Solving Cyclic Constraints

Stavros Tripakis*

Abstract. We present an algorithm to check feasibility and find solutions on a set of timing constraints of repetitive nature. Our model permits to specify a real-time system executing an infinite number of *cycles*, where at each cycle a set of *events* must occur. The occurrence times of events inside the same cycle, and between consecutive cycles is guarded by a set of simple linear constraints. Given such a model, we show how to check *feasibility* (does the system has an infinite execution where all constraints are met ?) and how to build an execution. A prototype implementation demonstrates the efficiency of our technique even for systems of large size.

Keywords. Linear-constraint solving, Constraint propagation, Real-time systems, Verification, Scheduling.

1 Introduction

Real-time systems are often designed to execute in *cycles*, which are to be repeated (in principle) an infinite number of times. A cycle represents a time interval in which a set of *events* occur. Such an event could correspond to polling a sensor, activating a link, calling a procedure, and so on. It is often the case (especially in *hard* real-time systems) that correctness depends on a number of *timing constraints* on these events. For example, a sensor must be polled at least every 20 milliseconds, or a message should be sent at least 10 and at most 50 milliseconds after the link is activated, and so on. If the number of events in a cycle is large, the number of timing dependencies between events can become also large and complex. Then, it is not at all obvious that an infinite execution that meets all the constraints exists. Even if it does exist, it may be very difficult to find just by looking at the specification.

These are the problems we tackle in this paper. Given a set of events and a set of timing constraints, we answer the following questions:

- Are the constraints *feasible*, that is, does there exist an infinite execution of the system where at each cycle all the events occur and all constraints are met ?

- If the answer to the previous question is yes, then can we find such an infinite execution ?

More precisely, we introduce the model of a *cyclic constraint system* (CCS). A CCS consists of a finite set of events $\{a, b, ...\}$ and a finite set of simple linear constraints. Each event must occur exactly once every cycle. Each of the constraints can be of one of the following types: $a - b \geq k$, $a - b \leq k$, $a' - b \geq k$, or $a' - b \leq k$, where a, b are events and k is a natural constant. In-

*Currently at the University of California at Berkeley (stavros@eecs.berkeley.edu). This work was done while the author was still at Verimag, France.

tuitively, $a, b, \ldots$ are used as variables, representing the occurrence times of events $a, b, \ldots$ at a given (current) cycle, while $a', \ldots$ represent the occurrence times of events $a, \ldots$ at the *next* cycle. For example, the constraint $a - b \geq k$ (resp. $a - b \leq k$) means that, at each cycle, event a should occur at least k time units after event b (resp. b should occur at most k time units after a). The constraint $a' - b \geq k$ (resp. $a' - b \leq k$) means that, at each cycle, b should occur at least k time units before the occurrence of a in the next cycle (resp. b should occur no more than k time units before the occurrence of a in the next cycle). An example of the execution of a CCS $(\{a, b, c, d\}, \{a - b \leq 3, b' - d \geq 5\}$ is shown in figure 1.

Using a CCS we can easily specify, for instance, periodic events: the two constraints $a' - a \leq 5$ and $a' - a \geq 5$ (that is, $a' - a = 5$) a occurs every 5 time units. *Partial orders* are also straightforward to specify, for instance, $b - a \leq 1$ and $c - a \leq 1$ say that b and c occur after a at each cycle, however, no specific order between them is assumed.

Notice that the occurrence of an event is taken to be instantaneous.[1] Also notice that the amount of time spent at each cycle need not be the same: the CCS with only one event a and no constraints has a trivial infinite execution where a can occur at anytime, as long as it occurs infinitely often. The notion of cycle is therefore implicit in the model: a cycle is delimited only by the two consecutive occurrences of an event.

Not all CCSs are feasible. For example, the CCS with two events a and b and constraints $a' - a = 3$ and $b' - b = 4$, does not have any infinite execution: the first constraint delimits cycles to a period of 3, while the second requires them to have a period of 4.[2]

Given a CCS we would like to check whether it is feasible, and if it is, construct an infinite execution (this would correspond to building a *real-time schedule* for the system). We provide an algorithm to do this effectively in section 3. To check feasibility, we introduce a novel, to our knowledge, technique, called *constraint induction*. It consists in iteratively strengthening a set of constraints J, until J becomes either *stable* (i.e., cannot be strengthened anymore) or unsatisfiable. If J becomes stable, then we show how to extract a solution of it and how to extend this solution periodically, to build an infinite execution. If J becomes unsatisfiable then the original CCS is not feasible.

