Model checking of component connectors

Izadi, M.

6 Model Checking

In the previous chapters, we introduced Büchi automata of records and their augmented versions as the operational models of Reo connectors considering unconditional fairness and context dependency requirements. Now we have an operational semantics for Reo based on Büchi automata. Thus, it is very natural to use these models for automata theoretic model checking of Reo nets. Generally speaking, *automata theoretic methods of model checking* verify if a system satisfies a desired property using three steps. First, we model the system by an automaton and specify the property by a formula in a temporal logic. The next steps are applicable if there is a well established translation of the formulas of the selected temporal logic into the selected type of automata. In this case, the second step is to translate the negation of the formula that expresses the desired property into an automaton of the same type as the one used to model the system. Now, we have two automata of the same type. The last step is to check the intersection of their languages. If the intersection is empty, it will be a proof that the system satisfies the property. Otherwise, each member of the intersection set is a counterexample trace of the system that violates the property. The model checking process can be improved (both from time and space complexity points of view) if we can do the generation of the state space of the automaton model of the system or the translation of the property formula into its equivalent automaton and the checking of the emptiness of their intersection in parallel. If some form of this parallelism is possible, we call the model checking process *on-the-fly*.

In this chapter, we try to use the automata theoretic method of model checking for systems modeled by ABARs. First, we introduce an action (or transition) based linear temporal logic (called ρLTL) interpreted over computations of ABARs. Then, we show that ρLTL formulas can be translated into ABARs both using inductive and on-the-fly methods. In each case, we obtain a technique to verify Reo nets.

6.1 Record-based linear-time temporal logic

In this section we introduce a record based linear time temporal logic (ρLTL) which is an extension of linear time temporal logic (LTL) [145] for reasoning about data-flow, synchronization and context dependencies of Reo connectors. We use as atomic propositions sets of port names, indicating the ports ready to communicate, and index the usual next state operator of LTL with a record, for the specification of communicating ports and of their respective data-flow.

Definition 6.1 Syntax of ρLTL. The set of *ρLTL formulas* over a finite set of port names $\mathcal{N}$ and a finite set of data $\mathcal{D}$ is defined inductively by the following syntax:

$$\phi ::= \ true \mid N \mid \neg\phi \mid \phi \vee \phi \mid \langle r \rangle \phi \mid \phi \, U \, \phi.$$

where $N \subseteq \mathcal{N}$ and $r \in Rec_{\mathcal{N}}(\mathcal{D})$.

Formulas of ρLTL are interpreted over infinite guarded strings. A necessary condition to interpret a formula for a guarded string is that both use the same set of port names $\mathcal{N}$ and data set $\mathcal{D}$, which we assume to hold in the sequel. Intuitively, N holds for a guarded string

if N is the first *guard* of the string, whereas $\langle r \rangle \phi$ holds if r is the first *action* of the string and ϕ holds for its remaining suffix.

Formally, given an infinite guarded string $M = N_0 r_0 N_1 r_1 \cdots$, we define M^i as the guarded string $N_i r_i N_{i+1} r_{i+1} \cdots$. Here we consider only guarded strings for which $N_i \subseteq \mathcal{N}$ and $r_i \in Rec_{\mathcal{N}}(\mathcal{D})$, for all $i \geq 0$.

Definition 6.2 Semantics of ρLTL. Let $M = N_0 r_0 N_1 r_1 \cdots$ be an infinite guarded string over a name set $\mathcal{N}$ and a data domain $\mathcal{D}$ such that $\forall i \geq 0, dom(r_i) \subseteq N_i$. The semantics of a ρLTL formula is defined inductively over such M's as follows:

$$
\begin{array}{lll}
M \models true & & \\
M \models N & \text{iff} & N_0 = N \\
M \models \phi_1 \vee \phi_2 & \text{iff} & M \models \phi_1 \text{ or } M \models \phi_2 \\
M \models \neg\phi & \text{iff} & M \not\models \phi \\
M \models \langle r \rangle \phi & \text{iff} & r_0 = r \text{ and } M^1 \models \phi \\
M \models \phi_1 U \phi_2 & \text{iff} & \exists j \geq 0 \text{ such that } M^j \models \phi_2 \text{ and } \forall 0 \leq i < j, M^i \models \phi_1
\end{array}
$$

Based on the above semantic definitions and the intuitions behind them, the temporal operators $\langle r \rangle$ and U are called *(action-based) next* and *until* operators respectively. As usual, we denote by $\| \phi \|$ the set of all models of the ρLTL formula ϕ, and define logical equivalence $\equiv$ of ρLTL formulas as $\phi_1 \equiv \phi_2$ if and only if $\| \phi_1 \| = \| \phi_2 \|$. If B is an ABAR and ϕ a ρLTL formula, we write $B \models \phi$ if $L(B) \subseteq \| \phi \|$.

Several other operators can be derived from the basic operators of ρLTL. $false$ is defined as $\neg true$. The Boolean operators $\wedge$ and $\rightarrow$ are derived in the obvious way:

$$\phi_1 \wedge \phi_2 = \neg(\neg\phi_1 \vee \neg\phi_2),$$

$$\phi_1 \rightarrow \phi_2 = \neg(\phi_1 \vee \neg\phi_2).$$

The temporal modalities *eventually* and *always* can be derived as usual:

$$\Diamond\phi = true U \phi,$$

$$\Box\phi = \neg\Diamond\neg\phi.$$

The dual operator of *until* is the *release* operator defined as:

$$\phi \, R \, \psi = \neg(\neg\phi \; U \; \neg\psi).$$

The *weak* variant 'W' of the until operator is obtained as:

$$\phi \; W \; \psi = (\phi \; U \; \psi) \vee \Box\phi.$$

Using the fact that if $M = N_0 r_0 N_1 r_1 \cdots$ is a guarded string that is used as the semantic domain of ρLTL formulas then $\forall i \geq 0, dom(r_i) \subseteq N_i$ and that $\mathcal{N}$ is finite, we can conclude hat:

$$\langle r \rangle \phi \equiv \Big(\bigvee_{\mathcal{N} \supseteq N \supseteq dom(r)} N \Big) \wedge \langle r \rangle \phi.$$

Based on the above fact, we can also derive other nice equivalences such as this:

$$\{A\} \wedge \langle [A = 1, B = 1] \rangle true \ \equiv \ false.$$

The dual operator of $\langle r \rangle \phi$ is

$$[r]\phi = \neg \langle r \rangle \neg \phi$$

which intuitively holds for a guarded string if either its first action is other than r or its continuation satisfies ϕ. In fact, $[r]\phi \equiv \neg \langle r \rangle true \vee \langle r \rangle \phi$. For example, the formula $[r]false$ is satisfied by all guarded strings having a record other than r as their first action. We prove this equivalence in the following lemma:

Lemma 6.1

$$[r]\phi \equiv \neg \langle r \rangle true \vee \langle r \rangle \phi.$$

Proof. Let $M = N_0 r_0 N_1 r_1 \cdots$ be a guarded string. Now,

$$\begin{array}{lll} M \models [r]\phi & \text{iff} & M \models \neg \langle r \rangle \neg \phi \\ & \text{iff} & M \not\models \langle r \rangle \neg \phi \\ & \text{iff} & \text{it is not the case that } (r_0 = r \text{ and } M^1 \models \neg \phi) \\ & \text{iff} & r_0 \neq r \text{ or } M^1 \models \phi \\ & \text{iff} & M \models \neg \langle r \rangle true \text{ or } M \models \langle r \rangle \phi \\ & \text{iff} & M \models \neg \langle r \rangle true \vee \langle r \rangle \phi. \end{array}$$

□

6.1.1 Some useful encodings

Because there are only finitely many records in $Rec_{\mathcal{N}}(\mathcal{D})$, the standard *next* operator of linear time temporal logic can be defined as:

$$\bigcirc \phi = \bigvee_{r \in Rec_{\mathcal{N}}(\mathcal{D})} \langle r \rangle \phi.$$

It is not hard to see that the next operator is self-dual, in the sense that

$$\neg \bigcirc \phi \equiv \bigcirc \neg \phi.$$

Further, because our models are infinite strings, $\bigcirc true \equiv true$, meaning that connectors are reactive and cannot stop the data flow (progress is always possible).

