Università degli Studi di Bologna

DEIS

A Meta Constraint Logic Programming Architecture for Qualitative and Quantitative Temporal Reasoning

Evelina Lamma   Paola Mello   Michela Milano

October 10, 1995

DEIS Technical Report no. DEIS-LIA-95-001   *LIA Series no. 1*

A Meta Constraint Logic Programming Architecture for Qualitative and Quantitative Temporal Reasoning

Evelina Lamma [1] Paola Mello [2] Michela Milano [1]

[1] *DEIS, Università di Bologna*
Viale Risorgimento 2, 40136 Bologna, Italy
Tel: +39 51 6443033, Fax: +39 51 6443073
{elamma,mmilano}@deis.unibo.it
[2] *Dipartimento di Informatica, Università di Bari*
Via Orabona 4, 70126 Bari, Italy
pmello@deis.unibo.it

October 10, 1995

Abstract. In this paper we present a two module CLP architecture (object- and meta-level modules) for reasoning with both qualitative and quantitative temporal constraints. The object level module is a finite domain constraint solver while the meta-level system is an extension of CLP for dealing with qualitative temporal information. For each module we define, from a theoretical and practical point of view, the operational semantics and the constraint domain. Moreover, we define the interactions between the two levels in terms of linking rules.

The paper shows how Constraint Logic Programming (CLP) can be used as an effective tool for representing and reasoning about time by specializing the CLP general scheme for the quantitative and qualitative temporal domains.

The advantages of using a meta architecture mainly concerns modularity, flexibility and expandibility of the architecture. The expressive power of CLP is also increased. CLP uses constraints for guiding the computation but it is not able to reason and to infer information about constraints. On the other hand, our meta architecture performs computation on constraints and infers information about them.

Finally, the paper describes an implementation of the system.

Keywords: *Temporal Reasoning, Constraint Logic Programming, Meta-Level Reasoning*

Contents

1 Introduction

Temporal reasoning plays a central role in many Artificial Intelligence applications such as planning [5, 7, 46], scheduling [33, 42] natural language understanding [12, 6], and diagnosis [24, 26, 41]. Several frameworks have been proposed in order to deal with this kind of reasoning[1], most notably Allen's Interval Algebra [2], Vilain and Kautz's Point Algebra [51, 52], Ladkin and Maddux's formalism [32], Dechter, Meiri and Pearl's STP and TCSP [16], and Dean and McDermott's Time Map Management [15]. In addition, some promising approaches have been proposed in order to integrate qualitative and quantitative temporal reasoning [29, 36].

These frameworks are classified as constraint propagation methods as opposed to the Temporal Logics methods [3, 19]. In these frameworks, in general, the representation of the problem is given in terms of a constraint graph where nodes are the variables of the problem (points or intervals) and arcs are constraints (temporal relations between pairs of variables[2]). Given a constraint graph, we are interested in determining whether the graph is consistent, i.e., if it contains at least a compatible solution with constraints, or in determining an equivalent minimal representation, i.e., a representation which has the same solution set of the original network and is minimal with respect to the "tighter" relation[3], or in computing the transitive closure of the network. All the researchers who have worked in the temporal reasoning field have proposed algorithms for solving these problems (see, for example, [2, 16, 43, 47, 51, 52, 53, 54]) but, in general, they did not focus on defining a programming paradigm for solving these problems.

In this paper, we show how Constraint Logic Programming [27, 28] can be a flexible and suitable tool for solving Constraint Satisfaction Problems (see [48, 50]) and, in particular, temporal reasoning which can be seen as a constraint propagation process. In fact, CLP handles very efficient constraint propagation mechanisms and the possibility of extending the built-in solvers.

We propose an approach based on Constraint Logic Programming which follows, for the most part, the Meiri's framework and is able to treat both qualitative and quantitative temporal constraints. Our approach is based on two related networks: one is the usual graph where nodes are variables (intervals and points) and arcs represent temporal relations, while the second is a *meta-constraint*-graph which treats temporal relations as variables, composition and intersection as operations and tighter relations as constraints. These networks are handled and maintained by two constraint solvers: the first one is a finite domain constraint solver CLP(FD), as introduced in the CHIP language [18], while the second is a meta constraint solver which handles qualitative information reasoning on the constraints of the object level.

This approach allows us to treat both qualitative and quantitative temporal constraints, and the interaction between the two kinds of constraints.

The main advantage of using Constraint Logic Programming is that of the possibility of

[1] For a survey see [4].

[2] Usually, in temporal constraint propagation frameworks, we treat only binary constraints, i.e., relations between two variables. Of course, each n-ary relation between variables can be represented in terms of binary constraints.

[3] A constraint C' is tighter than C" if every pair of values allowed by C' is also allowed by C".

implementing other modules interacting with the temporal reasoner (a planner, a scheduler or a natural language processor) using a logic programming language. Therefore, this temporal reasoner can easily interact with the rest of the system.

The integration of CLP and temporal reasoning motivated other works such as [9, 25, 34]. These works try to integrate temporal reasoning and Constraint Logic Programming. The first two approaches differ with our approach since they are flat (a one level architecture), while our system is based on a meta architecture. The advantages of using a meta architecture [1] mainly concern the modularity, flexibility and expandibility of the architecture. What is more, we can gain greater expressive power from CLP. In fact, CLP uses constraints for guiding the computation and replaces the unification mechanism with constraint solving, but it is not able to reason and to infer information about constraints. A meta architecture performs computation on constraints and infers information about them, such as for example, determining the minimal network.

The third work, by T. Frühwirth in [25] proposes the Constraint Handling Rules, a Constraint Logic Programming extension which handles qualitative and quantitative temporal constraints by using higher level guarded rules. A very important relation between our work and that of T. Frühwirth concerns the fact that Constraint Handling Rules represent a very general architecture for prototyping and implementing constraint solvers. Our approach can be implemented in a straightforward way by using Constraint Handling Rules.

The paper is organized as follows. In Section 2 we will present two related works which represent the starting point of our research. In Section 3 we will sketch some features of the Constraint Logic Programming paradigm and, in particular, of CLP(FD) while focusing on its limitation for treating qualitative temporal reasoning. Moreover, we will offer some suggestions for solving these limitations. Section 4 is devoted to the architecture of the general purpose temporal reasoner, i.e., the qualitative and the quantitative modules and their interactions. In this Section we also define formally the constraint domains of the two solvers. Section 5 defines the operational semantics of the system in terms of a qualitative solver, a quantitative solver and of the interaction between the two. In Section 6 we discuss the implementation of the CLP solvers developed. Section 7 is devoted to some examples. Section 8 concerns related works. The conclusion and a mention of future works follow.

2 The Background

Two notable works are worth being mentioned since they represent the starting point of the framework which we present here and will be frequently referred to in the remainder of the paper. Firstly, Meiri's work on qualitative and quantitative networks [36], then Tsang's approach to temporal reasoning as a Constraint Satisfaction Problem (or Constraint Labeling Problem) [45].

Meiri's framework treats either temporal intervals or points. Meiri defines a general network-based computational model for temporal reasoning, capable of handling both qualitative and quantitative temporal relations in a unified framework. Qualitative constraints between two temporal entities are disjunctions such as:

$$(O_i\ r_1\ O_j) \vee \ldots \vee\ (O_i\ r_n\ O_j)$$

where r_is are basic Interval-Interval relations of the Interval Algebra, Point-Point relations of the Point Algebra and Point-Interval relations of the Ladkin and Maddux framework [32]. For this purpose, a Qualitative Algebra has been defined whose elements are all legal constraints (all combinations of primitive constraints of the same type). The operations on constraints are intersection and composition. Intersection combines two relations that hold on the same pair of objects. Composition admits the determining the relation between two primitive temporal entities I and J starting from the relations between I and K and K and J. For this purpose transitivity tables have been defined.

Quantitative constraints of Meiri's framework refer to absolute locations or the distances between points. These relations are unary and binary constraints defined for the TCSP model [16]. Unary constraints restrict the absolute location of a point to a set of intervals:

$$(P_i \in I_1) \vee \ldots \vee (P_i \in I_n).$$

Binary constraints (called bound on distance constraints) restrict the distance between two points to a set of intervals:

$$(P_i - P_j \in I_1) \vee \ldots \vee (P_i - P_j \in I_n).$$

Meiri defines also relationships between qualitative and quantitative temporal constraints (for details see [36]).

Similar to all the constraint propagation methods, this framework is based on a graph representation where nodes are variables (points and intervals) and arcs are the abovementioned relations. In this graph, qualitative and quantitative constraints coexist. An interesting task to be performed on such a network is that of determining whether it is consistent. If the network is consistent, we can be interested in finding a solution or all the solutions to the problem, or we can be interested in computing the minimal network. Unfortunately, for these kind of networks all these tasks have an exponential computational complexity. For this purpose, Meiri defines the so-called "islands of tractability", i.e., special classes of networks which are solvable in polynomial time.

In general networks, each arc is labeled by a disjunction of relations which hold between two variables. Meiri briefly suggests a singleton labeling method where arcs of the graph are labeled by a basic relation which belongs to the original disjunction of relations. In this way, we eliminate disjunctions from the graph and the simplified problem (whose structure is that of a STP) becomes tractable. This process must be iterated in order to label each arc with all its disjunctive relations. Therefore, if n is the number of nodes, k is the maximum number of basic labels of an arc and e is the number of arcs, the overall complexity[4] is $O(n^3k^e.)$. Anyway, Meiri suggests that "this brute-force enumeration can be pruned significantly by running a backtracking algorithm on a meta-CSP whose variables are the network arcs and whose domains are the possible basic labels. Backtrack assigns

[4] Note that the complexity of STP is $O(n^3)$

a basic label to an arc, as long as the corresponding STP network is consistent and, if no such assignment is possible, it backtracks". In this paper, we start from this observation in order to handle qualitative constraints.

Now we should start describing the second related paper [45]. Tsang states a relation between temporal reasoning and the Constraint Labeling Problem (also called Constraint Satisfaction Problem). He noticed that the problem of assigning a primitive relation to each temporal relation is a Constraint Labeling Problem. Associated with each temporal problem we have two different graphs: the first is the graph G where nodes N are variables of the problem (e.g., points or interval) and arcs R are constraints between variables:

$$G = (N, R).$$

The second graph is called the Constraint Network. In a Constraint Network nodes are the set of arcs of G, while arcs T are constraints on elements of R. Therefore, a Constraint Network CN is represented by:

$$CN = (R, T).$$

In addition, Tsang defines some search strategies suitable for temporal reasoning such as which variable has to be labeled first, or which value has to be tried next and so on.

We start from these two papers for building a general purpose temporal reasoner which is able to handle both qualitative and quantitative constraints following, mainly, the Meiri's framework by using a Constraint Logic Programming (CLP) paradigm. Our temporal reasoner uses a different propagation for qualitative and quantitative interactions more suitable for Constraint Logic Programming. The temporal reasoner is composed of a quantitative network (network G of Tsang) implemented by a CLP(FD) constraint solver and a qualitative network (network CN of Tsang) for reasoning on qualitative constraints (arcs) of the network G. This temporal reasoner will be described in the rest of this paper.

3 Constraint Logic Programming

Constraint Logic Programming (CLP) is a class of programming languages combining the advantages of Logic Programming (LP) ([30, 35]) and the efficiency of constraint solving. CLP "inherits" from LP the operational and declarative semantics which lead to simple, elegant and natural programs. However, CLP overcomes the disadvantages of LP by enhancing LP with constraint solving techniques. These disadvantages concern mainly the inefficiency of typical LP searching techniques that leads to a combinatorial explosion as the dimension of the problem increases. The second limitation of LP concerns the syntactic unification. In LP, terms are uninterpreted structures. Two terms can be considered unifiable if they are syntactically, and not semantically, identical.

CLP solves both these limitations. Syntactic unification can be avoided by exploiting the CLP scheme which allows working in pre-defined domains such as real, integer, rational, boolean or finite domain. Working in a pre-defined domain ensures that the programmer uses concepts that are close to the domain of discourse. Associated with each domain there are primitive operations and relations (constraints) on the domain itself.