We have implemented our method in a prototype tool, called CINDY, and we have performed a number of experiments, to test the practicality of our techniques. The performance results, reported in section 4 are quite encouraging: CINDY solves cyclic constraints of up to 500 events in less than 15 minutes.

2 Cyclic Constraint Systems

A *cyclic constraint system* (CCS) is a pair $S = (A, \Phi)$, where:

[1] Talking about instantaneous events is enough to capture *tasks* that may need to be executed inside a cycle: a task can be seen as two events, say a and b, representing the beginning and the end of the task. Both a and b should occur in a cycle and a has to precede b by as many time units as the time necessary to execute the task.

[2] If we want to specify more than one periodic events with different periods, we would have to use a unique "global" cycle which encompasses one or more periods for each event: in the previous example, this global cycle would have period 12. Of course, in that case we would also have to duplicate the event variables for each event, as many times as the number of periods of this event in the global cycle.

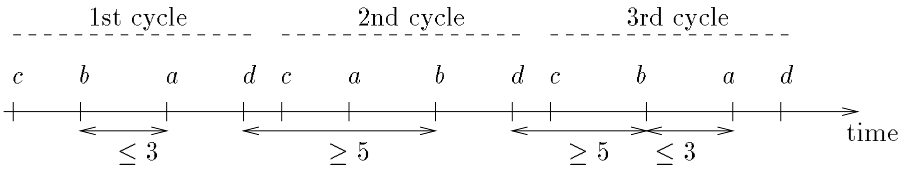

Figure 1: Example of execution of a cyclic constraint system with four events a, b, c, d and two constraints, $a - b \leq 3$ and $b' - d \geq 5$.

- A is a finite set of *events*;
- Φ is a finite set of *timing constraints* $\{\phi_1, \cdots, \phi_m\}$, each ϕ_i being defined according to the following syntax:

$$\phi \ ::= \ a - b \sim k \ \mid \ a' - b \sim k$$

  where $a, b \in A$ are events, $\sim \in \{\leq, \geq\}$ is a binary order relation and $k \in \mathsf{N}$ is a natural constant. We assume that Φ contains no *redundant* constraints, for instance, if both $a - b \leq 5$ and $a - b \leq 6$ belong to Φ, then the second constraint is redundant, since it is less strict than the first one.

An *execution* of S is a function $\tau : A \times \{1, 2, ...\} \mapsto \mathsf{R}_+$, giving, for each event $a \in A$ and each cycle $p = 1, 2, ...$, the occurrence time of a at p, $\tau(a, p)$. Occurrence times are assumed to be global, so we require that τ satisfies the following *consistency* property: for all $a \in A$, $p = 1, 2, ...$, $\tau(a, p) \leq \tau(a, p + 1)$ (that is, a occurs at cycle p before it occurs at cycle $p{+}1$). We also require that τ meets the timing constraints of S, namely:

1. If $a - b \sim k$ is in Φ, then for each $p = 1, 2, ...$, $\tau(a, p) - \tau(b, p) \sim k$ [3].
2. If $a' - b \sim k$ is in Φ, then for each $p = 1, 2, ...$, $\tau(a, p + 1) - \tau(b, p) \sim k$.

S is called *feasible* if it has at least one such execution. Given a CCS S, we want to check whether S is feasible and if it is, we want to find an execution of S.

3 The solution to the feasibility problem

Given a CCS S, we solve the feasibility problem in three steps. First, we build from S a system of linear inequalities $I^{1,2}$, representing the constraints imposed only on the first two cycles of S. Second, using a constraint-propagation technique, we iteratively strengthen $I^{1,2}$, until it becomes stable or unsatisfiable. In the latter case, S is not feasible. Otherwise, the third step consists in computing a solution of (the stabilized) $I^{1,2}$ and extending it to an execution of S.

3.1 Informal description through an example

Consider the cyclic schedule S with three events a, b, c and constraints:

$$a \leq b \leq c, \quad a' - a \leq 4, \quad a' - b \geq 2, \quad b' - c \geq 6.$$

[3] Here, the operator $\sim$ is interpreted as the usual order relation on reals.

