Faculty of Informatics
Masaryk University Brno

Quantitative Model Checking of Systems with Degradation (Full Paper)

by

Jiří Barnat
Ivana Černá
Jana Tůmová

FI MU Report Series FIMU-RS-2009-09

June 2009

Publications in the FI MU Report Series are in general accessible via WWW:

`http://www.fi.muni.cz/reports/`

Further information can be obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic

Quantitative Model Checking of Systems with Degradation (Full Paper)*

Jiří Barnat, Ivana Černá, Jana Tůmová
Faculty of Informatics, Masaryk University,
Botanická 68a, 60200 Brno,
Czech Republic
{xbarnat,cerna,xtumova}@fi.muni.cz

June 10, 2009

Abstract

In this paper we describe a rather specialized quality of a system – the degradation. We demonstrate systems that naturally incorporate degradation phenomenon and we show how these systems can be verified by adapting the standard automata-based approach to LTL model checking. We introduce Büchi Automata with Degradation Constraints (BADCs) to specify the desired properties of systems with degradation and we describe how these can be used for verification. A major obstacle in the verification process is that the synchronous product of the system and the Büchi automaton may be infinite, which we deal with by introducing a normal form of the Büchi automata and normalizing procedure. We also show that the newly introduced formalism can be used to distinguish MDPs indistinguishable by any LTL, PCTL or even PCTL* formula.

1 Introduction

In order to reduce project design costs or to fit the tight time-to-market schedule, numerous software tools including formal verification ones are used in software and hardware development process. Quantitative properties of systems being developed are an

*This work has been supported in part by the Czech Science Foundation grants No. 201/09/P497 and 201/09/1389.

inseparable part of the specifications in many cases. As a result, specialized software tools were designed and are publicly available to help system designers analyze various quantitative aspects of systems. For example, tools such as PRISM [12], LiQuor [6] or ProbDiVinE-MC [4] are used to design and analyze systems with probabilistic actions, tools such as UPPAAL [5] or KRONOS [18] are used to verify timing constraints of real-time systems, MRMC [14] tool analyzes Markov rewards, etc.

In this paper we introduce a rather specialized quality of a system – the degradation. The degradation phenomenon is quite common for objects that are subjects to physics laws. For example, we can measure the degradation of electric charge in some electronic devices, degradation of power or quality of a transmitted signal in broadcasting network, etc. However, the phenomenon is not bound to physical objects only and is present in many other kinds of systems including software ones. For example, a database index degrades with every database update, memory consistency degrades every time an allocation or deallocation of a memory block occurs (memory fragmentation), etc.

To model systems with degradation we use the following approach. Let us assume that an attribute of the model is subject to the degradation. The idea of the degradation is to express the consistency level (or quality) of the attribute using a real number. If the attribute is in perfect shape the associated number equals to one, if the consistency is degraded to 75%, the number equals to 0.75, etc. Since we do not admit negative consistency or consistency better than 100%, the number associated with the attribute is always a number between zero and one.

The level of degradation is manipulated by performing system actions. Every action of the system may either further degrade the attribute, or it may leave it as it is. Henceforward, we assume that the amount of degradation caused by an action of a system is associated with the action and is given as a real number again between zero and one. So, if the current level of degradation is l and the degradation associated with an action is d, the new level of degradation will be $d \cdot l$ after the action is executed.

To our best knowledge, there are no appropriate formalisms developed to properly deal with the degradation aspects of a system. So far, the possibilities to handle the degradation might have been twofold. The first approach would involve using a standard model checker, e.g. SPIN [13]. We can introduce a floating point variable to keep the amount of degradation and describe how the degradation evolves by explicit manipulation with the variable. The second approach could be to use a formalism such as

Markov Decision Processes (MDPs) to express the degradation phenomenon by means of probability. Unfortunately, neither of the approaches is suitable for modeling the real degradation phenomenon in more complex systems. In particular, both approaches lacks the general possibility to verify linear properties of runs of the system under consideration. For example, the property that system designers might be interested in is a repeated *response-with-limited-degradation*, such as: whenever A happens, B happens before the degradation of A drops below certain level. This property cannot be verified using the first approach as a fresh degradation variable needs to be introduced every time A happens. This would require a finite but unbounded number of degradation variables to be introduced in the system description, which is rather problematic regarding the restrictions of the standard model checker input languages. The other approach is unsuitable as well. MDPs require that a state of the system evolves into its immediate descendants in such a way that the sum of degradations distributed among the descendants equals to one for a given action. This is quite restrictive and also unrealistic for many systems. For the same reasons, a system-wide fixed degradation constant, as suggested in [8], is inappropriate.

In this paper we demonstrate systems that naturally incorporate degradation phenomenon. We introduce quantitative linear properties that relate to systems with degradation and define Büchi Automata with Degradation Constraints as the standard formalism to express the desired degradation properties. We adapt the standard automata-based approach to LTL model checking to perform model checking procedure for quantitative linear properties over systems with degradation. The problem with the adaption is that the product automaton to be analyzed may be infinite. To avoid this, we suggest to transform the property Büchi automaton into the so called normal form which than guarantees finiteness of the product automaton. A separate section of the paper relates systems with degradation to MDPs. We demonstrate that expressive power of our specification formalism differs from that of PLTL, PCTL, or PCTL*.

2 Systems with Degradation

Example I: Signal Coverage Problem

Let us suppose, we want to get some signal from a start point S to an end point E. Unfortunately, the points are too far from each other, so the signal cannot reach the destination without unrepairable signal degradation. A possible solution to the problem is to build relays in between S and E that restore the quality of the signal while the signal is still fully re-constructible. Furthermore, let us assume we have a map of possible places where a relay may be built including pairwise signal degradation values as illustrated in Figure 1. For the sake of simplicity, let us assume the signal goes through these places. Using a system with degradation, we can easily check, whether the signal reaches the target point in proper shape if the relays are built at the A-points. Another example of a degradation property might be to check whether some of the A-points are redundant.

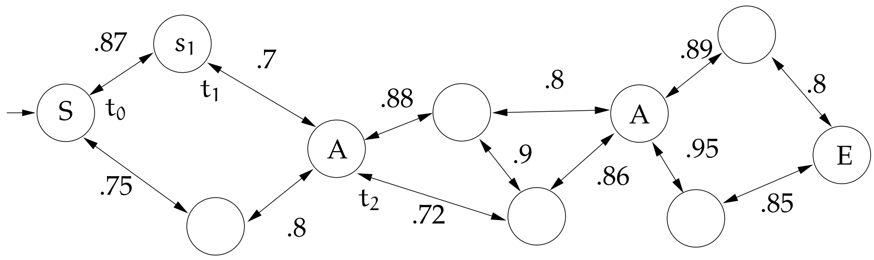


Figure 1: Signal coverage map.

Example II: Magnetic Disk

A common problem that must be dealt with in a firmware of a storage device is a periodical refreshment of data being kept. There are numerous reasons for it, but for the sake of simplicity, let us just suppose that the data integrity are degraded by certain amount, let us say 5%, with every read operation. On the other hand every write or refresh operation restores the integrity of data to 99%. To be on a safe side, the producers of the storage device would like to guarantee that any piece of data is refreshed before its integrity drops below a certain level, let us say 85%. However, the device cannot simply refresh data after every read operation as this would lead to an unacceptable level of power consumption. Therefore, the data are refreshed periodically on a time basis. Note that the read operations may take various amount of time depending on the position of a reading head and the location from where the data are read, which we model using a non-deterministic choice. To answer the question whether the device meets the producers' requirements, we model the device and the controller as depicted in Figure 2. We can verify that no read action is performed if the data degradation is below 85% and refresh actions are performed only if the data degradation is below 90%.

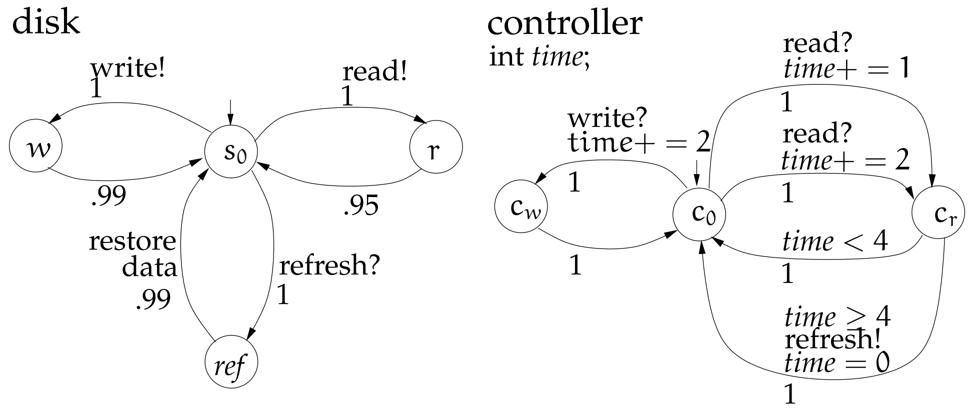


Figure 2: Magnetic disk example.

Transition Systems with Degradation

Informally, systems with degradation are systems that involve an attribute whose quality degrades (e.g. the data integrity in the magnetic disk example). We formalize such systems as *Transition Systems with Degradation* (TSDs). Unlike the standard transition systems, every transition is associated with a degradation constant in a TSD. A degradation constant is a rational number from interval $(0, 1]$. The constants may differ for individual transitions in the system. Note that the formal definition of a TSD contains no specification of the attribute that degrades, it only captures how much it degrades along each transition.

A *transition system with degradation* is a tuple $\mathcal{M} = (S, \mathrm{Act}, \rightarrow, S_{init}, AP, \mathcal{L})$, where

- S is a finite, nonempty set of states,
- Act is a finite, nonempty set of actions,
- $\rightarrow \subseteq S \times \mathrm{Act} \times (0, 1] \times S$ is a transition relation,
- $S_{init} \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions,
- $\mathcal{L} : S \rightarrow 2^{AP}$ is a labeling function; $\mathcal{L}(s)$ denotes the set of atomic propositions that are true in state s.

Instead of $(s_1, a, d, s_2) \in \rightarrow$ we write $s_1 \xrightarrow{a,d} s_2$. A transition $s_1 \xrightarrow{a,d} s_2$ represents that the model can move from state s_1 to the state s_2 by a (nondeterministic) choice of action a. The degradation constant d associated with the transition gives the fraction to which the quality is degraded if the transition is executed. So if the level of degradation at state s_1 is let us say l and the action is executed, the level of degradation at state s_2 will be $l \cdot d$.

A *path* in a TSD $\mathcal{M} = (S, Act, T, S_{init}, AP, \mathcal{L})$ is an infinite sequence $\pi = s_0 t_0 s_1 t_1 \dots$ where $s_i \in S$ and $t_i = (s_i, a_i, d_i, s_{i+1}) \in T$ for all $i \geq 0$. A *trajectory* corresponding to the path $\pi = s_0 t_0 s_1 t_1 \dots$ is given by the projection of π to the state labels, $trajectory(\pi) = \mathcal{L}(s_0)\mathcal{L}(s_1) \dots$. A *trace* corresponding to the path $\pi = s_0 t_0 s_1 t_1 \dots$ is given by the projection of π to the state labels and degradation rates, $trace(\pi) = (\mathcal{L}(s_0), d_0)\ (\mathcal{L}(s_1), d_1)\ \dots$. Let $Traces(\mathcal{M})$ denote the set of all traces of paths in $\mathcal{M}$.

For instance, consider example in Figure 1 and its path $S t_0 s_1 t_1 A t_2 \dots$ with the trace $(S, 0.87), (s_1, 0.7), (A, 0.72), \dots$. The signal degradation between S and A is $0.87 \cdot 0.7 = 0.609$. This means the quality of the signal in A will be 60.9% of the quality in S.

3 Quantitative Linear Properties and Büchi Automata with Degradation Constraints

One way to express a desired behavior of a system is to give restrictions on individual runs of the system, i.e. paths in its model. Properties specified by path restrictions are called linear and are defined on trajectories, i.e. sequences of atomic propositions holding true along a path. However, for systems with degradation, we might be interested not only in sequences of atomic propositions, but also in quantitative aspects, the amount of quality degradation in particular. Formally, we want to analyze traces rather than trajectories.

Consider a TSD $\mathcal{M} = (S, Act, \rightarrow, S_{init}, AP, \mathcal{L})$ and a path $\pi = s_0 t_0 s_1 t_1 s_2 t_2 \dots$ in $\mathcal{M}$. The *amount of degradation along* π *between states* s_i *and* s_j, $i \leq j$, is defined as

$$D_i^j = \prod_{k=i}^{j-1} d_k.$$

In case $i = j$ the amount of degradation is equal to 1.

Quantitative linear properties are linear properties involving constraints on trajectories. These are expressed by specifying boundaries on the amount of degradation along a path between two states. Let us recall the signal coverage example. The question whether the signal reaches the target point in a proper shape is an example of quantitative linear property. In other words, we ask whether there exists a path from the sender to the receiver along which the amount of degradation of the signal does not drop below a given bound provided the signal is fully reconstructed in every relay (A-points).

Another interesting quantitative linear question might be whether there are redundant relays on the way. A relay is redundant if the signal can reach properly its destination without being refreshed at the relay.

Regarding the magnetic disk example the question whether a piece of data is read when its degradation is below 85% or a piece of data is refreshed when its integrity is not below 90% is an example of quantitative linear property as well.

Büchi Automata with Degradation Constraints

To express the quantitative linear properties of systems with degradation we introduce a modification of Büchi automata, the so called *Büchi Automata with Degradation Constraints* (BADC). The standard automata are enriched with a set of bounded variables allowing us to express the amount of degradation.

Let D be a finite set of *degradation variables* ranging over the rational numbers in between $(0, 1]$. A *degradation constraint over* D is of form $\varphi ::= x \bowtie d \mid \varphi \land \varphi$, where $\bowtie \in \{<, \leq, >, \geq\}$, $x \in D$, and d is a rational number in $(0, 1]$. Note that degradation constraints exclude disjunction as it can be expressed using two different transitions of a BADC. $DC(D)$ denotes the set of degradation constraints over D. A *degradation valuation* is a function $\nu : D \to (0, 1]$. The set of all possible degradation valuations is $Eval(D)$.

