Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Electronic Notes in Theoretical Computer Science 104 (2004) 81–97
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Timing in the Cospan-Span Model

A. Cherubini[1,2]

Dipartimento di Matematica
Politecnico di Milano
Milano, Italy

N. Sabadini,[1,3] R.F.C. Walters[1,4]

Dip. Scienze CC. FF. MM.
Università dell'Insubria
Como, Italy

Abstract

The compositional cospan-span model of concurrent systems introduced in [8],[10] has been shown to model cleanly a variety of phenomena from the original motivation in concurrency, to circuits [9], hierarchy [12], mobility [10], and coordination [10]. The purpose of this paper is to investigate further the expressivity of the cospan-span model, in particular for the modelling of discrete real-time systems. After reviewing the model, explaining its real-time interpretation and discussing some canonical examples, we develop in detail a case study – the specification and verification of a level crossing. In this development similarities with the timed-automata model of Alur and Dill [1] emerge.

Keywords: Cospan, span, timed automata, compositional

[1] The research has been financially supported by the Dipartimento di Scienze Chimiche, Fisiche e Matematiche of the University of Insubria, Como, Italy, and by the Italian Progetti Cofinanziati MIUR: *Computational Metamodels* (COMETA), and *Linguaggi Formali e Automi: teoria e applicazioni.*
[2] Email: aleche@mate.polimi.it
[3] Email: nicoletta.sabadini@uninsubria.it
[4] Email: robert.walters@uninsubria.it

doi:10.1016/j.entcs.2004.08.020

1 Introduction.

The compositional cospan-span model of concurrent systems introduced in [8], [10] has been shown to model cleanly a variety of phenomena from the original motivation in concurrency, to circuits [9], hierarchy [12], mobility [10], and coordination [10]. The elements of the model are cospan of spans of graphs - which in this paper we shall call simply automata. These automata have states and transitions, as well as interfaces and conditions. There are two classes of operations on these automata - parallel (or product) and sequential (or sum) operations - hence yielding an algebra of automata. A system is an expression (or even a recursive equation) in this algebra.

The purpose of this paper is to investigate further the expressivity of the cospan-span model, in particular for the modelling of discrete real-time systems. In fact there is no need to change the mathematics in order to model time constraints, just the need to consider a *new interpretation* of the cospan-span model, which differs essentially from that used in earlier papers. Whereas previously we have used these automata to model concurrent systems in which the timing of transitions was imprecise, in this interpretation every transition is supposed to have a *fixed unit duration.* That is, model of time we use is that there is a *fixed atomic time interval* and that all behaviours have a duration which is a multiple of this fixed unit of time. In [7] the interpretation of parallel composition was that components *synchronize* on common atomic actions, whereas here the interpretation is that time passes and components share values on common interfaces.

There is however a feature of the cospan-span model which assumes a new importance with this interpretation. In practical examples the atomic time interval may need to be taken as very brief, and transitions of this duration may not have conceptual significance, being instead only parts of a higher level action. A top-down description clearly needs to consider actions that have an extended, even variable, duration. The general notion of *action* therefore is modelled not by a single transition but by a summand in an expression, what we call a *non-atomic action*, which has a set of initial states and of final states as well as internal states so the completion of an action does not necessarily require only a unit time interval. In addition we may consider examples in which two automata synchronize on a non-atomic action - a protocol - rather than on single transitions. An extreme case of this is that components do not synchronize, but are truly asynchronous.

In section 2 we review the model, introducing in particular as a special component a counter, and also explaining in section 2.3 how a transition may be may be regarded as a summand of an expression, suggesting the definition of non-atomic action as summand. In section 3 we explain the real-time in-

terpretation of the cospan-span model, and discuss some key examples, and we develop in detail a case study – the specification and verification of a level crossing - a variation of the train crossing problem [2],[3],[4]. We describe the system top-down with respect to the parallel operations, and verify mathematically certain temporal properties. It is important to note that this top-down description is not informal but is a feature of the algebraic formalism we introduce. The sequential aspect of the example is not sufficiently complicated to justify a top-down description with respect to the sequential operations: it is our intention in a later paper to analyse a more elaborate case study making clear the importance of the non-atomic actions.

In this development similarities with the timed-automata model of Alur and Dill [1] emerge, and we discuss them in section 4. In modelling the control program of the train a particular component is fundamental – a counter which may be set and reset, and which may be used to make control decisions about timing. Alur and Dill base their formalism on counters: they introduce finite state automata together with a finite set of clocks which record the passage of continuous time. The clocks can be reset with any state transition of the automaton. In our view there is some confusion in Alur and Dill between the modelling of a system and the specification of desired properties of the system. One possible advantage of our formalism is that the basic elements are mathematically very pure and simple, we have well-based mathematical operations for composing automata, and a strong connection between this algebra and a geometric representation of systems composed of components.

2 Cospans of spans of graphs

By a *graph* $\mathbf{G}$ we mean a directed multigraph, that is, a set $vert(\mathbf{G})$ of vertices and a set $arc(\mathbf{G})$ of (directed) arcs, together with two functions d_0, $d_1 : arc(\mathbf{G}) \to vert(\mathbf{G})$ which specify the source and target, respectively, of each arc. The algebra we use to model timed systems has as elements certain "cospans of spans of graphs", which we will call for simplicity just *automata*.

Definition 2.1 An *automaton* $\mathcal{G}$ consists of a graph $\mathbf{G}$, four sets X, Y, A, B and four functions

$$\partial_0 : arc(\mathbf{G}) \to X,\ \partial_1 : arc(\mathbf{G}) \to Y,$$
$$\gamma_0 : A \to vert(\mathbf{G}),\ \gamma_1 : B \to vert(\mathbf{G}).$$

A *behaviour* of $\mathcal{G}$ is a path in the graph $\mathbf{G}$. The *duration* of a behaviour is the number of steps in the path.

