Probabilistic Model Checking of DTMC Models of User Activity Patterns

Oana Andrei[1], Muffy Calder[1], Matthew Higgs[1], and Mark Girolami[2]

[1] School of Computing Science, University of Glasgow, G12 8RZ, UK
[2] Department of Statistics, University of Warwick, CV4 7AL, UK

Abstract. Software developers cannot always anticipate how users will actually use their software as it may vary from user to user, and even from use to use for an individual user. In order to address questions raised by system developers and evaluators about software usage, we define new probabilistic models that characterise user behaviour, based on activity patterns inferred from actual logged user traces. We encode these new models in a probabilistic model checker and use probabilistic temporal logics to gain insight into software usage. We motivate and illustrate our approach by application to the logged user traces of an iOS app.

1 Introduction

Software developers cannot always anticipate how users will *actually* use their software, which is sometimes surprising and varies from user to user, and even from use to use, for an individual user. We propose that temporal logic reasoning over formal, probabilistic models of actual logged user traces can aid software developers, evaluators, and users by: *providing insights* into application usage, including differences and similarities between different users, *predicting* user behaviours, and *recommending* future application development.

Our approach is based on systematic and automated logging and reasoning about users of applications. While this paper is focused on mobile applications (apps), much of our work applies to any software system. A logged user trace is a chronological sequence of in-app actions representing how the user explores the app. From logged user traces of a population of users we infer *activity patterns*, represented each by a Discrete-Time Markov Chain (DTMC), and for each user we infer a user *strategy* over the activity patterns. For each user we deduce a meta model based on the set of all activity patterns inferred from the population of users and the user strategy, and we call it the *user metamodel*. We reason about the user metamodel using probabilistic temporal logic properties to express hypotheses about user behaviours and relationships within and between the activity patterns, and to formulate app-specific questions posed by developers and evaluators.

We motivate and illustrate our approach by application to the mobile, multiplayer game Hungry Yoshi [1], which was deployed in 2009 for iPhone devices and has involved thousands of users worldwide. We collaborate with the Hungry Yoshi developers on several mobile apps and we have access to all logged user data. We have chosen the Hungry Yoshi app because its functionality is relatively simple, yet adequate to illustrate how formal analysis can inform app evaluation and development.

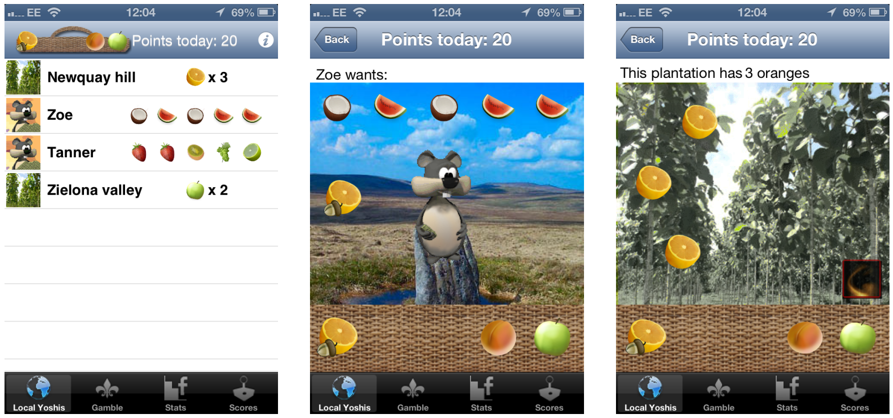
The main contributions of the paper are:

– a formal and systematic approach to formal user activity analysis in a probabilistic setting,
– inference of user activity patterns represented as DTMCs,
– definition of the DTMC user metamodel and guidelines for inferring user metamodels from logged user data,
– encoding of the user metamodel in the PRISM model checker and temporal logic properties defined over both states and activity patterns as atomic propositions,
– illustration with a case study of a deployed app with thousands of users and analysis results that reveal insights into real-life app usage.

The paper is organised as follows. In the next section we give an overview of the Hungry Yoshi app, which we use to motivate and illustrate our work. We list some example questions that have been posed by the Hungry Yoshi developers and evaluators; while these are specific to the Hungry Yoshi app, they are also indicative questions for any app. In Sect. 3 we give background technical definitions concerning DTMCs and probabilistic temporal logics. In Sect. 4 we define inference of user activity patterns, giving a small example as illustration and some example results for Hungry Yoshi. In Sect. 5 we define the user metamodel, we illustrate it for Hungry Yoshi and we give an encoding for the PRISM model checker. In Sect. 6 we consider how to encode some of the questions posed in Sect. 2.1 in probabilistic temporal logic, and give some results for an example Hungry Yoshi user metamodel. In Sect. 7 we reflect upon the results obtained for Hungry Yoshi and some further issues raised by our approach. In Sect. 8 we review related work and we conclude in Sect. 9.

2 Running Example: Hungry Yoshi

The mobile, multiplayer game Hungry Yoshi [1] is based on picking pieces of fruit and feeding them to creatures called *yoshis*. Players' mobile devices regularly scan the available WiFi access points and display a password-protected network as a *yoshi* and a non-protected network as a *fruit plantation*. Plantations grow particular types of *fruit* (possibly from *seeds*) and yoshis ask players for particular types of fruit. Players score points if they pick the fruit from the correct plantations, store them in a basket, and give them to yoshis as requested. There is further functionality, but here we concentrate on the key user-initiated events, or *button taps*, which are: *see a yoshi*, *see a plantation*, *pick fruit* and *feed a yoshi*. The external environment (as scanned by device), combined with user choice, determines when yoshis and plantations can be observed. The game was instrumented by the developers using the SGLog data logging infrastructure [2], which streams logs of specific user system operations back to servers on



(a) Main menu (b) Yoshi Zoe (c) Newquay plantation

Fig. 1: Hungry Yoshi screenshots: two plantations (Newquay hill and Zielona valley) and two yoshis (Zoe and Taner) are observed. The main menu shows the available plantations and yoshis with their respective content and required types of fruit. The current basket contains one orange seed, one apricot and one apple.

the developing site as user traces. The developers specify directly in the source code what method calls or contextual information are to be logged by SGLog. A sample of screenshots from the game is shown in Fig. 1.