Two important equivalences are the definitions of *until* (U) and *release* (R) temporal operators using a recursive style [29]:

$$\phi_1 \ U \ \phi_2 \ \equiv \ \phi_2 \vee (\phi_1 \wedge \bigcirc(\phi_1 \ U \ \phi_2)),$$

$$\phi_1 \ R \ \phi_2 \ \equiv \ \phi_2 \wedge (\phi_1 \vee \bigcirc(\phi_1 \ R \ \phi_2)).$$

Definition 6.3 A *data constraint* δ for a set of names $N \subseteq \mathcal{N}$ is a satisfiable propositional formula built from the atoms $d_A \in P$, $d_A = d$, and $d_A = d_B$, where $A, B \in N$, $d \in \mathcal{D}$ and $P \subseteq \mathcal{D}$.

Data constraints, together with a set of names on which they are defined can be viewed as a symbolic representation of a set of records. We can therefore define a derived operator $\langle N, \delta\rangle\phi$, where δ is a data constraint for N, by setting

$$\langle N, \delta\rangle\phi = \bigvee\{\langle r\rangle\phi \mid dom(r) = N, r \models \delta\}\,,$$

where $r \models (d_A \in P)$ if $r.A \in P$, $r \models (d_A = d)$ if $r.A = d$ and $r \models (d_A = d_B)$ if $r.A = r.B$ (disjunction and negation are as expected).

In [17, 18], timed scheduled-data expressions are introduced to specify data stream logic. Leaving out time, *scheduled-data expressions* are ordinary regular expressions built from either data constraints or records. Scheduled-data expressions α are incorporated in data stream logic by formulas of the type $\langle\langle\alpha\rangle\rangle\phi$. More formally, a scheduled-data expression α can be defined using the following abstract grammar:

$$\alpha ::= 0 \mid 1 \mid \langle N, \delta\rangle \mid r \mid \alpha + \alpha \mid \alpha \times \alpha \mid \alpha;\alpha \mid \alpha^*.$$

While in general standard linear temporal logic cannot express regular expressions for prefixes of infinite strings, we can encode scheduled-data expressions in our action based linear temporal logic ρLTL by using a function $(-)^\dagger$ that maps scheduled-data expressions of the form $\langle\langle\alpha\rangle\rangle\phi$ into ρLTL formulas. The function $(-)^\dagger$ is defined recursively as follows:

$$\begin{array}{lcl}
(true)^\dagger & = & true \\
(N)^\dagger & = & N \\
(\langle\langle 0\rangle\rangle\phi)^\dagger & = & false \\
(\langle\langle 1\rangle\rangle\phi)^\dagger & = & (\phi)^\dagger \\
(\langle\langle \langle N, \delta\rangle\rangle\phi)^\dagger & = & \langle N, \delta\rangle(\phi)^\dagger \\
(\langle\langle \langle r\rangle\rangle\phi)^\dagger & = & \langle r\rangle(\phi)^\dagger \\
(\langle\langle \alpha_1 + \alpha_2\rangle\rangle\phi)^\dagger & = & (\langle\langle \alpha_1\rangle\rangle\phi)^\dagger \vee (\langle\langle \alpha_2\rangle\rangle\phi)^\dagger \\
(\langle\langle \alpha_1 \times \alpha_2\rangle\rangle\phi)^\dagger & = & (\langle\langle \alpha_1\rangle\rangle\phi)^\dagger \wedge (\langle\langle \alpha_2\rangle\rangle\phi)^\dagger \\
(\langle\langle \alpha_1;\alpha_2\rangle\rangle\phi)^\dagger & = & (\langle\langle \alpha_1\rangle\rangle(\langle\langle \alpha_2\rangle\rangle\phi))^\dagger \\
(\langle\langle \alpha^*\rangle\rangle\phi)^\dagger & = & (\langle\langle \alpha\rangle\rangle true)^\dagger\, U (\phi)^\dagger \\
(\phi \wedge \psi)^\dagger & = & (\phi)^\dagger \wedge (\psi)^\dagger
\end{array}$$

Note that 0 is the unit with respect to the union operator $+$, and 1 is the unit with respect to the composition operator ;. In fact we have

$$\begin{array}{l}
\langle\langle 0 + \alpha\rangle\rangle\phi \equiv \langle\langle 0\rangle\rangle\phi \vee \langle\langle \alpha\rangle\rangle\phi \equiv false \vee \langle\langle \alpha\rangle\rangle\phi \equiv \langle\langle \alpha\rangle\rangle\phi \\
\langle\langle 1;\alpha\rangle\rangle\phi \equiv \langle\langle 1\rangle\rangle(\langle\langle \alpha\rangle\rangle\phi) \equiv \langle\langle \alpha\rangle\rangle\phi\,.
\end{array}$$

Scheduled-data expressions allow us to express formulas that hold only for externally observable steps, thus not sensible for a finite number of internal steps. Given a ρLTL formula ϕ, we define $\Diamond_\tau\phi = \langle\langle\tau^*\rangle\rangle([\tau]false \wedge \phi)$. Informally, $\Diamond_\tau\phi$ holds if ϕ holds after finitely many internal τ steps.

6.1.2 Specifying Reo connectors

We now present some examples of specification of basic Reo connectors using ρLTL formulas. First, for simplicity, the ABARs depicted in Figure 6.1 are considered as models of some

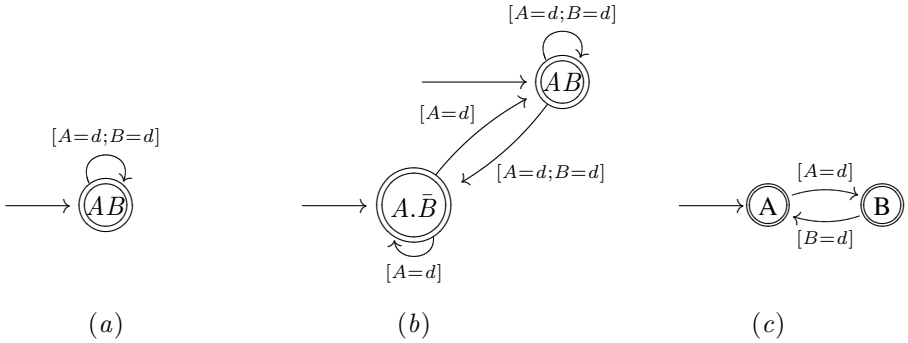

Figure 6.1: ABAR models of some basic Reo connectors: (a) Sync channel, (b) Context-Dependent LossySync channel, and (c) FIFO1 channel.

of the basic Reo connectors between two ports A and B over a singleton data set $\{d\}$. The labels of the states are sets of port names that are ready before firing the outgoing transitions.

Now, consider a synchronous channel from a port A to a port B. If both ports are enabled, then the channel must let the data flow. This can be expressed by the following ρLTL formula:

$$\phi \;=\; \Box(\{A, B\} \rightarrow \langle\langle\{A, B\}, d_A = d_B\rangle\rangle true)\,. \tag{6.1}$$

The above formula is clearly satisfied by our ABAR model of synchronous channel in Figure 6.1. However, it is also satisfied by the ABAR modeling a lossy synchronous channel. This is because (6.1) does not guarantee that data cannot flow through a single port. We remedy this by adding to the specification of a synchronous channel the following

$$\phi_1 \;=\; \Box(\neg\langle\langle\{A\}, true\rangle\rangle true \vee \neg\langle\langle\{B\}, true\rangle\rangle true)\,.$$

The above formula does not hold for the lossy synchronous channel. In fact, for such a connector it holds that if port A is enabled but B is not, then the data at A is lost. This is expressed by

$$\phi_2 \;=\; \Box((\{A\} \wedge \neg\{A, B\}) \rightarrow \langle\langle\{A\}, true\rangle\rangle true)$$

