On-the-fly Verification for Product-LTL

Deepak D'Souza* P. Madhusudan†

1 Introduction

The propositional linear temporal logic LTL [P 77] is a well studied tool for specifying and reasoning about distributed programs. An important application of LTL is in the area of model-checking, where given a (usually finite-state) program and a specification of the program written as an LTL formula, the problem is to check if all the runs of the program are models of the formula, i.e. satisfy the specification. We will work in a setting where the program is modelled as a synchronised product of a finite number of sequential programs. In [T 95] Thiagarajan defines a temporal logic Product-PTL (or Product-LTL), which can describe boolean combinations of LTL assertions about the individual sequential components. In this paper, we propose an *on-the-fly* technique for model-checking such programs against Product-LTL specifications.

When concurrent or distributed programs are modelled as a finite-state transition system, there is an inherent exponential blow-up of the state-space in terms of the number of processes. This so-called "state-space explosion" problem is the main practical difficulty in verifying large programs. Much of this state-space explosion arises because the model describes *all* possible interleaved runs of the original program. However, it has been noticed that to check certain *trace consistent* properties one can equate many interleaved runs of the program, and check the property along a single representative run of each such equivalence class. This is the basis for many partial-order based verification methods ([G 96], [V 90]) which lead to marked reduction in time and space in practical model-checking.

In our setting, where we work with networks of communicating automata which co-ordinate by synchronising on common actions, there is a natural notion of independence – two actions are independent if they are performed by disjoint sets of agents. Using standard notions of Mazurkiewicz traces, this leads to an equivalence relation on global runs of the network. Now, models of Product-LTL formulas are always trace-consistent [T 95], and this is what makes it an interesting logic to study from the model-checking point of view.

For practical verification, it is desirable to have what is known as an *on-the-fly* model-checker. To understand this, let us consider the following automata-theoretic approach to model-check Product-LTL specifications. Given a distributed program and a Product-LTL specification φ, we do the following:

1. View the program as a product automaton, say $\mathcal{M}$.
2. Build a product automaton for the negation of the specification, say $\mathcal{A}_{\neg\varphi}$, which describes all the ways in which a program might fail to meet the specification φ.
3. Construct the intersection of $\mathcal{M}$ and $\mathcal{A}_{\neg\varphi}$, say $\mathcal{C}$.
4. Check whether the language accepted by $\mathcal{C}$ is empty. The language is empty if and only if the program satisfies the specification.

Now, for doing Step 4 of the above procedure, there are good algorithms (like [CVWY 92]) which do a depth-first search on $\mathcal{C}$ and which can report non-emptiness after looking at just some initial portion of the automaton. Hence $\mathcal{C}$ can be incrementally constructed as and when required by the algorithm running Step 4. If we now have a procedure for building $\mathcal{A}_{\neg\varphi}$ in Step 2 in an incremental as-is-needed (or on-the-fly) manner, then we could construct $\mathcal{C}$ on the fly, and, in case Step 4 reports an intersection early, we can report that the program fails to meet the specification and stop. In such a case, we would save the space and time needed to construct the complete product automaton $\mathcal{A}_{\neg\varphi}$. Since for a given φ, $\mathcal{A}_{\neg\varphi}$ could be exponential

*SPIC Mathematical Institute, 92 G. N. Chetty Rd, Madras - 600017. email: deepak@smi.ernet.in
†Institute of Mathematical Sciences, Taramani, Madras - 600113. email: madhu@imsc.ernet.in

in $|\varphi|$, we could benefit considerably by such a method. Hence we will have an on-the-fly model-checker if we can do Step 2 on the fly.

In [VWS 83] the authors first showed how to construct a Büchi automaton which will accept the models of a given LTL formula. In [GPVW 95], the authors give a simple on-the-fly algorithm to generate automata for LTL formulas and in [M 96] we showed how to extend this to an action-based logic $LTL(\Sigma)$. In this paper, we show how we can use this algorithm to construct a product automaton for Product-LTL formulas, in an on-the-fly manner.