2.1 Example Questions from Developers and Evaluators

Key to our formal analysis is suitable hypotheses, or *questions*, about user behaviour. For Hungry Yoshi, we interviewed the developers and evaluators of the game to obtain questions that would provide useful insights for them. Interestingly most of their hypotheses were *app-specific*, and so we focus on these here, and then indicate how each could be generalised. We note that to date, tools available to the developers and evaluators for analysis include only SQL and iPython stats scripts.

1. When a yoshi has been fed n pieces of fruit (which results in extra points when done without interruption for n equal 5), did the user interleave *pick fruit* and *feed a yoshi* n times or did the user perform n *pick fruit* events followed by n *feed a yoshi* events? And afterwards, did he/she continue with that pick-feed strategy or change to another one? Which strategy is more likely in which activity pattern? More generally, when there are several ways to reach a goal state, does the user always take a particular route and is this dependent on the activity pattern?
2. If a user in one activity pattern does not *feed a yoshi* within n button taps, but then changes to another activity pattern, is the user then likely to *feed a yoshi* within m button taps? More generally, which events cause a change of activity pattern, and which events follow that change of activity pattern?

3. What kind of user tries to *pick fruit* 6 times in a row (a basket can only hold 5 pieces of fruit)? More generally, in which activity pattern is a user more likely to perform an inappropriate event?
4. If a user reads the instructions once, then does that user reach a goal state in fewer steps than a user who does not read the instructions at all? (Thus indicating the instructions are of some utility.) More generally, if a user performs a given event, then it is more likely that he/she will perform another given event, within n button taps, than users that have not performed the first event? Is this affected by the activity pattern?

3 Technical Background

We assume familiarity with Discrete-Time Markov Chains, probabilistic logics PCTL and PCTL*, and model checking [3, 4]; basic definitions are below.

A *discrete-time Markov chain* (DTMC) is a tuple $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ where: S is a set of states; $\bar{s} \in S$ is the initial state; $\mathbf{P} : S \times S \to [0, 1]$ is the transition probability function (or matrix) such that for all states $s \in S$ we have $\sum_{s' \in S} \mathbf{P}(s, s') = 1$; and $L : S \to 2^{AP}$ is a labelling function associating to each state s in S a set of valid atomic propositions from a set AP. A *path* (or execution) of a DTMC is a non-empty sequence $s_0 s_1 \ldots$ where $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. A path can be finite or infinite. Let $\mathit{Path}^{\mathcal{D}}(s)$ denote the set of all infinite paths of $\mathcal{D}$ starting in state s.

Probabilistic Computation Tree Logic (PCTL) [3] and its extension PCTL* allow one to express a probability measure of the satisfaction of a temporal property. Their syntax is the following:

$$\begin{array}{rl} \textit{State formulae} & \Phi ::= \mathit{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathbf{P}_{\bowtie p}[\Psi] \\ \textit{PCTL Path formulae} & \Psi ::= \mathsf{X}\,\Phi \mid \Phi\,\mathsf{U}^{\leq n}\,\Phi \\ \textit{PCTL* Path formulae} & \Psi ::= \Phi \mid \Psi \wedge \Psi \mid \neg\Psi \mid \mathsf{X}\,\Psi \mid \Psi\mathsf{U}^{\leq n}\,\Psi \end{array}$$

where a ranges over a set of atomic propositions AP, $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0, 1]$, and $n \in \mathbb{N} \cup \{\infty\}$.

A state s in a DTMC $\mathcal{D}$ satisfies an atomic proposition a if $a \in L(s)$. A state s satisfies a state formula $\mathbf{P}_{\bowtie p}[\Psi]$, written $s \models \mathbf{P}_{\bowtie p}[\Psi]$, if the probability of taking a path starting from s and satisfying Ψ meets the bound $\bowtie p$, i.e., $\Pr_s\{\omega \in \mathit{Path}^{\mathcal{D}}(s) \mid \omega \models \Psi\} \bowtie p$, where $\Pr_s$ is the probability measure defined over paths from state s. The path formula $\mathsf{X}\,\Phi$ is true on a path starting with s if Φ is satisfied in the state following s; $\Phi_1\,\mathsf{U}^{\leq n}\,\Phi_2$ is true on a path if Φ_2 holds in the state at some time step $i \leq n$ and at all preceding states Φ_1 holds. This is a minimal set of operators, the propositional operators *false*, disjunction and implication can be derived using basic logical equivalences and a common derived path operators is the *eventually* operator F where $\mathsf{F}^{\leq n}\,\Phi \equiv \mathit{true}\,\mathsf{U}^{\leq n}\,\Phi$. If $n = \infty$ then superscripts omitted. We assume the following two additional notations. Let φ denote the state formulae from the propositional logic fragment of PCTL, i.e., $\varphi ::= \mathit{true} \mid a \mid \neg\varphi \mid \varphi \wedge \varphi$, where $a \in AP$. Let $\mathcal{D}_{|\varphi}$ denote the DTMC obtained from $\mathcal{D}$ by restricting the set of states to those satisfying φ.

