4

VERIFICATION WITH RELATIVE TIMING

When you are courting a nice girl an hour seems like a second. When you sit on a red-hot cinder a second seems like an hour. That's relativity.

—Albert Einstein - Quoted in the News Chronicle, 1949

Summary

This chapter presents the theoretical aspects of our relative timing-based verification approach for timed systems. Most of the material was already published in [PCKP00].

First, two small examples conduct a review of the notion of relative timing and an outline of the overall verification strategy.

Next, the different theoretical aspects of the verification approach are introduced. A trace semantics is defined to unify the reasoning with the different computational models used by the verification. The main notion presented is that of enabling compatibility, which makes possible that the timing analysis over the set of events in a trace, can be also applied over a set of traces which share the same enabling orderings. Event structures are then introduced as a model that represents succinctly a set of enabling-compatible traces, and for which efficient timing analysis algorithms exist. The enabling-compatible product of transition systems is then presented as a way to refine the untimed state space of a system with a set of relative timing constraints.

Finally, all these ideas are combined together in a fully automated iterative verification methodology. Relevant aspects such as the correctness and the convergence of the approach are discussed.

4.1 Introduction

The verification of concurrent systems typically suffers from the well known state explosion problem. In systems with a finite number of states, the problem is often alleviated by using symbolic techniques to represent the reachable states. This is also combined with partial order techniques or abstractions that reduce the complexity of the models. However, when time is an essential dimension in the verification problem, complexity is drastically affected. Since the correctness of the system depends on the actual values of event delays and not only on its functional behavior, the verification becomes unmanageable even for moderate-size systems. More precisely, computing the reachability space of a timed system is proved to be a PSPACE-complete problem [AD94], and demonstrated to be highly complex in several practical contexts. Although several techniques have been devised to alleviate such complexity (see Chapter 3), the size of the untimed state space is still the major bottleneck for the analysis of highly concurrent systems.

This chapter describes a novel approach that extends the applicability of the conventional methods based on symbolic reachability analysis, to the verification of safety properties in timed systems. The approach is based on two fundamental facts:

- The observation that the set of runs of a transition system can be covered by a set of event structures [NPW81]. This reduces the verification problem to that of: the timing analysis over small sets of events from which timing constraints that prove the correctness or incorrectness of a system can be derived; and the incorporation of such constraints into the system along an incremental refinement process.

- The use of *relative timing* [SGR99] allows to represent the timed domain of a system in an efficient way. When considering precise delay bounds in timed systems, the complexity blow-up often makes the analysis an intractable problem, even for small systems. Instead, relative timing considers the *effect* of delays in a system in terms of relative ordering of events (*e.g.* a happens before b).

The verification approach can be briefly summarized as follows. Rather than calculating the exact timed state space, an *off-line* timing analysis is performed on a set of event structures that covers the runs leading to system failures. Several timing analysis algorithms have been provided for acyclic graphs, including exact and approximated methods. In our case, the timing analysis is efficiently performed by using McMillan and Dill's algorithm [MD92], which is the precursor of most latter algorithms. The resulting timing constraints are incorporated to the system in the form of relative timing information along a series of iterative refinements of the original untimed state space. If some of the runs leading to failure situations cannot be proved to be timing-inconsistent, then the system is incorrect and the failure run is a counterexample.

Due to the incremental incorporation of timing information along the verification, our approach works with over-approximations of the actual timed state space of the system. Being the completely untimed state space used as starting point the roughest approximation possible. This fact allows the efficient verification of safety properties but makes impossible the verification of liveness properties, for example. For safety properties, it is enough to prove that no "undesired" situations (states) are reachable by the system. If "undesired" states do not appear in the over-approximations, they will neither appear in the exact timed state space, but not vice versa. Therefore, the verification can produce "false-negatives" but never "false-positives", *i.e.* it is conservative for safety properties. On the contrary, for liveness properties it must be proved that some "desired" situation is actually reachable. For that kind of proof, the exact timed state space (or an under-approximation for conservativeness) must be computed.

The use of event structures for timing analysis was also proposed in [KBS02]. However, no algorithm was presented that can handle a general class of transition systems for verification. Moreover, the approach presented here, not only verifies the correctness of the system with respect to a set of given safety properties, but also provides as back-annotation a set of timing constraints sufficient to prove such correctness. This information is crucial in frameworks in which synthesis and verification are iteratively invoked to design systems that must meet functional and non-functional constraints.

We want to remark that the application of the method for the verification of untimed systems does not involve any additional overhead with respect to the conventional symbolic methods (*e.g.* [BCM+92]).

4.1.1 Relative Timing

So far we have talked about the idea of relative timing but no illustrative example has been provided that can help to understand some of its benefits, specially in areas other than the verification of timed systems. In this section we reproduce partially an example from [CKK+02] where relative timing is used to improve the synthesis of asynchronous control circuits. The synthesis process takes relative timing information into account thus allowing the generation of smaller and faster circuits.

Example 4.1 *Consider the asynchronous circuit in Figure 4.1 (a). The delays of the gates are represented by intervals of the form* $[d, D]$*, which indicate that the output of the gate driving a given signal* x *will change* $\delta(\mathsf{x})$ *time units after the gate became enabled, with* $d \leq \delta(\mathsf{x}) \leq D$*. That is, the firing time is bounded by the given delay interval.*

After the occurrence of a rising transition of signal y*, the behavior represented by the STG of Figure 4.1 (b) is enabled to happen. The rising transition of signal* b *appears to be concurrent with the rising transition of signals* c *and* d*. The corresponding underlying TS is depicted in Figure 4.1 (c).*

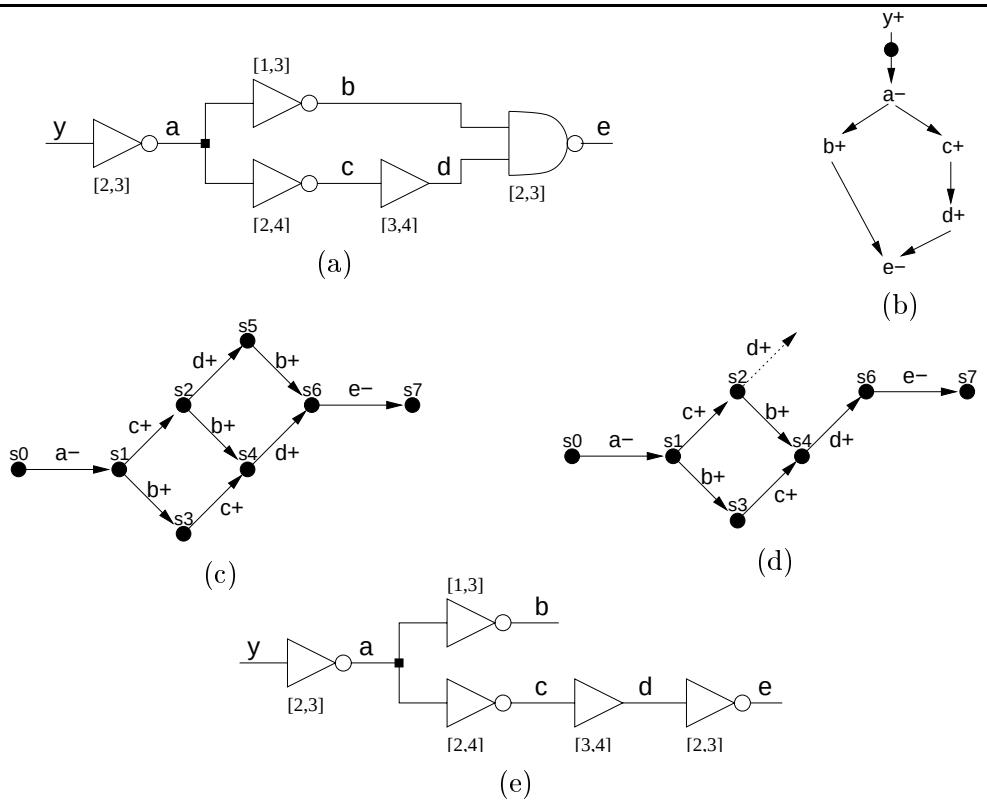


Figure 4.1 Relative timing in the synthesis of circuits: (a) timed circuit, (b) portion of the STG and (c) corresponding TS for the untimed behavior, (d) corresponding LzTS and (e) optimized circuit.

If the actual delays of the gates driving these signals are considered, it is easy to realize that b+ *will always happen before* d+*. Clearly, the earliest time for* d+ *to occur is 5 time units after* a−*, whereas the latest time for* b+ *to occur is only 3 time units after* a−*. This observation can be translated into the fact that state* s_5 *of the untimed TS will never be reached (see the resulting LzTS in Figure 4.1 (d)).*

Provided that b+ *will always happen before* d+*, the causality relation* b+ $\longrightarrow$ e− *is always guaranteed by the actual delays and the causality relation* d+ $\longrightarrow$ e−*. Thus, a potential optimization of the circuit may consider the relative timing constraint between* b+ *and* d+*, and ignore the explicit causality relation* b+ $\longrightarrow$ e−*, which leads to the optimized circuit of Figure 4.1 (e).* ■ 4.1

Along the process described in the example, neither the exact times at which each event occurs nor the exact times at which the states are reached need to be determined. Instead the reasoning is done in terms of "*which event occurs before each other*". This type of reasoning is particularly useful in the early stages of the design flow, when the exact

timed behavior of a system is difficult to determine and precise delay constraints are hard to satisfy. Conversely, it is much simpler to deal with constraints that just state which event must be faster than other, without taking care of the exact delay slack between them. Moreover, it is much easier to keep these type of constraints satisfiable along the successive design steps.

Using similar ideas, Intel's Strategic CAD Lab has recently designed an asynchronous instruction length decoder for the x86 instruction set [RSG+99]. The circuit exhibits a promising increase in performance with respect to its synchronous counterpart, thanks to the optimizations achieved using the relative timing information. The techniques pioneered by this design have been evolved and formalized using the LzTS model [CKK+98] and automated in the logic synthesis tool PETRIFY [CKK+97].

Finally remark, that although this section has referred to asynchronous circuits, they are just an example of application. The relative timing paradigm is applicable to the design, synthesis and verification of timed systems in general.

4.2 Overview

This section provides an overview of the verification approach with relative timing presented in this chapter. For that purpose, an simple illustrative example is developed.

This work develops a formal approach to verify that a system with certain timing constraints satisfies a given safety property P. The system is modeled by means of a timed transition system, A, composed by an underlying transition system, A^-, and two functions, δ^l and δ^u, which associate minimal and maximal delays, respectively, to each event of the system. A given sequence of events of a TTS is said to be timing-consistent if it is possible to assign increasing time values to all the events such that their firing times are within the allowed bounds. The modeling formalism of timed transition systems was introduced in Section 2.3.

The verification problem is posed in terms of the following language inclusion test: $\mathcal{L}(A) \subseteq \mathcal{L}(P)$ [Gup92], where $\mathcal{L}(A)$ corresponds to the set of all possible behaviors of A, and $\mathcal{L}(P)$ is the set of all possible behaviors satisfying property P. The approach consists in building successive conservative approximations of $\mathcal{L}(A)$ starting from $\mathcal{L}(A^-)$, by adding relative timing constraints [SGR99] in an iterative manner. We start from the TS $A_0 = A^-$, *i.e.* the original system without timing constraints, and try to prove the inclusion $\mathcal{L}(A_0) \subseteq \mathcal{L}(P)$. If the inclusion holds, then $\mathcal{L}(A) \subseteq \mathcal{L}(A_0) \subseteq \mathcal{L}(P)$, which indicates that A satisfies P without any timing assumption. The verification succeeds.

If P is not satisfied in some state, a run ρ that leads to a failure is generated. If the run is timing-consistent, then the system is incorrect, *i.e.* violates the required property. However, if the run is not timing-consistent, it can be used to refine the untimed state space and remove other timing-inconsistent runs leading to failure states. To do this, a

suffix ρ' of the run ρ is taken and an event structure that covers ρ' is built. Timing analysis on the event structure is performed by using the polynomial algorithm for acyclic marked graphs in [MD92]. A set of relative timing constraints are derived that prove the timing-inconsistency of ρ'.

The state space of the (timed) event structure is composed with the untimed abstraction of the system A_0, in such a way that at least the failure run ρ is removed and no timing-consistent run is removed. A series of successive approximations A_i of A are constructed iteratively, with containment $\mathcal{L}(A) \subseteq \mathcal{L}(A_i)$ and monotonic convergence, $\mathcal{L}(A_{i+1}) \subseteq \mathcal{L}(A_i)$. At every step $\mathcal{L}(A_i) \subseteq \mathcal{L}(P)$ is checked. Verification stops successfully if the inclusion holds, or fails if a counterexample run is found. For a discussion on the convergence of the method refer to Section 4.6.6.

Iterative approaches for the verification of real-time systems have been also presented in [AK95, BSV95]. The major novelty of the approach presented in this thesis is the use of event structures to perform efficient off-line timing analysis, and to incorporate the resulting timing information in the form of relative timing constraints.

EXAMPLE 4.2 *Figure 4.2 depicts the TTS modeling a simple timed system. Figures 4.2 (a) and (b) show respectively, the underlying (untimed) TS and the delay intervals of events* a, b, c *and* g. *The delay interval for the rest of events is assumed to be unbounded,* i.e. $[0, \infty)$. *Figure 4.2 (e) depicts the state space of the system, when the delays are taken into account the shadowed states are not reachable. A crucial observation is that all runs in the TS of Figure 4.2 (a) that start and end at state* s_0 *can be covered by the two event structures depicted in Figures 4.2 (c) and (d): black states are covered by the event structure (c), white states are covered by the event structure (d) and grey states are covered by both event structures. Thanks to this fact the later verification process can be carried out with just a couple of refinements.*

Assume that the property to be verified indicates that event g *must always precede event* d *in any possible run after having visited state* s_0. *The property holds in the timed state space since no state where* d *can fire before* g *is reachable. Conversely, the property does not hold in the untimed state space, for example in state* s_{10} *where* d *can fire before* g.

