Journal of Universal Computer Science, vol. 12, no. 11 (2006), 1466-1499
submitted: 1/5/06, accepted: 15/10/06, appeared: 28/11/06 © J.UCS

An Interval Constraint Branching Scheme for Lattice Domains

Antonio J. Fernández
(University of Málaga, Spain
afdez@lcc.uma.es)

Patricia M. Hill
(University of Leeds, UK
hill@comp.leeds.ac.uk)

Abstract This paper presents a branching schema for the solving of a wide range of interval constraint satisfaction problems defined on any domain of computation, finite or infinite, provided the domain forms a lattice. After a formal definition of the branching schema, useful and interesting properties, satisfied by all instances of the schema, are presented. Examples are then used to illustrate how a range of operational behaviors can be modelled by means of different schema instantiations. It is shown how the operational procedures of many constraint systems (including cooperative systems) can be viewed as instances of this branching schema. Basic directives to adapt this schema to solving constraint optimization problems are also provided.
Key Words: interval constraint, constraint branching, constraint solving, constraint optimization, constraint propagation.
Category: D.3.3, I.1.2

1 Introduction

Solving a constraint satisfaction problem (CSP) means finding assignments of values to the variables such that all constraints are satisfied. A CSP can have many solutions; usually either any one or all of the solutions must be found. However, sometimes, because of the cost of finding all solutions, *partial* CSPs are used where the aim is just to find the best solution within fixed resource bounds. An example of a partial CSP is a *constraint optimization problem* (COP) that assigns a value to each solution and tries to find an optimal solution (with respect to these values) within a given time frame.

Constraint solving algorithms have received intense study from many researchers, and one of the main issues has been in developing new and more efficient methods to solve classical CSPs [Freuder and Hubbe, 1995, Wallace, 1993] and partial CSPs [Freuder and Wallace, 1992, Meseguer and Larrosa, 1995]. See [Kumar, 1992, Ruttkay, 1998, Smith, 1995, Van Hentenryck, 1995] for more information on constraint solving algorithms and [Kondrak and Van Beek, 1997, Nadel, 1989] for selected comparisons.

There are two steps in constraint solving, constraint propagation and constraint branching. The basic idea of constraint propagation algorithms (also called *filtering* or *arc consistency algorithms*) consists of removing, from the domains associated to the constrained variables, inconsistent values that can never be part of any solution. This process reduces significantly the search tree and possibly the computational effort to find a solution if one exists (or to prove the optimality of this in case of an optimization problem) or to demonstrate that there is no solution. In general, the results are propagated through the whole constraint set and the process is repeated until a stable set is obtained. In the scientific literature, one can find a number of proposals that identify some general principles for constraint propagation (e.g., [Apt, 1999, Apt, 2000, Apt, 2005, Apt and Monfroy, 2001, Bistarelli et al., 1999, Bistarelli, 2004, Fernández and Hill, 2004, Schiex et al., 1995, Van Hentenryck et al., 1992]).

The main drawback of constraint propagation algorithms is that they are usually incomplete in the sense that often they are not enough for solving a CSP, that is to say, they do not eliminate all the non-solution elements from the domains and as consequence, it is necessary to employ some additional strategy or another kind of algorithm to solve it. One complementary method to the propagation algorithms is the so called *constraint branching* that divides the variable domains to construct new subproblems (i.e., branches in the search tree) on which constraint propagation is reactivated. However, regarding constraint branching, to our knowledge, there is no study specifically focused to find a general framework. Perhaps the main reason for it is that constraint branching is basically accepted as a means of generating further constraint propagation and the branching heuristics are very specific to the computation domain. So, most of the work existing in the literature that deals with constraint branching is concerned with specific computation domains.

In [Fernández and Hill, 2004], an interval constraint propagation schema for solving CSPs (a set of interval constraints defined on a set of lattice structure computation domains) is described. The schema removes inconsistent values from the initial domain of the variables that cannot be part of any solution. It is shown how the results are propagated through the whole constraint set and the process is repeated until a stable set is obtained. However, although the propagation schema is correct, guaranteed to find a *most general solution* to the constraint store representing a CSP ([Fernández and Hill, 2004][Theorem 1]), it is incomplete in the sense that it may not determine exactly which values in the domains (i.e., intervals) of the constrained variables are the correct answers to the problem.

This paper proposes a branching schema that is complementary to the constraint propagation schema described in [Fernández and Hill, 2004]. The com-

bination of these two schemas forms a complete[1] interval constraint solving framework that can be used on any set of domains which have the structure of a lattice, independently of their nature and, in particular, their cardinality. As consequence it can be used for most existing constraint domains (finite or continuous) and, as for the framework described in [Fernández and Hill, 2004], is also applicable to combined domains and cooperative systems.

As for the constraint propagation algorithm in [Fernández and Hill, 2004], we define a number of interesting properties that are satisfied by any instance of the branching schema and show that the operational procedures of many interval constraint systems (including cooperative systems) are instances of our branching schema.

Therefore, this paper represents an attempt to find general principles for interval constraint branching, on any (set of) domain(s) *not necessarily numeric* with lattice structure.

The structure of the paper is as follows: [Section 2] concerns related work; [Section 3] introduces some preliminary concepts used throughout the paper; [Section 4] provides further key concepts specific to the proposed constraint branching schema; [Section 5] defines the main functions involved in interval constraint branching; [Section 6] describes a parameterized generic branching schema that allows (complete) solving of interval CSPs defined on any set of lattices, stating its main properties; [Section 7] gives an outline of how to extend this schema for partial constraint solving (i.e., constraint optimization); [Section 8] illustrates, by means of an example, how the choice of the parameters in the schema can influence the solving; [Section 9] concludes the paper. Proofs of stated results are included in the Appendix.

2 Related work

Here we discuss some relevant works that have been done on constraint branching in specific computation domains. Of course the list is not exhaustive.

For instance, considering the discrete domain, branching is usually called *labeling* [Van Hentenryck, 1989], and labeling often consists of a combination of two processes: variable and value ordering, that basically means selecting a variable and assigning it a value from its domain in order to reactivate constraint propagation. The order in which variables and values are instantiated is guided by heuristics and is assumed to have a significant influence on the shape of the search tree and thus the performance of the solution [Apt, 2003]. An empirical study on the influence of different branching strategies in labeling was developed recently in [Park, 2006]. Surprisingly, one of the conclusions reached in

[1] 'Complete' in the sense that the correctness and completeness of the branching schema can be guaranteed.

this work was that, for finite domain CSPs, the choice of the branching strategy does not matter much if an effective variable ordering is selected. However, in optimization problems, there was some evidence that domain splitting improves the running time. In any case, this work considered only three branching strategies (namely 2-way branching, k-way branching and domain splitting - see below) in the finite domain so that the conclusions cannot be generalized to other computation domains. Also [van Hoeve and Milano, 2004](extended version) analyzes constraint branching by comparing labeling and partitioning (see below) and draws interesting conclusions (e.g., partitioning on depth-first based search strategies should be preferred to labeling when a partial order is defined on the computation domain; however partitioning reduces the effect of constraint propagation with respect to labeling). This work also proposes postponing branching decisions when the value ordering heuristic produces ties.

In addition to the variable and value ordering, it is well-known that the election of the tree traversal heuristic greatly influences the performance of the solving, and for this reason we can find different proposals for traversing the search tree in specific ways that try boosting the search for a(n optimal) solution in a CSP (COP). For instance [Harvey and Ginsberg, 1995] proposes a limited discrepancy search strategy as a replacement to the standard depth-first search, whereas [van Hoeve, 2006] uses the solution generated by a semidefinite relaxation to define search ordering and tree traversal.

On infinite domains, labeling is rarely applied as for FD. Of course there are exceptions such as that shown in [Monfroy, 1996, Monfroy et al., 1995] that applies labeling to process the solutions on infinite and continuous domains. Before applying labeling, the only values a variable can take are roots of an univariate polynomial so that in fact only discrete and finite domains are considered.

Traditionally, on the continuous domain (i.e., the real domain) the branching process consists of splitting the domain of a variable in two or more parts (i.e., intervals) so as to continue with the search for a solution in each of the derived partitions. This process is usually called *domain splitting* or *interval partitioning* and was implemented in well-known systems such as CLP(BNR) [Older and Benhamou, 1993] and CLIP [Hickey, 2000]. These systems provide interval constraint solving on which a real variable has associated an interval (in the usual meaning of set theory) and a classical strategy of "divide and conquer" in the solving of problems involving real numbers is usually employed. When no more propagation is possible, the interval solver uses a sort of domain splitting to return each answer. This method is called *split-and-solve* [Benhamou and Older, 1997]. The *split-and-solve* method repeatedly selects a variable, splits its associated interval into two or more parts and uses backtracking to look for solutions in each partition. Of course, there is the necessity of a termination test that avoids the infinite splitting of ranges (at least theoretically

because in practice the real domain is finite since the precision of a machine is finite). Particularly, CLP(BNR) extends this strategy to the Boolean and integer domains. Other methods of interval partitioning for continuous constrained optimization can be found in the literature (e.g., [Pedamallu et al., 2006]).

There are many papers that are concerned with the constraint optimization problem and branch and bound ($B\&B$), the most common technique used to solve these problems. For instance, [Bistarelli et al., 2000] presents an extension of the definition and properties of the labeling procedure from classical CSPs to the soft CSP framework. An implementation of a complete solver based on $B\&B$ with variable labeling for soft constraint propagation is published in [Bistarelli et al., 2002]. In a very different work, [Appleget and Wood, 2000] describes a technique called 'explicit-constraint branching' to improve the performance of $B\&B$ algorithms for solving certain mixed-integer programs. Also, in [Vu et al., 2005], a uniform view of search strategies inside the whole constraint solving process, here named branch-and-prune methods, for solving CSPs completely defined in the real domain is presented. This work is based on an interval reasoning. Other similar proposals, to that of branch-and-prune, of constraint branching embedded in methods like $B\&B$ can be found in the literature; e.g., [Perregaard, 2003] examines the feasibility of applying the ideas for strengthening cuts to the simple branching disjunctions for the purpose of creating better mixed integer branches in $B\&B$ methods applied to solved mixed integer programs.

3 Preliminaries and previous work

Here we summarize the main notation and terminology used throughout the paper. To make the paper self-contained, some of the concepts and notations introduced here are taken from previous work [Fernández and Hill, 2004].

3.1 Sets and lattices

If L is a set, then $\#L$ denotes its cardinality, $\wp(L)$ its power set and $\wp_f(L)$ the set of all the finite subsets of L. Also, if L is a partially ordered set, then L is a *lattice* if $lub_L(x, y)$ and $glb_L(x, y)$ exist, for any two elements $x, y \in L$, where $glb_L/2$ and $lub_L/2$ denote, respectively, the functions that return the *greatest lower bound* and the *least upper bound* of two elements in L. The *dual of* L, denoted by $\hat{L}$, is the lattice that contains the dual element of any element in L, that is to say, $\hat{L} = \{\hat{a} \mid a \in L\}$, and where the ordering is reversed with respect L, that is to say, if $a, b \in L$, then $\hat{a} \preceq \hat{b}$ if and only if $b \preceq a$

The domain on which the values are actually computed, is called a *computation domain*. The key aspect of the constraint system described in this paper is

that it can be built on any computation domain provided it is a lattice. Throughout the paper, we let $\mathcal{L}$ denote a (possibly infinite) set of computation domains, with lattice structure, containing at least one element L. If it exists, $\bot_L$ (resp. $\top_L$) denotes *the bottom element* (resp. *the top element*) of L. With each computation domain $L \in \mathcal{L}$, we associate a set of variable symbols V_L that is disjoint from $V_{L'}$ for any $L' \in \mathcal{L}\backslash\{L\}$. We define $\mathcal{V}_\mathcal{L} = \cup\{V_L | L \in \mathcal{L}\}$. It is assumed (without loss of generality) that all $L \in \mathcal{L}$ are lifted lattices[2]. In rest of the paper, $(L, \preceq, glb_L, lub_L, \bot_L, \top_L)$ denotes a (possible lifted) lattice on L with ordering $\preceq$, (possibly fictitious) bounds $\bot_L$ and $\top_L$.

Example 1. Most classical constraint domains are lattices. For instance,

$$(Integer, \leq, mini, maxi, \bot_{Integer}, \top_{Integer}),$$
$$(\Re, \leq, mini, maxi, \bot_\Re, \top_\Re),$$
$$(Bool, \Rightarrow, \wedge, \vee, false, true),$$
$$(Set\ L, \subseteq, \cap, \cup, \emptyset, L)$$

are lattices for the integers, reals, Booleans and sets, respectively, under their usual orders where *mini* and *maxi* functions return, respectively, the minimum and maximum element of any two elements in the integers or reals. Note that *Integer* and $\Re$ are lifted lattices and include the fictitious elements $\top_{Integer}$, $\bot_{Integer}$, $\top_\Re$ and $\bot_\Re$. For the Booleans, it is assumed that $Bool = \{false, true\}$. For the set lattice, we assume that $Set\ \ L = \wp(L)$, for each $L \in \mathcal{L}$, where $\mathcal{L} = \{Integer, \Re, Bool\} \cup \{Set\ \ L \mid L \in \mathcal{L}\}$. Note that $\mathcal{L}$ is an infinite set of computations domains.