Many of the properties we will examine require PCTL*, because we want to examine sequences of events: this requires multiple occurrences of a bounded until operator. This is not fully implemented in the current version of PRISM (only a single bounded U is permitted[3]) and so we combine probabilities obtained from PRISM *filtered properties* to achieve the same result. Filtered probabilities check for properties that hold *from sets of states* satisfying given propositions. For a DTMC $\mathcal{D}$, we define the filtered probability of taking all paths that start from any state satisfying φ and satisfy (PCTL) ψ by:

$$Prob^{\mathcal{D}}_{filter(\varphi)}(\psi) \stackrel{def}{=} filter_{s\in\mathcal{D},s\models\varphi}\mathrm{Pr}_s\{\omega \in Path^{\mathcal{D}}(s) \mid \omega \models \psi\}$$

where *filter* is an operator on the probabilities of ψ for all the states satisfying φ. In the examples illustrated in this paper we always use *state* as the filter operator since φ uniquely identifies a state.

4 Inferring User Activity Patterns

The role of inference is to construct a representation of the data that is amenable to checking probabilistic temporal logic properties. Developers want to be able to select a user and explore that user's model. While this could be achieved by constructing an independent DTMC for each user, there is much to be gained from sharing information between users. One way to do this is to construct a set of *user classes* based on attribute information, and to learn a DTMC for each class. This is the approach taken in [5] for users interacting with web applications, and is a natural way to aggregate information over users and to condition user-models on attribute values. One issue with this approach is that it assumes within-class use to be homogeneous. For example, all users in the same city using the same browser are modelled using the same DTMC.

In this work we take a different approach to inference. We have found the common representations of context - such as location, operating system, or time of day - to be poor predictors of mobile application use. For this reason we construct user models based on the log information alone, without any ad-hoc specification of user classes. By *letting the data speak for itself*, we hope to uncover interesting activity patterns and meaningful representations of users.

4.1 Statistical Model and Inference

We extend the standard DTMC model by introducing a *strategy* for each user over activity patterns. More formally, we assume there exists a finite K number of activity patterns, each modelled by a DTMC denoted $\alpha_k = (S, \iota^{init}, \mathbf{P}_k, L)$, for $k = 1, \ldots, K$. Note only the transition probability $\mathbf{P}_k$ varies over the set of DTMCs, all the other parameters are the same. For some enumeration of users $m = 1, \ldots, M$, we represent a user's strategy by a vector θ_m such that $\theta_m(k)$ denotes the probability that user m transitions according to α_k.

[3] Because currently the LTL-to-automata translator that PRISM uses does not support bounded properties.

Statistical model. The data for each user is assumed to be generated in the following way. We assume all users to be independent and all DTMCs to be available to all users at all points in time. A user chooses an initial state according to ι^{init}. When in state $s \in S$, user m selects the kth DTMC with probability $\theta_m(k)$. If the user chooses the kth DTMC, then they transition from state s to $s' \in S$ with probability $\mathbf{P}_k(s, s')$. This simple description specifies all the probabilistic dependencies required to compute the likelihood of the data given the parameters of the model. While it is possible to extend the model so θ is state-dependent, this will require us to either lose the distributed representation of the user population, or to increase the number of parameters in a way that leads to a high combinatorial degree of complexity.

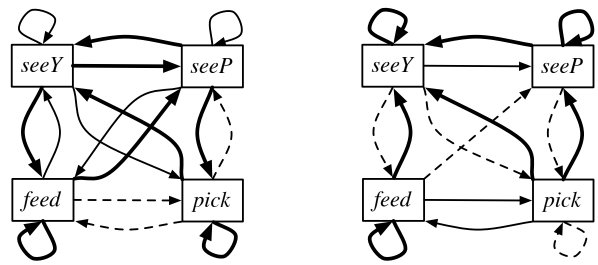
Inference. Inference is performed by maximising the log-likelihood of the data over the parameters of the model. This cannot be done analytically and we use a numerical method: the expectation-maximisation (EM) algorithm of [6]. For $K > 1$, the log-likelihood has multiple maxima and we restart the algorithm multiple times and select the output parameters with the highest log-likelihood over all runs. For the data considered here, restarting the algorithm 1000 times was sufficient to reproduce the same output parameters.

4.2 Example Activity Patterns from Hungry Yoshi

In Fig. 2 we give the activity patterns inferred from a dataset of user traces for 164 users randomly selected from the user population, for $K = 2$. A more detailed overview is given in the work-in-progress paper [7]. For brevity, we do not include the exact values of $\mathbf{P}_1$ and $\mathbf{P}_2$, but thicker arcs correspond to transition probabilities greater than 0.1, thinner ones to transition probabilities in $[0.01, 0.1]$, and dashed ones to transition probabilities smaller than 10^{-12}. Intuitively, we can see that given the game is essentially about seeing yoshis and feeding them, α_1 looks like a better way for playing the game. For example in α_2 it is quite rare to reach *feed* from *seeY* and *seeP*, and also rare to move from *seeP* to *pick*. Hungry Yoshi is a simple app with only two distinctive activity patterns, in a more complex setting we might not be able to have any intuition about the activity patterns.

5 User Metamodel

We define the formal model of the behaviour of a user m with respect to the population of users, which we call the *user metamodel* (UMM). The UMM for user m is a DTMC obtained by "flattening" the transition model over states *and* strategies. The resulting DTMC describes how the user transitions between composite states of the form (s, k) where s is an observable state and k indicates the activity pattern at that time. The UMM can be defined formally in the following way.



α1 α2

Fig. 2: Two user activity patterns α_1 and α_2 inferred from Hungry Yoshi usage.

