Model checking of component connectors

Izadi, M.

5 Context Dependent Connectors

In the previous chapter we addressed one specific shortcoming of the constraint automata as a model of Reo networks, namely the impossibility to model desirable fairness constraints. In this chapter we address another deficiency of constraint automata, that is, their inability to model behavior that depends on pending I/O operations on the ports of a connector. This latter property is called *context dependency*, which manifests itself when the behavior of a connector can change depending upon not only the presence of requests on a connector boundary, but also on their absence.

5.1 Introduction

The prototypical Reo connector featuring a context depended behavior is the *context dependent* lossy synchronous channel (not to be confused with the previous non-deterministic and fair lossy synchronous channels): if the port connected at the source is ready to send data but the port at the sink is not ready to receive, then the data at the source is lost. Until now, we have ignored such requirements and lossy synchronous channels have been modeled by constraint automata or BARs using a (fair) non-deterministic choice. While this is sufficient for modeling Reo networks like the exclusive router presented in Figure 3.2, in general, the presence of context dependent lossy synchronous channels increase the expressiveness of Reo models [13].

First, we describe precisely our definition of the notion of context dependency (or context sensitivity). Context dependency means that the choice of the transition/behavior of a channel/system depends on the (un)availability of I/O requests on its ports. In a trivial sense, all automata/systems can be considered context dependent, because their choice of a transition, of course, depends on the availability of I/O requests on their ports. But, this overly general sense of context dependency is useless. So, we restrict the term *context dependency* to refer to only those cases where the behavior of a channel/system depends on the unavailability of I/O requests over its ports.

In order to address context dependent behavior, we extend the BAR models with the possibility of testing if some ports of the environment are ready to communicate or not. That is, we consider a Büchi variant of Kozen's finite automata on guarded strings [100]. In our case, an infinite guarded string is an alternating sequence of sets of *ready* ports and records of *fired* ports (together with their respective data-flow). The difficulty in correctly addressing a context dependent behavior is not in its modeling per se, but in its effect when composing different connectors. In fact, as for the combination of synchronous and mutual exclusion constraints, also context dependencies should propagate across a connector. This means that the models of two connectors when composed should agree on both the synchronized and mutually excluded common ports, as well as on the tests of the common ports. With this aim, we present a novel definition of a composition operator that generalizes the automata product construction by allowing the alphabets of the two automata to be different.

Our model, called *augmented Büchi automaton of records* (ABAR), has the advantage over previous models for Reo in that it covers the basic concepts of Reo as well as the context sensitive behavior within a standard automata theoretical framework. In fact, we show that

not only every BAR model of a connector can be transformed into an ABAR model but also the context dependent requirements can be modeled by ABARs. The benefits are a clear and easy notation for the representation of component connectors, as well as the efficient existing tool support for automatic analysis.

5.2 Guarded Languages and Augmented Büchi Automata

In this section we augment our model for component connectors so to take into account context dependencies like the ones of the *lossy synchronous channel*: if the port connected at the source is ready for accepting data but the port at the sink it is not ready for receiving it, then the data at the source is lost. In the previous chapter, we have ignored such a requirement and modeled the loss of data by means of a (fair) non-deterministic choice with a BAR. In this section, we extend Büchi automata of records with the capability of modeling coordination strategies based on pending and ignored ports. The idea is to enrich the states of a BAR automaton with expressions for testing if the ports shared with the environment are ready to communicate or not. Intuitively, a transition

$$q \xrightarrow{r} p$$

can be taken only if the ports of the system successfully pass the test associated with a state q. This implies that we must be able to safely eliminate states associated with tests that always fail, and that passing a test has to guarantee that at least as many ports are ready to communicate as needed by every outgoing transitions.

More formally, we consider the set $\mathcal{N}$ of port names as our *primitive test* symbols. Next, we define the set $Exp_{\mathcal{N}}$ of expression for Boolean tests for $\mathcal{N}$ as follows:

Definition 5.1 Let $\mathcal{N}$ be a set of port names. The set $Exp_{\mathcal{N}}$ of expression for Boolean tests for $\mathcal{N}$ is defined by the grammar

$$e ::= 1 \mid 0 \mid A \mid \bar{A} \mid e \cdot e$$

where $A \in \mathcal{N}$.

Each test expression $e \in Exp_{\mathcal{N}}$ is evaluated over a set $N \subseteq \mathcal{N}$ of ports (ready to communicate):

Definition 5.2 Given a set $N \subseteq \mathcal{N}$ of ports, we define when N passes the test expression e, denoted by $N \models e$, as follows:

$$\begin{array}{lll} N \models 1 & & \\ N \not\models 0 & & \\ N \models A & \text{iff} & A \in N \\ N \models \bar{A} & \text{iff} & A \notin N \\ N \models e_1 \cdot e_2 & \text{iff} & N \models e_1 \text{ and } N \models e_2 \end{array}$$

Informally, every collection of ports ready to communicate passes the test expression 1 while every collection of ports ready to communicate containing A passes the primitive test A. The conjunction of two tests e_1 and e_2 is the test $e_1.e_2$, while the negation of a primitive test A is denoted by $\bar{A}$. Note that while we use all test expressions in positive normal form, in general the negation can be used over every test expression, say e, using $\bar{e}$. Then the positive form can be obtained. In this case, the other boolean connectives can be defined as derived operators, for instance we define the disjunction of two expressions e_1 and e_2 to be given by $\overline{\bar{e_1}.\bar{e_2}}$. We use $\equiv$ to denotes the propositional logic equivalence on $Exp_{\mathcal{N}}$.

Now, we define when a record can be executed:

Definition 5.3 Given a record $r \in Rec_{\mathcal{N}}(\mathcal{D})$, let $wp(r)$ be the *weakest precondition* for r to be executed. It is defined inductively on the size of $dom(r)$ as the following expression (up to $\equiv$):

$$wp(\tau) = 1$$
$$wp(r) = A \cdot wp(r \setminus A) \quad \text{if } A \in dom(r)$$

Intuitively, the expression $wp(r)$ is a test checking if all the ports synchronized by r are ready to communicate. Thus, in this case, a transition labeled by r can be fired.

We are now ready to introduce our extension of BARs for modeling both synchronization and context dependencies.