Lattice Products Let L_1 and L_2 be two (lifted) lattices. Then the direct product $\langle L_1, L_2\rangle$ and the lexicographic product (L_1, L_2) are lattices where:

$$\begin{aligned} glb(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) = &\quad \langle glb_{L_1}(x_1, y_1), glb_{L_2}(x_2, y_2)\rangle; \\ glb((x_1, x_2), (y_1, y_2)) = &\text{ if } x_1 = y_1 \text{ then } (x_1, glb_{L_2}(x_2, y_2)) \\ &\text{elsif } x_1 \prec y_1 \text{ then } (x_1, x_2) \\ &\text{elsif } x_1 \succ y_1 \text{ then } (y_1, y_2) \\ &\text{else } (glb_{L_1}(x_1, y_1), \top_{L_2}); \end{aligned}$$

lub is the dual of *glb*;

$$\top_{\langle L_1, L_2\rangle} = \langle \top_{L_1}, \top_{L_2}\rangle \text{ and } \bot_{\langle L_1, L_2\rangle} = \langle \bot_{L_1}, \bot_{L_2}\rangle;$$
$$\top_{(L_1, L_2)} = (\top_{L_1}, \top_{L_2}) \text{ and } \bot_{(L_1, L_2)} = (\bot_{L_1}, \bot_{L_2}).$$

[2] The *lifted lattice* of L is $L \cup \{\bot_L, \top_L\}$ where $\bot_L$ is the greatest lower bound of L, if it exists, and is a new element not in L such that $\forall a \in L, \bot_L \prec a$, otherwise; similarly, $\top_L$ is the least upper bound of L, if it exists, and is a new element not in L such that $\forall a \in L, a \prec \bot_L$, otherwise.

Moreover,

$$\langle x_1, y_1\rangle \preceq \langle y_1, y_2\rangle \text{ iff } x_1 \preceq y_1 \text{ and } x_2 \preceq y_2;$$
$$(x_1, y_1) \preceq (x_2, y_2) \text{ iff } x_1 \prec x_2 \text{ or } x_1 = x_2 \text{ and } y_1 \preceq y_2.$$

3.2 The bounded computation domains

To allow for continuous and infinite domains, any underlying computation domain L is first replaced by two extended forms of the domain, a left and a right *bounded computation domain*. For it, we defined open and closed bounds of the intervals; we first defined the *bracket domain* B as a lattice containing just ')' and ']' with ordering ')' $\prec_B$ ']'. We let '}' denote any element of B. Then we constructed the (right) *simple bounded computation domain* (for L) to be the lattice resulting from the lexicographic product (L, B) and is denoted L^s. Throughout the paper, an element t=(a,'}') $\in L^s$ will often be denoted as 'a}' or $\mathrm{a}_\}$. The *mirror (of L^s)* (also called the left *bounded computation domain*) is the lexicographic product $(\hat{L}, B)$ (where $\hat{L}$ is the dual lattice of L) and is denoted by $\overline{L^s}$. The *mirror* of an element t=(a,'}') $\in L^s$ is the element $(\hat{\mathrm{a}}, \text{'}\}\text{'}) \in \overline{L^s}$ and is also denoted as $\overline{t}$, '{a', $\hat{\mathrm{a}}_\}$ or simply $\overline{\mathrm{a}_\}}$ as it is evident that if $\overline{t} = \hat{\mathrm{a}}_\}$ then $t = \mathrm{a}_\}$.

Example 2. When $L = Integer$, $6_]$ denotes $(6, \text{']'})$ and $\overline{6_]}$ denotes $(\hat{6}, \text{']'})$. Also

$0_) \prec 0_] \prec 1_) \prec 1_] \prec \ldots \prec \top_{L_)} \prec \top_{L_]}$ in $Integer^s$,
$\overline{\top_{L_)}} \prec \overline{\top_{L_]}} \prec \ldots \prec \overline{1_)} \prec \overline{1_]} \prec \overline{0_)} \prec \overline{0_]}$ in $\overline{Integer^s}$,
$glb_{L^s}(3_], 5_]) = lub_{L^s}(3_], 3_)) = 3_]$, denoted also as 3],
$glb_{\overline{L^s}}(\overline{3_]}, \overline{5_]}) = \overline{lub_{L^s}(3_], 5_])} = \overline{5_]} = (\hat{5}, \text{']'})$, denoted also as [5,
$lub_{\overline{L^s}}(\overline{3_]}, \overline{5_]}) = \overline{glb_{L^s}(3_], 5_])} = \overline{3_]} = (\hat{3}, \text{']'})$, denoted also as [3.

Moreover, when $L = Set\ Integer$, $\{1,3\}_)$ denotes $(\{1,3\}, \text{')'})$ and $\overline{\{1,3\}_)}$ denotes $(\widehat{\{1,3\}}, \text{')'})$. Also

$\{1\}_] \prec \{1,3\}_) \prec \{1,3\}_] \prec \{1,3,5\}_]$ in $Set\ Integer^s$,
$\overline{\{1,3,5\}_]} \prec \overline{\{1,3\}_)} \prec \overline{\{1,3\}_]} \prec \overline{\{1\}_]}$ in $\overline{Set\ Integer^s}$,
$glb_{L^s}(\{4\}_], \{4,6\}_]) = \{4\}_]$, denoted also as $\{4\}]$
$lub_{L^s}(\{3\}_], \{4,6\}_]) = \{3,4,6\}_]$, denoted also as $\{3,4,6\}]$
$lub_{\overline{L^s}}(\overline{\{4\}_]}, \overline{\{4,6\}_)}) = \overline{glb_{L^s}(\{4\}_], \{4,6\}_])}, \overline{\{4\}_]} = (\{\hat{4}\}, \text{']'})$, denoted also as $[\{4\}$.

To enable user defined propagation and constraint cooperation we also extended the simple bounded computation domain (i.e., L^s) to include an additional construct called an *indexical* (i.e., functions that propagate the bounds

of the interval associated to constrained variables) to form a new domain called the *bounded computation domain* L^b (see [Fernández and Hill, 2004]).

3.3 The interval domain

The interval domain (for some $L \in \mathcal{L}$) which was used for the constraint propagation consists of the set of pairs $\langle \overline{s}, t \rangle$, where $\overline{s} \in \overline{L^s}$ is in the left and $t \in L^s$ is in the right bounded computation domain. Then, the *interval domain* R^b_L over L is defined as the direct product $\langle \overline{L^b}, L^b \rangle$ whereas the *simple interval domain* R^s_L over L is defined as the direct product $\langle \overline{L^s}, L^s \rangle$. A *simple range* is an element of R^s_L. An element $r \in R^b_L$ is called a *range* and, also, if $r \in R^s_L$, then we also say it is *simple*. A simple range $r = \langle \overline{s}, t \rangle$ (also denoted as $\overline{s}, t$) is *consistent* if [3] $s \preceq_{L^s} t$ and, if $s = a_)$ and, for some $b \in B$, $t = a'_b$, then $a \neq a'$. Note that $R^s_L \subset R^b_L$.

Example 3. In the domains $\Re$, *Integer* and *Set Integer*,

$\langle \overline{2.3_]}, 8.9_) \rangle \in R^s_{\Re}$ is consistent, simple and can also be written $[2.3, 8.9)$;
$\langle \overline{2.3_]}, 2.2_] \rangle \in R^s_{\Re}$ is inconsistent, simple and can also be written $[2.3, 2.2]$;
$glb_{R^b_{\Re}}(\ \langle \overline{3.2_]}, 6.7_] \rangle, \langle \overline{1.8_)}, 4.5_] \rangle\ ) = \langle \overline{3.2_]}, 4.5_] \rangle$ can also be written $[3.2, 4.5]$;
$lub_{R^b_{\Re}}(\ \langle \overline{3.2_]}, 6.7_] \rangle, \langle \overline{1.8_)}, 4.5_] \rangle\ ) = \langle \overline{1.8_)}, 6.7_] \rangle$ can also be written $(1.8, 6.7]$;
$\langle \overline{1_]}, 10_] \rangle \in R^s_{Integer}$ is consistent; $\langle \overline{1_)}, 1_] \rangle, \langle \overline{5_]}, 2_] \rangle \in R^s_{Integer}$ are inconsistent;
$\langle \overline{\{1\}_]}, \{1,3\}_] \rangle \in R^s_{Set\ Integer}$ is consistent and can also be written $[\{1\}, \{1,3\}]$;
$\langle \overline{\{1,3\}_]}, \{1\}_] \rangle, \langle \overline{\{1,3\}_]}, \{1,4\}_] \rangle \in R^s_{Set\ Integer}$ are inconsistent and simple.

Non-simple ranges (i.e., those belonging to $R^b_L \backslash R^s_L$) are constructed via operators and indexicals - i.e., overloaded functions $min/1$, $max/1$ and $val/1$ (see [Fernández and Hill, 2004]).

3.4 The constraint domain

$X \in \wp_f(\mathcal{V}_\mathcal{L})$ denotes the set of constrained variables in a CSP. Let $x \in V_L$. Then $x \sqsubseteq r$ is called an *interval constraint* for L with *constrained variable* x if $r \in R^b_L$. Also, $x \sqsubseteq r$ is simple (resp. consistent) if r is simple (resp. consistent), and non-simple (resp. inconsistent) otherwise. If $t \in L$, then $x = t$ is a shorthand for $x \sqsubseteq [\mathbf{t}, \mathbf{t}]$. The *interval constraints domain over X for L* is the set of all interval constraints for L with constrained variables in X and is denoted by $\mathcal{C}^X_L$. The union

$$\mathcal{C}^X \stackrel{\text{def}}{=} \bigcup \{\mathcal{C}^X_L \mid L \in \mathcal{L}\}$$

[3] Despite that $\overline{s}$ and t belong to different domains, s and t can be compared as, by applying the duality principle of lattices [Davey and Priestley, 1990], both s and t belong to L^s.

is called the *interval constraint domain over* X for $\mathcal{L}$. The ordering for $\mathcal{C}^X$ is inherited from the ordering in R^s_L. We define $c_1 \preceq_{\mathcal{C}^X} c_2$ if and only if, for some $L \in \mathcal{L}$, $c_1 = x \sqsubseteq r_1, c_2 = x \sqsubseteq r_2 \in \mathcal{C}^X_L$ and $r_1 \preceq_{R^s_L} r_2$. The intersection in a domain $L \in \mathcal{L}$ of two simple constraints $c_1, c_2 \in \mathcal{C}^X_L$ where $c_1 = x \sqsubseteq r_1$, $c_2 = x \sqsubseteq r_2$ and $x \in V_L$ is defined as $c_1 \cap_L c_2 = glb_{\mathcal{C}^X_L}(c_1, c_2) = x \sqsubseteq glb_{R^s_L}(r_1, r_2)$. If $S_x \subseteq \mathcal{C}^X_L$ is a set of simple constraints with constrained variable x, then we define $\bigcap_L S_x = glb_{\mathcal{C}^X_L}(S_x)$.

If $S \in \wp_f(\mathcal{C}^X)$, then S is a *constraint store* for X. If S contains only simple constraints, then it is *simple*. If S is simple, then it is *consistent* if all its constraints are consistent. The set of all simple constraint stores for X is denoted by $\mathcal{S}^X$. A constraint store S is *stable* if there is exactly one simple constraint for each $x \in X$ in S. The set of all simple stable constraint stores for X is denoted by $\mathcal{SS}^X$.

Let $S, S' \in \mathcal{SS}^X$ where c_x, c'_x denote the (simple) constraints for $x \in X$ in S and S', respectively. Then $S \preceq S'$ if and only if, for each $x \in X$, $c_x \preceq c'_x$. Let $\top_{\mathcal{SS}^X}$ be the set $\{x \sqsubseteq \bot_{R^s_L} \mid x \in X \cap V_L, L \in \mathcal{L}\}$. Then, with these definitions, $\mathcal{SS}^X$ forms a lattice.

3.5 Stabilization, propagation and solution

Let $S \in \mathcal{S}^X$, $S' \in \mathcal{SS}^X$ and, for each $x \in X$, $S_x = \{c \in S \mid c = x \sqsubseteq r\}$. Then, if $S' = \{ \bigcap_L S_x \mid L \in \mathcal{L}, x \in X \cap V_L \}$, we say that S' is the *stabilized store* of S and write $S \mapsto S'$.