Let τ be a candidate execution S. For simplicity, we write a^p, b^p, c^p, instead of $\tau(a,p), \tau(b,p), \tau(c,p)$, for $p = 1,2$. Then, τ must satisfy at least the following system of constraints:

$$I^{1,2} \quad : \quad \begin{array}{ll} a^1 \le b^1 \le c^1, & a^2 - a^1 \le 4, \\ a^2 \le b^2 \le c^2, & b^1 - a^2 \le -2, \\ c^1 \le a^2, & c^1 - b^2 \le -6 \end{array}$$

The system of constraints $I^{1,2}$ "covers" the first two cycles, that is, it contains all constraints on occurrence-time differences of events in cycle 1, cycle 2, or between the two. Inconsistency of $I^{1,2}$ would imply that S cannot complete even two successive cycles. This is not the case here, and we can use constraints $a^2 - a^1 \le 4$ and $b^1 - a^2 \le -2$ to deduce:

$$b^1 - a^1 \le 2 \quad (*)$$

This is an extra piece of information on cycle 1, obtained using information on both cycles 1 and 2.

Now, we can "shift" the "constraint window" $I^{1,2}$ to cycles 2 and 3, and infer the constraint:

$$b^2 - a^2 \le 2 \quad (**)$$

which is identical to $(*)$, up to renaming. Notice that $(**)$ is not deduced explicitly (as it could have been, from the corresponding system of constraints $I^{2,3}$ on a^p, b^p, c^p, $p = 2,3$), but is obtained from $(*)$ simply by renaming. This is the idea of *constraint induction*: *transpose* to cycle 2 the amount of information obtained for cycle 1.

Consequently, $(**)$ can be "re-injected" into $I^{1,2}$ to strengthen it, permitting to deduce stricter constraints on cycle 1. Indeed, from $(**)$ and $c^1 - b^2 \le -6$ we deduce $c^1 - a^2 \le -4$, which, added to $a^2 - a^1 \le 4$, gives $c^1 - a^1 \le 0$. Using $a^1 \le b^1 \le c^1$ we finally obtain $a^1 = b^1 = c^1$. We can now iterate the process of constraint induction: $a^1 = b^1 = c^1$ is renamed to $a^2 = b^2 = c^2$ and the latter is re-injected into $I^{1,2}$, which now becomes unsatisfiable. This means that S is not feasible and, more precisely, that it cannot execute its loop more than 3 times (the number of iterations to end up with an unsatisfiable system).

Alternatively, $I^{1,2}$ could have stabilized to a satisfiable system of constraints, which would imply feasibility of S. Stabilization in a finite number of steps is guaranteed, as is shown in the next section.

3.2 Formal description and results

We first formalize the constraint systems of the type of $I^{1,2}$ shown in the previous section, and define the necessary operations. Then, we present the constraint induction technique.

3.2.1 Simple Linear Constraint Systems

Let $Y = \{y_1, ..., y_n\}$ be a set of non-negative real variables. A *simple linear constraint system* (SLCS) on Y is a conjunction of n^2 constraints:

$$J = \bigwedge_{1 \le i,j \le n} y_i - y_j \le k_{ij}$$

where $k_{ij} \in \mathsf{Z} \cup \{\infty\}$.

A *solution* of J is a function $V : Y \mapsto \mathsf{R}_+$ respecting all the constraints of J, that is, for all $1 \le i,j \le n$, $V(y_i) - V(y_j) \le k_{ij}$. J is called *satisfiable* if it admits a solution V.

Canonical form. An SLCS J on Y defines a (possibly empty) set of solutions on Y. In general, two (syntactically) different SLCSs J_1 and

J_2 may define the same set of solutions. In that case, we say that J_1 and J_2 are *equivalent*, denoted $J_1 \equiv J_2$. As an example, consider the SLCS J_1 and J_2 on $Y = \{y_1, y_2, y_3\}$, shown below:

$$J_1 : \begin{array}{c} 1 \leq y_2 - y_1 \leq 3 \\ 2 \leq y_3 - y_1 \leq 4 \\ -3 \leq y_2 - y_3 \leq 1 \end{array} \quad J_2 : \begin{array}{c} 1 \leq y_2 - y_1 \leq 3 \\ 2 \leq y_3 - y_1 \leq 4 \\ -3 \leq y_2 - y_3 \leq 2 \end{array}$$

J_1 and J_2 differ only on the upper bound of $y_2 - y_3$, which is looser in J_2. However, using $1 \leq y_2 - y_1$ and $y_3 - y_1 \leq 4$ we can deduce $y_2 - y_3 \leq 1$, which corresponds to the stricter bound of J_1. Now the two systems are element-wise syntactically identical, denoted $J_1 = J_2$.

