (arithmetic) **expressions**.*

$\square$

Note 1. We write binary functions in infix notation. For example, *+(2,x)* would be *2 + x*.

Example 1 (expressions).

boolean expression: $(x \wedge \neg y) \vee (\neg x \wedge y)$
linear expression: $-2 \cdot x + 4 \cdot y - 3 \cdot z + 6$
arithmetic expression: $-x \cdot y + 4 \cdot y - 3 \cdot z \cdot z \cdot z \cdot z$

$\square$

Definition 2 (constraint). *Let x,y be variables, $D \subseteq \mathbb{N}$ a domain, and s,t integer expressions.*
*An expression $x \in D$ is called **domain** (integer) **constraint**.*
*Domain constraints and simple variable equalities like x=y are called **basic** (integer) **constraints**.*
*For $\sim \in \{=, \neq, \leq, \geq, <, >\}$ $s \sim t$ is called **nonbasic** (integer) **constraint** and in particular*

- *$s = t$ **equality constraint**,*
- *$s \neq t$ **disequality constraint**,*
- *$s \leq t$ $(s \geq t)$ **inequality constraint**, and*
- *$s < t$ $(s > t)$ **strict inequality constraint**.*

*Further we call $\langle C_{nonbasic}; C_{basic} \rangle$ for a set of nonbasic constraints $C_{nonbasic}$ and a set of basic constraints C_{basic} (integer) **constraint satisfaction problem (CSP)**.*

$\square$

Note 2. If the expressions and relations of a (nonbasic) constraint are not of interest, we can write it also $C(x_1, ..., x_n)$, where $x_1, ..., x_n$ are the contained variables. If we then assign values to the variables we get an *extensional* view on a constraint, whereas the version from the definition is *intensional*.

Definition 3 (solution). *Let $\langle C; x_1 \in D_1, ..., x_n \in D_n \rangle$ be a CSP. If for the values $s_1, ..., s_n$ with $s_1 \in D_1, ..., s_n \in D_n$ all constraints are satisfied, we call $(s_1, ..., s_n)$ a **solution** of the CSP.*

Example 2 (constraints).

domain constraints: $x \in [0, 9]$, $y \in \{0, 9\}$, $z \in \mathbb{N}$
equality constraint: $4 \cdot x - 3 = z$
inequality constraint: $z + 1 \leq 2 \cdot y$

$\square$

3 Propagation

In a constraint programming system like Mozart [2] basic constraints are stored in a so called *constraint store*, whereas each nonbasic constraint is represented by an agent called *propagator*. By the variables contained in the nonbasic constraints the propagators are connected to the constraint store and try to narrow down their domains. The whole architecture is called *constraint space* (see Fig. 3).

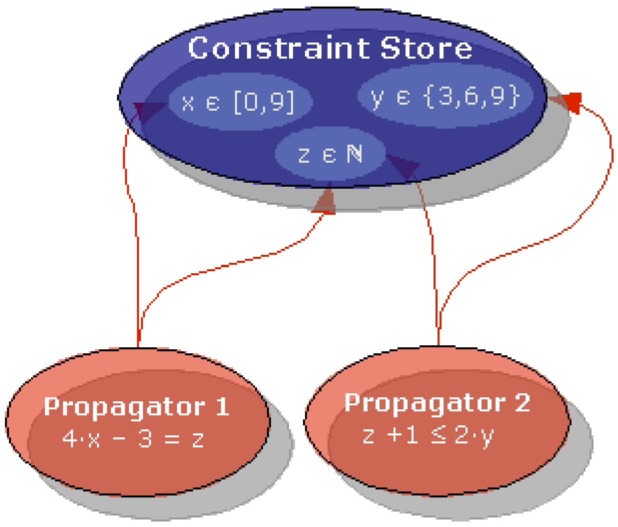

Fig. 3. constraint space

For every nonbasic constraint declaration there is exactly one single propagator. Even constraints with the same relation symbol and same expression types have different propagators. Nevertheless the propagation they accomplish would have the same operational semantics and so they would belong to the same class. Such a common class of propagators we call *generic propagator*. A generic propagator provides some *propagation rules* and is characterized by

- relation type
- functions / expression type
- propagation scheme

The *propagation scheme* constitutes the way, *how* we narrow down a domain and has to be declared in addition to a constraint. Therefore we consider two schemes:

1. *bounds propagation*
2. *domain propagation*

Bounds propagation (also known as *interval propagation*) tries to contract an interval by increasing the lower bound and/or decreasing the upper bound, while domain propagation also tries to cut out elements in between (see example 3).

