Interface Timing Verification with Delay Correlation Using Constraint Logic Programming

Pierre Girodias, Eduard Cerny
LASSO, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Abstract

Using constraint logic programming and relational interval arithmetic, as implemented in CLP (BNR) Prolog, we develop a simple yet complete method for interface timing verification. We show how the problems raised by timing verification (consistency, causality and compatibility) can be formulated as constraint satisfaction problems and solved using relational interval arithmetic when the timing constraints are of the linear, earliest or latest type; we examine the effect of correlation between timing delays (within their specified intervals) and show how an interval delay narrowing method can be applied in this context. The original contribution of this paper is to provide a unifying framework for interface timing verification and to present a method that allows delay correlation to be considered.

1 Introduction

Interface timing verification has the ultimate objective of ascertaining that, given the interface specification of various hardware components, their interconnection will satisfy each others protocol and timing requirements. Each interface is specified as a collection of events and constraints relating the occurrence times of these events.

In practice, several aspects of this problem can be reduced to computing the maximum time separations between pairs of events, usually an *NP*-complete problem [15]. On the basis of this observation, efficient algorithms have been reported for restricted versions of the problem [15, 18, 20]. Unfortunately, these algorithms are very specific and are difficult to extend, for example, to include new kinds of timing constraints.

Girodias et al. [8] showed that the constraint resolution techniques based on relational interval arithmetic (RIA) as used for instance in CLP (BNR) Prolog [1] are computationally equivalent to several of these algorithms [7, 21, 15]. In effect, these interval consistency techniques are complete for special cases of linear constraints, min constraints and max constraints. Furthermore, when linear constraints are combined with either min or max constraints, the completeness results still remain valid. In brief, CLP (BNR) Prolog can solve several instances of the verification problem exactly and when other kinds of constraints are added the solution method can be gracefully extended, at worst, producing false negative results.

We show in this paper that CLP provides a unifying basis for the interface verification problem. In particular, we develop a verification methodology that solves the problems of consistency, causality and compatibility by formulating them as constraint satisfaction problems and solving them efficiently using CLP (BNR) Prolog. Moreover, the expressiveness of constraint languages allows us to incorporate delay correlation into our method in an intuitive way.

It is interesting to note that Bieker and Marwedel [3] have reported similar advantages gained from the use of CLP for the generation of test programs for RTL-level descriptions of hardware.

The paper is organized as follows: In Section 2, we describe interface timing verification. In particular, in Section 2.1, we define a formalism for timing specification, in Section 2.2, we present a methodology for interface timing verification, and, in Section 2.3, we show how to model delay correlation. In Section 3, constraint logic programming is introduced. In Section 4, we describe how interface verification can be formulated in CLP. In Section 5, we show experimental results and then draw some conclusions in Section 6.

2 Interface timing verification

Consider the interface specifications in Fig. 1 and 2, and Table 1. We adopt in this paper the formalism of timing diagrams defined by Khordoc et al. [12]

2.1 Timing diagram specifications

A *timing diagram* (TD) captures the behavior of the interface of a device in terms of ports, waveforms and events.

The TD of Fig. 1specifies the behavior of a memory

ED&TC '97 on CD-ROM

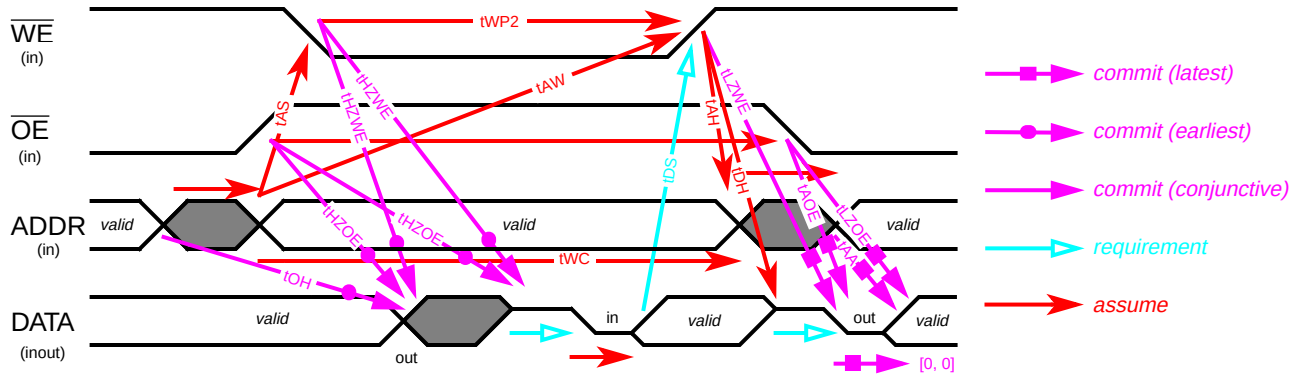

Figure 1 Timing diagram specification of a memory device (end of read cycle - write cycle - beginning of read cycle). Unlabelled arcs denote an unbounded but strict progress in time (> 0).