2 Preliminaries

For a set X, let X^ω denote the set of infinite sequences over X, i.e. the set of maps $f : \mathbb{N} \rightarrow X$. Let X^∞ denote the set $(X^* \cup X^\omega)$. For $x \in X^\infty$, let $pre(x)$ be the set of finite prefixes of x. For a string $\sigma \in X^\infty$, and $i \geq 0$ let σ_i denote the i-th suffix of σ. In case $i \geq |\sigma|$, σ_i is defined to be the empty word ϵ. Similarly, for a map $f : pre(\sigma) \rightarrow Y$, we define $f_i : pre(\sigma_i) \rightarrow Y$ given by $f_i(\tau) = f(\tau' \cdot \tau)$ where $\sigma = \tau' \cdot \sigma_i$.

Definition 2.1 *A Labelled Generalised Büchi Automaton (LGBA) over an alphabet Σ is a tuple $(Q, \longrightarrow, I, F, \mathcal{G}, l)$ where Q is a finite set of states, $\longrightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation, $I \subseteq Q$ is a set of start states, $F \subseteq Q$ is a set of finitary final states, $\mathcal{G} \subseteq 2^Q$ is a set of sets of infinitary final states, and l is the labelling function which associates with each state a set of labels from a finite domain D; i.e. $l : Q \rightarrow 2^D$.*

Let $A = (Q, \longrightarrow, I, F, G, l)$ be an $LGBA$ over Σ. Let $\sigma \in \Sigma^\infty$. A *run* ρ of A on σ is a map $\rho : pre(\sigma) \rightarrow Q$ such that (i) $\rho(\epsilon) \in I$, and (ii) $\forall \tau a \in pre(\sigma)\ \rho(\tau) \xrightarrow{a} \rho(\tau a)$.

For a run ρ on σ define $inf(\rho) = \{q \in Q \mid \text{for infinitely many } \tau \in pre(\sigma), \rho(\tau) = q\}$.

A run ρ of A on σ is *accepting* if either σ is finite and $\rho(\sigma) \in F$, or, σ is infinite and $inf(\rho) \cap G \neq \emptyset$, for each $G \in \mathcal{G}$. We say that a run ρ on σ is *labelled* by a map $\xi : pre(\sigma) \rightarrow D$ if for all $\tau \in pre(\sigma)\ \xi(\tau) \in l(\rho(\tau))$.

A pair (σ, ξ) is *accepted* by A if there exists an accepting run of A on σ labelled by ξ. We define the language accepted by A as $L(A) = \{(\sigma, \xi) \mid \sigma \in \Sigma^\infty,\ \xi : pre(\sigma) \rightarrow D, \text{ and } (\sigma, \xi) \text{ is accepted by } A\}$.

3 Action-based LTL

Let Σ be a finite set of action symbols and P a finite set of propositional symbols. We define an action based version of LTL called $LTL(\Sigma)$ whose models are sequences of propositional valuations along with a sequence of actions.

3.1 Syntax and semantics

The formulas of $LTL(\Sigma)$ are given by the following abstract syntax

$$\phi ::= p\ (p \in P) \mid \neg\phi \mid \phi \vee \phi \mid O\phi \mid \langle a \rangle \phi \mid \phi U \phi\,.$$

Let φ be an $LTL(\Sigma)$ formula. A model for φ is of the form (σ, ξ) where $\sigma \in \Sigma^\infty$, and $\xi : pre(\sigma) \rightarrow 2^P$. The satisfaction relation $(\sigma, \xi) \models_\Sigma \varphi$ is defined inductively:

$$\begin{array}{lll}
(\sigma,\xi) \models_\Sigma p & \text{iff} & p \in \xi(\epsilon) \\
(\sigma,\xi) \models_\Sigma \neg\alpha & \text{iff} & (\sigma,\xi) \not\models_\Sigma \alpha \\
(\sigma,\xi) \models_\Sigma \alpha \vee \beta & \text{iff} & (\sigma,\xi) \models_\Sigma \alpha \text{ or } (\sigma,\xi) \models_\Sigma \beta \\
(\sigma,\xi) \models_\Sigma O\alpha & \text{iff} & \sigma \neq \epsilon \text{ and } (\sigma_1,\xi_1) \models_\Sigma \alpha \\
(\sigma,\xi) \models_\Sigma \langle a \rangle \alpha & \text{iff} & \sigma = a \cdot \sigma_1 \text{ and } (\sigma_1,\xi_1) \models_\Sigma \alpha \\
(\sigma,\xi) \models_\Sigma \alpha U \beta & \text{iff} & \exists k \geq 0 \text{ s.t. } (\sigma_k,\xi_k) \models_\Sigma \beta \text{ and } \forall i : 0 \leq i < k,\ (\sigma_i,\xi_i) \models_\Sigma \alpha.
\end{array}$$

Define $L_\Sigma(\varphi) = \{(\sigma, \xi) \mid (\sigma, \xi) \models_\Sigma \varphi\}$.

3.2 Formula automaton

In [M 96] Madhusudan gives an algorithm to build a formula automaton (an $LGBA$) for an $LTL(\Sigma)$ formula, in an on-the-fly manner. We can think of the algorithm as comprising two procedures: INIT, which gives us the initial states of the automaton, and ADJ which gives us the set of states adjacent to a given state. States in the automaton are simply finite sets of $LTL(\Sigma)$ formulas.

ADJ is a non-deterministic procedure which given a state, returns a set of (action, state) pairs meant to be the adjacency list of the given state. Let us fix a function

$$adj : \mathcal{P}_{fin}(LTL(\Sigma)) \rightarrow 2^{\Sigma \times \mathcal{P}_{fin}(LTL(\Sigma))},$$

such that for each $X \subseteq_{fin} LTL(\Sigma)$ there is a run of ADJ(X) which returns the set $adj(X)$. Similarly, let $init$ be a function which given $\varphi \in LTL(\Sigma)$ returns a set of states returned by some run of INIT . The thesis then proves the following property: Let

$$A_\varphi = (Q, \longrightarrow, I, F, \mathcal{G}, l) \quad \text{where}$$

- Q is any finite subset of $\mathcal{P}_{fin}(LTL(\Sigma))$.
- $\forall p \in Q,\ p \xrightarrow{a} q$ iff $(a, q) \in adj(p)$.
- $I = init(\varphi)$.
- $F = \emptyset$.
- Let $\{\alpha_1 U \beta_1 , \ldots, \alpha_r U \beta_r \}$ be the set of until formulas in $CL(\varphi)$ – the subformula closure of φ. Define $\mathcal{G} = \{G_1, \ldots, G_r\}$ where $G_i = \{q \in Q \mid \beta_i \in q \text{ or } \alpha_i U \beta_i \notin q\}$.
- $l : Q \rightarrow 2^{2^P}$ given by $l(q) = \{X \subseteq P \mid (q \cap P) \subseteq X, \text{ and } (\overline{X} \cap q) = \emptyset\}$. ($\overline{X}$ denotes the set $\{\neg x \mid x \in X\}$.)

Then, A_φ accepts the set of infinite models of φ.

Further property: It is easy to adapt the algorithm for finite models, and extend the above result to say that, in addition, when

- $F = \{q \in Q \mid q$ has no formulas of the form $\langle a \rangle \alpha$ or $O\alpha\}$, and
- $\mathcal{G} = \{G_1, \ldots, G_s\}$ where $\alpha_1 U \beta_1 , \ldots, \alpha_s U \beta_s$ are the until formulas in $\bigcup Q$, and $G_i = \{q \in Q \mid \beta_i \in q \text{ or } \alpha_i U \beta_i \notin q\}$,

then A_φ accepts exactly $L_\Sigma(\varphi)$.

Note that the automaton A_φ is completely determined by the choice of Q. Further, a property of ADJ is that ADJ(X) always returns a subset of $CL(X)$. It follows that choosing Q to be any set containing $2^{CL(\varphi)}$ gives us a concrete realisation of A_φ.