Example 3 (domain/bounds propagation).
bounds propagation:

$$\frac{\langle 2 \cdot x = y; x \in [1,10], y \in [1,7]\rangle}{\langle 2 \cdot x = y; x \in [1,3], y \in [2,6]\rangle}$$

domain propagation:

$$\frac{\langle 2 \cdot x = y; x \in [1,10], y \in [1,7]\rangle}{\langle 2 \cdot x = y; x \in [1,3], y \in \{2,4,6\}\rangle}$$

$\square$

Although it seems that bounds propagation is only appropriate for constraints containing variables, which are declared to range over an interval and domain propagation for those containing variables with arbitrary domain declarations, the propagation scheme is independent from the manner of domain constraints declarations. For exemplification see example 4 and compare with example 3.

Example 4 (propagation scheme and domain constraint declaration).

bounds propagation for interval with holes:

$$\frac{\langle 2 \cdot x = y; x \in [1,10], y \in [1,3] \cup [5,7]\rangle}{\langle 2 \cdot x = y; x \in [1,3], y \in [2,3] \cup [5,6]\rangle}$$

$\square$

In general propagation is not a complete solution method. That means a CSP can have a unique solution, but propagation delivers a plenty of solutions or a CSP has no solution but propagation doesn't detect a failure. So propagation needs to be complemented by *search* (splitting). Search is a topic of its own and as a generic mechanism it isn't bound to a certain type of constraints, so that we don't want to discuss it here (see for [1]). But nevertheless we need a goal we want to reach with propagation. This goal is **local consistency**. There are a lot of different definitions of consistency in the literature [1][5][6]. The notions of *bounds consistency* (Def. 4) and *domain consistency* (Def. 5) are appropriate for the mentioned respective propagation schemes. The intuitive idea is that for achieving bounds consistency only the minimum and the maximum values of a domain have to participate in a solution of a constraint, whereas in the case of domain consistency every value of a domain has to participate in a solution.

Definition 4 (bounds consistency). *Let $x_1 \in D_1, ..., x_n \in D_n$ be some domain constraints. A nonbasic constraint $C(x_1, ..., x_n)$ is called* ***bounds consistent*** *if for each variable x_i and each value $d_i \in \{min(D_i), max(D_i)\}$, there exist values $d_1 \in D_1, ..., d_{i-1} \in D_{i-1}, d_{i+1} \in D_{i+1}, ..., d_n \in D_n$, such that $d_1, ..., d_n$ is a solution of C.*

□

Definition 5 (domain consistency). *Let $x_1 \in D_1, ..., x_n \in D_n$ be some domain constraints. A nonbasic constraint $C(x_1, ..., x_n)$ is called* ***domain consistent*** *if for each variable x_i and each value $d_i \in D_i$, there exist values $d_1 \in D_1, ..., d_{i-1} \in D_{i-1}, d_{i+1} \in D_{i+1}, ..., d_n \in D_n$, such that $d_1, ..., d_n$ is a solution of C.*
Domain consistency is also called ***hyper-arc consistency****.*

□

In general you can choose freely, if you want to use bounds or domain propagation. This leads to a tradeoff between search space size on one side and computation cost on the other. As domain propagation also cuts holes out of a domain the search space it leaves is in the worst case as large as for bounds propagation and often smaller. Therefore computation is cheaper for bounds propagation as we only have to care about the bounds of an interval to check consistency. This means if bounds propagation and domain propagation lead to the same search space, bounds propagation is preferable. Schulte and Stuckey [5] analysed some integer constraint propagators where this is the case.