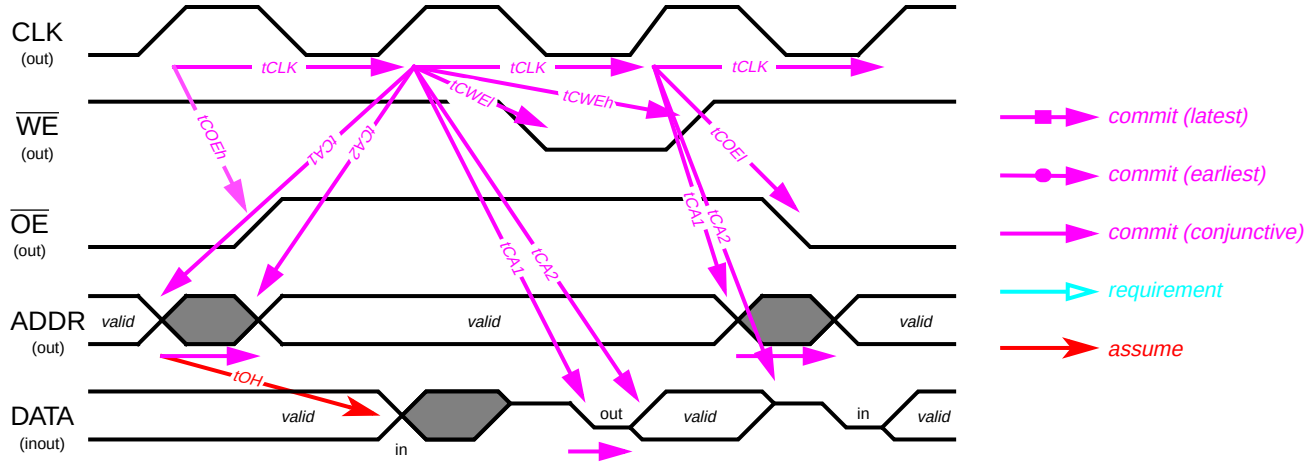

Figure 2 Timing diagram specification of a memory controller. (End of read cycle - write cycle - beginning of read cycle). Unlabelled arcs denote an unbounded but strict progress in time (> 0).

	Name	Min	Max	Name	Min	Max
Memory device	tAA	> 0	10	tHZOE	> 0	∞
	tAH	> 0	∞	tHZWE	> 0	7
	tAOE	> 0	7	tLZOE	> 0	7
	tAS	> 0	∞	tLZWE	9	∞
	tAW	10	∞	tOH	9	∞
	tDH	> 0	∞	tWC	15	∞
	tDS	8	∞	tWP2	12	∞
Memory controller	tCA1	18	19.8	tCOEh	31.8	33
	tCA2	18	20	tCWEl	22	28
	tCLK	25.73	25.73	tCWEh	38.8	45
	tCOEl	16	18	tOH	3	∞

Delays are in ns.

Table 1: Timing constraints for the memory device and memory controller (Fig. 1 and 2)

interface in which the end of a read cycle is followed by a full write cycle and the beginning of another read cycle. Fig. 2 specifies the corresponding behavior of a memory controller to be connected to this memory device.

The signal values on a port are shown as *waveforms*, a sequence of stable values and transitions. The latter are referred to as *events*. Details about signal transitions and their graphical representation can be found in [17]. Events have directions *in* or *out*, either specified explicitly (*inout* ports) or inherited from the port (in or out ports).

Dependencies between events are expressed through timing relations that constrain the occurrence times of events. A *timing relation* specifies an occurrence time window between one or more *source* events and a *sink* event. The occurrence time of the sink event is determined, in accordance with the type of the timing relation, by the occurrence times of the source events. There are three types of timing relations. Let *preds(i)* be the set of source events for a sink event i, and t_j be the occurrence time of an event j.

