Approximate Stream Reasoning with Metric Temporal Logic under Uncertainty

Daniel de Leng, Fredrik Heintz
Department of Computer and Information Science
Linköping University, 581 83 Linköping, Sweden
{daniel.de.leng, fredrik.heintz}@liu.se

Abstract

Stream reasoning can be defined as incremental reasoning over incrementally-available information. The formula progression procedure for Metric Temporal Logic (MTL) makes use of syntactic formula rewritings to incrementally evaluate formulas against incrementally-available states. Progression however assumes complete state information, which can be problematic when not all state information is available or can be observed, such as in qualitative spatial reasoning tasks or in robotics applications. In those cases, there may be uncertainty as to which state out of a set of possible states represents the 'true' state. The main contribution of this paper is therefore an extension of the progression procedure that efficiently keeps track of all consistent hypotheses. The resulting procedure is flexible, allowing a trade-off between faster but approximate and slower but precise progression under uncertainty. The proposed approach is empirically evaluated by considering the time and space requirements, as well as the impact of permitting varying degrees of uncertainty.

1 Introduction

Temporal logics allow us to make statements about propositions across time, making them powerful in areas such as runtime verification. Metric Temporal Logic (MTL) by (Koymans 1990) extends the expressiveness of the well-known Linear Temporal Logic (LTL) (Emerson 1990) by adding metric intervals for the temporal operators. The extension makes it possible to describe bounded intervals for logical formulas, further enhancing their use in practical applications. MTL thus makes it possible to precisely describe complex temporal statements that go beyond the absolutes of the LTL temporal operators. This makes MTL a useful tool in realtime applications such as robotics. While model checking for MTL has been shown to be undecidable (Alur, Feder, and Henzinger 1996), we focus on the computationally simpler task of path checking, in which we check whether a given path satisfies a formula. In this paper, we take a *stream reasoning* approach to path checking, which uses incremental reasoning over incrementally-available information.

The syntactic rewriting technique used is known as *progression* (Bacchus and Kabanza 1996; 1998). Progression works by incrementally taking states from a state sequence and computing a new formula that incorporates this state information using syntactic rewriting. If the new formula holds over the unseen remainder of the state sequence, then the original formula is guaranteed to hold over the complete state sequence. Consequently, the evaluation of an MTL formula through progression is linear in the size of the formula, but the formula may grow exponentially due to the rewritings. A key advantage is that we may terminate the procedure once a formula is determined to be true or false, without having to consider the potentially infinite state sequence.

One key assumption for progression is that the states received are complete, i.e. all propositions have a truth value assigned to them. Essentially, progression requires every state to provide a complete 'snapshot' of the world. This assumption is however unreasonable in many applications for which acquiring such a snapshot is not feasible, e.g. robots relying on local sensor data. *The main contribution of this paper is therefore an approximate progression procedure for path checking with partial states, allowing for a trade-off between precision and space requirements.* We also consider the impact of knowledge concerning the probabilities of individual states in cases where there is uncertainty. This paper is a companion paper to our earlier work (de Leng and Heintz 2018), which laid some of the groundwork we expand upon here.

The key motivation behind supporting progression with multiple hypotheses is not just limited to the possibility that we receive partial states—reasoning with background knowledge can by itself lead to incomplete information. For example, the Region Connection Calculus with eight jointly exhaustive pairwise disjoint (JEPD) relations known as RCC-8 by (Randell, Cui, and Cohn 1992) uses qualitative reasoning based on composition tables that allow us to reduce the uncertainty between the qualitative spatial relations that may exist between regions, without narrowing this relation down to precisely one spatial relation. This effectively means we acquire multiple consistent models, any of which could be the 'true' model, and all of which are valid hypotheses.

The remainder of this paper is organized as follows. In Section 2 we consider some of the related work on progression and partiality. We then give an overview of the preliminaries concerning MTL and progression, including the notation used in this paper, in Section 3. Section 4 discusses the theory behind stochastic partial-state progression, followed by an overview of the related procedure in Section 5. An empirical evaluation of the procedure is presented in Section 6. Finally, the paper concludes with Section 7 with a summary and a discussion of future work.

2 Related Work

Partial-state progression is a useful technique when considering applications such as safe robotics. Progression variants have for example been used for execution monitoring (Kvarnström, Heintz, and Doherty 2008) in autonomous UAV applications, in which path-checking of MTL formulas was used to check whether the execution of a plan is in accordance with expectations.

More recently, Desi et al. (Desai, Dreossi, and Seshia 2017) focused on a combination of model checking and runtime verification for making formal safety guarantees in robot software, where they make use of Signal Temporal Logic (STL) as a language for formalizing logical statements. STL is similar to MTL in extending the temporal operators from LTL to range over time intervals, but instead of propositions it considers inequality checks over quantitative signals. While we focus on binary MTL statements, partial-state progression could be extended to work with STL given that the extra information in STL statements can be utilized.

The recent work by Adolf et al. (Adolf et al. 2017) on stream runtime monitoring in unmanned aircraft systems further shows the need for and interest in the ability to monitor robot systems during runtime for debugging and the monitoring of safety restrictions. Progression of MTL formulas has also been used for monitoring purposes. For example, Basin et al. (Basin, Bhatt, and Traytel 2017; Basin, Krstić, and Traytel 2017) proposed an MTL (and related Metric Dynamic Logic; MDL) monitor for complex event processing which is almost event-rate independent, meaning it can handle a dense stream with high quantities of events occuring within fixed time intervals.

Our approach makes it possible to keep track of the probability of partial-state progression having ended up in some MTL formula given a partially-observed incomplete state sequence. This is somewhat related to the recent work by Medhat et al. (Medhat et al. 2016), who proposed absolute and relative 'counting quantifiers', allowing them to express and monitor constraints that concern a lower or upper bound on a certain number or percentage of instances. Their approach differs from ours in that they extend LTL with counting quantifiers whereas our probability mass exists at the meta-logic level. The ability to consider the probability of having ended up in some MTL formula is also potentially useful in dealing with multiple consistent interpretations when performing qualitative spatial reasoning tasks. The combination of a temporal logic and qualitative spatial reasoning for runtime verification tasks is also related to the recent work by Nenzi et al. (Nenzi et al. 2015), who extended STL to consider spatial information.