Definition 6 (propagation rule). *Let $\langle C; D\rangle$ and $\langle C'; D'\rangle$ be two CSPs. If a propagator infers $\langle C'; D'\rangle$ from $\langle C; D\rangle$ by an inference rule R, we call R a* ***propagation rule*** *and write*

$$\frac{\langle C; D\rangle}{\langle C'; D'\rangle} R\ .$$

□

The operational semantics of a (generic) propagator is represented by its propagation rules (Def. 6), which we can divide in two classes:

1. *normalisation rules*
2. *domain reduction rules*

Normalisation rules in general don't change the domains of the variables and only transform CSPs into a an appropriate form for the domain reduction rules and often even introduce new variables and domains. Reducing domains, that is the actual intention of propagation, is done by the domain reduction rules.

Example 5 (propagation rules).
a normalisation rule:

$$\frac{\langle x_1 \wedge x_2 \vee x_3 = x_4; x_1 \in \{0,1\}, ..., x_4 \in \{0,1\}\rangle}{\langle x_5 = x_1 \wedge x_2\ ,\ x_5 \vee x_3 = x_4; x_1 \in \{0,1\}, ..., x_5 \in \{0,1\}\rangle}\ BooleanNormal$$

a domain reduction rule:

$$\frac{\langle x = y; x \in D_x, y \in D_y \rangle}{\langle x = y; x \in D_x \cap D_y, y \in D_y \cap D_x \rangle}\ \mathit{Equality}$$

□

4 Generic Propagators

A generic propagator is a (template) class for propagators and provides propagation rules for a whole family of constraints. Fig. 4 gives a taxonomy of some generic integer propagators, out of which we want to discuss the rules of the *boolean propagators*, the *linear bounds propagators*, and the *linear domain propagators*. For the general arithmetic propagators see [1] [4].

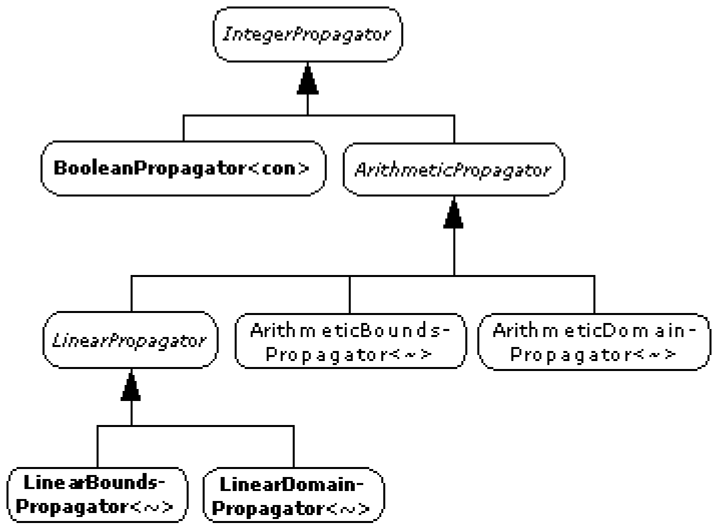
IntegerPropagator
BooleanPropagator<con>
ArithmeticPropagator
LinearPropagator
ArithmeticBounds-Propagator<~>
ArithmeticDomain-Propagator<~>
LinearBounds-Propagator<~>
LinearDomain-Propagator<~>

Fig. 4. generic propagators taxonomy ($\sim \in \{=, \neq, \leq, \geq, <, >\}$, $con \in \{\wedge, \vee, \neg\}$)

4.1 Boolean Propagator

Boolean propagators are also called *0/1-propagators* and treat equalities over boolean expressions and, according to this, only variables ranging over a subset of $\mathbb{B}$. The boolean constants *true* and *false* can be modeled by the following variables with the respective domain constraints:

- $x_T := x \in \{1\}$
- $x_F := x \in \{0\}$

In the following we discuss three types of boolean propagators, one for every connective out of $\neg, \wedge, \vee$. Therefore we define a *simple boolean constraint* for each connective:

- $\neg x = y$
- $x \wedge y = z$
- $x \vee y = z$

One particular propagator will then provide propagation rules for every simple boolean constraints. To transform arbitrary boolean constraints into simple ones, we assume some general normalisation rules (examples 5, 6).