More formally, each CLP language can be seen as an instance of a general CLP(X) scheme where each instance can be specified by instantiating the parameter X (for more details see [28]). This parameter X represents a 4-tuple $(\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$ where Σ is a signature, $\mathcal{D}$ is a Σ-structure, $\mathcal{L}$ a class of Σ-formulas and $\mathcal{T}$ is a first-order Σ-theory.

The signature Σ represents the predefined predicates and function symbols, $\mathcal{D}$ is the domain structure, $\mathcal{L}$ is a class of constraints which can be expressed in the domain and $\mathcal{T}$ is the axiomatization of some properties of $\mathcal{D}$. Each CLP language can be defined by specifying all the 4-tuple components. Each CLP system is determined by the constraint domain $(\mathcal{D}, \mathcal{L})$ and by the operational semantics.

The other limitation of LP, which concerns the inefficiency of the searching techniques, is solved by CLP which is able to reduce the dimension of the search space by exploiting the so-called Consistency Techniques (see [48]). They can be defined as intelligent search techniques based on the *a-priori pruning* of the search space by propagating constraints. *A-priori pruning* consists in reducing the search space of the problem by preventing *a priori* failures. Each variable is associated with a domain, i.e., a set of feasible values for the variable itself. When a constraint involving this variable is solved, the associated domain is consequently reduced by eliminating the values incompatible with the constraint. Therefore, the possible values for the variable are reduced *a priori*, limiting the search space of the problem.

3.1 CLP on Finite Domains - CLP(FD)

CLP on finite domains (CLP(FD)) is a constraint logic programming language where constraint solving is done by the so called Consistency Techniques which are derived from Constraint Satisfaction Problems [21, 22, 23, 31, 37, 39, 40]. Variables range on a finite domain which is initially set to [0..max]. These domains are gradually reduced by the propagation of constraints. An inconsistency is detected when a domain becomes empty.

As an example, let us suppose that we have three variables X, Y, Z ranging on the domain [1..10], and the constraint

$$X < Y$$

which produces the propagation: $D_X = [1..9]$ and $D_Y = [2..10]$ and the constraint

$$Y < Z$$

which produces the propagation: $D_Y = [2..9]$ and $D_Z = [3..10]$.
The fact that D_Y has changed produces the final propagation of the constraint which lead to the reduction of the domains of X, Y, Z to :

$$D_X = [1..8], D_Y = [2..9], D_Z = [3..10].$$

This kind of propagation is termed "arc-consistency" and ensures that for each value of a variables' domain D_X there exists at least one value in each other domain which is consistent with the constraints containing the variable X. Recently, it has been shown that CLP on finite domains can be extended with the "path-consistency" propagation technique [13]. Path-consistency is much more powerful than arc-consistency because it finds inconsistencies when arc-consistency does not. However, both these techniques have been proved to be incomplete (see [37, 39]). In fact, there are cases in which variables are arc-consistent (path-consistent) with constraints but a solution does not exist. Moreover, in an arc-consistent (path-consistent) network the system has to perform some search in order to find a solution. Many optimized search strategies, such as heuristic labeling strategies have been defined (see for instance [48]).

As concerns a more formal definition of CLP on finite domains, we refer to [28] where this language is defined in the following way: let D=Z and

$$\Sigma = \{\{\in [m,n]\}_{m \leq n}, +, =, \neq, \leq\}.$$

The symbols in Σ have their usual meaning apart from $\{\in [m,n]\}_{m \leq n}$ which is defined as follows: for every pair of integers m and n, $\mathcal{D}$ interprets the interval constraint $x \in [m,n]$ as $m \leq x \leq n$. $\mathcal{L}$ is the class of constraints which are generated by the primitive constraints where each variable is subject to an interval constraint. Then $\mathcal{FD} = (\mathcal{D}, \mathcal{L})$ is the constraint domain. A typical constraint which can be expressed in this domain is

$$x \in [3,7] \wedge y \in [1,9] \wedge x \neq 5 \wedge 4x - y \leq 16$$

The operational semantics for this class of CLP systems will be defined in Section 5.

3.2 CLP(FD) and its Limitations for Temporal Reasoning

In this section, we show which is the expressible "part" of temporal reasoning with CLP on finite domains. Intuitively, CLP(FD) is able to express quantitative reasoning. In fact, we can interpret the constraints:

$$(x \in [3..16]) \wedge (y \in [1..19]) \wedge (x - y \leq 7) \wedge (x - y \geq 16)$$

in a temporal reasoning context. We can see the variables x and y as temporal points which have a temporal location on the time line which is defined by their domains. In particular, x is a temporal point located between 3 and 16 time units, while y between 1 and 19. The last constraint links the two variables (time points) to assume a relative position.

The propagation of constraints leads to the reduction of the variables domains to

$$(x \in [4..16]) \wedge (y \in [1..13])$$

while suspending the two constraints

$$(x - y \leq 7) \wedge (x - y \geq 16).$$

Therefore, with CLP(FD) we can express the quantitative constraints within temporal points, i.e., ranges of absolute locations on the time line and bound on difference constraints (see [16]).

Temporal intervals can be expressed as pairs of points linked by the relation $I_{inf} < I_{sup}$ where I_{inf} and I_{sup} are the starting and ending points of the interval I. The duration of intervals can be expressed in terms of "bound on difference" constraints between points.

It is worth noting that CLP(FD) is able to represent qualitative constraints such as $x < y$ but it is not able to reason about them. In fact, CLP on finite domains does not suffice for reasoning about time. This inadequacy arises for three reasons. We call it the Composition problem, the Intersection problem and the Qualitative-Quantitative problem:

- Composition: in the example of Section 3.1 it is worth noting that when the propagation terminates we are not able to infer information on the relationship that holds between X and Z. Therefore, starting from the domains $D_X = [1..8]$ and $D_Z = [3..10]$ the system does not know which relation holds between these two variables. In a temporal reasoner this is a very important information because it is fundamental in the building of the transitive closure of the temporal network. The information that should be inferred from the two constraints $X < Y$ and $Y < Z$ is the relation $X < Z$. This relation should be inferred from the transitivity rule, and, in particular, it is the result of the composition of constraints, see for example [2, 51]. Therefore, the first step to perform in order to build a temporal reasoner starting from Constraint Logic Programming is to embed in CLP the capability of performing the composition of qualitative constraints.

- Intersection: the second problem of CLP on finite domain is related to the first and derives from the computation of the minimal network. If we state the constraints $X < Y$, $Y < Z$ and $X < Z \vee X = Z$, they are consistent but not minimal. A constraint solver should be able, in this case, to compose the first two constraints and to intersect the result with the third constraint. CLP on finite domains does not create the minimal network but ensures a certain level of consistency.

- Qualitative-Quantitative: another limitation of CLP on finite domains is that it does not produces a minimal network by eliminating redundant constraints but reasons only on values. Suppose that we state that $X < Y \vee X = Y$ and X and Y are temporal points ranging on $D_X = [1..5]$ and $D_Y = [6..10]$ respectively. It can be seen immediately that X and Y cannot be equal because does not exist any pair of values from D_X and from D_Y that satisfies the constraint. Therefore, the tightest constraint that holds between X and Y is $X < Y$. However, a constraint solver on finite domains propagates constraints and simply concludes that the constraint is satisfied. Therefore, the third requirement of the temporal reasoner is the interaction between domain's values and qualitative constraints between variables.

3.3 How to Overcome these Problems

In the previous section we have underlined that CLP(FD) can be a suitable tool for representing and reasoning on quantitative temporal reasoning but presents some limitations for handling qualitative reasoning. Therefore, we propose an approach for temporal reasoning that uses Constraint Logic Programming and solves the above mentioned limitations. In our approach we use a constraint solver on finite domains in order to handle quantitative constraints and we embed the capability of handling qualitative temporal constraints directly in a meta-constraint solver [5]. We build a meta-constraint solver on a meta-constraint graph where the variables are relations (called relation-variables) and unary constraints restrict a relation-variable to take its values from a finite domain of disjunctive relations.

Therefore, for each pair of variables X and Y (nodes of the graph) we have a meta-variable, namely R_{XY}, which ranges on a finite domain of relations. This domain contains the relations that hold between X and Y. Suppose that $X < Y \vee X = Y$, the domain of the meta-variable R_{XY} is $D_{XY} = [\{<\}, \{=\}, \{<, =\}]$.

Intersection and composition are embedded in the constraint solver. The fact that in the constraint graph two constraints hold between the same pair of variables means that the constraints should be intersected. In the meta-constraint graph we have a relation between the two variables representing the fact that the domains of the two variables are intersectable. This meta-relation can be handled by a meta-unification mechanism as described in the following.

The fact that in the constraint graph two constraints hold between variables X and Y and between variables Y and Z means that these constraints should be composed in order to find the relation between X and Z. Therefore, in the meta-constraint graph there should be a relation between the meta-variables R_{XY} and R_{YZ} that states that their composition is equal to R_{XZ}.

In his work, Tsang [45] underlines the presence of other binary constraints between relation-variables. For example, the constraint "if A meets B then C meets D" is a binary constraint between the meta-variables R_{AB} and the meta-variables R_{CD}. Therefore, if the meta-variable representing the relation between A and B is instantiated to the value "meets" or if the domain of R_{AB} contains the value "meets" then the domain of R_{CD} should contain the value "meets" as well. This constraint should be embedded in the solver as a user-defined constraint by specifying the way it should be propagated and solved.

Therefore, the general purpose temporal reasoner will be composed by two constraint solvers: the former for quantitative and the latter for qualitative temporal constraints. These constraint solvers have the capability of propagating their "own" constraints and the capability of interacting with each other in order to perform a more powerful propagation.

The overall temporal reasoner is depicted in Figure 1.

The next Section is devoted to explaining in detail the two modules and the interaction between them.

4 A General Purpose Temporal Reasoner based on CLP

In this section, we will explain how the two modules perform the propagation of constraints and how they interact in order to perform a quantitative-qualitative constraint propagation that solves the limitation of CLP(FD) (described in Section 3.2) when used for resolving temporal reasoning.

[5] For a detailed description of the representation and use of meta-knowledge refer to [1, 11]

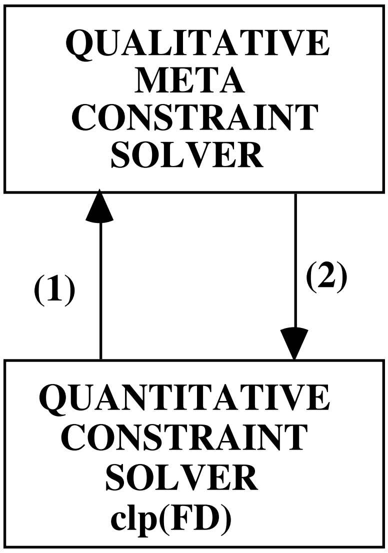

Figure 1: Temporal reasoner general architecture

4.1 Quantitative Constraint Solver

The quantitative constraint solver is based only on CLP(FD) propagation mechanisms. In CLP on finite domains one can express unary constraints between points such as:

$$P_i \in I$$

and binary constraints between pairs of points such as:

$$(P_i - P_j) \in I.$$

For example, if variable X ranges on the domain $D_X = [10..20]$ and variable Y ranges on the domain $D_Y = [40..50]$ and the constraint $Z = Y - X$ is stated, then the constraint solver is able to associate with the variable Z the domain $D_Z = [20..40]$. Again, if the variable X ranges on the domain $D_X = [10..20]$ and the variable Z on the domain $D_Z = [30..40]$, then the propagation of the constraint $Z = Y - X$ associates with the variable Y the domain $D_Y = [40..60]$. These constraints, after the propagation, are delayed because the variables are not yet instantiated and may cause further propagations.