Further, in a lossy synchronous channel, data cannot flow through port B alone, that is

$$\phi_3 \;=\; \Box\neg\langle\langle B, true\rangle\rangle true.$$

Thus, a possible specification of the synchronous channel is

$$Sync = \phi \wedge \phi_1$$

while a specification of a lossy synchronous channel of Reo is

$$LossySync = \phi \wedge \phi_2 \wedge \phi_3.$$

Differen than the two previous channels, a FIFO1 channel is asynchronous, meaning that data does not flow simultaneously through its ports A and B, that is

$$\psi_1 \;=\; \Box\neg\langle\langle\{A, B\}, true\rangle\rangle true.$$

Further, a data item received through port A is never lost, as it is output to port B as soon as B is enabled. Of course, this does not need to be immediate and it can even be the case that B is never enabled. This is specified by means of a weak until operator allowing possibly infinitely many internal steps between the two observable actions:

$$\psi_2 \;=\; \Box \bigwedge_{d\in\mathcal{D}} \langle [A=d]\rangle(\langle\tau\rangle true \wedge \neg(\{B\}\vee\{A,B\}))\; W\; \langle [B=d]\rangle)\,.$$

To complete the specification of a FIFO1 channel, we need the converse of the above property, stating that after a data item flows through port B the store of the channel is empty and hence a new data item can flow through port A as soon as A is enabled:

$$\psi_3 \;=\; \Box\langle\langle\{B\}, true\rangle\rangle(\langle\tau\rangle true \wedge \neg(\{A\}\vee\{A,B\}))\; W\; \langle\langle\{A\}, true\rangle\rangle)\,.$$

Thus, in a FIFO1 channel, data flow through its two ports alternately, and never simultaneously. Summarizing, a specification for the FIFO1 channel is

$$FIFO1 \;=\; \psi_1 \wedge \psi_2 \wedge \psi_3.$$

6.2 From formulas to automata: model checking

In this section we introduce a global translation of ρLTL formulas into ABARs. Our construction is based on the translation from ordinary LTL formulas to Büchi automata [149], adapted to take into account the next state operator indexed by records. For simplicity, the resulting ABAR will have multiple sets of accepting states in which, a run is accepted if and only if for each accepting states set there exists at least one state that appears infinitely often in that run. Namely, we translate formulaes to generalized ABARs. To obtain an ordinary ABAR, one can use the fact that for each generalized Büchi automaton there is a language-equivalent ordinary Büchi automaton [138].

For technical convenience we will work with a positive form of ρLTL called ρLTL+.

Definition 6.4 Let $\mathcal{N}$ and $\mathcal{D}$ be respectively a finite nonempty set of port names and a finite nonempty set of data. The set of ρLTL+ formulas over sets $\mathcal{N}$ and $\mathcal{D}$ is the set of all formulas defined using the following abstract grammar:

$$\phi ::= \; true \,|\, false \,|\, N \,|\, \phi\wedge\phi \,|\, \phi\vee\phi \,|\; \bigcirc\phi \,|\, \langle r\rangle\phi \,|\, [r]\phi \,|\, \phi U\phi \,|\, \phi R\phi$$

where $N\subseteq\mathcal{N}$ and $r\in Rec_{\mathcal{N}}(\mathcal{D})$.

It is obvious that every ρLTL formula is equivalent to a positive one by pushing the negation inside every operator and replacing every instance of $\neg N$ with $\bigvee_{N'\subseteq\mathcal{N}, N'\neq N} N'$. Note that the size of the resulting positive formula is linear in the size of the ρLTL formula. The inclusion of the ordinary next state operator $\bigcirc\phi$ is to simplify the presentation.

We begin the translation of ρLTL+ formulas to automata by defining the closure $CL(\phi)$ of a ρLTL+ formula ϕ. Note that the closure mat include formulas that are not in the language of ρLTL+ (such as $\psi = \neg\langle r\rangle true$).

Definition 6.5 The closure $CL(\phi)$ of a ρLTL+ formula ϕ is the smallest set of ρLTL formulas such that:

- $\phi \in CL(\phi)$,
- $true, false \in CL(\phi)$,
- if there is $N \subseteq \mathcal{N}$ that $N \in CL(\phi)$ then for all $N' \subseteq \mathcal{N}$, $N' \in CL(\phi)$,
- if $\phi_1 \vee \phi_2 \in CL(\phi)$ then $\phi_1, \phi_2 \in CL(\phi)$,
- if $\phi_1 \wedge \phi_2 \in CL(\phi)$ then $\phi_1, \phi_2 \in CL(\phi)$,
- if $\bigcirc\psi \in CL(\phi)$ then $\psi \in CL(\phi)$ and for all $N' \subseteq \mathcal{N}$, $N' \in CL(\phi)$,
- if $\langle r\rangle\psi \in CL(\phi)$ then $\psi \in CL(\phi)$ and $dom(r) \in CL(\phi)$,
- if $[r]\psi \in CL(\phi)$ then $\neg\langle r\rangle true, \langle r\rangle\psi \in CL(\phi)$,
- if $\phi_1 \; U \; \phi_2 \in CL(\phi)$ then $\phi_1, \phi_2, \bigcirc(\phi_1 \; U \; \phi_2) \in CL(\phi)$,
- if $\phi_1 \; R \; \phi_2 \in CL(\phi)$ then $\phi_1, \phi_2, \bigcirc(\phi_1 \; R \; \phi_2) \in CL(\phi)$.

The set $CL(\phi)$ is finite, and its size is linear in the size of the formula ϕ.

The states of the ABAR associated with a formula ϕ are the propositionally and temporally consistent subsets of $CL(\phi)$, the so called *atoms*. Unlike the original Vardi-Wolper construction in [149] which allows only maximal consistent subsets, we allow any downward consistent subset of the closure to be an atom. Formally, we define atoms as follows:

Definition 6.6 An atom $A \subseteq CL(\phi)$ is a set such that

1. $true \in A$ and $false \notin A$,
2. for all $N \in CL(\phi)$, $N \in A$ if and only if for all $N' \neq N$, $N' \notin A$,
3. if $\phi_1 \vee \phi_2 \in A$ then $\phi_1 \in A$ or $\phi_2 \in A$,
4. if $\phi_1 \wedge \phi_2 \in A$ then $\phi_1 \in A$ and $\phi_2 \in A$,
5. if $\phi_1 \; U \; \phi_2 \in A$ then $\phi_2 \in A$ or $\phi_1, \bigcirc(\phi_1 \; U \; \phi_2) \in A$,
6. if $\phi_1 \; R \; \phi_2 \in A$ then $\phi_1, \phi_2 \in A$ or $\phi_2, \bigcirc(\phi_1 \; R \; \phi_2) \in A$,
7. if $[r]\psi \in A$ then $\neg\langle r\rangle true \in A$ or $\langle r\rangle\psi \in A$,
8. if $\langle r\rangle\psi \in A$ then there is $N \supseteq dom(r)$ such that $N \in A$,
9. if $\neg\langle r\rangle true \in A$ then there is $N \neq dom(r)$ such that $N \in A$,
10. if $\bigcirc\psi \in A$ then there is $N \subseteq \mathcal{N}$ that $N \in A$.

Now, we define the generalized ABAR counterpart of every ρLTL+ formula:

Definition 6.7 Let ϕ be a ρLTL+ formula over a finite name set $\mathcal{N}$ and a finite data set $\mathcal{D}$. We define $ABAR(\phi) = \langle Q, Rec_{\mathcal{N}}(\mathcal{D}), \rightarrow, Q_0, \mathcal{F}, V \rangle$ to be the generalized augmented Büchi automaton of records such that

- Q is the set of all atoms of ϕ,
- Q_0 is the set of atoms containing ϕ itself,
- the labeling function $V : Q \rightarrow (2^{\mathcal{N}} \rightarrow \{true, false\})$ is defined such that for all $q \in Q$ and $N \subseteq \mathcal{N}$, $V(q)(N) = true$ if and only if $N \in q$.
- the transition relation $\rightarrow \subseteq Q \times Rec_{\mathcal{N}}(\mathcal{D}) \times Q$ is defined such that $\forall p, q \in Q$ and for all $r \in Rec_{\mathcal{N}}(\mathcal{D})$ such that $dom(r) \subseteq N$ where N is the only set for which $V(q)(N) = true$, there is transition $q \xrightarrow{r} p$ if and only if
  - for all $\langle r' \rangle \psi \in q$, $r' = r$ and $\psi \in p$,
  - for all $\bigcirc \psi \in q$, $\psi \in p$,
  - for all $\neg \langle r' \rangle true \in q$, $r \neq r'$,

  (if for all $N \subseteq \mathcal{N}$, $V(q)(N) = false$ then only $r = \tau$ should be considered),
- $\mathcal{F}$ consists of the accepting sets

$$F_{\alpha \, U \, \beta} = \{q \in Q | \alpha \; U \; \beta \notin q \text{ or } \beta \in q\}$$

  for each $\alpha \, U \, \beta \in CL(\phi)$.