The analysis starts by generating a run that leads to the failure situation, for example a run from s_0 *to* s_{10} *followed by the firing of* d *before* g *can be generated (Figure 4.3 (a)). Next, an event structure that captures the causality relations of the events in the run is derived (Figure 4.3 (b)). Notice that, in the event structure,* c *is only triggered by* a *but not triggered by* b, *as one may expect by looking at the transition system. This is due to the fact that the event structure only contains those causality relations derived from the run. In the failure run under analysis,* c *is not enabled in* s_1 *and is enabled after having fired* a *from* s_1. *Thus* a *triggers* c, *while* b *is concurrent to it.*

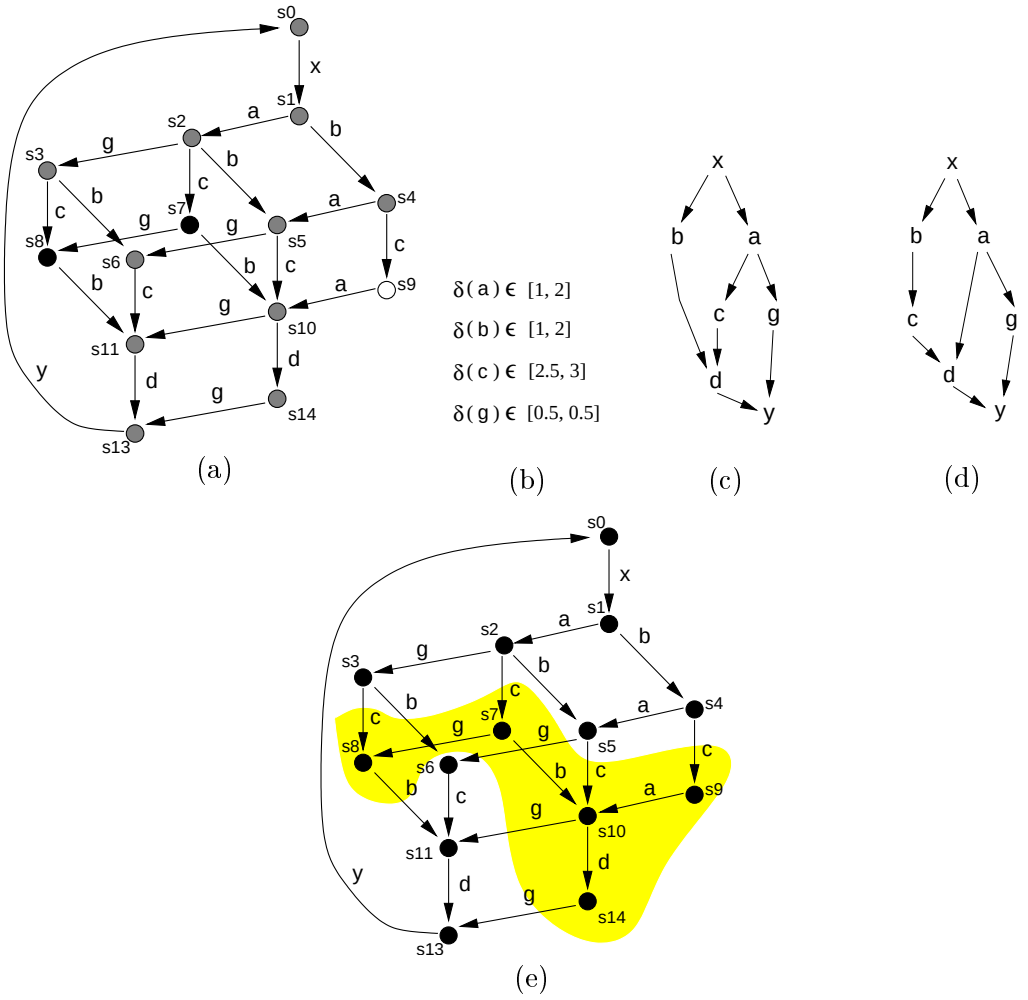


Figure 4.2 Example of verification with relative timing: (a,b) TTS and delay intervals. (c,d) Event structures covering the runs starting from s_0. (e) Timed state space (shaded states are unreachable).

By timing analysis over the event structure, we find that b *and* g *always precede* c. *These timing relations are shown as the dotted arcs incorporated to the event structure in Figure 4.3 (c). Such timing analysis is only valid for the causal relations expressed in the event structure, but it is not valid, for example, in the case when* b *triggers* c. *Figure 4.3 (d) depicts the state space of the timed event structure, where the shadowed states are not reachable due to the timing relations. Namely, event* c *is prevented to fire in some states, where its firing would be inconsistent with the timing analysis.*

Finally, all this information is incorporated into the system (Figure 4.3 (e)) by composing the original system and the event structure. An event structure being derived from a

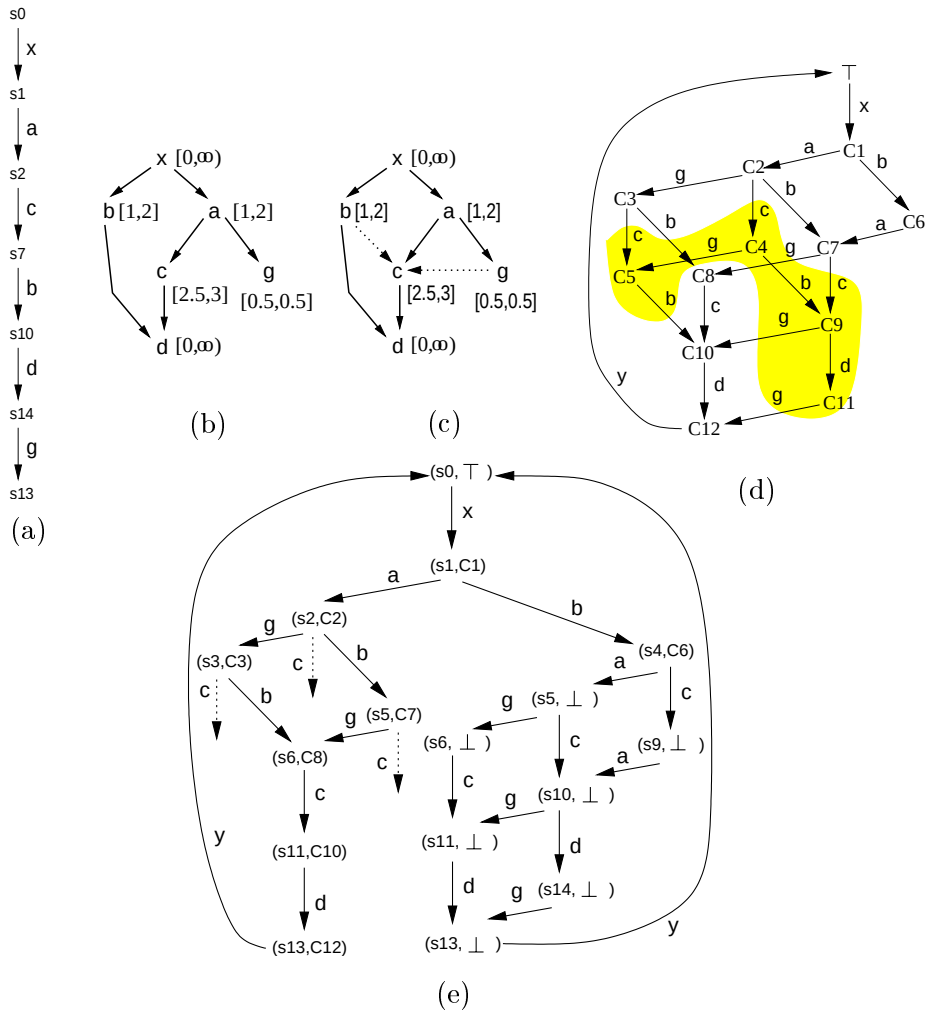


Figure 4.3 Example of verification with relative timing (first iteration): (a) A failure run and its corresponding event structure (b). The event structure annotated with timing arcs (c). (d) State space of the event structure (shaded states are unreachable). (e) LzTS obtained after composition.

particular run gives only partial behaviors of the original system. When the behaviors of the system and the event structure mismatch, the special symbol $\perp$ *is used. Some states in the composed system are split into two instances depending on whether they are reached by runs matching (*enabling compatible*) the event structure or not (see states* s_5, s_6, s_{11} *and* s_{13}*). Figure 4.3 (e) shows the resulting system. Notice that the set of runs is smaller than that of the original system, but larger than that of the actual state space when the delays are considered (Figure 4.2 (e)), and that only timing-inconsistent runs have been removed.*

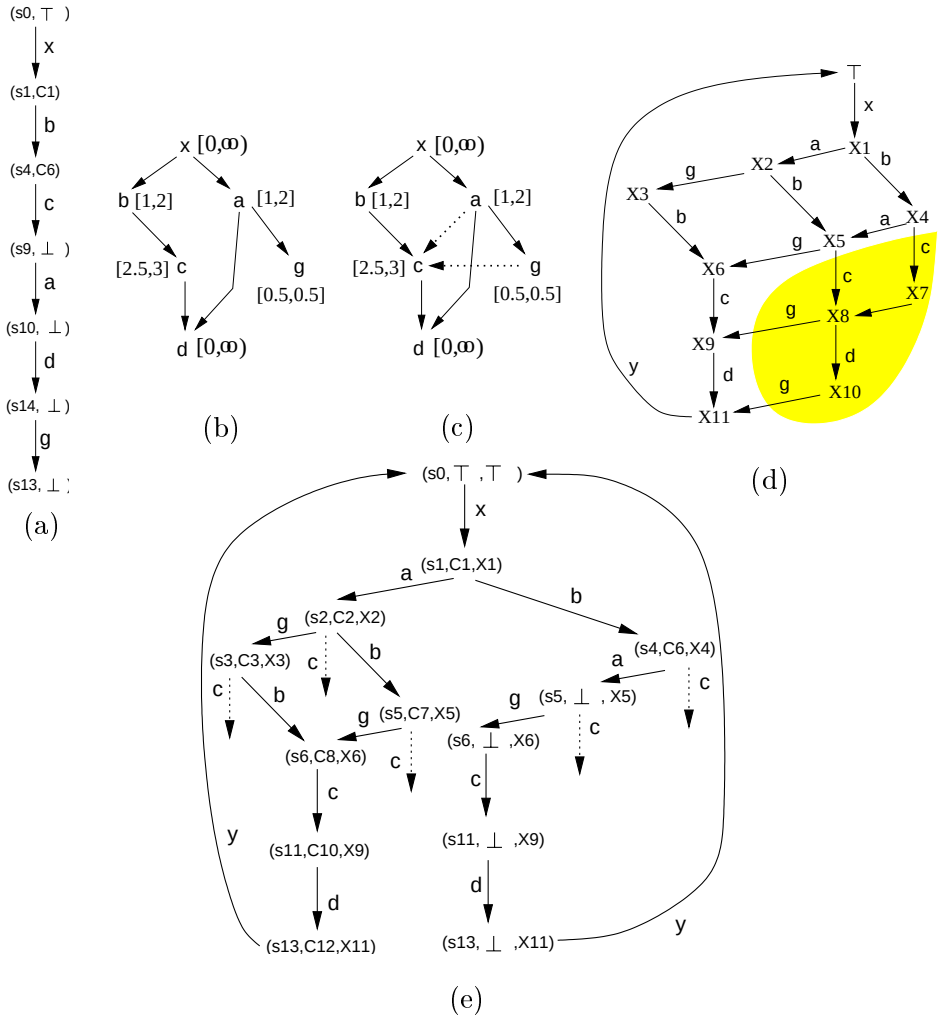


Figure 4.4 Example of verification with relative timing (second iteration): (a) A failure run and its corresponding event structure (b). The event structure annotated with timing arcs (c). (d) State space of the event structure (shaded states are unreachable). (d) LzTS obtained after composition.

This completes the first refinement of the untimed state space taken as starting point. This step has removed some of the failure runs but not all of them. For example, if the new state $(s_{10}, \bot)$ *is reached,* d *can fire before* g *and this contradicts the property under verification. Figure 4.4 summarizes one more refinement. In the resulting system all the failure runs have been removed, which proves that the system satisfies the property. Although it is not generally true, in this case the final state space contains exactly the same runs than the actual state space shown in Figure 4.2 (e).*

■ 4.2

Several objects and notions have been mentioned along the previous example, such as event structures, enabling compatibility, etc. These and other notions, as well as the theoretical aspects of the verification with relative timing are presented in detail in the following sections.

4.3 Trace semantics

As we have discussed in the previous section, the verification problem is posed in terms of the language generated by the system under verification and the language of all behaviors satisfying a given property. The verification process involves lazy transition systems and event structures (see Section 4.4) as major models. The process consists in an iterative incremental refinement of the system under verification with the timing information derived from the event structures.

A common semantics that unifies the models involved in the verification process can be defined in terms of *traces*. Based on traces, we will derive several notions that formalize our refinement approach for verification. This flow, illustrated in Figure 4.5, covers the contents of Sections 4.3 and 4.4.

4.3.1 Traces and languages

We extend the usual notion of trace [Maz88] by associating the set of enabled events to the firing of each event in a sequence of event firings. Thus, each element of the trace keeps track of which events are enabled and which event fires at each step.

DEFINITION 4.1 (TRACE)

Let Σ *be an alphabet of events. A trace* $\theta = E_1 \xrightarrow{\mathsf{e}_1} E_2 \xrightarrow{\mathsf{e}_2} \cdots$ *is a sequence such that* $\forall i \geq 1 : E_i \subseteq \Sigma$ *and* $\mathsf{e}_i \in E_i$*, where* E_i *denotes the set of events enabled when* e_i *fires.*

■ 4.1

Henceforth, and for the sake of simplicity, all events in a trace will be assumed to be distinct. This assumption can always be enforced by renaming different occurrences of the same event. This renaming does not affect the validity of the theory presented.

Although it is an abstract notion, a trace has a direct correspondence with the notion of run in transition systems. Since a TS is a particular case of LzTS, and a TTS is described in terms of a TS plus certain delays, the following definition also applies to those models.

DEFINITION 4.2 (TRACES IN LAZY TRANSITION SYSTEMS)

Each run $\rho = \mathsf{s}_1 \xrightarrow{\mathsf{e}_1} \mathsf{s}_2 \xrightarrow{\mathsf{e}_2} \cdots$ *of a* LzTS *defines a trace* $\theta_\rho = E_1 \xrightarrow{\mathsf{e}_1} E_2 \xrightarrow{\mathsf{e}_2} \cdots$ *where* E_i *is the set of events enabled at* s_i, i.e. $E_i = \mathcal{E}(\mathsf{s}_i)$.

■ 4.2

Notions defined over the runs of a transition system can be naturally extended for their traces counterparts. Specially relevant for the verification problem are the notion of

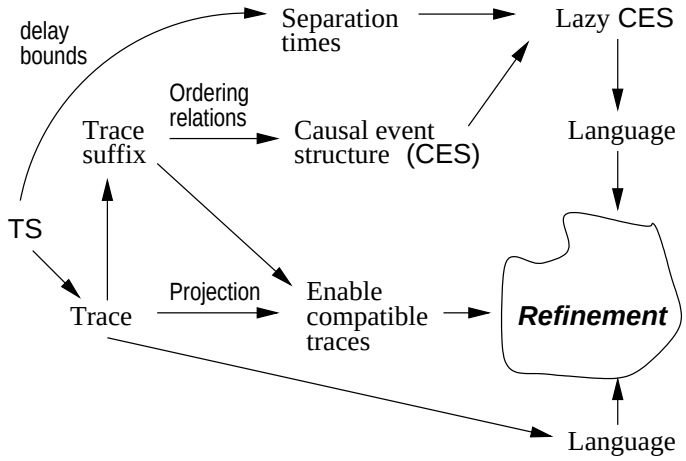


Figure 4.5 From traces to language refinement.

enabling interval of an event along a run (see Definition 2.4), and the notion of timing-consistent run (see Definition 2.7).