We should give to the system the full representation of the network in terms of paths linking all the variables of the network. For example, if X and Y are linked by several paths, these paths should be given to the constraint solver explicitly. In the example given by Dechter, Meiri and Pearl in [16], we have four temporal variables, namely X_1, X_2, X_3, X_4, and a time point X_0 representing the "beginning of the world". The constraints defined by the problem are:

$$\boxed{\begin{array}{l} 10 \leq X_1\text{-}X_0 \leq 20, \\ 30 \leq X_2\text{-}X_1 \leq 40, \\ 10 \leq X_3\text{-}X_2 \leq 20, \\ 40 \leq X_4\text{-}X_3 \leq 50, \\ 60 \leq X_4\text{-}X_0 \leq 70. \end{array}}$$

These constraints can be written[6] as:

```
X₁::10..20, % The first eleven constraints are stated by the problem
Z21::30..40,
Z21#=X2-X1,
X4::60..70,
Z43::40..50,
Z43#=X4-X3,
Z23::10..20,
Z23#=X2-X3,
Z42#=X4-X2,
Z41#=X4-X1,
Z31#=X3-X1,
% The last constraints perform a path-consistency equivalent algorithm
Z31#=Z21-Z23,
Z31#=(-Z43)-(-Z41),
Z21#=Z23-(-Z31),
Z21#=Z24-(-Z41),
Z43 #=Z41-Z31,
Z43#=Z42-(-Z23),
Z42#=Z43-Z23,
Z42#=Z41-Z21,
Z41#=Z43-(-Z31),
Z41#=Z42-(-Z21),
Z23#=Z21-Z31,
Z23#=(-Z42)-(-Z43).
```

The first eleven constraints define the structure of the problem. In their paper [16] Dechter, Meiri and Pearl have shown that the *Floyd-Warshall*'s all-pairs shortest-paths algorithm represents a complete algorithm for finding the consistency of an STP and for computing the minimal network. Moreover, they have shown that for the Simple Temporal Problem this algorithm is equivalent to the path consistency algorithm PC-1 whose structure is the following[7]:

1. `repeat`
2. `S` $\leftarrow$ `T ;`
3. `for k:=1 to n do`
4. `for i,j:=1 to n do`
5. $T_{ij} \leftarrow T_{ij} \oplus T_{ik} \otimes T_{kj}$,
6. `if` T_{ij} = $\emptyset$ `then inconsistency`
7. `until S=T`

Tab. 4.1: A path consistency algorithm

[6] This representation of constraints is that of ECL^{i}PS^e [20] where the predicate symbol `::` associates a variable `Var` to a domain `Domain` in the following way:

`Var :: Domain.`

The constraint `#=` is the equality constraint. The difference with the symbol `=` is that while the first symbol represents a constraint and therefore performs a propagation on the variable domain, the second performs only a test. Therefore, the constraint:

`X#=Y`

where `X` and `Y` are domain variables performs the intersection of domains.

[7] The symbol $\oplus$ represents the intersection while $\otimes$ the composition of constraints.

A path consistency propagation algorithm computes the consistency of every path in the network by performing the innested cycles 3 and 4. On the other hand, a CLP(FD) solver performs a lower consistency check: an arc consistency algorithm which does not find all the paths of the network given the variables of the problem. However, if we add to a program some constraints describing the paths in the graph we are able to obtain path consistency. The following theorem follows from the work of Freuder [21].

Theorem 4.1 *Let G=(V,A) a graph where each node represents a variable (temporal point) of the problem and each arc, linking a pair of variables, is a binary constraint defining the minimal and maximal distance between the two points. Let us consider three variables X, Y and Z linked by the following bound on distance constraints:*

$$a_1 \leq X - Y \leq a_2,$$

$$b_1 \leq X - Z \leq b_2,$$

$$c_1 \leq Z - Y \leq c_2.$$

The path through nodes X, Y and Z is path consistent iff the following constraints are arc consistent:

$$XY :: a_1..a_2,$$

$$XZ :: b_1..b_2,$$

$$ZY :: c_1..c_2,$$

$$XY\# = XZ + ZY,$$

where $XY = X - Y$, $XZ = X - Z$ *and* $ZY = Z - Y$[8].

Proof. (Sketched) We will show that the path-consistency algorithm depicted in the table 4.1 is equivalent to the arc-consistency algorithm on the variables XY, XZ and YZ.

The unary constraints

$$XY :: a_1..a_2$$

$$XZ :: b_1..b_2$$

$$ZY :: c_1..c_2$$

represent the possible values admitted by the arcs linking variables X and Y, X and Z, Z and Y respectively. A constraint linking the three (arc)variables XY, XZ and ZY represents a constraint on the path linking the three variables X, Y and Z.

On each path, in order to achieve path-consistency, we should perform an intersection and a composition (step 5 depicted in table 4.1).

[8] A similar idea to build a network where variables represent durations of events (distance between points) was proposed in [8].

The intersection of two constraints I and J (each representing an interval of feasible values for a pair of variables), denoted by the symbol $I \oplus J$, is defined as the set theoretic intersection $I \cap J$ of the two intervals [9] I and J.

$$I \oplus J = I \cap J$$

The composition of two constraints I and J, denoted by the symbol $I \otimes J$, admits only those values r for which there exist $i \in I$ and $j \in J$, such that $i + j = r$.

$$I \otimes J = \{r \mid \exists i \in I,\ j \in J\ s.t.\ r = i + j\}$$

These operations should be performed in order to achieve the path consistency on each path of the graph. Now we should prove that an arc propagation on the constraint

$$XY\# = XZ + ZY$$

is equivalent to the path consistency algorithm on the corresponding path. This comes from the definition of intersection and composition. In a finite domain constraint solver the propagation of the constraint $A\# = B$ performs an intersection on the domains of the variables A and B, which are linked, after the propagation, to the intersection of the two domains. Therefore, the domain resulting from $XZ + ZY$ intersects with the domain of variable XY (representing the distance from X to Y).

Finally, in a finite constraint solver the operator + performs, by definition, the composition of unary constraints XZ and ZY. The resulting domain variable XY ranges on a domain containing only those values xy for which there exist $xz \in D_{XZ}$ and $zy \in D_{ZY}$, such that $xz + zy = xy$.

$$XZ + ZY = \{xy \mid \exists xz \in D_{XZ},\ zy \in D_{ZY}\ s.t.\ xz + zy = xy\}$$

The operation of intersection and composition should be performed for each path linking three nodes. It is worth noting that there should be many constraints containing the variable XY: one for each path containing variables X and Y. Therefore, we should add to the program as many constraints as paths in order to achieve path consistency. $\square$

With this theorem we are able to reach, without modifying the CLP(FD) solver, the same result obtained by [13] which modifies the structure of the CLP(FD) solver for achieving path-consistency.

The result of the computation of the above-mentioned example consists of delayed constraints in which each variable is associated with the minimal domains:

[9] With an abuse of notation, we use the constraint symbols I and J to represent the corresponding intervals.

`X`$_1$`::10..20,`
`X`$_2$`::40..50,`
`X`$_3$`::20..30,`
`X`$_4$`::60..70,`
`Z`$_{21}$`::30..40,`
`Z`$_{31}$`::10..20,`
`Z`$_{41}$`::50..60,`
`Z`$_{23}$`::10..20,`
`Z`$_{42}$`::20..30,`
`Z`$_{43}$`::40..50.`

Therefore, with CLP on finite domains we are able to solve a STP, to find its consistency and to compute the minimal network associated provided that we add some "path-equivalent" constraints.

As concerns the complexity of the above mentioned computation, we will demonstrate that it is equal to the complexity of a path consistency algorithm. In fact, we have the following theorem:

Theorem 4.2 *Let G=(V,A) a graph where each node represents a variable (temporal point) of the problem and each arc, linking a pair of variables, is a binary constraint defining the minimal and maximal distance between the two points. Performing an arc consistency algorithm on variables representing the distance between two points has the same complexity as that of performing a path consistency on the variables of the original problem.*

Proof. Given a complete graph G=(V,A) let us suppose that n is the number of variables, e is the number of binary constraints and a is the edge degree of each vertex[10]. The complexity of performing a path consistency algorithm PC-2, see [37, 38], is $O(n^3a^3)$.

Now we analyse the behavior of the arc consistency algorithm on a wider graph (we call it *distance graph*), i.e., the graph of the nodes representing the distance between two variables of the original problem.

If n is the number of original variables, the distance graph will contain $\frac{1}{2}n(n-1)$ nodes. The distance graph is not complete. In particular, for each distance node XY we have $2(n-2)$ arcs, corresponding to the 3-length paths containing the original variables X and Y. Therefore, in the distance graph we have the following number of binary constraints:

$$e = \frac{1}{2}(\frac{1}{2}n(n-1))(2(n-2))$$

The complexity of performing an arc consistency algorithm AC-3, see [37, 38], is $O(ea^3)$, but e is proportional to n^3. Therefore, the overall complexity is the same. □

For the TCSP framework, it is possible to solve a TCSP through a labeling procedure where each label is defined as the assignment of an interval to each arc. Each label defines an STP graph whose edges are labeled by the selected intervals. The complexity of this process is the following: if n is the number of nodes, a is the maximum number of basic

[10] For the sake of clarity, we suppose that each vertex has the same number of values in its domain.

labels of an arc and e is the number of arcs, the overall complexity is $O(n^3a^e)$. With a constraint solver on finite domains we can, therefore, solve also the TCSP framework by solving the corresponding STPs.

4.2 Qualitative Constraint Solver

The qualitative constraint solver is based on a meta-constraint solver which uses finite domains of relations. A meta-variable R_{XY} is automatically generated each time a new qualitative constraint between variables X and Y is introduced. Qualitative constraints are disjunctions of primitive constraints between variables. Therefore, if between variables X and Y we have a relation of the kind:

$$(X\ r_1\ Y) \vee \ldots \vee\ (X\ r_n\ Y)$$

which can be written as:

$$X\{r_1, \ldots, r_n\}Y$$

where r_is are primitive qualitative relations[11], then the system automatically creates a meta-variable R_{XY} whose domain D_{XY} is the powerset of $\{r_1, \ldots, r_n\}$.

Let's analyse how the system performs the intersection and composition between qualitative constraints. The intersection between two constraints $X\{r_1, \ldots, r_n\}Y$ and $X\{r'_1, \ldots, r'_n\}Y$ is the powerset of the set theoretic intersection between the two vectors $\{r_1, \ldots, r_n\}$ and $\{r'_1, \ldots, r'_n\}$. A meta-variable representing the possible relations between X and Y is associated with each constraint. Therefore, the intersection operation can be performed by imposing the two meta-variables R_{XY} and R'_{XY} (corresponding to the two constraints) to be equal. In the common constraint solvers, imposing the equality of two variables requires a meta-unification which can be handled and managed by the constraint solver by intersecting the variable domains.

In a finite domain constraint solver, if variable X ranges on $D_X = [1..10]$ and Y on $D_Y = [7..14]$ the meta-unification $X = Y$ associates to each variable the domain $D_X = D_Y = [7..10]$. In the same way, if the meta-variable has an associated domain:

$$D_{XY} = [\{<\}, \{=\}, \{<, =\}, \{\}]$$

because of the existence of the qualitative constraint $X < Y \vee X = Y$[12], and the meta-variable R'_{XY} has an associated domain

$$D'_{XY} = [\{>\}, \{<, >\}, \{<\}, \{\}]$$

because of the existence of the qualitative constraint $X < Y \vee X > Y$, then the intersection between the two constraints is performed by a meta-unification that leads to the following propagation of constraints:

[11] With primitive relations we mean primitive constraints which are elements of $\mathcal{L}$ in the finite domain framework.

[12] The elements of the meta-variable domains means that between variables X and Y we can have either the relation $<$ or the relation $=$ or $\leq$ or an inconsistent relation. This is necessary for detecting inconsistencies in qualitative reasoning.

$$D_{XY} = D'_{XY} = [\{<\}, \{\}].$$

corresponding to the qualitative domain $X < Y$ in the finite domain constraint solver. The composition should be defined directly in the constraint solver. This operation depends on the particular domain on which the solver works. In [2] Allen defines a transitivity table for the composition of qualitative relations on temporal intervals, while in [51] Vilain and Kautz define transitivity tables for relations on temporal points. Therefore, in the constraint solver there should be an embedded definition of how this operation should be performed i.e., a sort of transitivity table.