Definition 5.4 An *augmented Büchi automaton of records* (abbreviated by ABAR) is a pair $\langle B, l\rangle$ consisting of a BAR $B = \langle Q, Rec_{\mathcal{N}}(\mathcal{D}), \rightarrow, Q_0, F\rangle$ and labeling function $l{:}Q \rightarrow Exp_{\mathcal{N}}$ such that for all $q \in Q$, if $q \xrightarrow{r} p$ then $l(q)$ implies $wp(r)$.

As a consequence of the above definition, if $l(q) = \bar{A}$, then all transitions outgoing from q must be internal, i.e., they must be labeled by τ. Similarly, all transitions outgoing from a state labeled by 1 must be internal.

We will define ABARs as acceptors of infinite guarded strings [85]. We define our notion of infinite guarded strings:

Definition 5.5 An infinite *guarded string* over the alphabet $Rec_{\mathcal{N}}(\mathcal{D})$ is an alternating infinite sequence $N_0 r_0 N_1 r_1 \cdots$ where $r_i \in Rec_{\mathcal{N}}(\mathcal{D})$ and each N_i is a subset of ports in $\mathcal{N}$. We define a guarded language over the alphabet $Rec_{\mathcal{N}}(\mathcal{D})$ as a set of infinite guarded strings over the same alphabet.

Intuitively, a guarded string represents an execution of the system, where for each step it records the ports ready for communication and the actual data flow among a subset of them. More formally,

Definition 5.6
(1) Let $\gamma = N_0 r_0 N_1 r_1 \cdots$ be an infinite guarded string over alphabet $Rec_{\mathcal{N}}(\mathcal{D})$. We define an *infinite computation* for γ in an ABAR $\langle B, l\rangle$ (over the same alphabet) to be an infinite sequence $\pi = q_0, r_0, q_1, r_1, ...$, of alternating states and records in which $q_0 \in Q_0$, $N_i \models l(q_i)$ and $q_i \xrightarrow{r_i} q_{i+1}$ for all $i \in \mathbb{N}$.
(2) An infinite guarded strings γ is *accepted* by ABAR $\langle B, l\rangle$ if there is an infinite computation for γ in $\langle B, l\rangle$ with at least one of the final states occurring infinitely often.
(3) The *guarded language* of an ABAR $\langle B, l\rangle$, denoted by $GL(B)$, is the set of all infinite guarded strings accepted by it.

Note that the condition of an ABAR $\langle B, l\rangle$ that for every state q, if $q \xrightarrow{r} p$ then $l(q)$ implies $wp(r)$ means that for every guarded string $N_0 r_0 N_1 r_1 \cdots$ accepted, $dom(r_i) \subseteq N_i$ for all $i \geq 0$.

Definition 5.7
(1) We say that two ABARs B_1 and B_2 are *guarded-language equivalent* if $GL(B_1) = GL(B_2)$.
(2) We say that ABAR B and BAR B' are *(language) equivalent* if after ignoring the labeling function of B and considering its language of infinite strings of record we have $L(B) = L(B')$.

Given an ABAR $\langle B, l\rangle$ we can construct a guarded-language equivalent ABAR $\langle B', l'\rangle$ such that $l'(q) \not\equiv 0$ for all states q of B'. In fact, we can safely delete these inconsistent states from the set of states of B and their incoming and outgoing transitions because no set of names N will ever pass the test 0 (not even the empty set of names).

An augmented Büchi automaton of records can be considered as a Büchi automaton of records, if we ignore the labeling function.

Conversely, every Büchi automaton of records B can be transformed into a canonical ABAR $\langle B, l\rangle$ by assigning to each state q of B the conjunction of all $wp(r)$ for each record r labeling outgoing transitions from q. Namely:

Definition 5.8 Let $B = \langle Q, Rec_{\mathcal{N}}(\mathcal{D}), Q_0, \rightarrow, F\rangle$ be a BAR. The *canonical ABAR* for B is the ABAR $\langle B, l\rangle$ where the labeling function is define as follows:

$$\forall q \in Q, l(q) = \bigwedge_{r \in W} wp(r)$$

in which r's are the members of the following set of records:

$$W = \{r \mid \exists q \xrightarrow{r} q' \in B\}.$$

If B be a BAR and B' be its canonical ABAR then considering their languages of streams of records they are equivalent. Let us to have an example:

Example 5.1 Consider the BAR model illustrated in Figure 5.1(a). In fact it is a model of a FIFO2 channel from port A to C obtained after joining two FIFO1 channels and hiding the intermediate port (see Example 4.21). The canonical ABAR for it is illustrated in Figure 5.1(b).

Transforming a BAR into its canonical ABAR and back produces the same BAR, while the converse holds only for an ABAR without states with negative tests.

Although ABARs are as expressive as BARs, in terms of the languages of records that they recognize, they are more concrete. We will use this extra information when composing them. For the moment, we observe that for an ABAR $\langle B, l\rangle$ we can give a formal definition of its pending and ignored ports. Given a set N of ports, we say that $A \in N$ is *ignored* by a transition $q \xrightarrow{r} p$ if $N \models l(q)$ but $A \notin dom(r)$, that is, the port A may be ready to communicate but it is excluded by r. Similarly, we say that a port A is *pending* in a state q if

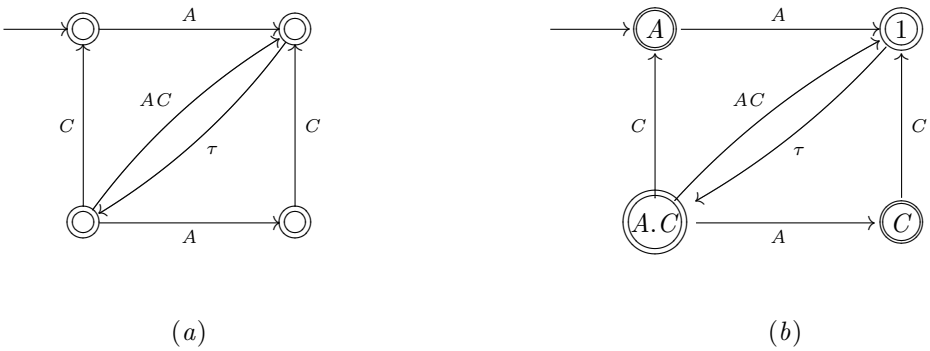
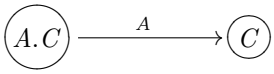


Figure 5.1: A BAR model of a FIFO2 channel and its canonical ABAR.

it is ignored by all transitions outgoing from q. For example, consider the ABAR illustrated in Figure 5.1(b). In the following transition, the port $C \in \{A, C\}$ has been ignored:



Also, suppose that B is an ABAR all whose components are the same as the ABAR illustrated in Figure 5.1(b) except that its initial state is $A \cdot C$. In this case, the port C is suspended in the initial state.