A *Büchi Automaton with Degradation Constraints (BADC)* is a tuple $\mathcal{A} = (L, \Sigma, D, T, l_{init}, F)$, where

- L is a finite nonempty set of states (locations),
- Σ is a finite alphabet,
- D is a finite set of degradation variables,
- $T \subseteq L \times \Sigma \times DC(D) \times 2^D \times L$ is a set of transitions,
- $l_{init} \in L$ is an initial location,
- $F \subseteq L$ is a finite set of locations (Büchi accepting condition).

A 5-tuple $t = (l, \alpha, \varphi, R, l') \in T$ represents the transition from location l to l' labeled with α that is enabled if constraint φ is satisfied. R is a set of degradation variables which are reset to 1 when executing the transition. For the transition $t = (l, \alpha, \varphi, R, l')$ we denote $label(t) = \alpha$, $constraint(t) = \varphi$ and $reset(t) = R$.

A *path* in a BADC $\mathcal{A} = (L, \Sigma, D, T, l_{init}, F)$ originating at location l_0 (or simply from l_0) is an infinite sequence of locations and transitions $\pi = l_0 t_0 l_1 t_1 \ldots$, where $l_i \in L$ and $t_i = (l_i, \alpha, \varphi, R, l_{i+1}) \in T$ for all $i \geq 0$.

A finite *path* from l_0 to l_n is a finite prefix $\pi_{l_0}^{l_n} = l_0 t_0 l_1 \ldots l_{n-1} t_{n-1} l_n$ of a path from l_0. A finite path $\pi_{l_0}^{l_n}$ is *simple* if $\forall 0 \leq i, j \leq n-1, i \neq j$ implies $t_i \neq t_j$. A simple path $\pi_{l_0}^{l_n}$ forms an *elementary cycle* if $l_0 = l_n$ and $\forall 0 \leq i, j \leq n-1, i \neq j$ implies $l_i \neq l_j$.

The semantics of a BADC $\mathcal{A} = (L, \Sigma, D, T, l_{init}, F)$ is given by an infinite labeled transition system $\mathcal{M}_{\mathcal{A}} = (S, \Sigma', \rightarrow, S_{init})$, where

- $S = L \times Eval(D)$
- $\Sigma' = \Sigma \times (0, 1]$
- $\rightarrow \subseteq S \times \Sigma' \times S$

  $(l_1, \nu_1) \xrightarrow{\alpha, d} (l_2, \nu_2)$ whenever there is a transition $(l_1, \alpha, \varphi, R, l_2) \in T$ such that

  - $\nu_1 \models \varphi$
  - $\nu_2(x) = \begin{cases} d, & \text{if } x \in R \\ \nu_1(x) \cdot d & \text{otherwise} \end{cases}$
- $S_{init} = \{(l_{init}, \nu_{init}) \mid \nu_{init}(x) = 1 \text{ for all } x \in D\}$

A run for a word $\sigma = (\alpha_0, d_0)(\alpha_1, d_1) \ldots \in (\Sigma \times (0, 1])^\omega$ is an infinite sequence $\rho = (l_0, \nu_0)(l_1, \nu_1) \ldots$ such that $(l_0, \nu_0) \in S_{init}$ and $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ for all $i \geq 0$. A run $\rho = (l_0, \nu_0)(l_1, \nu_1) \ldots$ is accepting if $l_i \in F$ for infinitely many indices i. $L_\omega(\mathcal{A}) = \{\sigma \in (\Sigma \times (0, 1])^\omega \mid$ there exists an accepting run for σ in $\mathcal{A}\}$.

Figures 3.a and 3.b depict the "redundant A-point" quantitative linear property for the signal coverage example and the property of the magnetic disc example, respectively.

4 Model Checking Algorithm

Model checking is a technique that for a given finite state model and a temporal property decides whether the model satisfies the property. In our case we are given a TSD model of a system with degradation and a BADC automaton specifying prohibited quantitative linear behaviors. In this section we develop an algorithm deciding whether a given TSD model exhibits a forbidden behavior. Our model checking algorithm follows the automata-based approach to LTL model checking [17]. First, we define a product automaton and prove that this automaton accepts exactly the intersection of the

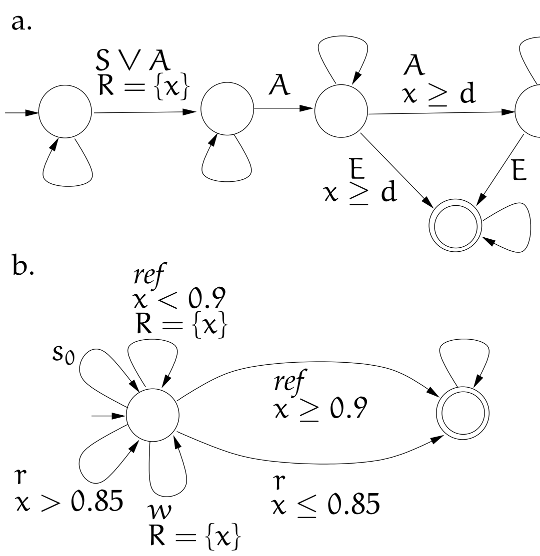


Figure 3: Quantitative properties of Sender/Receiver example (a) and Magnetic disc example (b).

BADC language and the language of TSD traces. Next, we demonstrate that checking non-emptiness of the product automaton is equivalent to finding an accepting cycle in the product automaton graph and can be tested effectively by a number of known techniques like the Nested Depth First Search [7] or OWCTY [10].

Product Automaton

Product automaton of a TSD $\mathcal{M} = (S, Act, \rightarrow, S_{init}, AP, \mathcal{L})$ and a BADC $\mathcal{A} = (L, 2^{AP}, D, T, l_{init}, F)$ is an automaton $\mathcal{M} \otimes \mathcal{A} = (Q, Act, \delta, Q_{init}, Q_F)$, where

- $Q = S \times L \times Eval(D)$
- $\delta : Q \times Act \rightarrow 2^Q$
  $(s', l', \nu') \in \delta((s, l, \nu), a)$ whenever
  - $\exists\, m = (s, a, d, s') \in \rightarrow$
  - $\exists\, t = (l, \mathcal{L}(s), \varphi, R, l') \in T$, such that $\nu \models \varphi$ and $\forall\, x \in D$:

$$\nu'(x) = \begin{cases} d & \text{if } x \in R \\ \nu(x) \cdot d & \text{otherwise} \end{cases}$$

- $Q_{init} = \{(s_{init}, l_{init}, \nu_{init}) \mid s_{init} \in S_{init}, \text{ and } \nu_{init}(x) = 1 \text{ for all } x \in D\}$
- $Q_F = \{(s, l, \nu) \mid l \in F\}$

The product automaton $\mathcal{M} \otimes \mathcal{A}$ can be viewed as an oriented graph $G_{\mathcal{M}\otimes\mathcal{A}} = (Q, E)$. Vertices of $G_{\mathcal{M}\otimes\mathcal{A}}$ are the states of the product automaton and there is an edge from the vertex (s, l, ν) to the vertex (s', l', ν') if $\exists a \in Act : (s', l', \nu') \in \delta((s, l, \nu), a)$. Accepting cycle in the product automaton graph $G_{\mathcal{M}\otimes\mathcal{A}}$ is a cycle containing an accepting state. Henceforward, we consider only the subgraph of $G_{\mathcal{M}\otimes\mathcal{A}}$ reachable from the set of initial vertices Q_{init}, i.e. whenever we mention the product automaton graph, we implicitly mean its reachable subgraph.

We say that a product automaton $\mathcal{M} \otimes \mathcal{A}$ is finite if its graph is finite.

Lemma 4.1. *If a product automaton $\mathcal{M} \otimes \mathcal{A}$ is finite then there is an accepting run of $\mathcal{M} \otimes \mathcal{A}$ if and only if the graph $G_{\mathcal{M}\otimes\mathcal{A}}$ contains an accepting cycle. [2]*

The main obstacle in the verification process is that the product automaton graph may be infinite. An example of such a situation is depicted in Figure 4. Here, the infinity is caused by decreasing value of the variable x always meeting the constraint $x \leq 0.5$.

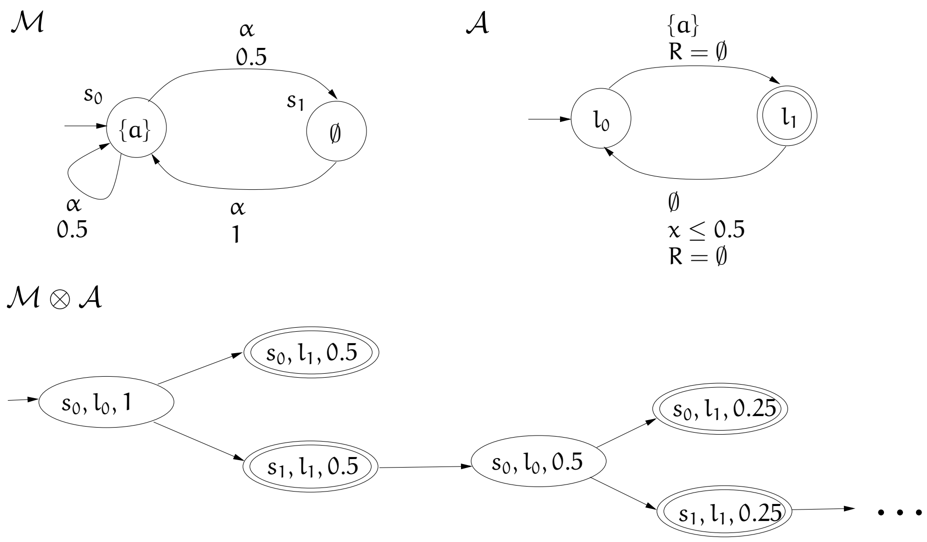


Figure 4: Infinite product.

The key observation allowing for model checking of BADC properties of systems with degradation is that for a special type of BADC automata, the so called *normalized* BADC, it is guaranteed that the product graph is finite. In what follows we give the definition of a normalized BADC and prove that the product automaton of a TSD and a normalized BADC is finite. In the next section we provide an algorithm which transforms any BADC to an equivalent normalized BADC.

Let us consider a BADC $\mathcal{A} = (L, \Sigma, D, T, l_{init}, F)$. A degradation variable $x \in D$ is *in normal form* (or normalized, for short) in $\mathcal{A}$ if for each elementary cycle

$$\pi_{l_0}^{l_0} = l_0 t_0 l_1 t_1 \ldots l_{n-1} t_{n-1} l_0$$

from l_0 to l_0 in $\mathcal{A}$ there is a transition $t_i, 0 \leq i \leq n-1$, such that at least one of the following two conditions holds:

- *constraint*$(t_i) = x \bowtie d$ or $x \bowtie d \wedge \psi$, where $d \in (0, 1]$, $\bowtie \in \{\geq, >\}$, and $\psi \in DC(D)$
- $x \in \mathtt{reset}(t_i)$

$\mathcal{A}$ is *in normal form* (or normalized, for short) if each degradation variable $x \in D$ is in normal form in $\mathcal{A}$.

Lemma 4.2. *The product automaton of a TSD $\mathcal{M} = (S, \mathtt{Act}, \rightarrow, S_{init}, AP, \mathcal{L})$ and a normalized BADC $\mathcal{A} = (L, 2^{AP}, D, T, l_{init}, F)$ is finite.*

Proof. To prove the finiteness of the graph $G_{\mathcal{M}\otimes\mathcal{A}}$ we have to demonstrate the finiteness of its set of states $Q \subseteq S \times L \times Eval(D)$. As S and L are both finite (from the definition of TSD and BADC) it is enough to prove that every constraint variable $x \in D$ attains only finitely many different values in $G_{\mathcal{M}\otimes\mathcal{A}}$.

Let $\rho = (s_0, l_0, \nu_0), (s_1, l_1, \nu_1) \ldots (s_k, l_k, \nu_k)$ be a finite path such that the degradation variable x is reset only in states (s_0, l_0, ν_0) and (s_k, l_k, ν_k). Formally, every edge $(s_i, l_i, \nu_i) \rightarrow (s_{i+1}, l_{i+1}, \nu_{i+1})$ of ρ can be projected to the corresponding transition $m_i = (s_i, a_i, d_i, s_{i+1})$ of $\mathcal{M}$ and the transition $t_i = (l_i, \mathcal{L}(s_i), \varphi_i, R_i, l_{i+1})$ of $\mathcal{A}$. The variable is reset in a state (s_i, l_i, ν_i) iff $x \in R_i$. The initial value of x on ρ is d_0 and along the path is changed to $\prod_{i=0}^{1} d_i$, $\prod_{i=0}^{2} d_i$, $\ldots$, $\prod_{i=0}^{k-1} d_i$, d_k. This sequence of x-values is non-increasing (with the possible exception of the last value d_k). We are to prove that there is a bound B (depending only on $\mathcal{M}$ and $\mathcal{A}$) such that the value of x is decreased on ρ at most B times. The existence of the bound B, together with the fact that there are only finitely many different degradation constants d in transitions of $\mathcal{M}$, assure that x attains only a finite number of different values along a path in $G_{\mathcal{M}\otimes\mathcal{A}}$.

We define constants $C_{\mathcal{M}}$, $C_{\mathcal{A}}$, and $L_{\mathcal{A}}$ distinguishing extremal values in $\mathcal{M}$ and $\mathcal{A}$. For the BADC $\mathcal{A}$ we define $C_{\mathcal{A}}$ as the minimal value such that there is a transition t with $\mathtt{constraint}(t) = x \bowtie C_{\mathcal{A}}$ or $x \bowtie C_{\mathcal{A}} \wedge \psi$, where $\bowtie \in \{\geq, >\}$, $d \in (0, 1]$ and $\psi \in DC(D)$. For the TSD $\mathcal{M}$ we define $C_{\mathcal{M}}$ as the minimal number such that the product of any $C_{\mathcal{M}}$ degradation constants d from transitions of $\mathcal{M}$ is less than $C_{\mathcal{A}}$. $L_{\mathcal{A}}$ is the length of the longest elementary cycle in $\mathcal{A}$.