In order to have an algorithmic way to decide whether two SLCSs are equivalent, or whether an SLCS is satisfiable, we must reduce the problem of checking semantic equivalence to that of checking syntactic identity. Thus, we define the *canonical form* of a system J_1 as the system of constraints $J_1^* \equiv J_1$ such that for every other system J_2, if $J_2 \equiv J_1$ then $J_2^* = J_1^*$. By convention, we let the canonical form of an unsatisfiable system of constraints be $\bot$. The canonical form of a finite-variable system exists, is unique, and can be effectively computed by strengthening its constraints to the "tightest" possible (see section 3.3). For the two systems above, $J_1^* = J_2^* = J_1$.

Projection. Let J be an SLCS on Y and let $Y' \subseteq Y$. The projection of J to Y', denoted $J/_{Y'}$, is the SLCS J' on Y', such that V is a solution of J iff $V/_{Y'}$ is a solution of J, where $V/_{Y'}$ is the function obtained from V by restricting the domain of V to Y'. (In terms of logic, $J/_{Y'}$ is equivalent to $\exists Y \setminus Y' \,.\, J$).

Intersection. Let J and J' be two SLCSs on Y and Y', respectively. The intersection $J \cap J'$ is defined as the *looser* SLCS on $Y \cup Y'$, which is stricter than both J and J'. More precisely, V is a solution of $J \cap J'$ iff $V/_Y$ is a solution of J and $V/_{Y'}$ is a solution of J'.

Renaming. Let J be an SLCS on Y and $Z = \{z_1, ..., z_n\}$ be a set of variables such that $Z \cap Y = \emptyset$. $J[y_1 := z_1, ..., y_n := z_n]$ (in short, $J[Y := Z]$) is the SLCS J' on Z, such that V is a solution of J iff $V[Y := Z]$ is a solution of J', where $V[Y := Z] : Z \mapsto \mathsf{R}_+$ is defined by $V[Y := Z](z_i) = V(y_i)$.

3.2.2 The Constraint Induction technique

Consider a cyclic schedule $S = (A, \Phi)$, with $A = \{a_1, ..., a_n\}$ and $\Phi = \{\phi_1, \cdots, \phi_m\}$. Let $X^1 = \{x_1, ..., x_n\}$ and $X^2 = \{x_{n+1}, ..., x_{2 \cdot n}\}$ be two sets of non-negative real variables. We first build the SLCS $I^{1,2} = \bigwedge_{1 \leq i,j \leq 2 \cdot n} x_i - x_j \leq k_{ij}$ on $X^1 \cup X^2$, where k_{ij} is derived from Φ according to the rules given below:

1. If $a_i - a_j \leq k$ is in Φ then $k_{ij} = k$.
2. If $a'_i - a_j \leq k$ is in Φ then $k_{n+i,j} = k$.
3. If $a_i - a_j \geq k$ is in Φ then $k_{ji} = -k$.
4. If $a'_i - a_j \geq k$ is in Φ then $k_{j,n+i} = -k$.
5. Otherwise, $k_{ij} = \infty$.

Before stating the main result of this paper, we need a definition. A solution V of a SLCS J on $X^1 \cup X^2$ is called *periodic* if for all $1 \leq i, j \leq n$, $V(x_{n+i}) - V(x_{n+j}) = V(x_i) - V(x_j)$.

Theorem 3.1 (Constraint induction)

1. *Let $\hat{J}$ be the following greatest fix-point:*

$$\hat{J} \;=\; \nu J \,.\, \left((I^{1,2})^* \cap (J/_{X^1})[X^1 := X^2]\right)^*$$

The SLCS $\hat{J}$ exists and can be computed as the limit of the series:

$$\begin{aligned} J^0 &= (I^{1,2})^* \\ J^{i+1} &= \left(J^i \cap (J^i/_{X^1})[X^1 := X^2]\right)^* \end{aligned}$$

which is attained in a finite number of steps, that is, there exists a natural constant q such that $J^{q+1} = J^q = \hat{J}$.