Definition 5.9 We say that two ABARs $\langle B_1, l_1 \rangle$ and $\langle B_2, l_2 \rangle$ are *visibly equivalent* if they have no state labeled by an expression logically equivalent with 0 and $L_{vis}(B_1) = L_{vis}(B_2)$.

Remark 5.1 Sometimes, it is more readable in the definition of an ABAR to assign a set of sets of port names as the label of a state, instead of using a boolean test expression as its label. In other words, based on Definition 5.2 each boolean test expression e can be interpreted as the set of all subsets of the set of port names that satisfy e. Thus, these sets can be directly assigned to the states as their labels. More formally, let $\mathcal{N}$ be the set of port names, $B = \langle Q, Rec_{\mathcal{N}}(\mathcal{D}), \rightarrow, F \rangle$ be a BAR and $\langle B, l \rangle$ be an ABAR using BAR B with a proper labeling function $l{:}Q \rightarrow Exp_{\mathcal{N}}$. We define a labeling function $V{:}Q \rightarrow (2^{\mathcal{N}} \rightarrow \{true, false\})$ such that:

$$\forall N \subseteq \mathcal{N}{:}\ \ V(q)(N) = true \text{ if and only if } N \models l(q).$$

We can semantically consider $\langle B, V \rangle$ as equivalent with the ABAR $\langle B, l \rangle$. In the next chapter, we will translate temporal formulas of our proposed temporal logic, called ρLTL, into ABARs of the form $\langle B, V \rangle$.

5.3 Modeling Reo connectors by ABARs

Now we present the ABAR models of basic Reo connectors and some other useful examples.

Example 5.2 Figure 5.2 shows three visibly equivalent ABAR models of the context dependent lossy synchronous channel from source port A to sink port B over a singleton data domain.

The ABAR illustrated in Figure 5.2(a), which is the most compact one, expresses that if both sink and source ports are ready to communicate simultaneously they exchange data. But if the source is ready while the sink is not the data will be lost. The ABAR model in Figure 5.2(b) expresses the same while it also models the state in which no port is ready. Finally, the ABAR model in Figure 5.2(c) not only models the above mentioned properties but it also allows the sink port to be *suspended* while the source port has no data to deliver or is not ready to communicate. Note that the behavior of a context dependent lossy synchronous channel (as an open system) is deterministic, in the sense that, there is no scenario for the behavior of its environment that allows the channel to be able to make a choice between some transitions. Thus, all possible runs of the context dependent lossy synchronous channel are fair. Therefore, all states in Figures 5.2(a), (b) and (c) are accepting.

Now, to show that the ABAR model is able to express fairness conditions, we consider a closed system containing a context dependent lossy synchronous channel plus its environment, and require that in each trace of this system only finitely many times the input data into the channel can be lost. The enhanced ABAR model supporting this stronger fairness condition is shown in Figure 5.3. Similarly, the ABAR models of Figure 5.2(b) and (c) can be enhanced to support such fairness conditions.

Example 5.3 In Figure 5.4 we show the BAR and two ABAR models of a synchronous channel with source end B and sink C over a singleton data set. The model illustrated in Figure 5.4(b) is the canonical extension of the BAR model in Figure 5.4(a). Compare the expressiveness of the two ABAR models for synchronous channel. While the ABAR in Figure 5.4(b) accepts only the infinite guarded string

$$\{B, C\}[B = d, C = d]\{B, C\}[B = d, C = d]\cdots,$$

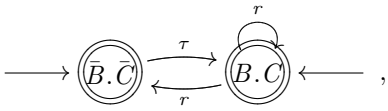
the automaton illustrated in Figure 5.4(c) accepts infinitely many strings, including

$$\{\}\tau\{B, C\}[B = d, C = d]\{\}\tau\{B, C\}[B = d, C = d]\cdots.$$

According to the definition of a synchronous channel in Reo, the channel coordinates the data exchange between the ports to be simultaneous. The ABAR model illustrated in Figure 5.4(c) more explicitly shows the semantics for Reo's Sync channel than the ABAR model illustrated in Figure 5.4(b). It is easy to see that the two automata are visibly equivalent. In a similar way, the synchronous channel can more explicitly be modeled by considering two other passible states, one with the label $B.\bar{C}$ and the other with the label $\bar{B}.C$.

Other basic Reo connectors can also be modeled by ABARs:

A synchronous drain (and similarly for the synchronous spout) between two ports B and C can be modeled as a synchronous channel, but for the data values passing through the two ports that in this case needs not to be the same:



,

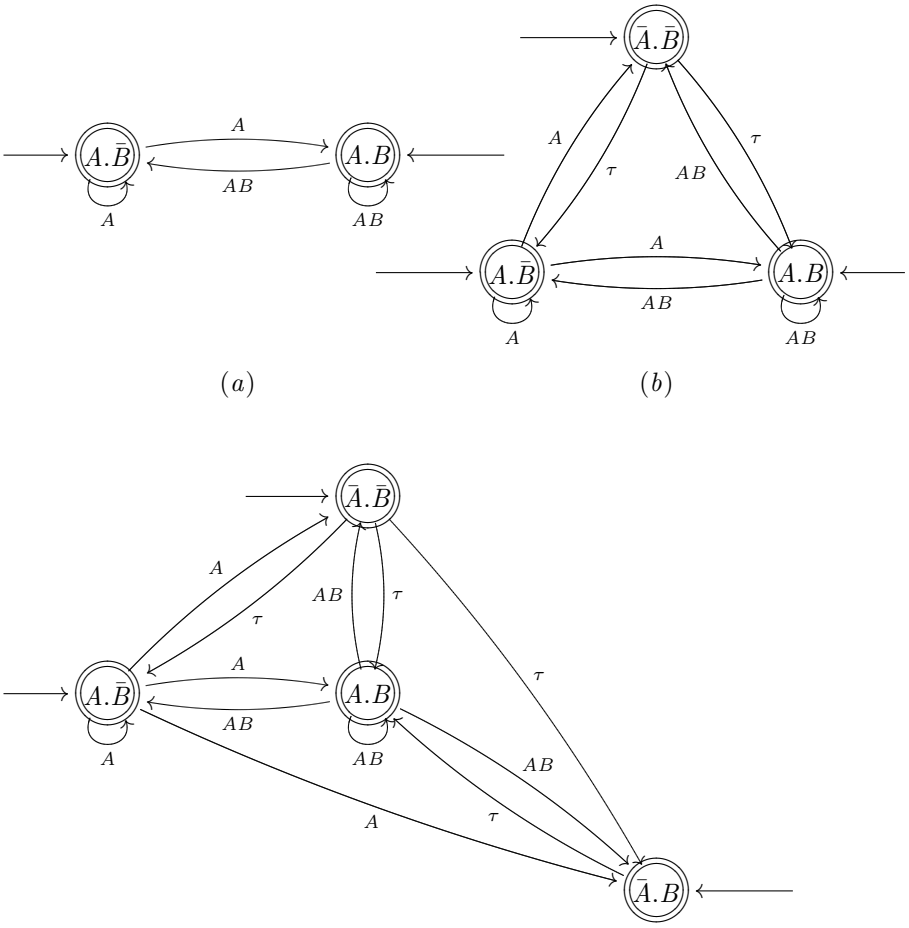


Figure 5.2: Three ABAR models of the context dependent lossy synchronous channel

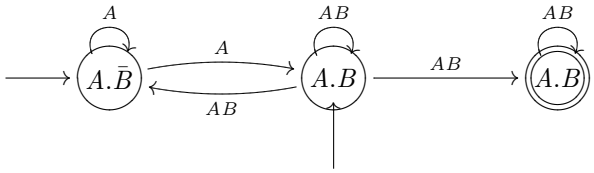


Figure 5.3: The ABAR model of a fair closed system of a context dependent lossy synchronous channel and its environment

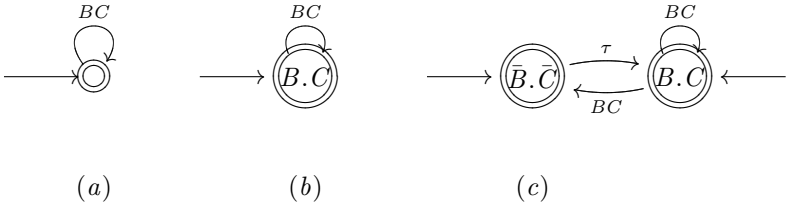
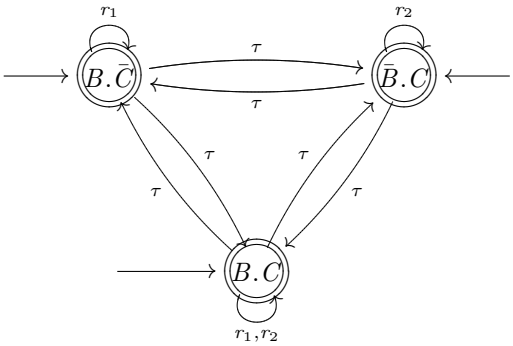


Figure 5.4: Models for a Reo synchronous channel (Sync) from source node B to sink C: (a) Its BAR model; (b) The canonical ABAR model for (a); and (c) The more explicit ABAR model.

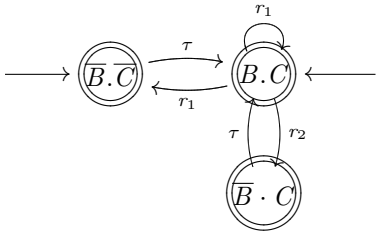
where $dom(r) = \{B, C\}$. Note that based on our definition of context dependency (presented in Section 5.1) synchronous drain and synchronous spout channels are not context dependent.

The asynchronous version of a drain channel between B and C can be modeled by the following ABAR:



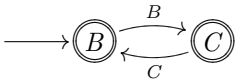
where $dom(r_1) = \{B\}$ and $dom(r_2) = \{C\}$. Note that this channel is (non-trivially) nondeterministic: when write requests exists on both of its ports, the channel can choose to consume either one of them. Thus, in the case of this channel, we can consider some fairness conditions, such as, the requirement that the input data on each port should be consumed infinitely often. Obviously, we can model this fair asynchronous drain by an ABAR with more states not all of which are accepting.

A filter channel from B to C is a synchronous channel that allows for the communication of data items that have a special value. We can model this pattern using the record $[B = p, C = p]$ where p is the special value of the filter. Thus, the ABAR model of filter channel is:



where $r_1 \doteq [B = p, C = p]$, $dom(r_2) = \{B\}$, and $r_2.B \neq p$.

Finally, a FIFO1 channel from B to C is an asynchronous channel that has a buffer with capacity one. Thus, the ABAR model of a FIFO1 channel over a singleton data set is:



5.4 Composing ABAR Models

Now, we introduce the counterpart composition operators that we introduced for BAR's in the case of ABAR's. Again we show that the join operation can be split into two more basic operations: name extension and product.

5.4.1 Product and Join

In this section we give a definition of product and join of two ABAR's.

Definition 5.10 Let $\langle B_1, l_1\rangle$ and $\langle B_2, l_2\rangle$ be two ABAR over the same alphabet, say $Rec_{\mathcal{N}}(\mathcal{D})$. Their *product* is defined as the ABAR $\langle B, l\rangle$, where $B = B_1 \times B_2$ and $l(\langle q, p\rangle) = l_1(q).l_2(p)$.

Similarly, we define the join of two ABARs in terms of the join of their underlying BAR's.

Definition 5.11 Let $\langle B_1, l_1\rangle$ and $\langle B_2, l_2\rangle$ be two ABAR over the same alphabet, say $Rec_{\mathcal{N}}(\mathcal{D})$. Their *join* $\langle B_1, l_1\rangle \bowtie \langle B_2, l_2\rangle$ is defined as the ABAR $\langle B, l\rangle$, where $B = B_1 \bowtie B_2$ and $l(\langle q, p\rangle) = l_1(q).l_2(p)$.

It is easy to check that the join of ABARs is again an ABAR. In fact, if $q_1 \xrightarrow{r_1} p_1$ is a transition in $\langle B_1, l_1\rangle$ and $dom(r_1)$ has no name in common with those used by another ABAR $\langle B_2, l_2\rangle$, then $l_1(q_1).l_2(q_2)$ implies $wp(r_1)$ for all state q_2 of B_2. Similarly, if $q_2 \xrightarrow{r_2} p_2$ is another transition in $\langle B_2, l_2\rangle$ such that $comp(r_1, r_2)$, then $l_1(q_1).l_2(q_2)$ implies $wp(r_1 \cup r_2)$.