Figure 4.6 (a) depicts a run, taken from the TTS in Figure 4.2, and its corresponding trace. A state of the run is substituted in the trace by the set of events enabled at such state in the transition system. The enabling intervals of the events in the trace are depicted as vertical lines in Figure 4.6 (b).

Next, the language of a system is defined by the set of traces that it can generate. In the case of a TTS only the traces defined by the runs that satisfy the delays associated to the events of the system are considered. That is:

Definition 4.3 (Languages)

The language $\mathcal{L}(A)$ *of a* LzTS A *is the set of traces defined by all runs of* A. *The* language $\mathcal{L}(A)$ *of a* TTS $A = \langle A^-, \delta^l, \delta^u \rangle$ *is the set of traces defined by all timing-consistent runs of* A^-.

■ 4.3

Lemma 4.1 (Language inclusion)

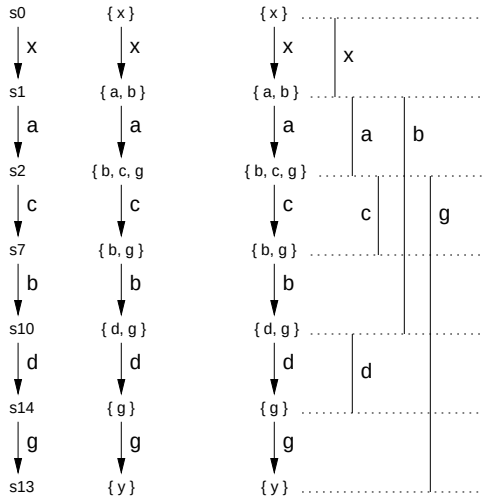
Let $A = \langle A^-, \delta^l, \delta^u \rangle$ *be a* TTS. *Then, its language is a subset of that of its underlying* TS, i.e. $\mathcal{L}(A) \subseteq \mathcal{L}(A^-)$.

■ 4.1

The proof of the lemma directly follows from Definition 2.7 and Definition 4.3.

4.3.2 Trace-based verification

In order to solve the verification problem for safety poperties, the language of the TTS that models the system under verification must be computed or, at least, conserva-



$$\delta^l(\mathsf{x}) \leq ft(\mathsf{x}) \leq \delta^u(\mathsf{x})$$
$$\delta^l(\mathsf{a}) \leq ft(\mathsf{a}) - ft(\mathsf{x}) \leq \delta^u(\mathsf{a})$$
$$\delta^l(\mathsf{b}) \leq ft(\mathsf{b}) - ft(\mathsf{x}) \leq \delta^u(\mathsf{b})$$
$$\delta^l(\mathsf{c}) \leq ft(\mathsf{c}) - ft(\mathsf{a}) \leq \delta^u(\mathsf{c})$$
$$\delta^l(\mathsf{g}) \leq ft(\mathsf{g}) - ft(\mathsf{a}) \leq \delta^u(\mathsf{g})$$
$$\delta^l(\mathsf{d}) \leq ft(\mathsf{d}) - ft(\mathsf{b}) \leq \delta^u(\mathsf{d})$$

$$ft(\mathsf{x}) \leq ft(\mathsf{a}) \leq ft(\mathsf{c}) \leq ft(\mathsf{b}) \leq ft(\mathsf{d}) \leq ft(\mathsf{g})$$

(c)

Figure 4.6 A trace taken from the TTS in Figure 4.2: (a) the original run (left) and the trace (right), (b) enabling intervals of the events in the trace, and (c) timing analysis of the trace.

tively estimated. According to Definition 4.3 this requires a mechanism to check whether a trace of the underlying TS is timing-consistent or not (see Definition 2.7). Checking the timing-consistency of a trace can be formulated quite simply by means of a set of expressions that bound the firing times of the events in the trace according to their delays.

The firing time of event e_i, denoted by $ft(\mathsf{e}_i)$, is bounded according to the expression:

$$ft(\mathsf{e}_j) + \delta^l(\mathsf{e}_i) \;\leq\; ft(\mathsf{e}_i) \;\leq\; ft(\mathsf{e}_j) + \delta^u(\mathsf{e}_i) \tag{4.1}$$

where e_j is the event that triggers e_i in the trace. Event enabled in the initial state of the trace have no trigger event, therefore its firing time is only bounded by its delays.

On the other hand, there are events in the trace which are disabled by the firing of another event. For example in the right of Figure 4.7, event e_k is enabled in the trace but its actual firing is prevented by the firing of another (disabler) event e_i. The disabling of e_k must occur before the maximum delay since e_k was enabled, has elapsed. Otherwise e_k should have fired yet. Conversely, the disabling can occur as soon as e_k is enabled, no matter if its minimum delay has already elapsed or not. Therefore, the firing time of the disabler event e_i is bounded according to the following expression:

$$ft(\mathsf{e}_j) \;\leq\; ft(\mathsf{e}_i) \;\leq\; ft(\mathsf{e}_j) + \delta^u(\mathsf{e}_k) \tag{4.2}$$

where e_j is the event that triggers e_k in the trace. Since for the disabling to occur in the trace (firing of e_i), the disabled event e_k must be already enabled by the firing of

Figure 4.7 Enabling and disabling in a trace: (a) event e_j enables event e_i, and (b) event e_k is disabled by the firing of event e_i (the disabler).

e_j, the firing of e_j always happens before that of e_i. Hence, the inequality in the left of expression (4.2) is actually redundant.

Finally, the order in which the events fire along the trace provides additional information for the timing analysis, *i.e.* time must monotonically increase as long as new events fire. Assuming that the events that fire in the trace are numbered according to their firing order, the following expression must hold:

$$\forall\ 1 \leq i\ :\ ft(e_i)\ \leq\ ft(e_{i+1}) \tag{4.3}$$

The conjunction of expressions (4.1), (4.2) and (4.3) determine the timing-consistency of the trace. If a solution can be found that assigns firing times to the events of the trace according to the set of inequalities, the trace is timing-consistent. Otherwise, the trace is not timing-consistent and therefore it does not belong to the language generated by the system. Checking the timing-consistency of a trace using the formulation provided by expressions (4.1), (4.2) and (4.3) can be easily performed using linear programming.

EXAMPLE 4.2 (CONT.) *Figure 4.6 (c) shows the set of constraints of the linear programming model to check the timing-consistency of the trace of Figure 4.6 (b). Since no disabling situation appears in the trace, only expressions (4.1) and (4.3) apply.*

In this case, the problem has no solution if the delays shown in Figure 4.2 (b) are considered. Therefore, the trace is not timing-consistent and does not belong to the language of the system. Notice that this result is coherent with that obtained in Example 4.2. The trace was removed from the LzTS *obtained after the first refinement of the verification approach (see Figure 4.3).* ■ 4.2

Provided the formulation developed above for the timing analysis on a trace, a trace-based method for the verification of timed systems can be devised. The method must consider all the traces leading from the initial state of the system up to the states where violations of the properties under verification occur. The system is correct if no failure

trace exists when the delays are taken into account, *i.e.* if none of the failure traces in timing-consistent. On the contrary, a timing-consistent failure trace provides a counterexample that proves the incorrectness of the system.

The impossibility of this trace-based method for verification is obvious. The number of traces between two states of the system may be extremely large or even infinite if cycles are allowed. Moreover, the number of failure states would suffer from the state explosion problem as well. Therefore, some strategy to alleviate this complexity is required.

4.3.3 Enabling compatibility

This section provides a fundamental result that helps addressing the complexity problem exposed by the trace-based verification method outlined above. In short, Theorem 4.1 estates that the results obtained from the timing analysis over a given trace can be applied to all those traces that have the same causality relations. The theorem is based upon the notion of *enabling-compatibility*, that characterizes the relation between traces in which events are enabled (disabled) by the same triggers (disablers), and events fire in the same order. Since the time at which an event fires or is disabled only depends on the instant it became enabled plus certain delays within the given bounds, the timing analysis for a trace is also applicable to all the traces that are enabling-compatible.

The notion of trace, as it is given by Definition 4.1, does not explicitly distinguish between those events in a trace that fire after being enabled for some time, and those events that are disabled by the firing of another event in the trace. However, the disabling phenomenon is relevant for the timing analysis over a set of traces and must be properly modeled. In the following definition, that complements Definition 4.1, each element of the trace keeps track of which events are enabled, which event fires at each step, and which events are disabled due to such firing.

DEFINITION 4.4 (DISABLING IN A TRACE)
Let $\theta = E_1 \xrightarrow{\mathsf{e}_1} E_2 \xrightarrow{\mathsf{e}_2} \cdots$ *be a trace. The set* $D_i \subset E_i$, $i \geq 1$ *is the set of events disabled by the firing of event* e_i *in* θ, *defined by* $D_i = \{\mathsf{d} \in E_i \mid \mathsf{d} \neq \mathsf{e}_i \wedge \mathsf{d} \notin E_{i+1}\}$. *The set of all the events disabled along trace* θ *is the set* $D(\theta) = \bigcup_{i \geq 1} D_i$. *We denote by* e_i ***dis*** d *the fact that event* e_i *disables event* d . *Event* e_i *is the* disabler *of* d *in* θ. ■ 4.4

EXAMPLE 4.3 *The circuit in Figure 4.8 (a) reacts to changes at the input signal* a *by producing some changes at the output signals* d, e *and* f. *In a particular run, after firing* a+, *the AND gate driving signal* d *is enabled to rise since inputs* b *and* e *are both high. However, a negative transition of* e *disables the gate switch. This situation can be observed in the corresponding untimed* TS *of Figure 4.8 (b). Transition* d+ *is enabled*

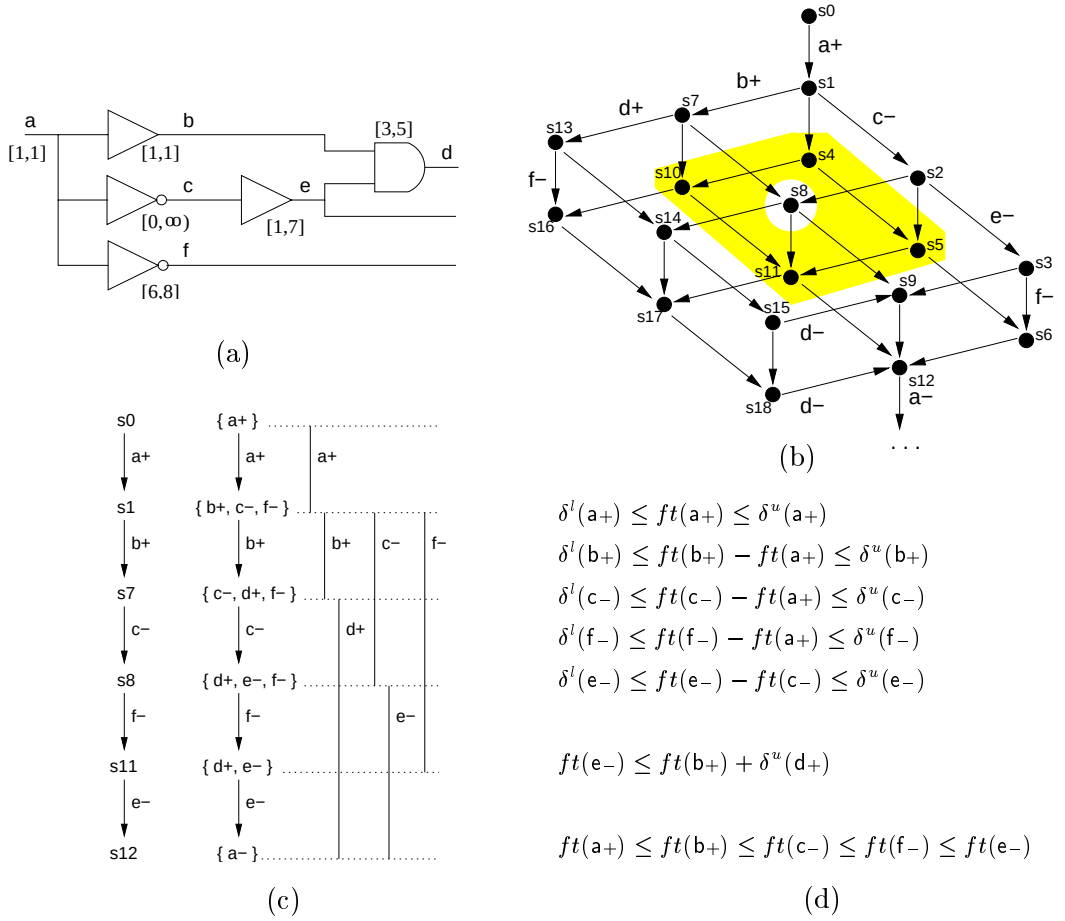


Figure 4.8 (a) Circuit with a potential disabling at gate d. (b) Portion of the timed state space (shadowed states are unreachable). (c) A run and the corresponding trace illustrating the enabling intervals and the disabling of event d+ due to the firing of e−. (d) Timing analysis of the trace.

in state s_7 *after* b+. *However, when* e− *occurs and state* s_{12} *is reached,* d+ *cannot happen anymore. Thus, event* d+ *is enabled for some time and then becomes disabled.*

The trace in Figure 4.8 (c) illustrates the disabling phenomenon. Given $E_5 = \{d+, e-\}$ *corresponding to state* s_{11} *and the firing of* e−, $E_6 = \{a-\}$ *(*s_{12}*) is reached where* d+ *is no longer enabled. Thus we have* $D_5 = \{d+\} \subseteq E_5$ *and* e− ***dis*** d+ .