Example 6 (boolean normalisation rules).

$$\langle (x_1 \wedge x_2) \vee (x_3 \wedge \neg(x_1 \vee x_2)) = x_4 \ ; \ D \rangle$$

$$(\underbrace{x_1 \wedge x_2}_{y_1}) \vee (x_3 \wedge \neg(x_1 \vee x_2)) = x_4$$

$$y_1 \vee (x_3 \wedge \neg(\underbrace{x_1 \vee x_2}_{y_2})) = x_4 \ , \ x_1 \wedge x_2 = y_1$$

$$y_1 \vee (x_3 \wedge \underbrace{\neg y_2}_{y_3}) = x_4 \ , \ x_1 \wedge x_2 = y_1 \ , \ x_1 \vee x_2 = y_2$$

$$y_1 \vee (\underbrace{x_3 \wedge y_3}_{y_4}) = x_4 \ , \ x_1 \wedge x_2 = y_1 \ , \ x_1 \vee x_2 = y_2 \ , \ \neg y_2 = y_3$$

$$\langle y_1 \vee y_4 = x_4 \ , \ x_1 \wedge x_2 = y_1 \ , \ x_1 \vee x_2 = y_2 \ , \ \neg y_2 = y_3 \ , \ x_3 \wedge y_3 = y_4 \ ; \ D' \rangle$$

$\square$

Domain reduction rules for simple boolean constraints are very simple. For disjunction and conjunction we have 6 rules, respectively, whereas for negation we have 4 rules. All these rules are analogous to the known truth tables. For example, if we look at conjunction and assume, that x and y are restricted to *1*, we can apply the following rule:

$$\frac{\langle (x \wedge y) = z \ ; \ x \in \{1\}, y \in \{1\}, z \in D_z \rangle}{\langle \emptyset \ ; \ x \in \{1\}, y \in \{1\}, z \in D_z \cap \{1\} \rangle} \ AND_1$$

The only thing to note is that new domain of z is $D_z \cap \{1\}$ and not just $\{1\}$. The intersection is necessary to detect failure, if for example $D_z = \{0\}$.

4.2 Linear Bounds Propagator

Linear bounds propagators treat constraints of the form $s \sim t, \sim \in \{=, \neq, \leq, \geq, <, >\}$ where s and t are linear expressions (see example 2). As propagation scheme they use bounds propagation. Though not necessary, we want to assume for this subsection, that every variable is declared to range over an interval. Further we want to assume that we have some general normalisation rules, which transform arbitrary linear expressions to sums in the *normalised* form $\sum_{i=1}^{n} a_i \cdot x_i \;\sim\; b$, where all x_i are pairwise different.

Although we have six relation types, we can reduce every constraint except a disequality constraint to an inequality constraint by normalisation rules[1]. This makes the domain reduction rule for inequality constraints most important for these propagators. Because the formal description looks quite complicated and hides a very simple idea, we first want to get an intuitive idea of that rule by an example.

Therefore we take the following CSP:

$$\langle 3 \cdot x + 4 \cdot y - 5 \cdot z \leq 7 \; ; \; x \in [4, 10000] \; , \; y \in [2, 4] \; , \; z \in [0, 5] \rangle$$