As for BAR's, the join of two ABAR's with the same alphabet coincides with their product. In general, the join operator is not a congruence with respect to the visible equivalence.

To see this, it is enough to take two visibly equivalent ABARs with one state labeled in one automaton with $A.B$ and in the other automaton by $A.\bar{B}$. The join of one of them with an automata with a state labeled by B is different than the join of the other.

We now give an example of connector composition.

Example 5.4 Consider the context dependent lossy synchronous channel from port A to port B given in Figures 5.2(a), (b) and (c) and the synchronous channel from B to C as modeled in Figure 5.4(c). Their joint automata are respectively the ABAR models shown in Figures 5.5(a), (b) and (c). Note that they are very similar to their corresponding models of the context dependent lossy synchronous channel between port A and C, except that we can still observe the data-flow through the port B. After hiding port B, each automaton will be the same as its corresponding context dependent lossy synchronous channel.

5.4.2 Hiding of Port Names

Now, we define the hiding operator for the case of ABAR's as the counterpart of the hiding operator we previously defined for BAR's.

Definition 5.12 Let $\langle B, l\rangle$ be an ABAR. The hiding of a port A results in the ABAR $\langle B{\downarrow_A}, l'\rangle$ where $l'(q)$ is the expression $l(q)[1/\bar{A}][1/A]$ which means that we first substitute 1 for every occurrence of $\bar{A}$ and then substitute 1 for every occurrence of A in $l(q)$.

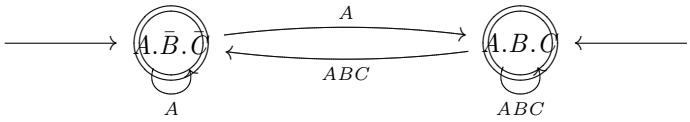
For example, using the above definition, if we hide port B in all three ABAR models illustrated in Figure 5.5 we obtain exactly the ABAR models of a lossy synchronous channel from A to C as illustrated in Figure 5.2 (after a renaming of the sink port which now is C not B).

Now consider some more complex examples of joining of context dependent connectors and then hiding the common ports:

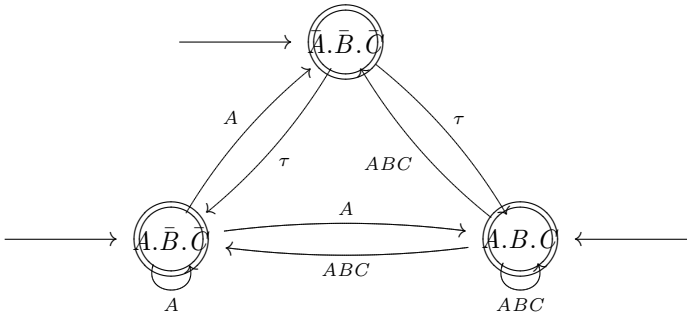
Example 5.5 In Figure 5.6 we consider the composition of a context dependent lossy synchronous channel from port A to B with another one from port B to C. The resulting ABARs before and after hiding the common port B are illustrated in Figures 5.6(c) and (d). As we expect, in the product automaton (before hiding), the first channel (from port A to B) indeed always acts as a normal synchronous channel; i.e., it never loses. The resulting connector is exactly an ABAR model of a context dependent lossy synchronous channel from A to C.

Example 5.6 In Figure 5.7 we consider the composition of a context dependent lossy synchronous channel from port A to B with a FIFO1 channel from port B to C, after hiding the common port B. Note that in the initial state of the resulting connector the buffer is empty and in the two other states the buffer is full. As we expect, whenever the buffer is empty no data value from port A is lost, whereas this happen when the buffer is full.

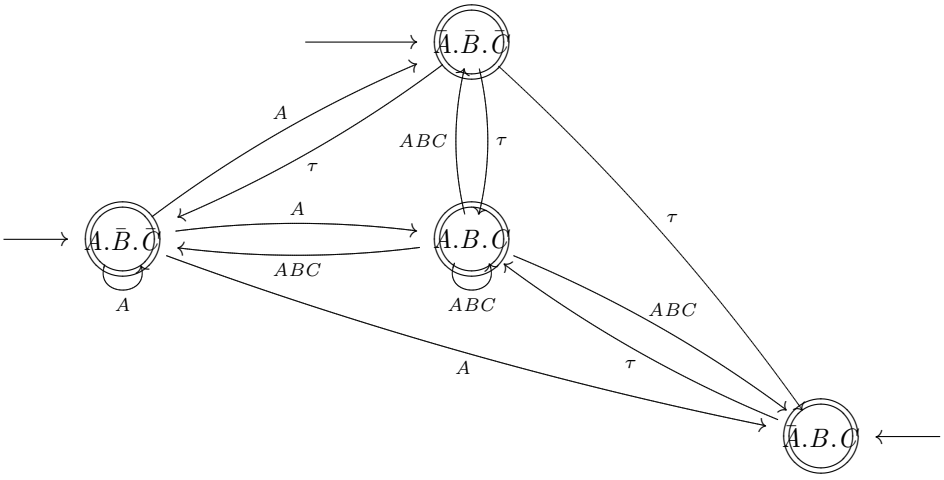
Example 5.7 Consider the composition of context dependent lossy synchronous and FIFO1 channels in a reverse direction as we did in Example 5.6. In Figure 5.8 we consider the composition of a FIFO1 channel from port A to B with a context dependent lossy synchronous channel from port B to C, after hiding the common port B. In the two initial states the buffer is empty and in the others it is full. As we expect, when the buffer is empty firing port A



(a)



(b)



(c)

Figure 5.5: The composition of the ABAR models of a context dependent lossy synchronous channel and a synchronous channel