The graph $\mathbf{G}$ is called the *centre* of the automaton. We remark that ∂_0,

∂_1 may be thought of as *labellings* of the arcs of $\mathbf{G}$ in the alphabets X, Y, respectively. If the left label of a transition is x and the right y we denote the combined labelling by x/y. The labellings will be used in the restricted product of two automata with interface, the operation which expresses communicating parallel processes. Alternatively, one may think of the vertices and arcs of $\mathbf{G}$ as the *states* and *transitions* of the system, whereas the elements of X, Y are the transitions of the interfaces. We call X the *left interface* of the automaton, and Y the *right interface* - automata communicate through these interfaces. Often the interface sets will be products of sets; for example the left interface of $\mathbf{G}$ may be $X = U \times V$, and the right interface may be $Y = Z \times W$, and we then speak of U and V as the left interfaces, and Z and W as the right interfaces. If we ignore the functions γ_0, γ_1 a automaton is a (particular type of) *span of graphs* as defined in [8], where the reader may find more details and examples.

The set A represents a *condition* on the states in which the automaton may come into existence, and the set B a condition in which it may cease to exist. We call A the *in-condition* of the automaton, and B the *out-condition*. The functions γ_0, γ_1 of a automaton will be used in the restricted sum of automata - a generalized sequential operation. Often the condition sets will be sums of sets; for example the in-condition may be $A = D + E$, and we then speak of D and E as in-conditions.

There is a useful graphical representation of automata and their operations which we use in the paper, and which is described in [10].

For simplicity we use the same names ∂_0, ∂_1, γ_0, γ_1 for the four functions of any automaton where there is no risk of confusion, introducing further suffixes when clarification is needed. We use symbols $X, Y, Z, U, V, W...$ for the (left and right) interfaces, and symbols $A, B, C, D, E, F, I, ...$ for the (in- and out-) conditions.

Remark 2.2 For modelling systems with parts which have an idling action on an interface we frequently require that an interface contains a special element, the null symbol ε, which represents the *null transition* on the interface.

2.1 Parallel or product operations

Definition 2.3 Given two automata

$$\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1),\ \mathcal{H} = (\mathbf{H}, Y, Z, C, D, \partial_0, \partial_1, \gamma_0, \gamma_1)$$

the *restricted product* (communicating parallel composition) of $\mathcal{G}$ and $\mathcal{H}$, denoted $\mathcal{G} \cdot \mathcal{H}$ is the automaton whose set of vertices is $vert(\mathbf{G}) \times vert(\mathbf{H})$ and whose set of arcs is that subset of $arc(\mathbf{G}) \times arc(\mathbf{H})$ consisting of pairs of arcs

(g, h) such that $\partial_1(g) = \partial_0(h)$. The interfaces and conditions of $\mathcal{G} \cdot \mathcal{H}$ are $X, Z, A \times C, B \times D$, and the four functions are

$$\begin{aligned} \partial_{0,\mathcal{G}\cdot\mathcal{H}}(g,h) &= \partial_{0,\mathcal{G}}(g),\ \partial_{1,\mathcal{G}\cdot\mathcal{H}}(g,h) = \partial_{1,\mathcal{H}}(h), \\ \gamma_{0,\mathcal{G}\cdot\mathcal{H}} &= \gamma_{0,\mathcal{G}} \times \gamma_{0,\mathcal{H}}\ ,\ \gamma_{1,\mathcal{G}\cdot\mathcal{H}} = \gamma_{1,\mathcal{G}} \times \gamma_{1,\mathcal{H}}\ . \end{aligned}$$

Closely related to the restricted product is the free product of automata.

Definition 2.4 Given two automata

$$\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1),\ \mathcal{H} = (\mathbf{H}, Z, W, C, D, \partial_0, \partial_1, \gamma_0, \gamma_1)$$

the *free product* (parallel composition with no communication) of $\mathcal{G}$ and $\mathcal{H}$, denoted $\mathcal{G} \times \mathcal{H}$ is the automaton whose set of vertices is $vert(\mathbf{G}) \times vert(\mathbf{H})$ and whose set of arcs is $arc(\mathbf{G}) \times arc(\mathbf{H})$. The interfaces and conditions of $\mathcal{G} \times \mathcal{H}$ are $X \times Z, Y \times W, A \times C, B \times D$, and the four functions are

$$\begin{aligned} \partial_{0,\mathcal{G}\times\mathcal{H}} &= \partial_{0,\mathcal{G}} \times \partial_{0,\mathcal{H}},\ \partial_{1,\mathcal{G}\times\mathcal{H}} = \partial_{1,\mathcal{G}} \times \partial_{1,\mathcal{H}}, \\ \gamma_{0,\mathcal{G}\times\mathcal{H}} &= \gamma_{0,\mathcal{G}} \times \gamma_{0,\mathcal{H}}\ ,\ \gamma_{1,\mathcal{G}\times\mathcal{H}} = \gamma_{1,\mathcal{G}} \times \gamma_{1,\mathcal{H}}\ . \end{aligned}$$