Example 4. Suppose $r, w \in V_{\Re}$ and $i \in V_{Integer}$. Then $S \mapsto S'$ if, for instance,

$$\begin{aligned} S = \{\ & r \sqsubseteq \langle \overline{8.3}_), 20.4_] \rangle, \quad w \sqsubseteq \langle \overline{1.2}_], 10.5_) \rangle, \quad i \sqsubseteq \langle \overline{0}_], 10_] \rangle, \\ & r \sqsubseteq \langle \overline{1.0}_], 15.0_] \rangle, \quad w \sqsubseteq \langle \overline{5.6}_), 15.3_) \rangle, \quad i \sqsubseteq \langle \overline{2}_], 15_) \rangle\ \}, \\ S' = \{\ & r \sqsubseteq \langle \overline{8.3}_), 15.0_] \rangle, \quad w \sqsubseteq \langle \overline{5.6}_), 10.5_) \rangle, \quad i \sqsubseteq \langle \overline{2}_], 10_] \rangle\ \}. \end{aligned}$$

Constraint store stabilization defines an *ordering* in the sense that if $S, S' \in \mathcal{SS}^X$, $X' \subseteq X$, $C \in \mathcal{S}^{X'}$, and $S \cup C \mapsto S'$, then $S' \preceq S$.

Constraint propagation allows the generation of new simple interval constraints during the process of constraint solving. Interval constraints are propagated with respect to a constraint store $S \in \mathcal{SS}^X$ via an evaluation function (see [Fernández and Hill, 2004][Section 4.2] for details). Also, if $C \subseteq \mathcal{C}^X$ and $C' = \{c' \mid \exists c \in C \,.\, c$ is propagated wrt. $S \in \mathcal{SS}^X$ to $c'\}$ then we say that C *is propagated to* C' (*using* S) and write $C \rightsquigarrow^S C'$.

A solution is a constraint store that cannot be further reduced by constraint propagation. More formally, a *solution for* $C \in \wp(\mathcal{C}^X)$ is a consistent store $R \in \mathcal{SS}^X$ where $C \rightsquigarrow^R C', R \cup C' \mapsto R$. This concept establishes an *ordering of the solution wrt. the store to solve* in the sense that, if R is a solution for $C \cup S$, then, $R \preceq S$.

3.6 Precision

A generic concept of *constraint precision* was also defined. Let $\mathcal{CC}_L^X$ be the set of all consistent (and thus simple) interval constraints for L with constrained variables in X, $x \in X \cap V_L$ for any $L \in \mathcal{L}$ and $\Re\mathcal{I}$ denote the lexicographic product $(\Re^+, \mathit{Integer})$ where $\Re^+$ is the (lifted) domain of non-negative reals. Then we define

$$\begin{aligned} &\mathit{precision}_L :: \mathcal{CC}_L^X \to \Re\mathcal{I} \\ &\mathit{precision}_L\big(\, x \sqsubseteq \langle \overline{a_b}, c_d \rangle \,\big) = \big(\, \hat{a} \diamond_L c, b \diamond_B d \,\big) \end{aligned}$$

where $\diamond_L :: \big\{\, (\hat{a}, c) \;\big|\; a, c \in L, a \preceq c \,\big\} \to \Re^+$ is a (system or user defined) strict monotonic function and $\diamond_B :: B \times B \to \{0, 1, 2\}$ is the strict monotonic function

$$\begin{aligned} &\text{`]'} \diamond_B \text{`]'} \stackrel{\text{def}}{=} 2 \qquad \text{`]'} \diamond_B \text{`)'} \stackrel{\text{def}}{=} 1 \\ &\text{`)'} \diamond_B \text{`]'} \stackrel{\text{def}}{=} 1 \qquad \text{`)'} \diamond_B \text{`)'} \stackrel{\text{def}}{=} 0. \end{aligned}$$

Observe that $\mathit{precision}_L$ is defined only on consistent constraints and thus the function $\diamond_L$ only needs to be defined when its first argument is less than or equal to the second. This function must be defined for each computation domain including any fictitious top or bottom elements.

Example 5. Let *Integer*, $\Re$ and *Set Integer* and let also $\Re^2 = \langle \Re, \Re \rangle$. Suppose that $i_1, i_2 \in \mathit{Integer}$, $r_1, r_2, w_1, w_2 \in \Re$ and $s_1, s_2 \in \mathit{Set\ Integer}$ where $i_1 \preceq i_2$, $r_1 \preceq r_2$, $w_1 \preceq w_2$ and $s_1 \preceq s_2$. Then

$$\begin{aligned} \hat{i_1} \diamond_{\mathit{Integer}} i_2 &= i_2 - i_1, \\ \hat{r_1} \diamond_{\Re} r_2 &= r_2 - r_1, \\ \widehat{\langle r_1, w_1 \rangle} \diamond_{\Re^2} \langle r_2, w_2 \rangle &= +\sqrt{(r_2 - r_1)^2 + (w_2 - w_1)^2}, \\ \hat{s_1} \diamond_{\mathit{Set\ Integer}} s_2 &= \#s_2 - \#s_1. \end{aligned}$$

Assume that $i \in V_{\mathit{Integer}}$, $r \in V_{\Re}$, $y \in V_{\Re^2}$ and $s \in V_{\mathit{Set\ Integer}}$. Then

$$\begin{aligned} &\mathit{precision}_{\mathit{Integer}}(i \sqsubseteq \langle \overline{1_]}, 4_] \rangle) = (3.0, 2), \\ &\mathit{precision}_{\Re}(r \sqsubseteq \langle \overline{3.5_)}, 5.7_) \rangle) = (2.2, 0), \\ &\mathit{precision}_{\Re^2}(y \sqsubseteq \langle \overline{(2.0, 3.0)_]}, (3.4, 5.6)_] \rangle) = (2.95, 2), \\ &\mathit{precision}_{\mathit{Set\ Integer}}(s \sqsubseteq \langle \overline{\{\}_]}, \{3, 4, 5\}_) \rangle) = (3.0, 1). \end{aligned}$$

Note that the binary operators used in this example, that is, $-$ and $+$ as well as the unary operators $\#$ and 'square' need to be defined for both the lifted bounds. The unary operator 'square root' must be defined just for the lifted upper bound.

The precision of a consistent simple stable constraint store $S \in \mathcal{SS}^X$ is the sum of the precisions of each of its elements (i.e., constraints) and where the sum in $\Re\mathcal{I}$ is defined as $(a_1, a_2) + (b_1, b_2) = (a_1 + b_1, a_2 + b_2)$.

Proposition 1. *Suppose $S, S' \in \mathcal{SS}^X$ are consistent stores where $S \prec_s S'$. Then precision$(S) <_{\Re\mathcal{I}}$ precision(S').*

4 Key concepts for branching

In the rest of the paper, we continue to use L to denote any domain in $\mathcal{L}$, $X \in \wp_f(\mathcal{V}_\mathcal{L})$ the set of constrained variables, $\mathcal{C}_L^X$ the set of all interval constraints for L with constrained variables in X, $\mathcal{C}^X$ the interval constraint domain over X for $\mathcal{L}$ and $\mathcal{SS}^X$ the set of all simple stable constraint stores for X. Also $L_<$ denotes any totally ordered lattice in $\mathcal{L}$.

Notation. If $\{c_1, \ldots, c_n\} \in \mathcal{SS}^X$ and $i \in \{1 \ldots, n\}$, then

$$\{c_1, \ldots, c_n\}[c_i/c'] = \{c_1, \ldots, c_{i-1}, c', c_{i+1}, \ldots, c_n\}.$$

Definition 2. (Divisibility) Let $c = x \sqsubseteq \overline{s}, t$ be a consistent interval constraint in $\mathcal{C}_L^X$. Then, c is *divisible* if $s \neq_{L^s} t$ and *non-divisible* otherwise.

Let $S \in \mathcal{SS}^X$ be a consistent constraint store. Then S is *divisible* if there exists $c \in S$ such that c is divisible and *non-divisible* otherwise.

Thus a non-divisible constraint will have the form $x \sqsubseteq$ **[a,a]** denoting $x =$ a; hence it may be viewed as an assignment of a variable to a specific value.

Example 6. Let $x, y \in V_{Integer}$, $r, w \in V_\Re$ and $S, S' \in \mathcal{SS}^{\{x,r\}}$. Then,

$$\begin{array}{llll} x \sqsubseteq [\mathbf{1}, \mathbf{4}] & \text{and} & r \sqsubseteq (\mathbf{1.0}, \mathbf{3.2}] & \text{are divisible;} \\ y \sqsubseteq [\mathbf{2}, \mathbf{2}] & \text{and} & w \sqsubseteq [\mathbf{1.5}, \mathbf{1.5}] & \text{are non-divisible.} \end{array}$$

Also

$$\begin{array}{ll} S = \{\ x \sqsubseteq [\mathbf{1}, \mathbf{1}],\ r \sqsubseteq [\mathbf{1.0}, \mathbf{1.0}]\ \} & \text{is non-divisible;} \\ S' = \{\ x \sqsubseteq [\mathbf{1}, \mathbf{4}],\ r \sqsubseteq [\mathbf{1.0}, \mathbf{1.0}]\ \} & \text{is divisible.} \end{array}$$

Proposition 3. *Let $X \in \wp_f(V_\mathcal{L})$.*

(1) Let also $c, c' \in \mathcal{C}_L^X$ such that $c \prec_{\mathcal{C}_L^X} c'$. Then, if c is consistent, c' is divisible.

(2) Let also $S, S' \in \mathcal{SS}^X$ such that $S \prec_s S'$. Then, if S is consistent, S' is divisible.

As already mentioned, in [Fernández and Hill, 2004] we defined the concept of *solution* for a constraint store to be a consistent stable store that produces no further constraint narrowing by means of constraint propagation. In this paper we want to capture the more common meaning of *solution* as the assignment of values to variables that satisfies all the constraints. So as to distinguish the previous concept defined in [Fernández and Hill, 2004] from the concept defined in this paper, we use the term *solution* to refer the concept already defined and the term *authentic solution* to refer the new concept defined in this paper.

Definition 4. (Authentic and partial solution) Let $C \in \wp_f(\mathcal{C}^X)$ be a constraint store for X and $R \in \mathcal{SS}^X$. Then, R is an *authentic solution for* C if R is both non-divisible and a solution for C.

$R' \in \mathcal{SS}^X$ is a *partial solution for* C if there exists an authentic solution R'' for C such that $R'' \prec_s R'$. In this case we say that R' *covers* R''.

Example 7. Let $x, y \in V_{Integer}$, $X = \{x, y\}$, $C \in \wp_f(\mathcal{C}^X)$ where

$$C \;=\; \{\; x \sqsubseteq [0, max(y), \quad y \sqsubseteq min(x), 100] \;\}$$

and $S, S' \in \mathcal{SS}^X$ where

$$S \;=\; \{\; x \sqsubseteq [\mathbf{1}, \mathbf{4}],\; y \sqsubseteq [\mathbf{2}, \mathbf{5}] \;\}, \qquad S' \;=\; \{\; x \sqsubseteq [\mathbf{1}, \mathbf{1}],\; y \sqsubseteq [\mathbf{3}, \mathbf{3}] \;\}.$$

Then, S is a solution (and also a partial solution) for C whereas S' is an authentic solution for C.

The set of all authentic solutions for C is denoted as $Sol_a(C)$.

Definition 5. (Constraint store stack) A *constraint store stack for* X is a (possibly empty) finite sequence $(S_1, \ldots, S_\ell)$ of stores in $\mathcal{SS}^X$. $Stack(X)$ denotes the set of all constraint store stacks for X. The operation $push/2$ is defined for any $P \in Stack(X)$ and $S \in \mathcal{SS}^X$ as follows

$\mathbf{P}recondition : \{\; P = (S_1, \ldots, S_\ell) \;\}$
$\quad push(P, S)$
$\mathbf{P}ostcondition : \{\; P = (S_1, \ldots, S_\ell, S_{\ell+1}),\; S_{\ell+1} = S \text{ and } P \in Stack(X) \;\}.$

where the operation $top/1$ over P is defined as:

$\mathbf{P}recondition : \{P = (S_1, \ldots, S_\ell) \text{ and } \ell > 0\}$
$\quad top(P) = S$
$\mathbf{P}ostcondition : \{S = S_\ell\}.$

Let $P' = (S'_1, \ldots, S'_{\ell'}\} \in Stack(X)$ be another constraint store stack for X. Then $P \preceq_p P'$ if and only if for all $S_i \in P$ $(1 \leq i \leq \ell)$, there exists $S'_j \in P'$ $(1 \leq j \leq \ell')$ such that $S_i \preceq_s S'_j$. In this case we say that P' *covers* P.