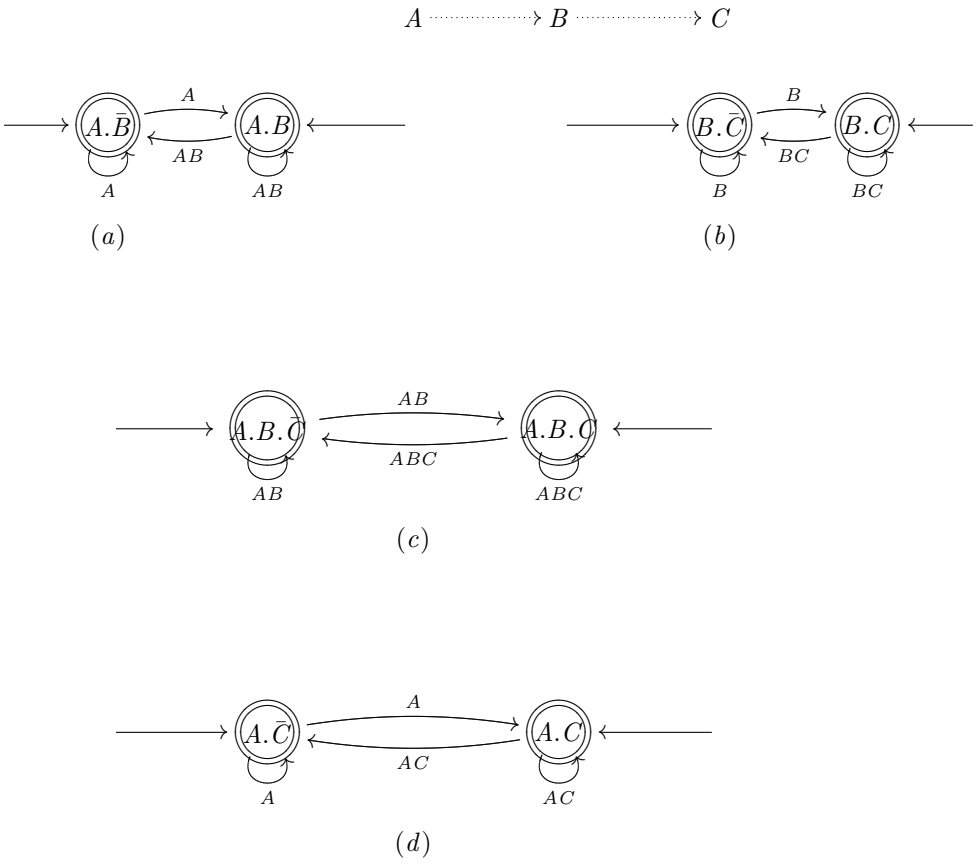


Figure 5.6: The composition of two context dependent lossy synchronous channels.

causes the buffer to become full. In the next stage, if C is ready to get data, it receives it, but if C is not ready data will be lost and either way the buffer becomes empty.

Example 5.8 In Figure 5.9 we consider the composition of a synchronous channel from port A to B with a FIFO1 channel from port B to C, after hiding the common port B. As we expect, the obtained ABAR is visibly equivalent to a FIFO1 channel from source port A to sink port C.

Example 5.9 Consider the composition of a synchronous and a FIFO1 channel in a reverse direction as we did in Example 5.8. In Figure 5.10 we consider the composition of a FIFO1 channel from port A to B with a synchronous channel from port B to C, after hiding the common port B. As we expect the obtained model is a FIFO1 channel from A to C.

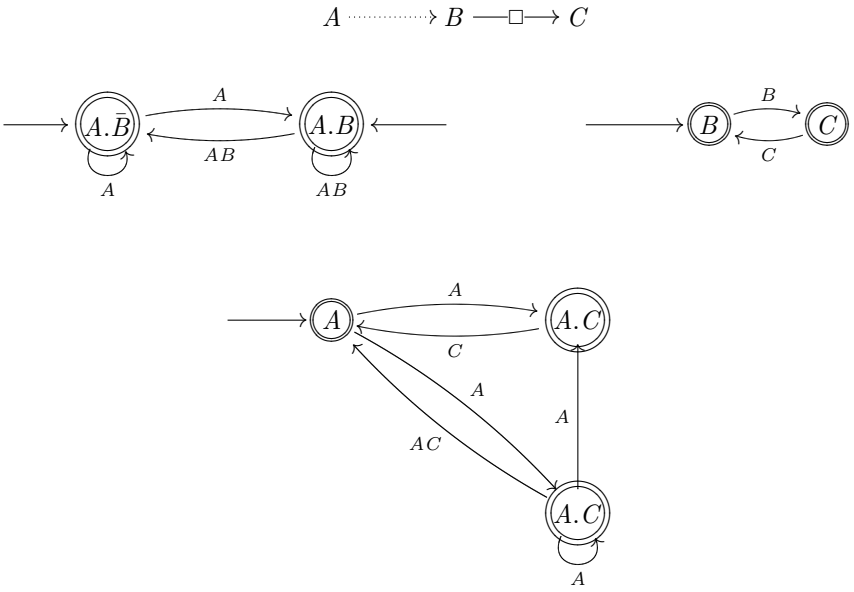


Figure 5.7: The composition of a context dependent lossy synchronous channel with a FIFO1 channel.

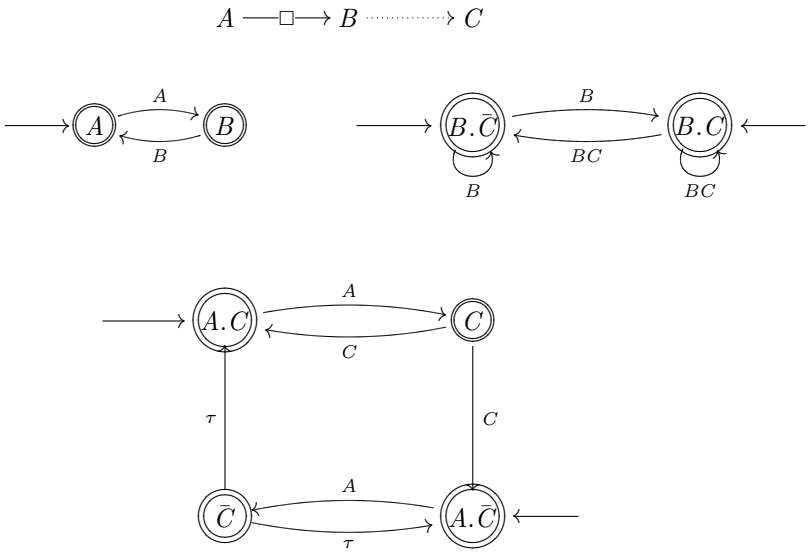


Figure 5.8: The composition of a FIFO1 channel with a context dependent lossy synchronous channel.