Let us suppose the value of x is decreased on ρ more than $C_{\mathcal{M}}+ L_{\mathcal{A}}$ times. After the first $C_{\mathcal{M}}$ decreases the value of x is less than $C_{\mathcal{A}}$. The length of the suffix ρ' of ρ, starting in the state where the value of x decreased below $C_{\mathcal{A}}$ for the first time, is greater than $L_{\mathcal{A}}$. The variable x is normalized in $\mathcal{A}$ and thus there is a transition $(s_i, l_i, \nu_i) \to (s_{i+1}, l_{i+1}, \nu_{i+1})$ of ρ' such that *constraint*$(t_i) = x \bowtie d$ or $x \bowtie d \wedge \psi$, where $\bowtie \in \{\geq, > \}$, $d \in (0, 1]$ and $\psi \in DC(D)$. However, this constraint cannot be satisfied as the value $\nu_i(x) < C_{\mathcal{A}} \leq d$. This contradicts the assumption about ρ. □

Lemma 4.3. *$L_\omega(\mathcal{A}) \cap Traces(\mathcal{M}) \neq \emptyset \iff G_{\mathcal{M}\otimes\mathcal{A}}$ contains an accepting cycle.*

Synchronizing the path $\pi = s_0 a_0 s_1 a_1 \ldots$ and the run $\rho' = (l_0, \nu_0)(l_1, \nu_1)\ldots$ we obtain a run $\rho = (s_0, l_0, \nu_0)(s_1, l_1, \nu_1)(s_2, l_2, \nu_2)\ldots$ in the product $\mathcal{M} \otimes \mathcal{A}$ with infinitely many indices i, such that $(s_i, l_i, \nu_i) \in Q_F$, i.e. $G_{\mathcal{M}\otimes\mathcal{A}}$ contains an accepting cycle. □

The number of states of the product is $\mathcal{O}(|S| \cdot |L| \cdot \Pi_{d\in D_N} \log_{step} min(d))$, where $|S|$ is the number of states of a TSD, $|L|$ is the number of locations in an BADC before normalization, D_N is the set of degradation variables after normalization, *step* is the maximal degradation constant different from 1 occurring in the TSD, and *min*(d) is the minimal threshold connected with degradation variable d occurring in the BADC.

An optional way to construct the product automaton without normalization is to modify the procedure of construction of the product automaton as follows. As soon as the value of a degradation variable drops below the minimal threshold occurring in the BADC, the value is tagged with a special flag denoting *below minimal threshold* and it is not manipulated in succeeding states anymore. This approach leads to a finite product automaton with $\mathcal{O}(|S| \cdot |L| \cdot (\log_{step} min)^{|D|})$ states, where $|S|$, $|L|$, and *step* are as in the previous case, $|D|$ is the number of degradation variables, and *min* is the overall minimal threshold occurring in the BADC.

The reason, why we have introduced normalization is that it helps to rapidly reduce the size of the product automaton in many cases. It is basically a heuristic to minimize the number of different values each degradation variable may get. Figure 5 illustrates an original BADC, its normalized form and a transition system. The product of the original BADC and the TSD has 239 states, whereas in the case of the normalized BADC the product has only 46 states. The normalization procedure adds resets of variables whenever it is possible. See, e.g., the self-loop on state l_1.

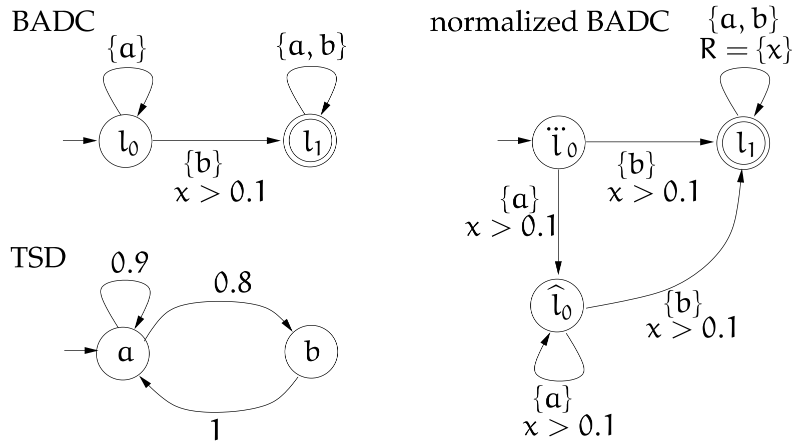


Figure 5: BADC, normalized BADC and TSD.

5 Normalization of BADC

In this section, we describe how to transform a BADC into an equivalent BADC in the normal form.

Let us say that a degradation variable x is *bounded* in degradation constraint $\varphi \in DC(D)$ if $\varphi = x \bowtie d$ or $\varphi = x \bowtie d \wedge \psi$, $\psi \in DC(D)$. More precisely, x is n-*bounded* in constraint $\varphi \in DC(D)$ if $\varphi = x \bowtie_1 d_1 \wedge \ldots \wedge x \bowtie_n d_n$ or $\varphi = x \bowtie_1 d_1 \wedge \ldots \wedge x \bowtie_n d_n \wedge \psi$, where x is not bounded in ψ. x is (n-)bounded in transition t if x is (n-)bounded in *constraint*(t).

The transformation algorithm (see Algorithm 1) works in several stages.

In the initial stage (see Procedure ONECONSTRAINTONVARIABLE), the given BADC is transformed into a BADC in which every degradation variable x is bounded by at most one inequality $x \bowtie d_x$. This is accomplished by introducing new degradation variables into the BADC. In the next stage, we iteratively pick a degradation variable $x \in DC$ and transform the BADC so that x becomes normalized while preserving the normal form of the already processed degradation variables.

Normalization of the degradation variable x involves two procedures. The first procedure (see Procedure RESETWHEREPOSSIBLE) identifies those transitions where the variable x can be safely reset. To this end it computes the set Π of all simple paths π satisfying three conditions: π starts in the initial location or in a location immediately after reset of x, no reset of x occurs along π, and π ends in a location from which there is a transition with a bound on x. Now we can split all the transitions into two disjoint sets: those which occur on a path from Π (the set T_P) and those which do not (the set T_N).

Algorithm 1 Normalization of BADC

Input: BADC $\mathcal{A}^0 = (L^0, \Sigma, D^0, T^0, l^0_{init}, F^0)$

Output: BADC $\mathcal{A} = (L, \Sigma, D, T, l_{init}, F)$ in normal form

$\mathcal{A} := \mathcal{A}^0$

ONECONSTRAINTONVARIABLE

Done $:= \emptyset$

while *Done* $\neq D$ **do**

  pick $x \in D \setminus$ *Done*

  $(L_R, L_P, L_x) :=$ RESETWHEREPOSSIBLE(x)

  SPLITINTOLAYERS(x)

  Done $:=$ *Done* $\cup \{x\}$

  Assert: Each $x \in$ *Done* is normalized in $\mathcal{A}$

end while

Assert: $\mathcal{A}$ is in normal form

Assert: $L_\omega(\mathcal{A}^0) = L_\omega(\mathcal{A})$

Algorithm 2 Procedure ONECONSTRAINTONVARIABLE

for all $x \in D^0$ **do**

  $\mathcal{A}' := \mathcal{A}$

  $T_x := \{t_i \in T \mid x \text{ is bounded in } t_i\}$

  for all $t_i \in T_x$ **do**

    $m_i := n$ such that x is n-bounded in t_i

    $D := (D \setminus \{x\}) \cup \{x_{i1} \ldots, x_{im_i} \mid x_{i1}, \ldots, x_{im_i}$ are new, unique degradation variables$\}$

    replace *constraint*$(t_i) = x \bowtie_1 d_1 \wedge \ldots \wedge x \bowtie_{m_i} d_{m_i} \wedge \varphi$ **with** $x_{i1} \bowtie_1 d_1 \wedge \ldots \wedge x_{im_i} \bowtie_{m_i} d_{m_i} \wedge \varphi$

  end for

  for all $t \in T$ **do**

    if $x \in$ *reset*(t) **then**

      reset$(t) :=$ (*reset*$(t) \setminus \{x\}) \cup \{x_{i1}, \ldots, x_{im_i} \mid t_i \in T_x\}$

    end if

  end for

  Assert: $L_\omega(\mathcal{A}') = L_\omega(\mathcal{A})$

end for

Assert: $\forall x \in D : \exists! t \in T$ with bounded x

Assert: $\forall x \in D : \exists! t \in T$ with 1-bounded x

Assert: $L_\omega(\mathcal{A}^0) = L_\omega(\mathcal{A})$

We can reset the variable x on the transitions from T_N without changing the language of the BADC. Simultaneously, three other sets of locations, namely L_R, L_P, and L_x, are computed. L_R is the set of locations in which a path $\pi \in \Pi$ originates. L_P are locations

Algorithm 3 **Procedure** RESETWHEREPOSSIBLE(x)

$T_x := \{t_x \in T \mid x \text{ is bounded in } t_x\}$
$L_x := \{l_x \in L \mid \exists \text{ transition } t_x \in T_x \text{ from location } l_x\}$
$\Pi := \{\pi \mid \pi \text{ is a simple path } l_0 t_0 l_1 \dots l_n t_n l_x \text{ in } A,\ l_0 = l_{init} \text{ or } \exists \text{ transition } t \in T \text{ to } l_0 \text{ with } x \in \mathtt{reset}(t),$
$\quad \forall\, 0 \leq i \leq n : x \notin \mathtt{reset}(t_i), \text{ and } l_x \in L_x\}$
$L_R := \{l_0 \mid \exists\, \pi \in \Pi \text{ originating at } l_0\}$
$L_P := \{l_i, l_x \mid \exists\, \pi = l_0 t_0 \dots l_n t_n l_x \in \Pi, 0 \leq i \leq n\}$
$T_P := \{t_i \mid \exists\, \pi = l_0 t_0 \dots l_n t_n l_x \in \Pi, 0 \leq i \leq n\}$
$T_N := T \setminus T_P$
for all $t \in T_N$ **do**
    $\mathit{reset}(t) := \mathit{reset}(t) \cup \{x\}$
end for
return (L_R, L_P, L_x)
Assert: $L_\omega(\mathcal{A}^0) = L_\omega(\mathcal{A})$

occurring along a path $\pi \in \Pi$, and finally L_x are locations in which a path $\pi \in \Pi$ ends. Note that $L_R, L_x \subseteq L_P$.

Procedure SPLITINTOLAYERS finishes the normalization of the variable x. It manipulates the rest of the transitions that may cause that x is not normalized, namely those from the set T_P. The modification of the BADC is a bit more involved here and requires a replication of locations. Each replica of the location bears a specific information about the actual value of x. We replace each location $l \in L_P$ with two new locations $\check{l}$ and $\widehat{l}$. Moreover, if $l \in L_R$, we introduce a new location $\ddot{l}$. The information associated with the replicas is intuitively characterized as follows:

- $\ddot{l}$-locations: Whenever the location $l \in L_R$ is entered via a transition with reset of x from location k in the original BADC, the location $\ddot{l}$ is entered in the transformed one from location k if $k \notin L_P$ or from any replica of k if $k \in L_P$. The value $\nu(x)$ is the same in $\ddot{l}$ and in the corresponding l in the original BADC.

- $\check{l}$-locations: Let $x \bowtie d_x$ be the only degradation constraint which bounds x in $\mathcal{A}$. Let us define a lower bound $\mathtt{lb}(x)$ as $\mathtt{lb}(x) = x \bowtie d_x$ if $\bowtie \in \{<, \leq\}$ and $\mathtt{lb}(x) = \neg(x \bowtie d_x)$ otherwise. Whenever the location $l \in L_P$ is entered from a location k in which $\nu(x) \models \mathtt{lb}(x)$ via a transition without reset of x in the original BADC, the location $\check{l}$ is entered in the transformed one from any replica of k (necessarily, $k \in L_P$). Due to the monotonicity of degradation, starting from the state k the value of x remains less or less-or-equal than d_x until a reset of x. Therefore, we do not need to keep the value $\nu(x)$ in $\check{l}$ the same as in l (it suffices to know that

Algorithm 4 **Procedure** SPLITINTOLAYERS(x)

Let $x \bowtie d$ be the constraint on x

$lb(x) := \bowtie \in \{<, \leq\}\ ?\ x \bowtie d : \neg(x \bowtie d)$

$L := L \cup \{\widehat{l}, \check{l} \mid l \in L_P, \widehat{l}, \check{l}$ are new, unique locations$\} \cup \{\ddot{l} \mid l \in L_R, \ddot{l}$ is a new, unique location$\}$

$l_{init} := \ddot{l}_{init}$ if $l_{init} \in L_R$

$F := (F \setminus L_P) \cup \{\widehat{l}, \check{l} \mid l \in L_P \cap F\} \cup \{\ddot{l} \mid l \in L_R \cap F\}$

for all $t = (l_1, a, \varphi, R, l_2) \notin T_x$ **do**

  case $l_1 \notin L_P$, $x \in R$, and $l_2 \in L_R$: replace t with $(l_1, a, \varphi, R, \ddot{l}_2)$

  case $l_1 \in L_P$, $x \in R$, and $l_2 \in L_R$: replace t with $(\widehat{l}_1, a, \varphi, R, \ddot{l}_2)$, and $(\check{l}, a, \varphi, R, \ddot{l}_R)$

  case $l_1 \in L_R$, $x \in R$, and $l_2 \in L_R$: add transition $(\ddot{l}_1, a, \varphi, R, \ddot{l}_2)$

  case $l_1 \in L_P$, $x \notin R$, and $l_2 \in L_P$: replace t with $(\widehat{l}_1, a, \varphi \wedge \neg lb(x), R, \widehat{l}_2)$, $(\widehat{l}_1, a, \varphi \wedge lb(x), R \cup \{x\}, \check{l}_2)$, and $(\check{l}_1, a, \varphi, R \cup \{x\}, \check{l}_2)$

  case $l_1 \in L_R$, $x \notin R$, and $l_2 \in L_P$: add transitions $(\ddot{l}_1, a, \varphi \wedge \neg lb(x), R, \widehat{l}_2)$, and $(\ddot{l}_1, a, \varphi \wedge lb(x), R \cup \{x\}, \check{l}_2)$

  case $l_1 \in L_P$, and $l_2 \notin L_P$: replace t with $(\widehat{l}_1, a, \varphi \wedge, R \cup \{x\}, l_2)$, and $(\check{l}_1, a, \varphi, R \cup \{x\}, l_2)$

  case $l_1 \in L_R$, and $l_2 \notin L_P$: add $(\ddot{l}_1, a, \varphi, R \cup \{x\}, l_2)$

end for

for all $t_x = (l_x, a, \varphi, R, l_2) \in T_x$ **do**

  if $\bowtie \in \{<, \leq\}$ **then**

    $\bar{l}_2 := l_2 \notin L_P\ ?\ l_2\ :\ (x \in R\ ?\ \ddot{l}_2\ :\ \check{l}_2)$

    replace t_x with $(\widehat{l}_x, a, \varphi \wedge lb(x), R \cup \{x\}, \bar{l}_2)$, $(\check{l}_x, a, \varphi, R \cup \{x\}, \bar{l}_2)$, and if $l_x \in L_R$ add $(\ddot{l}_x, a, \varphi \wedge lb(x), R \cup \{x\}, \bar{l})$

  else

    $\bar{l}_2 := l_2 \notin L_P\ ?\ l_2\ :\ (x \in R\ ?\ \ddot{l}_2\ :\ \widehat{l}_2)$

    replace t_x with $(\widehat{l}_x, a, \varphi \wedge \neg lb(x), R, \bar{l}_2)$, and if $l_x \in L_R$ add $(\ddot{l}_x, a, \varphi \wedge \neg lb(x), R, \bar{l})$

  end if

end for

Assert: $L_\omega(\mathcal{A}^0) = L_\omega(\mathcal{A})$

$\nu(x)$ remains below d_x). Thus we can add reset of x on each transition entering the $\check{l}$-location.