Definition 2.5 Given an automaton

$$\mathcal{G} = (\mathbf{G}, X \times Y, Z \times Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1),$$

the *place feedback of* $\mathcal{G}$ with respect to Y, denoted $\mathsf{Pfb}_Y(\mathcal{G})$ is the automaton whose set of vertices is $vert(\mathbf{G})$ and whose set of arcs is that subset of $arc(\mathbf{G})$ consisting of arcs g such that $(pr_Y \circ \partial_1)(g) = (pr_Y \circ \partial_0)(g)$. The interfaces and conditions of $\mathsf{Pfb}_Y(\mathcal{G})$ are X, Z, A, B,with the four functions defined as follows:

$$\begin{aligned} \partial_{0,\mathsf{Pfb}_Y(\mathcal{G})} &= pr_X \circ \partial_{0,\mathcal{G}},\ \partial_{1,\mathsf{Pfb}_Y(\mathcal{G})} = pr_Z \circ \partial_{1,\mathcal{G}}, \\ \gamma_{0,\mathsf{Pfb}_Y(\mathcal{G})} &= \gamma_{0,\mathcal{G}},\ \ \gamma_{1,\mathsf{Pfb}_Y(\mathcal{G})} = \gamma_{1,\mathcal{G}}. \end{aligned}$$

The diagrammatic representation of $\mathsf{Pfb}_Y(\mathcal{G})$ involves joining the right interface Y to the left interface Y.

If we ignore the functions γ_0, γ_1 of the automata the restricted product of automata is the *span composition* of [8] and the free product is the *tensor product* of the corresponding spans of graphs. For some examples of how these operations may be used to model concurrent systems see that paper. From a circuit theory point of view these operations correspond, respectively, to the series and parallel operations of circuit components.

2.2 Sequential or sum operations

Definition 2.6 Given two automata

$$\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1),\ \mathcal{H} = (\mathbf{H}, X, Y, B, C, \partial_0, \partial_1, \gamma_0, \gamma_1)$$

the *restricted sum* (sequential composite) of $\mathcal{G}$ and $\mathcal{H}$, denoted $\mathcal{G} + \mathcal{H}$ is the automaton whose set of arcs is $arc(\mathbf{G}) + arc(\mathbf{H})$ and whose set of vertices is $(vert(\mathbf{G}) + vert(\mathbf{H}))/\sim$; that is $(vert(\mathbf{G}) + vert(\mathbf{H}))$ quotiented by the relation $\gamma_{1,\mathcal{G}}(b) \sim \gamma_{0,\mathcal{H}}(b)$ (for all $b \in B$). The interfaces and conditions of $\mathcal{G} + \mathcal{H}$ are X, Y, A and C, and the four functions are

$$\begin{aligned} \partial_{0,\mathcal{G}+\mathcal{H}} &= (\partial_{0,\mathcal{G}} \,|\, \partial_{0,\mathcal{H}}),\ \partial_{1,\mathcal{G}+\mathcal{H}} = (\partial_{1,\mathcal{G}} \,|\, \partial_{1,\mathcal{H}}), \\ \gamma_{0,\mathcal{G}+\mathcal{H}} &= in_{vert(\mathbf{G})} \circ \gamma_{0,\mathcal{G}},\ \ \gamma_{1,\mathcal{G}+\mathcal{H}} = in_{vert(\mathbf{H})} \circ \gamma_{1,\mathcal{H}}\,. \end{aligned}$$

The diagrammatic representation of the restricted sum is as follows:

Definition 2.7 Given two automata

$$\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1),\ \mathcal{H} = (\mathbf{H}, X, Y, C, D, \partial_0, \partial_1, \gamma_0, \gamma_1)$$

the *unrestricted sum* of $\mathcal{G}$ and $\mathcal{H}$, denoted $\mathcal{G} \oplus \mathcal{H}$ is the automaton whose set of arcs is $arc(\mathbf{G}) + arc(\mathbf{H})$ and whose set of vertices is $vert(\mathbf{G}) + vert(\mathbf{H}))$. The interfaces and conditions of $\mathcal{G} \oplus \mathcal{H}$ are $X, Y, A + C$ and $B + D$, and the four functions are

$$\begin{aligned} \partial_{0,\mathcal{G}\oplus\mathcal{H}} &= (\partial_{0,\mathcal{G}} \,|\, \partial_{0,\mathcal{H}}),\ \partial_{1,\mathcal{G}\oplus\mathcal{H}} = (\partial_{1,\mathcal{G}} \,|\, \partial_{1,\mathcal{H}}), \\ \gamma_{0,\mathcal{G}\oplus\mathcal{H}} &= \gamma_{0,\mathcal{G}} + \gamma_{0,\mathcal{H}},\ \ \gamma_{1,\mathcal{G}\oplus\mathcal{H}} = \gamma_{1,\mathcal{G}} + \gamma_{1,\mathcal{H}}. \end{aligned}$$

Definition 2.8 Given an automaton

$$\mathcal{G} = (\mathbf{G}, X, Y, A + B, C + B, \partial_0, \partial_1, \gamma_0, \gamma_1),$$

the *sequential feedback* of $\mathcal{G}$ with respect to $\mathcal{B}$, denoted $\mathsf{Sfb}_B(\mathcal{G})$ is the automaton whose set of arcs is $arc(\mathbf{G})$ and whose set of vertices is $vert(\mathbf{G})/\sim$; that is $vert(\mathbf{G})$ quotiented by the relation $(\gamma_1 \circ in_B)(b) \sim (\gamma_0 \circ in_B)(b)$ (for all $b \in B$). The interfaces and conditions of $\mathsf{Sfb}_B(\mathcal{G})$ are X, Y, A and C, and the four functions are defined as follows:

$$\begin{aligned} \partial_{0,\mathsf{Sfb}_B(\mathcal{G})} &= \partial_{0,\mathcal{G}},\ \partial_{1,\mathsf{Sfb}_B(\mathcal{G})} = \partial_{1,\mathcal{G}}, \\ \gamma_{0,\mathsf{Sfb}_B(\mathcal{G})} &= \gamma_{0,\mathcal{G}} \circ in_A,\ \ \gamma_{1,\mathsf{Sfb}_B(\mathcal{G})} = \gamma_{1,\mathcal{G}} \circ in_C. \end{aligned}$$

The diagrammatic representation of $\mathsf{Sfb}_Y(\mathcal{G})$ involves joining the incondition B to the outcondition B.