- *A conjunctive timing relation* constrains the occurrence time of the sink event to fall within the intersection of the timing separations specified relative to each source event:
$$\forall\, j \in preds(i),\ -s_{ij} \le t_i - t_j \le s_{ji}, \quad (1)$$
i.e., $t_i - t_j \in [-s_{ij}, s_{ji}]$, the separation interval.
- An *earliest timing relation* determines that the occurrence time of the sink event be set according to the source event that elicits the earliest response:
$$t_i = min_{j \in preds(i)} (t_j + \delta_{ji}), \quad (2)$$
where $\delta_{ji} \ge 0$ is a delay from event j to event i.
- A *latest timing relation* determines that the occurrence time of the sink event be set according to the last source event that elicits the latest response:
$$t_i = max_{j \in preds(i)} (t_j + \delta_{ji}), \quad (3)$$
where $\delta_{ji} \ge 0$ is a delay from event j to event i[1].

Each timing relation has a specific *intent*. *Commit* relations dictate the behavior of the specified device. *Assume* relations describe behaviors that the device expects from the environment. In other words, commit relations affect *out* events produced by the device, and assume relations describe *in* events produced by the environment. Finally, r*equirements* are relations that must be met once devices are connected.

Definition 1 Timing diagram. A timing diagram is a 4-tuple $TD = (E, C, A, R)$ where

- $E = \{e_0, e_1, \ldots, e_{n-1}\}$ is the set of events characterizing the behavior of the device;
- C is the set of conjunctive, earliest and latest commit timing relations c_i;
- A is the set of assume timing relations a_i;
- R is the set of timing requirements r_i. ❑

The semantics of timing diagram specifications [5, 12] lead us to the verification procedure outlined in the following section.

2.2 A verification methodology

A unique question sums up the interface timing verification problem: Do the specifications describe a feasible space in time once the devices are interconnected? However, circumscribing what is the feasible space is not a trivial task in the presence of conjunctive constraints.

Brzozowski et al. [4] proposed an approach to interface timing verification based on the notions of *consistency* and *satisfiability*. However, Cerny and Khordoc [5] showed that in the presence of conjunctive timing relations these notions are insufficient for verifying that two or more devices can correctly inter-operate based on their interface specifications. It is also necessary to verify that each specification is *causal*.

Definition 2 Consistency. Let t_i be the occurrence time of an event $e_i \in E$. A timing diagram specification $TD = (E, C, A, R)$, $|E| = n$, is consistent if there exists a vector $T = (t_1, t_2, \ldots, t_n)$ such that all the timing relations in $C \cup A$ are satisfied. In other words, given any reference event $e_k \in E$, it is possible to compute time separations to all events $e_i \in E$. ❑

Definition 3 Composition of timing diagrams. The composition $TD = (E, C, A, R)$ of two timing diagrams $TD_1 = (E_1, C_1, A_1, R_1)$ and $TD_2 = (E_2, C_2, A_2, R_2)$, which specifies the behavior of the system resulting from the interconnection of the respective devices, is the union of TD_1 and TD_2 defined as follows:

- $C = C_1 \cup C_2$,
- $E = E_1 \cup E_2$[2],
- $A = A_1 \cup A_2$,
- $R = R_1 \cup R_2$.

We write $TD_1 \parallel TD_2 = TD$[3] ❑

Definition 4 Satisfiability. Given two specifications $TD_1 = (E_1, C_1, A_1, R_1)$ and $TD_2 = (E_2, C_2, A_2, R_2)$, and their composition $TD = (E, C, A, R)$, let $\{T_C\}$ and $\{T_{A \cup R}\}$ be the sets of event occurrence times satisfying respectively C and $A \cup R$. The assumptions (i.e., the assume timing relations) and the requirements made in the specifications of the two composed devices are said to be

1. A delay δ_{ji} can be defined in terms of a lower bound l_{ij} and an upper bound u_{ij} rather than by a single-point value: $0 \le l_{ij} \le u_{ij}$

2. For some events $e \in E$, we have $e \approx e_1 \in E_1$ and $e \approx e_2 \in E_2$, where e_1 is an in event and e_2 is an out event (or vice-versa).

3. There exists a number of ways in which devices specified as timing diagrams can be composed together. In the context of interface timing verification, we restrict the discussion to parallel composition [5, 2] with communication through common events of which only one can be of the out direction.

satisfied by the composition if $\{T_C\} \subseteq \{T_{A \cup R}\}$. ❑

Intuitively, a timing specification is *causal* if future event occurrence times in one device do not restrict the past event occurrence times in another device. In other words, it is sufficient that each device know the occurrence times of past events (specified by any device) and the local commit constraints, to choose the occurrence times of its out events. One way to verify that the preceding holds is to partition events into disjoint subsets, each such subset containing events of one direction only, that can be partially ordered in time, and to verify that the time frame of any subset is independent of later subsets of events.

