Checking Models for Activity Recognition

Martin Nyolt, Kristina Yordanova and Thomas Kirste
Institute of Computer Science, University of Rostock, Germany

Keywords: Activity Recognition, Model Checking.

Abstract: Model checking is well established in system design and business process modelling. Model checking ensures and automatically proves safety and soundness of models used in day-to-day systems. However, the need for model checking in activity recognition has not been realised. Models for activity recognition can be built by prior knowledge. They can encode typical behaviour patterns and allow causal reasoning. As these models are manually designed they suffer from modelling errors. To address the problem, we discuss different classes of sensible properties and evaluate three different models for activity recognition. In all cases, modelling errors and inconsistencies have been found.

1 INTRODUCTION

Activity recognition is an important yet challenging task with applications in ubiquitous and pervasive computing, smart environments, and ambient assisted living. The key problem is to identify actions and activities of agents (i. e. humans) as well as interactions with the environment and other agents. For this purpose, sensors are usually distributed throughout the environment or the agents by means of dense sensing (Chen et al., 2012), computer vision (Aggarwal and Ryoo, 2011), or wearable sensing (Lara and Labrador, 2013). One approach is to build human behaviour models based on prior knowledge about (causal) activity structure, incorporating domain and application specific knowledge (Hiatt et al., 2011; Krüger et al., 2013). These generative models, when encoding the behaviour of the environment and the agents, allow reasoning about the current activity structure, and at the same time enable assisting actions when used as input to a classical planning algorithm.

These models have to be causally correct and represent all desired details. For instance, if a person has taken a knife from a counter, the model should always allow to put the knife down as long as the person is at the counter or a table. On the one hand, sophisticated models cover many causal dependencies between fine-grained activities in a detailed state space. Consider a system assisting an elderly in a kitchen. To provide adequate assistance, the system must detect and track activities involving many utensils at different places, possibly guiding towards the location of the utensil appropriate for the current situation. On the other hand, a model that allows all unrelated physical interactions quickly becomes too complex, rendering (probabilistic) inference infeasible due to large state spaces. Therefore, techniques have been developed to reduce these models on a symbolic level (Yordanova et al., 2014), reducing the number of required actions and states to distinguish. The result is a conflict between creating complex models and reducing and fine-tuning the model. This tuning is error-prone and increases chances for models to become inconsistent. Clearly the model should be free of errors, otherwise it could lead to undesired or even harmful actions of the system. Therefore, the need for checking consistency of activity recognition models arises.

This paper presents first results and experiences on model checking for activity recognition. Model checking is well established for verifying soundness and correctness of models (Clarke et al., 1999). It has been successfully applied in system design and business process modelling, among others. To the best of our knowledge, model checking has not been applied to activity recognition models before.

In most previous publications activity recognition models are assumed to exist. Only little work has been done on the model development process for behaviour models for activity recognition (Yordanova and Kirste, 2014). This paper raises awareness and discusses the need for model checking. Three activity recognition models are evaluated, which all have been designed to work in and have been applied to a real smart environment (Krüger et al., 2012; Krüger et al., 2013; Krüger et al., 2014). We present and discuss common modelling errors and sensible types of

Nyolt M., Yordanova K. and Kirste T..
Checking Models for Activity Recognition.
DOI: 10.5220/0005275204970502
In *Proceedings of the International Conference on Agents and Artificial Intelligence* (ICAART-2015), pages 497-502
ISBN: 978-989-758-074-1

properties to detect them. Additionally, we discuss the impact of model checking on activity recognition.

2 BACKGROUND

2.1 Related Work

Due to its wide success, model checking has found application in AI and many researchers also discovered model checking as a utility for activity recognition. Magherini et al. (Magherini et al., 2013) investigate the applicability of temporal logic and model checking to the problem of plan recognition (assuming actions observations). They present a linear temporal logic reasoning about past events and real-time constraints on actions. The application scenario is the recognition of activities of daily living, tracking the activities and calling assistance on potentially erroneous (e. g. dangerous) behaviour. Each activity is represented as one formula in the temporal logic, and at least one additional formula for erroneous executions of the actions. Inference works by finding that activity (i. e. formula) that is satisfied by the observed action sequence.

Cimatti et al. (Cimatti et al., 2003) design a planner in non-deterministic domain, i. e. domains with actions that have non-deterministic effects. They apply techniques from symbolic model checking and represent sets of states (possible outcomes of actions) as propositional formulae. Based on this model-based planner (MBP), Gromyko et al. (Gromyko et al., 2006) synthesise a controller for non-deterministic discrete event systems. Given required properties in temporal logic and a system, they generate a controller for this system such that it satisfies the properties.