Definition 5.1 (User Metamodel). *Given K activity patterns $\alpha_1, \dots, \alpha_K$ and θ_m the strategy of user m for choosing activity patterns, the* user metamodel *for m is a DTMC $\mathcal{M} = (S_{\mathcal{M}}, \iota^{init}_{\mathcal{M}}, \mathbf{P}_{\mathcal{M}}, \mathcal{L}_{\mathcal{M}})$ where:*

- $S_{\mathcal{M}} = S \times \{1, \dots, K\}$,
- $\iota^{init}_{\mathcal{M}}(s,k) = \theta_m(k) \cdot \iota^{init}(s)$,
- $\mathbf{P}_{\mathcal{M}}((s,k),(s',k')) = \theta_m(k') \cdot \mathbf{P}_{k'}(s,s')$,
- $\mathcal{L}_{\mathcal{M}}(s,k) = \mathcal{L}(s) \cup \{\alpha = k\}$.

We label each state (s,k) with the atomic proposition $\alpha = k$ to denote that the state belongs to the activity pattern α_k.

5.1 Example UMM from Hungry Yoshi

An intuitive graphical description of the UMM for the Hungry Yoshi game for $K = 2$ is illustrated in Fig. 3. For example, $\theta_m(1)$ is the probability that user m continues with activity pattern α_1, i.e. takes a transition between states in α_1. The probability that the user changes the activity pattern and makes a transition according to α_2 is proportional to $\theta_m(2)$. Figure 3 is not a direct representation of the transition probability matrix of the UMM DTMC, but it illustrates how that matrix is derived from the matrices of the individual user activity patterns. Note that the activity patterns have the same sets of states. For instance, in the Hungry Yoshi example, consider we are in state *seeY* with α_1; we can move to state *feed* following the same pattern α_1 with the probability $\theta_m(1) \cdot P_1(seeY, feed)$, *or* we can change the activity pattern and move to state *feed* following α_2 with the probability $\theta_m(2) \cdot P_2(seeY, feed)$.

5.2 Encoding a UMM in PRISM

We use the probabilistic model checker PRISM [8]. We assume some familiarity with the modelling language (based on the language of reactive modules), which includes global variables, modules with local variables, labelled-commands corresponding to transitions and multiway synchronisation of modules. Below we

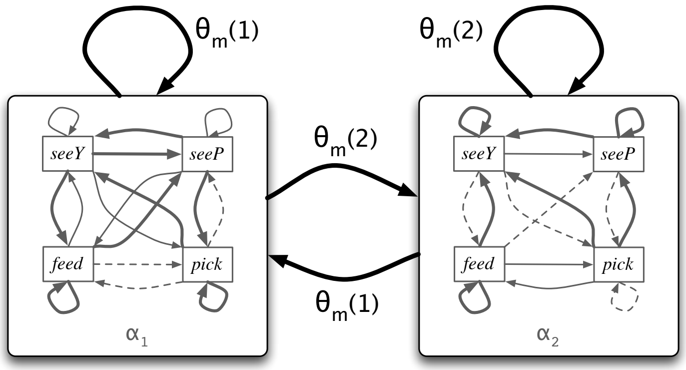


Fig. 3: An intuitive view of computing the transition probability matrix of the user metamodel for the Hungry Yoshi app.

illustrate the PRISM encoding of the UMM for user m, where K is the number of activity patterns, n is the number of states in each activity pattern α_k.

$$
\begin{array}{l}
\textbf{module}\ \mathit{UserMetamodel\_m} \\
\quad s : [0\,..\,n]\ \textbf{init}\ 0; \\
\quad k : [0\,..\,K]\ \textbf{init}\ 0; \\
\\
\quad [\,]\ (s = 0) \longrightarrow \theta_m(1) * \iota^{init}(1) : (s' = 1)\,\&\,(k' = 1) + \ldots + \\
\qquad\qquad\qquad\quad \theta_m(K) * \iota^{init}(n) : (s' = n)\,\&\,(k' = K); \\
\quad [\,]\ (s = 1) \longrightarrow \theta_m(1) * \mathbf{P}_1(1,1) : (s' = 1)\,\&\,(k' = 1) + \ldots + \\
\qquad\qquad\qquad\quad \theta_m(K) * \mathbf{P}_K(1,n) : (s' = n)\,\&\,(k' = K); \\
\quad \vdots \\
\quad [\,]\ (s = n) \longrightarrow \theta_m(1) * \mathbf{P}_1(n,1) : (s' = 1)\,\&\,(k' = 1) + \ldots + \\
\qquad\qquad\qquad\quad \theta_m(K) * \mathbf{P}_K(n,n) : (s' = n)\,\&\,(k' = K); \\
\textbf{endmodule}
\end{array}
$$

The representation is straightforward, consisting of one module with $(n + 1)$ commands for all n states of any activity pattern and for one initial state. The initial state $(s = 0, k = 0)$ is a dummy that encodes the global initial distribution ι^{init} for the user activity patterns. All activity patterns have the same set of states and we enumerate them from 1 to n; we can label them conveniently with atomic propositions. For instance, in a Hungry Yoshi UMM the states $(0, k)$ to $(4, k)$ are labelled by the atomic proposition *init*, *seeY*, *feed*, *seeP*, *pick* respectively. For each state $(s, \cdot)$, with $s > 0$, we have a command defining all possible $n \cdot K$ probabilistic transitions. $\mathbf{P}_k(i, j)$ is the transition probability from state i to state j in α_k, and $\theta_m(k)$ is the probability of user m to choose the activity patterns α_k, for all $i, j \in \{1, \ldots, n\}$, $k \in \{1, \ldots, K\}$. If the probability of an update is null, then the corresponding transition does not take place.