2. *S is feasible iff $\hat{J}$ is satisfiable. If $\hat{J}$ is satisfiable, then it admits a periodic solution $\hat{V}$, which can be effectively extracted from $\hat{J}$ in time $O(n^2)$.*

3. *If $\hat{J}$ has a periodic solution $\hat{V}$ and $\tau : A \times \mathsf{N}_+ \mapsto \mathsf{R}_+$ is defined, for all $p = 1, 2, \ldots, 1 \leq i \leq n$, as*

$$\tau(a_i, p) = \hat{V}(x_i) + (p-1) \cdot (\hat{V}(x_{n+i}) - \hat{V}(x_i))$$

then τ is an execution of S.

The proof of the theorem is somewhat involved and can be found in the appendix.

3.3 Algorithmics and complexity

The overall algorithm to solve a CCS $S = (A, \Phi)$ consists in three steps:

1. Build the system of constraints $I^{1,2}$ from Φ.
2. Compute $\hat{J}$ using the fix-point iteration of part 1 of theorem 3.1.
3. If $\hat{J} = \bot$ then S is not feasible. Otherwise, compute $\hat{V}$ and extend it to an execution of S using part 2 of theorem 3.1.

The first step is straightforward, consisting in a simple syntactic transformation. The second step is performed by the following algorithm:

$J := (I^{1,2})^*$;
change := *true* ;
`while` ($J \neq \bot$ `and` *change*) `do`
  $J' := \left(J \cap (J/_{X^1})[X^1 := X^2]\right)^*$;
  `if` $J' = J$ `then`
    change := *false* ;
  `else`
    $J := J'$;

In order to implement the above algorithm we need a data structure to represent SLCSs and procedures to compute the canonical form, intersection, projection and substitution. For an SLCS on n variables, we shall use a *difference bound matrix* (DBM) [Dil89], which is a square matrix M of dimension n. The element M_{ij} of M is a constant $k \in \mathsf{Z} \cup \{\infty\}$, encoding the constraint $x_i - x_j \leq k$.

If M is a matrix representing J, the matrix M^* for J^* is obtained by applying to M a shortest-path algorithm, such as Floyd-Warshall's all-pairs-shortest-paths [PS82], where M is viewed as a graph with nodes labeled from 1 to n, and weighted edges $i \xrightarrow{M_{ij}} j$. If a negative-cost cycle is found during the algorithm then J is unsatisfiable, in other words, $J^* = \bot$. If $J = J^* \neq \bot$ and the set of variable of J is Y, then the matrix for $J/_{Y'}$, for $Y' \subseteq Y$, is obtained by removing the rows and columns of M corresponding to variables in $Y \setminus Y'$. Variable substitution is also straightforward to implement.

If M and M' are matrices for J and J', respectively, the intersection $J \cap J'$ is represented by a matrix M'' whose element M''_{ij} is defined to be the *minimum* of M_{ij} and M'_{ij} (any integer is taken to be strictly smaller than ∞). Finally, the identity test $J = J'$ is performed by simply testing whether M_{ij} is identical to M'_{ij} for all i, j.

The third step of the feasibility algorithm involves computing $\hat{V}$. This is done following the constructive proof of theorem 3.1 (part 2), given in the appendix.

Complexity. Step 1 of the feasibility algorithm is a simple syntactic transformation which can be done in $O(n + m)$ time, where $n = |A|$ is the number of events and $m = |\Phi|$ is the number of timing constraints of the cyclic schedule. Step 3 can be performed in $O(n^2)$ time (see the appendix).

Regarding the complexity of the matrix operations used in step 2, the most costly one is the shortest-path algorithm, with worst-case complexity $O(N^3)$, where N is the matrix dimension. Since we use matrices of dimension $N = |X^1 \cup X^2| = 2 \cdot n$, the body of the loop of the constraint-induction in step 2 has worst-case complexity $O(n^3)$.

The number of iterations of the loop until stabilization, q, generally depends on the constants appearing in the timing constraints of Φ. A worst-case upper bound for q is $2 \cdot \Sigma\{k \mid a - b \sim k \in \Phi\}$, that is, twice the sum of all constants appearing in Φ. In practice, q rarely grew beyond 3 and remained lower than 7 in all the experiments we have performed (section 4).[4]

Concerning space complexity, the amount of memory required by the algorithm is quite small: an array of size $4 \cdot (2 \cdot n)^2$ bytes is sufficient.

[4] This does not mean that the worst-case bound is not achievable: for example, the CCS with three events a, b, c and constraints $a \leq b \leq c$, $a' - a \leq k$, $a' - b \geq 1$, $b' - c \geq k + 1$, is found infeasible after k iterations, for any k.