Algorithm 1: Classical Progression

```
function PROGRESS ($\phi$, $s_i$) :
if $\phi = \phi_1 \vee \phi_2$ then
    return PROGRESS($\phi_1, s_i$) $\vee$ PROGRESS($\phi_2, s_i$)
else if $\phi = \neg\phi_1$ then
    return $\neg$PROGRESS($\phi_1, s_i$)
else if $\phi = \phi_1 \, \mathcal{U}_I \, \phi_2$ then
    if $I < 0$ then
        return $\bot$
    else if $0 \in I$ then
        return PROGRESS($\phi_2, s_i$) $\vee$ (PROGRESS($\phi_1, s_i$) $\wedge$
          $\phi_1 \, \mathcal{U}_{I-\Delta} \, \phi_2$)
    else
        return PROGRESS($\phi_1, s_i$) $\wedge \phi_1 \, \mathcal{U}_{I-\Delta} \, \phi_2$
    end
else
    if $\phi \in s_i$ then
        return $\top$
    else
        return $\bot$
    end
end
```

3 Classical Progression for MTL

MTL is an extension of LTL with temporal operators ranging over intervals. We denote the set of all MTL *propositional symbols* by $\mathcal{P}$. We define a *state* $s \subseteq \mathcal{P}$ to be a set of true propositions, and its complement $\mathcal{P} \setminus s$ to denote the set of false propositions. A state thus models complete information. Since we are interested in temporal reasoning, we consider sequences of states called *streams*. A stream is denoted by a total ordering $\rho = \{(s_0, t_0), (s_1, t_1), \dots\}$ for states s_i, time-stamps $t_i \in \mathbb{N}$, and time-points $i \in \mathbb{N}$. A *stream prefix* is denoted by $\rho_{\leq\tau} = \{(s,t) \mid t \leq \tau\}$.

An MTL-formula is well-formed iff it adheres to the MTL syntax:

Definition 1 (MTL Syntax)**.** *The syntax for MTL is as follows for atomic propositions* $p \in \mathcal{P}$*, temporal intervals* $I \subseteq [0, \infty]$*, and well-formed formulas (wffs)* ϕ *and* ψ*:*

$$p \mid \neg\phi \mid \phi \vee \psi \mid \phi \, \mathcal{U}_I \, \psi$$

In this paper we also make use of connectives $\{\wedge, \rightarrow, \leftrightarrow\}$ with their classical semantics, as well as the temporal operators 'eventually' $\Diamond_I \phi =_{def} \top \mathcal{U}_I \phi$ and 'always' $\Box_I \phi =_{def} \neg\Diamond_I \neg\phi$, and verdicts 'true' $\top =_{def} p \vee \neg p$ and 'false' $\bot =_{def} \neg\top$. Lastly, the temporal operator intervals may be omitted for cases where $I = [0, \infty]$.

Definition 2 (MTL semantics)**.** *The semantics of MTL are defined recursively for a wff* ϕ *and a stream* ρ *at time* t_i*:*

$$\rho, t_i \models p \text{ iff } p \in s_i \text{ for } p \in \mathcal{P}$$
$$\rho, t_i \models \neg\phi \text{ iff } \rho, t_i \not\models \phi$$
$$\rho, t_i \models \phi \vee \psi \text{ iff } \rho, t_i \models \phi \text{ or } \rho, t_i \models \psi$$
$$\rho, t_i \models \phi \, \mathcal{U}_{[\delta_1, \delta_2]} \, \psi \text{ iff } \exists t_\alpha \in [t_i + \delta_1, t_i + \delta_2] :$$
$$\rho, t_\alpha \models \psi \text{ and } \forall t_\beta \in [t_i, t_\alpha) : \rho, t_\beta \models \phi$$

The goal of path checking is to determine whether a well-formed MTL formula ϕ is satisfied by the suffix of

ρ starting at time-stamp t, written as $\rho, t \models \phi$. It differs from model checking in that the formula is only checked against one path starting from time-stamp t, rather than for every stream suffix satisfied by the model. In this paper, we focus on progression (Bacchus and Kabanza 1996; 1998) with streams of incomplete states as our method for path checking. We refer to their works for a detailed description of the PROGRESS procedure, an updated version of which is listed in Algorithm 1, taking into account the notation used in this paper. In the remainder of this paper, we shorten PROGRESS to PROG for equations and assume Δ, which denotes the time between states, to be constant. One important result we rely on in this work is the correctness result for PROGRESS:

Theorem 1 (Correctness of PROGRESS (Bacchus and Kabanza 1996; 1998))**.** *The PROGRESS procedure is* correct*;*

$$\rho, t_i \models \phi \textit{ iff } \rho, t_{i+1} \models \texttt{PROG}(\phi, s_i)$$

for streams ρ, time-points i, and wffs ϕ.

4 Stochastic Partial-State Progression

Recall that we want to perform progression over streams that may contain incomplete states. To perform ρ-progression, we need a stream ρ and a wff ϕ. We will assume that a stream generator can be described stochastically, using a time sequence of stochastic variables. If we wish to assume that ρ is based on a stationary distribution, these stochastic variables would be equivalent across time-points.

State Universe

Streams are composed of states, which we now assume can be incomplete. We denote *incomplete states* and their associated *incomplete streams* using the hat-notation, i.e. $\widehat{s} \subseteq 2^{\mathcal{P}}$ denotes an incomplete state and $\widehat{\rho} = \{\rho_1, \ldots, \rho_N\}$ denotes an incomplete stream. An incomplete state $\widehat{s}$ may be interpreted as being in DNF, i.e. as a disjunction over the contained-within complete states. Likewise, all $\rho \in \widehat{\rho}$ are exclusively composed of complete states, denoting all permutations of the incomplete states. With the notation in place, we can now define the aforementioned time-independent stochastic variable as the *state universe*.