If we want to reduce the domain of a variable by means of bounds propagation we have find out the maximum or minimum bounds, which are implicitly contained in the inequality. For that reason, we separate each variable and get for x:

$$x \leq \frac{7 - 4 \cdot y + 5 \cdot z}{3}$$

If we now put in the highest possible values for all variables with positive polarity and the lowest possible values for all variables with negative polarity, i.e. the upper and lower bounds of the intervals, we get

$$x \leq \frac{7 - 4 \cdot l_y + 5 \cdot h_z}{3}$$

Since we are interested in integer bounds we have to use the next integer below that fraction. This we can achieve with the floor function:

$$x \leq \left\lfloor \frac{7 - 4 \cdot l_y + 5 \cdot h_z}{3} \right\rfloor$$

With all values put in we compute

$$x \leq \left\lfloor \frac{7 - 4 \cdot 2 + 5 \cdot 5}{3} \right\rfloor = \lfloor 8 \rfloor = 8$$

and see that x can not be greater than *8*. So we substitute the the domain constraint of x and get the CSP

[1] In general in an implementation it would not make sense to reduce all the relations to inequality.

$$\langle 3 \cdot x + 4 \cdot y - 5 \cdot z \leq 7 \; ; \; x \in [4,8] \; , \; y \in [2,4] \; , \; z \in [0,5] \rangle$$

We can now compute the implicte maximum upper bound of y analogous and get

$$y \leq \frac{7 - 3 \cdot x + 5 \cdot z}{4} \leq \left\lfloor \frac{7 - 3 \cdot l_x + 5 \cdot h_z}{4} \right\rfloor = \lfloor 5 \rfloor = 5 \; .$$

But as y is already stronger restricted and has an upper bound of *4*, the computed information is useless and the old domain constraint will be preserved.

If we now look at z, we can see a difference to x and y. That is the negative polarity. This entails a turnaround of the inequality symbol while we separate z, because additionally we have to multiply the whole inequality with *(-1)*. With the turnaround we can not compute a maximum upper bounder any longer, but a minimum lower bound. For that reason we now use the lowest possible values for variables with positive polarity, the highest possible values if there were some with negative polarity and the ceiling function to get next upper integer:

$$z \geq \frac{-7 + 3 \cdot x + 4 \cdot y}{5} \leq \left\lfloor \frac{-7 + 3 \cdot l_x + 4 \cdot l_y}{5} \right\rfloor = \lceil 2.6 \rceil = 3 \; .$$

As *3* is higher than *0*, we have a new domain constraint for y and the final CSP

$$\langle 3 \cdot x + 4 \cdot y - 5 \cdot z \leq 7 \; ; \; x \in [4,8] \; , \; y \in [2,4] \; , \; z \in [3,5] \rangle \; .$$

With this intuitive idea of the domain reduction rule for inequality constraints it's straightforward to the general case (Def. 7).

Definition 7 (LinearInequality). *Let* $\sum_{i=0}^{n} a_i \cdot x_i \leq b$ *be a normalised inequality constraint.*

$$\frac{\langle \sum_{i=1}^{n} a_i \cdot x_i \leq b \; ; \; x_1 \in [l_1, h_1], ..., x_n \in [l_n, h_n] \rangle}{\langle \sum_{i=1}^{n} a_i \cdot x_i \leq b \; ; \; x_1 \in [l'_1, h'_1], ..., x_n \in [l'_n, h'_n] \rangle} \; \textbf{\textit{LinearInequality}}$$

$\forall x_j | a_j > 0 :$

$$x_j \leq \left\lfloor \frac{b - \sum_{i=1, i \neq j, a_i > 0}^{n} a_i \cdot l_i + \sum_{i=1, a_i < 0}^{n} a_i \cdot h_i}{a_j} \right\rfloor = \overline{x_j}$$

$$h'_j = min(\overline{x_j}, h_j), l'_j = l_j$$

$\forall x_j | a_j < 0 :$

$$x_j \geq \left\lfloor \frac{-b + \sum_{i=1, a_i > 0}^{n} a_i \cdot l_i - \sum_{i=1, i \neq j, a_i < 0}^{n} a_i \cdot h_i}{a_j} \right\rfloor = \underline{x_j}$$

$$h'_j = h_j, l'_j = max(\underline{x_j}, l_j)$$

$\square$

Constraints with the other inequality relation symbol can be reduced to the one already known be a normalisation rule (Def. 8), an equality constraint can transformed to two inequality constraints (Def. 9).