Figure 4.8 (d) shows a linear programming model to check the timing-consistency of the trace. According to Definition 2.7 for timing-consistency and the formulation of the problem in the previous section, the inequality $ft(e-) \le ft(b+) + \delta^u(d+)$ *indicates that* e− *must disable* d+ *before its maximum possible firing time has elapsed. This constraint to expression (4.2) from Section 4.3.2.*

■ 4.3

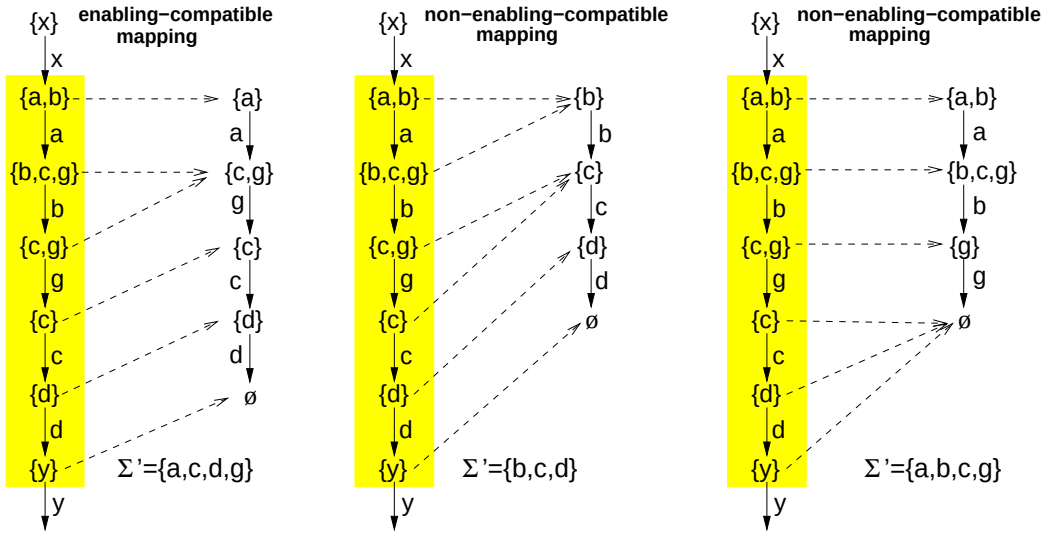


Figure 4.9 Enabling-compatible (left) and non-enabling-compatible mappings (center and right).

With all the above, the following definition introduces the cornerstone notion of the verification strategy presented in this thesis.

DEFINITION 4.5 (ENABLING-COMPATIBLE TRACE MAPPING)

Let $\theta = \cdots \longrightarrow E_0 \xrightarrow{e_0} E_1 \xrightarrow{e_1} E_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} E_{n+1} \longrightarrow \cdots$ *be a trace over the alphabet of events* Σ *and let* $\theta' = E'_1 \xrightarrow{e'_1} E'_2 \xrightarrow{e'_2} \cdots \xrightarrow{e'_m} E'_{m+1}$ *be a trace over the alphabet* $\Sigma' \subseteq \Sigma$. *Let* $\theta_t = E_1 \xrightarrow{e_1} E_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} E_{n+1}$ *be a fragment of* θ.

An enabling-compatible *mapping of* θ_t *onto* θ' *is a function* $\mathsf{map} : \{E_1, \ldots, E_{n+1}\} \mapsto \{E'_1, \ldots, E'_{m+1}\}$ *such that:*

a) $\mathsf{map}(E_1) = E'_1$ (initialization)

b) $\forall\ 1 \leq i \leq n,\ \mathsf{map}(E_i) = E_i \cap \Sigma'$ (projection)

c) $\forall\ 1 \leq i \leq n,\ (\mathsf{map}(E_i) = \mathsf{map}(E_{i+1}) \ \wedge\ \mathsf{e}_i \notin \Sigma') \ \vee$
$(\mathsf{map}(E_i) = E'_j \ \wedge\ \mathsf{map}(E_{i+1}) = E'_{j+1} \ \wedge\ \mathsf{e}_i = \mathsf{e}'_j)$ (firing)

■ 4.5

The mapping of θ onto θ' is a function that preserves the enabledness of the events in Σ'. Initially, the events enabled in E'_1 must also be enabled in E_1 (initialization condition). Next, the events of Σ' enabled along θ and θ' must be the same (projection condition). Moreover, θ may fire events that are not relevant to θ' (when $\mathsf{map}(E_i) = \mathsf{map}(E_{i+1})$ in the firing condition). The second part of of the firing condition captures implicitly the disabling of events produced by the firing of e_i in θ and the firing of e'_j in θ', that is if an event is disabled in one trace it must be disabled also in the other trace. Since the firing

time of an event only depends on its enabling time and its delay (see Section 2.3), this notion will allow us to apply the timing analysis of θ' to θ in the fragment θ_t.

Figure 4.9 shows three examples of trace mapping of the shadowed fragment. The mapping at the left, with $\Sigma' = \{\mathsf{a}, \mathsf{c}, \mathsf{d}, \mathsf{g}\}$, is enabling-compatible. The mapping at the center, with $\Sigma' = \{\mathsf{b}, \mathsf{c}, \mathsf{d}\}$, is not enabling-compatible since it violates the projection condition when taking $E_i = \{\mathsf{b}, \mathsf{c}, \mathsf{g}\}$ and $\mathsf{map}(E_i) = \{\mathsf{b}\}$. Clearly, a enables c in θ, whereas c is enabled by b in θ'. The mapping at the right, with $\Sigma' = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{g}\}$, is not enabling-compatible since it violates the projection condition when taking $E_i = \{\mathsf{c}, \mathsf{g}\}$ and $\mathsf{map}(E_i) = \{\mathsf{g}\}$. The firing of b in θ' disables c whereas this does not happen in θ.

The following theorem is the main theoretical result of the verification approach presented in this thesis. The theorem relies on the notion of enabling-compatibility between traces. Since the time at which events fire or are disabled only depends on the enabling instant plus certain delay, the timing analysis on a trace is also applicable to all the traces that share the same enabling, disabling and firing order. That is the timing analysis applies to the set of enabling-compatible traces.

Theorem 4.1 (Enabling-compatibility and timing-consistency)
Let θ, θ' and θ_t be traces with the same conditions as in Definition 4.5. Let map *be an enabling-compatible mapping from θ_t onto θ'. Let δ^l and δ^u be two functions that assign arbitrary min/max delays to the events in Σ' and 0 and ∞ delays to the events in $\Sigma \setminus \Sigma'$, respectively.*
Then, θ is timing-consistent $\iff$ θ' is timing-consistent.

Proof:

Given that events not in Σ' have delays in the interval $[0, \infty)$, no attention must be paid to their timing-consistency. So the proof can be concentrated on the events in Σ' that appear in θ_t and θ'.

$\boxed{\Rightarrow}$

Let $\tau_1 \leq \cdots \leq \tau_n$ be the time stamps assigned to $E_1, \ldots, E_{n+1}$ that make θ timing-consistent. The same time stamps can be assigned to θ' as follows. Let j be the smallest index such that $\mathsf{map}(E_j) = E'_i$. Then we assign the time stamp t_j to E_i. Under this assignment we have that for any $\mathsf{e}_k \in \Sigma'$, the time stamp assigned to $\mathsf{FirstEnabled}(\theta, E_j, \mathsf{e}_k)$ is the same as the one assigned to $\mathsf{FirstEnabled}(\theta', \mathsf{map}(E_j), \mathsf{e}_k)$. This is ensured by Definition 4.5, that enforces the set of enabled events in Σ' to be the same in E_j and $\mathsf{map}(E_j)$. Then, since the disabling of an event $\mathsf{e} \in \Sigma \cap \Sigma'$ must be due to the firing of another event $\mathsf{e}_l \in \Sigma \cap \Sigma'$ and such events are enabled and fire at the same time in both sides, the disabling of e must also occur at the same time stamp in θ and θ'. Now, by Definition 2.7 of timing-consistency, it immediately follows that the assignment of time stamps also makes θ' timing-consistent.

$\boxed{\Leftarrow}$

Given a set of consistent time stamps assigned to $E'_1, \ldots, E'_{m+1}$, *they can also be assigned to* $E_1, \ldots, E_{n+1}$ *by the function* map^{-1}. *Timing-consistency immediately follows by using a reasoning similar to the previous case.* ■ 4.1

The previous theorem states that the timing analysis of a trace can be reduced to the timing analysis of those events that are causally related (events in Σ'). Therefore, the events that are concurrent with all the events of Σ' can be abstracted out. Hence, the timing analysis for one trace can be applied to all those traces that have the same causality relations among the events in Σ'.

We want to remark that the notion of enabling-compatibility as well as the previous theorem have been updated with respect to those in [PCKP00] in order to properly accommodate the disabling notion into the theory.

4.4 Event structures

This section presents the basic theory on *causal event structures* (CES). A CES describes all the possible sequential and concurrent executions of a set of events. Thus, it allows to capture a set of enabling-compatible traces under a single mathematical object. Event structures are the only object for which we perform timing analysis, which is rather simple and efficient because CESs are acyclic directed graphs. The timing constraints derived from such analysis apply to the whole set of enabling-compatible traces covered by the CES.

The usual notion of CES is not able to model the disabling of events, which is a relevant phenomenon for our verification approach. A class of event structures with conflict relations was proposed in [NPW81]. However, such relations are symmetric and correspond to mutual disabling of competing events. A more general relation is required for our purposes that models the asymmetry of event disabling, such as it appears in digital circuits, for example. The main reason for considering asymmetric conflict relations is because, in our verification approach, event structures are derived from traces and a particular trace can only capture a single branch of a conflict relation. Consider the portion of TS in Figure 4.10 (a). It contains a symmetrical conflict since x and y mutually disable each other. Two important facts are observed in the trace of Figure 4.10 (b) extracted from such TS:

- Only the disabling of y due to the firing of x is captured by the trace.
- Since y is disabled, it can no longer fire along the trace and therefore it cannot enable other events.

The first fact leads to the need for considering an asymmetric conflict relation. The second fact imposes the restriction that a disabled event cannot have causal successors in the event

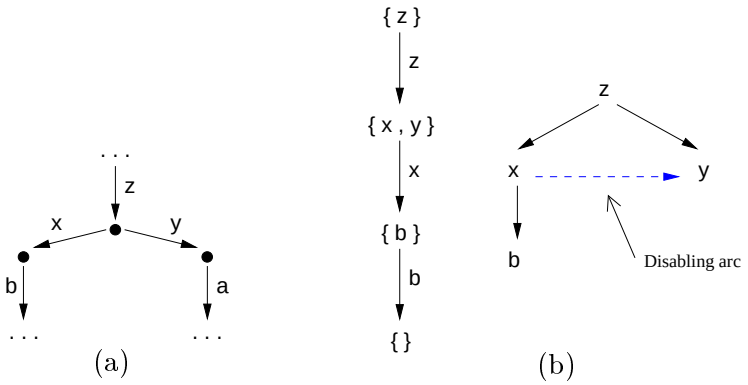


Figure 4.10 (a) Portion of a TS with a symmetric disabling relation. (b) A trace and the corresponding CES that capture the disabling.

structure. With these two ideas in mind an asymmetric conflict relation, denoted by $\triangleright$, is added to the notion of *causal event structure* used in [PCKP00], such that the disabling of events along a trace is properly handled.

Definition 4.6 (Causal event structure)

A causal event structure (CES), $CS = \langle \Sigma, \prec, \triangleright \rangle$, *is a finite set* Σ *of* events, *a* precedence relation $\prec \subset \Sigma \times \Sigma$ (*irreflexive, antisymmetric and transitive*) *called the* causality relation, *and a* conflict relation $\triangleright \subset \Sigma \times \Sigma$ (*irreflexive and antisymmetric*). $\triangleright$ *is inherited via* $\prec$ *in the sense of:* $\forall\, e_1, e_2, e_3 \in \Sigma \mid e_1 \triangleright e_2 \wedge e_1 \prec e_3 \Rightarrow e_3 \triangleright e_2$. *Moreover* $\prec$ *and* $\triangleright$ *satisfy the following two properties:*

- $\prec \cap \triangleright = \emptyset$ *and*
- $e_1 \triangleright e_2 \Rightarrow \nexists e_3 \in \Sigma : e_2 \prec e_3$.

That is, the causality and the disabling relations are disjoint, and disabled events cannot be causal predecessors of other events in the CES.

■ 4.6

Notice that this definition excludes symmetric conflicts, *i.e.* mutual disabling between events, by the anti-symmetry of $\triangleright$. This fact does not constitute a limitation of the model but an intended feature that fits in our purposes.

Given a CES $CS = \langle \Sigma, \prec, \triangleright \rangle$ and two events $e, e' \in \Sigma$, the *disabling relation* between e and e' is defined as follows:

$$e \triangleright_\mu e' \stackrel{def}{=} e \triangleright e' \wedge \forall e_1, e_1' \in \Sigma : [e_1 \prec e \wedge e_1' \prec e' \wedge e_1 \triangleright e_1' \Rightarrow e_1 = e \wedge e_1' = e']$$

$\triangleright_\mu$ identifies the minimal elements (under $\prec$) of the $\triangleright$ relation. The $\triangleright$ relation identifies pairs of events which are inconsistent due to the disabling of some of the predecessors, and propagates to causally-related events generating other conflicts.

A CES can then be depicted as a Hasse diagram, by showing the transitivity-irredundant $\prec$ relations in form of solid arcs and the $\triangleright_\mu$ relations as dashed arcs. Figure 4.10 (b) shows a CES that contains the disabling relation $\mathsf{x} \triangleright_\mu \mathsf{y}$. Other examples of causal event structures without disabling relations can be seen in Figure 4.3 (b) and Figure 4.4 (b).

Given a CES $CS = \langle \Sigma, \prec, \triangleright \rangle$ and a set of events $X \subseteq \Sigma$, the following sets can be defined [RE88]:

$$
\begin{aligned}
({}^{\rightarrow}X)_{\prec} &\stackrel{def}{=} \{\mathsf{e}_1 \in \Sigma \mid \exists\, \mathsf{e}_2 \in X \ : \ \mathsf{e}_1 \prec \mathsf{e}_2\} \\
(X^{\rightarrow})_{\prec} &\stackrel{def}{=} \{\mathsf{e}_1 \in \Sigma \mid \exists\, \mathsf{e}_2 \in X \ : \ \mathsf{e}_2 \prec \mathsf{e}_1\} \\
({}^{\circ}X)_{\prec} &\stackrel{def}{=} \{\mathsf{e}_1 \in X \mid \nexists\, \mathsf{e}_2 \in X \ : \ \mathsf{e}_2 \prec \mathsf{e}_1\} \\
(X^{\circ})_{\prec} &\stackrel{def}{=} \{\mathsf{e}_1 \in X \mid \nexists\, \mathsf{e}_2 \in X \ : \ \mathsf{e}_1 \prec \mathsf{e}_2\} \\
(\downarrow X)_{\prec} &\stackrel{def}{=} \{\mathsf{e}_1 \in \Sigma \mid \exists\, \mathsf{e}_2 \in X \ : \ \mathsf{e}_1 \preceq \mathsf{e}_2\} \ = \ ({}^{\rightarrow}X)_{\prec} \cup X
\end{aligned}
$$

When it is clear from the context, we will just write ${}^{\rightarrow}X$, $X^{\rightarrow}$, ${}^{\circ}X$, X° and $\downarrow X$. Intuitively, ${}^{\rightarrow}X$ is the part of CS *before* X, $X^{\rightarrow}$ is the part of CS *after* X, ${}^{\circ}X$ are the *root* (with no predecessors) events of CS, and X° are the *sink* (with no successors) events of CS. Finally, $\downarrow X$ is called the *left-closure* of X.