Definition 3 (State Universe)**.** *The set of states $2^{\mathcal{P}}$ is associated with a time-independent stochastic variable $\mathbf{S}_n$ representing the* state universe *at time-point $n \in \mathbb{N}$:*

$$\mathbf{S}_n \sim \text{Discrete}(\theta_{\mathbf{n}}),$$

where $\theta_n = \{\theta_{n,j}\}_{j=0}^{|2^{\mathcal{P}}|-1}$ represents a probability mass function (pmf) using vector-notation.

We write $P[\mathbf{S}_n = s_j] = \theta_{n,j}$ to denote the probability of drawing a state at time-point $n \in \mathbb{N}$. The probability of complete states s given observed incomplete states $\widehat{s}_n$ is denoted by

$$P[\mathbf{S}_n = s \mid \widehat{s}_n] = \frac{P[\mathbf{S}_n = s]\, \mathbb{I}(s \in \widehat{s}_n)}{\sum_{s' \in \widehat{s}_n} P[\mathbf{S}_n = s']},$$

where $\mathbb{I}$ is a boolean indicator function (similar to Iverson's bracket notation) such that

$$\mathbb{I}(x) = \begin{cases} 1 \text{ if } x \text{ is true,} \\ 0 \text{ if } x \text{ is false.} \end{cases}$$

A complete prefix is defined as a sequence of complete states up to and including time-point $n \in \mathbb{N}$, where we can describe the probability of randomly drawing such a prefix by

$$P[\mathbf{S}_{\leq n} = \rho_{\leq n}] = \prod_{i=0}^{n} P[\mathbf{S}_i = s_i]$$

due to the time-independence of the state universe.

A sequence of *incomplete* states—represented as a set of disjunctive complete prefixes—is called an *incomplete prefix*, denoted by $\widehat{\rho}_{\leq n}$. We denote the probability of drawing incomplete prefixes from $\widehat{\rho}_{\leq n}$ for $n \in \mathbb{N}$ by

$$P[\mathbf{S}_{\leq n} = \rho_{\leq n} \mid \widehat{\rho}_{\leq n}] = \frac{P[\mathbf{S}_{\leq n} = \rho_{\leq n}]\, \mathbb{I}(\rho_{\leq n} \in \widehat{\rho}_{\leq n})}{\sum_{\rho'_{\leq n} \in \widehat{\rho}_{\leq n}} P\left[\mathbf{S}_{\leq n} = \rho'_{\leq n}\right]}.$$

Prefixes and Extensions

We refer to the repeated application of PROGRESS to a complete prefix $\rho_{\leq n}$ as *prefix progression*:

Definition 4 (Prefix Progression)**.** *We denote the repeated application of PROGRESS to an initial formula ϕ, called* prefix progression*, by*

$$\texttt{PROG}^{n}(\phi, \rho) = \texttt{PROG}(\texttt{PROG}^{n-1}(\ldots), s_n),$$

where $n \in \mathbb{N} \cup \{\infty\}$. For base-case $n = 0$ we use $\texttt{PROG}^{0}(\phi, \rho) = \phi$.

Because the MTL semantics is defined over infinite-length streams, we sometimes need to consider the probability of an incomplete prefix $\widehat{\rho}_{\leq n}$ being a model for an MTL statement ϕ at time t_0. Since $\widehat{\rho}_{\leq n}$ is an incomplete prefix for values of $n \in \mathbb{N}$, we extend it to an incomplete stream by appending fully-unknown states, which we denote by superscript ∞:

$$\widehat{\rho}^{\infty}_{\leq n} = \widehat{\rho}_{\leq n} \cup \left\{(2^{\mathcal{P}}, t_i) \mid i > n\right\}.$$

We use the same notation for fully-known prefixes $\rho_{\leq n}$, where the result is denoted by $\rho^{\infty}_{\leq n}$ representing an incomplete stream.

Lemma 1 (Correctness of Prefix Progression)**.** *The application of progression over prefixes is correct relative to the semantics of MTL:*

$$\forall \rho \in \rho^{\infty}_{\leq n}\, [\rho, t_0 \models \phi] \text{ iff } \texttt{PROG}^{n}(\phi, \rho_{\leq n}) = \top$$

for wff ϕ, prefix $\rho_{\leq n}$, and any time-point $n \in \mathbb{N}$.

Proof. Considering both directions separately:

($\Rightarrow$) Assume that for all $\rho \in \rho^{\infty}_{\leq n}$, it holds that $\rho, t_0 \models \phi$. By applying Theorem 1 n times, we obtain $\rho, t_n \models \texttt{PROG}^{n}(\phi, \rho_{\leq n})$. Let us denote $\texttt{PROG}^{n}(\phi, \rho_{\leq n}) = \psi$. Then it must be the case that $\rho', t_n \models \psi$ for *all* possible complete streams ρ' that could be constructed from infinite sequences of incomplete states. So it must be the case that ψ is a tautology, hence $\texttt{PROG}^{n}(\phi, \rho_{\leq n}) = \top$.