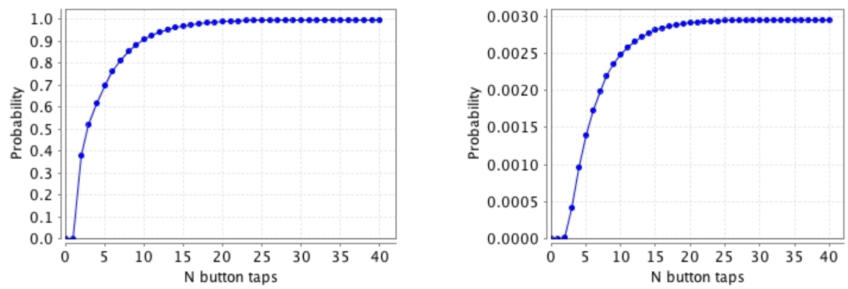


Fig. 4: Question 1: the probability of feeding a yoshi for the first time within N button taps for the activity pattern α_1 on the left and for α_2 on the right.

6 Analysing the Hungry Yoshi UMM

In this section, we give some example analysis for a UMM. Namely, we encode and evaluate quantitatively several example questions from Sect. 2.1 for the UMM with the user strategy for transitioning between activity patterns defined by $\theta = (0.7, 0.3)$. The PRISM models and property files are freely available[4].

Recall that to score highly, a user must feed one or more yoshis (the appropriate fruit) often. An informal inspection of α_1 and α_2 indicates that α_2 is a less effective strategy for playing the game, since paths from *seeP* and *seeY* to *feed* are unlikely. Now, by formal inspection of the UMM (encoded in PRISM), we can investigate this hypothesis more rigorously. We consider properties that are parametrised by a number of button taps (e.g. N, $N1$, $N2$) and by activity pattern (e.g. α_1, α_2), so we use the PRISM experiment facility that allows us to evaluate and then plot graphically results for a range of formulae.

Question 1. How many button taps N does it take to feed a yoshi for the first time? We encode this by the probabilistic until formula:

$$p_1(i) = Prob^{\mathcal{M}}_{init}((\neg feed)\ \mathsf{U}^{\leq N}\ ((\alpha = i) \wedge feed))$$

and equivalently in PRISM: `P=?[(!"feed") U<=N (alpha=i)&("feed")]`.

For activity pattern α_1, Figure 4 shows that within 2 button taps the probability increases rapidly, and after 5 button taps the probability is more than 70%. Contrast this with the results for α_2: the probability increases rapidly after 3 button taps but soon it reaches the upper bound of 0.003. Comparing the two results, α_1 is clearly more effective.

Now we consider more complex questions concerning sequences of feeding and picking; recall that a basket can hold at most 5 fruits and extra points are gained by feeding a yoshi its required 5 fruits without any other interruption. In

[4] Available from `http://dcs.gla.ac.uk/~oandrei/yoshi` .

Question 2 we consider feeding a full basket to a yoshi, without any interruptions; in Question 3 we consider picking a full basket, without being interrupted by a *feed*, followed by feeding the full basket to a yoshi, which is again defined by five consecutive *feeds*, without any interruptions. Note that when considering feeding the full basket to a yoshi, we exclude all interruptions, i.e. any interleavings with *pick*, *seeY*, and *seeP*.

Question 2. What is the probability of feeding the same yoshi a full basket? We calculate the probability of reaching the state *feed* within N button taps and then visiting it (with the same activity pattern $i \in \{1,2\}$) for another four times without visiting any other state:

$$p_2(i) = Prob^{\mathcal{M}}_{init}(\mathsf{F}^{\leq N}(\alpha = i \wedge feed)) \cdot (Prob^{\mathcal{M}_{|\alpha=i}}_{feed}(\mathsf{X}\, feed))^4$$

We calculate this probability in PRISM using the property:

```
P=?[F<=N((alpha=i)&"feed")] *
  pow(filter(state,P=?[X(alpha=i&"feed")],(alpha=i&"feed")),4)
```

The results are shown in Fig. 5 for both activity patterns and a range of number of button taps. While the results for α_1 (converging to 0.018) are higher than for α_2 (effectively 0); they are both small. There could be several causes for this. For example, players are only made aware of the possibility of extra points at the end of the instructions pages, or available fruit depends on the external environment. If designers/evaluators want this investigated further, then we would require to record and extract more detail from the logs, for example to log numbers of available WiFi access points and scrolls through instruction pages.

Question 3. What is the probability of filling up a basket of fruit without feeding a yoshi, and only after the basket is full feeding the same yoshi the whole basket? We calculate the probability of reaching the state *feed* only after visiting the state *pick* five times (without feeding) and then visiting the state *feed* four more times without visiting any other state, for each activity pattern $i \in \{1,2\}$:

$$\begin{aligned} p_3(i) = {} & Prob^{\mathcal{M}}_{init}[(\neg pick)\,\mathsf{U}^{\leq N}((\alpha = i) \wedge pick)] \cdot \\ & (Prob^{\mathcal{M}_{|\alpha=i}}_{pick}[\mathsf{X}((\neg feed \wedge \neg pick)\,\mathsf{U}\,(pick))])^4 \cdot \\ & Prob^{\mathcal{M}_{|\alpha=i}}_{pick}[(\neg feed)\,\mathsf{U}\, feed] \cdot (Prob^{\mathcal{M}_{|\alpha=i}}_{feed}[\mathsf{X}\, feed])^4 \end{aligned}$$

The corresponding PRISM property is:

```
P=?[!("pick") U<=N ((alpha=i)&"pick")] * pow(filter(state,
   P=?[X ((alpha=i)&(!"feed")&(!"pick") U ((alpha=i)&"pick"))],
   ((alpha=i)&"pick")),4) * filter(state,
   P=?[(alpha=i)&(!"feed")U((alpha=i)&"feed")],((alpha)=i&"pick")) *
   pow(filter(state,P=?[X((alpha=i)&"feed")],((alpha=i)&"feed")),4)
```

The results are presented in Fig. 6. Again, while the probabilities are low (presumably for the reasons outlined above for Question 2) the user that picks

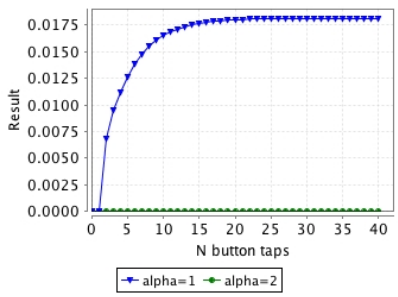


Fig. 5: Question 2: the probability of feeding one yoshi the whole fruit basket without interruptions.