More formally, Cerny and Khordoc [5] proposed an algorithm to identify causal specifications that is based on the notions of block machines and triggers. A *block* is a set of events of the same direction. A *trigger* is the source event of a timing relation that spans two or more distinct blocks. A *block machine* is the partition of the events and constraints a TD into blocks. (See [5] for a formal description of the semantics of block machines.)

Definition 5 Causality. A timing specification is causal if there exists a block machine such that

(1) Triggers always precede in time the events of the block(s) they trigger (*well-defined triggers* [5]).

(2) Timing separations imposed by the local constraints within a block between its triggers must be less tight than the separations produced by earlier blocks (*past-dominated machines* [5]). ❑

For the verification of the second condition, assume that two events *a* and *b* are the triggers of a block B_k. Then the timing separation imposed between *a* and *b* by the constraints of the set of blocks $\{B_i\}$ where B_i is a block that precedes B_k in topological order, is tighter than the separation implied by the local constraints of B_k.

Definition 6 Compatibility. Two timing diagrams $TD_1 = (E_1, C_1, A_1, R_1)$ and $TD_2 = (E_2, C_2, A_2, R_2)$ are compatible if

(1) TD_1 is consistent and causal,

(2) TD_2 is consistent and causal,

(3) The composition $TD_1 \parallel TD_2 = TD = (E, C, A, R)$ is consistent,

(4) $\{T_C\} \subseteq \{T_{A \cup R}\}$ (satisfiability in *TD*). ❑

The previous definitions can easily be generalized for any number of interconnected devices. Note that the steps in the definition of compatibility implicitly define a verification methodology.

A possible way to address the issues of consistency, satisfiability and causality is to compute the maximum achievable separation [15]

$$d_{ij} = \max(t_j - t_i) \qquad (4)$$

between pairs of events *i* and *j*, occurring at times t_i and t_j, respectively. This is equivalent to solving the underlying constraint system [8].

2.3 Delay correlation

Assuming that there exists no correlation between the delays of the various interacting components may produce wider time intervals than possible in some practical applications, because delays on a single chip are usually correlated to within 10% and delay constraints related to the same wire should be nearly 100% correlated. Without this correlation taken into account, the verification could be needlessly pessimistic.

The method (originally introduced by W. Older for simple delay networks and with a different correlation function) to be described can deal with correlation within any subsets of delays in the overall specification, however, for simplicity we shall assume here that all delays on commit timing relations are mutually correlated, i.e., all belong to the same subset. To illustrate delay correlation, suppose that we have two timing delays: $\delta_1 \in [l_1, u_1]$ and $\delta_2 \in [l_2, u_2]$.

Alternatively, each delay can be written in a normalized form [11] as $\delta_1 = l_1 + \Delta_1 \cdot z_1$ and $\delta_2 = l_2 + \Delta_2 \cdot z_2$, with $\Delta_i = u_i - l_i$, $z_i \in [0, 1]$ ($i = 1, 2$), where z_i are random variables.

If δ_1 and δ_2 are correlated, it may be approximated [11] as $|z_1 - z_2| \leq \varepsilon = 1 - \gamma$ where γ is the degree of correlation. That is, if $\gamma = 1$ then they are 100% correlated meaning that $z_1 = z_2$, while $\gamma = 0$ means that the delays are not correlated and z_1 and z_2 can be chosen freely from the interval [0,1]. Typically, $\gamma = 0.9$ for correlation of delays on a single chip, leaving 10% for local variation.

The introduction of correlation between delays makes certain combinations of delays impossible and thus assume/requirement timing constraint violations that could occur under uncorrelated delays may not happen when correlation exists.

Most existing timing verification methods have great difficulty with handling correlation. In general, it degenerates to a form of enumeration of possible delays. In our approach, we first replace the constraint between z_1 and z_2 by an equivalent one that refers to a common random variable reference, with a local random choice to within ε of that reference. This allows easy generalization to *n* timing delays.

Let *p* be a random variable, $p \in [0, \gamma]$, and x_k, $k = 1, \ldots, n$, be random variables such that $x_k \in [0, \varepsilon]$, and let $z_k = p + x_k$. Clearly for any *p*, x_i and x_j, $i, j = 1, \ldots, n$, $|z_i - z_j| = |(p + x_i) - (p + x_j)| = |x_i - x_j| \leq \varepsilon$, and the combined effect is such that the effective values of z_k, $k = 1, \ldots, n$, range over [0,1]. Assuming uniform distributions for *p* and x_k, we can describe the intervals defining the k^{th} timing delay using the possible intervals of *p* and x_k. Since for any given chip *p* can be any value from $[0, \gamma]$, this is its interval value. For each z_k the interval becomes $z_k \in [0, \gamma + \varepsilon] = [0, 1]$ and thus the individual delays are as specified, namely $\delta_k \in [l_k,$

u_k].