5 Branching process

Branching [Apt, 2003] often involves two kinds of choice usually called *variable ordering* and *value ordering*. Variable ordering selects a constrained variable and value ordering splits the domain associated to the selected variable in order to introduce a choice point. In this section we explain these by describing the main functions that define them.

The *selecting function* provides a schematic heuristic for variable ordering.

Definition 6. (Selecting function) Let $S = \{c_1, \ldots, c_n\} \in \mathcal{SS}^X$. Then

$$choose :: \{S \in \mathcal{SS}^X \mid S \text{ is divisible}\} \to \mathcal{C}^X$$

is called a *selecting function for* X if $choose(S) = c_j$ where $1 \leq j \leq n$ and c_j is divisible.

Example 8. Suppose that $X = \{x_1, \ldots, x_n\}$ is a set of variables constrained respectively in $L_1, \ldots, L_n \in \mathcal{L}$ and that $S = \{c_1, \ldots, c_n\} \in \mathcal{SS}^X$ is any divisible constraint store for X where for all $i \in \{1, \ldots, n\}$, c_i is the simple interval constraint in S with constrained variable x_i. A naive strategy that selects the "left-most" divisible interval constraint in S is specified below.

$$\begin{aligned}
&\mathbf{P}recondition : \{S = \{c_1, \ldots, c_n\} \in \mathcal{SS}^X \text{ is divisible}\} \\
&\quad choose_{naive}(S) = c_j \\
&\mathbf{P}ostcondition : \{j \in \{1, \ldots, n\},\ c_j \text{ is divisible and} \\
&\qquad\qquad \forall i \in \{1, \ldots, j-1\} : c_i \text{ is non-divisible}\}.
\end{aligned}$$

When branching, some interval constraints need to be partitioned, into two or more parts, so as to introduce a choice point. We define a *splitting function* which provides a heuristic for value ordering.

Definition 7. (Splitting function) Let $L \in \mathcal{L}$ and $k > 1$. Then

$$split_L :: \mathcal{C}_L^X \to \underbrace{\mathcal{C}_L^X \times \ldots \times \mathcal{C}_L^X}_{k \text{ times}}$$

is a *k-ary splitting function for* L if, for all divisible $c \in \mathcal{C}_L^X$, $split_L(c) = (c_1, \ldots, c_k)$ satisfies the following conditions:

Completeness : $\forall c' \prec_{\mathcal{C}_L^X} c$ with c' non-divisible, $\exists i \in \{1, \ldots, k\}\ .\ c' \preceq_{\mathcal{C}_L^X} c_i$.

Contractance : $c_i \prec_{\mathcal{C}_L^X} c,\ \forall i \in \{1, \ldots, k\}$.

Example 9. Let $X = \{i, b, r, s\}$ be a set of variables where $i \in V_{Integer}$, $b \in V_{Bool}$, $r \in V_{\Re}$ and $s \in V_{Set\ Integer}$ and let[4] $i \sqsubseteq [\mathbf{a}, \mathbf{a}']$, $b \sqsubseteq [false, true]$, $r \sqsubseteq \{\mathbf{c}, \mathbf{d}\}$ and $s \sqsubseteq \{\mathbf{e}, \mathbf{f}\}$ be divisible interval constraints in $\mathcal{C}^X$ where $\mathbf{a}, \mathbf{a}' \in Integer$, $\mathbf{c}, \mathbf{d} \in \Re$ and $\mathbf{e}, \mathbf{f} \in Set\ Integer$. Then, the following functions are binary splitting functions respectively for the domains *Integer*, *Bool*, $\Re$ and *Set Integer*

$$split_{Integer}(\ i \sqsubseteq [\mathbf{a}, \mathbf{a}']) = (\ i \sqsubseteq [\mathbf{a}, \mathbf{a}],\ i \sqsubseteq [\mathbf{a} + \mathbf{1}, \mathbf{a}']\ ),$$
$$split_{Bool}(\ b \sqsubseteq [false, true]\ ) = (\ b \sqsubseteq [false, false], b \sqsubseteq [true, true]\ ),$$
$$split_{\Re}(r \sqsubseteq \{\mathbf{c}, \mathbf{d}\}) \ = \ (\ r \sqsubseteq \{\mathbf{c}, \mathbf{c}'),\ r \sqsubseteq [\mathbf{c}', \mathbf{d}\}\ ),$$
$$split_{Set\ Integer}(s \sqsubseteq \{\mathbf{e}, \mathbf{f}\}) = (\ s \sqsubseteq \{\mathbf{e}, \mathbf{f} \backslash \mathbf{g}],\ s \sqsubseteq [\mathbf{e} \cup \mathbf{g}, \mathbf{f}\}\ ).$$

Here, $split_{Integer}$ is a naive enumeration strategy in which values are chosen from left to right; $split_{Bool}$ divides the only divisible Boolean interval constraint into the two non-divisible Boolean interval constraints; $split_{\Re}$ computes the mid point $\mathbf{c}' = \frac{\mathbf{c}+\mathbf{d}}{\mathbf{2.0}}$ of the interval $[\mathbf{c}, \mathbf{d}]$; and $split_{Set\ Integer}$ is a valid splitting function for the domain of sets of integers if we define $\mathbf{g} = \{\mathbf{l}\}$ and $\mathbf{l} \in \mathbf{f} \backslash \mathbf{e}$.

Lemma 8. *Let choose/1 be a selecting function for X, $C \in \wp_f(\mathcal{C}^X)$, $S = (c_1, \ldots, c_n) \in \mathcal{SS}^X$ a divisible constraint store, $c_j = choose(S)$, $c_j \in \mathcal{C}_L^X$ for some $L \in \mathcal{L}$, $split_L/1$ a k-ary splitting function for L and $(c_{j1}, \ldots, c_{jk}) = split_L(c_j)$. Then*

(a) $\forall i \in \{1, \ldots, k\} : S[c_j/c_{ji}] \prec_s S$;

(b) if $S' \in Sol_a(C)$ and $S' \prec_s S$, then $\exists i \in \{1, \ldots, k\} : S' \preceq_s S[c_j/c_{ji}]$.

5.1 The precision map as a normalization rule

The precision map mentioned in [Section 3] also provides a way to normalize the selecting functions (i.e., the variable ordering) when the constraint system supports multiple domains.

Example 10. The well-known *first fail principle* [Haralick and Elliot, 1980] often chooses the variable constrained with the smallest domain. However, in systems supporting multiple domains it is not always clear which is the smallest domain (particularly if there are several infinite domains). In our framework, one way to "measure" the size of the domains is to use the precision map defined on each computation domain.

[4] Observe that in the integer and Boolean domains only intervals with close brackets are considered since open brackets can always be transformed to close brackets (see [Fernández and Hill, 2004][Section 8.1]) e.g., $i \sqsubseteq (\mathbf{1}, \mathbf{8})$ is equivalent to $i \sqsubseteq [\mathbf{2}, \mathbf{7}]$. Note also that in the Boolean domain there is just one unique case of divisible interval constraint (i.e., $b \sqsubseteq [false, true]$) and thus only this case is considered in the definition of $split_{Bool}$.

For instance, suppose that $X = \{x_1, \ldots, x_n\}$ is a set of variables constrained, respectively, in $L_1, \ldots, L_n \in \mathcal{L}$ and that $S = \{c_1, \ldots, c_n\} \in \mathcal{SS}^X$ is any divisible constraint store for X where for each $i \in \{1, \ldots, n\}$, c_i is the simple interval constraint in S with constrained variable x_i. Here the first fail principle can be emulated by defining *choose*/1 to select the interval constraint with the smallest precision. We denote this procedure by $choose_{ff}$.

P*recondition* : $\{S = \{c_1, \ldots, c_n\} \in \mathcal{SS}^X$ is divisible$\}$
$choose_{ff}(S) = c_j$
P*ostcondition* : $\{j \in \{1, \ldots, n\}, c_j$ is divisible and
$\forall i \in \{1, \ldots, n\} \backslash \{j\} : c_i$ divisible $\implies$ $precision_{L_j}(c_j) \leq_{\Re\mathcal{I}} precision_{L_i}(c_i)\}$.

Observe that it is straightforward to include more conditions e.g., in case of ties, if c_i, c_k, c_j have the same (minimum) precision, the "left-most" constraint can be chosen i.e., $c_{minimum(i,k,j)}$.

6 Branching operational schema

In this section, we continue to use L to denote any domain in $\mathcal{L}$, $X \in \wp_f(\mathcal{V}_\mathcal{L})$ the set of constrained variables, $\mathcal{C}^X$ the set of all interval constraint domain for X and $\mathcal{SS}^X$ the set of all simple stable constraint stores for X.

In [Fernández and Hill, 2004], *solve*/2, a generic *operational schema* for interval constraint propagation that computes a solution (if it exists) for $C \cup S$ is defined. To guarantee termination, an extended schema $solve_\varepsilon/2$, where $\varepsilon \in \Re^+$ is also defined. The schema $solve_\varepsilon(C, S)$ where $C \in \wp_f(\mathcal{C}^X)$ and $S \in \mathcal{SS}^X$ is given in [Figure 1] where c_x, c'_x denote, respectively, the consistent constraints for $x \in X$ in S and S'.

6.1 $solve_\varepsilon/2$: new properties

Before describing the branching schema, we state here some additional properties of the schema $solve_\varepsilon/2$ that were not stated in [Fernández and Hill, 2004].

Lemma 9. *(More properties of* $solve_\varepsilon/2$*) Let* $C \in \wp_f(\mathcal{C}^X)$, $S, S^f \in \mathcal{SS}^X$ *and* $\varepsilon \in \Re^+ \cup \{0.0\}$. *Suppose that* S^f *is the value of the constraint store* S *after a terminating execution of* $solve_\varepsilon(C, S)$. *Then,*

(a) $S^f \preceq_s S$;

(b) $\forall R \in Sol_a(C \cup S) : R \preceq_s S^f$;

(c) If $Sol_a(C \cup S)$ *is not empty and* S^f *is non-divisible then* $S^f \in Sol_a(C \cup S)$;

procedure $solve_{\varepsilon}(C, S)$
begin
  if S *is consistent then* (0)
    $C := C \cup S$; (1)
    repeat
      $C \rightsquigarrow^{S} C'$; %% Constraint Propagation (2)
      $S' := S$; (3)
      $S' \cup C' \mapsto S$; %% Store stabilization (4)
    until S *is inconsistent or*
        $(\forall x \in X \cap V_L : precision_L(c'_x) - precision_L(c_x) \preceq (\varepsilon, 0)$; (5)
  endif;
endbegin.

Figure 1: *solve*/2: a generic schema for interval constraint propagation

(d) If $\varepsilon = 0.0$ *and* S^f *is non-divisible then* $S^f \in Sol_a(C \cup S)$.

Property (a) makes sure that the propagation procedure never gains values, property (b) guarantees that no solution covered by a constraint store is lost in the propagation process and properties (c) and (d) ensure the computed answers are correct[5].

6.2 Branching Schema

We now describe a generic schema for branching, complementary to $solve(C, S)$, that will provide completeness for the interval constraint solver.

There are a number of values and subsidiary procedures that are assumed to be defined externally to the main branch procedure:

- a selecting function *choose*/1 for X;
- a k-ary splitting function $split_L$ for each domain $L \in \mathcal{L}$ (for some integer $k > 1$);
- a precision map for each $L \in \mathcal{L}$;

[5] In [Fernández and Hill, 2004], we 'only' assured that, if a solution exist, the final state of constraint store S contains the most general solution (that probably is not an authentic solution).

– a constraint store stack P for X.

It is assumed that the external procedures have an implementation that terminates for all possible values.

The branching schema is given in [Figure 2]. This requires the following parameters: a finite set $C \in \wp_f(\mathcal{C}^X)$ of interval constraints to be solved, a constraint store $S \in \mathcal{SS}^X$, a bound $p \in \Re\mathcal{I}$ and also requires a non-negative real bound α.

procedure $branch_\alpha(C, S, p)$
begin
  $solve_\varepsilon(C, S)$; (1)
  if S is consistent then (2)
    if S is non-divisible or ($p < \top_{\Re\mathcal{I}}$ and $p - precision(S) \leq (\alpha, 0)$) then (3)
      $push(P, S)$; (4)
    else (5)
      $c_j \leftarrow choose(S)$; (6)
      $(c_{j1}, \ldots, c_{jk}) \leftarrow split_{L_j}(c_j)$, where $c_j \in \mathcal{C}^X_{L_j}$ and $L_j \in \mathcal{L}$; (7)
      $branch_\alpha(C, S[c_j/c_{j1}], precision(S)) \;\vee$
      $\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \;\vee$
      $branch_\alpha(C, S[c_j/c_{jk}], precision(S));$ } %% Choice Points (8)
    endif;
  endif;
end.