Definition 2.9 Given a set X we define two special automata as follows (in and out conditions may be arbitrary sets):

(i) the *identity automata* $\mathbf{1}_X$ with left and right interface X has one state, and for each $x \in X$ an edge labelled on both the left and right by x. It is denoted diagrammatically by a wire.

(ii) the *diagonal* automaton Δ_X with left interface X and right interface $X \times X$ has one state and to each $x \in X$ an edge labelled x on the left and x, x on the right. It is denoted diagrammatically by a fan out.

The diagonal can be used to model broadcast communication.

Remark 2.10 It is clear that by combining the operations we have described above systems with a complex geometry may be produced. It is often convenient in sketching systems not to place all the input (output) interfaces on the left (right) of components but to distinguish them by the directions of the arrows (although it is clear one could always rearrange the system to maintain the left/right input/output convention). In some cases an interface may be neither clearly input or output in which case we omit the arrow.

Example 2.11 A *counter* of capacity n (an n-counter) is an automaton with two interfaces $X = \{\varepsilon, set_0, set_1, set_2, set_3, ..., set_n\}$ and $Y = \{\varepsilon, alarm\}$ and n states $0, 1, 2, 3, ..., n$. From 0 to 0 there is a transition labelled ε/ε; from j to $j-1$ there is a transition labelled ε/ε $(j = 2, 3, ..., n)$. From any j to k there is a transition labelled set_k/ε; from 1 to 0 there is a transition labelled $\varepsilon/alarm$. The idea is that in any state the counter may be set, with null input the counter decrements until at the transition 1 to 0 the counter sends an alarm. The action set_0 may also be called *reset*.

2.3 Generators for the algebra of finite automata with interfaces X, Y

If we fix left and right interfaces X, Y and consider those automata

$$\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1)$$

with $arc(\mathbf{G}) = \emptyset$ we see that in this case an automaton amounts just to a pair of functions $\gamma_0 : A \to vert(\mathbf{G})$, $\gamma_1 : B \to vert(\mathbf{G})$; that is, a cospan of sets. Further automata compose (under the restricted sum) like cospans [11]. Even more particularly amongst the cospan there are those with $\gamma_1 = 1_B$ which amount to functions from A to B and which compose under the restricted sum as functions. Dually there are those with $\gamma_0 = 1_A$ which might be called the opposite of functions and compose as such.

Note that every cospan of sets may be constructed from functions and cofunctions between the objects $\mathbf{0}$ (empty set), $\mathbf{1}$ (one element set) and $\mathbf{2} = \mathbf{1} + \mathbf{1}$

using the sequential operations above. The unique function $\mathbf{2} \to \mathbf{1}$ forms the basis for constructing surjective functions; the transposition $\mathbf{2} \to \mathbf{2}$ forms the basis for constructing bijections; the function $\mathbf{0} \to \mathbf{1}$ may be used to construct injections.

For $x \in X, y \in Y$ consider the automaton $arr_{x,y}$, the automata with two states 0 and 1 and a single transition from 0 to 1 labelled x/y, and with incondition 0, outcondition 1. Any finite automata $\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1)$ may be constructed from the functions and cofunctions between the objects $\mathbf{0}, \mathbf{1}$ and $\mathbf{2}$, together with the automata $arr_{x,y}$ $(x \in X,\ y \in Y)$ using only the sequential operations. The formula is $\mathcal{G} = \mathsf{Sfb}_{vert(\mathbf{G})}(dom^0 + \oplus(arr_{x,y}) + cod)$, where dom^0 is the opposite of the source function $arc(\mathbf{G}) \to vert(\mathbf{G})$, cod is the target function $arc(\mathbf{G}) \to vert(\mathbf{G})$, and the unrestricted sum is over all transitions in $\mathbf{G}$. The consequence is that we could write all automata as expressions in a few constants, one advantage of this syntactical view being that we can do top-down analysis, and make recursive definitions [10]. (A more precise statement of these remarks to be reported by the authors and R. Rosebrugh in a further publication is that cospans form the generic symmetric monoidal category with a particular type of bimonoid object.)

3 Discrete-time systems

Definition 3.1 A *discrete-time system* is an expression of automata, using the parallel and sequential operations of automata, and the constants identity, diagonal, and the generating constants described in section 2.3. A *behaviour* of a system is a path in the automata resulting from evaluating the expression. The *duration* of a behaviour is length of the path.

Definition 3.2 Suppose $\mathcal{G}$ is given as an expression of automata involving only sequential operations, and $\mathcal{H}$ is one of the operands in the expression, then we call $\mathcal{H}$ a *non-atomic action* of $\mathcal{G}$. A behaviour of a non-atomic action consists of a path from an incondition to an outcondition. The duration of a non-atomic action is hence variable.

As explained above a discrete-time system has a geometry. In the remainder of this section we give some simple examples illustrating the expressiveness of the model. In all of the examples the inconditions and outconditions will be the evident single initial and final states of the actions under consideration. Clearly single transitions are special examples of non-atomic actions - which therefore should more precisely be called not-necessarily-atomic actions! The next example shows that actions may be strongly non-atomic.