($\Leftarrow$) Assume that $\texttt{PROG}^{n}(\phi, \rho_{\leq n}) = \top$. Since this is a tautology, it is therefore the case that $\rho, t_n \models \texttt{PROG}^{n}(\phi, \rho_{\leq n})$ for all streams $\rho \in \rho^{\infty}_{\leq n}$. From Theorem 1 it then follows that $\forall \rho \in \rho^{\infty}_{\leq n}\, [\rho, t_0 \models \phi]$. □

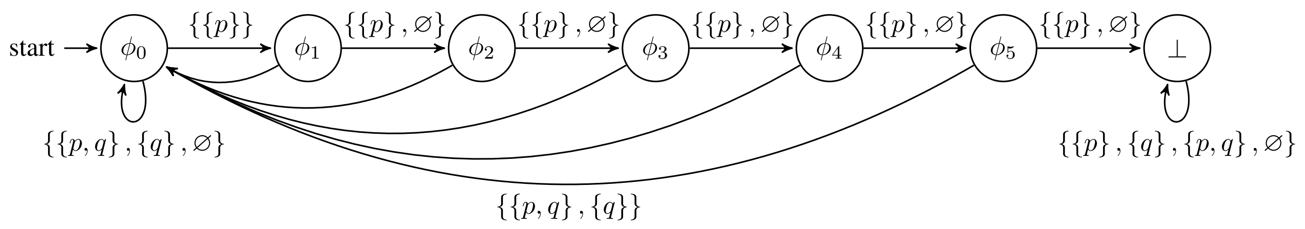

Figure 1: Progression graph for $\Box(p \to (\Diamond_{[0,5]}\ q))$, where vertices represent formulas and edges represent sets of states.

Progression Graphs

Given a potentially-incomplete stream, the repeated application of the PROGRESS procedure to an MTL formula using all possible states yields a potentially large set of formulas in the limit, which can be bounded through the use of formula simplification. We used elimination of conjunctions and disjunctions with verdicts as children, as well as temporal subsumptions, as these patterns commonly arise from the application of the progression procedure. These simplification rules are shown below:

$$\begin{gathered}
\phi \land \top \Rightarrow \phi \\
\phi \land \bot \Rightarrow \bot \\
\phi \lor \top \Rightarrow \top \\
\phi \lor \bot \Rightarrow \phi \\
\Box_{[i,j]}\phi \land \Box_{[i,k]}\phi \Rightarrow \Box_{[i,\max(j,k)]}\phi \\
\Box_{[i,j]}\phi \lor \Box_{[i,k]}\phi \Rightarrow \Box_{[i,\min(j,k)]}\phi \\
\phi\, \mathcal{U}_{[i,j]}\ \psi \land \phi\, \mathcal{U}_{[i,k]}\ \psi \Rightarrow \phi\, \mathcal{U}_{[i,\min(j,k)]}\ \psi \\
\phi\, \mathcal{U}_{[i,j]}\ \psi \lor \phi\, \mathcal{U}_{[i,k]}\ \psi \Rightarrow \phi\, \mathcal{U}_{[i,\max(j,k)]}\ \psi
\end{gathered}$$

Note that the simplification rules for $\Diamond_{[i,j]}\phi$ follow directly from $\top\, \mathcal{U}_{[i,j]}\ \phi$.

As a concrete example, consider the formula $\phi_0 = \Box\left(p \to \left(\Diamond_{[0,5]}\ q\right)\right)$. This formula contains two propositions p and q, which gives four possible states. Repeatedly applying progression for all of these four states yields the following resulting formulas:

$$\phi_i = (\Diamond_{[0,5-i]}\ q) \land \phi_0 \text{ for } i \in [1,5],$$

together with verdict $\bot$. By representing these formulas as vertices and connecting them with sets of states, we obtain the graph shown in Figure 1. The graph connections are designed such that for each edge (ϕ, ψ, s) it is the case that $\texttt{PROG}(\phi, s) = \psi$, thereby allowing the graph structure to graphically encode the formula progression procedure. For a fully-known stream, progression would correspond to transitions between formulas such that we can only be in one given formula at any given time. It also allows us to go beyond standard progression by considering a stochastic interpretation, where we associate a *probability mass* m_n with every formula in the graph. The meaning of probability mass in vertices at some time-point n is the probability of progression having ended up in those associated formulas by time-point n, given an incomplete stream. When a graph is first initialized, all probability mass therefore resides in the vertex associated with the original formula.

Definition 5 (Progression Graph). *A progression graph is a directed graph $G_n(\phi) = (\phi, V, E, m_n)$ at time-point n consisting of a wff ϕ, a set of formulas V for which $\phi \in V$, a set of directed labelled transitions*

$$E = \left\{(v, v', s) \in V \times V \times 2^{\mathcal{P}} \mid \texttt{PROG}(v, s) = v'\right\},$$

and a probability mass function $m_n : V \to [0,1]$ describing a probability distribution over formulas in $v \in V$ defined as

$$m_n(v) = \sum_{\rho_{\leq n} \in \widehat{\rho}_{\leq n}} \Big(P\left[\mathbf{S}_{\leq n} = \rho_{\leq n} \mid \widehat{\rho}_{\leq n}\right] \\ \mathbb{I}(\texttt{PROG}^n(\phi, \rho_{\leq n}) = v)\Big),$$

and $m_0(\phi) = 1$ corresponds to the base-case.

The values of the probability mass at any given time-point thus depend on the observed incomplete stream prefix and the state universe. Finally note that, albeit structurally similar to *deterministic timed automata* (Alur and Dill 1994), progression graphs instead are used to push probability mass between formulas and consequently lack the notion of clocks or accepting states.

Partial-State Progression

The interpretation of a progression graph has thus far been grounded in incomplete prefixes. Combined with the fact that the structure of the progression graphs are grounded in the classical progression procedure, we can ground the interpretation of progression graphs in the MTL semantics as well. Specifically, the probability mass component of progression graphs allows us to quantify the probability of meta-logical statements concerning the semantic entailment relation between incomplete streams and MTL formulas.