Figure 2: $branch_\alpha/3$: a generic schema for interval constraint solving

Theorem 10. *(Properties of the $branch_\alpha/3$ schema) Let $C \in \wp_f(\mathcal{C}^X)$, $S \in \mathcal{SS}^X$, $\varepsilon, \alpha \in \Re^+$ and $p = \top_{\Re\mathcal{I}}$. Then, the following properties are guaranteed:*

1. Termination*: if $\alpha > 0.0$ and the procedure $solve_\varepsilon/2$ terminates for all values*[6] *then $branch_\alpha(C, S, p)$ terminates;*

[6] Observe that termination of this procedure is always guaranteed if $\varepsilon > 0.0$ -see Theorem 2 in [Fernández and Hill, 2004].

2. Completeness: *if* $\alpha = 0.0$ *and the execution of* $branch_\alpha(C, S, p)$ *terminates, then the final state for the stack* P *contains all the authentic solutions for* $C \cup S$*;*

3. Approximate completeness: *if the execution of* $branch_\alpha(C, S, p)$ *terminates and* $R \in Sol_a(C \cup S)$*, then the final state for the stack* P *contains either* R *or a partial solution* R' *that covers* R*.*

4. Correctness: *if* $\alpha = 0.0$ *and* $\varepsilon = 0.0$*, the stack* P *is initially empty and the execution of* $branch_\alpha(C, S, p)$ *terminates with* R *in the final state of* P*, then* $R \in Sol_a(C \cup S)$*.*

5. Approximate correctness or control on the result precision: *If* P_{α_1} *and* P_{α_2} *are non-empty constraint store stacks for* X *resulting from any terminating execution of* $branch_\alpha(C, S, p)$ *(where initially* P *is empty) when* α *has the values* α_1 *and* α_2*, respectively, and* $\alpha_1 < \alpha_2$ *then*

$$P_{\alpha_1} \preceq_p P_{\alpha_2}.$$

(In other words, the set of (possibly partial) solutions in the final state of the stack is dependent on the value of α *in the sense that lower* α*, better the set of solutions.)*

Observe that, just as for the bound ε in the $solve_\varepsilon/2$ procedure, the bound α also guarantees termination and allows the precision of the results to be controlled. Note also that this schema can be used for any set of computation domains for which a splitting function and precision map are defined.

Example 11. Let $\mathcal{L} = \{Integer, \Re, Set\ Integer\}$. Instances of the $branch_\alpha/3$ schema can be specified for each $L \in \mathcal{L}$. These are defined by instantiating the subsidiary procedures specified in [Section 6.2]. For instance, assuming as selecting function $choose_{ff}$ as defined in Example 10, instances are defined by considering the definitions for $\diamond_L/2$ (that constructs the precision map) and $split_L$ of Examples 5 and 9 respectively.

7 Interval constraint optimization

The schema in [Figure 2] can be adapted to solve COPs by means of three new subsidiary functions.

Definition 11. (Subsidiary functions and values) Let $L_< \in \mathcal{L}$ be a totally ordered domain[7]. Then we define

[7] Normally $L_<$ would be $\Re$.

- a *cost function*, $fcost :: \mathcal{SS}^X \rightarrow L_<$;
- an *ordering relation*, $\diamond :: L_< \times L_< \in \{>, <, =\}$;
- a *cost bound*, $\delta \in L_<$.

Then the *extended branching schema*, $branch_{\alpha+}/3$, is obtained from $branch_\alpha/3$ by replacing Line 4 in [Figure 2] with:

$$\begin{array}{ll} & \text{if } fcost(S) \diamond \delta \text{ then} \\ & \quad \delta \leftarrow fcost(S); \\ & \quad push(P, S); \\ & \text{endif;} \end{array} \qquad (4^*)$$

Theorem 12. *(Properties of the $branch_{\alpha+}/3$ schema) Let $C \in \wp_f(\mathcal{C}^X)$, $S \in \mathcal{SS}^X$, $\varepsilon, \alpha \in \Re^+$ and $p = \top_{\Re\mathcal{I}}$. Suppose that the procedure $solve_\varepsilon/2$ terminates for all values*[8]*. Then, the following properties are guaranteed:*

1. Termination*: if $\alpha > 0.0$ then the execution of $branch_{\alpha+}(C, S, p)$ terminates;*
2. *If fcost is a constant function with value δ and $\diamond$ is $=$, then all properties shown in Theorem 10 hold for the execution of $branch_{\alpha+}(C, S, p)$.*
3. Soundness on optimization*: If at least one authentic solution with a cost higher than $\bot_{L_<}$ (resp. lower than $\top_{L_<}$) exists for $C \cup S$, $\alpha = 0.0$, $\diamond$ is $>$ (resp. $<$), $\delta = \bot_{L_<}$ (resp. $\top_{L_<}$), the stack P is initially empty and the execution of $branch_{\alpha+}(C, S, p)$ terminates with P non-empty, then the element on the top of P is the first authentic solution found that maximizes (resp. minimizes) the cost function.*

Unfortunately, if $\alpha > 0.0$, we cannot guarantee that the top of the stack contains an authentic solution or even a partial solution for the optimization problem. However, if the cost function $fcost/1$ is monotonic, solutions can be compared.

Theorem 13. *(Approximate soundness) Suppose that, for $i \in \{1, 2\}$, P_{α_i} is the constraint store stack resulting from the execution of $branch_{\alpha_i+}(C, S, p)$ where $\alpha_i \in \Re^+ \cup \{0.0\}$. Then, if $\alpha_1 < \alpha_2$ the following property hold.*

If P_{α_1} and P_{α_2} are not empty, and $top(P_{\alpha_2})$ is an authentic solution or covers a solution for $C \cup S$, then, if $fcost/1$ is monotonic and $\diamond$ is $<$ (i.e., a minimization problem),

$$fcost(top(P_{\alpha_1})) \preceq_{L_<} fcost(top(P_{\alpha_2})),$$

[8] Again note that termination of this procedure is always guaranteed if $\varepsilon > 0.0$ -see Theorem 2 in [Fernández and Hill, 2004].

and, if fcost/1 *is anti-monotone and* $\diamond$ *is* $>$ *(i.e.,a maximization problem),*

$$fcost(top(P_{\alpha_1})) \succeq_{L_<} fcost(top(P_{\alpha_2})).$$

A direct consequence of this theorem is that by using a(n) (anti-)monotone cost function, the lower α is, the better the (probable) solution is.

Observe that it is straightforward to transform the schema $branch_{\alpha+}/3$ into a $B\&B$ schema by adding, before Line 1 in [Figure 2], a test such as

$$\text{if } fcost(S) \diamond \delta \text{ then } \ldots$$

With this addition, branches in the search tree that have not the possibility to improve the best solution found up to that point are not considered for further branching. This $B\&B$ schema clearly depends on the definition of the function *fcost*/1; this should return the highest (resp. lowest) cost possible for its argument (i.e., considering all combinations of all the values that the constrained variables can take in the store) in the solving of a maximization (resp. minimization) problem.

8 The schema parameters

In this section, we show how the choice of the parameters in the definition of $branch_\alpha/3$ determines the method of solving for a set of interval constraints i.e., the schema $branch_\alpha/3$ allows a set of interval constraints to be solved in many different ways, depending on the values for *fcost*, δ and $\diamond$.

Theorem 12(2) has shown that to solve classical CSPs, *fcost* should be defined as the constant function[9] δ and the parameter $\diamond$ should have the value $=$. Moreover, Theorem 12(3) has shown that a CSP is solved as a COP by instantiating $\diamond$ as either $>$ (for maximization problems) or $<$ (for minimization problems). In all cases, the value δ should be instantiated to the initial cost value from which an optimal solution must be found. Some possible instantiations are summarized in [Table 1] where [Column 1] indicates the type of CSP, [Column 2] gives any conditions on the cost function, [Column 3] gives the range of the cost function (usually, this is $\Re$), [Column 4] gives the initial definition of the $\diamond$ operator, and [Column 5] displays the initial value for δ.

The schema also permits a mix of maximization and minimization criteria (or even to give priority to some criteria over others). This is the case (see [Row 4] of [Table 1]) when $L_<$ is a compound domain and the ordering in $L_<$ determines how the COP will be solved.

[9] Usually $\delta \in \Re$.

CSP Type	*fcost*	$L_<$	$\diamond$	δ
Classical CSP	constant	$\Re$	$=$	$fcost(S)$
Typical Minimization COP	any cost function	$\Re$	$<$	$\top_{\Re}$
Typical Maximization COP	any cost function	$\Re$	$>$	$\bot_{\Re}$
Max-Min COP	any cost function	$\Re \times \Re$	$<$	$\top_{\Re\times\Re}$

Table 1: CSP type depends on parameters instantiation

8.1 An example

The following example illustrates the flexibility of the schema to solve a set of interval constraints in different ways. Let $X = \{x_1, x_2, x_3, x_{12}, x_{123}\} \subset V_{Integer}$ and the following set of constraints defined on X

$$\begin{aligned} C = \{\ & x_1 + x_2 + x_3 \leq 1, \\ & x_1 \leq 1,\ x_2 \leq 1,\ x_3 \leq 1, \\ & x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0\ \}. \end{aligned}$$

A way to define these constraints is via the high level constraints $\leq/2$ and $plus/3$ as defined in [Fernández and Hill, 2004][Example 12] (for $L = Integer$) so that C can be coded as

$$\begin{aligned} \{\ & plus(x_1, x_2, x_{12}),\ plus(x_{12}, x_3, x_{123}),\ x_{123} \leq 1, \\ & x_1 \leq 1,\ x_2 \leq 1,\ x_3 \leq 1, \\ & 0 \leq x_1,\ 0 \leq x_2,\ 0 \leq x_3\ \} \end{aligned}$$

Observe that due to the plus constraint definition two variables $x_{12}, x_{123} \in V_{Integer}$ have been added. However these additions can be easily reduced by considering alternative definitions e.g., it is possible to declare a plus constraint with four arguments as $plus(x, y, z, w) \equiv plus(x, y, xy),\ plus(xy, z, w)$.

Consider also the following cost functions

$$fcost_1, fcost_2 :: \mathcal{SS}^X \to \Re, \qquad fcost_3, fcost_4 :: \mathcal{SS}^X \to \Re^2,$$

defined for each $S = \{x_1 \sqsubseteq r_1,\ x_2 \sqsubseteq r_2,\ x_3 \sqsubseteq r_3,\ x_{12} \sqsubseteq r_{12},\ x_{123} \sqsubseteq r_{123}\}$ in $\mathcal{SS}^X$ as follows

$$\begin{aligned} & fcost_1(S) = 1.0; && \%\% \text{ Constant function} \\ & fcost_2(S) = mid(r_1) + mid(r_2) + mid(r_3); && \%\% \ x_1 + x_2 + x_3 \\ & fcost_3(S) = (fcost_2(S), mid(r_1) + mid(r_3)); && \%\% \ (x_1 + x_2 + x_3, x_1 + x_3) \\ & fcost_4(S) = (fcost_2(S), mid(r_2) + mid(r_3)); && \%\% \ (x_1 + x_2 + x_3, x_2 + x_3) \end{aligned}$$

where $mid(\{a,b\})$ is a function from $R^s_{\Re}$ to $\Re$ that returns the mid point in the range $\{a,b\}$ i.e., $mid(\{a,b\}) = \frac{a+b}{2.0}$ (e.g., $mid([\mathbf{1.0,4.0}]) = 2.5$).

[Table 2] shows different instantiations of the schema $branch_\alpha(C,S,p)$ with $\alpha = 0.0$ and $\varepsilon = 0.0$, $choose_{naive}$ as defined in Example 8 and $split_{Integer}$ as defined in Example 9. It is assumed that initially $p = \top_{\Re\mathcal{I}}$, the global stack P is empty and S is the top element of $\mathcal{SS}^X$. In [Table 2]:

- column 1 indicates the way in which the CSP is solved (where Max-Min means that we have mixed criterias for the optimization);
- column 2 indicates the initial value of δ;
- column 3 indicates the cost function;
- column 4 indicates the $\diamond$ relation;
- column 5 indicates where, in the final state of the stack P, the authentic solution(s) is (are) positioned; and
- column 6 references the subfigure of [Figure 3] that shows the final state of the stack P[10].

Note that since the integer domain is finite, termination is guaranteed even if $\alpha = 0.0$ and $\varepsilon = 0.0$.