By substituting x_k and p into the equations for z_k and ultimately δ_k, the constraints characterizing equation for δ_k can be written as

$$\delta_k = l_k + \Delta_k \cdot (p + x_k),\ 0 \le p \le \gamma,\ 0 \le x_k \le \varepsilon \qquad (5)$$

where $1 \le k \le n$.

These constraints are added to the timing relations obtained from the TDs. It would now seem that due to the initial interval of [0, γ] on p and [0, ε] on x_k we have not introduced any correlation effect. This is true only if in the original system of timing constraints (i.e., without delay correlation) all specified delays are possible. However, using CLP based on RIA, when the interval of uncertainty associated with a timing delay δ_k is reduced due to interval propagation through the original system of timing constraints, this reduction may back propagate to the common variable p, reducing its interval. In turn, this reduction in p propagates to all the other correlated timing delays which reduces the uncertainty intervals on the occurrence time of events, etc., thus possibly eliminating a false timing violation (false under correlated delays).

CLP (BNR) Prolog includes the relational interval arithmetic framework necessary to solve these complex systems of constraints. The next section introduces the underlying constraint logic programming paradigm.

3 Constraint logic programming

Constraint logic programming (CLP) [6, 10, 19] provides a unified framework for the modeling, the analysis and, ultimately, solving constraint satisfaction problems (CSP).

3.1 Constraint satisfaction problems[4]

The following definitions are based on [19].

Definition 7 Constraint Satisfaction Problem. A CSP is a 3-tuple $P = (I, D, C)$ where

- $I = \{x_0, x_1, \ldots, x_{n-1}\}$ is a finite set of variables,
- $D = \{D_0, D_1, \ldots, D_{n-1}\}$ is a finite set of domains,
- C is a set of constraints.

Variable x_i takes values from domain D_i. We denote this by $x_i \in D_i$. ❑

Definition 8 Constraint. A constraint $c(x_0, x_1, \ldots, x_{n-1}) \in C$ between the variables of I is a subset of the Cartesian product $D_0 \times D_1 \times \ldots \times D_{n-1}$ which specifies which values of the variables are compatible with each other. ❑

In practice, this subset does not need to be given explicitly, but can be defined by equations, inequalities, or programs.

Definition 9 Solution of a CSP. A solution s to a CSP P is an assignment of values to all variables, such that it satisfies all the constraints. We write $P \models s$. Let S be the set of all possible solutions. By extension, we write $P \models S$. ❑

Definition 10 Global consistency [16, 14]. A CSP $P = (I, D, C)$ is globally consistent if and only if $\forall\ x_j \in I$, $\forall\ a \in D_j$, $x_j = a$ belongs to a solution of P. A constraint system is inconsistent if it admits no solution. We then write $P \models \varnothing$. ❑

Definition 11 Equivalence between CSPs. Two CSPs P and P' are equivalent if and only if they have the same set of solution:

$$P \models S \Leftrightarrow P' \models S$$ ❑

Solving a CSP $P = (I, D, C)$ (solving the set C of constraints) can be viewed as the process of finding an equivalent CSP P' which is globally consistent.

The search for global consistency is an *NP*-hard problem. Criteria that are not as strict as global consistency are therefore aimed for, e.g., partial consistency, where some conditions are guaranteed over all the elements of a domain or over the bounds of a domain [13].

3.2 CLP (BNR) Prolog and partial consistency

The algorithms and heuristics used to solve CSPs are referred to as *consistency techniques*. Many different ones have been elaborated, sometimes to address issues specific of a particular problem: local propagation of known states, relaxation, propagating degrees of freedom, etc.

In timing verification, constraints (timing relations) between events are usually specified as min-max intervals of values, with one or more punctual exact solutions lying somewhere in between. Therefore, to treat the interface timing verification problem as a CSP, a solution method based on relational interval arithmetic (RIA) seems natural.

RIA is implemented in CLP (BNR) Prolog [1] in such a way that the natural unification mechanism of Prolog is supplemented with a consistency technique based on the propagation of the lower and the upper bounds of interval-valued variables. For details on the symbolic aspect of interval propagation, the reader is referred to [8].