It is worth noting that some qualitative frameworks such as Allen's framework [2] and Meiri's framework [36] are not tractable in polynomial time. Therefore, the meta-constraint solver is able to express these constraints but it is not able to process them in polynomial time.

More formally, let D be the set of possible constraints such as, for example, constraints between time points: $\mathrm{D}=\{<, =, >\}$. Let $\mathcal{P}(\mathrm{D})$ be the powerset of D, i.e., the set containing all the possible subsets of D[13]:

$$\mathcal{P}(\mathrm{D}) = \{\{\}, \{<\}, \{>\}, \{=\}, \{<, =\}, \{<, >\}, \{>, =\}, \{<, =, >\}\}.$$

Let now V be a subset of $\mathcal{P}(\mathrm{D})$, i.e., $\mathrm{V} \subseteq \mathcal{P}(\mathrm{D})$, we define the following signature:

$$\Sigma = \{\{\in V\}, +_m, =_m, <_m\}^{14}.$$

The symbols in Σ are interpreted by the Σ-structure $\mathcal{D}$ in the following way. The symbol $\{\in V\}$ is defined as follows: for each $\mathrm{V} \subseteq \mathcal{P}(\mathrm{D})$, $\mathcal{D}$ interprets the constraint $r \in V$ as the usual instance of the set V, therefore, r can assume one value among all the possible elements of V (subsets of constraints). The symbol $+_m$ is interpreted by $\mathcal{D}$ as the composition operator. The composition of two meta-variables R_{XZ} and R_{ZY} admits only those values r_{xy} for which there exists $r_{xz} \in D_{XZ}$ and $r_{zy} \in D_{ZY}$ such that $r_{xy} = r_{xz} \otimes r_{zy}$ where the symbol $\otimes$ represents the composition of primitive constraints and is defined by the transitivity table on the constraints of D (see for example the transitivity table defined by Allen in [2] if you are working with temporal interval constraints, or that defined by Vilain and Kautz in [51] if you are working with temporal point constraints). The symbol $=_m$ is interpreted by $\mathcal{D}$ as the intersection of constraints and in particular it unifies the domains of two variables of $\mathcal{P}(\mathrm{D})$. The symbol $<_m$ is interpreted as the relation "tighter" among constraints. A set of constraints C' is tighter than another set of constraints C" if each pair of values allowed by C' is also allowed by C". In our case, the relation tighter can be translated in the set inclusion. This relation is a partial order relation on the elements of the domain. In fact, it defines a lattice where the bottom is $\perp = \{\}$ and the top is $\top = \mathcal{P}(\mathrm{D})$. Therefore, a constraint imposing a relation on non-comparable elements should fail.

[13] It is worth noting that all the powerset of whatever set of constraints contains always the empty set $\{\}$. For the sake of readability we omit it throughout the paper.

[14] The subscript m in symbols of Σ has been introduced in order to distinguish these operation in the object system and in the meta system.

It is worth noting that, as in the quantitative solver, we should perform an arc consistency on the meta-variables representing arcs of the object level graph. In fact the constraint $R_{XY} =_m R_{XZ} +_m R_{ZY}$ performs a composition of R_{XZ} and R_{ZY} in the way defined above, and intersects the computed domain with the domain of R_{XY}. This constraint behaves in the same way of the constraint $XY\# = XZ + ZY$ described in Section 4.1.

The operational semantics of the qualitative constraint solver will be presented in Section 5.

For the sake of clarity, we should notice that the qualitative constraint solver can work independently from the quantitative part. For example the following qualitative program:

```
qualitative:- R_XY ::  [{<},{=},{<,=},{}],
R_YZ ::  [{<},{}],
R_XZ =_m R_XY +_m R_YZ,
R_KH ::  [{=},{}],
R_KH <_m R_XY
```

contains unary constraints which link the variables to take their values from a set of values, and binary constraints containing operations on meta-variables, i.e., intersection and composition, or relations such as the tighter relation. The propagation of constraints during the computation reduces the variable domains by eliminating values that cannot appear in a solution.

4.3 Interactions between Qualitative and Quantitative solvers

One of the most interesting parts of the temporal reasoner concerns the integration of the qualitative and the quantitative constraint solvers. This integration concerns the propagation of quantitative constraints when qualitative domains are modified and the propagation of qualitative constraints when quantitative domains are modified. For this part we do not follow the Meiri's framework but a more suitable propagation for Constraint Logic Programming.

The first propagation (arrow number 2 in the Figure 1) is straightforward. It is equivalent to the addition in a CLP program of a new qualitative constraint in the quantitative network. For example, if variables X and Y are constrained by $X < Y \vee X = Y$ and they range on the domains $D_X = [1..10]$ and $D_Y = [1..10]$, they have an associated meta-variable R_{XY} which has an associated domain $D_{XY} = [\{<\}, \{=\}, \{<,=\}]$. Suppose now that, for the propagation of constraints associated with X and Y, there is another meta-variable R'_{XY} with the associated domain $D'_{XY} = [\{<\}, \{>\}, \{<,>\}]$. The two meta-variables should be intersected because they refer to the same pair of variables. The intersection between the two qualitative constraints leads to the following propagation of constraints: $D_{XY} = D'_{XY} = [\{<\}]$. Therefore, in the quantitative solver the propagation is performed by adding the new constraint $X < Y$ to the constraint store. Consider working with temporal points, the propagation will eliminate from the domains of variables X and Y values which are incompatible with the constraint "before", and in particular, the propagation leads to $D_X = [1..9]$ and $D_Y = [2..10]$.

It is worth noting that this propagation is not the same reported by Meiri but it is much more suitable for the Constraint Logic Programming implementation.

The other interaction (arrow number 1 in the Figure 1) between the qualitative and quantitative solvers concerns the propagation that should be performed when a change in a quantitative domain influences a qualitative domain. This propagation is much more computational expensive: in fact, we should check if for each value of a quantitative domain of a variable X and each value in a qualitative domain of a meta-variable R_{XY} linking X and Y, there is a value of Y compatible with the previous values. Note that the domain values of X are integers while the domain' values of R_{XY} are relations.

Therefore, we have to build a matrix M where the rows are the possible values of X, the columns represent the possible values of R_{XY} and M_{ij} is a boolean value that represents the consistency (value 1) or the inconsistency (value 0) of the pairs of values i for X and j for R_{XY}. At the end of the propagation if a column of a matrix is zeroed then the qualitative constraint can be eliminated from the domain of R_{XY}. For example, suppose that X has an associated domain $D_X = [3..5]$, Y the domain $D_Y = [6..9]$ and X and Y are related by $X \leq Y$, which in terms of meta-variables is represented by the variable R_{XY} and the domain $D_{XY} = [\{<\}, \{=\}, \{<, =\}]$. Intuitively, the constraint "equal" is no longer entailed by the values of the domains of X and Y. In the matrix M row $M_{1,i}$ is associated to the value 3 of D_X, row $M_{2,i}$ to the value 4 of D_X, row $M_{3,i}$ to the value 5 of D_X. Columns of matrix M should be associated with the elements of R_{XY}. Without any loss of information, columns of M can correspond to single constraints. Therefore, column $M_{j,1}$ is associated with the value "before" of D_{XY} and column $M_{j,2}$ to the value "equal" of D_{XY}, we can see that the second column contains only zeroes. It means that for each combination of values of X and values of R_{XY} there is no value in the domain of Y which is compatible with this combination.

$$M = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

This process is general, but very heavy handed. If we work in a specific domain we can make use of special purpose techniques for the interaction between qualitative and quantitative constraints. For example, if we work in the "temporal point" domain, we are able to define very simple rules of propagation as follows: suppose there are two variables X and Y and a constraint between them (which can be the combination of all Point-Point relations). This constraint depends on the intersection of the domains of X and Y. There are three cases:

[1] the intersection contains more than one element;

[2] the intersection contains only one element;

[3] the intersection is empty.

In the first case the constraints between X and Y will not change because in the intersection there are some values that entail all the constraints. For example, if $D_X =$

[1..10] and $D_Y = [5..14]$ are the domains associated with X and Y, then $D_X \cap D_Y = [5..10]$. If it is the case, we can infer nothing on the constraints between X and Y and, therefore, the domain of R_{XY} does not change.

Another situation occurs when the intersection contains only one value. In this case, the constraint "equal" is entailed by the intersection while all the other values entail the relation "before" or the relation "after" but not both. For example, if $D_X = [1..10]$ and $D_Y = [10..14]$ are the domain associated to X and Y, then $D_X \cap D_Y = 10$. Therefore, if between the variables X and Y there is the constraint $X < Y \vee X = Y \vee X > Y$, the constraint $X > Y$ is no longer entailed by the values of the single variables.

The third case happens when the intersection is empty. Obviously, the constraint "equal" is no longer entailed. Moreover, the values of the variables entail only one of the relations "before" or "after". For example, if $D_X = [1..10]$ and $D_Y = [11..14]$ are the domains associated with X and Y, then $D_X \cap D_Y = \emptyset$. Therefore, if between the variables X and Y there is the constraint $X < Y \vee X = Y \vee X > Y$, then the constraint $X = Y \vee X > Y$ is no longer entailed by the values of the single variables.

It is worth noting that this propagation is equivalent to that of Meiri. In fact, the relationship between qualitative and quantitative constraints stated by Meiri are the following:

- If $0 \in \{I_1, \ldots, I_k\}$, then "="$\in$ QUAL(C);
- If $\exists v > 0$ s.t. v$\in \{I_1, \ldots, I_k\}$, then "<"$\in$ QUAL(C);
- If $\exists v < 0$ s.t. v$\in \{I_1, \ldots, I_k\}$, then ">"$\in$ QUAL(C).

where $\{I_1, \ldots, I_k\}$ is the set of intervals representing the distance between two temporal points.

The first relationship is represented in our system by the fact that the intersection between two domain constraints is not empty. In this case, the distance between the two points contains the value 0 and, therefore, the constraint "=" is entailed. The second and the third relationships correspond in our system to the fact that the intersection between two domain constraints is empty. Therefore, if a value $x \in D_X$ and $y \in D_Y$ exists such that $x > y$, their distance is negative and therefore there exists a value $v < 0$ s.t. $v \in \{I_1, \ldots, I_k\}$. The opposite case happens when $x < y$.

5 Operational Semantics

As already stated before, a constraint system is determined by the definition of the constraint domain and by the operational semantics. The constraint domain of the finite domain constraint solver has been defined in Section 3.1, while the constraint domain of the qualitative system has been defined in Section 4.2.

The operational semantics of a constraint system [28] is defined in terms of "top-down execution", and the behavior of the system is described in terms of transition rules which allow the system to change states. States are represented by triples $\langle A, C, S \rangle$ where A is a multiset of atoms and constraints, C is the constraint store of active constraints, i.e., constraints that are *awake*, and S is the constraint store of passive constraints, i.e., constraints that are *asleep*. The failure of the system is represented by a state called *fail*. The transition system is very general but can be parameterized by defining the set of active and passive constraints and two predicates tailored on the particular domain: *consistent* and *infer*.

In the following, we report the general scheme of the operational semantics of a CLP system, taken from [28], and then we will specialize this scheme for the finite domain solver, for the meta-constraint solver and for the interactions between the two systems.

The first transition is denoted by $\rightarrow_r$ where r stands for *resolution*:

$$\langle A \cup a, C, S\rangle \rightarrow_r \langle A \cup B, C, S \cup (a = h)\rangle$$

if a is selected by the computation rule, a is an atom, and $h \leftarrow B$ is a rule of the program P, renamed to new variables; h and a have the same predicate symbol.

It is worth noting that in this transition we do not need to use a substitution because the term $a = h$ is stored in the constraint store S and will be processed in the following steps. Unification, in fact, can be seen as a constraint solving process.

The $\rightarrow_r$ can lead to a failure if h and a don't have the same predicate symbol for each clause of the program P:

$$\langle A \cup a, C, S\rangle \rightarrow_r fail.$$

The second transition is denoted by $\rightarrow_c$, where c stands for *constraint* and introduces constraints into the constraint store:

$$\langle A \cup c, C, S\rangle \rightarrow_c \langle A, C, S \cup c\rangle$$

where c is a constraint selected by the computation rule.