Definition 6 (Model Probability for Incomplete Prefixes). *The probability of an extended incomplete prefix $\widehat{\rho}^{\infty}_{\leq n}$ for time-point $n \in \mathbb{N}$ being a model for an MTL statement ϕ at time t_0 is denoted by: $P[\widehat{\rho}^{\infty}_{\leq n}, t_0 \models \phi]$.*

Lemma 2 (Correctness of Model Probability for Incomplete Prefixes). *The model probability for an incomplete prefix is determined by:*

$$P[\widehat{\rho}^{\infty}_{\leq n}, t_0 \models \phi] = \sum_{\rho_{\leq n} \in \widehat{\rho}_{\leq n}} \Big(P\left[\mathbf{S}_{\leq n} = \rho_{\leq n} \mid \widehat{\rho}_{\leq n}\right] \\ \mathbb{I}\left(\forall \rho \in \rho^{\infty}_{\leq n}\left[\rho, t_0 \models \phi\right]\right)\Big).$$

Proof. We have to consider three cases: (1) all streams within $\widehat{\rho}^{\infty}_{\leq n}$ are models of ϕ at time t_0; (2) none are models of ϕ, or; (3) some are models and some are not models. In order for a *prefix* to be a model, all of its extensions to infinite-length streams must be models. Additionally, there are potentially many complete prefixes within the incomplete prefix $\widehat{\rho}_{\leq n}$. The probability $P\left[\mathbf{S}_{\leq n} = \rho_{\leq n} \mid \widehat{\rho}_{\leq n}\right]$ corresponds to the probability of drawing a prefix $\rho_{\leq n}$ from a distribution over $\widehat{\rho}_{\leq n}$. The sum of all of these probabilities sums up to 1. However, we only want to consider those prefixes which are models of ϕ at time t_0. Therefore, we use the indicator function to eliminate the probabilities of prefixes which are not models of ϕ at time t_0, thereby excluding prefixes following cases (2) and (3) while keeping those following (1). $\square$

From Lemma 2, we are able to show that the probability mass for verdict vertices is sound by considering the specific cases of formulas $\top$ and $\bot$.

Theorem 2 (Soundness of Partial-State Progression)**.** *Given a probabilistic progression graph $G_n(\phi)$ for the progression of a wff ϕ starting at time t_0 using a partially-observed incomplete stream described by a prefix $\widehat{\rho}_{\leq n}$ with $n \in \mathbb{N}$:*

$$m_n(\top) = P\left[\widehat{\rho}^{\infty}_{\leq n}, t_0 \models \phi\right],$$
$$m_n(\bot) = P\left[\widehat{\rho}^{\infty}_{\leq n}, t_0 \not\models \phi\right].$$

Proof. From the definition of probability mass, we obtain:

$$m_n(\top) = \sum_{\rho_{\leq n} \in \widehat{\rho}_{\leq n}} \big(P\left[\mathbf{S}_{\leq n} = \rho_{\leq n} \mid \widehat{\rho}_{\leq n}\right] \mathbb{I}(\mathtt{PROG}^n(\phi, \rho_{\leq n}) = \top)\big).$$

Per Lemma 1, the indicator is subject to the equivalence

$$\mathbb{I}(\mathtt{PROG}^n(\phi, \rho_{\leq n}) = \top) = \mathbb{I}(\forall \rho \in \rho^{\infty}_{\leq n}\left[\rho, t_0 \models \phi\right]).$$

By substituting the indicator we can thus rewrite $m_n(\top)$ to

$$m_n(\top) = \sum_{\rho_{\leq n} \in \widehat{\rho}_{\leq n}} \Big(P\left[\mathbf{S}_{\leq n} = \rho_{\leq n} \mid \widehat{\rho}_{\leq n}\right] \mathbb{I}\left(\forall \rho \in \rho^{\infty}_{\leq n}\left[\rho, t_0 \models \phi\right]\right)\Big),$$

which is equivalent to $P\left[\widehat{\rho}^{\infty}_{\leq n}, t_0 \models \phi\right]$ per Lemma 2. The proof for $m_n(\bot)$ follows analogously. $\square$

5 An Approximation Procedure

One problem with progression graphs of formulas with large temporal intervals is that they have a tendency of 'blowing up', thereby requiring a lot of space. To combat this, we build upon our pre-existing approaches to partial-state progression (de Leng and Heintz 2018) that sought to tackle the problem of high space requirements by trading accuracy for space consumption, called *leaky partial-state progression*—but which does not handle probability distributions over a state universe, as we have in this work.

Our proposed procedure combining our stochastic state universe with a leaky progression graph is shown in Algorithm 2, using approximations of the probability mass m_n, denoted by $\widehat{m}_n$, implemented as a map. Similarly, *ttl* (time-to-live) and *expanded* are assumed to be implemented as maps, hence the bracket notation. The initial graph is composed of a single vertex representing the original formula ϕ to be progressed, with the initial probability mass fully contained within this vertex, i.e. $\widehat{m}_0[\phi] = m_0(\phi) = 1$; subsequent approximations may deviate from the true pmf. The *ttl* for the singular vertex is initially assumed to be MAX_TTL. The usage of probability mass over time makes it possible to track the probability of different progressed formulas, including the two different verdicts, over the course of progression of an incomplete stream $\widehat{\rho}$. In subsequent calls to PPROGRESS, the probability mass from the previous iteration becomes a new starting point. The procedure iterates over the set of formulas V and checks the outgoing edges. If the formula has not yet been expanded, it performs the classical PROGRESS procedure to generate and created directed edges to product formulas for all possible states. It then redistributes the probability mass from the parent formula to the *reachable* child formulas in accordance with the probability distribution over states: a child is reachable iff there exists an edge label that is a member of the input state $\widehat{s}$. Finally, PPROGRESS can delete vertices and leak their associated probability mass when the MAX_NODES value is exceeded. The sorting operation in line 24 of Algorithm 2 is intended to illustrate a sorting which orders the set of vertices by ttl first and probability mass second, thus prioritizing those vertices with a low ttl and probability mass for deletion. At the end of each call to PPROGRESS, the resulting progression graph is returned.

As illustrated, keeping track of m_n is possible by applying incremental updates to the pmf based on incrementally-observed incomplete states. The form of these incremental updates is obtained by utilizing the temporal independence of the state universe:

Lemma 3 (Incremental Updates)**.** *An (unapproximated)* update *from m_{n-1} to m_n given an incomplete state $\widehat{s}_n$, where $n \in \mathbb{N}$, follows the relationship*

$$m_n(v) = \sum_{(v',v,s) \in E} \left(m_{n-1}(v')\, P\left[\mathbf{S}_n = s \mid \widehat{s}_n\right]\right)$$

for a (fully-expanded) progression graph $G_n(\phi) = (\phi, V, E, m_n)$.