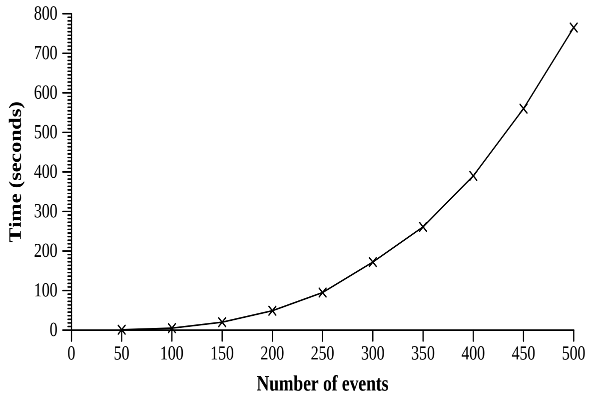

Figure 2: Performance results for random tests.

4 Experiments

We have implemented the constraint induction technique to a prototype called CINDY, developed as part of the real-time tool suite of VERIMAG. CINDY takes as input a CCS and checks its feasibility. We have tested the performance of CINDY on a PC-i686 at 133MHz with 512 Mbytes of main memory, the goal being to get a feeling on the practicality of the technique, rather than solving a real problem.

We checked feasibility of a series of CCSs with n events, n ranging from 10 to 500, and n^2 randomly generated timing constraints. The induction loop took no more than 7 iterations to stabilize, and the CCSs were found sometimes feasible and sometimes not. Time performance of the algorithm is shown in figure 2. It can be seen that CINDY can solve CCSs of 500 events (or more) in less than 15 minutes.

5 Conclusion and perspectives

We have presented a novel, to our knowledge, model for specifying systems of timing constraints of repetitive nature. We have also designed an efficient algorithm to check feasibility and compute solutions for this kind of constraints. A prototype implementation demonstrated good time performance of the algorithm for systems of large size, generated randomly. We hope that our technique will be useful to designers of real-time systems which operate in cycles and have a large number of events and timing constraints, which make proving feasibility and building scheduling scenarios a non-trivial problem.

Regarding perspectives, first, a number of algorithmic optimizations could be applied to improve performance. In particular, the shortest-path algorithm currently used to compute the canonical form of a system of constraints (with complexity $O(n^3)$ for n variables) can be replaced by more efficient *incremental* algorithms such as those presented in [Chl95].

Second, it would be interesting to consider ways to classify scenarios and methods to compute the best scenario. For the cases of shortest- (or longest-) repetition-delay best, the algorithm for computing $\hat{V}$ can be extended to compute the best scenario at no extra cost.

Third, possible extensions of the model include allowing constraints which refer to more than one cycle ahead. The constraint-induction technique can be easily extended in this case, by using a system of constraints on variables of three or more cycles, as necessary. Complexity is affected only by a factor of m^3, where m is the number of cycles we are allowed to look ahead.

Finally, it is worth mentioning that constraint induction is a general constraint-propagation technique which could be applied to more general classes of constraints (for example, general linear inequalities), provided that one can prove termination of the fix-point iteration (i.e., convergence of the strengthening steps). Apart from real-time scheduling, this perspective is also interesting for other domains, such as operations research.

References

[Chl95] N. Chleq. Efficient algorithms for networks of quantitative temporal constraints. In *Constraint'95*, 1995.

[Dil89] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Workshop on Computer Aided Verification, CAV89*, Grenoble, June 1989. Lecture Notes in Computer Science, Springer-Verlag.

[PS82] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

A Proof of theorem 3.1

For technical convenience, we rename variables in $\hat{J}$ as follows: $[x_1 := y_0, ..., x_{2\cdot n} := y_{2\cdot n-1}]$. Then, $\hat{J}$ can be represented as a system of inequalities $y_i - y_j \leq k_{i,j}$, for $0 \leq i, j < 2 \cdot n$.

Part 1

We assume that all constraints are in normalized form $y - y' \leq c$, where $c \in \mathsf{Z} \cup \{\infty\}$. The proof is by contradiction, assuming that there is a constraint $y - y' \leq c$ which is strengthened infinitely often, that is, c decreases without lower bound. Since the operation which "destabilizes" the set of constraints is variable substitution (after each step the set of constraints is fixed), we can assume that $y - y' \leq c$ is a constraint of the first iteration, that is, $y, y' \in \{y_0, ..., y_{m-1}$.