In many cases applying model checking techniques, the idea is to compactly represent states and efficiently generate a search tree. This is often done using efficient data structure such as Binary Decision Diagrams (BDDs). In contrast to Magherini et al., we do not recognise activities using model checking techniques. We apply model checking in its original sense: proving that the models (used for activity recognition) fulfil all desired properties.

2.2 State Space Models for Activity Recognition

This section introduces the types of models used for activity recognition. In Sect. 3 we then describe how to apply model checking.

```
(:action print                      (:action repair
  :parameters (?j -job)               :parameters(?a -agent)
  :precondition (and                  :precondition (and
    (not (printed ?j))                  (at printer ?a)
    (not jammed)                        (hands-free ?a)
    (has paper printer))                (jammed))
  :effect (printed ?j))               :effect
                                        (not jammed))
```
Listing 1: Simple PDDL domain definition for the office scenario (excerpt).

When reasoning about human behaviour, state space models (Koller and Friedman, 2009) built from prior knowledge are one option as a flexible foundation for reasoning about human behaviour. They have recently been used for different activity and intention recognition applications (Hiatt et al., 2011; Krüger et al., 2013; Ramírez and Geffner, 2010) for several reasons. First, learning (discriminative or generative) probabilistic models requires a vast amount of training data. Acquiring this data involves a lot of human subjects and is therefore time-consuming and costly. Second, human behaviour varies considerably even for small domains like food preparation. It is infeasible to record sufficient training data for all possible variations. Finally, generative models easily allow *prediction*, goal recognition (Ramírez and Geffner, 2010), and support assistance (Hiatt et al., 2011; Hoey et al., 2011).

In addition, this general framework allows to easily model long-term dependencies. For example in a meeting scenario, a projector only can be turned off after it has been turned on, no matter how many other activities have been executed. This framework also supports latently infinite state spaces, like counting how many objects are currently taken by some agent.

In our examples we use a syntax based on PDDL (Fox and Long, 2003) for model specification. PDDL is used because it is widely adopted for planning (and can thus be directly applied for assistance problems), but also previous activity and goal recognition approaches are based on it (Krüger et al., 2013; Ramírez and Geffner, 2010). Actions are defined in terms of preconditions and effects on state predicates. Listing 1 shows a simple excerpt from one of our models. The model also comprises an initial world state and a goal formula identifying goal states for a given application domain. Additionally, the syntax is enriched with a duration model and observation model. For activity recognition, the model is then compiled to a probabilistic model for Bayesian inference, i. e. computing the most likely state given past observations. Because we want to reason about detailed context and activities, the model easily becomes complex.

3 MODEL-CHECKING APPROACH

We employ the PRISM tool as a state-of-the-art, off-the-shelf model checker (Kwiatkowska et al., 2011). We have chosen PRISM because it supports a variety of models, the property language covers many features, it can produce counter-examples for certain properties, and uses BDDs for efficiently storing large state spaces. A compiler transforms the PDDL specification into PRISM's syntax. The compiler instantiates the actions to ground actions (removing any parameters) and resolves conditional effects (which are not supported by PRISM) by generating different actions for each possible combinations of conditional effects. Although our action models contain durations, the system itself evolves in discrete steps. Our focus is only on qualitative aspects of the model, so the duration models and the observation models can be simply dropped.

The model checker can verify if a model (i. e. the sequence of states generated by the model) satisfies a boolean formula written in temporal logic. These formulas are often called properties, as they describe desired behaviour of the model. Any violation of the properties is considered a modelling error. Temporal formulae are defined over sequences of states. A formula ϕ references atomic state propositions and usually consist of one or more of the operators `X` ϕ (ϕ must hold in the next state), `F` ϕ (ϕ must hold in at least one state of the sequence), `G` ϕ (ϕ must hold in all states of the sequence), `E` ϕ (ϕ must hold in at least one possible path, e. g. due to non-determinism), or `A` ϕ (ϕ must hold in all possible paths), among others.

The formula in PRISM can refer to state variables as well as labels, which are user-defined formulas on state variables. The compiler automatically generates a "goal" label from the goal definition of the model, making it easier to reference goal states. Properties are usually of the form `A` ϕ, as all possible executions of behaviour models should satisfy the property. An example in PRISM syntax is `A [ F "goal" ]`, stating that it is always possible to reach a goal state.