CSP Type	δ	Cost function	$\diamond$	Solution	Figure
Classical CSP	1.0	$fcost_1$	$=$	Any in the stack	3(a)
Maximization COP	$\bot_{\Re}$	$fcost_2$	$>$	stack top	3(b)
Minimization COP	$\top_{\Re}$	$fcost_2$	$<$	stack top	3(c)
Max-Min COP (i)	$(\bot_{\Re}, \top_{\Re})$	$fcost_3$	$<_1$	stack top	3(d)
Max-Min COP (ii)	$(\bot_{\Re}, \top_{\Re})$	$fcost_4$	$<_1$	stack top	3(e)
Max-Min COP (iii)	$(\bot_{\Re}, \top_{\Re})$	$fcost_3$	$<_2$	stack top	3(f)
Max-Min COP (iv)	$(\bot_{\Re}, \top_{\Re})$	$fcost_4$	$<_2$	stack top	3(g)

Table 2: Different solvings of the CSP

In [Table 2], each execution of the schema gives rise to a different way of solving $C \cup S$. In particular, [Row 1] indicates how to solve the problem as a

[10] In [Figure 3], (a,b,c) denotes a constraint store

$$S = \{x_1 \sqsubseteq [a,a],\ x_2 \sqsubseteq [b,b],\ x_3 \sqsubseteq [c,c],\ x_{12} \sqsubseteq [d,d],\ x_{123} \sqsubseteq [e,e]\ \}$$

where d and e are arbitrary integers. For instance, $(0,1,0)$ denotes the constraint store $S = \{x_1 \sqsubseteq [0,0], x_2 \sqsubseteq [1,1], x_3 \sqsubseteq [0,0], \ldots\}$. In addition, in the subfigures of [Figure 3], the value to the right of each element (a,b,c) denotes its cost.

Solution S	$fcost_1(S)$	$fcost_2(S)$	$fcost_3(S)$	$fcost_4(S)$
(1,0,0)	1.0	1.0	(1.0,1.0)	(1.0,0.0)
(0,1,0)	1.0	1.0	(1.0,0.0)	(1.0,1.0)
(0,0,1)	1.0	1.0	(1.0,1.0)	(1.0,1.0)
(0,0,0)	1.0	0.0	(0.0,0.0)	(0.0,0.0)

Table 3: Evaluation of the solutions to the problems

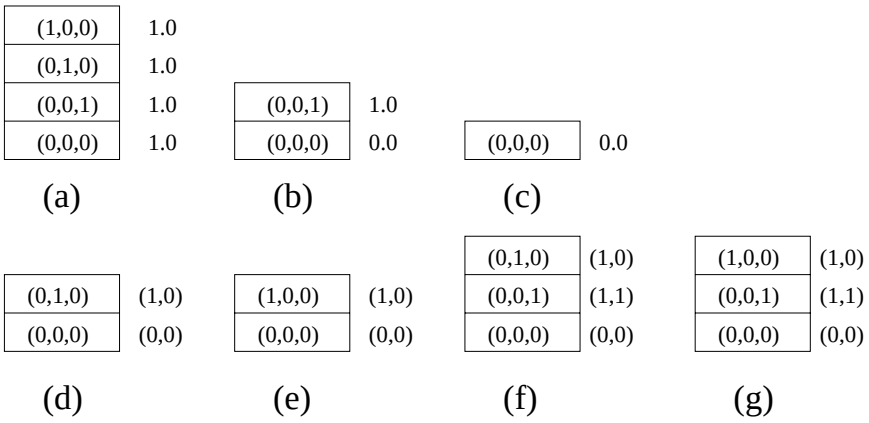

Figure 3: Final state of global stack P for different CSP solvers

classical CSP. Here *fcost* is a constant function with value δ (where δ is 1.0) and $\diamond$ is $=$. In this case, all authentic solutions are pushed on the stack (see [Figure 3(a)]) as stated in Theorem 12(2) (see also Theorem 10(2)). [Rows 2-3] show how the problem can be solved by maximizing and minimizing the function $fcost_2$ respectively. The optimal solution is that on the top of the stack (see [Figures 3(b) and 3(c)]). [Rows 4-7] indicate how to mix optimization criteria where $<_1$ and $<_2$ are defined on $\Re^2$ as follows:

$$(a,b) <_1 (c,d) \iff (a \geq c \wedge b < d) \vee (a > c \wedge b \leq d);$$
$$(a,b) <_2 (c,d) \iff a > c \vee a = c \wedge b < d.$$

[Row 4] corresponds to the problem of maximizing $x_1 + x_2 + x_3$ and minimizing $x_1 + x_3$; [Row 5] corresponds to maximizing $x_1 + x_2 + x_3$ and minimizing $x_2 + x_3$; [Row 6] corresponds to first maximizing $x_1 + x_2 + x_3$ and, if this cannot be further optimized, then minimizing $x_1 + x_3$ (this is consequence of the ordering $<_2$ that gives priority to the maximization of the first component over the minimization of the second one); and [Row 7] does the same but minimizing $x_2 + x_3$. [Figure 3] shows the final state of the global stack for each of these cases. [Table 3] shows the cost of each authentic solution with respect to the

four cost functions considered in this example. Note that, in all the cases, we have assumed a (left-to-right) depth-first strategy for tree traversal.

By defining alternative orderings on $\Re^2$, problems involving other mixed optimization criterias can also be solved.

9 Concluding Remarks

This paper is an attempt to find general principles for branching in interval constraint solving. The branching schema provided here is a generic schema for solving sets of interval constraints on finite and continuous domains as well on combined domains and it is useful to prove and devise generic properties of interval constraint solving.

Our branching schema generalizes the well-known split-and-solve method of the CLP(BNR) system [Benhamou and Older, 1997] to any domain with lattice structure what means that it is valid both for classical domains (i.e., real, integers, Boolean and sets) and for new (possibly combined) domains. In this generalization, we propose an interval branching schema that extends the generic and cooperative interval propagation schema described in [Fernández and Hill, 2004]. This extension provides a generic schema for interval constraint solving that allows problems defined on any set of lattices to be solved in terms of interval constraints.

To achieve this, we have first defined the concept of authentic solution as an assignment of values to variables that satisfies all the constraints. Then, by using a schematic formulation for the branching process, we have indicated which properties of the main procedures involved in branching are responsible for the key properties of interval constraint solving. Then we have extended the schema for optimization and have shown that, in some cases, the methods for solving CSPs depend on the ordering of the range of the cost functions.

Key properties such as correctness and completeness are proved and by means of a *precision map* similar to that defined for the propagation schema described in [Fernández and Hill, 2004], we have shown that termination may be guaranteed. We have shown by example how the precision map is a means to normalize the heuristic for variable ordering on systems supporting multiple domains (e.g., cooperative systems).

We have indicated how the branching schema can be adapted as a $B\&B$ schema. The branching schema can also be used for most existing constraint domains (finite or continuous) and, as for the propagation framework described in [Fernández and Hill, 2004], is also applicable to multiple domains and cooperative systems. The behavior of several existing interval constraint systems such as clp(FD) [Codognet and Diaz, 1996], clp(B) and clp(B/FD) [Codognet and Diaz, 1994], DecLic [Goualard et al., 1999], CLIP [Hickey, 2000],

Conjunto [Gervet, 1997] or CLP(BNR) [Benhamou and Older, 1997] can be explained as an instance of the schema presented in this paper.

Acknowledgements

The first author was partially supported by Spanish MCyT under contracts TIN2004-7943-C04-01 and TIN2005-08818-C04-01.

References

[Appleget and Wood, 2000] Appleget, J. and Wood, R. (2000). Explicit-constraint branching for solving mixed-integer programs. In Laguna, M. and González, J., editors, *Computing Tools for Modeling, Optimization and Simulation*, pages 245–262, Boston. Kluwer Academic Publishers.

[Apt, 1999] Apt, K. (1999). The essence of constraint propagation. *Theoretical Computer Science*, 221(1-2):179–210.

[Apt, 2000] Apt, K. (2000). The role of commutativity in constraint propagation algorithms. *ACM Transactions on Programming Languages and Systems*, 22(6):1002–1036.

[Apt, 2003] Apt, K. (2003). *Principles of constraint programming*. Cambridge University Press.

[Apt, 2005] Apt, K. R. (2005). Explaining constraint programming. In Middeldorp, A., van Oostrom, V., van Raamsdonk, F., and de Vrijer, R., editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 55–69. Springer.

[Apt and Monfroy, 2001] Apt, K. R. and Monfroy, E. (2001). Constraint programming viewed as rule-based programming. *Theory and Practice of Logic Programming*, 1(6):713–750.

[Benhamou and Older, 1997] Benhamou, F. and Older, W. (1997). Applying interval arithmetic to real, integer and Boolean constraints. *The Journal of Logic Programming*, 32(1):1–24.

[Bistarelli, 2004] Bistarelli, S. (2004). *Semirings for Soft Constraint Solving and Programming*. Springer.

[Bistarelli et al., 2000] Bistarelli, S., Codognet, P., Georget, Y., and Rossi, F. (2000). Labeling and partial local consistency for soft constraint programming. In Pontelli, E. and Costa, V. S., editors, *Second International Workshop on Practical Aspects of Declarative Languages (PADL 2000)*, volume 1753 of *Lecture Notes in Computer Science*, pages 230–248. Springer.

[Bistarelli et al., 2002] Bistarelli, S., Frühwirth, T., and Marte, M. (2002). Soft constraint propagation and solving in CHRs. In *The 2002 ACM Symposium on Applied Computing (SAC)*, pages 1–5, Madrid, Spain. ACM.

[Bistarelli et al., 1999] Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., and Fargier, H. (1999). Semiring-based CSPs and valued CSPs: frameworks, properties and comparison. *Constraints*, 4(3):199–240.

[Codognet and Diaz, 1994] Codognet, P. and Diaz, D. (1994). clp(B): combining simplicity and efficiency in Boolean constraint solving. In *6th International Symposium on Programming Languages Implementation and Logic Programming (PLILP'94)*, number 844 in LNCS, pages 244–260, Madrid, Spain. Springer-Verlag.

[Codognet and Diaz, 1996] Codognet, P. and Diaz, D. (1996). Compiling constraints in clp(FD). *The Journal of Logic Programming*, 27(3):185–226.

[Davey and Priestley, 1990] Davey, B. and Priestley, H. (1990). *Introduction to lattices and order*. Cambridge University Press, Cambridge, England.

[Fernández and Hill, 2004] Fernández, A. J. and Hill, P. (2004). An interval constraint system for lattice domains. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(1):1–46.

[Freuder and Hubbe, 1995] Freuder, E. and Hubbe, P. (1995). Extracting constraint satisfaction subproblems. In *14th International Joint Conference on Artificial Intelligent (IJCAI'95)*, pages 548–557, Québec, Canada. Morgan Kaufman.

[Freuder and Wallace, 1992] Freuder, E. and Wallace, R. (1992). Partial constraint satisfaction. *Artificial Intelligence*, 58(21-70):21–70.

[Gervet, 1997] Gervet, C. (1997). Interval propagation to reason about sets: definition and implementation of a practical language. *Constraints*, 1(3):191–244.

[Goualard et al., 1999] Goualard, F., Benhamou, F., and Granvilliers, L. (1999). An extension of the WAM for hybrid interval solvers. *The Journal of Functional and Logic Programming*, 1999(1):1–36. Special issue of Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages.

[Haralick and Elliot, 1980] Haralick, R. and Elliot, G. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313.

[Harvey and Ginsberg, 1995] Harvey, W. D. and Ginsberg, M. L. (1995). Limited discrepancy search. In *14th International Joint Conference on Artificial Intelligent (IJCAI'95)*, pages 607–615, Québec, Canada. Morgan Kaufman.

[Hickey, 2000] Hickey, T. (2000). CLIP: a CLP(Intervals) dialect for metalevel constraint solving. In Pontelli, E. and Costa, V., editors, *2nd International Workshop on Practical Aspects of Declarative Languages (PADL'2000)*, number 1753 in LNCS, pages 200–214, Boston, USA. Springer-Verlag.

[Kondrak and Van Beek, 1997] Kondrak, G. and Van Beek, P. (1997). A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, 89(1-2):365–387.

[Kumar, 1992] Kumar, V. (1992). Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 13(1):32–44.

[Meseguer and Larrosa, 1995] Meseguer, P. and Larrosa, J. (1995). Constraint satisfaction as global optimization. In *14th International Joint Conference on Artificial Intelligent (IJCAI'95)*, pages 579–585, Québec, Canada. Morgan Kaufman.

[Monfroy, 1996] Monfroy, E. (1996). *Solver collaboration for constraint logic programming*. PhD thesis, Centre de Recherche en Informatique de Nancy, INRIA-Lorraine.

[Monfroy et al., 1995] Monfroy, E., Rusinowitch, M., and Schott, R. (1995). Implementing non-linear constraints with cooperative solvers. Research Report 2747, Centre de Recherche en Informatique de Nancy, INRIA-Lorraine.

[Nadel, 1989] Nadel, B. (1989). Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224.

[Older and Benhamou, 1993] Older, W. and Benhamou, F. (1993). Programming in CLP(BNR). 1st International Workshop on Principles and Practice of Constraint Programming (PPCP'93), Informal Proceedings, pages: 228-238, Brown University, Newport, Rode Island.