The third transition rule is denoted by $\rightarrow_i$, where i stands for *inference* and infers new active constraints and modifies (generally simplifies) the passive constraints starting from the current collections of constraints C and S with the predicate $(C', S') = infer(C, S)$

$$\langle A, C, S\rangle \rightarrow_i \langle A, C', S'\rangle.$$

The last transition rule, denoted by $\rightarrow_s$, where s stands for *satisfiability*, tests whether the active constraints are consistent with the predicate $consistent(C)$. In complete constraint systems[15] the predicate $consistent(C)$ holds iff $\mathcal{D} \models \tilde{\exists}$C[16].

$$\langle A, C, S\rangle \rightarrow_s \langle A, C, S\rangle$$

if $consistent(C)$,

$$\langle A, C, S\rangle \rightarrow_s fail$$

otherwise.

[15] Some CLP systems, however, have a incomplete consistency check.

[16] The symbol $\tilde{\exists}$C denotes the existential closure of C

A derivation is a sequence of transitions $\langle A_1, C_1, S_1\rangle \to \ldots \to \langle A_i, C_i, S_i\rangle \to \ldots$ where $\to$ is one of the above-mentioned transitions, the starting state is $\langle Goal, \emptyset, \emptyset\rangle$ and the final state is a state which cannot be rewritten further. This final state is *successful* if the final state has the form $\langle \emptyset, C, S\rangle$, otherwise it is *failed*.

5.1 Operational Semantics of CLP(FD)

The operational semantics of CLP(FD) is defined in [49]. In this Section we will give only a brief outline of it, while in the next Sections we will focus on the operational semantics of the meta-constraint solver and on the linking rules between the two systems.

Firstly, we define the active and the passive constraints in a CLP(FD) language. The active constraints C are unary constraints, i.e., constraints that restrict the values of a single variable to range in a domain. These constraints are expressed as $x \in [m, n]_{m \leq n}$ that is interpreted by $\mathcal{D}$ as the constraints $m \leq x \leq n$ for each pair of integers m and n. On the other hand, passive constraints, S, are binary (in general n-ary) constraints which are built starting from variable, operation and constraint symbols in the computational domain. Some passive constraints are $x + y = 5$ or $x \geq y$ or $x + y = z$ where x, y and z are finite domain variables constrained by active constraints.

The predicate *infer(C,S)* performs the arc-consistency propagation and infers new active constraints by restricting the domains of the variables linked by passive constraints. Therefore, the new sets of constraints C' and S' are inferred by *infer(C,S)* in the way that $\mathcal{D} \models (C \wedge S) \leftrightarrow (C' \wedge S')$. For example, if C is $x \in [0, 15]$ and $y \in [6, 8]$, and the passive constraint store S contains only one element $x < y$, (C', S') are derived from *infer(C,S)* in this way: C' contains $x \in [0, 7]$ and $y \in [6, 8]$ while $S' = S$.

The second predicate to be considered is *consistent(C)* which has to be applied to active constraints and succeeds if C is consistent and fails otherwise. Usually, in a finite domain constraint solver the predicate *consistent* holds if all the variables range on a non-empty domain. Therefore, *consistent(C)* holds if $\forall c \in C$ of the form $x \in [m, n]$, the interval $[m, n]$ is not empty, i.e., $m < n$. Thus, the operational semantics of the CLP(FD) is completely defined.

5.2 Operational Semantics of the Meta-Constraint Solver for Qualitative Constraints

We now define the operational semantics of the meta-constraint solver which reasons on qualitative constraints. The meta-constraint solver reasons on the constraints of the object system. The active constraints of the meta level C_m are defined on the powerset of the object level passive constraints. Suppose that in the object system we have the following passive constraint store: $x < y$ and $y < z \vee y = z$. The meta constraint solver automatically builds two meta variables R_{XY} and R_{YZ} constrained, by unary active constraints, to take their values from the domains $D_{XY} = [\{<\}]$ and $D_{YZ} = [\{<\}, \{=\}, \{<, =\}]$. Therefore, the active constraints of the meta level can be defined as $r_{xy} \in \mathcal{P}(S_{obj})$ where S_{obj} represents the object level passive constraint store. Moreover, the two meta-variables are linked by a meta-constraint which imposes their composition. The passive meta-constraint store, S_m, contains the constraint: $r_{xy} +_m r_{yz} =_m r_{xz}$. Therefore, the active constraint store in the

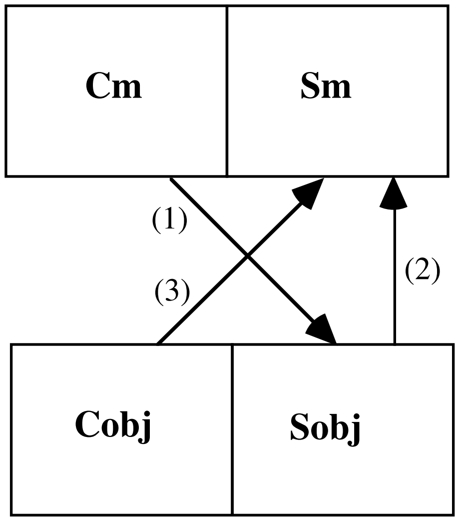

Figure 2: Transitions between the object and the meta constraint stores

meta level contains only unary constraints on meta-variables which impose the variable to range in a domain of sets of constraints, while the passive constraint store contains the binary (in general n-ary) constraints which are built starting from variable, operation and constraint symbols in the computational domain.

The predicates $infer(C_m, S_m)$ and $consistent(C_m)$ are similar to those defined for the object level. In fact, *infer(C,S)* performs an arc consistency algorithm on the meta-variables domains and infers new active meta-constraints by restricting the domains of the variables related by passive meta-constraints. For example, if
$r_{xy} \in [\{<\}]$, $r_{yz} \in [\{<\}]$ and $r_{xz} \in [\{<\}, \{=\}, \{<, =\}]$
are active constraints of the meta level while $r_{xy} +_m r_{yz} =_m r_{xz}$ is a passive constraint, the predicate $infer(C_m, S_m)$ infers C'_m as $r_{xy} \in [\{<\}]$, $r_{yz} \in [\{<\}]$ and $r_{xz} \in [\{<\}]$, while $S'_m = S_m$.

The predicate $consistent(C_m)$ holds if all the meta-variables range on a non-empty domain. Therefore, $consistent(C_m)$ holds if $\forall c \in C_m$ of the form $x \in \mathcal{P}(S_{obj})$, the set $\mathcal{P}(S_{obj})$ is not empty.

It is worth noting that the object-level system and the meta system have the same structure. Therefore, further levels, reasoning on the constraint of the underlying constraint solver, could be added to the system.

5.3 Linking Rules between Object- and Meta-Level

The most interesting feature of the system concerns the interaction between the two levels. A changing in the meta-level should be reflected in the object-level and viceversa. Therefore, some linking rules between the two systems should be defined.

If we consider the constraint stores of the two systems, i.e., the object- and meta-active and passive constraint stores, they can be linked as depicted in the Figure 2.

Consider the transition from the meta level toward the object level (arrow number

(1) in Figure 2). Intuitively, each change in the meta-active constraint store concerns the passive constraints of the object system. The object level passive constraint store has to be modified (extended or simply changed) each time a meta-level active constraint (unary constraint) changes. This transition is formalized by the following linking rule which acts on the active constraints of the meta level and produces new object level passive constraints:

$$\frac{C_m}{S_{obj} \cup S_{obj-m}}$$

where the constraint store S_{obj-m} is derived from the translation of the meta constraints in object level constraints by computing the least upper bound (lub) of the domain constraints of C_m. More formally, for each unary constraint $r_{xz} \in D_{xz}$, $D_{xz} \subseteq \mathcal{P}(D)$ in C_m

$$S_{obj-m} = \bigcup_{r_{xz} \in D_{xz}} lub(r_{xz}).$$

For instance suppose that after the computation of $infer(C_m, S_m)$ the set of active meta constraints C'_m contains the constraint

$$r_{xz} \in [\{<\}, \{=\}, \{<, =\}, \{\}].$$

The lub of the variable's domain is $\{<, =\}$, and this lub represents the constraints that should hold between object variables x and z in the object system.

For the sake of completeness, if a meta variable r_{xz} has an associated domain containing only the empty set, i.e., $r_{xz} \in [\{\}]$, it means that a failure in the object system occurred. This object level failure can be either propagated in the meta system (as a failure) or handled by the meta-reasoner in order, for example, to retract some constraints. On the other hand, a failure in the meta-system corresponds to an empty domain of a meta-variable, i.e., $r_{xz} \in [\,]$.

The other transition (arrow number (2) in Figure 2) concerns the propagation in the meta system when passive constraints in the object system are modified. First of all, a transition must occur when a new passive binary constraint in the object system is encountered. This constraint generates a meta active constraint. It constrains a meta-variable to take its values from a domain which contains the powerset of constraints holding between the two variables occurring in the passive object constraint.

It is worth noting that, as for the transition $\rightarrow_c$ in the two systems, when a new constraint is encountered it should be first added to the passive constraint store. Then the transition $\rightarrow_i$ moves the unary constraints from the passive to the active set and performs the propagation. The transition $\rightarrow_c$ takes a constraint from the set of atoms and constraints A and adds the constraint to the passive constraint store. In the same way, transition (2) in Figure 2 generates a meta-constraint starting from an object level passive constraint and adds this new unary meta constraint to the passive meta constraint store. Then the transition $\rightarrow_i$ in the meta system will move this constraint into the active constraint store and perform the propagation.

The transition can be written as:

$$\frac{S_{obj}}{S_m \cup S_{m-obj}}$$

where the constraint store S_{m-obj} is obtained by the reification of object level constraints into meta constraints by computing the powerset of the binary relations holding between each pair of variables in the object level system. More formally:

$$S_{m-obj} = \bigcup_{\forall x,y \in D,\ x\{r_1,r_2,\ldots,r_n\}y,\ r_1,r_2,\ldots,r_n \in \mathcal{L}} \mathcal{P}(r_1, r_2, \ldots, r_n)$$

For instance, suppose that the set of passive object constraints S_{obj} contains the constraint $x \leq y$ which can be written as $x\{<,=\}y$, the transition creates a meta-variable r_{xy} ranging on the powerset of $\{<,=\}$.

The object-meta interactions involve also the propagation of active meta-constraints when object active constraints do not entail some constraint any more (arrow number (3) in Figure 2). For example, if the object level active constraints are $x \in [1..5]$ and $y \in [7..12]$ and the object level active constraint store contains the constraint $x \leq y$, a transition must occur which eliminates the constraint $x = y$ which is no longer entailed by the variable's domain values. Intuitively, this constraint should be deleted by a propagation in the meta system which performs qualitative reasoning. Therefore, the linking rule from the object to the meta system should add to the passive meta constraint store the constraints entailed by the quantitative values of the object level variables. Again the transition $\rightarrow_i$ in the meta system moves this constraint in the active constraint store and performs the propagation. The corresponding linking rule has the form:

$$\frac{C_{obj}}{S_m \cup S'_{m-obj}}$$

where S'_{m-obj} is the constraint store entailed by the active object constraints. This rule is a special purpose transition for the kinds of constraints we are considering. For the temporal constraint point of view this procedure is described in Section 4.3.

This transition can be also described in terms of $\rightarrow_i$ transition of the object level system. In fact, each time a unary constraint changes, we can directly add to the passive object constraint store the binary relations entailed by the unary constraints. Then the linking rule which links S_{obj} to S_m (arrow number (2) in Figure 2) reifies the new passive object constraints. However, the addition of a new transition to the object level system changes its behavior in some cases. Therefore, it cannot be considered a CLP(FD) solver any more.

It is worth noting that each time a (meta or object level) constraint store changes due to the propagation of other level constraints, the system performs its own propagation in order to check the consistency and to infer new constraints.

5.4 Top-Down Execution

The operational semantics of our system is defined, according to [28], in terms of a transition system on *states*. In the previous Sections we have defined the primitive transitions either in the object system or in the meta system. Furthermore, we have defined three transitions from the object toward the meta system and viceversa.