The interval values of variables returned by CLP (BNR) Prolog are sometimes larger than the true intervals [8]. The space they define can be wider than the exact solution space to the problem. A solution may be returned when, in fact, the exact solution may be the empty set and the system is inconsistent. Only failure to return a satisfying solution set (i.e., one or more interval-valued variables are equal to the empty interval) can be accepted as a definite exact answer: the constraint system is inconsistent.

Fortunately, this does not constitute an insurmountable difficulty. Certain classes of problems can be shown to produce only exact solutions. Other classes of problems can still be analyzed by reasoning on the complement of the

4. Constraint networks.

solution space and through case analysis. For instance, the larger intervals that might result from mixing linear, min and max binary constraints can be avoided by decomposing the set of non-linear constraints into sub-cases that have exact solutions (e.g., a min constraint can be split by imposing for each possible sub-case a specific minimal value). Furthermore, the introduction of delay correlation in timing specifications yields ternary constraints that can be solved by enumerating sub-intervals of variables.

In particular, Girodias et al [8] showed that the systems of linear, max and min constraints that are typical of timing verification, could be solved exactly with CLP (BNR) Prolog.

4 Interface timing verification with CLP (BNR) Prolog

Analyzing a problem in CLP (BNR) Prolog is done by constructing a program, a program being a sequence of logic propositions (see [6] about the construction of Prolog programs.) If the problem admits one or more solutions, submitting the program to the resolution engine will produce them; if not, the program will be rejected.

4.1 From timing diagrams to CSPs

The program to be constructed in CLP (BNR) Prolog is the expression of a CSP. It is interesting to note that the notions of consistency are similar in the contexts of interface timing verification and CLP. In fact, formulating timing diagram specifications to analyze the issues of consistency, satisfiability, causality and compatibility as CSPs is a simple process. Given a specification $TD = (E, C_{TD}, A, R)$, we produce a CSP $P = (I, D, C_P)$ where

- $e_i \in E \Leftrightarrow t_i \in I$, t_i is the occurrence time of an event i,
- $C_P \subseteq F_C(C_{TD}) \cup F_A(A) \cup F_R(R)$, where F_x are functions that transform the original sets of timing relations to produce new sets.

Depending on the problem to be solved and how the set of constraints C_P is constructed from the set of timing relations, we then verify that $P \models \varnothing$ or $P \not\models \varnothing$. For example, a specification $TD = (E, C_{TD}, A, R)$ is consistent if $P = (I, D, C_P) \not\models \varnothing$, where $C_P = C_{TD} \cup A \cup R$.

Given $\neg q$, the complement of a constraint q[5], then satisfiability in a composition $TD = (E, C_{TD}, A, R)$ can be checked by constructing distinct CSPs $P_i = (I, D, C_P)$ where $C_P = C_{TD} \cup \{\neg q_i\}$, $q_i \in A \cup R$, such that $\forall P_i$, $P_i \models \varnothing$.

Causality of a specification can also be ascertained by constructing a series of CSPs that should prove inconsistent. For instance, to verify the second condition of causality (Def. 5), the specification is partitioned in blocks B_i ordered according to a topological order. If the events a and b are triggers of a block B_i such that the local constraints of B_i (constraints defined over the events assigned to the block) imply $a - b < x$, and P_j is the CSP that models the set of blocks $\{B_j\}$, $0 < j < i$, and includes the constraint $a - b \geq x$, then $P_j \models \varnothing$. In other words, the separation between a and b in B_i is tighter than the separation imposed by earlier blocks. The first condition can be verified in a similar manner.

The reader is referred to [9] for a formal expression of the notions of satisfiability, causality and compatibility in terms of CSPs.

Extending CSP formulations to include delay correlation is simple. The set of variables and the associated domains are expanded to include the extra variables required to specify delay correlation. The constraints relating these new variables are added to the set of constraints.

Note, however, that the above results address satisfiability (in the general sense), a decision problem, while one may also be interested in finding optimal solutions: the maximal separation between pairs of events under restricted classes of constraints. Using an appropriate formulation in CLP (BNR) Prolog will implicitly produce this information.

Specifically, since the event occurrence times are constrained only relative to each other and no absolute time reference is given, it is necessary to designate one event as the reference for all other events, and set its occurrence time to the value 0. All other event occurrence times are originally considered to lie in the interval $[-\infty, \infty]$. It follows that the event occurrence time intervals computed by the CLP (BNR) Prolog resolution mechanism are in fact the timing separations between the reference event and the other events. If there are n events and event k is the reference point, we have $T_i = [-d_{ki}, d_{ik}]$, where T_i is a variable representing the domain of the occurrence time t_i of event i, $1 \leq i \leq n$. By solving the constraint system for n different reference points, all timing separations can be found.