Proof. We need to show that the full update from Definition 5 for time-point n is equivalent to the full update for time-point $m = n - 1$ followed by an incremental update at time-point n as shown in the above relationship. By plugging Definition 5 into the incremental update rule, we get

$$\sum_{(v',v,s) \in E_n} \left(\sum_{\rho_{\leq m} \in \widehat{\rho}_{\leq m}} \Big(P\left[\mathbf{S}_{\leq m} = \rho_{\leq m} \mid \widehat{\rho}_{\leq m}\right] \mathbb{I}(\mathtt{PROG}^m(\phi, \rho_{\leq m}) = v')\Big) P[\mathbf{S}_n = s \mid \widehat{s}_n] \right).$$

The inner sum ranging over $\rho_{\leq m} \in \widehat{\rho}_{\leq m}$ can be rewritten to instead range over paths in the graph, which incorporates

Algorithm 2: Approximate Partial-State Progression

```
function PPROGRESS ($G_n$, $ttl$, $expanded$, $\widehat{s}$) :
$G_{n+1} \leftarrow (V_n, E_n, [\,])$
foreach $v \in V_n$ do
    $ttl[v] \leftarrow ttl[v] - 1$
    if $\widehat{m}_n[v] > 0$ then
        if $\neg expanded[v]$ then
            foreach $s \in 2^{\mathcal{P}}$ do
                $v' \leftarrow \mathtt{PROGRESS}(v, s, \Delta)$
                if $v' \notin V$ then
                    $V_{n+1} \leftarrow V_{n+1} \cup \{v'\}$
                    $ttl[v'] \leftarrow \mathtt{MAX\_TTL}$
                    $expanded[v'] \leftarrow false$
                end
                $E_{n+1} \leftarrow E_{n+1} \cup \{(v, v', s)\}$
                $expanded[v] \leftarrow true$
            end
        end
        foreach $(v, v', s) \in E_{n+1}$ do
            $\widehat{m}_{n+1}[v'] \leftarrow \widehat{m}_{n+1}[v'] + \widehat{m}_n[v] \times P[\mathbf{S}_n = s \mid \widehat{s}]$
            $ttl[v'] \leftarrow \mathtt{MAX\_TTL}$
        end
    end
end
while $|sort(V_{n+1})| > \mathtt{MAX\_NODES}$ do
    $v \leftarrow head(V_{n+1})$
    $m_{n+1}[v] \leftarrow \mathtt{nil}$
    $ttl[v] \leftarrow \mathtt{nil}$
    $expanded[v] \leftarrow \mathtt{nil}$
    foreach $(v', v, s) \in E_{n+1}$ do
        $expanded[v'] \leftarrow false$
    end
    $V_{n+1} \leftarrow V_{n+1} \setminus \{v\}$
    $E_{n+1} \leftarrow E_{n+1} \setminus \{(w, w', s) \in E_{n+1} \mid w = v \vee w' = v\}$
end
return $G_{n+1}$, $ttl$, $expanded$
```

the indicator function:

$$\sum_{(v',v,s)\in E} \left(\sum_{(\phi,v')\in E^m} \Big(P\left[\mathbf{S}_{\leq m} = \rho_{\leq m} \mid \widehat{\rho}_{\leq m}\right] \Big) P[\mathbf{S}_n = s \mid \widehat{s}_n] \right).$$

We can now collapse the two sums into one sum ranging over paths from ϕ to v, appending the incomplete state $\widehat{s}$ to the incomplete stream $\widehat{\rho}_{\leq m}$ to obtain $\widehat{\rho}_{\leq n}$:

$$\sum_{(\phi,v)\in E^n} \left(P\left[\mathbf{S}_{\leq n} = \rho_{\leq n} \mid \widehat{\rho}_{\leq n}\right] \right).$$

Plugging the indicator function back in we then obtain

$$\sum_{\rho_{\leq n}\in\widehat{\rho}_{\leq n}} \big(P\left[\mathbf{S}_{\leq n} = \rho_{\leq n} \mid \widehat{\rho}_{\leq n}\right] \mathbb{I}(\mathtt{PROG}^n(\phi, \rho_{\leq n}) = v) \big),$$

which matches Definition 5 for $m_n(v)$. □

The MAX_NODES and MAX_TTL values act as parameters that allow us to adjust the precision of the PPROGRESS procedure. If MAX_NODES and MAX_TTL are both set to infinity, the approximated probability mass will match the actual probability mass for each vertex. We can now show the correctness of the PPROGRESS procedure:

Theorem 3 (Correctness of PPROGRESS)**.** *For every progression graph G_{n-1}, PPROGRESS produces an approximated pmf $\widehat{m}_n[v]$ such that*

$$m_n(v) \in [\widehat{m}_n[v], \widehat{m}_n[v] + \ell_n],$$

where $\ell_n = 1 - \sum_{v\in V} \widehat{m}_n[v]$ denotes the leaked probability mass.

Proof. Algorithm 2 starts with an expansion phase on lines 3–23, followed by a shrinking phase on lines 24–34. During expansion, lines 6–17 perform the actual expansion task on the graph, whereas lines 18–21 perform the incremental update from Lemma 3. Note that the incremental updates for non-zero probability mass utilize only children of these associated vertices, which are provided through the expansion of non-zero probability mass nodes prior to performing the incremental updates. This means that $\widehat{m}_n[\phi] = m_n(\phi)$ when $\ell_n = 0$, which is the case whenever $\mathtt{MAX\_NODES} \geq |V|$ after the expansion phase but before the shrinking phase. For the case when $\mathtt{MAX\_NODES} < |V|$ after the expansion phase but before the shrinking phase, the shrinking phase will delete vertices—and leak their associated probability mass—until $\mathtt{MAX\_NODES} = |V|$. That means that the probability mass for any formula ϕ at time-point n will be at least $\widehat{m}_n[\phi]$ and at most $\widehat{m}_n[\phi]$ plus all of the leaked mass ℓ_n. □

6 Empirical Evaluation

The PPROGRESS procedure detailed in Algorithm 2 was implemented in Java[1] and used for empirical evaluation. We performed our experiments using a fourth-generation Intel Xeon E5-1650 CPU (6 cores, 12 threads) with 50GiB of RAM allocated to the JVM. All experiments presume a uniformly-distributed stream universe.