- $\widehat{l}$-locations: $\widehat{l}$-locations are dual to $\check{l}$-locations. Whenever the location $l \in L_P$ is entered from a location k in which $\nu(x) \not\models lb(x)$ via a transition without reset of x in the original BADC, the location $\widehat{l}$ is entered in the transformed one from $\widehat{k}$ and in case $k \in L_R$ also from $\ddot{k}$. It cannot be entered from $\check{k}$ as we know that $\nu(x) \models lb(x)$ in $\check{k}$. The value $\nu(x)$ is the same in $\widehat{l}$-location and in the corresponding l-location in the original BADC. Note that any transition leading to a $\widehat{l}$-location contains a bound of form $x > d_x$ or $x \geq d_x$.

Transitions entering l are naturally replaced by transitions entering $\ddot{l}$, $\breve{l}$ or $\widehat{l}$ keeping the above characteristics. Normal form is guaranteed by the fact that for every degradation variable S every transition in the resulting BADC either resets the value of x or contain a constraint of the form $x > d_x$ or $x \geq d_x$.

For an illustrative example of the transformation see Figures 6, 7, and 8.

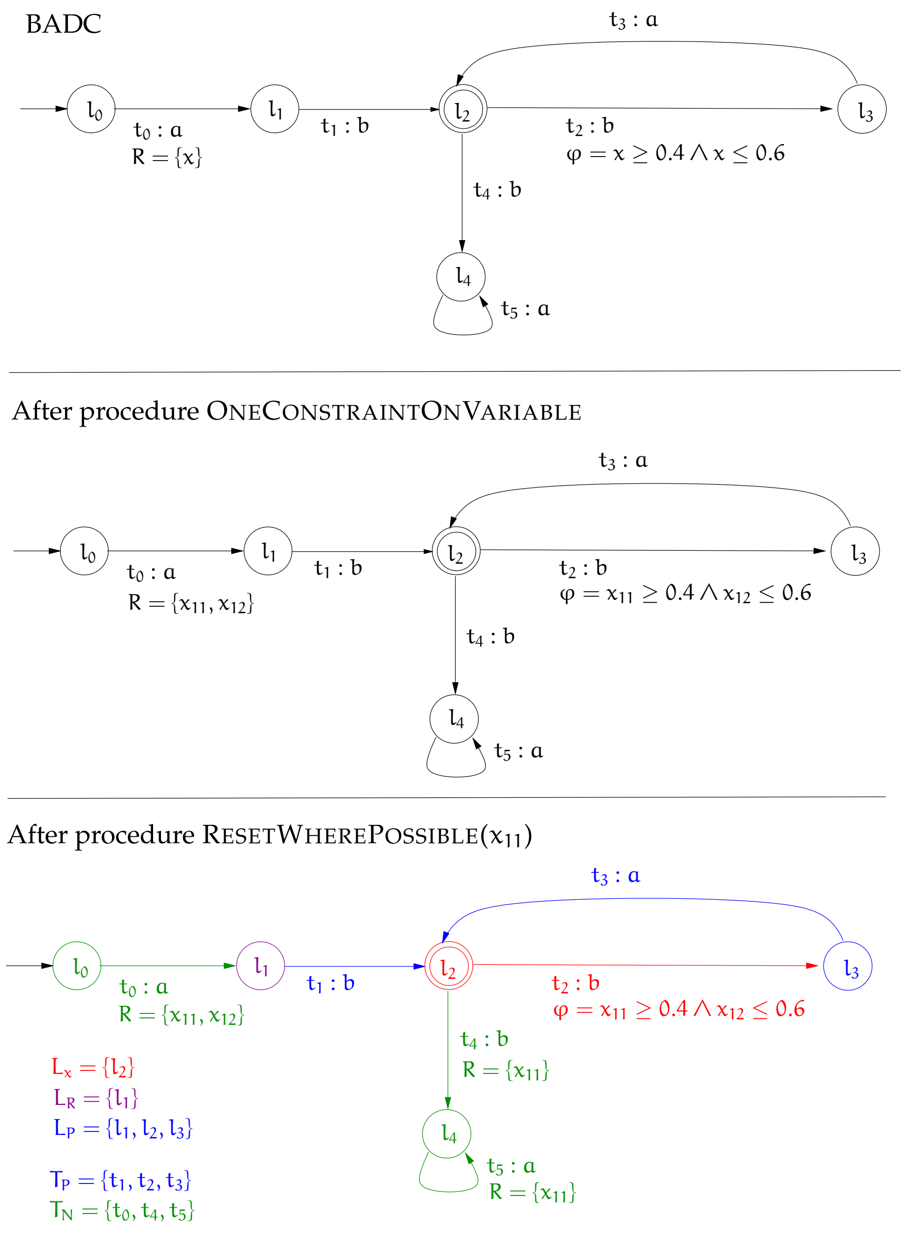
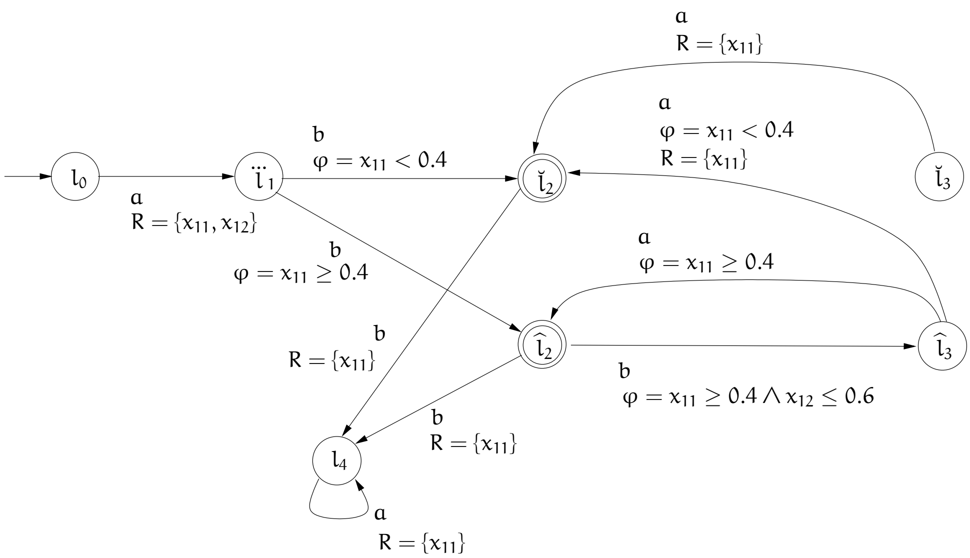


Figure 6: Example of Normalization I.

After procedure SPLITINTOLAYERS(x_{11})



After procedure RESETWHEREPOSSIBLE(x_{12})

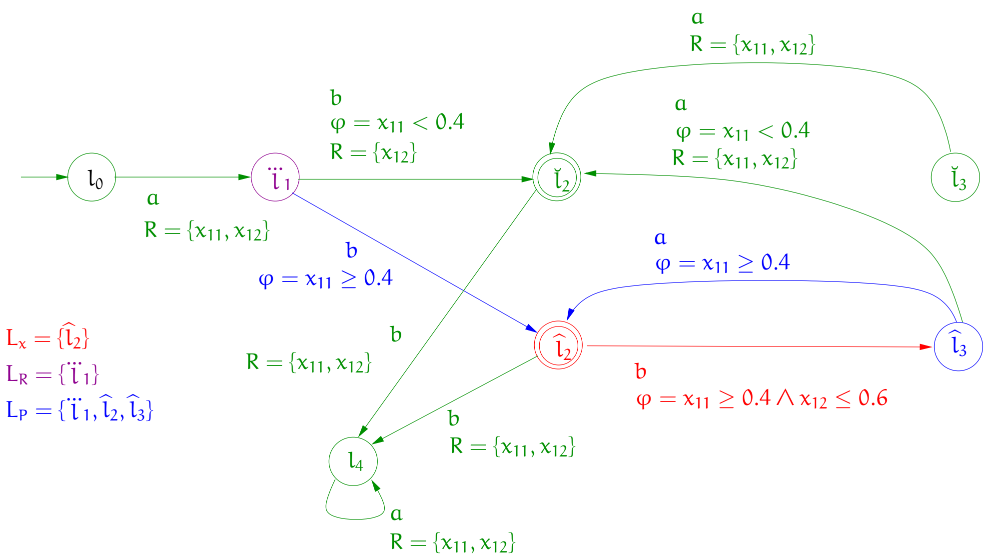
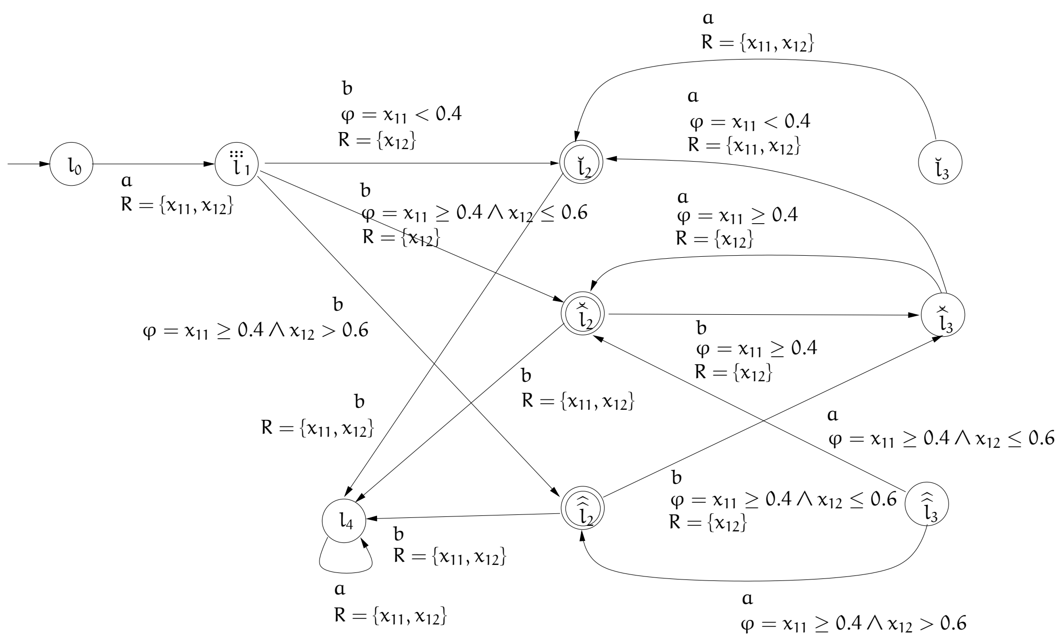


Figure 7: Example of Normalization II.

After procedure SPLITINOTLAYERS(x_{12})



After removing locations unreachable from the initial location and incident transitions

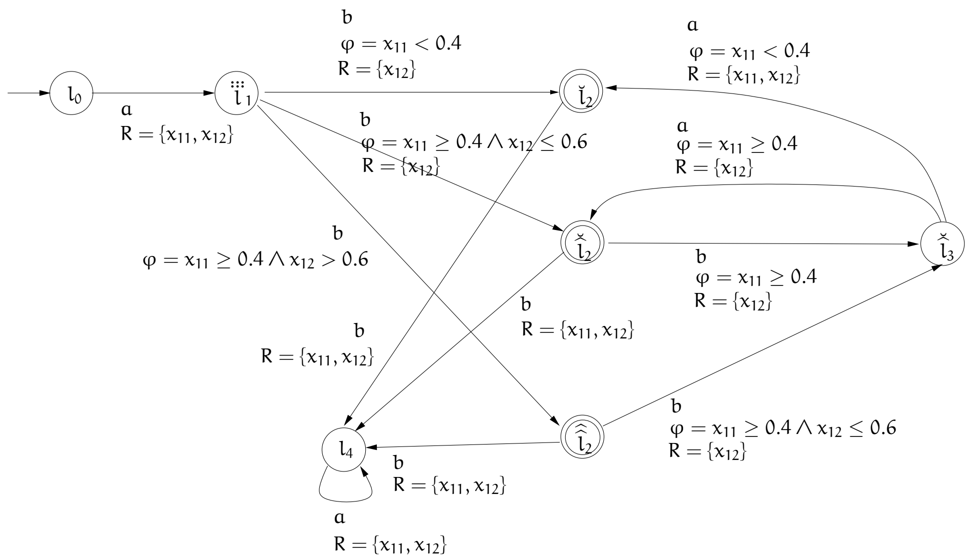


Figure 8: Example of Normalization III.

Correctness of Normalization

In this subsection we prove the correctness of the construction.