Now we should define our system behavior by providing one or more sequences of operators (transitions) to be applied to the states of the system which lead it either to a solution or to a failure.

A derivation is a sequence of transitions:

$$\langle A_1, C_1, S_1\rangle \rightarrow \langle A_2, C_2, S_2\rangle \ldots \rightarrow \langle A_i, C_i, S_i\rangle \rightarrow \ldots$$

The initial state depends on the program to be executed. In fact, if the system has to deal with a qualitative and quantitative program, the initial state of the object level is $\langle A, \emptyset, \emptyset\rangle$ and the initial state of the meta constraint solver is $\langle \emptyset, \emptyset, \emptyset\rangle$. The active and passive constraint stores are built during the computation by applying the linking rules.

On the other hand, if the system has to deal with a qualitative program, the initial state of the meta constraint solver is $\langle A_m, \emptyset, \emptyset\rangle$ while the object level module does not perform any computation.

A state which cannot be rewritten further is called *final state*. The *final state* succeeds if it has the form $\langle \emptyset, C, S\rangle$. A derivation fails if it is finite and the final state is *fail*. A derivation flounders if it is finite and the final state has the form $\langle A, C, S\rangle$ where $A \neq \emptyset$.

The behavior of the two systems, i.e., the object and the meta level systems, can be described in terms of their derivations. In particular, let $\rightarrow_{ris} = \rightarrow_r \rightarrow_i \rightarrow_s$ and let $\rightarrow_{cis} = \rightarrow_c \rightarrow_i \rightarrow_s$, the two systems can be described in terms of $\rightarrow_{ris}$ and $\rightarrow_{cis}$. Therefore, they are called *quick checking*, according with the definition of [28].

Moreover, a transition starting from a state with a nonempty collection of atoms and constraints A either fails or contains a $\rightarrow_c$ transition or contains a $\rightarrow_r$ transition. The two systems are said to be *progressive*, in accordance with the definition of [28].

It is worth noting that the two systems are not *ideal*. A CLP system is *ideal* if it is *quick checking*, *progressive*, *infer* is defined as $infer(C, S) = (C \cup S, \emptyset)$ and the predicate $consistent(C)$ holds iff $\mathcal{D} \models \tilde{\exists} C$. Our systems have a different definition of $infer$. In fact, $infer(C, S) = (C', S')$ maintains passive constraints (binary constraints) which are used to modify active (unary) constraints.

An interesting point concerns the transitions from the meta to the object level and viceversa. We should define a partial order between these transitions. Referring to the Figure 2 we should define transitions (1), (2) and (3). Let us start with the transition (1).

When the meta level system performs an $infer$ and a $consistent$ transition, i.e., $\rightarrow_{is}$, it produces new active consistent unary constraints. These constraints should be reflected at the object level via a transition, called $\rightarrow_{c'}$, and added to the object passive constraint store. This transition is very similar to $\rightarrow_c$ of the object system which takes a new constraint c from the set A and adds c to the passive constraint store. The only difference concerns the "source" of the constraint c which, in this case, comes from another level. After a $\rightarrow_{c'}$ transition the object level performs a $\rightarrow_{is}$ transition for processing the new constraints. The $\rightarrow_{c'}$ transition can be written as:

$$\frac{\langle A_m, C_m, S_m\rangle \rightarrow_{is} \langle A_m, C'_m, S'_m\rangle}{\langle A_{obj}, C_{obj}, S_{obj}\rangle \rightarrow_{c'} \langle A_{obj}, C_{obj}, S_{obj} \cup S_{obj-m}\rangle}$$

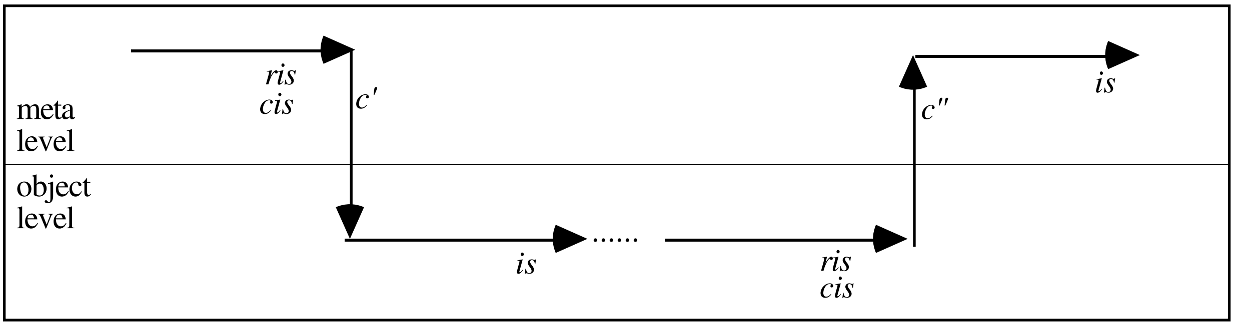

Figure 3: Transitions between the object and the meta systems

where S_{obj-m} has been defined in Section 5.3.

The transitions (2) and (3), which are conceptually separated, can be unified in a unique transition $\rightarrow_{c"}$ which translates active and passive object level constraint stores in passive meta level constraints. Therefore, after an *infer* and *consistent* transition, i.e., $\rightarrow_{is}$ in the object level, the $\rightarrow_{c"}$ transition translates active and passive constraints stores in the meta passive constraint store.

$$\frac{\langle A_{obj}, C_{obj}, S_{obj} \rangle \rightarrow_{is} \langle A_{obj}, C'_{obj}, S'_{obj} \rangle}{\langle A_m, C_m, S_m \rangle \rightarrow_{c"} \langle A_m, C_m, S_m \cup S_{m-obj} \cup S'_{m-obj} \rangle}$$

where S_{m-obj} and S'_{m-obj} have been defined in Section 5.3 and represent respectively the reflection (reification) of S'_{obj} and C'_{obj}.

As for $\rightarrow_{c'}$, the $\rightarrow_{c"}$ transition can be considered a sort of $\rightarrow_c$ transition with the only difference that the constraints added to the passive constraint store are not taken from the set A but come from another level.

The partial order between the two level transitions is depicted in Figure 3. We can notice also that the interactions between the two systems can be described in terms of $\rightarrow_{c'is}$ and $\rightarrow_{c"is}$ transitions. Therefore, the overall system is *quick checking*.

6 The Implementation

The general purpose temporal reasoner has been implemented under the ECRC Common Logic Programming System (ECL^{i}PS^e) [20]. ECL^{i}PS^e is a Prolog-based system suitable for developing Logic Programming extensions.

The implementation of the temporal reasoner is based on the use of Finite Domain (FD) Library integrated in ECL^{i}PS^e. This library implements a constraint solver over finite domains containing integer as well as atomic elements. The syntax and semantics of the domain variables and constraints generally corresponds to that of CHIP [18], although there are some small differences. The FD Library provides a set of predicates for manipulating the variables domains as well as the predicates for defining user-defined constraints.

6.1 Implementation of the Quantitative Solver

For the quantitative solver we have used the finite domain constraint solver embedded in ECL^{i}PS^e. However, we have implemented two extensions: the first one concerns an interface to the system which allows the user to express the variables of the problem and the relations among them in a easier way; the second concerns the possibility of developing a path consistency check by using the arc consistency algorithm of the CLP(FD), following the approach in section 4.1.

Let us start with the representation of the variables and the quantitative relations of the problem. We can deal with temporal points, while translating temporal intervals in terms of their starting and ending points. Of course, in this framework we are able to deal with the "pointisable" fragment of the Interval Algebra [2]. However, intervals are linked only by qualitative relations while their quantitative location is determined in terms of their starting and ending points. Therefore, in this section we will present only the point representation and the quantitative relations between points. The temporal point `P` is represented by the Prolog fact:

```
point(P)
```

where `P` is a domain variable whose domain contains the possible values (temporal locations) for `P`.

Two points `P1` and `P2` can be related by a bound on distance constraint such as:

$$\mathtt{a} \leq \mathtt{P2 - P1} \leq \mathtt{b}.$$

In this case, the user representation of the constraint is the following:

```
distance(P2,P1,D)
```

where `D` is a domain variable whose domain contains the possible values (temporal distances) for the difference `P2 - P1`. In the example above the domain of `D` will be the interval `a..b`.

The variables and the constraints are translated in a CLP(FD) program and completed with the introduction of the constraints representing all the possible paths among variables (see section 4.1). For this purpose, we have introduced two predicates in the solver: the first one `add_relations` completes the network in the sense that it adds all the relations between non-linked variables; the second `add_paths` introduces in the CLP(FD) program all the relations representing paths of the network.

At the end of this step the system passes the control to the finite domain constraint solver which propagates quantitative constraints by using its own propagation mechanisms.

6.2 Implementation of the Qualitative Solver

In this section, we sketch the representation of intervals, of qualitative point-point, interval-interval, interval-point and point-interval relations and the behavior of the qualitative solver.

Intervals are represented by a name and their starting and ending points. In fact, the interval `I` is represented by the Prolog fact:

`interval(I,(I-,I+))`

where `I` is the interval name and `I-` and `I+` its starting and ending points and can be related to other intervals by Interval-Interval relations. The relations point-interval and interval-point of `I` with its starting and ending points `I-` and `I+` are the relation `I-` starts `I` and `I-` ends `I` (see [32]).

Every qualitative relation between two temporal objects is represented by the Prolog fact:

$$\texttt{arc(O}_i\texttt{,O}_j\texttt{,R)}$$

where $\texttt{O}_i$ and $\texttt{O}_j$ are temporal objects while `R` is a list of relations. As already pointed out, we are only able to treat point-point relations. Therefore, we should translate interval-interval, interval-point and point-interval relations in point-point relations involving interval starting and ending point. For this purpose we have implemented a module (called interval-point module) which performs this translation starting from the general `arc/3` facts.

Starting from this representation, the qualitative solver builds the meta-domain variables, ranging on the qualitative relations linking pairs of points, and constrains the meta-variables with the intersection and composition operations. As in the previous case (see subsection 6.1), the solver performs the closure of the network by adding fully relaxed relations between every pair of variables. Then the qualitative solver performs the propagation by eliminating values (relations) from the meta-variable domains that cannot appear in a solution and computes the minimal network and the transitive closure.

As a final step the interval-point module translates point-point relations in the original interval-interval, interval-point and point-interval relations.

6.3 Implementation of the Interactions between the two Solvers

The interactions between the two solvers concern the propagation of quantitative constraints when qualitative constraints change and viceversa. As concerns the first interaction, the implementation is straightforward: in fact, when a qualitative constraint linking two variables changes, a new constraint is added in the CLP(FD) solver which directly propagates the constraint by eliminating values in those domains which are incompatible with the new constraint. This interaction is performed by the predicate `qual_quant/3`.

For example, if a constraint between variables `X` and `Y` is no longer $\leq$ but only $<$, then it should be propagated in the quantitative solver just by imposing:

```
qual_quant(X,<,Y):- X#<Y.
```

The other interaction is more complex. We have implemented a user-defined constraint `quant_qual/3` which implements the quantitative qualitative propagation. This constraint performs the propagation presented in Section 4.3 which eliminates relations between variables which are no longer entailed by the variables domain values.

As an example, we report the structure of the user-defined constraint `quant_qual/3`[17]:

[17] For the syntax refer to [20].

```
quant_qual(X,XY,Y) :-
          dvar_domain(XY,D), dom_copy(D,D1),
          var_fd(XYT,D1), quant_qual(X,XY,Y,XYT),
          dvar_domain(X,DX), dom_size(DX,SX),
          dvar_domain(Y,DY), dom_size(DY,SY),
          (SX = 1, SY = 1 ->
                    true;
                    make_suspension(quant_qual(X,XY,Y),3,Susp),
                    insert_suspension((X,Y), Susp, any of fd, fd)),
          wake.
```

This predicate verifies if each primitive relation in the domain of the meta-variable XY is entailed by the values in the domains of X and Y. For this purpose the predicate

quant_qual(X,XY,Y,XYT)

is called which performs the real propagation. If the check succeeds and if the domains of X and Y contain more than one element, the predicate quant_qual(X,XY,Y) is suspended to be waken when some modifications occur in the domains of X or Y.