In order for c to decrease, there should be a "path" of constraints $y - w_1 \leq c_0$, $w_1 - w_2 \leq c_1$, ..., $w_k - y' c_k$, where $w_1, ..., w_k \in Y$, such that $c_0 + \cdots + c_k < c$ (so that we get $y - y' = y - w_1 + w_1 - w_2 + \cdots + w_k - y' \leq c_0 + \cdots + c_k$). Now, there should be at least one constraint of the second iteration, say, $w_j - w_{j+1} \leq c_j$, which is also strengthened infinitely often. This is because after each step the set of constraints is fixed and there is a finite number of constraint paths like the one above.

Since w_j, w_{j+1} are variables of the second iteration and y' is a variable of the first iteration, there are two implicit constraints $y' - w_j \leq 0$ and $y' - w_{j+1} \leq 0$. From the latter we conclude that $c_{j+1} + \cdots + c_k \geq 0$, otherwise we would have $0 = y' - w_{j+1} + w_{j+1} - y' = y' - w_{j+1} + w_{j+1} - w_{j+2} + \cdots + w_k - y' \leq 0 + c_{j+1} + \cdots + c_k < 0$, and the set of constraints would be inconsistent. Thus, there is a constant c' such that $0 \leq c' \leq c_{j+1} + \cdots + c_k$ and $w_{j+1} - y' \leq c'$ from some point on in the fix-point computation. (Notice that c' cannot increase, since at each iteration the constraints are strengthened.)

Now, since $w_j - w_{j+1}$ is strengthened infinitely often, there exists some point in which $w_j - w_{j+1} \leq -(c'+1)$. At this point, we have $0 = y' - w_j + w_j - w_{j+1} + w_{j+1} - y \leq 0 - (c'+1) + c' = -1$, which implies that the set of constraints is inconsistent, contradicting our initial assumption.

Part 2

For part 2 we have to show how to construct the solution of $\hat{V}$ from $\hat{J}$.

For simplicity, we write y_i instead of $\hat{V}(y_i)$.

From the definition of $\hat{J}$, it follows that $\hat{J}/_{X^2} \subseteq (\hat{J}/_{X^1})[X^1 := X^2]$, that is, every solution of $\hat{J}/_{X^2}$ is a solution of $(\hat{J}/_{X^1})[X^1 := X^2]$. Thus, we have as our hypothesis:

$$\forall 0 \leq i, j < n, \ . \ k_{i+n,j+n} \leq k_{i,j}$$

The second hypothesis we shall be using is that $\hat{V}$ is in canonical form, which implies that:

$$\forall 0 \leq i, j, l < 2 \cdot n \ . \ k_{i,j} \leq k_{i,l} + k_{l,j}$$

The construction of $\hat{V}$ is as follows. First, we set $y_0 = 0$ [5] and choose y_n such that:

$$\forall 0 \leq i < n \ . \ -k_{i,n+i} \leq y_n \leq k_{n+i,i}$$

To see that such a value can always be chosen, let the minimum of $k_{n+i,i}$ be at i_0 and the maximum of $-k_{i,n+i}$ be at j_0. Then, we must show that: $-k_{j_0,n+j_0} \leq k_{n+i_0,i_0}$. From the hypotheses, we have:

$$\begin{array}{ll} & k_{n+j_0,n+i_0} \leq k_{j_0,i_0} \\ \Rightarrow & k_{n+j_0,n+i_0} \leq k_{j_0,n+j_0} + k_{n+j_0,n+i_0} + k_{n+i_0,i_0} \\ \Rightarrow & 0 \leq k_{j_0,n+j_0} + k_{n+i_0,i_0} \end{array}$$

which directly implies the result.