Lemma 5.1. *Assertions on lines 16, 17 in Algorithm 2 hold.*

Proof. Follows directly from lines 6 and 7. □

Lemma 5.2. *Assertion on line 13 in Algorithm 2 holds.*

Proof. Consider the automata $\mathcal{A}' = (L, \Sigma, D', T', l_{init}, F)$ and $\mathcal{A} = (L, \Sigma, D, T, l_{init}, F)$ on line 14. Let $x \in D^0$ and $x_{11}, \ldots, x_{1m_1}, \ldots, x_{n1}, \ldots, x_{nm_n}$ be the new counters added to D on line 6 during the iteration of cycle 1-14 for variable x.

We prove that for each run $\rho' = (l_0, \nu'_0)(l_1, \nu'_1)\ldots$ for word $\sigma = (\alpha_0, d_0)(\alpha_1, d_1)\ldots$ in the model $\mathcal{M}'_\mathcal{A}$ there is a run $\rho = (l_0, \nu_0)(l_1, \nu_1)\ldots$ for σ in $\mathcal{M}_\mathcal{A}$ and vice versa.

First, we show that for each $i \geq 0$ such that $\nu_i(x_{11}) = \ldots = \nu_i(x_{1m_1}) = \ldots = \nu_i(x_{n1}) = \ldots \nu_i(x_{nm_n}) = \nu'_i(x)$ it holds

- $(l_i, \nu'_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu'_{i+1})$ in $\mathcal{M}'_\mathcal{A} \Leftrightarrow (l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ in $\mathcal{M}_\mathcal{A}$
- $\nu_{i+1}(x_{11}) = \ldots = \nu_{i+1}(x_{nm_n}) = \nu'_{i+1}(x)$.

Obviously, $\nu_0(x_{11}) = \ldots = \nu_0(x_{nm_n}) = \nu'_0(x) = 1$.

Let us assume that $\nu_i(x_{11}) = \ldots = \nu_i(x_{nm_n}) = \nu'_i(x)$ for each $i \geq 0$ and let $(l_i, \nu'_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu'_{i+1})$ in $\mathcal{M}'_\mathcal{A}$ via transition $t'_i = (l_i, \alpha_i, \varphi', R', l_{i+1}) \in T'$. It holds that $t'_i \in T$ on line 2. Let $t_i = (l_i, \alpha_i, \varphi, R, l_{i+1}) \in T$ be the transition t'_i on line 14, i.e. after φ' is possibly changed on line 7 into φ and R' is possibly changed on line 11 into R. Note that $\nu'_i \models \varphi' \Leftrightarrow \nu_i \models \varphi$, and moreover $x \in R' \Leftrightarrow \{x_{11}, \ldots, x_{nm_n}\} \in R$. Thus $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ in $\mathcal{M}_\mathcal{A}$ via t_i and $\nu_{i+1}(x_{11}) = \ldots = \nu_{i+1}(x_{nm_n}) = \nu'_{i+1}(x)$.

Dually, let us assume that for $i \geq 0$ it holds $\nu_i(x_{11}) = \ldots \nu_i(x_{nm_n}) = \nu'_i(x)$ and let $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ in $\mathcal{M}_\mathcal{A}$ via transition $t_i = (l_i, \alpha_i, \varphi, R, l_{i+1}) \in T$. Let $t'_i = (l_i, \alpha_i, \varphi', R', l_{i+1}) \in T'$ be the transition t_i before φ is possibly changed on line 7 and R is possibly changed on line 11. Because $\nu'_i \models \varphi' \Leftrightarrow \nu_i \models \varphi$, and $x \in R' \Leftrightarrow \{x_{11}, \ldots, x_{nm_n}\} \in R$, $(l_i, \nu'_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu'_{i+1})$ in $\mathcal{M}'_\mathcal{A}$ via t'_i and $\nu_{i+1}(x_{11}) = \ldots = \nu_{i+1}(x_{nm_n}) = \nu'_{i+1}(x)$.

Altogether for each run $\rho' = (l_0, \nu'_0)(l_1, \nu'_1)\ldots$ for word $\sigma = (\alpha_0, d_0)(\alpha_1, d_1)\ldots$ in $\mathcal{M}'_\mathcal{A}$ there exists a run $\rho = (l_0, \nu_0)(l_1, \nu_1)\ldots$ for σ in $\mathcal{M}_\mathcal{A}$ and vice versa. Furthermore, if ρ' is an accepting run then ρ is an accepting run as well. □

Lemma 5.3. *Assertion on line 18 in Algorithm 2 holds.*

Proof. Follows directly from Lemma 5.2. □

Consider a BADC $\mathcal{A} = (L, \Sigma, D, T, l_{init}, F)$. An *extended run* for $\sigma = (\alpha_0, d_0)(\alpha_1, d_1)\ldots$ in $M_{\mathcal{A}}$ is an infinite sequence $\rho_E = (l_0, \nu_0)t_0(l_1, \nu_1)t_1\ldots$, such that $l_0 = l_{init}$, $\nu_0(x) = 1$ for each $x \in D$, and for each $i \geq 0$ it holds that $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ *via transition* t_i, i.e such that $t_i = (l_i, \alpha_i, \varphi_i, R_i, l_{i+1})$, $\nu_i \models \varphi_i$, and $\nu_{i+1}(x) = d_i$ if $x \in R_i$, and $\nu_i.d_i$ otherwise.

Lemma 5.4. *Assertion on line 12 in Algorithm 3 holds.*

Proof. Let us consider BADC $\mathcal{A} = (L, \Sigma, D, T, l_{init}, F)$ before the procedure 3 is applied, denoted $\mathcal{A}' = (L, \Sigma, D, T', l_{init}, F)$ and let us assume that $L_\omega(\mathcal{A}') = L_\omega(\mathcal{A}^0)$.

Let $\rho' = (l_0, \nu'_0)t'_0(l_1, \nu'_1)t'_1\ldots$ be an extended run for word $\sigma = (\alpha_0, d_0)(\alpha_1, d_1)\ldots$ in $M'_{\mathcal{A}}$. We show that there exists an extended run $\rho = (l_0, \nu_0)t_0(l_1, \nu_1)t_1\ldots$ for σ in $M_{\mathcal{A}}$ (and vice versa).

Note that if $l_i = l_x \in L_x$ for some $i \geq 0$ and t'_i is with bounded x, then, according to the definitions of L_R, L_P and T_P, there exists $l_k \in L_R$, $k \leq i$, such that $k = 0$ or $x \in \mathtt{reset}(t'_{k-1})$ and $\forall m \in \{k, \ldots i\} : t'_m \in T_P$.

Let $t'_i = (l_i, \alpha_i, \varphi, R, l_{i+1})$. We put

$$t_i = \begin{cases} t'_i, & \text{if } t_i \in T_P \\ (l_i, \alpha_i, \varphi, R \cup \{x\}, l_{i+1}) & \text{if } t_i \in T_N \end{cases}$$

First of all, $\nu'_0(y) = \nu_0(y)$ for each $y \in D$.

Let $\nu'_i(y) = \nu_i(y)$ for each $y \neq x \in D$ and $(l_i, \nu'_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu'_{i+1})$ via t'_i and each t'_k, $k \leq i$ is without bounded x in $\mathcal{A}'$. Then $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ via t_i in $\mathcal{A}$ and $\nu'_{i+1}(y) = \nu_{i+1}(y)$ for each $y \neq x \in D$. Furthermore, if $x \in \mathtt{reset}(t'_i)$, then $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If in addition $\nu'_i(x) = \nu_i(x)$ and $t'_i \in T_P$ we have $\nu'_{i+1}(x) = \nu_{i+1}(x)$.

Let t'_i be the very first occurrence of transition with bounded x on ρ'. Then necessarily there exists $l_k \in L_R$, $k \leq i$, such that $k = 0$ or $x \in \mathtt{reset}(t'_{k-1})$ and $\forall m \in \{k, \ldots i\}$: $t'_i \in T_P$. According to the previous, $\nu'_k(y) = \nu_k(y)$ and inductively $\nu'_i(y) = \nu_i(y)$ for all $y \in D$, especially for $\nu'_i(x) = \nu_i(x)$. Because $\mathtt{constraint}(t'_i) = \mathtt{constraint}(t_i)$, we have $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ via t_i in $\mathcal{M}_{\mathcal{A}}$. Furthermore, $\nu'_{i+1}(y) = \nu_{i+1}(y)$ for each $y \neq x \in D$. Moreover, if $x \in \mathtt{reset}(t'_i)$, then $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If in addition $t'_i \in T_P$ we have $\nu'_{i+1}(x) = \nu_{i+1}(x)$.

Inductively, let $\nu'_i(y) = \nu_i(y)$ for each $y \neq x \in D$ and $(l_i, \nu'_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu'_{i+1})$ via t'_i. If t'_i is without bounded x in $\mathcal{A}'$, then $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ via t_i in $\mathcal{A}$

and $\nu'_{i+1}(y) = \nu_{i+1}(y)$ for each $y \neq x \in D$. Furthermore, if $x \in \mathtt{reset}(t'_i)$, then $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If in addition $\nu'_i(x) = \nu_i(x)$ and $t'_i \in T_P$ we have $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If t'_i is with bounded x in $\mathcal{A}'$, then necessarily $\nu'_i(x) = \nu_i(x)$ and $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ via t_i in $\mathcal{M}_\mathcal{A}$. Furthermore, $\nu'_{i+1}(y) = \nu_{i+1}(y)$ for each $y \neq x \in D$. Moreover, if $x \in \mathtt{reset}(t'_i)$, then $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If in addition $t'_i \in T_P$ we have $\nu'_{i+1}(x) = \nu_{i+1}(x)$.

All in all, if $\rho' = (l_0, \nu'_0)t'_0(l_1, \nu'_1)t'_1 \ldots$ is an extended run in $\mathcal{M}'_\mathcal{A}$ for σ then there exists a run $\rho = (l_0, \nu_0)t_0(l_1, \nu_1)t_1 \ldots$ in $\mathcal{M}_\mathcal{A}$ for σ.

Obviously, if ρ' is an accepting run in $\mathcal{A}'$, then ρ is an accepting run in $\mathcal{A}$.

Dually, let $\rho = (l_0, \nu_0)t_0(l_1, \nu_1)t_1 \ldots$ be an extended accepting run for word $\sigma = (\alpha_0, d_0)(\alpha_1, d_1)(\alpha_2, d_2) \ldots$ in $\mathcal{M}_\mathcal{A}$. We show that there exists an extended accepting run $\rho' = (l_0, \nu'_0)t'_0(l_1, \nu'_1)t'_1 \ldots$ for σ in $\mathcal{M}'_\mathcal{A}$. Let

$$t'_i = \begin{cases} t_i, & \text{if } t_i = (l_i, \alpha_i, \varphi, R, l_{i+1}) \\ & \quad t_i \in T_P \\ (l_i, \alpha_i, \varphi, R, l_{i+1}), & \text{if } t_i = (l_i, \alpha_i, \varphi, R \cup \{x\}, l_{i+1}) \\ & \quad t_i \in T_N \end{cases}$$

The proof is analogous as for the first part. □

Lemma 5.5. *Assertion on line 24 in Algorithm 4 holds.*

Proof. Let us consider BADC $\mathcal{A} = (\mathrm{Loc}, \Sigma, D, T, l_{init}, F)$ before the procedure 4 is applied, denoted $\mathcal{A}' = (\mathrm{Loc}', \Sigma, D, T', l'_{init}, F')$ and let us assume $L_\omega(\mathcal{A}') = L_\omega(\mathcal{A}^0)$.

Let $\rho' = (l'_0, \nu'_0)t'_0(l'_1, \nu'_1)t'_1 \ldots$ be an extended run for word $\sigma = (\alpha_0, d_0)(\alpha_1, d_1) \ldots$ in $\mathcal{M}'_\mathcal{A}$. We show that there exists an extended run $\rho = (l_0, \nu_0)t_0(l_1, \nu_1)t_1 \ldots$ for σ in $\mathcal{M}_\mathcal{A}$ (and vice versa). Throughout the proof we assume that $\nu_i(y) = \nu'_i(y)$ for all $y \neq x \in D$, which is obvious. Let

$$l_i = \begin{cases} l'_i & \text{if } l'_i \notin L_P \\ \check{l}'_i & \text{if } l'_i \in L_P \text{ and } x \notin \mathtt{reset}(t'_{i-1}), i \neq 0 \\ & \text{and } \nu'_{i-1}(x) \models \mathtt{lb}(x) \\ \hat{l}'_i & \text{if } l'_i \in L_P \text{ and } x \notin \mathtt{reset}(t'_{i-1}), i \neq 0 \\ & \text{and } \nu'_{i-1}(x) \not\models \mathtt{lb}(x). \text{ Then also } \nu'_i(x) = \nu_i(x). \\ \dddot{l}'_i & \text{if } l'_i \in L_P \text{ and } x \in \mathtt{reset}(t'_{i-1}) \text{ or } i = 0. \end{cases}$$

Note that l'_i belongs to exactly one of the categories described above, i.e. we define l_i unambiguously for each possible l'_i.

First of all, $l_0 = \dddot{l}'_0$ if $l'_0 \in L_P$ and $l_0 = l'_0$ otherwise. Let $(l'_i, \nu'_i) \xrightarrow{\alpha_i, d_i} (l'_{i+1}, \nu'_{i+1})$ via $t'_i = (l'_i, \alpha, \varphi', R', l'_{i+1})$. We show that $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ via transition $t_i = (l_i, \alpha, \varphi, R, l_{i+1})$. Particularly, we show case by case that such t_i exists.

1. Let $l'_i \notin L_P$, i.e. $l_i = l'_i$:

   a) $l'_{i+1} \notin L_P$. Then $t_i = t'_i$ and $l_{i+1} = l'_{i+1}$.

   b) $l'_{i+1} \in L_P$. That means necessarily that $x \in reset(t'_i)$ and $t'_i \neq t_x$, according to the definition of L_P. Then $l_{i+1} = \dddot{l}'_{i+1}$ and $t_i = (l'_i, \alpha, \varphi, R, \dddot{l}'_{i+1})$ (line 7).