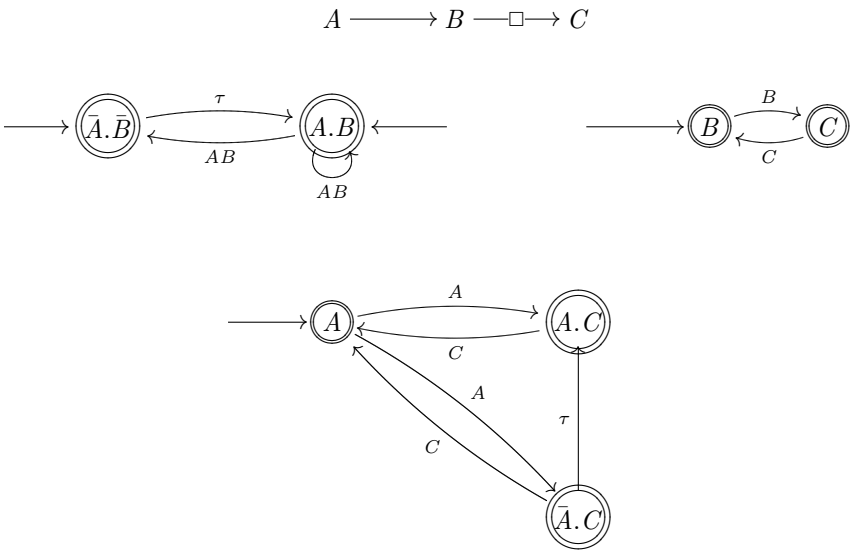


Figure 5.9: The composition of a synchronous channel with a FIFO1 channel.

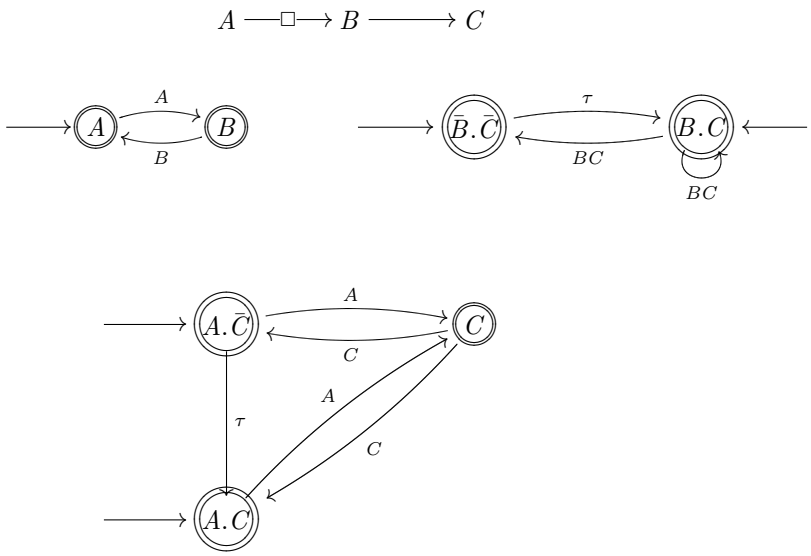


Figure 5.10: The composition of a FIFO1 with a synchronous channel.

5.4.3 Splitting the join

We now show that the procedure of splitting the join into name extension and production that we introduced for BARs is applicable to ABARs as well.

Theorem 5.1 Let $\langle B_1, l_1\rangle$ and $\langle B_2, l_2\rangle$ be two ABARs over the alphabet sets $Rec_{\mathcal{N}_1}(\mathcal{D})$ and $Rec_{\mathcal{N}_2}(\mathcal{D})$, respectively. Then,

$$\langle B_1{\uparrow}\mathcal{N}_2 \times B_2{\uparrow}\mathcal{N}_1, l'\rangle = \langle B_1, l_1\rangle \bowtie \langle B_2, l_2\rangle,$$

where $l'(\langle q_1, q_2\rangle) = l_1(q_1).l_2(q_2)$, and q_1 is a state of B_1 and q_2 is a state of B_2.

Proof. The proof is a simple extension of the proof of Theorem 4.7. □

Example 5.10 Consider Figure 5.11. Figures 5.11(a) and 5.11(b) show the simplest ABAR models of two FIFO1 channels (over a singleton data set $\mathcal{D} = \{d\}$). They are the same BAR models we presented in the previous chapter, now augmented with proper labels. The extension of the first ABAR with port name C appears in Figure 5.11(d), while the extension of the second automaton with port name A appears in Figure 5.11(e). Their product is the automaton in 5.11(c) which is obtainable using either the direct or the splitting definitions of the join operation.

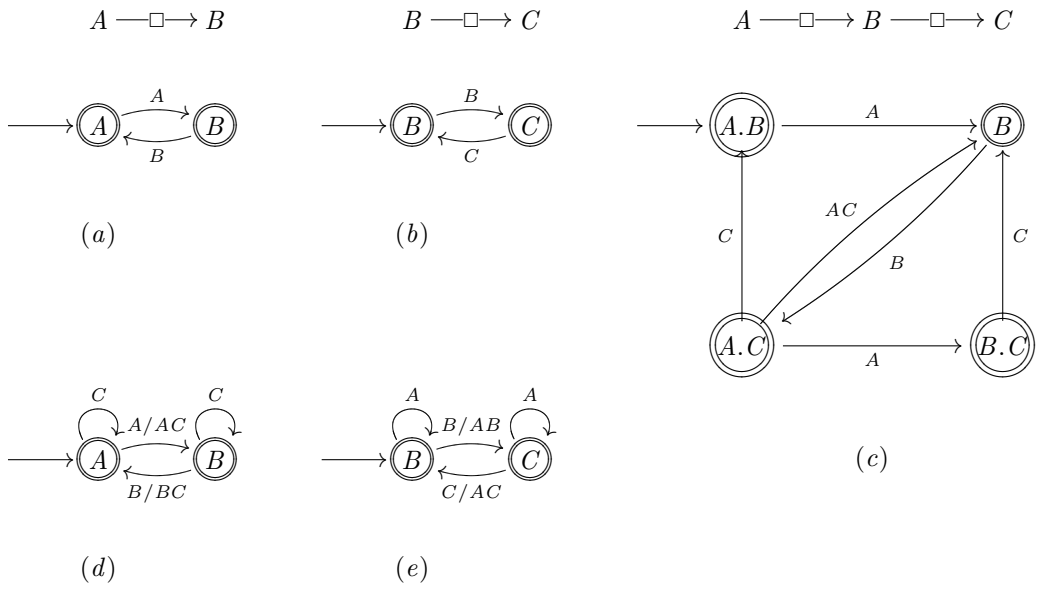


Figure 5.11: Direct and indirect joining of two FIFO1 buffers modeled by ABARs

5.5 Context Dependent Fair Constraint Automata

In Section 4.6 we introduced a new notion and semantics for constraint automaton called *fair constraint automaton* (FCA). The syntax of FCA is the same as that of ordinary constraint automaton except that it also has some final states. From the semantics point of view, each FCA is defined as an acceptor of infinite strings of records. An infinite string of records is accepted by an FCA if at least one of the accepting states occurs in the execution for the string infinitely many times. Now, we investigate the augmentation of FCAs by test expression labels as we did in this chapter for BARs. Let $C = \langle Q, \mathcal{N}, \longrightarrow, Q_0, F \rangle$ be a FCA over a port set $\mathcal{N}$ and a data set $\mathcal{D}$, as in Definition 4.14. Also, the set of all test expressions over the port set $\mathcal{N}$ and the data set $\mathcal{D}$ be the same as in Definition 5.1, with their semantics as in Definition 5.2. Now we define the augmentation of FCAs:

Definition 5.13 An augmented fair constrain automaton (abbreviated by AFCA) is a pair $\langle C, l \rangle$ consisting of an FCA $C = \langle Q, \mathcal{N}, \longrightarrow, Q_0, F \rangle$ and a labeling function $l : Q \to Exp_{\mathcal{N}}$ such that for all $q \in Q$, if $q \xrightarrow{N,q} q'$ then $l(q)$ implies $wp(N)$, where for every $N \subseteq \mathcal{N}$, $wp(N)$ is defined inductively as follows:

$$wp(\emptyset) = 1$$
$$wp(N) = A \cdot wp(N \setminus A) \quad \text{if } A \in N.$$

As a consequence of the above definition, if $l(q) = \bar{A}$, then all transitions outgoing from

q must be internal, i.e., they must be labelled by τ. Similarly, all transitions outgoing from a state labeled by 1 must be internal.

Same as the case of ABARs, we regard AFCAs as acceptors of infinite guarded strings of records:

Definition 5.14
(1) Let $\gamma = M_0 r_0 M_1 r_1 \cdots$ be an infinite guarded string over the alphabet $Rec_{\mathcal{N}}(\mathcal{D})$. We define an *infinite computation* for γ in an AFCA $\langle C, l\rangle$ (over the same sets $\mathcal{N}$ and $\mathcal{D}$) to be an infinite sequence $\pi = q_0, (N_0, g_0), q_1, (N_1, g_1), ...$, of alternating states and (port set , guard) pairs in which $q_0 \in Q_0$, $M_i \models l(q_i)$, and there is data assignment $\delta{:}N_i \rightarrow \mathcal{D}$ such that $\delta \models g_i$, $dom(r_i) = N_i$ and $\forall n \in N_i{:}\, r.n = \delta(n)$.
(2) An infinite guarded strings γ is *accepted* by AFCA $\langle C, l\rangle$ if there is an infinite computation for γ in $\langle C, l\rangle$ with at least one of the final states occurring infinitely often.
(3) The *guarded language* of an AFCA $\langle C, l\rangle$, denoted by $GL(C)$, is the set of all infinite guarded strings accepted by it.

Definition 5.15
(1) Two AFCAs C_1 and C_2 are *guarded-language equivalent* if $GL(C_1) = GL(C_2)$.
(2) AFCA C and BAR C' are *(language) equivalent* if after ignoring the labeling function of C and considering its language of infinite strings of record we have $L(C) = L(C')$.

Given an AFCA $\langle C, l\rangle$ we can construct a guarded-language equivalent AFCA $\langle C', l'\rangle$ such that $l'(q) \not\equiv 0$ for all states q of B'. In fact, we can safely delete these inconsistent states from the set of states of B and their adjunct transitions because no set of names N will ever pass the test 0 (not even the empty set of names).

An augmented fair constraint automaton (AFCA) can be considered as a fair constraint automaton (FCA), if we ignore its labeling function.

Conversely, every fair constraint automaton C can be transformed into a canonical AFCA $\langle C, l\rangle$ by assigning to each state q of C the conjunction of all $wp(N)$ for each $N \in \mathcal{N}$ that is the first component of a pair (N, g) labeling the outgoing transitions from q. Namely:

Definition 5.16 Let $C = \langle Q, \mathcal{N}, \longrightarrow, Q_0, F\rangle$ be an FCA over a port set $\mathcal{N}$ and a data set $\mathcal{D}$. The *canonical AFCA* for C is the AFCA $\langle C, l\rangle$ where the labeling function is define as follows:

$$\forall q \in Q, l(q) = \bigwedge_{N \in W} wp(N)$$

and

$$W = \{N \mid \exists q \xrightarrow{N,g} q' \in C\}.$$

Obviously, if C is an FCA and C' is its canonical AFCA then considering their languages of streams of records, they are equivalent. Transforming a BAR into its canonical ABAR and back will produce the same BAR, while the converse holds only for an ABAR without states with negative tests.

Let us compare the expressiveness of ABARs and AFCAs. Obviously, because the semantics of both augmented Büchi automata of records and augmented fair constraint automata are based on guarded languages of streams of records, each ABAR B over a name set $\mathcal{N}$ and a data set $\mathcal{D}$ can be considered as an AFCA if we replace every transition label $r \in Rec_{\mathcal{N}}(\mathcal{D})$

with (N, g) where, $N = dom(r)$ and g is the data constraint $\bigwedge_{n \in dom(n)}(d_n = r.n)$. Using this simple conversion of ABAR into AFCA, we can show that all ABARs that we introduced as models of Reo connectors can be considered as AFCA models of them.

Conversely, if C is an AFCA over a finite name set $\mathcal{N}$ and a finite data set $\mathcal{D}$ then C is equivalent with ABAR B all whose components are the same as C except that each transition $q \xrightarrow{N,g} q'$ of C is replaced with a set of transitions of the form $q \xrightarrow{r} q'$ where r satisfies the following conditions: (1) $dom(r) = N$, and (2) there exists a data assignment $\delta{:}N \to \mathcal{D}$ such that $\delta \models g$ and $\forall n \in N$, $\delta(n) = r.n$. Obviously, if at least one of the sets $\mathcal{N}$ or $\mathcal{D}$ is infinite then in replacing an AFCA's transitions with a set of transitions with record labels, we will need to have records with infinite domains or to replace the transition of the AFCA with an infinite set of transitions with record labels. This density in the syntax of AFCAs in comparison with the syntax of ABAR's is the main advantage of using AFCA instead of ABAR.