Example 3.3 (two actions which synchronize with an arbitrary duration)

Consider two automata $\mathcal{G}_1, \mathcal{G}_2$ and two non-atomic actions, one of $\mathcal{G}_1$, one of $\mathcal{G}_2$. The action of $\mathcal{G}_1$ is $\{a/b : 0 \to 1, \varepsilon/\varepsilon : 1 \to 1, d/e : 1 \to 2\}$; the action of $\mathcal{G}_2$ is $\{b/c : 0 \to 1, \varepsilon/\varepsilon : 1 \to 1, e/f : 1 \to 2\}$. In the restricted product $\mathcal{G}_1 \cdot \mathcal{G}_2$ these actions synchronize but with arbitrary duration. A typical behaviour is $(0,0) - a/c \to (1,1) - \varepsilon/\varepsilon \to (1,1) - \varepsilon/\varepsilon \to \cdots - \varepsilon/\varepsilon \to (1,1) - d/f \to (2,2)$.

There is a construction of the *desynchronization* of a system based on this idea. Consider a system which is given as an expression in parallel operations of automata. Suppose every transition in each component automaton of the system is replaced by an action consisting of three transitions analogous to the actions in this example then the components of the system synchronize with arbitrary duration instead of duration 1. As an example consider the dining philosopher problem as described in [9]. The desynchronized version in which each of the philosopher and fork actions are divided into begin-action and end-action has the property that the duration of taking a fork may be arbitrarily long, and hence while one philosopher is taking a fork another may eat several meals.

Example 3.4 (a non-atomic action of duration between 1 and 3, which synchronizes with one of duration between 2 and 4) Consider two automata $\mathcal{G}_1, \mathcal{G}_2$ and two non-atomic actions, one of $\mathcal{G}_1$, one of $\mathcal{G}_2$. The action of $\mathcal{G}_1$ has initial state 0 and final state 1 and transitions $\{a_{1,1}/b_{1,1} : 0 \to 1, a_{2,1}/b_{2,1} : 0 \to 2, a_{2,2}/b_{2,2} : 2 \to 1, a_{3,1}/b_{3,1} : 0 \to 3, a_{3,2}/b_{3,2} : 3 \to 4, a_{3,3}/b_{3,3} : 4 \to 1\}$; the action of $\mathcal{G}_2$ has initial state 0 and final state 1 and transitions $\{b_{2,1}/c_{2,1} : 0 \to 2, b_{2,2}/c_{2,2} : 2 \to 1, b_{3,1}/c_{3,1} : 0 \to 3, b_{3,2}/c_{3,2} : 3 \to 4, b_{3,3}/c_{3,3} : 4 \to 1, b_{4,1}/c_{4,1} : 0 \to 5, b_{4,2}/c_{4,2} : 5 \to 6, b_{4,3}/c_{4,3} : 6 \to 7, b_{4,4}/c_{4,4} : 7 \to 1\}$. In the restricted product $\mathcal{G}_1 \cdot \mathcal{G}_2$ these actions synchronize with duration 2 or 3. The two terminating behaviours possible are $(0,0) - a_{2,1}/c_{2,1} \to (2,2) - a_{2,2}/c_{2,2} \to (1,1)$, and $(0,0) - a_{3,1}/c_{3,1} \to (3,3) - a_{3,2}/c_{3,2} \to (4,4) - a_{3,3}/c_{3,3} \to (1,1)$

These two examples are simple but suggestive examples of processes synchronizing on a protocol.

Example 3.5 A counter as described above provides exactly an action with a settable duration, that is, the input can set the duration of the whole action of the counter.

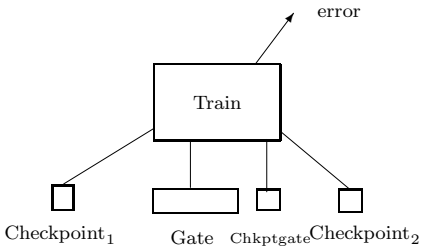
It is clear that with the apparatus of finite state automata and counters, control decisions may be made on the basis of timing. The next section gives a detailed example.

3.1 Modelling of a railway crossing

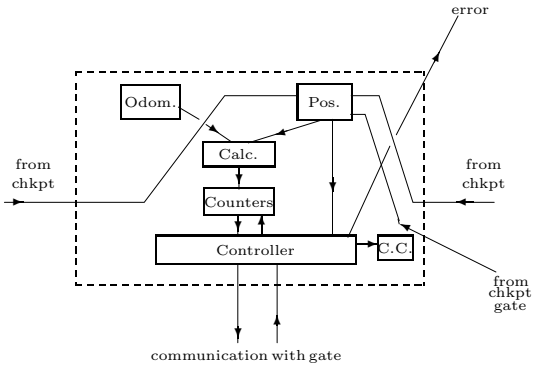
We would like to model a variation of the radio-based decentralized train control system described in [4]. We concentrate on formalizing as precisely as possible the system, using instead informal mathematical language for the description of the desired properties of the system. All automata in this example will have in and outconditions $\emptyset$ - the sum operations will not be used since we are describing the component automata at low level.

We have in mind the following system. A train is approaching a crossing. Before the crossing there is a checkpoint at which a localizing subsystem, Pos., on the train determines the position of the train in the network (using a chart). This information, together with the speed of the train obtained from the odometer, Odom., is used to calculate three alarms (which are placed in corresponding counters and decremented as time passes). The three alarms are inputs to the controller subsystem of the train. The first alarm is sounded when the train must signal the gate to close. The second alarm is sounded at the last moment it is possible for the train to stop before reaching the gate (in the case when the train has not received an acknowledgment that the gate is closed). The train sends a message to the cruise controller (C.C.) to brake the train. The third alarm is a time-out indicating a serious problem with the train or gate (at which an error signal is emitted); it is a bound on the time required by the train to traverse the crossing under normal conditions. The localizing subsystem Pos. emits a signal *gate* when the train passes the gate checkpoint, and a signal *pass* when the train passes the outgoing checkpoint. In the system we are modelling the train has no certain information about the timing of the gate. In case the train receives an acknowledgment from the gate it proceeds to the gate where there is a checkpoint, it proceeds, and on passing the outgoing checkpoint signals the gate to raise. We describe out model in a series of refinements. Notice that each of these descriptions has a mathematical meaning.