2. Let $l'_i \in L_P$ and $x \notin reset(t_{i-1'}), i \neq 0$ and $\nu'_{i-1}(x) \models lb(x)$, i.e. $l_i = \check{l}'_i$:

   a) $l'_{i+1} \notin L_P$. Then $l_{i+1} = l'_{i+1}$. If $t_i \neq t_x$ then $t_i = t'_i$ (line 12). If $t_i = t_x$ then $\nu'_i(x) \models lb(x)$, thus $\bowtie \in \{<, \leq\}$. $t_i = (\check{l}'_i, \alpha, \varphi, R \cup \{x\}, l'_{i+1})$ (line 18).

   b) $l'_{i+1} \in L_P, x \notin reset(t'_i)$ and $\nu_i(x)' \models lb(x)$. Then $l_{i+1} = \check{l}'_{i+1}$. If $t'_i \neq t_x$ then $t_i = (\check{l}'_i, \alpha, \varphi, R \cup \{x\}, \check{l}'_{i+1})$ (line 10). If $t'_i = t_x$ then $t_i = (\check{l}'_i, \alpha, \varphi, R \cup \{x\}, \check{l}'_{i+1})$ (line 18).

   c) $l'_{i+1} \in L_P, x \in reset(t'_i)$ and $\nu_i(x)' \not\models lb(x)$. This situation cannot happen, because $\nu_{i-1}(x) \models lb(x)$ and $x \notin reset(t_{x-1})$, thus $\nu_i(x) \models lb(x)$.

   d) $l'_{i+1} \in L_P, x \in reset(t'_i)$. Then $l_{i+1} = \dddot{l}'_{i+1}$. If $t'_i \neq t_x$ then $t_i = (\check{l}'_i, \alpha, \varphi, R, \dddot{l}'_{i+1})$ (line 8). If $t'_i = t_x$ then $\nu_i(x) \models lb(x)$, because $\nu_{i-1}(x) \models lb(x)$ and $x \notin reset(t_{x-1})$. Thus $\bowtie \in \{<, \leq\}$. Altogether $t_i = (\check{l}'_i, \alpha, \varphi, R \cup \{x\}, \dddot{l}'_{i+1})$ (line 18).

3. Let $l'_i \in L_P$ and $x \notin reset(t_{i-1}), i \neq 0$ and $\nu'_{i-1}(x) \not\models lb(x)$ and also $\nu'_i(x) = \nu_i(x)$, i.e. $l_i = \widehat{l}'_i$

   a) $l'_{i+1} \notin L_P$. Then $l_{i+1} = l'_{i+1}$. If $t_i \neq t_x$ then $t_i = (\widehat{l}'_i, \alpha, \varphi, R, l'_{i+1})$ (line 12). If $t_i = t_x \wedge \nu'_i(x) \models lb(x)$ then $\bowtie \in \{<, \leq\}$. $\nu'_i(x) = \nu_i(x)$, therefore $\nu_i(x) \models lb(x)$. $t_i = (\widehat{l}'_i, \alpha, \varphi \wedge lb(x), R \cup \{x\}, l'_{i+1})$ (line 18). If $t_i = t_x \wedge \nu'_i(x) \not\models lb(x)$ then $\bowtie \in \{>, \geq\}$. $\nu'_i(x) = \nu_i(x)$, therefore $\nu_i(x) \not\models lb(x)$. $t_i = (\widehat{l}'_i, \alpha, \varphi \wedge \neg lb(x), R, l'_{i+1})$ (line 21).

   b) $l'_{i+1} \in L_P, x \notin reset(t'_i)$ and $\nu_i(x)' \models lb(x)$. Then $l_{i+1} = \check{l}'_{i+1}$. If $t'_i \neq t_x$ then $t_i = (\widehat{l}'_i, \alpha, \varphi \wedge lb(x), R \cup \{x\}, \check{l}'_{i+1})$ (line 10). If $t'_i = t_x$ then $t_i = (\widehat{l}'_i, \alpha, \varphi \wedge lb(x), R \cup \{x\}, \check{l}'_{i+1})$ (line 18).

   c) $l'_{i+1} \in L_P, x \in reset(t'_i)$ and $\nu_i(x)' \not\models lb(x)$, i.e. $l_{i+1} = \widehat{l}'_{i+1}$. Note that because $x \notin reset(t'_i)$ and $\nu_i(x) = \nu_i(x)'$, then also $\nu_{i+1}(x) = \nu_{i+1}(x)'$. If $t'_i \neq t_x$ then

$t_i = (\widehat{l}'_i, \alpha, \varphi \wedge \neg lb(x), R, \widehat{l}'_{i+1})$ (line 10). If $t'_i = t_x$ then $t_i = (\widehat{l}'_i, \alpha, \varphi \wedge \neg lb(x), R, \widehat{l}'_{i+1})$ (line 21).

d) $l'_{i+1} \in L_P, x \in reset(t'_i)$, i.e. $l_{i+1} = \dddot{l}'_{i+1}$. If $t'_i \neq t_x$ then $t_i = (\widehat{l}'_i, \alpha, \varphi, R, \dddot{l}'_{i+1})$ (line 8). If $t'_i = t_x$ and $\nu_i(x)' \models lb(x)$ then $t_i = (\widehat{l}'_i, \alpha, \varphi \wedge lb(x), R \cup \{x\}, \dddot{l}'_{i+1})$ (line 18). If $t'_i = t_x$ and $\nu_i(x)' \not\models lb(x)$ then $t_i = (\widehat{l}'_i, \alpha, \varphi \wedge \neg lb(x), R, \dddot{l}'_{i+1})$ (line 21).

4. $x \in reset(t_{i-1})$ or $i = 0$, i.e. $l_i = \dddot{l}'_i$. In such case, also $\nu_i(x) = \nu_i(x)'$.

   a) $l'_{i+1} \notin L_P$. Then $l_{i+1} = l'_{i+1}$ If $t_i \neq t_x$ then $t_i = (\dddot{l}'_i, a, \varphi, R, l'_{i+1})$ (line 13). If $t_i = t_x \wedge \nu'_i(x) \models lb(x)$ implies $\bowtie \in \{<, \leq\}$. $\nu'_i(x) = \nu_i(x)$, therefore $\nu_i(x) \models lb(x)$. $t_i = (\dddot{l}'_i, \alpha, \varphi \wedge lb(x), R \cup \{x\}, l'_{i+1})$ (line 18). If $t_i = t_x \wedge \nu'_i(x) \not\models lb(x)$ implies $\bowtie \in \{>, \geq\}$. $\nu'_i(x) = \nu_i(x)$, therefore $\nu_i(x) \not\models lb(x)$. $t_i = (\dddot{l}'_i, \alpha, \varphi \wedge \neg lb(x), R, l'_{i+1})$ (line 21).

   b) $l'_{i+1} \in L_P, x \notin reset(t'_i)$ and $\nu_i(x)' \models lb(x)$. Then $l_{i+1} = \breve{l}'_{i+1}$. If $t'_i \neq t_x$ then $t_i = (\dddot{l}'_i, \alpha, \varphi \wedge lb(x), R \cup \{x\}, \breve{l}'_{i+1})$ (line 11). If $t'_i = t_x$ then $t_i = (\dddot{l}'_i, \alpha, \varphi \wedge g(x), R \cup \{x\}, \breve{l}'_{i+1})$ (line 18).

   c) $l'_{i+1} \in L_P, x \in reset(t'_i)$ and $\nu_i(x)' \not\models lb(x)$, i.e. $l_{i+1} = \widehat{l}'_{i+1}$. Note that because $x \notin reset(t'_i)$ and $\nu_i(x) = \nu_i(x)'$, then also $\nu_{i+1}(x) = \nu_{i+1}(x)'$. If $t'_i \neq t_x$ then $t_i = (\dddot{l}'_i, \alpha, \varphi \wedge \neg lb(x), R, \widehat{l}'_{i+1})$ (line 10). If $t'_i = t_x$ then $t_i = (\dddot{l}'_i, \alpha, \varphi \wedge \neg lb(x), R, \widehat{l}'_{i+1})$ (line 21).

   d) $l'_{i+1} \in L_P, x \in reset(t'_i)$, i.e. $l_{i+1} = \dddot{l}'_{i+1}$. If $t'_i \neq t_x$ then $t_i = (\dddot{l}'_i, \alpha, \varphi, R, \dddot{l}'_{i+1})$ (line 9). If $t'_i = t_x$ and $\nu_i(x)' \models lb(x)$ then $t_i = (\dddot{l}'_i, \alpha, \varphi \wedge lb(x), R \cup \{x\}, \dddot{l}'_{i+1})$ (line 18). If $t'_i = t_x$ and $\nu_i(x)' \not\models lb(x)$ then $t_i = (\dddot{l}'_i, \alpha, \varphi \wedge \neg lb(x), R, \dddot{l}'_{i+1})$ (line 21).

Altogether if $(l'_i, \nu'_i) \xrightarrow{\alpha_i, d_i} (l'_{i+1}, \nu'_{i+1})$ via transition $t'_i = (l'_i, \alpha, \varphi', R', l'_{i+1})$ then $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ via $t_i = (l_i, \alpha, \varphi, R, l_{i+1})$ for any $i \geq 0$, i.e. for each extended run $\rho' = (l'_0, \nu'_0)t'_0(l'_1, \nu'_1)t'_1 \ldots$ for word $\sigma = (\alpha_0, d_0)(\alpha_1, d_1) \ldots$ in $\mathcal{M}_{\mathcal{A}'}$ there exists an extended run $\rho = (l_0, \nu_0)t_0(l_1, \nu_1)t_1 \ldots$ for σ in $\mathcal{M}_{\mathcal{A}}$. Obviously, if ρ' is accepting, ρ is accepting as well.

Let, the other way around, $\rho = (l_0, \nu_0)t_0(l_1, \nu_1)t_1 \ldots$ be an extended run for $\sigma = (\alpha_0, d_0)(\alpha_1, d_1) \ldots$ in $\mathcal{M}_{\mathcal{A}}$ We show that there exists an extended run $\rho' = (l'_0, \nu'_0)t'_0(l'_1, \nu'_1)t'_1 \ldots$ for σ in $\mathcal{M}_{\mathcal{A}'}$.

Let $l'_i = l$ if $l_i \in \{l, \dddot{l}, \breve{l}, \widehat{l}\}$, and assume that

- If $l_i = l$, then $l \notin L_P$

- If $l_i = \ddot{l}$, then $l \in L_P$, $x \in reset(t'_{i-1})$ or $i = 0$. Then also $\nu_i(x) = \nu'_i(x)$.
- If $l_i = \check{l}$, then $l \in L_P$, $x \notin reset(t'_{i-1}), i \neq 0$ and $\nu'_{i-1}(x) \models lb(x)$.
- If $l_i = \widehat{l}$, then $l \in L_P$, $x \notin reset(t'_{i-1}), i \neq 0$ and $\nu'_{i-1}(x) \not\models lb(x)$. Then also $\nu_i(x) = \nu'_i(x)$.

First of all, $l_0 = \ddot{l}_{init}$ in case $l_0 \in L_P$, and $l_0 = l_{init}$ otherwise. Anyway, $\nu'_0(x) = \nu_0(x)$. Let $(l_i, \nu_i) \xrightarrow{\alpha_i, d_i} (l_{i+1}, \nu_{i+1})$ via $t_i = (l_i, \alpha, \varphi, R, l_{i+1})$. We show that $(l'_i, \nu'_i) \xrightarrow{\alpha_i, d_i} (l'_{i+1}, \nu'_{i+1})$ via $t'_i = (l'_i, \alpha, \varphi', R', l'_{i+1})$ case by case:

1. Let $l_i = l_1$ and $l_1 \notin L_P$.

   a) $l_{i+1} = \ddot{l}_2$. Then t_i is added to T on line 7.
   $t_i = (l_1, \alpha, \varphi, R, \ddot{l}_2)$. Transition t'_i is $(l_1, \alpha, \varphi, R, l_2)$, where $l_2 \in L_P$, $x \in reset(t'_i)$ and $\nu_{i+1}(x) = \nu'_{i+1}(x)$.

   b) $l_{i+1} = l_2$. Then $l_2 \notin L_P$. $t_i = t'_i = (l_1, \alpha, \varphi, R, l_2)$.

   c) l_{i+1} cannot be $\widehat{l}_2$ or $\check{l}_2$ for any l_2.

2. Let $l_i = \ddot{l}_1$, $i = 0$ or $l_1 \in L_P$, $x \in reset(t'_{i-1})$ and $\nu_i(x) = \nu'_i(x)$.

   a) $l_{i+1} = l_2$. Then t_i is to T on line 13, 18 or 21. In all three cases $l_2 \notin L_P$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.

   b) $l_{i+1} = \ddot{l}_2$. Then t_i is added to T on line 9,18 or 21.
   If $t_i = (\ddot{l}_1, \alpha, \varphi, R, \ddot{l}_2)$ is a transition added to T on line 9, then $l_2 \in L_P$ and $x \in R$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.
   If added on line 18, $t_i = (\ddot{l}_1, \alpha, \varphi \wedge lb(x), R \cup \{x\}, \ddot{l}_2)$. Therefore $\nu_i(x) = \nu'_i(x) \models lb(x)$. Then $\nu'_i(x) \models \varphi$ because $\bowtie \in \{<, \leq\}$ and t'_i is $t_x = (l_1, \alpha, \varphi, R, l_2)$.
   If added on line 21, $t_i = (\ddot{l}_1, \alpha, \varphi \wedge \neg lb(x), R, \ddot{l}_2)$. Therefore $\nu_i(x) = \nu'_i(x) \not\models lb(x)$. Then $\nu'_i(x) \models \varphi$ because $\bowtie \in \{>, \geq\}$ and t'_i is $t_x = (l_1, \alpha, \varphi, R, l_2)$.
   Together, in all the three cases $l_2 \in L_P$, $x \in reset(t'_i)$, $\nu_{i+1}(x) = \nu'_{i+1}(x)$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.

   c) $l_{i+1} = \check{l}_2$. Then t_i is added to T on line 11 or 18.
   If $t_i = (\ddot{l}_1, \alpha, \varphi \wedge lb(x), R \cup \{x\}, l_2)$ is a transition added to T on line 11, then $l_2 \in L_P$ and $x \notin R$. Transition t'_i is $(l_1, \alpha, \varphi, R, l_2)$. Also $\nu_i(x) = \nu'_i(x) \models lb(x)$.
   If $t_i = (\ddot{l}_1, \alpha, \varphi \wedge lb(x), R \cup \{x\}, l_2)$ is added on line 18, $l_2 \in L_P$ and $x \notin R$. $\nu_i(x) = \nu'_i(x) \models lb(x)$. Then $\nu'_i(x) \models \varphi$ because $\bowtie \in \{<, \leq\}$ and t'_i is $t_x = (l_1, \alpha, \varphi, R, l_2)$.