Before showing that the above construction is sound and complete, note that the resulting automaton is exactly an augmented BAR, namely the labeling function is so defined that for every transition $q \xrightarrow{r} p$ the label of q implies the weakest precondition of r. Also, note that each atom and thus each state q of the constructed automaton contains at most one of the sets of the form N. Thus, in each state q of the automaton there is at most one set N whose label is $true$, namely $V(q)(N) = true$.

The following theorem shows the correctness of the above construction:

Theorem 6.2 Let ϕ be a ρLTL+ formula over a names set $\mathcal{N}$ and a data set $\mathcal{D}$. The language accepted by $ABAR(\phi)$ is the set of all models of ϕ:

$$L(ABAR(\phi)) = \| \; \phi \; \| \; .$$

Proof.

Soundness ($L(ABAR(\phi)) \subseteq \| \; \phi \; \|$). Let $M = N_0 r_0 N_1 r_1 \cdots \in L(ABAR(\phi))$ be a guarded string accepted by the accepting computation $\pi = q_0 r_0 q_1 r_1 \cdots$ in automaton $ABAR(\phi)$. We show that for all $i \geq 0$ and every ρLTL formula ψ, if $\psi \in q_i$ then $M^i = N_i r_i N_{i+1} r_{i+1} \cdots \models \psi$. Using this fact and because $\phi \in q_0$ we obtain that $M \models \phi$ and thus $M \in \| \; \phi \; \|$.

The fact that for all $i \geq 0$ and every ρLTL+ formula ψ, if $\psi \in q_i$ then $M^i \models \psi$ is shown by induction on the structure of the formula ψ.

Base cases:

- $\psi = N$. Because $N \in q_i$, $V(q_i)(N) = true$. Using the facts that there is at most one set N for which $V(q_i)(N) = true$ and M is accepted by $ABAR(\phi)$, we know that $N_i = N$. Thus, $M^i \models \psi$.

- $\psi = \neg\langle r\rangle true$. Because $\neg\langle r\rangle true \in q_i$ we have $r_i \neq r$. Therefore, $M^i \models \psi$.

Inductive steps:

- $\psi = \psi_1 \vee \psi_2$. Because $\psi_1 \vee \psi_2 \in q_i$ using the definition of atoms we know that $\psi_1 \in q_i$ or $\psi_2 \in q_i$. By the induction hypothesis, $M^i \models \psi_1$ or $M^i \models \psi_2$. Thus, $M^i \models \psi$.

- $\psi = \psi_1 \wedge \psi_2$. The proof of this case is very similar to the previous case.

- $\psi = \bigcirc\psi_1$. Because $\bigcirc\psi_1 \in q_i$ using the definition of the transition relation we know that $\psi_1 \in q_{i+1}$. By the induction hypothesis, $M^{i+1} \models \psi_1$. Thus, $M^i \models \psi$.

- $\psi = \langle r\rangle\psi_1$. Because $\langle r\rangle\psi_1 \in q_i$ using the definition of the transition relation we know that $r_i = r$, $dom(r_i) \subseteq N_i$ and $\psi_1 \in q_{i+1}$. By the induction hypothesis, $M^{i+1} \models \psi_1$. Thus, $M^i \models \langle r\rangle\psi_1$.

- $\psi = [r]\psi_1$. Because $[r]\psi_1 \in q_i$ using the definition of atoms we know that $\langle r\rangle\psi_1 \in q_i$ or $\neg\langle r\rangle true \in q_i$:

  - If $\langle r\rangle\psi_1 \in q_i$ then by the proof of the previous case we know that $M^i \models \langle r\rangle\psi_1$. Thus, $M^i \models [r]\psi_1$.
  - If $\neg\langle r\rangle true \in q_i$ then using the base case, $M^i \models \neg\langle r\rangle true$. Thus, $M^i \models [r]\psi_1$.

- $\psi = \psi_1 \, U \, \psi_2$. Because $q_i\, q_{i+1} \cdots$ is an accepting run in the automaton, there is $k \geq i$ such that $q_k \in F_{\psi_1 \, U \, \psi_2}$. Let j be the least such k:

  - If $j = i$, then since $\psi_1 \, U \, \psi_2 \in q_i$ and $q_i \in F_{\psi_1 \, U \, \psi_2}$ using the definition of the final states we must have $\psi_2 \in p_i$. By the induction hypothesis, $M^i \models \psi_2$. Thus, $M^i \models \psi_1 \, U \, \psi_2$.
  - If $j > i$ then for all $i \leq l < j$, $\psi_1 \, U \, \psi_2 \in q_l$ and $\psi_2 \notin q_l$. Since q_i is an atom, $\psi_1 \in q_l$. By the induction hypothesis, for all $i \leq l < j$, $M^l \models \psi_1$. Now, $\psi_1 \, U \, \psi_2 \in q_{j-1}$ and $\psi_2 \notin q_{j-1}$, thus by the definition of atoms $\bigcirc(\psi_1 \, U \, \psi_2) \in q_{j-1}$. Therefore, $\psi_1 \, U \, \psi_2 \in q_j$. Since $q_j \in F_{\psi_1 \, U \, \psi_2}$ we should have $\psi_2 \in q_j$. By the induction hypothesis, $M^j \models \psi_2$. Thus we have for all $i \leq l < j$, $M^l \models \psi_1$ and $M^j \models \psi_2$. Therefore, $M^i \models \psi_1 \, U \, \psi_2$.

- $\psi = \psi_1 \, R \, \psi_2$. We have $\psi_1 \, R \, \psi_2 \in q_i$. By the definition of atoms, one of the following cases happens:

  - For all $j \geq i$, $\psi_2 \in q_j$ and $\psi_1 \, R \, \psi_2 \in q_j$. In this case by the induction hypothesis, for all $j \geq i$, $M^j \models \psi_2$. Thus, $M^i \models \psi$.
  - There is $j \geq i$ such that for all $i \leq l < j$, $\psi_2 \in q_l$, $\psi_1 \, R \, \psi_2 \in q_l$ and $\psi_1, \psi_2 \in q_j$. Then, for all $i \leq l < j$, $M^l \models \psi_2$ and $M^j \models \psi_1$ and $M^j \models \psi_2$. Thus, $M^i \models \psi$.

Completeness ($\| \phi \|_{\subseteq} \subseteq L(ABAR(\phi))$). Let the guarded string $M = N_0 r_0 N_1 r_1 \cdots$ be a model of ϕ. We show that $M \in L(ABAR(\phi))$. For this purpose for every $i \geq 0$ we define the set of formulas q_i as follows:

$$q_i = \{\psi \in CL(\phi) \mid M^i \models \psi\}.$$

Now we show that q_i's are atoms for ϕ and $\pi = q_0 r_0 q_1 r_1 \cdots$ is an accepting initial computation for M in $ABAR(\phi)$.

First note that each q_i satisfies the conditions to be an atom for ϕ (see Definition 6.6):

- (1) Obviously for all i, $true \in q_i$.
- (2) Let $N \in q_i$. Since $M^i \models N$, $N_i = N$. Thus, for all $N' \neq N$, $N' \neq N_i$. Therefore, for all $N' \neq N$, $M^i \not\models N'$. So, for all $N' \neq N$, $N \notin q_i$.
- (3) Let $\psi_1 \vee \psi_2 \in q_i$. Thus, $M^i \models \psi_1 \vee \psi_2$. Using the semantics of formulas, we have $M^i \models \psi_1$ or $M^i \models \psi_2$. Also, $\psi_1, \psi_2 \in CL(\phi)$. Thus, $\psi_1, \psi_2 \in q_i$.