7 Some Examples

In this Section we present some examples which apply this architecture. First, we will present an example extracted from Meiri's paper [36] which handles both qualitative and quantitative temporal knowledge.

Example 7.1 *John and Fred usually work at a local office, in which case John takes less than 20 minutes and Fred 15-20 minutes to get to work. Twice a week John works at the main office, in which case he takes at least 60 minutes. Today John left home between 7:05 and 7:10 a.m., and Fred arrived at work between 7:50 and 7:55 a.m. John and Fred met at the traffic light on their way to work.*

For expressing these constraints we use the representation given in Section 6. The representation of the problem is written in a configuration file. This file contains information about the above-mentioned story:

```
interval(J, (J-,J+)),
interval(F, (F-,F+)),
point(P0),
origin(P0)
arc(J, F, [s,si,o,oi,f,fi,d,di,=]),
distance(J-, P0, D1),
distance(F+, P0, D2),
distance(F+, F-, D3),
distance(J+, J-, D4),
D1::0..5, D2::50..55, D3::15..20, D4::0..20,60..max
```

where `J` is the interval representing the event John going to work, while `F` is the interval representing the event Fred going to work and `origin(P0)` is a predicate which holds if and only if `P0` is the beginning of the world. The result of the propagation is the same as that reported by Meiri, but it is interesting to follow the two solvers steps in order to understand their behavior.

Basically, the steps performed by the solvers are the following:

- The module interval-point translates the relations of the Interval Algebra in relations between their starting and ending points. The interval-interval relation:

  ```
  arc(J, F, [s,si,o,oi,f,fi,d,di,=]),
  ```

  is translated to six point-point relations:

  ```
  arc(J-, F-, [<,=,>]),
  arc(J+, F+, [<,=,>]),
  arc(J+, F-, [>]),
  arc(J-, F+, [<]),
  arc(J-, J+, [<]),
  arc(F-, F+, [<]).
  ```

- These relations are transformed in meta-variables of the meta-constraint solver:

  $\mathtt{R}_{J-F-}$ :: $[\{<\},\{=\},\{>\},\{<,=\},\{>,=\},\{<,>\},\{<,=,>\},\{\}]$,

  $\mathtt{R}_{J+F+}$ :: $[\{<\},\{=\},\{>\},\{<,=\},\{>,=\},\{<,>\},\{<,=,>\},\{\}]$,

  $\mathtt{R}_{J+F-}$ :: $[\{>\},\{\}]$,

  $\mathtt{R}_{J-F+}$ :: $[\{<\},\{\}]$,

  $\mathtt{R}_{J-J+}$ :: $[\{<\},\{\}]$,

  $\mathtt{R}_{F-F+}$ :: $[\{<\},\{\}]$,

  linked by meta-constraints:

  $\mathtt{R}_{J-F-} =_m \mathtt{R}_{J-F+} +_m \mathtt{R}_{F+F-}$,

  $\mathtt{R}_{J-F-} =_m \mathtt{R}_{J-J+} +_m \mathtt{R}_{J+F-}$,

  $\mathtt{R}_{J+F+} =_m \mathtt{R}_{J+F-} +_m \mathtt{R}_{F-F+}$,

  $\mathtt{R}_{J+F+} =_m \mathtt{R}_{J-J+} +_m \mathtt{R}_{J-F+}$.

  In this case, the propagation of meta-constraints does not change the meta variable domains.

- The quantitative solver transforms the distance constraints in quantitative relations for the CLP(FD) solver. Moreover, it generates all the possible paths of the graph (see section 4.1) and performs arc consistency on them. Therefore, the quantitative finite domain constraints are the following:

  ```
  J-::0..5, F+::50..55, D3::15..20, D4::0..20,60..max,
  D3 #= F+ - F-, D4 #= J+ - J-, D5 #= F- - J-,
  D6#= F+ - J+, D7#= F+ - J-, D8#= F- - J+,
  D5#=D8+D4, D5#=D7-D3, D6#=D7-D4, D6#=D3+D8.
  ```

  The propagation of constraints leads to the following domains:

  ```
  J-::0..5, F+::50..55, J+::60..max, F-::30..40, D3::15..20,
  D4::60..max, D5::25..40, D6::-max..-5, D7::45..55, D8::-max..-20
  ```

- The fourth step concerns the `qual_quan` and `quan_qual` propagations. The first one adds the constraints[18]:

  ```
  J+ #> F-, J- #< F+, J- #< J+, F- #< F+
  ```

  which do not alter the variable domains because they are already consistent with the constraints.

  On the other hand, the propagation `quan_qual` alters the meta-variable domains and, in particular, the meta-variables representing the following constraints

  ```
  J- [<,=,>] F-, J+ [<,=,>] F+.
  ```

  In fact, the domain of `J-::0..5` and the domain of `F-::30..40` entail only the constraint `J- #< F-` while the domain of `J+::60..max` and the domain of `F+::50..55` entail only the constraint `J+ #> F+`. The propagation of constraints reduces the meta-variables domains. This reduction lead to another propagation in the meta-constraint solver (which performs an *infer* procedure) but this latest propagation does not alter any other meta variable domains (the meta-constraint graph reaches the *quiescence*).

[18] The constraints:

```
J- [<,=,>] F-, J+ [<,=,>] F+,
```

are not added to the quantitative program because they do not restrict the variable domains.

- In the last step the module interval-point translates the relations between starting and ending points in relations of the Interval Algebra. In particular, the only relation entailed by the starting and ending point constraints is `F during J`.

After having determined the minimal network representing the above-mentioned problem, we can be interested in finding a feasible scenario (see, for instance [54]), i.e., an arrangement of the temporal objects along the time line which is consistent with the given constraints. In our architecture this problem can be solved in a straightforward way. In fact, the CLP(FD) primitive `indomain(X)`, when applied to a domain variable `X`, instantiates `X` to a feasible value of its domain. The propagation then eliminates, from other variable domains, those values which are incompatible with `X`. At the end of the propagation, either all the variables are instantiated or another `indomain` occurs, until there are no more unbound variables. In our example, we can instantiate the starting and ending points of the intervals `I` and `J` in order to obtain a feasible scenario. The clause:

```
scenario([]).
scenario([Var|Others]):-
      indomain(Var),
      scenario(Others).
```

instantiates the temporal points of the problem to feasible values.

Usually, according to the *first fail* principle [48], the variable to be instantiated next is the variable with the smallest domain (the most constrained variable). Therefore, the clause `scenario` becomes:

```
scenario([]).
scenario(List_Vars):-
      deleteff(Var,List_Vars,Others),
      indomain(Var),
      scenario(Others).
```

where `deleteff` takes the list of variables and chooses the one with the smallest domain. Other scenarios can be found by using the backtracking mechanism of CLP.

The same problem can be handled in a qualitative network. Each arc represents a set of relations between variables linked by an arc. We could be interested in finding a feasible scenario, i.e., an arrangement of the temporal relations between objects which is consistent with the other constraints, see [45]. In this case the next variable to be instantiated it could be the most constrained one, i.e., the variable with the minimum sized domain. Therefore, the second clause `scenario` can be used also for qualitative constraints. In the example above, the consistent scenario has been already found because meta-variables have no disjunctive constraint but each meta-variable is instantiated to a single primitive relation. In fact, we have the following qualitative constraints:

```
J+ #> F-, J- #< F+, J- #< J+, F- #< F+, J- #< F-, J+ #> F+.
```

In qualitative networks, a very important problem is to decide which value to try next. Usually, the ordering of the values according to their restrictiveness is a domain dependent problem. Tsang [45] proposes some orderings of temporal interval-interval relations, e.g., the more restrictive relations are those imposing a greater number of point-point constraints between their starting and ending points or in planning problems the more restrictive relations are those minimizing the overall duration of the schedules generated. In this case, the "tighter" relations between the constraints of the qualitative constraint solver can be useful in order to decide which value to try next.

8 Related Works

The integration of CLP and temporal reasoning motivated other works such as [9, 34, 25].

The first work proposes an integration of metric and symbolic constraints in a unified framework. They embed variables representing the distance between two intervals (their starting and ending points) into an Allen's symbolic interval-interval constraint. For example, we have the constraint:

```
I before(X) J if I+ + X = J-
```

where `I` and `J` are intervals, `I+`, `I-`, `J+`, `J-` their starting and ending points and `X` is a non negative number. This constraint is called a *metric-symbolic* relation. Then, they extend the concept of a *metric-symbolic* relation by allowing the variable `X` to be a domain variable ranging on a finite domain of integers. Finally, they define a way of coping with the disjunction of relations by implementing a new constraint: the union of domain variables.

The main difference with our work concerns the fact that we separate the qualitative and the quantitative constraints in two different networks and solve them with two different constraint solvers. In fact, while the approach presented in [9] is based on a flat architecture, our approach is based on a meta architecture gaining in expressivity and modularity.

The second work [34] concerns the integration in a finite domain constraint solver of the Interval Algebra qualitative constraints. They do not focus on the integration of qualitative and quantitative information but integrate in a CLP framework only one class of constraints. Our representation of constraints is very similar to their representation and is very easy to use. They use, however, a different propagation method. In fact, for each 3-length cyclic path (each triangle in the graph) they have constraints like:

$$\mathtt{consistent\_triangle}(\mathtt{X}_{ij},\mathtt{X}_{jk},\mathtt{X}_{ik})$$

where $\mathtt{X}_{ij}$, $\mathtt{X}_{jk}$, $\mathtt{X}_{ik}$ are domain variables. In our approach this constraint is represented and managed in the meta-level by the definition of a meta-level operation of composition and intersection. Similar to the previous approach, this one is also based on a flat architecture.

Another related work is that by T. Frühwirth in [25] which proposes the Constraint Handling Rules, a Constraint Logic Programming extension based on the guarded rules of Smolka [44], suitable for handling qualitative temporal reasoning. Frühwirth, with

Constraint Handling Rules, follows the Meiri's framework which handles both qualitative and quantitative temporal constraints.

Constraint Handling Rules are multi-headed guarded rules. There are two kinds of rules:

- Simplification Rules, of the form:

  $H_1, \ldots, H_i \Leftrightarrow G_1, \ldots, G_j \mid B_1, \ldots, B_k$

- Propagation Rules, of the form:

  $H_1, \ldots, H_i \Rightarrow G_1, \ldots, G_j \mid B_1, \ldots, B_k$

A simplification rule corresponds to logical equivalence and is useful in order to replace a constraint by simpler constraints. For example, the identity of two variables can be expressed with a simplification rule as:

```
X ≤ Y, Y ≤ X ⇔ X=Y
```

On the other hand, a propagation rule corresponds to an implication which adds new constraints that are logically redundant (but may cause simplification). For example, the transitivity property can be expressed in terms of the following propagation rule:

```
X ≤ Y, Y ≤ Z ⇒ X ≤ Z
```

These rules define when and how propagation has to be performed and should be written by the user in the code.

We think that a very important relation between our work and that of T. Frühwirth does not only concern the integration of qualitative and quantitative temporal reasoning, but the fact that Constraint Handling Rules represent a very general architecture for prototyping and implementing constraint solvers. Our approach has been implemented by using the CLP(FD) library of ECL^{i}PS^e, but it can be implemented in a straightforward way by using Constraint Handling Rules.

In fact, we can use this Constraint Logic Programming extension in order to write our meta-constraint solver and the linking rules between the object level and the meta level systems. Moreover, we can write some constraint handling rules for stating all the constraints corresponding to the paths of the original network (see Section 4.1). A Constraint Handling rule corresponding to the linking rule between the two systems which translates qualitative relations of the object system into unary relations on meta-variables (arrow number 2 in figure 2) can be written as:

```
arc(X,Y,Rel) ⇒ point(X), point(Y) | power(Rel,P), R_XY =_m P
```

where `Rel` is a list of qualitative relations between points `X` and `Y`, `power/2` is a predicate performing the powerset of `Rel`, while the last constraint limits the range of the meta-variable R_{XY} on the powerset of `Rel`.