Together, in both cases $l_2 \in L_P, x \notin reset(t'_i), \nu'_i(x) \models lb(x)$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.

d) $l_{i+1} = \widehat{l_2}$. Then t_i is added to T on line 11 or 21.

If $t_i = (\ddot{l}_1, \alpha, \varphi \wedge \neg lb(x), R, \widehat{l_2})$ is a transition added to T on line 11, then $l_2 \in L_P$ and $x \notin R$. Transition t'_i is $(l_1, \alpha, \varphi, R, l_2)$. $\nu_i(x) = \nu'_i(x) \models \neg lb(x)$. Also $\nu_{i+1}(x) = \nu'_{i+1}(x)$.

If $t_i = (\ddot{l}_1, \alpha, \varphi \wedge \neg lb(x), R, l_2)$ is added on line 21, $l_2 \in L_P$ and $x \notin R$. $\nu_i(x) = \nu'_i(x) \models \neg lb(x)$. Then $\nu'_i(x) \models \varphi$ because $\bowtie \in \{>, \geq\}$ and t'_i is $t_x = (l_1, \alpha, \varphi, R, l_2)$. Also $\nu_{i+1}(x) = \nu'_{i+1}(x)$.

Together, in both cases $l_2 \in L_P, x \notin R$, $\nu'_i(x) \neg \models lb(x)$, $\nu_{i+1}(x) = \nu'_{i+1}(x)$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$

3. Let $l_i = \check{l}_1$, $i \neq 0$, $l_1 \in L_P$, $x \notin reset(t'_{i-1})$ and $\nu'_{i-1}(x) \models lb(x)$.

a) $l_{i+1} = l_2$. Then t_i is added to T on line 12 or 18. In both cases $l_2 \notin L_P$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.

b) $l_{i+1} = \ddot{l}_2$. Then t_i is added to T on line 8 or 18.

If $t_i = (l_1, \alpha, \varphi, R, \ddot{l}_2)$ is a transition added to T on line 8, transition t'_i is $(l_1, \alpha, \varphi, R, l_2) \neq t_x$, where $l_2 \in L_P$, $x \in reset(t'_i)$ and $\nu_{i+1}(x) = \nu_{i+1}(x)'$.

If added on line 18, $t_i = (l_1, \alpha, \varphi, R \cup \{x\}, \ddot{l}_2)$, then $l_2 \in L_P$, and $x \in R$. Because $\nu'_{i-1}(x) \models lb(x)$ and $x \notin reset(t_{i-1})$, then also $\nu'_i(x) \models lb(x)$ and $\nu'_i(x) \models \varphi$ because $\bowtie \in \{<, \leq\}$. Transition t'_i is $t_x = (l_1, \alpha, \varphi, R, l_2)$.

Together, in both cases $x \in reset(t'_i)$, $l_2 \in L_P$, $\nu_{i+1}(x) = \nu'_{i+1}(x)$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.

c) $l_{i+1} = \check{l}_2$. Then t_i is added to T on line 10 or 18.

If $t_i = (l_1, \alpha, \varphi, R \cup \{x\}, l_2)$ is a transition added to T on line 10, then $l_2 \in L_P$ and $x \notin R$. Transition t'_i is $(l_1, \alpha, \varphi, R, l_2)$. $\nu'_i(x) \models lb(x)$, because $\nu'_{i-1}(x) \models lb(x)$ and $x \notin reset(t'_{i-1})$.

If t_i is added on line 18, $l_2 \notin L_P$ or $x \notin R$. Because $\nu'_{i-1}(x) \models lb(x)$ and $x \notin reset(t'_{i-1})$, then also $\nu'_i(x) \models lb(x)$ and $\nu'_i(x) \models \varphi$ because $\bowtie \in \{<, \leq\}$. Then t'_i is $t_x(l_1, \alpha, \varphi, R, l_2)$.

Together, in both cases $l_2 \in L_P, x \notin reset(t'_i), \nu'_i(x) \models lb(x)$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.

d) l_{i+1} cannot be $\widehat{l_2}$ for any l_2.

4. Let $l_i = \widehat{l_1}$, $i \neq 0$, $l_1 \in L_P$, $x \notin reset(t'_{i-1})$, $\nu'_{i-1}(x) \not\models lb(x)$ and $\nu_i(x) = \nu_i(x)'$.

a) $l_{i+1} = l_2$. Then t_i is added to T on line 12,18 or 21. In all three cases $l_2 \notin L_P$, $t'_i = (l_1, \alpha, \varphi, R, l_2)$..

b) $l_{i+1} = \ddot{l}_2$. Then t_i is added to T on line 8, 18 or 21.

If $t_i = (\widehat{l}_1, \alpha, \varphi, R, \ddot{l}_2)$ is a transition added to T on line 8, then $l_2 \in L_P$ and $x \in \mathit{reset}(t'_i)$. Transition t'_i is $(l_1, \alpha, \varphi, R, l_2) \neq t_x$ and $\nu_{i+1}(x) = \nu_{i+1}(x)'$.

If added on line 18, then $t_i = (\widehat{l}_1, \alpha, \varphi \wedge lb(x), R \cup \{x\}, \ddot{l}_2)$, $l_2 \in L_P$ and $x \in \mathit{reset}(t'_i)$. $\nu_i(x) = \nu_i(x)' \models lb(x)$ and $\nu'_i(x) \models \varphi$ because $\bowtie \in \{<, \leq\}$. Transition t'_i is $t_x = (l_1, \alpha, \varphi, R, l_2)$.

If added on line 21, then $t_i = (\widehat{l}_1, \alpha, \varphi \wedge \neg lb(x), R, \ddot{l}_2)$, $l_2 \in L_P$ and $x \in R$. $\nu_i(x) = \nu_i(x)' \models \neg lb(x)$ and $\nu'_i(x) \models \varphi$ because $\bowtie \in \{>, \geq\}$. Transition t'_i is $t_x = (l_1, \alpha, \varphi, R, l_2)$.

Together, in all the three cases $x \in \mathit{reset}(t'_i)$, $l_2 \in L_P$, $\nu_{i+1}(x) = \nu'_{i+1}(x)$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.

c) $l_{i+1} = \breve{l}_2$. Then t_i is added to T on line 10 or 18.

If $t_i = (\widehat{l}_1, \alpha, \varphi \wedge lb(x), R \cup \{x\}, l_2)$ is a transition added to T on line 10, then $l_2 \in L_P$ and $x \notin R$. Transition t'_i is $(l_1, \alpha, \varphi, R, l_2)$. $\nu_i(x) = \nu'_i(x) \models lb(x)$.

If $t_i = (\widehat{l}_1, \alpha, \varphi \wedge lb(x), R \cup \{x\}, l_2)$ is added on line 18, $l_2 \in L_P$ and $x \notin R$. $\nu_i(x) = \nu'_i(x) \models lb(x)$. Then also $\nu'_i(x) \models \varphi$ because $\bowtie \in \{<, \leq\}$ and transition t'_i is $(l_1, \alpha, \varphi, R, l_2) \neq t_x$.

Together, in both cases $l_2 \in L_P, x \notin \mathit{reset}(t'_i), \nu'_i(x) \models lb(x)$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.

d) $l_{i+1} = \widehat{l}_2$. Then t_i is added to T on line 10 or 21.

If $t_i = (\widehat{l}_1, \alpha, \varphi \wedge \neg lb(x), R, \widehat{l}_2)$ is a transition added to T on line 10, then $l_2 \in L_P$ and $x \notin R$. $\nu_i(x) = \nu'_i(x) \models \neg lb(x)$. Transition t'_i is $(l_1, \alpha, \varphi, R, l_2) \neq t_x$. Also $\nu_{i+1}(x) = \nu'_{i+1}(x)$.

If $t_i = (\widehat{l}_1, \alpha, \varphi \wedge \neg lb(x), R, \widehat{l}_2)$ is added on line 21, $l_2 \in L_P$ and $x \notin R$. $\nu_i(x) = \nu'_i(x) \models \neg lb(x)$. Then also $\nu'_i(x) \models \varphi$ because $\bowtie \in \{>, \geq\}$ and transit on t'_i is $t_x = (l_1, \alpha, \varphi, R, l_2)$. Also $\nu_{i+1}(x) = \nu'_{i+1}(x)$.

Together, in both cases $l_2 \in L_P, x \notin R$, $\nu'_i(x) \neg \models lb(x)$, $\nu_{i+1}(x) = \nu'_{i+1}(x)$ and $t'_i = (l_1, \alpha, \varphi, R, l_2)$.

Altogether, we have shown that for each extended run $\rho' = (l'_0, \nu'_0)t'_0(l'_1, \nu'_1)t'_1 \ldots$ for σ in $\mathcal{M}'_{\mathcal{A}}$ there exists an extended run $\rho = (l_0, \nu_0)t_0(l_1, \nu_1)t_1 \ldots$ for σ in $\mathcal{M}_{\mathcal{A}}$ and vice versa. Obviously, if ρ' is accepting, then ρ is accepting as well and the other way around. $\square$

Lemma 5.6. *Assertion on line 11 in Algorithm 1 holds.*

Proof. Follows directly from lemmas 5.2, 5.3, 5.4. $\square$

Lemma 5.7. *Assertion on line 9 in Algorithm 1 holds.*

Proof. Inductively. For $\mathtt{Done} = \emptyset$ holds trivially. Let us consider the iteration of the cycle 4-9 in Algorithm 1 for the variable x. For each $t_N \in T_N$ we have $x \in \mathtt{reset}(t_N)$. Furthermore, each transition $t_P \in T_P$ is processed in procedure SPLITINTOCASES(x) such that it is replaced with transitions of form t'_P, $x \in \mathtt{reset}(t'_P)$ or $\mathtt{constraint}(t'_P)$ contains bound $x \bowtie c$, where $\bowtie \in \{>, \geq\}$. This means that for *each* transition $t \in T$ on line 9 in Algorithm 1 it holds that $\mathtt{constraint}(t)$ contains bound of form $x \bowtie c$, where $\bowtie \in \{>, \geq\}$, or $x \in \mathtt{reset}(t)$. The iteration of the cycle 4-9 in 1 for the counter x does not change any resets or constrains of counter $y \neq x \in D$. Hence, each counter $x \in \mathtt{Done}$ is in $\mathcal{A}$ in normal form. $\square$

Lemma 5.8. *Assertion on line 12 in Algorithm 1 holds.*

Proof. Follows directly from Lemma 5.7. $\square$

Complexity of Normalization

The time complexity of the procedure ONECONSTRAINTONVARIABLE is $\mathcal{O}(|T| \cdot n)$, where $|T|$ is the number of transitions, and n denotes the overall number of occurrences of all degradation variables in the input BADC (i.e. $\Sigma_{d \in D} \Sigma_{t \in T}\, m$, where D is the set of degradation variables, and d is m-bounded in t). The number of iterations of the cycle 4-9 of the algorithm 1 is n. Both procedures RESETWHEREPOSSIBLE(X) and SPLITINTO-LAYERS(X) take $\mathcal{O}(|L'| + |T'|)$ time, where $|L'|$ is the number of locations and $|T'|$ is the number of transitions in the modified BADC just before the procedures are performed. During the procedures, each location is replaced with at most 4 new ones and each transition is replaced with at most 4 new ones. Altogether the worst-time complexity of the transformation is

$$\mathcal{O}(|T| \cdot n) + \mathcal{O}(\sum_{i=0}^{n} 4^i \cdot (|T| + |L|)) = \mathcal{O}(2^{2n} \cdot (|L| + |T|)),$$

where $|L|$ is the number of locations in the input BADC, and $|T|$, and n are defined as above.

6 Quantitative Linear Properties of MDPs

This section raises the question about the parallel between the systems with degradation and the Markov decision processes (MDPs) [2, 9, 16] as well as about the relationship between probabilistic logic PLTL, PCTL, PCTL* and the quantitative linear properties formalized via BADCs. It is easy to see that an MDP is just a special case of a system with degradation. However, Büchi automata with degradation constraints can distinguish otherwise indistinguishable MDPs.

Current model checking of MDPs aims particularly on properties expressed in *LTL* (Linear Temporal Logic) [15], *PCTL* (Probabilistic Computation Tree Logic) [11] and *PCTL** [1]. The problem of quantitative LTL model checking of an MDP is to determine minimal and/or maximal probability (w.r.t. all possible schedulers) of a set of paths in the MDP that satisfy the LTL formula. PCTL and PCTL* verification gives an answer to the question whether a given MDP satisfies a PCTL (or PCTL*) state formula.

MDPs as Transition Systems with Degradation

Let us consider a transition system with degradation $\mathcal{M} = (S, Act, \rightarrow, S_{init}, AP, \mathcal{L})$ and extend it with the following restrictions on the transition relation $\rightarrow$:

- for all $s_1, s_2 \in S$, $a \in Act$ there is at most one d such that $(s_1, a, d, s_2) \in \rightarrow$
- for all $s_1 \in S$, $a \in Act : \sum_{(s_1, a, d, s_2) \in \rightarrow} d = 1$ or 0.

We may think of the probability as a quality of the system that degrades in time. If probabilities are interpreted as degradations, the restricted transition systems with degradation are syntactically equivalent to Markov decision processes.

MDPs and Temporal Properties

In this subsection we demonstrate two MDPs which cannot be distinguished by any LTL, PCTL or even PCTL* formulas.

First, let us consider the MDP $\mathcal{M} = (S = \{s, t\}, Act = \{\alpha, \beta\}, P, s, \{a\}, \mathcal{L})$ as illustrated in Figure 9.a.

We show that the minimal and the maximal probability of a set of paths originating at a particular state and satisfying a linear temporal property is always either 0 or 1.