The next section illustrates our approach on a simple example.

4.2 Example

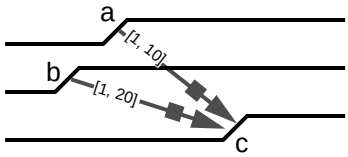

Figure 3 Specification with a latest timing relation.

Consider the timing diagram specification of Fig. 3. This latest timing relation could be expressed in CLP (BNR) Prolog syntax by the following program

```
p(Ta, Tb, Tc):-
```

5. To be correct, two constraints usually result from the construction of the complement.

```
    Dac: real(1, 10),
    Dbc: real(1, 20),
    { Tc == max(Ta + Dac, Tb + Dbc) }.
```

where `Ta`, `Tb` and `Tc` are interval-valued variables representing the domains of the occurrence times of events *a*, *b* and *c*, and `Dac` and `Dbc` are variables constrained by the intervals [1, 10] and [1, 20], respectively. Calling the program `p()` with either of the three parameters (i.e., the event occurrence times) set to the value 0 constitutes a test for consistency and will report a subset of all the maximum achievable separations from the reference event [8].

Let now the delays `Dac`, `Dbc` be correlated by a factor γ. Based on (5) in Section 2.3, a simple modification of the initial model is required:

```
p(Ta, Tb, Tc, Gamma, P):-
  % original formulation
  Dac: real(1, 10),
  Dbc: real(1, 20),
  { Tc == max(Ta + Dac, Tb + Dbc) },
  % new information
  Epsilon is (1.0 - Gamma),
  P: real(0.0, Gamma),
  Xac: real(0.0, Epsilon),
  Xbc: real(0.0, Epsilon),
  { Dac == 1 + (9 * (P + Xac)) },
  { Dbc == 1 + (19 * (P + Xbc)) }.
```

Two new parameters were added to the program. The first, `Gamma`, allows to experiment with various degrees of correlation. The second one, `P`, is used in queries to eliminate possible false negatives. For example, the consistency of the specification could be checked for a correlation factor of 75%:

```
?-p(Ta, Tb, Tc, 0.75, P),
  { Ta == 0 },
  solve(P).
```

With the introduction of linear constraints with three variables (instead of two) due to correlation and the max constraint, the CSPs produced as solutions by CLP (BNR) Prolog are no longer globally consistent. An enumeration of sub-intervals is required (function `solve()`). However, the impact on efficiency of this enumeration is minimal in practice. Since the enumeration takes place only (if ever—inconsistencies could be detected beforehand) after the network of constraints has been loaded and evaluated, its scope is limited both in the number of steps that must be taken by the resolution engine, and in the length of these steps: enumerating one domain may render the enumeration of other domains unnecessary; and, contrary to a systematic initial enumeration requested by the original definition of the min constraint, the network of constraints is built only once in this case.

We stress again that while the model presented here only allows for one set of delays to be correlated, extending it to multiple sets and hierarchical correlation relations is trivial.

5 Experimental results

We programmed a prototype in CLP (BNR) Prolog that accepts two specifications as input and outputs results regarding the issues of consistency, causality, satisfiability and compatibility. Recall Figures 1 and 2. They illustrate the timing diagram specifications of a memory device and a memory controller to be interconnected. Assume that the delays associated with the earliest and latest timing relations are correlated in each of the devices. Tables 2 and 3 summarize their verification with CLP (BNR) Prolog. Note that different degrees of correlation allow certain violations of timing relations to be ruled out. Table 4 reports execution statistics.

	Mem. device	Mem. controller
Consistency	✓	✓
Causality	✓	✓
Compatibility	possible incompatibility (see Table 3)	

✓ denotes that the test is successful.

Table 2: Results of the verification

	Delay Correlation Factor		
Timing delay	**0%**	**50%**	**75%**
tAH	✘✘✘	✘✘✘	✓
tDH	✘✘✘	✘✘✘	✓
tWP2	✘✘✘	✓	✓

✘✘✘ denotes a timing violation;
✓ denotes that the timing constraint is satisfied.

Table 3: Some possible violations reported by the satisfiability check.

	Nb. events	Nb. cons.	CPU times†			
			Caus.	Cons.	Satis.	Comp.
dev	15	22	5.16	0.51	N/A	N/A
ctrl	22	17	0.55	0.45	N/A	N/A
dev/ctrl	22	39	N/A	1.03	1.12/ 3,07‡	5.98/ **16.56**‡

†In seconds on a on a Sparcstation 10 with 128 MB.
‡Average/total for distinct correlation factors (0%, 50%, 75%).