The other conditions can be checked similarly.

Now, we show that for all $i \geq 0$, $q_i \xrightarrow{r_i} q_{i+1}$ is a transition in the automaton. For this purpose, we show that it satisfies the conditions of the transition relation in Definition 6.7. First note that since $M \models \phi$, we have the fact that $\forall i \geq 0, dom(r_i) \subseteq N_i$. Now we examine the conditions:

- Let $\langle r' \rangle \psi \in q_i$. Then, $M^i \models \langle r' \rangle \psi$. Thus, $r' = r_i$ and $M^{i+1} \models \psi$. Therefore, $r' = r_i$, $\psi \in q_{i+1}$, and $N_i \in q_i$ with $N_i \supseteq dom(r_i)$.
- Let $\bigcirc \psi \in q_i$. Then, $M^i \models \bigcirc \psi$. Thus, $M^{i+1} \models \psi$. Therefore, $\psi \in q_{i+1}$ and $N_i \in q_i$.
- Let $\neg \langle r' \rangle true \in q_i$. Then, $M^i \models \neg \langle r' \rangle true$. So $r_i \neq r'$ or $M^{i+1} \not\models true$. The second choice is impossible. Thus, $r_i \neq r'$.

So far, we have shown that π is a computation in the automaton $ABAR(\phi)$. Also, we know that π is an initial computation, because we have $M \models \phi$, thus $\phi \in q_0$. Therefore, $q_0 \in Q_0$.

Now, we show that π is a computation for the guarded string M. This fact is true because for each $i \geq 0$ the only $N \subseteq \mathcal{N}$ such that $M^i \models N$ and $N \in q_i$ is N_i. Thus, $\forall i \geq 0$, $V(q_i)(N_i) = true$. Thus, π is an initial computation for M.

Our proof is complete if we show that π is an accepting computation, namely that it meets at least one of the final states of every set of final states infinitely often. Suppose that it is not the case. Then, there is $j \geq 0$ such that for a formula of the form $\alpha U \beta$, we have $\forall k \geq j, q_k \notin F_{\alpha U \beta}$. Thus, $\forall k \geq j, \alpha U \beta \in q_k$ and $\beta \notin q_k$. So, $\forall k \geq j$, $M^k \models \alpha U \beta$ and $M^k \not\models \beta$. This contradicts the fact that $M^j \models \alpha U \beta$ since β never gets satisfied.

Therefore, π is an accepting initial computation for M in the automaton $ABAR(\phi)$. Thus, $M \in L(ABAR(\phi))$. □

The result reported in Theorem 6.2 can be used for an automata based procedure for model checking Reo connectors. Given an ABAR model B of a Reo connector, and a ρLTL formula ϕ over the same set of port names $\mathcal{N}$ and data set $\mathcal{D}$, saying that $B \models \phi$ is equivalent to check whether $L(B)$ does not contain any models of $\neg\phi$. From the above theorem, this is equivalent

to check if $L(B) \cap L(ABAR(\neg\phi)) = \emptyset$. Therefore, if this intersection is empty, it proves that the connector B satisfies the property ϕ. Otherwise, every element of this intersection is a counterexample. Recall that intersecting two Büchi automata is just a simple extension of the product construction, and checking for emptiness is decidable [138]. The complexity of the model checking procedure is linear in the number of states of B and exponential in the length of the formula ϕ [145].

6.3 On-the-fly translation

In this section, we sketch an algorithm to construct the ABAR for a ρLTL on-the-fly by generating the state space of the automaton incrementally, as required by the model checking procedure. The algorithm is a generalization of the on-the-fly approach proposed in [59] for standard LTL and extended with modalities for actions in a similar way as in [107].

6.3.1 A description of the algorithm

The algorithm works by building a graph underlying the ABAR to be defined for a formula ϕ. The nodes are labeled by sets of formulas that are obtained by decomposing them into their sub-formulas according to their boolean structures. Temporal formulas are handled by just deciding what should be true at the node and what must be true at every next node. For an on-the-fly construction of the graph, we need to store some information at every node of the graph. More specifically, a node is a structure containing the following fields:

1. *Name*. A string which is the name of the node.

2. *Incoming*. A set of elements of the form (q, X) where q is a node and $X \subseteq Rec_{\mathcal{N}}(\mathcal{D})$. Intuitively, a pair $(q, X) \in Incoming$ represents a transition from q to the current node labeled by the record r, for $r \in X$. A special element $init$ is used to mark initial nodes.

3. *Old*. A set of formulas that have already been processed and hold in the current node (provided the properties in *New* are satisfied).

4. *New*. A set of formulas that have not yet been processed and that have to be satisfied in the current node

5. $Next^{+}$. A set of next-state formulas that this node satisfies. They assert formulas that must be satisfied in any successor node.

6. $Next^{-}$. A set of records that are *not* allowed to label outgoing transitions from the current node.

The algorithm for building the graph of the automaton satisfying a ρLTL+ formula ϕ stores the nodes of the graph already computed in the list *Nodes_Set*. For all nodes in this list, it holds that the *New* field is empty. In this case, *Old* contains the set of formulas that the node

satisfies. The full graph can then be constructed using the information in the *Incoming* field of each node.

The algorithm starts with a node q_0 with its *New* field set to $\{\phi\}$, $Incoming = \{init\}$ and with all other fields initially set to empty. When processing a node q the algorithm removes a formula ψ from its *New* field and tries all possible ways to satisfy it, by looking at the syntactic structure of ψ:

- If $\psi = N$, where $N \subseteq \mathcal{N}$ then if there is N' ($N' \neq N$) in Old the node q is discarded because it contains a contradiction. Otherwise ψ is added to Old.

- If $\psi = \psi_1 \wedge \psi_2$ then both ψ_1 and ψ_2 are added to New because they both need to be satisfied in the node q.

- If $\psi = \psi_1 \vee \psi_2$ then a new node is created with the same fields as the current node q. Then ψ_1 is added to the *New* field of one node and ψ_2 to the other. The two nodes correspond to the two ways ψ can be satisfied.

- If $\psi = \bigcirc\varphi$ or $\psi = \langle r\rangle\varphi$ then ψ is added to the $Next^+$ field of the current node.

- The case where $\psi = [r]\varphi$ is novel with respect to the algorithm in [59]. Because $\psi \equiv \neg\langle r\rangle true \vee \langle r\rangle\varphi$, a new node is created with the same fields as the current node. The record r is added to the field $Next^-$ of one node, whereas the formula $\langle r\rangle\phi$ is added to the $Next^+$ field of the other node.

- If $\psi = \psi_1 \, U \, \psi_2$ then a new node is created with the same fields as the current node q. Because $\psi \equiv \psi_2 \vee (\psi_1 \wedge \bigcirc\psi)$, the formula ψ_2 is added to the *New* field of one node, while ψ_1 and $\bigcirc\psi$ are added to the fields *New* and $Next^+$ of the other node, respectively.

- If $\psi = \psi_1 \, R \, \psi_2$ then a new node is created with the same fields as the current node q. Because $\psi \equiv \psi_2 \wedge (\psi_1 \vee \bigcirc\psi)$, the formula ψ_2 is added to the *New* field of both nodes, ψ_1 is added to the *New* field of one node and $\bigcirc\psi$ to the $Next^+$ of the other node.

When the New field is empty, the current node is ready to be added to the set *Nodes_Set*. If there is already another node in the list with the same *Old*, $Next^+$, and $Next^-$ fields, then the only *Incoming* field of the copy that already exists needs to be updated by adding the edges in the *Incoming* field of the current node.

If there is no such node, then the current node is added to the list *Nodes_Set*, but different than the case of the original algorithm [59], there are several ways how a current node is formed for its successors: if the information about the labels of the outgoing transitions is inconsistent (i.e. $Next^+$ is empty or there is a record r in $Next^-$ that is also used in a next state formula $\langle r\rangle\varphi$ in $Next^+$ or there are two formulas $\langle r\rangle\varphi$ and $\langle r'\rangle\varphi'$ in $Next^+$ with $r \neq r'$) then there is no successor node.