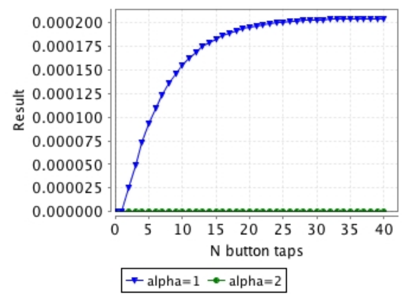


Fig. 6: Question 3: the probability of picking five pieces of fruit and then feeding one yoshi the whole basket.

a full basket and feeds it to a yoshi by following activity pattern α_1 does it with around 0.00019 probability within 20 steps into the game, whereas if they follow α_2 from the beginning, they almost never empty the basket. So again, α_1 proves to be more effective.

Now we turn our attention to a question that involves a *change* of activity pattern, i.e. a change in the playing strategy.

Question 4. What is the probability of starting with an activity pattern and not feeding a yoshi within N button taps, then changing to the other activity pattern and eventually first feeding a yoshi within N_2 button taps? We compute this probability as follows, where $\mathcal{L}_0 = \{\mathit{feed}, \mathit{pick}, \mathit{see}Y, \mathit{see}P\}$:

$$\begin{aligned} p_4(i) = \textstyle\sum_{\ell\in\mathcal{L}_0} & \mathit{Prob}^{\mathcal{M}}_{\mathit{init}}((\neg(\alpha=i)\wedge\neg\mathit{feed})\,\mathsf{U}^{\leq N}((\alpha=i)\wedge\ell))\cdot \\ & \mathit{Prob}^{\mathcal{M}_{|\alpha=i}}_{\ell}((\neg\mathit{feed})\,\mathsf{U}^{\leq N_2}\mathit{feed}) \end{aligned}$$

The corresponding PRISM property is:

```
P=?[(!(alpha=i)&!("feed")) U<=N (alpha=i&"feed")] *
 filter(state,P=?[(alpha=i)&!("feed") U<=N2 (alpha=i&"feed")],
 alpha=i&"feed") + P=?[(!(alpha=i)&!("feed")) U<=N (alpha=i&"pick")] *
 filter(state,P=?[(alpha=i)&!("feed") U<=N2 (alpha=i&"feed")],
 alpha=i&"pick") + P=?[(!(alpha=i)&!("feed")) U<=N (alpha=i&"seeY")] *
 filter(state,P=?[(alpha=i)&!("feed") U<=N2 (alpha=i&"feed")],
 alpha=i&"seeY") + P=?[(!(alpha=i)&!("feed")) U<=N (alpha=i&"seeP")] *
 filter(state,P=?[(alpha=i&!"feed")U<=N2(alpha=i&"feed")],alpha=i&"seeP")
```

Figure 7 shows the results for switching from activity patterns α_1 to α_2 and vice-versa respectively for less than 10 button taps to feed a yoshi after switching the activity pattern, while Figure 8 shows the same but for an unbounded number of button taps (to feed a yoshi). We can see that success is much more likely by switching from α_2 to α_1, than switching from α_1 to α_2, and a user needs about 4-5 button taps to switch from α_2 to α_1 to maximise their score. This latter

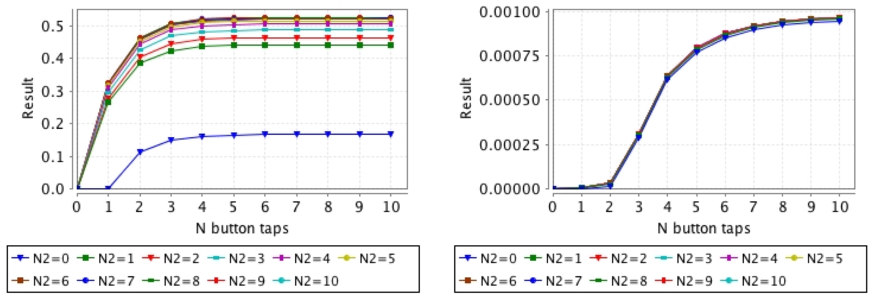


Fig. 7: Question 4 for $N_2 \leq 10$ and $i = 1$ on the left and $i = 2$ on the right.

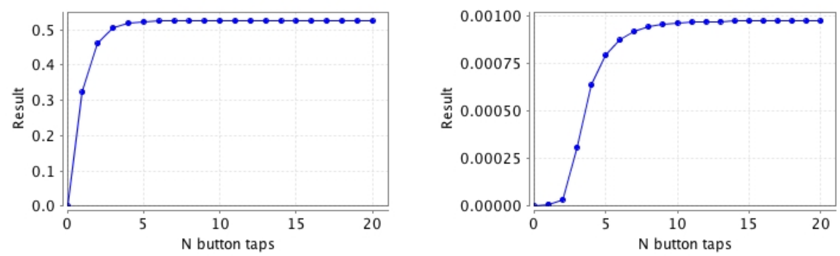


Fig. 8: Question 4 for $N_2 = \infty$ and $i = 1$ on the left and $i = 2$ on the right.

result is not surprising, considering that users might first inspect the game, which would involve visiting the 4 states.