Table 4: Performance measures

Other results (without delay correlation) regarding the

event separation calculation on the example from [15] can be found in [8].

6 Conclusions

We have shown in this paper how CLP (BNR) Prolog, and CLP in general, can be used for efficient interface timing verification. In terms of computational complexity, measures can be reported for restricted cases of the problem [8]. However, in the general case, the exact complexity is difficult to evaluate due to the event-driven nature of the evaluation mechanism and because sub-interval enumerations may be required. Performance measures are not reported but our experience shows, however, that the execution times for practical interfacing problems are never an issue. The examples we presented are typical: interface timing specifications have rarely more than 50 events and at most 100 constraints.

The lack of benchmarks is unfortunate. Yet, the fact that there exists no other "complete" method to establish a comparison is an interesting result in itself. In particular, CLP (BNR) Prolog allows delay correlation to be considered, an issue that is ignored by most other verification techniques.

Acknowledgments: the research was supported by NSERC Canada grants and by Nortel Technologies Ltd.

7 References

[1] F. Benhamou and W. J. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*. To appear in 1996.

[2] B. Berkane, S. Gandrabur, and E. Cerny. Timing Diagrams: Semantics and Timing Analysis. In *Proc. of the Third Asia Pacific Conf. on Hardware Description Languages, APCHDL'96*. pp. 112-119, Bangladore, India, January 1996.

[3] U. Bieker and P. Marwedel. Retargetable Self-Test Program Generation Using Constraint Logic Programming. In *Proc. of the 32nd Design Automation Conference, DAC'95*. San Francisco, CA., June 1995.

[4] J. A. Brzozowski, T. Gahlinger and F. Mavaddat. Consistency and Satisfiability of Waveform Timing Specifications. In *Networks*, 21:91-107. 1991.

[5] E. Cerny and K. Khordoc. Interface Specifications with Conjunctive Timing Constraints: Realizability and Compatibily. In *Proc. of the 2nd AMAST Workshop on Real-Time Systems*. Bordeaux, June 1995. Full length paper accepted subject to revision to ACM TODAES,

[6] A. Colmerauer. An Introduction to Prolog III. In *Communications of the ACM*, 33(7):69-90. July 1990.

[7] R. W. Floyd. Algorithm 97: Shortest Path, *CACM* 5(6):345. June 1962.

[8] P. Girodias, E. Cerny and W.J. Older. Solving Linear, Min and Max Constraint Systems Using CLP based on Relational Interval Arithmetic. In *Theoritical Computer Science, CP'95 Special Issue*, Vol. 173. *To appear in February 1997*.

[9] P. Girodias, E. Cerny, Interface Timing Verification with Delay Correlation Using Constraint Logic Programing. Report, Université de Montréal, September 1996.

[10] J. Jaffar and M. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503-581. 1994.

[11] H. F. Jyu, S. Devadas, and K.W. Keutzer. Statistical Timing Analysis of Combinational Logic Circuits, *IEEE Transactions On Very Large Scale Integration (VLSI) Systems*, 1(2):126-137. June 1993.

[12] K. Khordoc, M. Dufresne, E. Cerny, P.-A. Babkine and A. Silburt, Integrating Behaviour and Timing in Executable Specifications. *Proc. of the IFIP Conf. on HDL and their Applications*, pp. 385-402. 1993.

[13] O. Lhomme. Consistency techniques for numeric CSPs, In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence*. 1993.

[14] A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99-118. 1977.

[15] K. McMillan and D. Dill. Algorithms for Interface Timing Verification. In *Proc. of the IEEE Int. Conf. on Computer Design, ICCD'92*, pp. 48-51. 1992.

[16] U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, 7(2):95-132. 1974.

[17] P. Rony. Interfacing Fundamentals: Timing Diagram Conventions. *Computer Design*, pp. 152-153. January 1980.

[18] P. Vanbekbergen, G. Goossens, and H. De Man. Specification and Analysis of Timing Constraints in Signa Transitions Graphs. In *Proc. of the European Conf. on Design Automation, EDAC'92*, pp. 302-306, Brussels, Belgium. 1992.

[19] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. MIT Press, Cambridge, MA, 1989.

[20] E. A. Walkup, and G. Borriello. Interface Timing Verification with Applications to Synthesis, *Proc. of the Design Automation Conf.*. 1994.

[21] S. Warshall. A Theorem on Boolean Matrices, *JACM* 9:11-12. 1962.