Otherwise, if the formulas in the $Next^+$ field of the current node are only of type $\bigcirc\varphi$, then a successor node is created with a transition from the current node to the new node labeled by r for each record r not in the $Next^-$ field of the current node. The formulas to be satisfied by this new node are all formulas in the $Next^+$ field of the current node stripped off of their next state modality.

η	New_1	New_2	$Next_1$
$\psi_1 \vee \psi_2$	$\{\psi_1\}$	$\{\psi_2\}$	$\emptyset$
$\psi_1 \, U \, \psi_2$	$\{\psi_1\}$	$\{\psi_2\}$	$\{\bigcirc(\psi_1 \, U \, \psi_2)\}$
$\psi_1 \, R \, \psi_2$	$\{\psi_2\}$	$\{\psi_1, \psi_2\}$	$\{\bigcirc(\psi_1 \, R \, \psi_2)\}$

Table 6.1: Definitions of New_1, New_2 and $Next_1$ functions.

Finally, in the remaining case that there is a formula $\langle r \rangle \phi$ in $Next^+$ with no r in the $Next^-$ field, then a successor node is created with a transition labeled by r from the current node to the new node. As in the previous case, the formulas to be satisfied by this new node are all formulas in the $Next^+$ field of the current node stripped off of their next state modality.

6.3.2 The algorithm in detail

In this section we present the pseudo code of the algorithm sketched in the previous subsection. The algorithm constructs a graph of nodes and is called *Create_Graph*. It uses the function *Expand* which processes every node and updates the list of nodes *Nodes_Set*. For conciseness, we use functions New_1, New_2 and $Next_1$ which are defined in Table 6.1.

```
Creat_Graph(φ)
1.   return(Expand([Name: = New_Name(), Incoming: = {Init},
2.          New: = {φ}, Old: = ∅, Next+: = ∅, Next−: = ∅], ∅));
```

```
Expand(Node, Nodes_Set)
1. if New(Node) = ∅
2.    then if ∃ND ∈ Nodes_Set with Old(ND) = Old(Node) and
3.         Next+(ND) = Next+(Node) and Next−(ND) = Next−(Node)
4.         then { Incoming(ND): = Incoming(ND) ∪ Incoming(Node);
5.                return(Nodes_Set); }
6.         else if (∃ ⟨r⟩φ, ⟨r′⟩ψ ∈ Next+(Node) with r ≠ r′) or
7.                 (∃r ∈ Next−(Node), ⟨r⟩φ ∈ Next+(Node))
8.                 then return(Nodes_Set ∪ {Node})
9.                 else if ∄⟨r⟩φ ∈ Next+(Node)
10.                        then return(Expand([Name: = New_Name(),
11.                             Incoming: = {(Name(Node),
12.                                  Rec_N(D)\Next−(Node))}
13.                             New: = StriptNexts(Next+(Node))
14.                             old: = ∅
15.                             Next+: = ∅, Next−: = ∅],
16.                             Nodes_Set ∪ {Node}))
17.                        else return(Expand([Name: = New_Name(),
```

$Incoming := \{(Name(Node), \{r\})\},$
$New := StriptNexts(Next^+(Node)), Old := \emptyset$
$Next^+ := \emptyset, Next^- := \emptyset], Nodes\_Set \cup \{Node\}))$

else let $\eta \in New(Node)$
  then $New(Node) := New(Node) \backslash \{\eta\}$;
  switch
    case $\eta = N$ or $\eta = true$ or $\eta = false$:
      if $\eta = false$ or $\exists N' \in Old(Node)$ that $N' \neq N$
        then return$(Nodes\_Set)$
        else $\{Old(Node) := Old(Node) \cup \{\eta\}$;
          return$(Expand(Node, Nodes\_Set))\}$

    case $\eta = \phi\, U\, \psi$ or $\eta = \phi\, R\, \psi$ or $\eta = \phi \vee \psi$:
      $Node_1 := [Name := New\_Name(), Incoming := Incoming(Node),$
        $New := New(Node) \cup (New_1(\eta) \backslash Old(Node)),$
        $Old := Old(Node) \cup \{\eta\},$
        $Next^+ := Next^+(Node) \cup Next_1(\eta), Next^- := Next^-(Node)]$
      $Node_2 := [Name := New\_Name(), Incoming := Incoming(Node),$
        $New := New(Node) \cup (New_2(\eta) \backslash Old(Node)),$
        $Old := Old(Node) \cup \{\eta\},$
        $Next^+ := Next^+(Node), Next^- := Next^-(Node)]$
      return$(Expand(Node_2, Expand(Node_1, Nodes\_Set)))$

    case $\eta = \phi \wedge \psi$:
        $Old(Node) := Old(Node) \cup \{\eta\},$
        $New(Node) := New(Node) \cup (\{\phi, \psi\} \backslash Old(Node))$
      return$(Expand(Node, Nodes\_Set))$

    case $\eta = X\phi$ or $\eta = \langle r \rangle \phi$:
        $Old(Node) := Old(Node) \cup \{\eta\},$
        $Next^+(Node) := Next^+(Node) \cup (\eta\}$
      return$(Expand(Node, Nodes\_Set))$

    case $\eta = [r]\phi$:
      $Node_1 := [Name := New\_Name(), Incoming := Incoming(Node),$
        $New := New(Node), Old := Old(Node) \cup \{\eta\},$
        $Next^+ := Next^+(Node),$
        $Next^- := Next^-(Node) \cup \{r\}]$
      $Node_2 := [Name := New\_Name(), Incoming := Incoming(Node),$
        $New := New(Node), Old := Old(Node) \cup \{\eta\},$
        $Next^+ := Next^+(Node) \cup \{\langle r \rangle \phi\},$
        $Next^- := Next^-(Node)]$
      return$(Expand(Node_2, Expand(Node_1, Nodes\_Set)))$.

In the above algorithm, for each set of ρLTL+ formulas S we define:

$$StripNexts(S) = \{\phi \mid \bigcirc \phi \in S \text{ or } \langle r \rangle \phi \in S \text{ for some } r \in Rec_{\mathcal{N}}(\mathcal{D})\}.$$

6.3.3 The ABAR defined by the algorithm

The above sketched algorithm defines for each ρLTL+ formula ϕ a generalized ABAR $B(\phi)$ over port names $\mathcal{N}$ and data set $\mathcal{D}$ as follows. The states are the set of nodes in *Nodes_Set*, as returned by the algorithm. Every node with the *Init* in its *Incoming* field is an initial state. In each node (state) n, if there is (only one) $N \in Old(n)$ then the valuation function $V_B(n)$ assigns $true$ only to N, otherwise for all $N \subseteq \mathcal{N}$, $V_B(n)(N)$ is $true$. Note that for each node n at most one set $N \subseteq \mathcal{N}$ is in $Old(n)$ and for this N we have $V(n)(N) = true$. The transitions of the form $n \xrightarrow{r} n'$ are exactly those where $r \in X$ for (n, X) in the *Incoming* field of n' and $dom(r) \subseteq N$ and N is the only N for which $V(n)(N) = true$. Finally, for each sub-formula $\alpha U \beta$ of ϕ we define a set of accepting states $F_{\alpha U \beta}$ containing all nodes n for which $\alpha \ U \ \beta \notin t(q)$ or $\beta \in t(n)$, where $t(n)$ is the union of the fields *Old*, *New*, $Next^+$ and the set containing $\neg\langle r \rangle true$ for each record r in the $Next^-$ field of the node n:

$$t(n) = New(n) \cup Old(n) \cup Next^+(n) \cup \{\neg\langle r \rangle true \mid r \in Next^-(n)\}.$$

(Note that here we require function t to be defined only on finished nodes that belong to $Node\_Set$ and whose New field is empty. Our definition is more general because we want to use it in the next proofs.)