All analyses were performed on a standard laptop. Note that for brevity, the mobile app analysed here, and its formal model, are relatively small in size; more complex applications will yield more meaningful activity patterns and complex logic properties that can be analysed on the metamodels. While state-space explosion of the UMM could be an issue, it is important to note that the state-space does not depend on the number of users, but on the granularity of the states (logged in-app actions) we distinguish.

7 Discussion

We reflect upon the results obtained for the Hungry Yoshi example and further issues raised by our approach.

Hungry Yoshi usage. Our analysis has revealed some insight into how users have actually played the game: α_1 corresponds to a more successful game playing strategy than α_2 and a user is much more likely to be effective if they change from α_2 to α_1 (rather than vice-versa), thus we conclude that α_1 is *expert* behaviour and α_2 is *ineffective* behaviour. (Note that users can, and do, switch between both behaviours, e.g. a user who exhibits expert behaviour can still exhibit ineffective behaviour at some later time.) This interpretation of activity patterns can inform a future redesign that helps users move from ineffective to expert behaviour, or induces explicitly populations of users to follow selected computation paths to reach certain goal states. We note that the developers had very little intuition about how often, or if, users were picking a full basket and then feeding a yoshi (e.g. Questions 3 and 4 in Sect. 6), and so the results, which indicate this scenario is quite rare, provided a new and useful insight for them.

Why DTMCs? Our choice of DTMC models is based on the work of of [9] in modelling web-browsing activity, usage of Microsoft Word commands, and telephone usage across populations of individuals. Girolami et al. used probabilistic convex combinations of DTMCs and demonstrated empirically that such model was superior in predictive performance to single DTMCs and mixture (point-mass) of DTMCs. Future work involves developing algorithms for inference of Hierarchical Hidden Markov models, where the first abstract level in the hierarchy is the activity patterns.

Temporal properties. The properties refer to propositions about user-initiated events (e.g. *seeY, feed*) and activity patterns (e.g. α_1, α_2). A future improvement would be a syntax that parametrises the temporal operators by activity pattern. We note that PCTL properties alone were insufficient for our analysis and we have made extensive use of *filtered* properties. We also note that for some properties we have used PRISM rewards, e.g. to compare scores between activity patterns, but these are omitted in this short paper.

Reasoning about users. Model checking is performed on the UMM resulting from the augmentation of the set of K activity patterns with a strategy θ_m. It is simple to select a user by selecting a θ_m and to analyse the resulting UMM. Metrics on the set $\{\theta_m \mid m = 1, \ldots, M\}$ will be used in future work to characterise how the results of the analysis change depending on the value of one θ_m, in the hope that results of the analysis for one user can be generalised to users close by (under the given metric).

Formulating hypotheses: domain specific and generic. We have considered domain specific hypotheses presented by developers and evaluators, but could a formal approach help with hypothesis generation? For example, we could frame questions using the specifications patterns for probabilistic quality properties as defined in [10] (probabilistic response, probabilistic precedence, etc.). Referring to our questions in Sect. 2.1, we recognise in the first item the *probabilistic precedence* pattern, in the second one the *probabilistic response* pattern, and in the last two the *probabilistic constrained response* pattern. However, these patterns refer only to the top level structure, whereas all our properties consist of multiple levels of embedded patterns. Perhaps more complex patterns are required for our domain? The patterns of [10] were abstracted from avionic, defence, and automotive systems, which are typical reactive systems; does the mobile app domain, or domains with strong user interaction exhibit different requirements? We remark also that analysis of activity patterns is just one dimension to consider: there are many others that are relevant to tailoring software to users, for example software variability and configuration, and user engagement. These are all topics of further work.

Choosing K activity patterns. What is the most appropriate value for K, can we guide its choice? While we could use model selection or non-parametric methods to infer it, there might be domain-based reasons for fixing K. For example, we can start with an estimate value of K and then compare analysis activity patterns: if properties for two different activity patterns give very close quantitative results then we only need a smaller K.

What to log? This is a key question and depends upon the propositions we examine in our properties, as well as the overheads of logging (e.g. on system performance, battery, etc.) and ethical considerations (e.g. what have users agreed). Formal analysis will often lead to new instrumentation requirements, which in turn will stimulate new analysis. For example, our analysis of Hungry Yoshi has indicated a need for logged traces to include more information about current context, e.g. the observable access points (yoshis).

8 Related Work

Our work is a contribution to the new *software analytics* described in [11], focusing on local methods and models, and user perspectives. It is also resonates with their prediction that by 2020 there will be more use of analytics for mobile apps and games. Recent work in analysis of user behaviours in systems, especially XBox games, is focused on understanding how features are used and how to drive users to use desirable features. For example, [12] investigates video game skills development for over 3 million users based on analysis of users' TrueSkill rating [13]. Their statistical analysis is based on a single, abstract "skill score", whereas our approach is based on reasoning about computation paths relating to in-app events and temporal property analysis of activity patterns. Our approach can be considered a form of run-time quantitative verification (by probabilistic model checking) as advocated by Calinescu et al. in [14]. Whereas they consider functional behaviour of service-based systems (e.g. power management) and software evolution triggered by a violation of correctness criteria because software does not meet the specification, or environment change, we address evolution based on behaviours users *actually* exhibit and how these behaviours relate to system requirements, which may include subtle aspects such as user goals and quality of experience. Perhaps of more relevance is the work on off-line runtime verification of logs in [15] that estimates the probability of a temporal property being satisfied by program executions (e.g. user traces). Their approach and results could help us determine how logging sampling in-app actions and app configuration affects analysis of user behaviour. The work of [16] employing

Hidden Markov Chains models (HMMs) is related to our approach, however our focus on capturing behavioural characteristics that are shared across a population forces us to consider a model whose distributed representation cannot be captured by HMMs. Finally we note the very recent work of [5] on a similar approach and comment the major differences in Sect. 4. In addition they analyse REST architectures (each log entry corresponds to a web page access), whereas the mobile apps we are analysing are not RESTful, we can include more fine grained and contextual data in the logged user data.