First a high level view of the train and track near the gate.



Next we give more detail of the train. (Calc.=calculation of alarms.).



The checkpoints are very simple automata with only an output interface with alphabet$\{\varepsilon, id\}$. They have one state and two transitions, one labelled id, and the other labelled ε. The idea is that a checkpoint is continuously broadcasting its identifier id but that signal only reaches the train when it is within range, and hence the signal read by the train is ε except when the train passes the checkpoint.

Next we describe (an abstraction of) the three counters $Count$. It has one input interface $R = \{\varepsilon, rs_1, rs_2\}$ (resets) and one output interface $A = \{\varepsilon, al_1, al_2, al_3\}$ (alarms). It has states $0, 1, 2, 3$ where 0 is both the state in which all counters are set, 1 is the state in which the first counter is zero, 2 the state in which the first and second counters are zero, and 3 the state in which all are zero. The transitions are

$$\varepsilon/\varepsilon : 0\to 0, \quad \varepsilon/\varepsilon : 1\to 1,$$

$$\varepsilon/\varepsilon : 2\to 2, \quad \varepsilon/\varepsilon : 3\to 3,$$

$$\varepsilon/al_1 : 0\to 1, \ \varepsilon/al_2 : 1\to 2,$$

$$\varepsilon/al_3 : 2\to 3, \ rs_2/\varepsilon : 1\to 2,$$

$$rs_3/\varepsilon : 2\to 3, \ rs_3/\varepsilon : 3\to 3$$

Next we describe the controller $Contr$. The input interfaces are $P = \{\varepsilon, pass, gate\}$, $A = \{\varepsilon, al_1, al_2, al_3\}$, $G_{ack} = \{\varepsilon, ack\}$. The output interfaces are $E = \{\varepsilon, error\}$, $S = \{\varepsilon, start, stop\}$, $R = \{\varepsilon, rs_2, rs_3\}$, $G_{com} = \{\varepsilon, up, down\}$. The states are s_0, s_1, s_2, s_3, s_4, and the transitions are

$$\varepsilon,\varepsilon,\varepsilon/\varepsilon,\varepsilon,\varepsilon,\varepsilon : s_0 \to s_0, \qquad \varepsilon,\varepsilon,\varepsilon/\varepsilon,\varepsilon,\varepsilon,\varepsilon : s_1 \to s_1,$$

$$\varepsilon,\varepsilon,\varepsilon/\varepsilon,\varepsilon,\varepsilon,\varepsilon : s_2 \to s_2, \qquad \varepsilon,\varepsilon,\varepsilon/\varepsilon,\varepsilon,\varepsilon,\varepsilon : s_3 \to s_3,$$

$$\varepsilon,\varepsilon,\varepsilon/\varepsilon,\varepsilon,\varepsilon,\varepsilon : s_4 \to s_4, \qquad \varepsilon,al_1,\varepsilon/\varepsilon,\varepsilon,\varepsilon,down : s_0 \to s_1,$$

$$\varepsilon,al_2,\varepsilon/\varepsilon,stop,\varepsilon,\varepsilon : s_1 \to s_2, \qquad \varepsilon,al_3,e/error,\varepsilon,\varepsilon,\varepsilon : s_2 \to s_2,$$

$$\varepsilon,al_3,\varepsilon/error,\varepsilon,\varepsilon,\varepsilon : s_3 \to s_3, \qquad \varepsilon,al_3,\varepsilon/errore,\varepsilon,\varepsilon,\varepsilon : s_4 \to s_4,$$

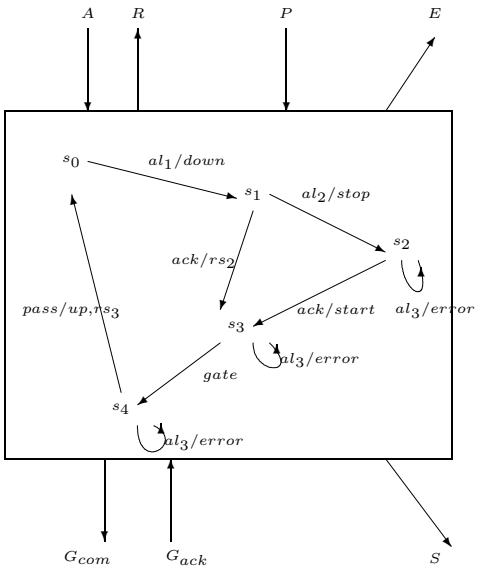
$$\varepsilon,\varepsilon,ack/\varepsilon,\varepsilon,rs_2,\varepsilon : s_1 \to s_3, \qquad \varepsilon,\varepsilon,ack/\varepsilon,start,\varepsilon,\varepsilon : s_2 \to s_3,$$

$$\varepsilon,al_2,ack/\varepsilon,\varepsilon,\varepsilon,\varepsilon : s_1 \to s_3, \qquad \varepsilon,al_3,ack/error,start,\varepsilon,\varepsilon : s_2 \to s_3$$

$$gate,\varepsilon,\varepsilon/\varepsilon,\varepsilon,\varepsilon,\varepsilon : s_3 \to s_4, \qquad gate,al_3,\varepsilon/error,\varepsilon,\varepsilon,\varepsilon : s_3 \to s_4,$$

$$pass,\varepsilon,\varepsilon/\varepsilon,\varepsilon,rs_3,up : s_4 \to s_0.$$

To present the controller in a more readable form we *sketch* the automaton but including *only the principal transitions* and giving *only the principal labels* on each transition.