More formally, we define ABAR $B(\phi)$ as follows:

Definition 6.8 Let ϕ be a ρLTL+ formula over a finite name set $\mathcal{N}$ and a finite data set $\mathcal{D}$. We define $B(\phi) = \langle Q_B, Rec_{\mathcal{N}}(\mathcal{D}), \rightarrow_B, Q_{0B}, \mathcal{F}_B, V_B \rangle$ to be the generalized augmented Büchi automaton of records such that

- $Q_B = Node\_Set$ is the set of all nodes generated by the algorithm,
- $Q_{0B} = \{n \mid n \in Node\_Set \text{ and } Init \in Incoming(n)\}$,
- the labeling function $V_B : Node\_Set \rightarrow (2^{\mathcal{N}} \rightarrow \{true, false\})$ is defined such that for all $n \in Node\_Set$, if there exists an $N \subseteq \mathcal{N}$ such that $N \in Old(n)$ then $V_B(n)(N) = true$ otherwise $\forall N \subseteq \mathcal{N}$, $V_B(n)(N) = true$,
- the transition relation $\rightarrow_B \subseteq Node\_Set \times Rec_{\mathcal{N}}(\mathcal{D}) \times Node\_Set$ is defined such that $\forall n, n' \in Node\_Set$ and $\forall r \in Rec_{\mathcal{N}}(\mathcal{D})$, we have $n \xrightarrow{r}_B n'$ if and only if $\exists (n, X) \in Incoming(n')$ such that $r \in X$ and $V_B(n)(dom(r)) = true$,
- $\mathcal{F}'_B$ consists of the accepting sets

  $$F'_{\alpha U \beta} = \{n \in Node\_Set \mid \alpha \ U \ \beta \notin t(n) \text{ or } \beta \in t(n)\}$$

  for each $\alpha U \beta \in CL(\phi)$.

Theorem 6.3 Let ϕ be a ρLTL+ formula over a finite names set $\mathcal{N}$ and a finite data set $\mathcal{D}$ and $B(\phi)$ be the ABAR produced by the above algorithm. Then, the accepted language of $B(\phi)$ is the set of all models of ϕ, that is

$$L(B(\phi)) = \| \phi \| \ .$$

Proof. Obviously this theorem is correct if we show that both automata $ABAR(\phi)$ (which we construct it globally) and $B(\phi)$ accept precisely the same language, namely $L(ABAR(\phi)) = L(B(\phi))$. We will prove this fact in Section 6.3.4 after presenting some lemmas. □

As explained before, a formula about a Reo connector can be verified by (1) constructing the ABAR translation of negation of the formula, (2) constructing the product automaton using the ABAR model of the Reo connector, and (3) checking the resulting automaton for emptiness.

6.3.4 Proof of the correctness

In this section we prove Theorem 6.3 in detail. As we mentioned in the theorem's proof scheme, we will show that for a ρLTL+ formula ϕ both automata $ABAR(\phi)$ (which we construct globally as in Definition 6.7) and $B(\phi)$ (which we construct by the on-the-fly algorithm as in Definition 6.8) accept precisely the same language, namely $L(ABAR(\phi)) = L(B(\phi))$.

Soundness ($L(B(\phi)) \subseteq L(ABAR(\phi))$).

First we present a simple lemma:

Lemma 6.4 Let $n \in Node\_Set$ be a node generated by the algorithm and $t(n)$ be the set of formulas for n as we defined in section 6.3.3. Also, define the set of formulas A^n as:

$$A^n = t(n) \cup \{true\}.$$

Then for each node n, A^n is an atom of ϕ.

Proof. First, clearly A^n is a subset of $CL(\phi)$. Now, refer to conditions (1) to (7) in Definition 6.6 which must be satisfied for a set of formulas to be an atom. Simply, we can show that all of them are satisfied by A^n. Based on the definition of A^n we know that $true \in A^n$ and since $false$ is never part of any node (see lines 25-26 of the *Expand* algorithm in Section 6.3.2), $false \notin A^n$. Thus A^n satisfies condition (1). Lines 24-28 of the *Expand* algorithm show that for all $N \subseteq \mathcal{N}$, $N \in A^n$ if and only if for all $N' \neq N$, $N' \notin A^n$. Thus condition (2) is also satisfied by A^n. For conditions (3) to (7), note that whenever a formula on the left hand side of these conditions are inserted into Old, the required formulas get inserted into New and these formulas will eventually get into Old and hence into A^n. For example, for condition (3), if $\phi_1 \vee \phi_2 \in A^n$ then it should be in $t(n)$. Thus, when processed, either ϕ_1 or ϕ_2 gets inserted into New. But each formula inserted into New will eventually get into all finished nodes under this node. Thus, A^n will have either ϕ_1 or ϕ_2. Similarly, other conditions can be verified. □

Now, we can prove the soundness lemma:

Lemma 6.5 Let ϕ be a ρLTL+ formula. Then,

$$L(B(\phi)) \subseteq L(ABAR(\phi)).$$

Proof. Let $M = N_0 r_0 N_1 r_1 \cdots \in L(B(\phi))$ be accepted by an accepting computation $\sigma = n_0 r_0 n_1 r_1 \cdots$ in $B(\phi)$. Thus, we know that $\forall i$, $V_B(n_i)(N_i) = true$. Consider $A_i = t(n_i) \cup \{true\}$. First, by Lemma 6.4, for all i, A_i is an atom. Also, it is clear that in $ABAR(\phi)$, $\forall i$, $V(n_i)(N_i) = true$.
We will show that $\pi = A_0 r_0 A_1 r_1 \cdots$ is an accepting computation for M in $ABAR(\phi)$:
1- We know that $t(n_0) \subseteq A_0$ and $\phi \in t(n_0)$. Hence, $\phi \in A_0$. Thus, $A_0 \in Q_0$.
2- Now consider the transition $n_i \xrightarrow{r_i}_B n_{i+1}$ in $B(\phi)$. First note that $\forall i, dom(r_i) \subseteq N_i$. Since in the *Expand* algorithm n_{i+1} was spawned from n_i, it is clear that $\langle r \rangle \psi \notin A_i$ (where $r \neq r_i$) and $r_i \notin Next^-(n_i)$. Now, for all $\bigcirc\psi \in t(n_i)$, $\psi \in t(n_{i+1})$ (by construction) and for all $\langle r_i \rangle \psi \in t(n_i)$, $\psi \in t(n_{i+1})$. Hence we have:

- for all $\langle r \rangle \psi \in A_i$, $r = r_i$,
- for all $\bigcirc\psi \in A_i$, $\psi \in A_{i+1}$,
- for all $\langle r_i \rangle \psi \in A_i$, $\psi \in A_{i+1}$ and
- $\neg\langle r_i \rangle true \notin A_i$.

Therefore, $\forall i$, $A_i \xrightarrow{r_i} A_{i+1}$ is a transition in $ABAR(\phi)$.
3- Also, if $n_i \in F'_{\alpha U \beta}$, then either $\alpha U \beta \notin t(n_i)$ or $\beta \in t(n_i)$. If $\alpha U \beta \notin t(n_i)$, then $\alpha U \beta \notin A_i$. If $\beta \in t(n_i)$, then $\beta \in A_i$. Thus, $A_i \in F_{\alpha U \beta}$. Since σ meets each set $F'_{\alpha U \beta}$ infinitely often, π meets each set $F_{\alpha U \beta}$ infinitely often.

By the above 1-3 facts, we conclude that π is an initial accepting computation for M in $ABAR(\phi)$. Therefore, $M \in L(ABAR(\phi))$. □

Completeness ($L(ABAR(\phi)) \subseteq L(B(\phi))$).
Now, we show that for every ρLTL+ formula ϕ, each model accepted by $ABAR(\phi)$ is also accepted by $B(\phi)$. We do this by mapping accepting computations of $ABAR(\phi)$ to accepting computations over the algorithm automaton $B(\phi)$. First we present a definition.

Let $n \in Node\_Set$ be a node constructed by the *Expand* algorithm starting with formula ϕ. We define $f(n)$ to be the set of all atoms for ϕ that can extend node n. More formally:

$$f(n) = \{A \mid \text{A is an atom for } \phi \text{ and } t(n) \subseteq A\}.$$

Now, we give some lemmas that will lead us to the proof of completeness.