9 Conclusions and Future Work

We have outlined our contribution to software analytics for user interactive systems: a novel approach to probabilistic modelling and reasoning over actual user behaviours, based on systematic and automated logging and reasoning about users. Logged user traces are computation paths from which we infer *activity patterns*, represented each by a DTMC. A user meta model is deduced for each user, which represents users as mixtures over DTMCs. We encode the user metamodels in the probabilistic model checker PRISM and reason about the metamodel using probabilistic temporal logic properties to express hypotheses about user behaviours and relationships within and between the activity patterns.

We motivated and illustrated our approach by application to the Hungry Yoshi mobile iPhone game, which has involved several thousands of users worldwide. We showed how to encode some example questions posed by developers and evaluators in a probabilistic temporal logic, and obtained quantitative results for an example user metamodel. After considering our formal analysis of two activity patterns, we conclude the two activity patterns distinguish *expert* behaviour from *ineffective* behaviour and represent different strategies about how to play the game. While in this example the individual activity pattern DTMCs are small in number and size, in more complex settings it will be impossible to gain insight into behaviours informally, and in particular to insights into relationships between the activity patterns, so automated formal analysis of the user metamodels will be essential.

In this paper we have focused on defining the appropriate statistical and formal models, their encoding, and reasoning using model checking. We have not explored here the types of insights we can gain into user behaviours from our approach, nor how we can employ these in system redesign and future system design, especially for specific subpopulations of users. Further, in this short paper, we have not considered the role of prediction from analysis and the possibilities afforded by longitudinal analysis. For example, how do the activity patterns and properties compare between users in 2009 and users in 2013? This is ongoing work within the *A Population Approach to Ubicomp System Design* project, where we are working with system developers on the practical application of our formal analysis in the design and redesign of several new apps. We are also investigating metrics of user engagement, tool support, and integration of this work with statistical and visualisation tools.

Acknowledgments. This research is supported by EPSRC Programme Grant *A Population Approach to Ubicomp System Design* (EP/J007617/1). The authors thank all members of the project, and Gethin Norman for fruitful discussions.

References

1. McMillan, D., Morrison, A., Brown, O., Hall, M., Chalmers, M.: Further into the Wild: Running Worldwide Trials of Mobile Systems. In Floréen, P., Krüger, A., Spasojevic, M., eds.: Proc. of Pervasive'10. Volume 6030 of LNCS., Springer (2010) 210–227
2. Hall, M., Bell, M., Morrison, A., Reeves, S., Sherwood, S., Chalmers, M.: Adapting ubicomp software and its evaluation. In: Proc. of EICS'09, New York, NY, USA, ACM (2009) 143–148
3. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
4. Kwiatkowska, M.Z., Norman, G., Parker, D.: Stochastic Model Checking. In Bernardo, M., Hillston, J., eds.: SFM. Volume 4486 of LNCS., Springer (2007) 220–270
5. Ghezzi, C., Pezzè, M., Sama, M., Tamburrelli, G.: Mining Behavior Models from User-Intensive Web Applications. In Jalote, P., Briand, L.C., van der Hoek, A., eds.: Proc. of ICSE'14, ACM (2014) 277–287
6. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society. Series B (Methodological) **39**(1) (1977) pp. 1–38
7. Higgs, M., Morrison, A., Girolami, M., Chalmers, M.: Analysing User Behaviour Through Dynamic Population Models. In: Proc. of CHI'13, Extended Abstracts on Human Factors in Computing Systems. CHI EA'13, ACM (2013) 271–276
8. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: Proc. of CAV'11. Volume 6806 of LNCS., Springer (2011) 585–591
9. Girolami, M., Kaban, A.: Simplicial Mixtures of Markov Chains: Distributed Modelling of Dynamic User Profiles. In Thrun, S., Saul, L., Schölkopf, B., eds.: Advances in Neural Information Processing Systems 16. MIT Press, Cambridge, MA (2004)
10. Grunske, L.: Specification patterns for probabilistic quality properties. In Schäfer, W., Dwyer, M.B., Gruhn, V., eds.: Proc. of ICSE'08, ACM (2008) 31–40
11. Menzies, T., Zimmermann, T.: Software Analytics: So What? IEEE Software **30**(4) (2013) 31–37
12. Huang, J., Zimmermann, T., Nagappan, N., Harrison, C., Phillips, B.: Mastering the art of war: how patterns of gameplay influence skill in Halo. In Mackay, W.E., Brewster, S.A., Bødker, S., eds.: Proc. of CHI'13, ACM (2013) 695–704
13. Herbrich, R., Minka, T., Graepel, T.: Trueskill$^{\mathrm{TM}}$: A Bayesian skill rating system. Proc. of NIPS'06 (2006) 569–576
14. Calinescu, R., Ghezzi, C., Kwiatkowska, M.Z., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Commun. ACM **55**(9) (2012) 69–77
15. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime Verification with State Estimation. In: Proc. of RV'11. Volume 7186 of LNCS., Springer (2011) 193–207
16. Bartocci, E., Grosu, R., Karmarkar, A., Smolka, S.A., Stoller, S.D., Zadok, E., Seyster, J.: Adaptive Runtime Verification. In Qadeer, S., Tasiran, S., eds.: Proc. of RV'12. Volume 7687 of LNCS., Springer (2012) 168–182