DEFINITION 4.7 (WORDS AND PREFIXES)

A word *of the events of* $CS = \langle \Sigma, \prec, \triangleright \rangle$ *is a sequence* $\omega = \mathsf{e}_1 \cdots \mathsf{e}_n \in \Sigma^n$ $(n \leq \mid \Sigma \mid)$, *such that all the events are distinct and* $\forall\ 1 \leq i, j \leq n$:

- $\mathsf{e}_i \prec \mathsf{e}_j \ \Rightarrow \ i < j$ *(events are ordered in* ω *according to* $\prec$*), and*
- $\mathsf{e}_i \triangleright_\mu \mathsf{e}_j \ \Rightarrow \ \mathsf{e}_j \notin \omega$ *(disabled events do not appear in* ω*).*

Given a word $\omega = \mathsf{e}_1 \cdots \mathsf{e}_i \mathsf{e}_{i+1} \cdots \mathsf{e}_n$, *the i-th prefix of* ω *is denoted by* $\omega_i = \mathsf{e}_1 \cdots \mathsf{e}_i$. *The empty prefix is denoted by* ω_0.

■ 4.7

Notice that an event e_j for which a disabling relation $\mathsf{e}_i \triangleright_\mu \mathsf{e}_j$ exists in the CES cannot fire along a word ω. However, e_j will become enabled somewhere in ω as long as its predecessors by $\prec$ fire in ω. Also, e_j will be disabled when its predecessors by $\triangleright$ fire in ω. This notions are formally captured by the following definition:

DEFINITION 4.8 (EVENTS ENABLED/DISABLED BY A PREFIX)

Let $CS = \langle \Sigma, \prec, \triangleright \rangle$ *be a* CES *and let* ω *be a word of* CS. *The set of events enabled by a prefix* ω_i *is defined as* $\mathcal{E}(\omega_i) = \{\mathsf{e}_k \notin \omega_i \mid \forall \mathsf{e}_j \in \Sigma \ : \ \mathsf{e}_j \prec \mathsf{e}_k \Rightarrow \mathsf{e}_j \in \omega_i \ \wedge \ \mathsf{e}_j \triangleright_\mu \mathsf{e}_k \Rightarrow \mathsf{e}_j \notin \omega_i\}$. *Similarly, the set of events disabled by a prefix* ω_i *is defined as* $\mathcal{D}(\omega_i) = \{\mathsf{e}_k \notin \omega_i \mid \exists \mathsf{e}_j \in \Sigma \ : \ \mathsf{e}_j \triangleright_\mu \mathsf{e}_k \Rightarrow \mathsf{e}_j \in \omega_i\}$.

■ 4.8

That is, an event e_k is enabled by a prefix ω_i if all the causal predecessor events (by $\prec$) are in ω_i, none of its disablers are in ω_i and e_k is not in ω_i. Also, an event is disabled by

ω_i if it was enabled and later disabled along ω_i but it never fired. $\mathcal{E}(\omega_i)$ contains all the events still enabled by the sequence of firings in ω_i, while $\mathcal{D}(\omega_i)$ accumulates all the events disabled along ω_i. In the sequel we will denote by $\mathcal{D} \subseteq \Sigma$ the set of all events that are disabled along some word of a CES, *i.e.* $\mathcal{D} = \{\mathsf{e}_i \in \Sigma \mid \exists\ \mathsf{e}_j \in \Sigma : \mathsf{e}_j \triangleright \mathsf{e}_i\}$.

The notion of word in a CES is similar to the general notion of trace. This fact is expressed by the following definition:

DEFINITION 4.9 (TRACES GENERATED BY WORDS)
Let $CS = \langle \Sigma, \prec, \triangleright \rangle$ *be a* CES *and let* $\omega = \mathsf{e}_1\mathsf{e}_2 \cdots \mathsf{e}_n$ *be a word of* CS. *The trace generated by* ω *is defined as:* $\theta_\omega = \mathcal{E}(\omega_0) \xrightarrow{\mathsf{e}_1} \mathcal{E}(\omega_1) \xrightarrow{\mathsf{e}_2} \cdots \xrightarrow{\mathsf{e}_{n-1}} \mathcal{E}(\omega_{n-1}) \xrightarrow{\mathsf{e}_n} \emptyset$.
■ 4.9

According the this definition, events disabled in the CES can appear only in the sets of enabled events in each state of θ_ω, but never in the transitions between states.

DEFINITION 4.10 (CAUSAL EVENT STRUCTURE GENERATED BY A TRACE)
Let $\theta = E_1 \xrightarrow{\mathsf{e}_1} E_2 \xrightarrow{\mathsf{e}_2} \cdots \xrightarrow{\mathsf{e}_{n-1}} E_n \xrightarrow{\mathsf{e}_n} E_{n+1}$ *be a finite trace with* $D(\theta)$ *as the set of disabled events along it. The causal event structure* $CS_\theta = \langle \Sigma, \prec, \triangleright \rangle$ *generated from* θ *is defined as follows:*

- $\Sigma = \bigcup_{i=1}^{n} E_i$.

  Not only the firing events are included but all those enabled along the trace, including the disabled ones.

- $\mathsf{e}_i \prec \mathsf{e}_j \ \Leftrightarrow\ i < j \ \wedge\ \{\nexists E_k \in \theta\ :\ \{\mathsf{e}_i, \mathsf{e}_j\} \subseteq E_k\} \ \wedge\ \mathsf{e}_i \notin D(\theta)$.
  The last condition emphasizes the fact that disabled events can not be causal predecessors of other event since they do not fire along the trace.

- $\mathsf{e}_i \triangleright \mathsf{e}_j \ \Leftrightarrow\ \begin{cases} \mathsf{e}_i \textbf{ dis } \mathsf{e}_j & \text{in } \theta \text{ , or} \\ \mathsf{e}_k \triangleright \mathsf{e}_j \ \wedge\ \mathsf{e}_k \prec \mathsf{e}_i \end{cases}$

  $\triangleright$ *captures the non-symmetric conflicts along the trace, that is* e_i *may disable* e_j *but this has nothing to do with* e_j *disabling* e_i. *Both relations can never appear together in the same trace.* $\triangleright_\mu$ *corresponds exactly to the* ***dis*** *relation of the trace.*
■ 4.10

EXAMPLE 4.3 (CONT.) *Figure 4.11 (a) depicts a trace extracted from the* TTS *in Figure 4.8. The trace contains the disabling of event* d+. *Figure 4.11 (b) shows the* CES *derived from the trace according to Definition 4.10. The disabling relation between* e− *and* d+ *is represented by the dashed arc.*

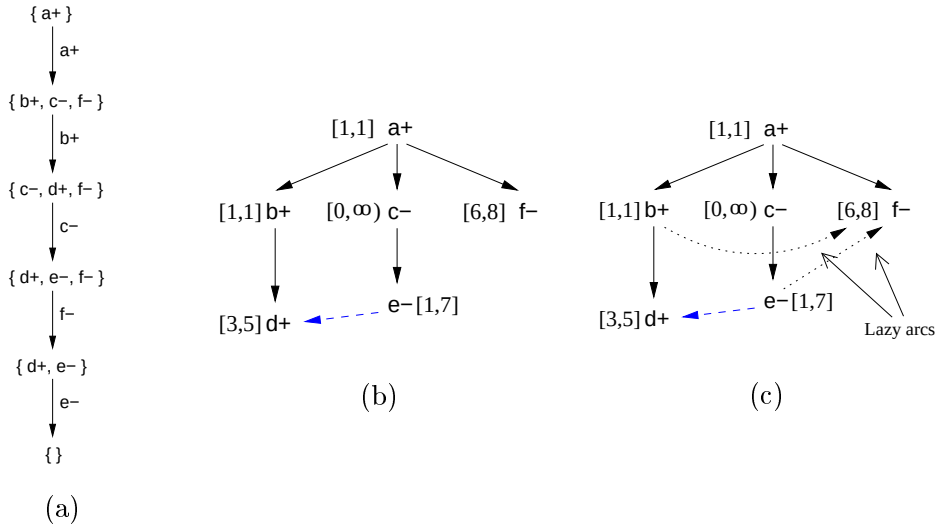


Figure 4.11 (a) A trace extracted from the TTS of Figure 4.8, (b) CES obtained from the trace (events are annotated with their delay bounds), and (c) lazy CES induced by the delays.

The following sequences of events are words of the CES *:* a+b+c−f−e− , a+f−c−e−b+ , a+c−b+e−f− , *etc.*

Consider ω = a+b+c−f−e− , *then* ω_1 = a+ *is the first prefix of* ω, ω_2 = a+b+ *is the second prefix of* ω, *etc. The set of events enabled and disabled by* ω_2 *are* $\mathcal{E}(\omega_2) = \{c-, d+, f-\}$ *and* $\mathcal{D}(\omega_2) = \emptyset$. *Similarly, the set of events enabled and disabled by* ω_5 *are* $\mathcal{E}(\omega_5) = \emptyset$ *and* $\mathcal{D}(\omega_5) = \{d+\}$. *Finally, according to Definition 4.9,* ω *generates the trace of Figure 4.11 (a).* ■ 4.3

Despite of the previous example, Figure 4.10, Figure 4.3 and Figure 4.4 show other examples of CESs derived from a given trace.

We use CESs derived from failure traces to perform timing analysis. Hence, relative timing relations among the events in the CES can be found that help to prove the timing-consistency or timing-inconsistency of a failure trace. Moreover, thanks to Theorem 4.1, the timing relations derived from the analysis also apply to the set of traces enabling-compatible with the failure trace.

4.4.1 Timing analysis on event structures

CESs with timing assumptions can be derived from traces with events annotated with minimum and maximum delay bounds (see Definition 4.10). These assumptions are captured by the notion of *maximal separation time* between the events of a CES. The *maximal separation time* of two events e_1 and e_2 is computed as the maximum difference be-

tween their firing times, provided any possible assignment of delays to the events in the CES. That is, $Sep_{max}(e_1, e_2) = max\{ft(e_1) - ft(e_2) \mid \textit{for any delay assignment}\}$, where ft denotes the firing time of an event.

In [MD92] several algorithms for the timing analysis on acyclic graphs where presented. Those algorithms included: a polynomial algorithm for the timing analysis with *max* constraints only; an exponential, but feasible in practice, algorithm for the case with *max* and linear constraints; and a branch and bound approach for the general case including *min/max* and linear constraints. The information obtained from these algorithms can be used to analyze whether two concurrent events are actually ordered in the timed domain. That is, e_1 precedes e_2 in the timed domain if $Sep_{max}(e_1, e_2) < 0$. Appendix A provides details on the timing analysis algorithm of [MD92] for *max* constraints.

The verification approach presented in [PCKP00] used the algorithm of [MD92] with only *max* constraints to perform timing analysis on CES derived from traces without disabling relations. However, such algorithm is not sufficient when the disabling of events is involved. The following example illustrates why.

Example 4.3 (cont.) *Recall the circuit of Figure 4.8 (a) where a falling transition of gate* e *disables the rising transition of gate* d*. Assume also, that a situation in which if "*f− occurs before e− once a+ has happened*" is considered a failure by the designer of the circuit. A trace that captures such failure and also includes the aforementioned disabling situation is shown in Figure 4.11 (a). Provided the delay bounds of the events shown in Figure 4.8 (a), it can be proved that the trace is not timing-consistent. Notice that if* e− *disables* d+*, then* e− *must fire before the maximum possible firing time of* d+ *has elapsed (see Definition 2.7). That is, the latest firing time of* e− *must be, some amount of time before 6 time units, after* a+ *fired. This means that* e− *will always fire before* f−*, whose minimum delay is also 6 time units and is also triggered by the reference event* a+*. This fact can be better analyzed by looking at the* CES *in Figure 4.11 (b), in which events have been annotated with their respective delay intervals.*

■ 4.3

According to the discussion in the previous example, the timing analysis on the CES should provide a relative timing relation showing the fact that e− always fires before f−. However, the *max*-only algorithm cannot handle the disabling situation and no such timing relation can be obtained. This indicates that e− and f− can occur concurrently in the timed domain. Which, in turn, implies that the failure trace is possible even when the delays are considered. Therefore the circuit is faulty. This leads to contradiction since the failure trace of the example is not timing-consistent.

The source of the contradiction resides in the fact that the disabling relation between e− and d+ cannot be expressed in the *max*-only algorithm for timing analysis, which can only handle the causal relations among the events. Moreover, such disabling relation is

relevant for the timing analysis of this case. If the disabling is not considered, the relative timing relation between $\mathsf{e}-$ and $\mathsf{f}-$ necessary to prove the timing-inconsistency of the failure trace, cannot be found.

Disabling relations are incorporated to the timing analysis of a CES in the form of linear constraints. Such constraints express the fact that if event e_i disables event e_k, then e_i fires before the maximum delay of e_k has elapsed, since it was enabled by its trigger e_j. Otherwise e_k should have fired before the disabling could take place. More formally, given a CES $CS = \langle \Sigma, \prec, \triangleright \rangle$, such that $\mathsf{e}_i \triangleright_\mu \mathsf{e}_k$ and $\mathsf{e}_j \prec \mathsf{e}_k$ $(\mathsf{e}_i, \mathsf{e}_j, \mathsf{e}_k \in \Sigma)$ a linear constraint of the form $ft(\mathsf{e}_i) \leq ft(\mathsf{e}_j) + \delta^u(\mathsf{e}_k)$ is added to the timing analysis. Then the timing analysis algorithm for *max* and *linear* constraints in [McM92] can be used.

In the previous example, a linear constraint corresponding to the disabling of $\mathsf{d}+$ by $\mathsf{e}-$ is imposed to the timing analysis, such that $\mathsf{e}-$ must fire before the delay of $\mathsf{d}+$ elapses. That is $ft(\mathsf{e}-) \leq ft(\mathsf{b}+) + \delta^u(\mathsf{d}+)$. Under this condition, which reflects what happens in the trace, we have that $Sep_{max}(\mathsf{e}-, \mathsf{f}-) < 0$. As we expected, this means that $\mathsf{e}-$ precedes $\mathsf{f}-$ in the timed domain.

The maximum separation times computed by the timing analysis algorithm can be incorporated to the CES in the form of relative timing constraints between pairs of events. Thus, e_1 precedes e_2 in the timed domain if $Sep_{max}(\mathsf{e}_1, \mathsf{e}_2) < 0$. We refer to these new relations as *lazy relations*, expressing the fact that e_2 is lazy to fire until e_1 has fired. This notion of laziness is similar to that for lazy transition systems (see Section 2.4).