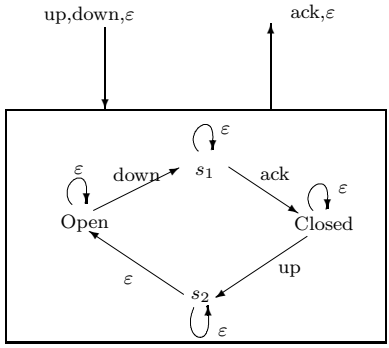


Finally we describe the gate $Gate$. It has input interface $G_{com} = \{\varepsilon, up, down\}$ and output interface $G_{ack} = \{\varepsilon, ack\}$. Its states are $Open, s_1, Closed$ and s_2,

and its transitions are

$$\varepsilon/\varepsilon{:}Open{\to}Open, \qquad \varepsilon/\varepsilon{:}s_1{\to}s_1,$$
$$\varepsilon/\varepsilon{:}Closed{\to}Closed, \qquad \varepsilon/\varepsilon{:}s_2{\to}s_2,$$
$$down/\varepsilon{:}Open{\to}s_1, \quad \varepsilon/ack{:}s_1{\to}Closed,$$
$$up/\varepsilon{:}Closed{\to}s_2, \qquad \varepsilon/\varepsilon{:}s_2{\to}Open.$$

A sketch of the gate, again only giving the principal labels on each transition:



Now the system *Railcrossing* composed of the counters, controller and gate is given by the algebraic expression

$$\mathsf{Pfb}_{R\times G_{ack}}((1_P{\times}Count{\times}1_{G_{ack}}){\cdot}Contr{\cdot}(1_E{\times}1_S{\times}1_R{\times}Gate)),$$

and has input interface P and output interfaces E and S. The reachable part of this composite has 10 states (the first being the initial state)

$$(0,s_0,Open),(1,s_1,s_1),(2,s_2,s_1),(2,s_3,Closed),(3,s_2,s_1),$$
$$(3,s_3,Closed),(2,s_4,Closed),(3,s_4,Closed),(3,s_0,s_2),(3,s_0,Open)$$

and 10 transitions of the form $\varepsilon/\varepsilon, \varepsilon : state \to state$, one for each of the ten

states, *plus* the following 16 transitions:

$$\varepsilon \,/\, \varepsilon,\varepsilon{:}(0,s_0,Open){\rightarrow}(1,s_1,s_1),\varepsilon \,/\, \varepsilon,stop{:}(1,s_1,s_1){\rightarrow}(2,s_2,s_1),$$

$$\varepsilon \,/\, \varepsilon,\varepsilon{:}(1,s_1,s_1){\rightarrow}(2,s_3,Closed),\varepsilon \,/\, \varepsilon,\varepsilon{:}(1,s_1,s_1){\rightarrow}(2,s_3,Closed),$$

$$\varepsilon \,/\, \varepsilon,start{:}(2,s_2,s_1){\rightarrow}(2,s_3,Closed),gate \,/\, \varepsilon,\varepsilon{:}(2,s_3,Closed){\rightarrow}(2,s_4,Closed),$$

$$pass \,/\, \varepsilon,\varepsilon{:}(2,s_4,Closed){\rightarrow}(3,s_0,s_2),\varepsilon \,/\, \varepsilon,\varepsilon{:}(3,s_0,s_2){\rightarrow}(3,s_0,Open),$$

$$\varepsilon \,/\, error,\varepsilon{:}(2,s_2,s_1){\rightarrow}(3,s_2,s_1),\varepsilon \,/\, \varepsilon,start{:}(3,s_2,s_1){\rightarrow}(3,s_3,Closed),$$

$$\varepsilon \,/\, error,start{:}(2,s_2,s_1){\rightarrow}(3,s_3,Closed),\varepsilon \,/\, error,\varepsilon{:}(2,s_3,Closed){\rightarrow}(3,s_3,Closed),$$

$$gate \,/\, \varepsilon,\varepsilon{:}(3,s_3,Closed){\rightarrow}(3,s_4,Closed),gate \,/\, error,\varepsilon{:}(2,s_3,Closed){\rightarrow}(3,s_4,Closed),$$

$$\varepsilon \,/\, error,\varepsilon{:}(2,s_4,Closed){\rightarrow}(3,s_4,Closed),pass/\varepsilon,\varepsilon{:}(3,s_4,Closed){\rightarrow}(3,s_0,s_2).$$

The first 8 transitions (together with the purely idling transitions) constitute the part of the automaton reachable from the initial state $(0, s_0, Open)$ without incurring an error signal. The assumption on the counters is that eventually they reach state 3. Those behaviours which reach a state in which the counters are in state 3 and which avoid an error signal all reach the state $(3, s_0, s_2)$ passing the gate when it is *closed*, and the second checkpoint. There is no guarantee that the state $(3, s_0, Open)$ is reached because there is no signal to that effect from the gate to the controller which generates the error messages.