We observe that A_φ is constructible on the fly. Essentially, this is because A_φ is an implicit graph, whose adjacency relation is given in terms of the ADJ procedure which produces the adjacent states in an incremental manner.

4 Product Automata and Product-LTL

For the rest of this paper, let us fix a distributed alphabet $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_k)$. Let $\Sigma = \bigcup_{i=1}^k \Sigma_i$. For $a \in \Sigma$, let $loc(a) = \{i \mid a \in \Sigma_i\}$. For $\sigma \in \Sigma^\infty$ let $\sigma \lfloor i$ denote the word we get by erasing from σ all occurrences of alphabets *not* in Σ_i.

Definition 4.1 *Let $D_1, \ldots, D_k$ be a partition on a finite set D. A product LGBA over $\widetilde{\Sigma}$ is a structure*

$$(\{A_i\}_{i=1}^k,\ \mathcal{I})$$

where each $A_i = (Q_i, \longrightarrow_i, F_i, \mathcal{G}_i, l_i)$ has the same structure as an LGBA over Σ_i, though without local start states. Further, the labelling function l_i associates with each state in Q_i a set of subsets of *D_i; i.e. $l_i : Q_i \rightarrow 2^{2^{D_i}}$. $\mathcal{I} \subseteq Q_1 \times \cdots \times Q_k$ is a set of global start states.*

Let $\mathcal{A} = (\{A_i\}_{i=1}^k,\ \mathcal{I})$ be a product $LGBA$. Let $A_i = (Q_i, \longrightarrow_i, F_i, \mathcal{G}_i, l_i)$. The language accepted by $\mathcal{A}$ is defined via a global transition relation defined as follows. Let $Q = Q_1 \times \cdots \times Q_k$. $\longrightarrow \subseteq Q \times \Sigma \times Q$ is given by $(p_1, \cdots, p_k) \xrightarrow{a} (q_1, \cdots, q_k)$ iff $\forall i \in loc(a),\ p_i \xrightarrow{a}_i q_i$ and $\forall i \notin loc(a),\ p_i = q_i$.

Let $\sigma \in \Sigma^\infty$. We say a map $\rho : pre(\sigma) \rightarrow Q$ is a run of $\mathcal{A}$ on σ if (i) $\rho(\epsilon) \in \mathcal{I}$, and (ii) for all $\tau \cdot a \in pre(\sigma)$, $\rho(\tau) \xrightarrow{a} \rho(\tau \cdot a)$.

Let ρ be a run of $\mathcal{A}$ on σ. We define $\rho \lfloor i$ as a map $\rho' : pre(\sigma \lfloor i) \rightarrow Q_i$ given by $\rho'(\tau) = \rho(\tau')[i]$, where $\tau' \in pre(\sigma)$ such that $\tau = \tau' \lfloor i$. Note that ρ' is well-defined due to the fact that $\rho(\tau')[i] = \rho(\tau'')[i]$ for all $\tau', \tau'' \in pre(\sigma)$ such that $\tau' \lfloor i = \tau'' \lfloor i$.

A run ρ of $\mathcal{A}$ on σ is *accepting* provided for each i, $\rho \lfloor i$ meets the acceptance condition of A_i; i.e. if $\sigma \lfloor i$ is finite then $(\rho \lfloor i)(\sigma \lfloor i) \in F_i$, and if $\sigma \lfloor i$ is infinite, then $inf(\rho \lfloor i) \cap G \neq \emptyset,\ \forall G \in \mathcal{G}_i$.

We say that a run ρ of $\mathcal{A}$ on σ is *labelled* by a map $\xi : pre(\sigma) \rightarrow 2^D$, if $\exists \xi^1, \ldots, \xi^k$ such that each ξ^i is a labelling of the run $\rho \lfloor i$ of A_i on $\sigma \lfloor i$, and for each i $\xi(\tau) \cap D_i = \xi^i(\tau \lfloor i)$.