Observation 6.1. *Let* η *be an arbitrary scheduler for* $\mathcal{M}$*. Then the Markov chain induced by* η *is* $\mathcal{M}_\eta = (S^+, P_\eta, s_{init}, AP, \mathcal{L}_\eta)$*, where*

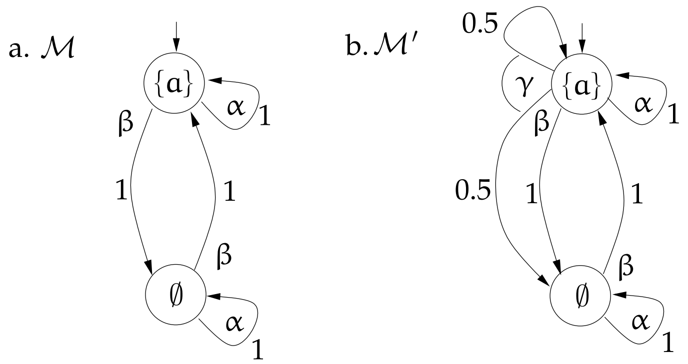


Figure 9: MDPs indistinguishable by any LTL, PCTL, or PCTL* formula.

$$P_\eta(s_0 \dots s_n, s_0 \dots s_n s_{n+1}) = P(s_n, \eta(s_0 \dots s_n), s_{n+1})$$
$$= \begin{cases} 1 & \text{if } s_n = s_{n+1} \text{ and } \eta(s_0 \dots s_n) = \alpha \text{ or} \\ & \text{if } s_n \neq s_{n+1} \text{ and } \eta(s_0 \dots s_n) = \beta \\ 0 & \text{otherwise} \end{cases}$$

and $\mathcal{L}_\eta(s_0 \dots s_n) = \mathcal{L}(s_n)$.

Note that for each state $s_0 \dots s_n$ in $\mathcal{M}_\eta$ there is exactly one state $s_0 \dots s_n s_{n+1}$ such that $P_\eta(s_0 \dots s_n, s_0 \dots s_n s_{n+1}) > 0$. In other words, there is exactly one path $\pi = (s_0)(s_0 s_1)(s_0 s_1 s_2) \dots$ in $\mathcal{M}_\eta$ originating at s_0. The set of all paths originating at s_0 in $\mathcal{M}_\eta$ is $\{\pi\}$ and its probability is 1. Similarly, there is exactly one path $\pi = (s_0 s_1)(s_0 s_1 s_2)(s_0 s_1 s_2 s_3) \dots$ in $\mathcal{M}_\eta$ originating at $s_0 s_1$ and the probability of the set of all paths $\{\pi\}$ originating at $s_0 s_1$ is equal to 1.

Let us consider the language L of words over the alphabet $2^{\{a\}}$ representing a linear temporal property. For each symbol $\gamma \in 2^{\{a\}}$ we distinguish three possible cases:

1. there is no word in L starting with γ,
2. L contains all words starting with γ, or
3. $\exists\ \sigma_1 = \gamma(2^{\{a\}})^\omega \in L$ and $\exists\ \sigma_2 = \gamma(2^{\{a\}})^\omega \notin L$

To simplify the following discussion we denote by symbol u the state s of $\mathcal{M}$ in case of $\gamma = \{a\}$ and the state t otherwise (i.e. if $\gamma = \emptyset$).

Lemma 6.2. *Suppose there is no word in* L *starting with the symbol γ. Then the minimal and the maximal probability of the set of paths of $\mathcal{M}$ originating at u with trajectories in* L *is 0.*

Proof. Directly from the fact that there is no path $\pi = u\alpha_0 s_1 \alpha_1 s_2 \alpha_2 \dots$ in $\mathcal{M}$ with the trajectory $\mathcal{L}(u)\mathcal{L}(s_1)\mathcal{L}(s_2) \dots \in L$. □

Lemma 6.3. *Suppose* L *contains all words starting with* γ. *Then the minimal and the maximal probability of the set of paths of* $\mathcal{M}$ *originating at* u *with trajectories in* L *is 1.*

Proof. For each path $\pi = u\alpha_0 s_1 \alpha_1 s_2 \alpha_2 \dots$ in $\mathcal{M}$ originating at u it holds that the corresponding trajectory $\mathcal{L}(u)\mathcal{L}(s_1)\mathcal{L}(s_2)\dots = \gamma\mathcal{L}(s_1)\mathcal{L}(s_2)\dots$ is present in L. Thus the probability of the set of paths originating at u with trajectories in L is 1 for any possible scheduler η in $\mathcal{M}$. □

Lemma 6.4. *Let us suppose there are* $\sigma_1 = \gamma(2^{\{a\}})^\omega \in L$ *and* $\sigma_2 = \gamma(2^{\{a\}})^\omega \notin L$. *Then the maximal probability of the set of paths of* $\mathcal{M}$ *originating at* u *with trajectories in* L *is 1 and the minimal probability is 0.*

Proof. We define schedulers η_1 and η_2 for $\mathcal{M}$ such that the trajectory of the paths originating at u in the induced Markov chain $\mathcal{M}_{\eta_1}$ and $\mathcal{M}_{\eta_2}$ are σ_1 and σ_2, respectively.

Let $\sigma_1 = \gamma A_1 A_2 \dots$. The scheduler η_1 is defined by the prescription ($s_0 = u$)

$$\eta_1(s_0 \dots s_n) = \begin{cases} \alpha & \text{if } \mathcal{L}(s_0 \dots s_n) = A_{n+1} \\ \beta & \text{if } \mathcal{L}(s_0 \dots s_n) \neq A_{n+1}. \end{cases}$$

The scheduler η_1 unambiguously determines the only path in $\mathcal{M}_{\eta_1}$ and the trajectory of this path is σ_1. This fact together with the Observation 6.1 implies that the maximal probability of the set of paths of $\mathcal{M}$ originating at u with trajectories in L is 1.

For the minimal probability and the scheduler η_2 the arguments are similar. □

L	probability of the set of paths with trajectories in L			
	originating at s		originating at t	
	min	max	min	max
$\{a\}(2^{\{a\}})^\omega \cap L = \emptyset$	0	0	–	–
$\{a\}(2^{\{a\}})^\omega \subseteq L$	1	1	–	–
$\{a\}(2^{\{a\}})^\omega \cap L \cap co-L \neq \emptyset$	0	1	–	–
$\emptyset(2^{\{a\}})^\omega \cap L = \emptyset$	–	–	0	0
$\emptyset(2^{\{a\}})^\omega \subseteq L$	–	–	1	1
$\emptyset(2^{\{a\}})^\omega \cap L \cap co-L \neq \emptyset$	–	–	0	1

Table 1: Summary of results, Lemma 6.2 - 6.4

We summarize results given by Lemmas 6.2 - 6.4 in Table 1. The symbol '–' indicates that we cannot say anything about the probability bound. Note that any language L satisfies exactly one of the three cases given on the first three lines of the table and exactly one of the three cases given on the second three lines of the table. Therefore, given an arbitrary linear temporal property L, the minimal and the maximal probability for the system M can be completely determined using just the table.

Let us now consider an MDP $\mathcal{M}'$ in Figure 9.b. Using similar arguments as for the MDP $\mathcal{M}$ we obtain the very same results about probability bounds for linear temporal properties for $\mathcal{M}'$, for the summary see Table 1.

The minimal and the maximal probabilities of the set of paths originating at the initial states s and s' with trajectories in L are the same for the MDP $\mathcal{M}$ and the MDP $\mathcal{M}'$, respectively. The same observation holds for the states t and t'. Thus there is a one-to-one correspondence between the states s and s' and also between the states t and t'. Therefore MDPs $\mathcal{M}$ and $\mathcal{M}'$ cannot be distinguished neither by qualitative verification nor by quantitative verification with any LTL formula.

Furthermore, if φ is a CTL path formula then both the minimal and the maximal probability of the set of paths satisfying φ is always either 0 or 1 for all the states both in $\mathcal{M}$ and $\mathcal{M}'$. Hence, for any PCTL or PCTL* formula $P_{\bowtie p}\varphi$ it holds that $\mathcal{M} \models P_{\bowtie p}\varphi \Leftrightarrow \mathcal{M}' \models P_{\bowtie p}\varphi$. As a result, the difference between $\mathcal{M}$ and $\mathcal{M}'$ cannot be captured by any PCTL or PCTL* formula.

MDPs and Quantitative Linear Properties

Now we are to define a quantitative linear property which allows us to distinguish the Markov decision processes $\mathcal{M}$ and $\mathcal{M}'$. The property is specified by the BADC in Figure 10. The property captures the existence of a path with the trajectory $\{a\}\emptyset^\omega$ such that the amount of degradation (probability) between the state s (s') and the first next occurrence of the state t (t', respectively) is at most 0.7. This property is false for $\mathcal{M}$ (there is only one path with the trajectory $\{a\}\emptyset^\omega$ and the amount of degradation is 1), but is true for $\mathcal{M}'$ (there is a path where the amount of degradation is 0.5).

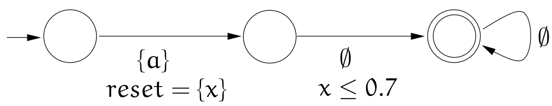


Figure 10: BADC distinguishing the two MDPs.

Using BADCs for expressing properties of MDPs brings us a new possibility to check for the presence of a specific path with a certain probability contribution. See for example the MDP as depicted in Figure 11. The probability of reaching s_1 from s_0 is 1 for all

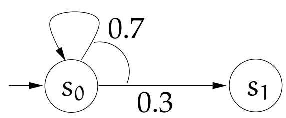


Figure 11: MDP demonstrating interesting BADC property.

(there is only 1) schedulers. Every finite path from s_0 to s_1 (there are infinitely many of them) contribute to the resulting probability measure with some portion. With BADC approach we can, for example, verify that the mentioned portion exceeds 0.2 for some paths, but is at most 0.3 for all paths.

7 Conclusions

Degradation phenomenon as presented in this paper is important from two different points of view. First, it allows system designers to capture and analyze new kind of qualities of their systems, which itself is quite interesting. A second aspect is that the new degradation approach provides a new theoretical way to describe and analyze quantitative linear properties for probabilistic systems, such as MDPs. A limited-degradation-response property is a nice example of a property evaluated over a single run, hence a property that cannot be expressed in any formalism built upon some probability measures. Linear properties and LTL model checking in particular, are well established and used-in-practice formalism. A sort of linear property verification approach for probabilistic systems has been missing so far.

We stress that the degradation cannot be easily modeled using other well known formalisms. Many of other formalisms are either too restrictive to express and check a limited-degradation-response properties, e.g. the standard non-deterministic systems or MDPs. Other formalisms are so rich that a general model checking procedure is undecidable, which is the case of e.g. general hybrid systems, other formalism are simply focused to other quantitative aspects of systems like, e.g. real-time model checking, or model checking rewards.

A straightforward extension is to define a sort of extended linear temporal logic that would allow us to express the desired degradation properties as formulas. For example, the limited-degradation-response property could be stated in an LTL like formalism as follows: $\mathsf{G}(A \implies \mathsf{F}_{\leq 0.8}B))$. An inseparable part of this task is also to design a transformation procedure that would for a given formula produce the corresponding normalized BADC. Finally, let us mention that we have implemented a prototype model checker that is able to verify MDPs against properties given as normalized BADCs on top of our verification tool set DiVinE [3] allowing thus to employ parallel architectures to verify large-scale systems. The models to be verified by DiVinE model checker are given as networks of asynchronously communicating extended finite automata. For the purpose of verification of systems with degradation, we only extended individual automata with the possibility of specification of individual degradation constants.

References

[1] Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Proceedings of the 7th International Conference on Computer Aided Verification*, pages 155–165, London, UK, 1995. Springer-Verlag.

[2] Christel Baier and Joost P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[3] J. Barnat, L. Brim, I. Černá, P. Moravec, P. Ročkai, and P. Šimeček. DiVinE – A Tool for Distributed Verification (Tool Paper). In *Computer Aided Verification*, volume 4144/2006 of *LNCS*, pages 278–281. Springer Berlin / Heidelberg, 2006.

[4] J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. Probdivine-mc: Multi-core ltl model checker for probabilistic systems. In *QEST '08: Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, pages 77–78, Washington, DC, USA, 2008. IEEE Computer Society.

[5] Gerd Behrmann, Alexandre David, Kim G. Larsen, Oliver Möller, Paul Pettersson, and Wang Yi. UPPAAL - present and future. In *Proc. of 40th IEEE Conference on Decision and Control*. IEEE Computer Society Press, 2001.

[6] F. Ciesinski and C. Baier. LiQuor: A tool for Qualitative and Quantitative Linear Time analysis of Reactive Systems. In *Proc. of QEST'06*, pages 131–132. IEEE Computer Society, 2006.

[7] Constantin Courcoubetis, Moshe Y. Vardi, Pierre Wolper, and Mihalis Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1(2/3):275–288, 1992.

[8] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. Model checking discounted temporal properties. *Theor. Comput. Sci.*, 345(1):139–170, 2005.

[9] C. Derman. *Finite State Markovian Decision Processes*. Academic Press, Inc., Orlando, FL, USA, 1970.

[10] Kathi Fisler, Ranan Fraer, Gila Kamhi, Moshe Y. Vardi, and Zijiang. Is there a best symbolic cycle-detection algorithm. In *In Proc. Tools and Algorithms for Construction and Analysis of Systems, volume 2031 of LNCS*, pages 420–434. Springer, 2001.

[11] Hans Hansson and Bengt Jonsson. A Framework for Reasoning about Time and Reliability. In *IEEE Real-Time Systems Symposium*, pages 102–111, 1989.

[12] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. of TACAS'06*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[13] Gerard J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[14] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Quantitative Evaluation of Systems (QEST)*, pages 243–244. IEEE Computer Society, 2005.

[15] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[16] M. L. Puterman. *Markov Decision Processes-Discrete Stochastic Dynamic Programming*. John Wiley &Sons, New York, 1994.

[17] M.Y. Vardi and P. Wolper. Reasoning about infinite computation paths. *Proceedings of 24th IEEE Symposium on Foundation of Computer Science, Tuscan*, pages 185–194, 1983.

[18] Sergio Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.