Then, for each $i = 1, ..., n-1$, we set $y_{n+i} = v$ and $y_i = v - y_n$, where v is chosen such that, $\forall 0 \leq j < i$.

$$y_n + y_{n+j} - k_{n+j,i} \leq v \leq y_n + y_{n+j} + k_{i,n+j}, \quad \text{and} \quad y_{n+j} - k_{n+j,n+i} \leq v \leq y_{n+j} + k_{n+i,n+j}$$

As previously, we have to ensure that such a v always exists. We show that this is the case by induction on i. We only consider two out of the four possible cases for the minimum and maximum of right-hand and left-hand values in

[5] Such a choice might result in some y_i values which are negative. This is no problem, since all values can be "shifted" at the end of the construction to the positive reals, by adding a certain constant without affecting the solution.

the constraints above, respectively. The rest of the cases are symmetrical. First, suppose that the minimum of the right-hand values is $y_{n+j_1} + k_{n+i,n+j_1}$ and that the maximum of the left-hand values is $y_{n+j_2} - k_{n+j_2,n+i}$. Then, we must show that: $y_{n+j_2} - k_{n+j_2,n+i} \leq y_{n+j_1} + k_{n+i,n+j_1}$. We have:

$$\begin{aligned} y_{n+j_2} - y_{n+j_1} &\leq k_{n+j_2,n+j_1} \\ &\quad \text{from the induction hypothesis} \\ &\leq k_{n+j_2,n+i} + k_{n+i,n+j_1} \\ &\quad \text{from the canonical form hypothesis} \end{aligned}$$

which directly implies the result.

As a second case, suppose that the minimum of the right-hand values is $y_n + y_{n+j_1} - k_{n+j_1,i}$ and that the maximum of the left-hand values is $y_{n+j_2} - k_{n+j_2,n+i}$. Then, we must show that: $y_{n+j_2} - k_{n+j_2,n+i} \leq y_n + y_{n+j_1} - k_{n+j_1,i}$. We have:

$$\begin{aligned} y_{n+j_2} - y_{n+j_1} &\leq k_{n+j_2,n+j_1} \\ &\quad \text{from the induction hypothesis} \\ &\leq k_{n+j_2,n+i} + k_{n+i,i} + k_{i,n+j_1} \\ &\quad \text{from canonical form} \\ &\leq k_{n+j_2,n+i} + y_n + k_{i,n+j_1} \\ &\quad \text{from the construction of } y_n \end{aligned}$$

which directly implies the result.

By construction, $y_i - y_j = y_{n+i} - y_{n+j}$, for all $0 \leq i, j < n$. Also, it is easy to see that the construction respects the constraints of $\hat{J}$, therefore, we have a solution of $\hat{J}$ meeting the conditions of theorem 3.1.

Regarding the time complexity of the above construction, observe that, in order to chose a value for y_{n+i}, $i = 0, ..., n-1$, we need the minimum and maximum of at most $2 \cdot n$ values, which can be found in $O(n)$. The value for y_i is then determined automatically. Therefore, the whole solution can be assembled in $O(n^2)$ time.

Part 3

For part 3 we have to prove that τ is an execution of S.

Observe that $\tau(a_i, 1) = \hat{V}(x_i)$ and $\tau(a_i, 2) = \hat{V}(x_{n+i})$, for $i = 1, ..., n$. Then, using the definition of $I^{1,2}$ and the fact that $\hat{V}$ is a solution of the latter, it is easy to see that the differences $\tau(a_i, 1) - \tau(a_j, 1)$ and $\tau(a_i, 2) - \tau(a_j, 1)$ satisfy the timing constraints of Φ, for all $1 \leq i, j \leq n$. For $p > 1$, we have:

$$\begin{aligned} &\tau(a_i, p) - \tau(a_j, p) \\ &\quad = \left(\hat{V}(x_i) + (p-1) \cdot (\hat{V}(x_{n+i}) - \hat{V}(x_i))\right) \\ &\qquad - \left(\hat{V}(x_j) + (p-1) \cdot (\hat{V}(x_{n+j}) - \hat{V}(x_j))\right) \\ &\quad = \hat{V}(x_i) - \hat{V}(x_j) \\ &\qquad + (p-1) \cdot \\ &\qquad\quad \left((\hat{V}(x_{n+i}) - \hat{V}(x_{n+j})) - (\hat{V}(x_i) - \hat{V}(x_j))\right) \\ &\quad = \hat{V}(x_i) - \hat{V}(x_j) \end{aligned}$$

since $\hat{V}(x_{n+i}) - \hat{V}(x_{n+j}) = \hat{V}(x_i) - \hat{V}(x_j)$. Similarly, we get $\tau(a_i, p+1) - \tau(a_j, p) = \hat{V}(x_{n+i}) - \hat{V}(x_j)$. Therefore, τ satisfies all the constraints of Φ, which are identical to those of $I^{1,2}$, up to renaming. Since, by definition, we also have $\tau(a_i, p) \leq \tau(a_i, p+1)$, τ is an execution of S.