We say that (σ, ξ) is *accepted* by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on σ labelled by ξ. The language accepted by $\mathcal{A}$ is defined as $L(\mathcal{A}) = \{(\sigma, x) \mid (\sigma, x) \text{ is accepted by } \mathcal{A}\}$.

Lemma 4.1 *Let $\mathcal{A}_1, \mathcal{A}_2$ be product LGBA's such that $\mathcal{A}_1 = (\{A_i\}_{i=1}^k,\ \mathcal{I}_1)$, $\mathcal{A}_2 = (\{A_i\}_{i=1}^k,\ \mathcal{I}_2)$. Then $L(\mathcal{A}_3) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ where $\mathcal{A}_3 = (\{A_i\}_{i=1}^k,\ \mathcal{I}_1 \cup \mathcal{I}_2)$.*

Definition 4.2 *For languages $L_1, \ldots, L_k$ such that for each i $L_i \subseteq \Sigma_i^\infty$, define*

$$L_1 \otimes \cdots \otimes L_k = \{\sigma \in \Sigma^\infty \mid \forall i\ \sigma \lfloor i \in L_i\}.$$

Lemma 4.2 *Let $A_i = (Q_i, I_i, \longrightarrow_i, F_i, G_i, l_i)$ $(i = 1, \ldots, k)$ be LGBA's over Σ_i's respectively. Then the product LGBA $\mathcal{A} = (\{A_i\}_{i=1}^k,\ I_1 \times \cdots \times I_k)$ accepts the language $L(A_1) \otimes \cdots \otimes L(A_k)$.*

4.1 Product-LTL: Syntax and semantics

Fix a set of propositions P, and a partition $P_1, \ldots, P_k$ of P. We use the following syntax for product-LTL (denoted $LTL^\otimes$:)

$$\phi ::= \psi_i\ (\psi_i \in LTL(\Sigma_i)) \mid \phi \vee \phi \mid \phi \wedge \phi,$$

where the set of propositional symbols for $LTL(\Sigma_i)$ is taken to be P_i.

The models for $LTL^\otimes$ formulas are pairs of the form (σ, ξ) where $\sigma \in \Sigma^\infty$, $\xi : pre(\sigma) \rightarrow 2^P$, satisfying the following "validity" condition: $\tau \cdot a \in pre(\sigma)$ and $a \notin \Sigma_i \Rightarrow \xi(\tau \cdot a) \cap P_i = \xi(\tau) \cap P_i$.

Given a "valid" model (σ, ξ), let $(\sigma, \xi) \lfloor i$ denote the pair $(\sigma \lfloor i, \xi')$ where $\xi' : pre(\sigma \lfloor i) \rightarrow 2^P$ given by $\xi'(\tau) = \xi(\tau')$, where $\tau' \in pre(\sigma)$ s.t. $\tau' \lfloor i = \tau$. Note that ξ' is well-defined due to the validity condition on (σ, ξ).

The satisfaction relation $(\sigma, \xi) \models_\otimes \varphi$ for $LTL^\otimes$ formulas is defined inductively as follows:

$$\begin{array}{lll}
(\sigma, \xi) \models_\otimes \psi_i \text{ (for } \psi_i \in LTL(\Sigma_i)) & \text{iff} & (\sigma, \xi) \lfloor i \models_{\Sigma_i} \psi_i \\
(\sigma, \xi) \models_\otimes \alpha \vee \beta & \text{iff} & (\sigma, \xi) \models_\otimes \alpha \text{ or } (\sigma, \xi) \models_\otimes \beta \\
(\sigma, \xi) \models_\otimes \alpha \wedge \beta & \text{iff} & (\sigma, \xi) \models_\otimes \alpha \text{ and } (\sigma, \xi) \models_\otimes \beta
\end{array}$$

For $\varphi \in LTL^\otimes$ define $L_\otimes(\varphi) = \{(\sigma, \xi) \mid (\sigma, \xi) \models_\otimes \varphi\}$.