DEFINITION 4.11 (LAZY CES GENENERATED BY A TRACE)
Let $\theta = E_1 \xrightarrow{\mathsf{e}_1} \cdots E_n \xrightarrow{\mathsf{e}_n} E_{n+1}$ *be a trace of a* TTS $A = \langle A^-, \delta^l, \delta^u \rangle$ *, and let* $CS_\theta = \langle \Sigma, \prec, \triangleright \rangle$. *The triple* $LCS = \langle \Sigma, \prec', \triangleright \rangle$ *is called a* lazy causal event structure (LzCES), *where* $\prec' = \prec \cup T$ *, and* $T \subseteq \Sigma \times \Sigma$ *is a set of* lazy relations *such that* $T = \{(\mathsf{e}_i, \mathsf{e}_j) \in \Sigma \times \Sigma \mid \mathsf{e}_i \not\prec \mathsf{e}_j \wedge \mathsf{e}_j \not\prec \mathsf{e}_i \wedge Sep_{max}(\mathsf{e}_i, \mathsf{e}_j) < 0\}$.
■ 4.11

The LzCES obtained after the timing analysis on the CES of Example 4.3 is shown in Figure 4.11 (c). The lazy relations corresponding to the relative timing information are depicted as dotted lines. Despite of the previous example, Figure 4.3 and Figure 4.4 show other examples of LzCESs derived from a given trace.

The delays assigned to the events in the CES for timing analysis play a crucial role in the context of our verification approach. Failure traces and/or CESs may come from a variety of sources: a designer, who's knowledge about the possible sources of failures in a system can be useful to guide the verification; a CES derived not from a whole trace that starts from the initial state, but from a portion of trace that ensures a localized failure analysis that involves less events; etc. In these cases, the prehistory of the enabledness of some events involved in the analysis may be unknown to the verification algorithm. As

a consequence, given a CES $CS = \langle \Sigma, \prec, \triangleright \rangle$, the minimum delay bound for all the root events $((^\circ\Sigma)_\prec)$ *i.e.* those for which no causal predecessor exists, is conservatively set to 0. That is, mimicking an infinitely early enabling time. With this strategy, timing analysis is still exact in case the CES has only one root event, since the relative firing order of all other events does not depend on the enabling time of their common predecessor.

A LzCES , obtained by the techniques described in previous sections, partially specifies the behavior of the system under verification and incorporates a set of relative timing constraints. Such timing constraints are mapped back to the system under verification by means of composing appropriately the system and the LzCES.

4.5 Enabling-compatible product

This section describes how to refine the set of traces produced by a LzTS by considering the timing constraints coming from event delay bounds. The timing constraints are derived by a timing analysis on a CES corresponding to an eligible trace of a LzTS in the untimed domain. The refinement is performed through the parallel composition of a LzTS and a LzCES. Defining such composition requires both descriptions to be represented in a uniform way. To satisfy this requirement we first introduce a state-based representation for CESs.

4.5.1 State-based representation of a CES

An underlying transition system can be obtained from a CES. This process relies on the notion of *configuration*, which plays the role of global state of the CES.

Definition 4.12 (Configuration)
Let $CS = \langle \Sigma, \prec, \triangleright \rangle$ *be a CES.* $\mathcal{C} \subseteq \Sigma$ *is a* configuration *iff:*

- $\mathcal{C}$ *is left-closed,* i.e. $\forall \mathsf{e}_i \in \mathcal{C}$ *all predecessors of* e_i *by* $\prec$ *are in* $\mathcal{C}$*, and*
- *disabled events do not belong to* $\mathcal{C}$*,* i.e. $\mathsf{e}_i \in \mathcal{C} \;\Rightarrow\; \nexists \mathsf{e}_j \in \Sigma \;:\; \mathsf{e}_j \triangleright_\mu \mathsf{e}_i$.

Notice that both $\emptyset$ *and the set of not disabled events* $\Sigma \backslash \mathcal{D}$ *are trivial configurations.*
Event $\mathsf{e} \in \Sigma$ *is* enabled *in configuration* $\mathcal{C}$ *iff* $^\rightarrow\{\mathsf{e}\} \subseteq \mathcal{C}$ *and* $\forall \mathsf{e}_j \in \Sigma \mid \mathsf{e}_j \triangleright \mathsf{e}_i \;:\; \mathsf{e}_j \notin \mathcal{C}$*. We denote by* $\mathcal{E}(\mathcal{C})$ *the set of all enabled events in configuration* $\mathcal{C}$*.*
■ 4.12

Configuration $\mathcal{C}$ precisely identifies a state of a CES, as the set of events occurred so far, such that if $\mathsf{e} \in \mathcal{C}$ all its causal predecessors must be also in $\mathcal{C}$.

Every prefix ω_i of a word ω in a CES is left-closed and disabled events do not fire along it (see Definition 4.7). Thus every prefix ω_i defines a configuration which is reached by firing the events from ω_i. Consideration of all possible words of a CES and their prefixes gives the *set of reachable configurations*, C, where the initial configuration due to the empty prefix ω_0 is denoted by $\top$. The set of reachable configurations together

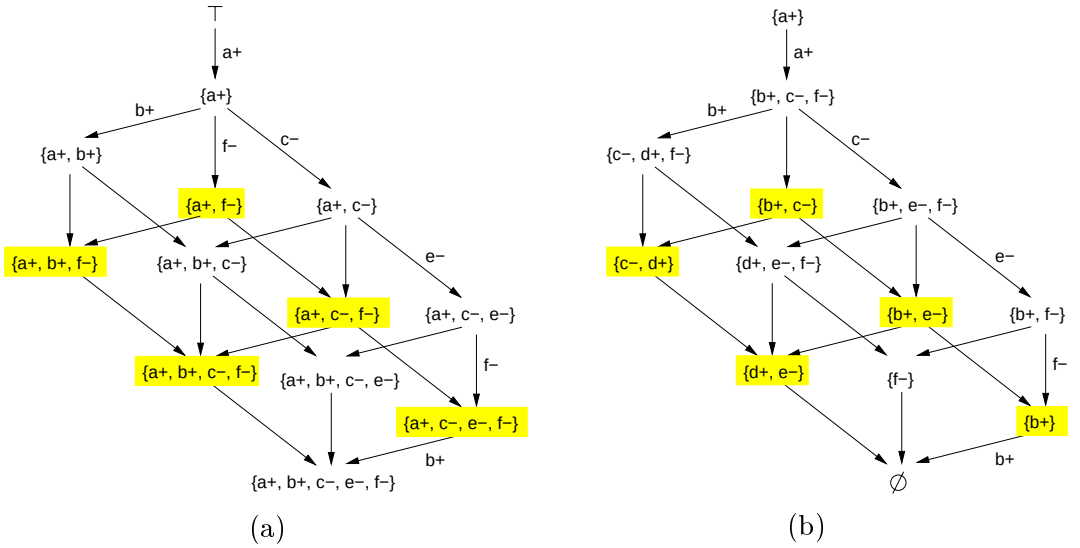


Figure 4.12 (a) Graph of reachable configurations for the LzCES of Figure 4.11 (c). (b) Corresponding graph of reachable enablings. Shadowed configurations are not reachable due to the laziness of event f−.

with the partial order induced by the strict set inclusion $\subset$, defines the *graph of reachable configurations.*

DEFINITION 4.13 (GRAPH OF REACHABLE CONFIGURATIONS)

Let $CS = \langle \Sigma, \prec, \triangleright \rangle$ *be a* CES, *and* C *be the set of reachable configurations of* CS. *The* graph of reachable configurations (GRC) *of* CS *is a Hasse diagram over* C *and the partial order* $\subset$ *interpreted in set-theoretical sense.*

■ 4.13

For the general case of a LzCES, $LCS = \langle \Sigma, \prec', \triangleright \rangle$, the graph of reachable configurations can be modeled by a LzTS $G = \langle C, \Sigma, T, \top, \mathsf{EnR} \rangle$ where: there is one state per configuration; $\mathcal{C}_1 \xrightarrow{\mathsf{e}} \mathcal{C}_2 \in T$ iff $\mathcal{C}_2$ is reached by firing $\mathsf{e} \in \Sigma$ from $\mathcal{C}_1$; the initial state corresponds to the initial configuration $\top$; and $\mathsf{EnR}(\mathsf{e}) = \{\mathcal{C} \in C \mid \mathsf{e} \in \mathcal{E}(\mathcal{C})\}$.

EXAMPLE 4.3 (CONT.) *Figure 4.12 (a) depicts the resulting graph of reachable configurations for the CES in Figure 4.11 (b). In this graph every arc* $(\mathcal{C}_1, \mathsf{e}, \mathcal{C}_2)$ *is attributed by an event* e *which expands configuration* $\mathcal{C}_1$ *into* $\mathcal{C}_2$ *(the firing event). The shadowed configurations are those unreachable due to the laziness of* f− *relative to* b+ *and* e− *as imposed by the lazy arcs of the LzCES of Figure 4.11 (c). Thus, we have that* $\mathsf{EnR(f+)} = \{\{\mathsf{a+}\}, \{\mathsf{a+}, \mathsf{b+}\}, \{\mathsf{a+}, \mathsf{c-}\}, \{\mathsf{a+}, \mathsf{b+}, \mathsf{c-}\}, \{\mathsf{a+}, \mathsf{c-}, \mathsf{e-}\}, \{\mathsf{a+}, \mathsf{b+}, \mathsf{c-}, \mathsf{e-}\}\}$ *and* $\mathsf{FR(f+)} = \{\{\mathsf{a+}, \mathsf{b+}, \mathsf{c-}, \mathsf{e-}\}\}$.

■ 4.3

The following theorem shows that a configuration in a CES is uniquely defined by the set of events enabled in it. The result applies in general to a LzCES since its set of reachable configurations is a subset of that of the original CES from which the LzCES was derived.

Theorem 4.2 (Configurations and enablings)
Any pair of configurations $\mathcal{C}_1$ *and* $\mathcal{C}_2$ $(\mathcal{C}_1 \neq \mathcal{C}_2)$ *of a CES* $CS = \langle \Sigma, \prec, \triangleright \rangle$ *has different sets* $\mathcal{E}(\mathcal{C}_1)$ *and* $\mathcal{E}(\mathcal{C}_2)$ *of enabled events,* i.e. $\mathcal{C}_1 \neq \mathcal{C}_2 \Rightarrow \mathcal{E}(\mathcal{C}_1) \neq \mathcal{E}(\mathcal{C}_2)$.

Proof:

By contradiction. Suppose that $\mathcal{E}(\mathcal{C}_1) = \mathcal{E}(\mathcal{C}_2)$. *Two cases arise:*

$\boxed{\mathcal{C}_1 \subset \mathcal{C}_2}$ *The configurations are ordered (similarly for* $\mathcal{C}_2 \subset \mathcal{C}_1$*).*

Then, any sequence of firings σ *, from* $\mathcal{C}_1$ *to* $\mathcal{C}_2$ *must contain at least one event* $\mathsf{e} \in \mathcal{E}(\mathcal{C}_1)$. *If* $\mathcal{E}(\mathcal{C}_1) = \mathcal{E}(\mathcal{C}_2)$ *, we have that* e *fires in* $\mathcal{C}_1$ *but is again enabled in* $\mathcal{C}_2$. *Therefore,* $\mathsf{e} \prec \mathsf{e}$ *, which is a contradiction.*

$\boxed{\mathcal{C}_1 \not\subset \mathcal{C}_2 \ \wedge \ \mathcal{C}_2 \not\subset \mathcal{C}_1}$ *The configurations are not ordered.*

Let us consider the nearest predecessor configuration $\mathcal{C}_3$ *such that* $\mathcal{C}_3 \subset \mathcal{C}_1$ *and* $\mathcal{C}_3 \subset \mathcal{C}_2$ *, and there is no configuration* $\mathcal{C}_4$ *such that* $\mathcal{C}_4 \subset \mathcal{C}_1$, $\mathcal{C}_4 \subset \mathcal{C}_2$ *and* $\mathcal{C}_3 \subset \mathcal{C}_4$.

Let e *be the first event firing in any feasible sequence of firings* σ_1 *from* $\mathcal{C}_3$ *to* $\mathcal{C}_1$. *Clearly,* e *does not appear in any feasible sequence of firings* σ_2 *from* $\mathcal{C}_3$ *to* $\mathcal{C}_2$ *, otherwise* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *would be reachable from* $\mathcal{C}_4$ *such that* $\mathcal{C}_3 \xrightarrow{\mathsf{e}} \mathcal{C}_4$. *Therefore,* e *should be still enabled in* $\mathcal{C}_2$. *Since we assumed that* $\mathcal{E}(\mathcal{C}_1) = \mathcal{E}(\mathcal{C}_2)$ *we have that* e *is again enabled in* $\mathcal{C}_1$ *and therefore,* $\mathsf{e} \prec \mathsf{e}$ *, which is a contradiction.* ■ 4.2

In the sequel we will indistinctly use configurations or their enablings to characterize the states of a CES. Based on this one-to-one correspondence instead of a graph of reachable configurations one could consider an isomorphic *graph of reachable enablings* (GRE). Figure 4.12 (b) shows the GRE corresponding to the GRC of Figure 4.12 (a).

4.5.2 Refining the reachability space by timing constraints

At this moment we have two objects at hand: a lazy TS A, and another lazy TS G obtained from an event structure CS_θ. CS_θ is derived from a particular trace θ of A (actually by an appropriate suffix), thus giving only a partial specification of the behavior of A. CS_θ is refined through timing analysis yielding the lazy TS G.

Refining the behavior of A by the timing constraints incorporated in G can be done by calculating the *enabling-compatible product* of G and A, which is a particular case of transition system product under the restrictions of making synchronization by the *same transitions* and the *same enabling conditions.*

For sake of simplicity, and before introducing the rules of the enabling-compatible product below, we will add the special configuration $\perp$ to G. $\perp$ denotes the fact that the

product is not synchronizing, *i.e.* there is no enabling-compatibility with the state space of the CES and therefore, timing analysis does not apply for the involved traces.

Given the system $A = \langle S, \Sigma_A, T_A, \mathsf{s}_0, \mathsf{EnR}_A \rangle$ and the state space of the LzCES containing the relative timing constraints $G = \langle C \cup \bot, \Sigma_G, T_G, \top, \mathsf{EnR}_G \rangle$, with $\Sigma_G \subseteq \Sigma_A$, the enabling-compatible product of A and G is a new LzTS $\langle S', \Sigma_A, T', \mathsf{s}'_0, \mathsf{EnR}' \rangle$ where:

- $S' \subseteq S \times (C \cup \bot)$,
- $\mathsf{s}'_0 = (\mathsf{s}_0, \top)$ if $\mathcal{E}(\top) \subseteq \mathcal{E}(\mathsf{s}_0)$, and $\mathsf{s}'_0 = (\mathsf{s}_0, \bot)$ otherwise, and
- $\forall \mathsf{e} \in \Sigma_A,\ \mathsf{EnR}'(\mathsf{e}) = \{(\mathsf{s}, \mathcal{C}) \in S' \mid \mathsf{s} \in \mathsf{EnR}_A(\mathsf{e})\}$.