Time and Space Requirements

We first compare the runtime and space requirements given a formula and a stream for varying values for the parameters MAX_TTL and MAX_NODES. Table 1 shows an empirical comparison of the approaches for a formula

$$\phi = \Box\left(\neg p \rightarrow \left(\Diamond_{[0,100]}\left(\Box_{[0,10]} p\right)\right)\right)$$

and a stream in which 80% of the samples are p and the remaining samples are unknown, i.e. $\{\{p\}, \varnothing\}$. Formula ϕ is chosen because it is a member of the class of *response* formulas—denoted by the pattern $\Box_I(\phi \rightarrow \Diamond_J \psi)$—which is a formula class most commonly observed in runtime verification (Dwyer, Avrunin, and Corbett 1999). We marked the best significant results in bold-face. To ensure a fair comparison, the choices for MAX_NODES limit the leaked mass to at most 1% of the total probability mass. For the formula ϕ this corresponds to $\mathtt{MAX\_NODES} \geq 175$ at a step-size of 25.

[1]The jprogress implementation is available at https://github.com/dnleng/jprogress.

MAX_TTL	MAX_NODES	Avg Duration (sec)	$\pm 2\sigma$	Iterations	Max Size	Median Size	Avg Density
∞	∞	143.996	$\pm$3.040	226,867	15,706	15,706	0.024
5	∞	125.709	$\pm$1.275	226,867	11,851	1,162	0.243
1	∞	91.166	$\pm$3.778	226,867	4,074	**335**	**0.665**
∞	250	126.553	$\pm$5.529	226,863	3,858	3,726	0.099
5	250	117.505	$\pm$2.002	226,863	3,855	1,163	0.254
1	250	90.815	$\pm$5.372	226,863	3,722	**335**	**0.665**
∞	225	124.171	$\pm$3.669	226,295	3,480	3,352	0.110
5	225	114.861	$\pm$2.248	226,295	3,480	1,164	0.259
1	225	91.011	$\pm$3.353	226,295	3,361	**335**	**0.665**
∞	200	116.160	$\pm$5.039	225,644	3,105	2,978	0.124
5	200	112,212	$\pm$1.590	225,644	3,105	1,165	0.266
1	200	90.385	$\pm$2.361	225,644	2,999	**335**	**0.665**
∞	175	112.549	$\pm$3.431	**222,599**	2,730	2,604	0.142
5	175	107.680	$\pm$2.083	**222,599**	2,730	1,164	0.277
1	175	89.174	$\pm$3.119	**222,599**	**2,653**	**335**	**0.665**

Table 1: Experimental results for ϕ using a stream with $P[\widehat{s} = \{\{p\}\}] = 0.8$ and $P[\widehat{s} = \{\{p\}, \varnothing\}] = 0.2$, terminating when 99% of mass resides in verdict nodes. The table shows the total duration of progression (averaged over ten runs, showing the 95% probability interval), number of progression calls until termination, maximum combined formula size, median combined formula size, and average mass density in terms of non-zero-mass nodes relative to the total number of nodes.

As expected, the time results show a correlation between the size of the progression graphs and the number of iterations required until termination. As the size of the graph decreases, so does the time it takes to perform a progression, with MAX_TTL being more influential than MAX_NODES under the 1% maximum loss constraint. The procedure consequently performs best with parameters MAX_TTL $=$ 1 and MAX_NODES $=$ 175, followed closely by the parameter sets for which MAX_TTL $=$ 1. For the space usage, we observe that the maximum combined size of the formulas in the graph decreases together with MAX_TTL and MAX_NODES. This behavior is expected as the vertices in the progression graph directly correspond to progressable formulas, and these constraints limit the number of such vertices. We likewise also observe the median size and average density decrease and increase, respectively, as the MAX_TTL and MAX_NODES decrease. The minimum median size observed is 335, which appears to be the most commonly observed graph size measured in the length of the contained formulas. The average density results also show how constraints on the time-to-live and the maximum number of vertices positively impacts the utilization of vertices in the progression graph. However, there is a balance between a high density requiring potentially many time-costly updates to the structure of the graph; and a low density requiring more space on average.

Sensitivity to Partiality

Next, we look into the sensitivity to partiality of the proposed graph-based partial-state progression techniques. In particular, we are interested in the effect of the quality of a stream on the evaluation of a formula. Thus far, we have used stream generators for which $P[\widehat{s} = \{\{p\}\}] = 0.8$ and $P[\widehat{s} = \{\{p\}, \varnothing\}] = 0.2$. In these experiments, we instead let these probabilities vary from $r \in [0.4; 1.0]$ with a step-size of 0.2. The resulting stream generator consequently produces $P[\widehat{s} = \{\{p\}\}] = 1 - r$ and $P[\widehat{s} = \{\{p\}, \varnothing\}] = r$. We again use the formula ϕ for a fair comparison.

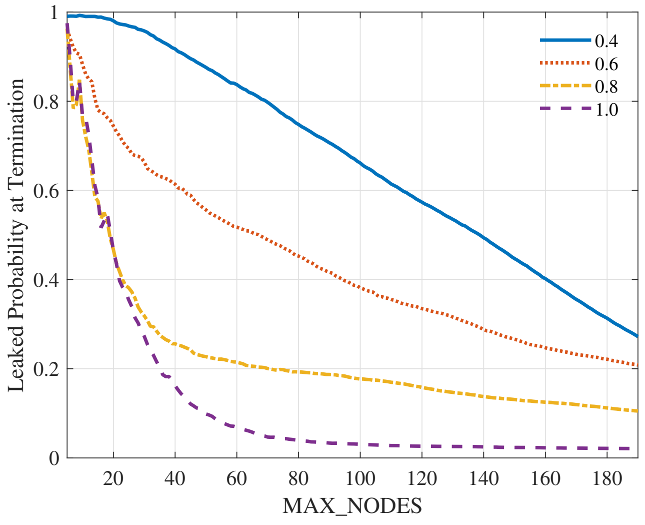

Figure 2: Verdict probability at termination for ϕ using a stream with $P[\widehat{s} = \{\{p\}, \varnothing\}] \in \{0.4, 0.6, 0.8, 1.0\}$.