Lemma 4.3 *Let $\alpha_1, \ldots, \alpha_k$ be formulas in $LTL(\Sigma_1), \ldots, LTL(\Sigma_k)$ respectively. Then $L_\otimes(\alpha_1 \wedge \cdots \wedge \alpha_k) = L_{\Sigma_1}(\alpha_1) \otimes \cdots \otimes L_{\Sigma_k}(\alpha_k)$.*

4.2 Formula automaton

Given $\varphi \in LTL^\otimes$ we would like to define a product automaton $\mathcal{A}_\varphi$ such that $L(\mathcal{A}_\varphi) = L_\otimes(\varphi)$. We proceed in the following steps:

1. Write φ in the form $\alpha \equiv \bigvee_{j=1}^m (\bigwedge_{i=1}^k \alpha_i^j)$, where $\forall j,\ \alpha_i^j \in LTL(\Sigma_i)$. It follows that $L_\otimes(\varphi) = \bigcup_{j=1}^m\ L_\otimes(\bigwedge_{i=1}^k \alpha_i^j)$.

2. We give $A_1, \ldots, A_k$, and $\mathcal{I}_1, \ldots, \mathcal{I}_m$ such that for each $j \in \{1, \ldots, m\}$ the product $LGBA$ $\mathcal{A}^j = (\{A_i\}_{i=1}^k,\ \mathcal{I}^j)$ accepts $L_{\otimes}(\bigwedge_{i=1}^k \alpha_i^j)$. This can be done by taking the formula automaton $A_{\alpha_i^j}$ for α_i^j defined in section 3, with the set of states Q taken to be $\bigcup_{j=1}^m 2^{CL(\alpha_i^j)}$. Clearly, for a fixed i, this automaton is the same for each $\alpha_i^1, \ldots, \alpha_i^m$, except for the set of initial states I_j^i. Let us write these automata as $A_{\alpha_i^j} = (A_i, I_i^j)$. Let $I^j = I_1^j \times \cdots \times I_k^j$. It now follows using lemmas 4.2 and 4.3 that $\mathcal{A}^j = (\{A_i\}_{i=1}^k,\ \mathcal{I}^j)$ accepts $L_{\otimes}(\bigwedge_{i=1}^k \alpha_i^j)$.

3. From lemma 4.1 it follows that the product automaton $\mathcal{A}_{\varphi} = (\{A_j\}_{j=1}^k,\ \bigcup_{i=1}^m \mathcal{I}^i)$ accepts $L_{\otimes}(\varphi)$.

We note that the product automaton $\mathcal{A}_{\varphi}$ is constructible on the fly, since we have an on-the-fly construction (from section 3) for each of the component automata A_i.

Acknowledgement We thank P. S. Thiagarajan for many useful discussions.

References

[CVWY 92] C. Courcoubetis, M. Vardi, P. Wolper, M. Yanakakis, *Memory efficient algorithms for the verification of temporal logic properties.* Formal Methods in System Design, 1992, 1, 275- 288.

[GPVW 95] R. Gerth, D. Peled, M. Vardi, P. Wolper. *Simple on-the-fly verification for Linear Temporal Logic.* Protocol Specification, Testing, and Verification, 1995.

[G 96] Patrice Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems.* LNCS 1032, Springer, 1996.

[M 96] P. Madhusudan. *An On-the-fly Algorithm for Linear Time Temporal Logic*, M. Sc. thesis, Anna University, 1996.

[P 77] A. Pnueli. *The temporal logic of programs.* Proceedings of the . 18th IEEE Symposium on Foundations of Computer Science, pages 46-57, 1977.

[T 95] P. S. Thiagarajan, *A Trace Consistent Subset of PTL* in LNCS 962, CONCUR '95: Concurrency Theory. Insup Lee, Scott A Smolka (Eds.), 6th International Conference, Philadelphia, USA.

[V 90] Antti Valmari, *Stubborn sets for Reduced State space generation.* LNCS 483, 1990, pages 491-515.

[VWS 83] M. Y. Vardi, P. Wolper, A. P. Sistla. *Reasoning about infinite computation paths.* Proceedings of 24th IEEE Symposium on Foundation of Computer Science, Tuscan, 1983, pages 185-194.