The transition relation T' is defined by the rules below. The rules are implied by the conditions of Definition 4.5 on enabling-compatibility of traces. The fact that $(\mathsf{s}, \mathcal{C}) \in S'$ denotes that s and $\mathcal{C}$ have been reached by prefixes that are enabling-compatible, and that $\mathsf{map}(\mathcal{E}(\mathsf{s})) = \mathcal{E}(\mathcal{C})$. Given a state of the product $(\mathsf{s}, \mathcal{C})$ with $\mathcal{C} \neq \bot$, we will say that the state is in the timed domain, indicating that the timing analysis performed on CS_θ can be applied to s.

The rules that define the enabling-compatible product are as follows:

Transitions entering the timed domain

Transition	**Conditions**
$(\mathsf{s}, \bot) \xrightarrow{\mathsf{e}} (\mathsf{s}', \top)$	$\mathsf{enter} \equiv \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T_A \ \wedge\ \mathcal{E}(\top) \subseteq \mathcal{E}(\mathsf{s}') \cap \Sigma_G$

These transitions are fired when the events enabled in $\top$ are also enabled in s'. Thus, timing analysis can start being applied from $(\mathsf{s}', \top)$.

Staying inside the timed domain

Transition	**Conditions**
$(\mathsf{s}, \mathcal{C}) \xrightarrow{\mathsf{e}} (\mathsf{s}', \mathcal{C})$	$\mathsf{inside1} \equiv \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T_A \ \wedge\ \mathcal{E}(\mathsf{s}) \cap \Sigma_G = \mathcal{E}(\mathsf{s}') \cap \Sigma_G$
$(\mathsf{s}, \mathcal{C}) \xrightarrow{\mathsf{e}} (\mathsf{s}', \mathcal{C}')$	$\mathsf{inside2} \equiv \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T_A \ \wedge\ \mathcal{C} \xrightarrow{\mathsf{e}} \mathcal{C}' \in T_G \ \wedge\ \mathcal{E}(\mathsf{s}') \cap \Sigma_G = \mathcal{E}(\mathcal{C}')$

Inside1 corresponds to the condition in which e does not synchronize with G. Here the enablings of configuration $\mathcal{C}$ must be preserved, *i.e.* the firing of e cannot disable or enable events in Σ_G.

For inside2, both A and G make a synchronized move which might affect the events from Σ_G in exactly the same way: if $\mathsf{a} \in \Sigma_G$ becomes enabled in A due to this move, it should also become enabled in G, and viceversa.

Exiting or staying outside the timed domain

Transition	**Conditions**
$(\mathsf{s}, \mathcal{C}) \xrightarrow{\mathsf{e}} (\mathsf{s}', \bot)$	$\mathsf{exit} \equiv \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T_A \ \wedge\ \neg(\mathsf{enter} \ \vee\ \mathsf{inside1} \ \vee\ \mathsf{inside2})$

It can be shown that, in the enabling-compatible product, only the traces of the original LzTS which are enabling-compatible with the event structure are refined. This refinement excludes the traces which are not timing-consistent with respect to the timing constraints coming from the timing analysis on the event structure. All other traces are not changed, thus guaranteeing the conservativeness of the approach.

Example 4.3 (cont.) *Figure 4.13 summarizes the refinement of the state space of the circuit of the running Example 4.3. The figure shows the circuit (a) and a portion of its untimed state space (b). A given trace and the LzCES derived from it using the delays of the circuit are shown in (c) and (d), respectively. The LzTS corresponding to the GRC of the LzCES is shown in (e). Finally, (f) shows a portion of the resulting LzTS after performing the enabling-compatible product of (e) and the original TS of (b). States annotated with $\perp$ correspond to those states where the enabling-compatibility is not satisfied, and thus are out of the product. Notice that all traces where event* d+ *fires lead out of the product, since they are not enabling-compatible with the LzCES (d), where event* d+ *is disabled.* ■ 4.3

Despite of the previous example, Figure 4.3 and Figure 4.4 show other examples of enabling-compatible product.

4.6 Verification methodology

The different elements of the verification methodology have been introduced along the previous sections. The complete verification algorithm is presented in this section. Relevant aspects such as the correctness and the convergence of the approach are discussed.

The proposed verification methodology follows a fully automated iterative approach. The verification flow is graphically depicted in Figure 4.14.

The verification starts by taking a LzTS equivalent to the underlying TS of the system under analysis, modeled as a TTS. In that case the enabling and the firing information of all the events coincide since no timing information has been considered yet.

Given a safety property P, a trace is identified that leads to some state in which P is violated. If the trace is timing-consistent then the system does not satisfy the required property and the trace provides a counter-example. On the contrary, if the trace is not timing-consistent, it is used to refine the untimed state space and remove other timing-inconsistent traces. Causality information between the events in the trace is extracted and a CES is built from it. Timing analysis on the CES is performed by using the algorithm in [McM92]. The extracted temporal information is used to obtain a LzCES which is composed with the original LzTS, thus including the temporal information necessary to prove that some of the states in the system are unreachable. In particular, at least the

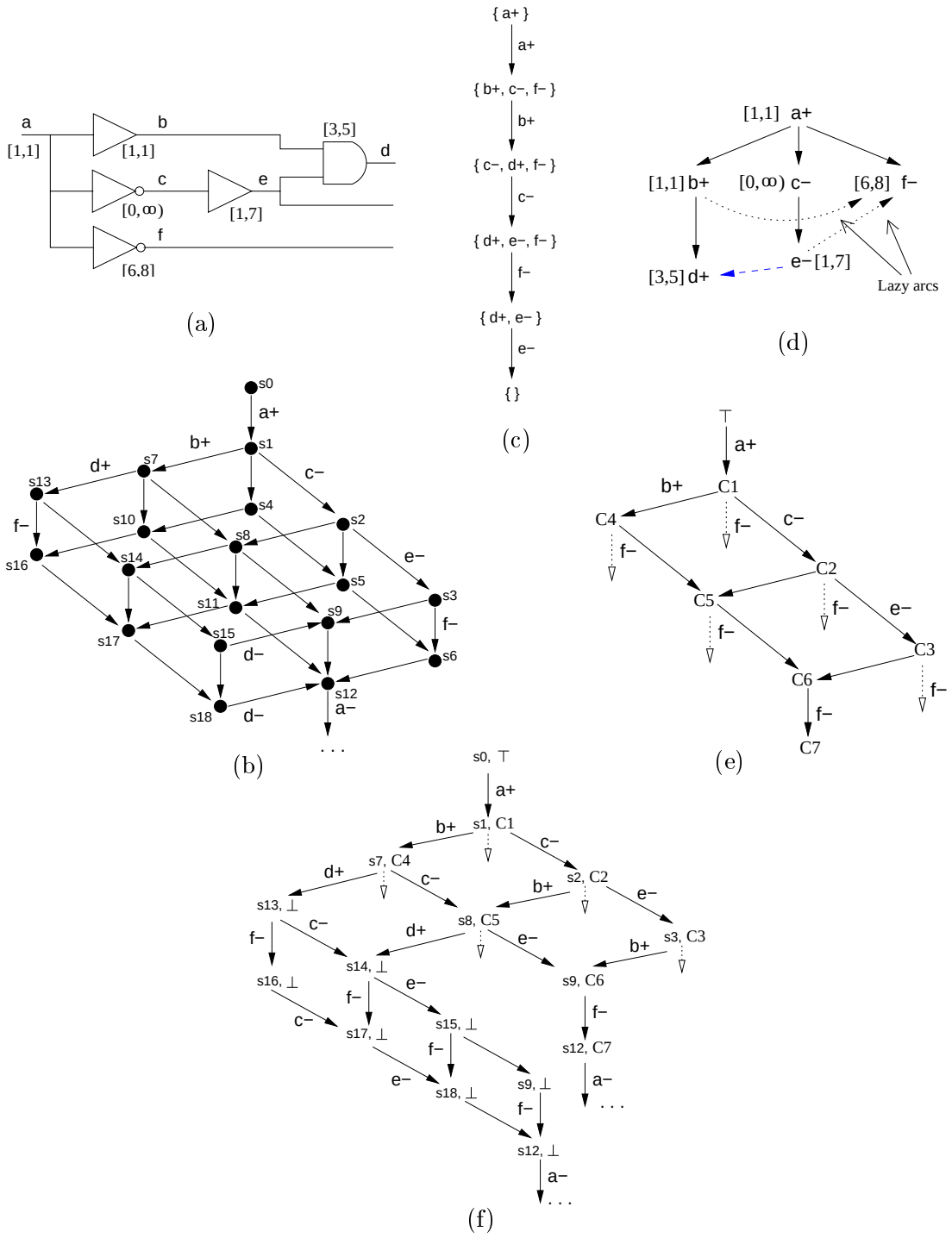


Figure 4.13 (a) Circuit with a potential disabling at gate d. (b) Portion of the untimed state space and (c) trace extracted from it. (d) LzCES induced by the trace (c) and delays. (e) LzTS of the corresponding GRC and (f) resulting LzTS after the enabling-compatible product of (b) and (e).

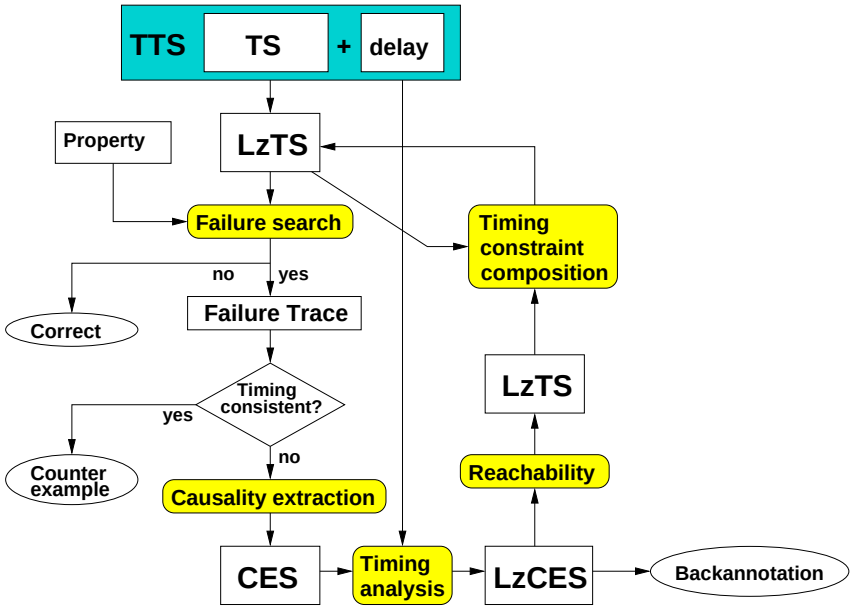


Figure 4.14 Flow of the verification methodology.

failure trace found in the initial step is removed. The process is repeated until no violation of the property P exists or a timing-consistent failure trace is found.

Along the series of refinements each LzCES is reported. At the end of the process, the resulting set of LzCESs constitute a set of sufficient relative timing constraints that prove the correctness of the system. They can also be used as valuable back-annotation information to help the designer improve his/her knowledge of the parts of the system which are critical for its correct operation.

4.6.1 Iterative refinement

Figure 4.15 shows the timed verification algorithm, where A is the TTS that models the system under verification, and P is a safety property. In general, a set of safety properties can be handled simultaneously with similar computational effort.

First, a LzTS A' is obtained corresponding to the underlying TS of A. The enabling information in A' coincides with that of firing, since no refinement with the timing information has been carried out yet.

The function *untimed_verification* checks whether a trace violating the property P is present in A'. If such a trace exists, a finite prefix, θ, demonstrating the wrong behavior is returned. This prefix is checked for timing-inconsistency by building and analyzing the corresponding causal event structure (see function *build_event_structure* in Figure 4.16). If no CES can disprove the feasibility of the trace θ the verification returns θ as an

```
function timed_verification ( A = ⟨S_A, Σ_A, T_A, s0_A, δ^l, δ^u⟩, P )
    A' = ⟨S_A, Σ_A, T_A, s0_A, EnR⟩ ;
    repeat
        θ := untimed_verification(A', P);
        if (empty θ) return(SUCCESS);
        LCS := build_event_structure(A', θ, δ^l, δ^u);
        if (empty LCS) return(FAIL, θ);
        A'' := compose(A', LCS);
        A' := A'';
    end repeat
end function
```

Figure 4.15 Main algorithm of the relative timing-based verification approach.

```
function build_event_structure ( A' = ⟨S, Σ, T, s0, EnR⟩, θ, δ^l, δ^u )
    θ'' := shortest_suffix(θ);
    repeat
        θ'' := add_predecessor(θ'', θ);
        CS := build_event_structure(A', θ'');
        if (timing_consistent(CS, δ^l, δ^u))
            L := compute_lazy_arcs(CS, δ^l, δ^u);
            LCS := add_lazy_arcs(CS, L);
            return (LCS);
        end if
    while (θ'' ≠ θ);
    return (empty CS);
end function
```

Figure 4.16 Algorithm for the derivation of a LzCES from a trace.

example of violation of P. Otherwise the system is refined through the composition with the LzCES LCS. LCS contains a set of relative timing constraints that apply over a set of enabling-compatible traces, including θ.

The *timed_verification* algorithm does not depend on any particular implementation of the *untimed_verification* function. We have implemented, however, an approach based on efficient symbolic model checking techniques [BCM+92]. Basically, we explore A' looking for failure states where P is violated. Then, a backward traversal is performed to generate

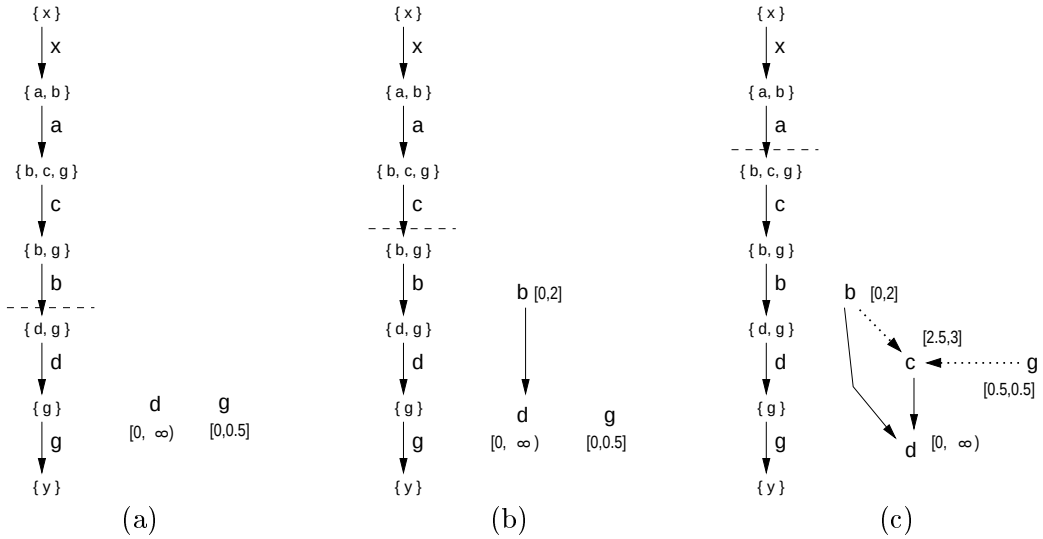


Figure 4.17 Generation of the sufficient shortest suffix of a trace. Three steps are needed to obtain a LzCES that proves the timing-inconsistency of the trace.

a trace, leading from the initial state to the failure one, reproducing the discrepancy with P. A fast simulation-guided traversal technique [PP03] has been also implemented. With this technique, the cost of the search for property violations is drastically reduced, and the failure trace is incrementally built during the process. Moreover, significant savings in CPU time and memory requirements are achieved.