[Park, 2006] Park, V. (2006). An empirical study of different branching strategies for constraint satisfaction problems. Master thesis, University of Waterloo, Waterloo, Ontario, Canada.

[Pedamallu et al., 2006] Pedamallu, C., Özdamar, L., and T.Csendes (2006). An interval partitioning approach for continuous constrained optimization. In *Models and Algorithms in Global Optimization*. Springer. Accepted for publication.

[Perregaard, 2003] Perregaard, M. (2003). *Generating Disjunctive Cuts for Mixed Integer*. PhD thesis, Graduate School of of Industrial Administration, Carnegie Mellon, Pittsburgh, PA.

[Ruttkay, 1998] Ruttkay, Z. (1998). Constraint satisfaction-a survey. *CWI Quaterly*, 11(2-3):163–214.

[Schiex et al., 1995] Schiex, T., Fargier, H., and Verfaillie, G. (1995). Valued constraint satisfaction problems: hard and easy problems. In *14th International Joint Conference on Artificial Intelligent (IJCAI'95)*, pages 631–637, Québec, Canada. Morgan Kaufman.

[Smith, 1995] Smith, B. (1995). A tutorial on constraint programming. Research Report 95.14, University of Leeds, School of Computer Studies, England.

[Van Hentenryck, 1989] Van Hentenryck, P. (1989). *Constraint satisfaction in logic programming*. The MIT Press, Cambridge, MA.

[Van Hentenryck, 1995] Van Hentenryck, P. (1995). Constraint solving for combinatorial search problems: a tutorial. In Montanari, U. and Rossi, F., editors, *1st International Conference on Principles and Practice of Constraint Programming (CP'95)*, number 976 in LNCS, pages 564–587, Cassis, France. Springer-Verlag.

[Van Hentenryck et al., 1992] Van Hentenryck, P., Deville, Y., and Teng, C. (1992). A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2-3):291–321.

[van Hoeve, 2006] van Hoeve, W. (2006). Exploiting semidefinite relaxations in constraint programming. *Computers and Operations Research*, 33(10):2787–2804.

[van Hoeve and Milano, 2004] van Hoeve, W. and Milano, M. (2004). Postponing branching decisions. In de Mántaras, R. L. and Saitta, L., editors, *16th European Conference on Artificial Intelligence (ECAI'2004)*, pages 1105–1106, Valencia, Spain. IOS Press. Extended version available in `http://www.cs.cornell.edu/~vanhoeve/papers/postpone.pdf`.

[Vu et al., 2005] Vu, X.-H., Silaghi, M.-C., Sam-Haroud, D., and Faltings, B. (2005). Branch-and-prune search strategies for numerical constraint solving.

[Wallace, 1993] Wallace, R. (1993). Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs. In Bajcsy, R., editor, *13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 239–247, Chambéry, France. Morgan Kaufmann.

Appendix: Proofs

Proof. (Proposition 1 on page 11) This is a direct consequence of the result in [Fernández and Hill, 2004][Proposition 7] that $precision_L$ is strict monotonic for any L. □

Proof. (Proposition 3 on page 11) We prove the cases separately.

Case (1). Suppose that $c = x \sqsubseteq r$ and $c' = x \sqsubseteq r'$, where $r = \overline{s}, t$ and $r' = \overline{s'}, t'$. By hypothesis c is consistent and thus r is consistent and also $r \prec_{R^s_L} r'$. By ordering on ranges, then r' is consistent and thus c' is consistent. Therefore, by definition of range

$$s \preceq_{L^s} t \wedge s' \preceq_{L^s} t'.$$

Moreover, as $r \prec_{R^s_L} r'$, by the ordering on ranges and the definition of simple interval domain,

$$\overline{s} \prec_{\overline{L^s}} \overline{s'} \wedge t \preceq_{L^s} t' \vee$$
$$\overline{s} \preceq_{\overline{L^s}} \overline{s'} \wedge t \prec_{L^s} t'.$$

It follows by the duality principle for lattices [Davey and Priestley, 1990] that

$$s' \prec_{L^s} t'.$$

Therefore, by Definition 2, c' is divisible.

Case (2). By hypothesis S is consistent and thus, by the ordering on $\mathcal{SS}^X$ S' and definition of consistent simple stable constraint store, is consistent. Moreover, for all $c \in S$, c is consistent and also, there exists $c' \in S'$ and $c \in S$ such that $c \prec_{\mathcal{C}^X} c'$. By Proposition 3(1), c' is divisible. Therefore, by Definition 2, S' is divisible.

□

Proof. (Lemma 8 on page 14) We prove cases separately.

Case (a). By Definition 6, c_j is divisible and, by the contractance property shown in Definition 7, for all $i \in \{1, \ldots, k\}$ $c_{ji} \prec_{\mathcal{C}_L^X} c_j$. Therefore, by the ordering on $\mathcal{SS}^X$, for all $i \in \{1, \ldots, k\}$ $S[c_j/c_{ji}] \prec_s S$.

Case (b). By Definition 4, $S' \in \mathcal{SS}^X$. Suppose that c_j is constrained on some variable $x \in V_L$ ($x \in X$) and let c'_j be the simple interval constraint for x in S'. Thus, by the ordering on $\mathcal{SS}^X$, $c'_j \preceq_{\mathcal{C}_L^X} c_j$. Moreover, by Definition 4, S' is non-divisible and thus, by Definition 2, c'_j is non-divisible. Also, by Definition 6, c_j is divisible so that $c'_j \neq c_j$ and thus $c'_j \prec_{\mathcal{C}_L^X} c_j$. As consequence, by the completeness property of the splitting functions shown in Definition 7,

$$\exists i \in \{1, \ldots, k\} \,.\, c'_j \preceq_{\mathcal{C}_L^X} c_{ji} \tag{1}$$

Therefore, again by the ordering on $\mathcal{SS}^X$, $\exists i \in \{1, \ldots, k\}$ such that $S' \preceq_s S[c_j/c_{ji}]$.

□

Proof. (Lemma 9 on page 15) In the following, let S_0 be the initial value of S and $C = C \cup S_0$. Suppose that the procedure terminates after k iterations of the *repeat* loop of $solve_\varepsilon$ (i.e., $S_k = S^f$) and that, for each i where $1 \leq i \leq k$, S_i is the value of the constraint store S at step 5 of the operational schema shown in [Figure 1], after completing i iterations of the *repeat* loop.

Now we prove the cases separately.

Case (a).

We show by induction on i, that after $i \geq 0$ iterations of the *repeat* loop

$$S_i \preceq_s S_0. \tag{2}$$

It follows that after k iterations $S_k \preceq_s S_0$ and thus $S^f \preceq_s S_0$.

The base case when $i = 0$ is obvious. For the inductive step, suppose that there are at least $i > 0$ iterations of the repeat loop and that, after $i-1$ steps, we have $S_{i-1} \preceq_s S_0$. Then, by Line 4 in the operational schema shown in [Figure 1]

$$S_{i-1} \cup C' \mapsto S_i,$$

It follows, from the ordering defined on the constraint store stabilization, that $S_i \preceq_s S_{i-1}$. Therefore by the inductive hypothesis $S_i \preceq_s S_0$.

Case (b).

Let $R \in Sol_a(C \cup S)$. By Definition 4, R is a solution for $C \cup S$, so that by induction on the number of iterations of the repeat loop of the schema $solve_\varepsilon/2$ and the ordering for stores defined on constraint stabilization, $R \preceq_s S_k$ for some $k \geq 0$ and $S_k = S_f$.

Case (c).

Let $R \in Sol_a(C \cup S)$. By Case 9(b), $R \preceq_s S^f$. Suppose that $R \prec_s S^f$. By Definition 4, R is a solution for $C \cup S$ and, by the definition of solution, R is consistent. Thus, by Proposition 3(2), S^f is divisible which contradicts the hypothesis. As consequence, $R = S^f$ and thus $S^f \in Sol_a(C \cup S)$.

Case (d).

By Definition 4, S^k (i.e., S^f) is consistent and thus the procedure $solve_\varepsilon(C, S)$ terminates because

$$precision(S_{k-1}) - precision(S_k) \leq (0.0, 0) \tag{3}$$

By Line 4 of the operational schema shown in [Figure 1], in the k-th iteration,

$$C \rightsquigarrow^{S_{k-1}} C'; \tag{4}$$

$$S_{k-1} \cup C' \mapsto S_k. \tag{5}$$

Then, by (5) and the ordering defined on the constraint store stabilization, $S_k \preceq_s S_{k-1}$. As consequence, from (3) and Proposition 1, $S_k = S_{k-1}$. By (4), (5) and the definition of solution, S_k is a solution for C (i.e., $C \cup S_0$). Therefore, as S_k is non-divisible, by Definition 4, $S_k \in Sol_a(C \cup S)$.

□

Now, we define some concepts that will be useful to prove the main properties of the schema shown in [Figure 2].

A path $q \in (Natural \backslash \{0\})^*$ is any finite sequence of (non-zero) natural numbers. The empty path is denoted by ϵ, whereas $q \,.\, i$ denotes the path obtained by concatenating the sequence formed by the natural number $i \neq 0$ with the sequence of the path q. The length of the sequence q is called the *length* of the path q.

Given a tree, we label the nodes by the paths to the nodes. The root node is labelled ϵ. If a node with label q has k children, then they are labelled, from left to right, $q \,.\, 1, \ldots, q \,.\, k$.

Definition 14. (Search tree for $branch_\alpha(C, S, p)$) Let $X \in \wp_f(\mathcal{V}_\mathcal{L})$, $S \in \mathcal{SS}^X$, $C \in \wp_f(\mathcal{C}^X)$, $\alpha \in \Re^+ \cup \{0.0\}$ and $p \in \Re\mathcal{I}$. The search tree for $branch_\alpha(C, S, p)$ is a tree that has S at the root node and, as children, has the search trees for the recursive executions of $branch_\alpha/3$ as consequence of reaching Line 8 of [Figure 2].

Given a search tree for $branch_\alpha(C, S, p)$, we say that $S_\epsilon = S$ is the constraint store and $p_\epsilon = p$ the precision at the root node ϵ. Let S_q be the constraint store and p_q the precision at a node q. If q has $k > 0$ children $q \,.\, 1, \ldots, q \,.\, k$, then S_q is consistent and, if S_q^f is the constraint store S_q after a terminating execution of $solve_\varepsilon(C, S_q)$, then S_q^f is divisible so that $choose(S_q^f) = c_j$ (for some $c_j \in \mathcal{C}_{L_j}^X$ and $L_j \in \mathcal{L}$) and, for some $k > 0$, $split_{L_j}(c_j) = (c_{j1}, \ldots, c_{jk})$. Then we say that $S_{q.i} = S_q^f[c_j/c_{ji}]$ is the constraint store and $p_{q.i} = precision(S_q^f)$ the precision at node $q \,.\, i$, for $i \in \{1, \ldots, k\}$.

In the following, each property stated in Theorem 10 on page 17 is proved independently.

Proof. (Property (1) of Theorem 10 on page 17). Termination) In the following, we show that the search tree for $branch_\alpha(C, S, p)$ is finite so that the procedure effectively terminates.

Let $S_\epsilon = S$ and $p_\epsilon = p$. If the search tree for $branch_\alpha(C, S_\epsilon, p_\epsilon)$ has only one node then the procedure terminates. Otherwise, the root node ϵ has k children with constraint stores S_i where $i \in \{1, \ldots, k\}$ and $S_i = S_\epsilon^f[c_j/c_{ji}]$. By Lemma 8(a) and Lemma 9(a), for all $i \in \{1, \ldots, k\}$, $S_i \prec_s S_\epsilon$ and, by Proposition 1, $precision(S_i) <_{\Re\mathcal{I}} precision(S_\epsilon)$. Then, $precision(S_i) <_{\Re\mathcal{I}} \top_{\Re\mathcal{I}}$. Suppose now that $precision(S_i) = (\top_\Re, n)$ for some $n \in Integer$. Then the test in Line 2 $p_i - precision(S_i) \leq_{\Re\mathcal{I}} (\alpha, 0)$ holds and the node containing S_i has no children. Otherwise,

$$p_i - precision(S_i) >_{\Re\mathcal{I}} (\alpha, 0) \tag{6}$$

and there exists some constant $\ell \in \Re$ such that

$$precision(S_i) <_{\Re\mathcal{I}} (\ell \times \alpha, 0).$$

We show by induction on the length $j \geq 1$ of a path q in the search tree that

$$precision(S_i) - precision(S_q^f) \geq_{\Re\mathcal{I}} \big((j-1) \times \alpha, 0\big).$$

It follows that $j \leq \ell$ and that, all paths have length $\leq \ell + 1$ (since the second condition in Line 3 of [Figure 2] holds) and thus there are no infinite branches.