Figure 2 shows the leaked probability for varying degrees of incompleteness in the produced streams. The graph for the 'false' verdict $\perp$ would be the inverse of Figure 2. We can again observe the dual nature of leaked mass versus $\perp$ verdicts. Additionally, we can observe a varying sensitivity to increasing the probability of incomplete states. As the number of incomplete states in a stream increases (due to their probability of occurring increasing), the inclination of the associated plots increases as well. We can observe an increase of incompleteness lead to a faster decrease of leaked probability as the value of MAX_NODES increases. This is expected behavior because there are now more progression traces of ϕ that result in the early falsification of the formula. The results show that the degree of incompleteness of a stream has a non-trivial impact on the result of progression and affects the choice of MAX_NODES.

7 Conclusions and Future Work

We have presented an approximate graph-based extension of the original MTL progression procedure (Bacchus and Kabanza 1996; 1998) to handle stochastic state information. The PPROGRESS procedure is shown to correctly reflect the probabilities of the verdicts $\top$ and $\bot$ given an MTL formula using an incremental update mechanism. The procedure additionally allows for a trade-off between accuracy and space requirements, by leaking probability mass from certain formulas based on their time-to-live and amount of contained probability mass. Our empirical evaluation illustrates this trade-off and the impact on both accuracy and space requirements.

For future work, we are interested in a number of extensions and applications. The stream universe is subject to a number of strict assumptions we would like to relax in future work. In this paper we assume the state universe to be given; in future work we wish to consider learning the probability distribution over the state universe through observation. Additionally, the PPROGRESS procedure utilizes a MAX_NODES parameter which to a large degree determines the amount of probability that will be leaked. It would be interesting to see if we could predict suitable values for MAX_NODES beforehand given a formula, in order to minimize the leaked probability at termination. In a similar light, we want to perform more detailed empirical evaluations for classes of MTL formulas. Finally, we are interested in looking further into the application of background theories, specifically ASP-based reasoning. This is motivated by the versatility of ASP-based reasoning, which has been shown (Brenton, Faber, and Batsakis 2016) to be able to perform spatial reasoning in RCC-8. We believe that the presented results together with the potential extensions demonstrates the usefulness of this line of partial-state progression, which is particulary beneficial to the area of stream reasoning.

Acknowledgments

This work is partially supported by grants from the National Graduate School in Computer Science, Sweden (CUGS). We would like to thank Mattias Tiger for the valuable discussions which helped improve the notation used in this paper, and the reviewers for their insightful comments.

References

Adolf, F.-M.; Faymonville, P.; Finkbeiner, B.; Schirmer, S.; and Torens, C. 2017. Stream runtime monitoring on UAS. In *Proceedings of the 17th International Conference on Runtime Verification*, 33–49.

Alur, R., and Dill, D. L. 1994. A theory of timed automata. *Theoretical Computer Science* 126(2):183–235.

Alur, R.; Feder, T.; and Henzinger, T. A. 1996. The benefits of relaxing punctuality. *Journal of the ACM (JACM)* 43(1):116–146.

Bacchus, F., and Kabanza, F. 1996. Planning for temporally extended goals. In *Proceedings of the 13th AAAI conference of Artificial Intelligence*, 1215–1222.

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22(1-2):5–27.

Basin, D.; Bhatt, B. N.; and Traytel, D. 2017. Almost event-rate independent monitoring of Metric Temporal Logic. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 94–112.

Basin, D.; Krstić, S.; and Traytel, D. 2017. Almost event-rate independent monitoring of Metric Dynamic Logic. In Lahiri, S., and Reger, G., eds., *Proceedings of the 17th International Conference on Runtime Verification*, 85–102.

Brenton, C.; Faber, W.; and Batsakis, S. 2016. Answer set programming for qualitative spatio-temporal reasoning: Methods and experiments. In *Technical Communications of the 32nd International Conference on Logic Programming*, volume 52, 4:1–4:15.

de Leng, D., and Heintz, F. 2018. Partial-state progression for stream reasoning with metric temporal logic. In *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning*, 633–634.

Desai, A.; Dreossi, T.; and Seshia, S. A. 2017. Combining model checking and runtime verification for safe robotics. In *Proceedings of the 17th International Conference on Runtime Verification*, 172–189.

Dwyer, M. B.; Avrunin, G. S.; and Corbett, J. C. 1999. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international conference on Software engineering*, 411–420. ACM.

Emerson, E. A. 1990. Temporal and modal logic. In *Formal Models and Semantics*. Elsevier. 995–1072.

Koymans, R. 1990. Specifying real-time properties with Metric Temporal Logic. *Real-Time Systems* 2(4):255–299.

Kvarnström, J.; Heintz, F.; and Doherty, P. 2008. A temporal logic-based planning and execution monitoring system. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, 198–205.

Medhat, R.; Bonakdarpour, B.; Fischmeister, S.; and Joshi, Y. 2016. Accelerated runtime verification of LTL specifications with counting semantics. In *Proceedings of the 16th International Conference on Runtime Verification*, 251–267.

Nenzi, L.; Bortolussi, L.; Ciancia, V.; Loreti, M.; and Massink, M. 2015. Qualitative and quantitative monitoring of spatio-temporal properties. In *Proceedings of the 15th International Conference on Runtime Verification*, 21–37.

Randell, D.; Cui, Z.; and Cohn, A. 1992. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, 165–176.