Another linking rule, maintaining the correspondence between the active meta constraints and the passive object constraints (arrow number 1 in figure 2), can be the following:

$\mathtt{R}_{XY}\mathtt{=}_m$ `P` $\Rightarrow$ `lub(P,Rel), arc(X,Y,Rel)`

where `P` is the the powerset of `Rel`, and the predicate `lub/2` computes the least lower bound of `P`.

The last linking rule (arrow number 3 in figure 2) maintains in the meta variable domains only the relations entailed by the values of quantitative constraints:

`distance(X,p,D1), distance(Y,p,D2),` $\mathtt{R}_{XY}\mathtt{=}_m$ `P` $\Rightarrow$ `entail(D1,D2,P)`

where `D1` and `D2` are domain variables representing the distances of `X` and `Y` respectively from a point p (the origin of the world). The predicate `entail/3` performs the propagation by eliminating from P those values which are incompatible with `D1` and `D2`.

Another example can be the handling rule that permits the addition of the constraints representing all the paths of the network in order to perform arc consistency on them:

`distance(X,Y,XY), distance(Y,Z,YZ)` $\Rightarrow$ `distance(X,Z,XZ) | XZ#=XY-YZ.`

9 Conclusions and Future Works

We have presented a Constraint Logic Programming two level architecture for handling both quantitative and qualitative temporal reasoning. The architecture is structured into two communicating modules: a finite domain constraint solver for the quantitative reasoning and a meta constraint solver for handling qualitative reasoning. The meta constraint solver reasons on the constraints of the quantitative module and it is able to perform operations on them such as composition and intersection and to state relations between constraints, e.g., the tighter relation. In this paper we have examined, from a theoretical and practical point of view, the structure of the two solvers and, more importantly, the relations and the interactions between the two.

This architecture has been implemented by using the CLP language ECL^{i}PS^e, developed by ECRC and, in particular, its finite domain library.

Some future research directions will be the following:

- The architecture will be extended in order to become a general scheme: by adding modules capable of reasoning on constraints of the underling levels we increase the expressive power and are able to achieve stronger levels of consistency by using weaker levels of consistency algorithms. We are generalizing this framework in order be able to specialize it in different domains without changing its structure. A possible specialization can be for example, a CLP(Intervals) as the object-level system

[10] and a qualitative solver performing the Interval Algebra reasoning as the meta system. This is a work in progress.

- Secondly, we will embed in the meta constraint solver the capability of making some recovery actions on the lower level system. In fact, the meta level is able to detect some inconsistencies of the low level system and can therefore maintain some previous consistent situations in order to recovery from a failure. Therefore, we will investigate the possibility of using some Assumption Truth Maintenance System techniques [17] for handling this kind of problems.

- Finally, we will apply the system to a real life application of scheduling trains. In the past we have developed a scheduler for this application based on another architecture (see [14]). We will port the scheduler under this architecture and we will analyse the advantages and disadvantages of both approaches.

Acknowledgments

This work has been partially supported by the M.U.R.S.T. Project 60% (Ministero dell'Università della Ricerca Scientifica e Tecnologica). We would like to thank Helene Pokidine who helped us in implementing the temporal reasoner.

References

[1] L.Aiello, C.Cecchi, D.Sartini, "Representation and Use of Metaknowledge" in *Proceedings of the IEEE*, vol. 74, No. 10, 1986, pp. 1304–1321.

[2] J.F.Allen, "Maintaining Knowledge About Temporal Intervals", in *Communications of the ACM*, vol. 26, 1983, pp. 832–843.

[3] J.F.Allen, "Toward a General Theory of Action and Time", in *Artificial Intelligence*, vol. 23, 1984, pp. 123–153.

[4] J.F.Allen, "Time and Time Again: the Many Ways to Represent Time", in *International Journal of Intelligent Systems*, vol.6, 1991, pp.341–355.

[5] J.F.Allen, "Planning as Temporal Reasoning", in *Proceedings of KR91*,1991, pp. 3–14.

[6] J.F.Allen, P.J.Hayes, "A Common-Sense Theory of Action and Time", in *Proceedings of IJCAI85*, 1985, pp. 528–531.

[7] J.F.Allen, J.A.Koomen, "Planning using a Temporal World Model", in *Proceedings of IJCAI83*, 1983, pp. 741–747.

[8] F.A.Barber, "A Metric Time-Point and Duration-Based Temporal Model", in *SIGART BULLETIN*, Vol 4, No 3, ACM press, 1993, pp.30–49.

[9] F.A.Barber, R.Berlanga, R.Fordellas, F. Ibañez, G.Martin, F.Toledo, "Integration of Metric and Symbolic Temporal Constraints in CLP", unpublished manuscript.

[10] F.Benhamon, D.McAllester, P.Van Hentenryck, "CLP(Intervals) Revisited", *Tech. Report* CS-94-18 Computer Science Department, Brown University, 1994.

[11] K.A.Bowen, R.A. Kowalski, "Amalgamating Language and Metalanguage in Logic Programming" in *Logic Programming*, K.Clark and S.Tarnlund Eds., Accademic Press NY, 1982, pp. 153–173.

[12] B.C.Bruce, "A Model for Temporal References and its Application in a Questions Answering Program, in *Artificial Intelligence*, vol 3, 1972, pp. 1–25.

[13] P.Codognet, G.Nardiello, "Path Consistency in clp(FD)", in *Proceedings of the First International Conference Constraint in Computational Logics CCL94*, 1994, pp. 201–216.

[14] A.Dalfiume, E.Lamma, P.Mello, M.Milano, "A Constraint Logic Programming Application to a Distributed Train Scheduling Problem", in *Proceedings Practical Applications of Prolog PAP95*, Paris (FR), April 1995.

[15] T.L.Dean, D.W.McDermott, "Temporal Data Base Management", in *Artificial Intelligence*, vol 32, 1987, pp. 1–55.

[16] R.Dechter, I.Meiri, J.Pearl, "Temporal Constraint Networks", in *Artificial Intelligence*, vol. 49, 1991, pp. 61–95.

[17] J. de Kleer, "An Assumption-based TMS", in *Artificial Intelligence*, n.28, vol 3, 1986, pp. 127–162.

[18] M.Dincbas, P.Van Hentenryck, M.Simonis, A.Aggoun, T.Graf, F.Berthier, "The Constraint Logic Programming Language CHIP" , in *Proceedings of the International Conference on Fifth Generation Computer System (FGCS88)*,1988, pp.693–702.

[19] E.Emerson, "Temporal and Modal Logic" in *Handbook of Theoretical Computer Science*, Vol B, Chapter 16, 1990, pp. 995–1072.

[20] ECL^{i}PS^e User Manual Release 3.3, ECRC 1992.

[21] E.C. Freuder, "Synthesizing Constraint Expressions", in *Communication of the ACM*, vol. 21, No11, 1978, pp. 958–966.

[22] E.C. Freuder, A.K. Mackworth, "The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems", in *Artificial Intelligence*, vol.25, 1885, pp.65–74.

[23] E.C. Freuder, A.K. Mackworth, "Special Issue on Constraint Based Reasoning", *Artificial Intelligence*, vol. 58, 1992.

[24] G.Friederich, F. Lackinger, "Diagnosis Temporal Misbehaviour" in *Proceedings IJCAI91*, 1991.

[25] T. Frühwirth, "Temporal reasoning with constraint handling rules", *Tech. Report* ECRC-94-05, ECRC, 1994, Germany.

[26] W. Hamscher, R. Davis "Diagnosis Circuit with State: an Inherently Underconstrained Problem", in *Proceedings AAAI84*, 1984.

[27] J.Jaffar , J.L.Lassez, "Constraint Logic Programming", in *Proceedings of the Conference on Principle of Programming Languages*, Munich 1987.

[28] J.Jaffar, M.J.Maher, "Constraint Logic Programming: a Survey", in *Journal of Logic Programming on 10 years of Logic Programming*, 1994.

[29] H.A.Kautz, P.B.Ladkin, "Integrating Metric and Qualitative Temporal Reasoning", in *Proceedings AAAI91*, pp.241–246.

[30] R.Kowalski, "Logic for Problem Solving", North-Holland,1979.

[31] V. Kumar, "Algorithms for Constraint-Satisfaction Problems: A Survey", in *AI Magazine*, vol. 13, 1992, pp. 32–44.

[32] P.B.Ladkin, R.D.Maddux, "On Binary Constraint Networks", *Tech. Report*, Kestrel Institute, 1989, Palo Alto, CA.

[33] E.Lamma, P.Mello, M.Milano, "Temporal Constraint Handling in Scheduling Problems", will appear on *Proceedings of Intersymp95*, August 1995.

[34] J.Lever, B.Richards, R.Hirsh, "Temporal Reasoning and Constraint Solving", *Deliverable CHIC*, ESPRIT Project EP5291, IC-Park, London, 1992.

[35] J.W.Lloyd, "Foundation of Logic Programming", Second Extended Edition, Springer-Verlag 1987.

[36] I.Meiri, "Combining Qualitative and Quantitative Constraints in Temporal Reasoning", in *Proceedings of AAAI91*, pp.260–267.

[37] A.K. Mackworth, "Consistency in Networks of Relations", in *Artificial Intelligence*, vol.8, 1977, pp. 99–118.

[38] R.Mohr, T.C.Henderson, "Arc and Path Consistency Revisited", in *Artificial Intelligence*, vol.28, 1986, pp. 225–233.

[39] A.Montanari, "Networks of Constraints: Foundamental Properties and Applications to Picture Processing", in *Information Sciences*, vol.7, 1974, pp. 95–132.

[40] B.Nadel, "Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms", in *Search in Artificial Intelligence*, eds. L.Kanal and V.Kumar, Springer Verlsg, NY, 1990, pp.287–342.

[41] W. Nejdl, J. Gamper Harness, "The Power of Temporal Abstraction in Model-Based Diagnosis of Dynamic Systems", in *Proceedings of ECAI 94*.

[42] J.F.Rit, "Propagating Temporal Constraint for Scheduling", in *Proceedings of AAAI90*, 1990, pp.383–388.

[43] E.Schwalb, R.Dechter, "Coping with Disjunctions in Temporal Constraint Satisfaction Problems", in *Proceedings of AAAI93*, 1993, pp.127–132.

[44] G.Smolka, "Residuation and Guarded Rules for Constraint Logic Programming", Digital Equipment Paris Research Laboratory Report, France 1991.

[45] E.P.K.Tsang, "The Consistent Labeling Problem in Temporal Reasoning", in *Proceedings of AAAI87*, 1987, pp.251–255.

[46] S.A.Vere, Planning in Time: Windows and Duration for Activities and Goals", in *IEEE Transaction on Pattern Analysis and Machine Intelligence*, May 1983, Vol PAMI-5, No3, pp246–267.

[47] P.VanBeek, R.Cohen, "Exact and Approximate Reasoning about Temporal Relations", in *Computational Intelligence*, Vol.6, 1990, pp.132–144.

[48] P.Van Hentenryck, "Constraint Satisfaction in Logic Programming", MIT Press, 1989.

[49] P.Van Hentenryck, Y.Deville "Operational Semantics of Constraint Logic Programming over Finite Domains", *Tech. Report* CS-90-23 Computer Science Department, Brown University, 1990.

[50] P.Van Hentenryck, H.Simonis, M.Dincbas, "Constraint Satisfaction using Constraint Logic Programming", *Tech. Report* CS-91-62, Dept. of Computer Science, Brown University, November 1991.

[51] M.B.Vilain, H.Kautz, "Constraint Propagation Algorithms for Temporal Reasoning" in *Proceedings of AAAI89*, 1989, pp. 377–382.

[52] M.B.Vilain, H.Kautz, P. Van Beek, "Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report", in *Readings in Qualitative Reasoning about Physical Systems*, D.S.Weld, J.De Kleer, Morgan Kaufmann, 1990, pp.373–381.

[53] P.Van Beek, "Approximation Algorithms for Temporal Reasoning", in *Proceedings of IJCAI89*, 1989.

[54] P.Van Beek, "Reasoning about Qualitative Temporal Information", in *Artificial Intelligence*, vol.58, 1992, pp.297–326.