Definition 8 (InequalityNormalisation). *Let* $\sum_{i=0}^{n} a_i \cdot x_i \geq b$ *be a normalised inequality constraint.*

$$\frac{\langle \sum_{i=1}^{n} a_i \cdot x_i \geq b \ ; \ x_1 \in [l_1, h_1], ..., x_n \in [l_n, h_n] \rangle}{\langle \sum_{i=1}^{n} -a_i \cdot x_i \leq -b \ ; \ x_1 \in [l_1, h_1], ..., x_n \in [l_n, h_n] \rangle} \ \textbf{\textit{InequalityNormalisation}}$$

□

Definition 9 (EqualityNormalisation). *Let* $\sum_{i=0}^{n} a_i \cdot x_i = b$ *be a normalised equality constraint.*

$$\frac{\langle \sum_{i=1}^{n} a_i \cdot x_i = b \ ; \ x_1 \in [l_1, h_1], ..., x_n \in [l_n, h_n] \rangle}{\langle \sum_{i=1}^{n} a_i \cdot x_i \geq b, \sum_{i=1}^{n} a_i \cdot x_i \leq -b \ ; \ x_1 \in [l_1, h_1], ..., x_n \in [l_n, h_n] \rangle} \ *$$

****EqualityNormalisation***

□

For disequality constraints we can't do very much. A disequality propagator in Mozart waits until at most one variable of a constraint is undetermined and then cuts out the apposite values. For some simple cases we can do even more (Def. 10).

Definition 10 (SimpleDisquality).

$$\frac{\langle x \neq y \ ; \ x \in [a, b], y \in [c, d] \rangle \quad (b < c \vee d < a)}{\langle \emptyset \ ; \ x \in [a, b], y \in [c, d] \rangle} \ \textbf{\textit{SimpleDisequality 1}}$$

$$\frac{\langle x \neq y \ ; \ x \in [a, b], y \in [a, a] \rangle}{\langle \emptyset \ ; \ x \in [a+1, b], y \in [a, a] \rangle} \ \textbf{\textit{SimpleDisequality 2}}$$

$$\frac{\langle x \neq y \ ; \ x \in [a, b], y \in [b, b] \rangle}{\langle \emptyset \ ; \ x \in [a, b-1], y \in [b, b] \rangle} \ \textbf{\textit{SimpleDisequality 3}}$$

□

Strict inequalities constraints can either be reduced analogous to inequality or they can be transformed to inequality and disequality constraints (Def. 11).

Definition 11 (StrictInequalityNormalisation).
Let $\sum_{i=0}^{n} a_i \cdot x_i < b$, $\sum_{i=0}^{n} a_i \cdot x_i > b$ *be normalised strict inequality constraints.*

$$\frac{\langle \sum_{i=1}^{n} a_i \cdot x_i < b \ ; \ x_1 \in [l_1, h_1], ..., x_n \in [l_n, h_n] \rangle}{\langle \sum_{i=1}^{n} a_i \cdot x_i \leq b, \sum_{i=1}^{n} a_i \cdot x_i \neq -b \ ; \ x_1 \in [l_1, h_1], ..., x_n \in [l_n, h_n] \rangle} \ *$$

****StrictInequalityNormalisation 1***

$$\frac{\langle \sum_{i=1}^{n} a_i \cdot x_i > b \ ; \ x_1 \in [l_1, h_1], ..., x_n \in [l_n, h_n] \rangle}{\langle \sum_{i=1}^{n} a_i \cdot x_i \geq b, \sum_{i=1}^{n} a_i \cdot x_i \neq -b \ ; \ x_1 \in [l_1, h_1], ..., x_n \in [l_n, h_n] \rangle} \ *$$

****StrictInequalityNormalisation 2***

□

4.3 Linear Domain Propagator

The linear domain propagator has the same setting as the linear bounds propagator with the difference that the propagation scheme is domain propagation. Furthermore we now also want to admit arbitrary domains (i.e intervals with holes). For the definition of the linear domain propagators we only have to make some extensions to the linear bounds propagators, i.e. to their rules, namely:

– modification of the *LinearInequality* rule (Def. 12)
– relaxation of the *SimpleDisequality* rules (Def. 13)
– modification of the *Equality* rule (see example 5)

Definition 12 (DomainInequality). *Let $\sum_{i=0}^{n} a_i \cdot x_i \leq b$ be a normalised inequality constraint, D_i some arbitrary integer domains, l_i its minimum value and h_i its maximum value.*

$$\frac{\langle \sum_{i=1}^{n} a_i \cdot x_i \leq b \ ; \ x_1 \in D_i, ..., x_n \in D_n \rangle}{\langle \sum_{i=1}^{n} a_i \cdot x_i \leq b \ ; \ x_1 \in D_1 \in D'_n \rangle} \ \textbf{\textit{DomainInequality}}$$

$\forall x_j | a_j > 0 :$

$$x_j \leq \left\lfloor \frac{b - \sum_{i=1, i \neq j, a_i > 0}^{n} a_i \cdot l_i + \sum_{i=1, a_i < 0}^{n} a_i \cdot h_i}{a_j} \right\rfloor = \overline{x_j}$$

$$h'_j = min(\overline{x_j}, h_j), l'_j = l_j$$

$$\underline{D'_j = [l'_j, h'_j] \cap D_j}$$

$\forall x_j | a_j < 0 :$

$$x_j \geq \left\lceil \frac{-b + \sum_{i=1, a_i > 0}^{n} a_i \cdot l_i - \sum_{i=1, i \neq j, a_i < 0}^{n} a_i \cdot h_i}{a_j} \right\rceil = \underline{x_j}$$

$$h'_j = h_j, l'_j = max(\underline{x_j}, l_j)$$

$$\underline{D'_j = [l'_j, h'_j] \cap D_j}$$

□

Definition 13 (DomainSimpleDisquality).

$$\frac{\langle x \neq y \ ; \ x \in D_x, y \in D_y \rangle \quad (D_x \cap D_y = \emptyset)}{\langle \emptyset \ ; \ x \in D_x, y \in D_y \rangle} \ \textbf{\textit{DomainSimpleDisequality 1}}$$

$$\frac{\langle x \neq y \ ; \ x \in D_x, y \in \{a\} \rangle}{\langle \emptyset \ ; \ x \in D_x - \{a\}, y \in \{a\} \rangle} \ \textbf{\textit{DomainSimpleDisequality 2}}$$

□

5 Outlook

This paper dealt with integer constraints on a very basic level with the intention to get an idea how propagation works in principle and how to define propagation rules for these constraints. Therefore we simplified some aspects like the propagation rules for some constraints over linear expressions and omitted other things like general arithmetic constraints. The latter one can be read in [1], where also an extension to real numbers can be found.

Integer constraints are the essence of many higher level constraints, as finite domains can be mapped to an integer domains. Such higher level constraints with specialised higher level propagators are for example planning/scheduling constraints or global constraints like *alldifferent* [6].

As already mentioned an important task in the field of integer constraints is the analysis of propagators, whether they lead to the same search space for domain and bounds propagation [5].

References

1. Krzysztof R. Apt. Principles of Constraint Programming. Cambridge University Press, 2003.
2. Mozart Consortium. The Mozart programming system, 1999. Available from `www.mozart-oz.org`.
3. Christian Schulte and Gert Smolka. Finite Domain Constraint Programming in Oz. A Tutorial. In *Mozart Documentation*, June 17th, 2004.
4. Denys Duchier, Leif Kornstaedt, Martin Homik, Tobias Mller, Christian Schulte and Peter Van Roy. System Modules - Constraint Programming. In *Mozart Documentation*, June 17th, 2004.
5. C. Schulte and P.J. Stuckey. When do bounds and domain propagation lead to the same search space. In *3rd International Conference on Principles and Practice of Declarative Programming*, pages 115-126, 2001.
6. W.J. van Hoeve. The Alldifferent Constraint: a Systematic Overview. Submitted manuscript, January 2005.