The base case when $j = 1$ follows from (6). Suppose next that $j > 1$ and that the hypothesis holds for a path q of length $j-1$. Let $q \,.\, i_q$ be a child of q of length j. Then, by the condition in Line 3 of the *if* sentence,

$$p_{q.i_q} - \mathit{precision}(S^f_{q.i_q}) >_{\Re\mathcal{I}} (\alpha, 0),$$

However, by the inductive hypothesis,

$$\mathit{precision}(S_i) - \mathit{precision}(S^f_q) \geq_{\Re\mathcal{I}} \big((j-2) \times \alpha, 0\big)$$

so that, as $\mathit{precision}(S^f_q)$ is $p_{q.i_q}$,

$$\begin{aligned}\mathit{precision}(S_i) - \mathit{precision}(S_{q.i_q}) \geq_{\Re\mathcal{I}}\\ (\alpha, 0) + \big((j-2) \times \alpha, 0\big) = \big((j-1) \times \alpha, 0\big).\end{aligned}$$

□

Proof. (Property (2) of Theorem 10 on page 17. Completeness) Let $R \in \mathit{Sol}_a(C \cup S)$. Then, R is non-divisible and consistent by Definitions 4 and 2. By the ordering of the solution wrt. the store to solve, $R \preceq_s S_\epsilon$ and, by Lemma 9(b), $R \preceq_s S^f_\epsilon$. If $R = S^f_\epsilon$ then tests in Lines 2-3 hold and R is pushed on the stack P. Otherwise, $R \prec_s S^f_\epsilon$. By Proposition 3(2), S^f_ϵ is divisible, (and thus by Definition 2 consistent). As $p_\epsilon = \top_{\Re\mathcal{I}}$, the condition in Line 3 does not hold and node ϵ has k children. By Lemma 8(a) and Lemma 9(a), for any q of length $m \geq 1$ and $i_q \in \{1, \ldots, k\}$, $S^f_{q.i_q} \prec_s S^f_q$. By Proposition 1, $\mathit{precision}(S^f_q) - \mathit{precision}(S^f_{q.i_q}) > (0.0, 0)$. Thus the condition $p_{q.i_q} - \mathit{precision}(S^f_{q.i_q}) \leq (\alpha, 0)$ in Line 3 never holds. It follows that all the branches in the tree terminate either with an inconsistent store (because test in Line 2 does not hold) or with a non-divisible store (that is also consistent by Definition 2) as result of holding tests in Lines 2 and 3. Now, we show by induction on the length $j \geq 1$ of a path q in the search tree that

$$R \prec_s S^f_q \implies \exists i_q \in \{1, \ldots, k\} : R \preceq_s S^f_{q.i_q}. \tag{7}$$

By hypothesis, the procedure terminates so that the search tree is finite. It follows that there exists some path $p = q \,.\, q'$ with a finite length $l \geq j$ such that $R = S^f_p$. Thus, S^f_p is non-divisible (and, by Definition 2 is consistent) and hence tests in Lines 2 and 3 hold so that R is pushed on the stack P.

In the base case, when $j = 1$, $S_i = S^f_\epsilon[c_j/c_{ji}]$ $(i \in \{1, \ldots, k\})$. By Lemma 8(b) and Lemma 9(b), $\exists i \in \{1, \ldots, k\} : R \preceq_s S^f_i$. Suppose next that $j > 1$ and that the hypothesis holds for a path q of length $j-1$ so that $R \preceq_s S^f_q$. If $R \prec_s S^f_q$ then, by Proposition 3(2), S^f_q is divisible (and thus consistent by Definition 2) so that the node S^f_q has k children. Therefore, by Lemma 8(b) and Lemma 9(b), $\exists i_q \in \{1, \ldots, k\} : R \preceq_s S^f_{q.i_q}$.

□

Proof. (Property (3) of Theorem 10 on page 17. Approximate completeness) Let $R \in Sol_a(C \cup S)$. By the ordering of the solution wrt. the store to solve, $R \preceq_s S_\epsilon$. Since the procedure terminates, as shown in proof of Theorem 10(1), all paths in the search tree have length $\leq \ell + 1$.

Therefore, as shown in proof of Theorem 10(2) (completeness proof), by following (7), there must exist some path q with no children and length $j \geq 1$ such that $R \preceq_s S_q^f$. If $R = S_q^f$ then R is put on the stack since, by Definition 2, R is consistent so that tests in Lines 2 and 3 hold. Otherwise, as shown in termination proof, the node S_q^f has no more children since the test $p_q - \mathit{precision}(S_q^f) \leq_{\Re\mathcal{I}} (\alpha, 0)$ holds and S_q^f is put on the stack. As $R \preceq_s S_q^f$, by Definition 4, either $S_q^f \in Sol_a(C \cup S)$ or is a partial solution for $C \cup S$ that covers R.

□

Proof. (Property (4) of Theorem 10 on page 17. Correctness) Let $R \in P$ after executing $branch_\alpha(C, S, p)$. As shown in completeness proof, if $\alpha = 0.0$ the test $p_q - \mathit{precision}(S_q^f) \leq (\alpha, 0)$ never holds, for all path q (in the search tree) of length $m \geq 1$ (observe also that Line 3 is never satisfied when $q = \epsilon$ since $p_\epsilon \not< \top_{\Re\mathcal{I}}$). Therefore, R is in P because there exists a path q where $S_q^f = R$ and the tests in Lines 2 and 3 hold. Thus, R is consistent and non-divisible and by Lemma 9(4), $R \in Sol_a(C \cup S_q)$.

By induction on the length of the path q it is straightforward to prove that $S_q \preceq_s S_\epsilon$. Now we prove that if $R \in Sol_a(C \cup S_q)$ then $R \in Sol_a(C \cup S_\epsilon)$. By Definition 4 R is a solution for $C \cup S_q$ and thus, by the definition of solution,

$$C \cup S_q \rightsquigarrow^R C'$$
$$R \cup C' \mapsto R.$$

Then, by definition of constraint propagation, $C' = C_1 \cup S_q$ where $C \rightsquigarrow^R C_1$. Moreover, $C \cup S_\epsilon \rightsquigarrow^R C''$ where $C'' = C_1 \cup S_\epsilon$. Since $S_q \preceq_s S_\epsilon$, by definition of stabilized constraint store,

$$R \cup C' \mapsto R \implies R \cup C'' \mapsto R.$$

Therefore

$$C \cup S_\epsilon \rightsquigarrow^R C''$$
$$R \cup C'' \mapsto R.$$

Thus, by the definition of solution, R is a solution for $C \cup S$ and, by Definition 4, $R \in Sol_a(C \cup S_\epsilon)$.

□

Proof. (Property (5) of Theorem 10 on page 17. Approximate correctness or control on the precision result) Suppose that $R \in P_{\alpha_1}$. Then R is consistent and there exists a path q of length $m \geq 0$ such that $S_q^f = R$ and S_q^f was pushed on the stack because the test in Line 3 holds but, for all proper prefixes of q, it does not hold. Thus either S_q^f is non-divisible or

$$p_q < \top_{\Re\mathcal{I}} \text{ and } p_q - \mathit{precision}(S_q^f) \leq (\alpha_1, 0).$$

In addition, for all proper prefixes q_1 of q, $S_{q_1}^f$ is divisible and, either

$$p_{q_1} = \top_{\Re\mathcal{I}} \text{ or } p_{q_1} - \mathit{precision}(S_{q_1}^f) > (\alpha_1, 0).$$

Since $\alpha_1 \leq \alpha_2$, we also have

$$p_q < \top_{\Re\mathcal{I}} \text{ and } p_q - \mathit{precision}(S_q^f) \leq (\alpha_2, 0).$$

Let q' be the smallest prefix of q such that

$$p_{q'} < \top_{\Re\mathcal{I}} \text{ and } p_{q'} - \mathit{precision}(S_{q'}^f) \leq (\alpha_2, 0).$$

Then $S_{q'}^f$ is in P_{α_2}. By repetitive application of Lemmas 8(a) and 9(a) it is straightforward to prove that $S_q^f \preceq_s S_{q_1}^f$ for all proper prefix q_1 of q. Thus, $S_q^f \preceq_s S_{q'}^f$. Therefore, as the choice of $R \in P_{\alpha_1}$ was arbitrary, by Definition 5, $P_{\alpha_1} \preceq_p P_{\alpha_2}$.

□

In the following we prove independently each property claimed in Theorem 12 (observe that the search tree for $\mathit{branch}_{\alpha+}(C, S, p)$ is the same as for $\mathit{branch}_{\alpha}(C, S, p)$).

Proof. (Theorem 12 on page 19. Property (1). Termination) This proof is as that of Theorem 10(1).

□

Proof. (Theorem 12 on page 19. Property (2)) Observe that if $\mathit{fcost}(S) = \delta$ for all $S \in \mathcal{SS}^X$, then test in Line 4* of the extended schema always holds. It is straightforward to prove, in this case, that the schemas $\mathit{branch}_{\alpha}/3$ and $\mathit{branch}_{\alpha+}/3$ are equivalent so that all properties of the schema $\mathit{branch}_{\alpha}/3$ hold in the schema $\mathit{branch}_{\alpha+}/3$.

□

Proof. (Theorem 12 on page 19. Property (3). Soundness on optimization) We prove the case when $\diamond$ and δ are, respectively, $>$ and $\bot_{L_<}$. The respective case

is proved analogously. As shown in proof of Theorem 10(2), for $\alpha = 0.0$, if $R \in Sol_a(C \cup S)$ then there exists in the search tree some path q of length $j \geq 0$, such that $R = S_q^f$ and the tests in Lines 2-3 hold by Definition 4. Thus, Line 4* is reached for all $R \in Sol_a(C \cup S)$, and as consequence, the top of P will contain the first authentic solution found that maximizes $fcost/1$.

□

Proof. (Theorem 13 on page 19: Approximate soundness) We prove the case when $\diamond$ is $<$. The respective case (i.e., $\diamond$ is $>$) is proved analogously. We show that during the execution of $branch_{\alpha_1+}(C, S, p)$, Line 4* is reached for some $S_{q'}^f \preceq_s top(P_{\alpha_2})$ (where q' is a path of length $m_1 \geq 0$) and thus $fcost(S_{q'}^f) \preceq_{L<} fcost(top(P_{\alpha_2}))$. It follows that either $S_{q'}^f = top(P_{\alpha_1})$ or $S_{q'}^f \neq top(P_{\alpha_1})$ because there is another store $S_{q''}^f = top(P_{\alpha_1})$ such that $fcost(top(P_{\alpha_1})) \preceq_{L<} fcost(S_{q'}^f)$. In both cases it follows that effectively $fcost(top(P_{\alpha_1})) \preceq_{L<} fcost(top(P_{\alpha_2}))$.

Observe that $top(P_{\alpha_2})$ is in P_{α_2} because there exists a path q of length $m \geq 0$ such that $S_q^f = top(P_{\alpha_2})$ and S_q^f was pushed on the stack because Line 4* is reached and the tests in Lines 2 and 3 holds but, for all proper prefixes of q, it does not hold. Thus S_q^f is consistent and either non-divisible or

$$p_q < \top_{\Re\mathcal{I}} \text{ and } p_q - precision(S_q^f) \leq (\alpha_2, 0).$$

In addition, for all proper prefixes q_1 of q, $S_{q_1}^f$ is divisible and, either

$$p_{q_1} = \top_{\Re\mathcal{I}} \text{ or } p_{q_1} - precision(S_{q_1}^f) > (\alpha_2, 0).$$

Since $\alpha_1 \leq \alpha_2$, we also have that, for all proper prefixes q_1 of q, either

$$p_{q_1} = \top_{\Re\mathcal{I}} \text{ or } p_{q_1} - precision(S_{q_1}^f) > (\alpha_1, 0).$$

Thus, the node with path q is in the search tree for $branch_{\alpha_1+}(C, S, p)$. Now, we have two cases: (1) S_f^q is consistent and non-divisible. As consequence, S_f^q has no children and Line 4* is reached in the execution of $branch_{\alpha_1+}(C, S, p)$. (2) S_f^q is consistent and divisible. By hypothesis S_f^q is an authentic solution R or covers an authentic solution R for $C \cup S$. Then, reasoning as in proof of Theorem 10(3) and by (7), there must exists some path q', containing the path q, with no children such that $S_{q'}^f$ is consistent and either $R = S_{q'}^f$ or

$$p_{q'} < \top_{\Re\mathcal{I}} \text{ and } p_{q'} - precision(S_{q'}^f) \leq (\alpha_1, 0).$$

In both cases, Line 4* is reached. Moreover, by repetitive application of Lemmas 8(a) and 9(a) it is straightforward to prove that $S_{q'}^f \preceq_s S_q^f$.

□