4.6.2 Off-line timing analysis of failures

The function *build_event_structure* (see Figure 4.16) builds the shortest suffix θ'' of the trace θ generated by the function *untimed_verification*. θ'' is built such that the timing analysis shows a timing-inconsistency with the delays δ^l and δ^u imposed by A. A causal event structure CS is constructed by using the causal relations of the events in θ'' (see Section 4.4).

Function *timing_consistent* performs timing analysis over CS. It implements the algorithm described in [MD92] for timing analysis over an acyclic graph of events with min/max and linear constraints (see Section 4.4.1).

If the timing analysis shows that the trace is not timing-consistent, function *compute_lazy_arcs* extracts a set of relative timing constraints from CS, *i.e.* a set of additional orderings between the events of θ'' imposed by the delay bounds. These new constraints are added to the initial CS in the form of lazy arcs by the function *add_lazy_arcs*. The resulting lazy event structure LCS models only those orderings of the events of θ'' which are timing-consistent with the delays imposed by A.

EXAMPLE 4.4 *Recall the example developed in Section 4.2. Consider the TS of Figure 4.2 (a) and the delay bounds specified in Figure 4.2 (b). Recall also that, in this example, the property being verified states that event* g *must always fire before event* d*. Thus, the following trace* $\{x\} \xrightarrow{x} \{a, b\} \xrightarrow{a} \{b, c, g\} \xrightarrow{c} \{b, g\} \xrightarrow{b} \{d, g\} \xrightarrow{d} \{g\} \xrightarrow{g} \{y\}$ *illustrates a violation of the property.*

The shortest possible suffix is given by the trace $\{d, g\} \xrightarrow{d} \{g\} \xrightarrow{g} \{y\}$*, from which the simple event structure of Figure 4.17 (a) is derived. Clearly, the timing analysis cannot be exact since* g *was already enabled in the pre-history of the trace. Thus, the lower delay bound of* g *is conservatively set to* 0*. The timing analysis can conclude nothing about the occurrence order of events* d *and* g*, since both can fire concurrently. The algorithm continues by moving one step backwards along the trace and repeats the same process again, building the corresponding CES incrementally. Figure 4.17 depicts the three attempts needed to find the shortest sufficient suffix of the original failure trace.*

According to the causality relations extracted from the suffix of Figure 4.17 (c), timing analysis concludes that events b *and* g *occur before event* c *(and consequently before* d*). This relative timing constraints are depicted by the dotted arcs in the corresponding lazy event structure. The derived timing relations demonstrate the infeasibility of the given failure trace in the timed domain.* ■ 4.4

We have illustrated the process of deriving a sufficient LzCES that proves the timing-inconsistency of a trace, by considering its shorter suffix. This, however, does not guarantee the maximum effectiveness of the later refinement of the state space of the system, with the timing constraints in the LzCES. In some cases, using the shortest prefix of the trace results in better pruning of the set of failure traces. Similarly, the removal from the LzCES of timing-unrelated concurrent events, or the addition of causal predecessors that improve the knowledge of the enabling prehistory of the events, may affect the quality of the LzCES obtained.

A set of trade-offs must be considered, which correlate aspects such as: how many events are added to the CES such that the timing analysis can still be carried out effectively; how readable will be the resulting LzCES so that it can be useful for back-annotation; how effective in removing failure traces will be the later enabling-compatible product with the system, etc.

4.6.3 Incorporation of relative timing constraints

Finally, we develop the composition algorithm (the *compose* function) that implements the enabling-compatible product (see Section 4.5) between A' and the LzCES LCS. The result is a new LzTS A'' in which all traces contradicting the timing orderings of the events in LCS have been removed from A'. Therefore $\mathcal{L}(A'') \subseteq \mathcal{L}(A')$. The resulting system A'' is a new LzTS where:

- The state space may be split in two parts: one following the enabling orders (enabling-compatible) of the events in LCS, and the other one where the enablings are not followed. The former corresponds to the state subspace where the constraints imposed by LCS apply (the timed subspace). In the latter, LCS does not apply (the untimed subspace).

- In the timed subspace, some events are prevented to fire when they are enabled. More precisely, the composition with LCS allows only those firing orderings which are consistent with the timing analysis.

4.6.4 Back-annotation

A nice feature of the verification approach is its back-annotation capability. Namely, the LzCESs used to represent the timing constraints applied along the series of iterative refinements of the state space of the system under verification, are reported at each iteration of the process.

Given the causality relations modeled by a LzCES and the delays of the events, each LzCES contains a set of additional ordering relations between the events in the timed domain. They provide a set of sufficient conditions for the system under verification to be correct. Moreover, the relative timing nature of such ordering relations and the fact that a CES often contains a small set of events, make the information contained in the LzCES rather easy to interpret.

As a result, the verification approach, not only verifies the correctness of a timed system with respect to a set of safety properties. In case the system does not satisfy the properties, a timed trace showing the sequence of events that lead to a failure and their firing times according to the delays of the system, is provided as counterexample to prove the system malfunction. Otherwise, if the system is correct, a set of timing constraints that prove such correctness is provided in the form of LzCESs.

All this back-annotation information may result crucial in design frameworks where synthesis and verification are invoked iteratively, for the design of systems that must meet functional and non-functional constraints.

4.6.5 Correctness

The correctness of the *timed_verification* algorithm is guaranteed by the following facts:

- The language of the TTS being verified is a subset of the language of the initial untimed abstraction, *i.e.* its underlying TS. This condition is proved by Lemma 4.1.

- Conservativeness: the *compose* function does not remove any trace which is timing-consistent with the delays δ^l and δ^u of the verified TTS. This is guaranteed by the composition rules of the enabling-compatible product (see Section 4.5).

- Convergence: for a particular class of systems the verification requires only few refinements to converge (more details in Section 4.6.6). For the general class of systems a pre-defined upper bound on the number of refinements can be imposed. Although this could produce false negatives during verification, it is in full correspondence to the conservative nature of the suggested verification approach. However, in most practical cases, those systems where the upper bound on the number of refinements is required, are systems which untimed state space is indeed too big to be handled by conventional symbolic techniques.

4.6.6 Convergence

Each composition step of the original LzTS A' with the lazy event structure LCS implicitly performs an unfolding of A' separating traces that are enabling-compatible with LCS and those which are not.

The convergence of the refinement procedure for the class of Marked Graphs is guaranteed by the known results on termination of separation times analysis in a finite number of unfolding iterations [HB94]. Nevertheless the upper bound on the number of iterations could be quite high (depends on the ratio of critical and sub-critical cycles). This is an inherent limitation of exact separation analysis and, for practical applications, it is better to work with pre-established separation bounds and do not unfold beyond those bounds. Although it gives only conservative verification, an acceptance of pre-defined upper bounds seems to be a reasonable option because the largest class of systems for which the separation times analysis could be performed exactly are free and unique choice systems [HB94]. Beyond them the calculation of separation times is inherently conservative.

However there is an important practical class of systems for which the refinement procedure is especially simple and is exact for few unfolding iterations. The characterization of this class is done in terms of the so-called *nodal states*.

DEFINITION 4.14 (NODAL STATE)

Let $A = \langle S, \Sigma, T, \mathsf{s}_0 \rangle$ *be a* TS. *A state* $\mathsf{s} \in S$ *is called* nodal *if* $\forall \mathsf{s}' \in S,\ \mathsf{s}' \xrightarrow{\mathsf{e}'} \mathsf{s} \in T,\ \mathsf{e} \in \mathcal{E}(\mathsf{s}) \Rightarrow \mathsf{e} \notin \mathcal{E}(\mathsf{s}')$.

■ 4.14

Definition 4.14 points that all direct predecessors of a nodal state are synchronized in that state, *i.e.* at the moment when a system arrives to a nodal state all concurrent activities have been finished. Figure 4.18 illustrates the concept by showing two portions of a transition system. State s in the left portion is a nodal state since all the events enabled in s (c and d) are not enabled in any of the precedessor states of s. Conversely, in the right portion state s is not nodal since event e is not newly enabled in s. Other examples of nodal states can be found, for example, in the TS of Figure 4.2, where states

Figure 4.18 Example of a nodal (left) and a not nodal state (right).

s_0, s_1 and s_{13} are nodal. This TS has no conflicting events (no choice) and therefore each of the nodal states is a "global synchronizer" because it breaks all the TS cycles.

Nodal states are natural points from which the timing analysis is convenient to start. Any event enabled somewhere in a path to a nodal state must fire before reaching this state and, hence, timing analysis from a nodal state does not depend on the prehistory of the process behavior. We will call a TS in which every trace passes through at least one nodal state as *strongly synchronized*. Note that the requirement of breaking traces by a *set* of nodal states is essential here because it is easy to construct an example of TS with choices, in which different branches of a choice would have different nodal states and none of them could serve as a "global synchronizer" for the whole TS.

In a strongly synchronized TS, given a failure trace θ with an "improper" ordering of the pair of events a and c, checking the timing-consistency by a and c might be reduced to consideration of the suffix θ_t starting from the nodal state closest to the enabling of events a and c.

By θ_t one can construct the corresponding CES to check whether a and c might occur in the order they have in θ. However in case of cyclic behavior, θ might continue in such a way that the first n occurrences of events a and c satisfy the checked properties while their $n+1$ occurrences have an "improper" ordering. The nice feature of strongly synchronized TSs is that timing analysis made for trace θ can be equally applied for "later" occurrences of a and c because the analysis, started at a nodal state, does not depend on the enabledness prehistory of the events. Therefore timing-inconsistency of θ implies also timing-inconsistency for any cyclic unfolding of θ, from which it immediately follows the exactness and convergence of the suggested procedure for verification.

The practical significance of the class of strongly synchronized TS could be shown by analyzing the known set of asynchronous circuits benchmarks (see Chapter 5): more than 80% of the specifications are strongly synchronized. Beyond the class of strongly synchronized TSs our verification procedure would be conservative in general. Still in many cases it might require just few iterations in unfolding the TS to reach the exact

separation analysis. For example, [AH99] shows the fast convergence of separation times analysis for pipelined specifications, which are inherently not strongly synchronized.

Finally remark that no formal study has been carried out about the convergence of the verification method in the absence of nodal states. Nevertheless, our intuition indicates that the method should generally converge after a bounded number of iterations that guarantee a precise-enough timing analysis. Similar results have been already obtained in the context of marked graphs, where a bounded number of unfoldings suffice to compute the cycle times of a system [NK94]. A detailed formal study on the topic is left for future work.

4.7 Conclusions

This chapter has presented a novel verification methodology for safety properties in timed systems. The methodology combines relative timing with conventional methods based on symbolic reachability analysis. Two fundamental facts are at the basis of the approach: the set of traces of a transition system can be covered by a set of event structures, and the use of relative timing allows to represent the timed domain of a system in an efficient way.

Rather than calculating the exact timed state space, the verification approach performs an *off-line* timing analysis on a set of event structures that covers the traces leading to failure states. This timing analysis is efficiently performed by using McMillan and Dill's algorithm [MD92]. The resulting timing constraints are incorporated to the system in the form of relative timing information along a series of iterative refinements of the original untimed state space. Finally, if some of the traces leading to failure situations cannot be proved to be timing-inconsistent, then the system is incorrect and the failure trace is a counterexample.

The approach presented here, not only verifies the correctness of the system with respect to a set of given safety properties, but also provides as back-annotation a set of timing constraints sufficient to prove correctness. This information is crucial in frameworks in which synthesis and verification are iteratively invoked to design systems that must meet functional and non-functional constraints.

The key features of the verification approach can be summarized by:

- Relative timing allows to avoid the computation of the exact timed state space of the system. Instead, the timed behavior of events is captured by means of partial orders that represent simple facts, such as if an event happens before another.

- The timing analysis is performed locally for a set of failure traces that are covered by an event structure. Therefore, only a subset of the events is involved and the timing analysis can be carried out efficiently.

- Because of the iterative nature of the approach, timing information is only considered in an *on-demand* basis, as long as it is required to prove the infeasibility in the timed domain of a set of failure traces.
- The verification not only proves or disproves the correctness of the system with respect to a set of safety properties. In case the system is correct the algorithm provides the set of relative timing relations used for the proof, which can be used as valuable back-annotation information. In case the system is incorrect, a counterexample failure trace is provided.

Several issues remain open for future developments of the proposed verification approach. Among others:

- Although BDDs are a good data structure for the representation of symbolic boolean information, they often suffer from a memory blow-up during the intermediate computations, thus limiting the applicability of certain algorithms. Therefore, it would be desirable to experiment with other data structures which provide similar benefits than BDDs and allow better manipulation of bigger sets of states.
- Similarly, in order to reduce the memory requirements during the verification of big systems, partial order techniques [GW91, Pel96, VdJL96, ABH+97, BJLY98] could be combined with symbolic methods for state space representation and exploration.
- Incorporate symbolic algorithms for timing analysis (*e.g.* [AH99]), such that actual delay values are not required for verification. Instead, the verification can be tuned to discover the appropriate delays that make a system correct for a given property.
- CESs can model only conjunctive causality relations. However, the causality relations in a TS can be more general, involving disjunctive causality or combinations of both. As a consequence, our approach may need several refinements in order to cover the different causality relations among a set of events. Therefore, it would be desirable to allow the CESs to incorporate other types of causality relations. This would require to review the notions of enabling-compatibility, the way timing analysis is carried out in a CES, the enabling-compatible product, etc.
- Another interesting feature to enrich the verification approach would be the possibility to quantify the effectiveness of an enabling-compatible product before actually performing it. This would allow to choose the best LzCESs at each iteration, so that the biggest number of failure traces are pruned, or the least possible state splitting is produced, etc.
- The back-annotation produced by the tool consists of a set of LzCESs that contain the relative timing constraints used along the verification process. Some of those

constraints may appear several times in different iterations, thus being redundant. Therefore, it would be desirable to have a mechanism to summarize the set of timing constraints and provide them in a more readable form to the user of the tool.