4 Relation with Timed Automata

In the past decade various automata based models for describing and reasoning about timing-based systems have been proposed, and timed automata introduced by Alur and Dill [1] have received enormous attention because they provide a natural way of expressing timing delays of real-time systems. A timed automata is a finite automata (with Büchi or Muller acceptance conditions) with a finite set of real-valued clocks. All clocks increase at the uniform rate counting time with respect to a fixed global time frame. The clocks can be reset to 0 (independently of each other) with the transitions of the automata, keeping track of the time elapsed since the last reset. The transitions of the automaton contain constraints on the clock values and a transition may be taken only if the current values of the clocks satisfy the associated constraints. The allowed constraints in the seminal paper of Alur

and Dill were Boolean combinations of simpler constraints comparing clock values with time constants. Since then generalizations have been introduced allowing diagonal clock constraints which compare the values of two clocks and constant updates which reset clocks to some constant. Timed automata give a good framework for modeling finite-state systems, namely a timed (Büchi) automaton A_P is associated to a finite-state process P so that $L(A_P)$ consists of the timed traces of P. The state-transition graph of the system is given by the transition table of the automaton, the timing delays of various physical components are represented by means of clocks, and the fairness conditions are expressed by the acceptance conditions. Usually an implementation is described as a composition of several components, each of them is modeled as a timed regular process, and it is possible to construct a timed automaton representing the composite process using a composition of timed automata defined by a suitable modification of the standard product construction for Büchi automata.

Comparing timed automata with our model the first difference to be underlined is the time domain. We use a discrete time domain while clocks in timed automata assume real values (or more precisely dense time). We assume further that each transition has a fixed a priori duration while in timed automata transitions are instantaneous and time can elapse in a state. This choice limits the expressiveness of our model (for instance we cannot recognize the language of Example 3.16 in [1]). However we believe that the assumption of discrete time domain is not a strong restriction with respect to a dense time in real-systems because almost all physical devices are digital, or approximable by discrete time. We refer to [6] for a wider discussion on this topic. A fundamental result in the timed automata theory is the decidability of region reachability, which yields model checking algorithms and tools for verifying real systems which are one of the appealing aspects of timed automata. We remark that the model checking algorithm can be restricted to timed automata whose clock constraints involve only integer constants (Lemma 4.1 in [1]). Moreover in [5] timed automata with discrete time domain and unit duration of transitions were introduced to allow general linear relations over clocks and parameterized constants as clock constraints which are useful in modeling and in analyzing and debugging with approximation techniques complex real-time systems - a fact which supports our idea that the discrete domain is not a strong limitation. This paper has been concerned with exploring the expressivity of the cospan-span model, but in a future paper we will describe algorithms for model-checking and the verification of correctness appropriate for our model.

A possible advantage of the cospan-span model with respect to timed au-

tomata is the structured geometric representation based on three level of abstraction and also a well-established algebra for composition of automata-with-interfaces, based on operations whose definitions are easily understood. Timed automata on the contrary are a low-level representation, even if translations from more structured representations like the real-time process algebra ATP have been presented. The difference is easily remarked referring to the example of an automatic controller that opens and closes a gate at a railroad crossing described by means of timed automaton in Section 7.5 of [1]. This example is a simplified version of the one we gave in section 3: only three components TRAIN,GATE and CONTROLLER form the systems because the example does not concern with how the train knows whether it is approaching or leaving the crossing, we could easily do the same example deleting the checkpoints and simplifying Train Controller and Gate Controller. In the Alur and Dill example the connections among the components cannot be immediately read from the graphical representation (only common labels on the transitions of different automaton represent communications between two components). In the automata representing TRAIN and GATE, specifications and system description are not completely separated, because clock constraints look like implementations. In our model a counter occurs as a subcomponent in the TRAIN model to guarantee that if the train does not enter and exit from the gate in the due time interval the error is detected allowing corrective action to be taken.

There are suggestive similarities between the cospan-span model and Statecharts, and we intend to make a careful comparison in future work.

References

[1] R. Alur and D.L. Dill. A theory of timed automata. Theoretical Computer Science, 126:183–235, 1994.

[2] J. Bohn, W. Damm, H. Wittke, J. Klose, A. Moik, Modeling and validating train system applications using Statemate and Live Sequence Charts, Integrated Design and Process Technology, IDPT-2002, June 2002.

[3] W. Damm, G. Dohmen, J. Klose, Secure decentralized control of railway crossings.

[4] W. Damm, J. Klose, Verification of radio-based signaling system using the Statemate verification environment, Formal methods in system design, 19, 121-141, 2001.

[5] Zhe Dang, O. H. Ibarra, R. A. Kemmerer, Decidable approximations on generalized and parametrized discrete timed automata, COCOON 2001: 529-539, Springer LNCS 2108, 2001.

[6] T.A.Henziger, Z.Manna, A.Pnueli, What good are digital clocks?, ICALP 92, LNCS 623,545,558,1992

[7] P. Katis, N. Sabadini, R.F.C. Walters, Bicategories of processes, Journal of Pure and Applied Algebra 115: 141–178, 1997.

[8] P. Katis, N. Sabadini, R.F.C. Walters, Span(Graph): A categorical algebra of transition systems, Proc. AMAST '97, SLNCS 1349, 307–321, Springer Verlag, 1997.

[9] P. Katis, N. Sabadini, R.F.C. Walters, On the algebra of systems with feedback and boundary, Rendiconti del Circolo Matematico di Palermo Serie II, Suppl. 63: 123-156, 2000.

[10] P. Katis, N. Sabadini, R.F.C. Walters, ; A formalisation of the IWIM Model. in: Proc. COORDINATION 2000,(Eds.) Porto A., Roman G.-C., LNCS 1906, 267-283, Springer Verlag, 2000.

[11] P. Katis, N. Sabadini, R.F.C. Walters, Feedback, trace and fixed-point semantics, Theoret. Informatics Appl. 36, 181-194, 2002.

[12] N. Sabadini, R.F.C. Walters, Hierarchical automata and P systems, CATS '03, ENTCS 78, 2003.