Lemma 6.6 When a node n is split in the algorithm into two nodes n_1 and n_2 (lines 30-39 and 51-58) the following holds:

$$f(n) = f(n_1) \cup f(n_2).$$

Similarly, when a node n is updated to become a new node n' (lines 24-28 and 41-49) the following holds:

$$f(n) = f(n').$$

Proof. The proof is obvious by tracing of the algorithm and calculating $t(n')$ for every new node n' using $t(n)$. □

When a node n is spawned with its Old, $Next^+$ and $Next^-$ fields empty, the algorithm starts processing the formulas in New. From this point onwards, one can view the algorithm as creating a tree rooted at n. The tree gets modified as formulas in New are processed. When a node (which must be a leaf) splits, we can view this as a creation of two children, since the algorithm will start expanding each child eventually. When a node is processed and its fields get modified, we view this as the creation of a single child. When a node is abandoned, we mark the node *bad*, no new edge comes out of this node. Finally a tree is produced which is rooted at n. We will call a leaf node *good* if it is not bad. Note that the good leaves at the end of the construction have their New fields empty and are exactly the nodes added to $Nodes\_Set$. The proof of the following lemma is by induction on the number of steps that have been performed so far by the algorithm, which have modified the tree.

Lemma 6.7 Let n be a node and A an atom such that $A \in f(n)$. At any point in the construction of the tree rooted at node n, there is a good leaf n' such that $A \in f(n')$ and for all formulas of the form $\alpha U \beta$ in $CL(\phi)$, if $A \in F_{\alpha U \beta}$ then $\alpha U \beta \notin Old(n')$ or $\beta \in t(n')$.

Proof. The proof can be simply done by induction on the number of steps in the algorithm that have changed the tree so far. □

Lemma 6.8 Let n be a rooted node and A an atom such that $A \in f(n)$. Then, there is a good leaf n' in the tree rooted by n such that $A \in f(n')$ and for all formulas of the form $\alpha U \beta$ in $CL(\phi)$, if $A \in F_{\alpha U \beta}$ then $n' \in F'_{\alpha U \beta}$.

Proof. By Lemma 6.7, at the end of the construction of the tree, there exists a leaf n' such that $A \in f(n')$ and for all formula of the form $\alpha U \beta$ in $CL(\phi)$, if $A \in F_{\alpha U \beta}$ then $\alpha U \beta \notin Old(n')$ or $\beta \in t(n')$. Since $New(n') = \emptyset$ and n' is a good leaf, thus for all formula of the form $\alpha U \beta$ in $CL(\phi)$, $\alpha U \beta \notin t(n')$ or $\beta \in t(n')$. Therefore, using Definition 6.8, $n' \in F'_{\alpha U \beta}$. □

Lemma 6.9 Let n be a node, A an atom that $A \in f(n)$, and let $A \xrightarrow{r} A'$ be a transition in $ABAR(\phi)$. Then, there is an $n' \in Node\_Set$ such that there is transition $n \xrightarrow{r} n'$ in $B(\phi)$ where, $A' \in f(n')$ and for all formulas of the form $\alpha U \beta$ in $CL(\phi)$, if $A' \in F_{\alpha U \beta}$ then $n' \in F'_{\alpha U \beta}$.

Proof. First, it is clear that there is no $\neg\langle r \rangle true$ or $\langle r' \rangle \psi$ $(r' \neq r)$ in $t(n)$, for otherwise it would belong to A as well and $A \xrightarrow{r} A'$ will not be possible in $ABAR(\phi)$. Hence, after n was processed by the algorithm, a node m must have been spawned with its New field set contains the formulas in $Next^+(n)$ stripped of their $\bigcirc$'s and $\langle r \rangle$'s. Now, if$\psi \in t(m)$ then $\bigcirc \psi \in t(n)$ or $\langle r \rangle \psi \in t(n)$. Then, $\bigcirc \psi \in A$ or $\langle r \rangle \psi \in A$. Thus, $\psi \in A'$. Hence, $t(m) \subseteq A'$ and $A' \in t(m)$. By Lemma 6.8, there exists a good leaf, and hence a node in $Node\_Set$, say n', in the tree rooted at m, such that $A' \in f(n')$ and for all formulas of the form $\alpha U \beta$ in $CL(\phi)$, if $A' \in F_{\alpha U \beta}$ then $n' \in F'_{\alpha U \beta}$. Also, there exists $(n, X) \in Incoming(n')$ such that $r \in X$. Hence $n \xrightarrow{r} n'$ in $B(\phi)$. This completes the proof. □

Lemma 6.10 Let A_0 be an initial atom (state) in $ABAR(\phi)$, namely $A_0 \in Q_0$. Then, there is a node n_0 in $B(\phi)$, namely $n_0 \in Q_{0B}$, such that $A_0 \in f(n_0)$.

Proof. The algorithm starts with a node m with $New(m) = \{\phi\}$. Since $A_0 \in Q_0$, $\psi \in A_0$. Thus, $t(m) \subseteq A_0$ and $A_0 \in f(m)$. Lemma 6.8 guarantees the existence of some good leaf n_0 which hence gets into $Node\_Set$, in the tree rooted at m such that $A_0 \in f(n_0)$. Also, any leaf of the tree rooted at m has $Init \in Incoming$ and hence $n_0 \in Q_{0B}$. □

Lemma 6.11 Let ϕ be a ρLTL+ formula. Then,

$$L(ABAR(\phi)) \subseteq L(B(\phi)).$$

Proof. Let $M = N_0 r_0 N_1 r_1 \cdots \in L(ABAR(\phi))$ and let $\pi = A_0 r_0 A_1 r_1 \cdots$ be an accepting initial computation for it in $ABAR(\phi)$. We exhibit an accepting initial computation of $B(\phi)$ that accepts M. First, by Lemma 6.10, there exists $n_0 \in Q_{0B}$ such that $A_0 \in f(n_0)$. We construct an accepting initial computation $n_0 r_0 n_1 r_1 \cdots$ for M by using Lemma 6.9 repeatedly, Assume that we have constructed a partial computation $n_0 r_0 \cdots n_i$ so far such that $A_i \in f(q_i)$ (this is true in the beginning for the partial computation n_0). Now since $A_i \xrightarrow{r_i} A_{i+1}$ and $A_i \in f(q_i)$ using Lemma 6.9, there exists n_{i+1} such that $n_i \xrightarrow{r_i} n_{i+1}$ and $A_{i+1} \in f(n_{i+1})$ and for all formulas of the form $\alpha U \beta$ in $CL(\phi)$, if $A_{i+1} \in F_{\alpha U \beta}$ then $n_{i+1} \in F'_{\alpha U \beta}$.

Thus we have extended the partial computation to $n_0 r_0 \cdots n_{i+1}$ with $A_{i+1} \in f(n_{i+1})$. Continuing in this fashion, we can build an infinite initial computation $\rho = n_0 r_0 n_1 r_1 \cdots$ such that $\forall j, A_j \in f(n_j)$ and for all formulas of the form $\alpha U \beta$ in $CL(\phi)$, if $A_j \in F_{\alpha U \beta}$ then $n_j \in F'_{\alpha U \beta}$. Since π meets every final set infinitely often, the computation ρ also meets all final sets infinitely often. Hence ρ is an accepting computation in $B(\phi)$.

The proof is complete if we show that $\forall j$, $V_B(n_j)(N_j) = true$. Since π is an accepting computation for M in $ABAR(\phi)$, we have $\forall j$, $V(A_j)(N_j) = true$. Thus, by the definition of the V function in $ABAR(\phi)$, we know that for all j, $N_j \in A_j$ and $\forall N \neq N_j, N \notin A_j$. Thus, if there exists $N \in Old(n_j)$ then $N \in t(n_j)$. Thus, $N = N_j$ and $V_B(n_j)(N_j) = true$. Otherwise, if there is no $N \in Old(n_j)$, by the definition of V_B, $\forall N$, $V_B(n_j)(N) = true$. Therefore, in both cases we have that $V_B(n_j)(N_j) = true$. Hence the lemma is proved. □

By Lemmas 6.5 and 6.11, we have a complete proof for Theorem 6.3. This shows that our on-the-fly construction of the automaton is correct.