Defining properties is the most important step, as this encodes all knowledge and intuition that should hold in the domain. This is particularly important for behavioural models as presented in Sect. 2.2. These models are described on a symbolic level (in this case using PDDL) in terms of the *action*'s preconditions and effects. In contrast to, for example, functional tests or unit tests in software engineering, the designer should not focus on properties of single actions. One might be tempted to compare properties with contracts in programming by contract with imperative implementations (Meyer, 1992), where functions also have pre-conditions and post-conditions. Although the concept is different in testing or verification, from a designer perspective formulating properties or contracts seems to be a comparable task. Both ensure that actions or functions are only called in certain contexts and have a desired outcome.

However, symbolic action descriptions as in PDDL are declarative, so the gain from imperative functions to declarative properties is not existent when formulating properties for PDDL domains. Even worse, the properties are likely to be defined by the model designer, who will simply re-state the actions' preconditions and effects, possibly making the same high-level conceptual errors. Therefore we argue it is most useful to describe properties that cover long-term dependencies, and not simply assertions to execute specific actions. We identified various types of properties to be important, which can be grouped into invariants and long-term causation.

Invariants. are properties that need to be satisfied at all times. The two most salient properties are *deadlock freeness* and *livelock freeness*. Deadlocks are states in which no action can be executed. Deadlock freeness can be checked using the formula `A [ G !"deadlock" ]` (PRISM automatically provides a label "deadlock" for deadlock states). Deadlocks should be avoided in behaviour models. Either deadlock states are impossible in the underlying domain, in which case they should be unreachable, or they are possible, in which case actions are probably missing and the inference process cannot recognise subsequent activities. The underlying assumption is that humans almost always find a solution or continue acting in any case, so that deadlocks do not occur in the application (of course, domain-specific exceptions may exist). Resolving the deadlock depends on the model and the intended behaviour. Any reasonable model checker can provide a counterexample path to violated formulas of the form `A G` ϕ, thus giving a sequence to a deadlock.

Livelocks, in our activity recognition case, are states in which actions are possible, but no goal state can be reached (checked with `A [ F "goal" ]`). They indicate a possible problem in the domain or model, where the overall task is impossible to accomplish. Livelocks are more difficult to track, as usually no counter-examples can be given. One strategy is to split the goal formula and verify that parts of the goal can be reached.

Additionally, domain-specific invariants are candidates for properties, for example the property that exactly one of a set of state variables is always true

(e. g. a user may hold only up to two objects in his hands). Especially important are those invariants which are related to multiple actions, where the invariants cannot be captured by preconditions and effects of a single action. An example might be multiple actions where it is possible that the user holds an additional object in hand (e. g. either by taking from a surface or by disassembling one object). These formulas are often of the form $\texttt{A G}\ \phi$, thus any violation can be proven with a counterexample.

Long-term Causations. between actions ensure properties which can only be implicitly encoded within multiple actions. This includes typical action sequences such as turning devices on and off (e. g. lamps or stoves) with intermediate actions, where turn on is only allowed for devices previously turned off and vice versa. Sometimes models include *decisions* of users (e. g. where to sit, when to present) which do not immediately influence the behaviour, but alter the progress of the model when the decisions are realised later. Here properties typically are of the scheme $\texttt{A [G (}\phi \Rightarrow F\,\psi\texttt{)]}$, meaning only when ϕ holds, the model always behaves in a particular way according to ψ. In real-world domains, *repeatability of certain actions* are desired, like locomotion or basic interactions with the environment. These properties often follow the scheme $\texttt{A [G (F}\ \phi\texttt{)]}$, stating that it must always be possible to reach a state where ϕ holds. On the other hand, some *actions can only be executed once* in certain applications, for instance ingredients can only be cut once.

Some of these properties, when violated, may not always indicate a modelling error. Instead, they can provide useful insight to the problem domain and spot unexpected effects that may or may not be desirable. Other violations may point out possible chances for reducing the model complexity and state space, such as limiting the model behaviour to the particular application domain. It must be noted that unsatisfied properties, deadlocks, and livelocks not necessarily influence recognition results negatively. If some behaviour not present in the datasets is not modelled, recognition accuracy usually increases because less behaviour must be discriminated. Therefore these "errors" may sometimes be intended. Nonetheless, it must be kept in mind that this hinders re-usability of the models and can lead to unexpected problems, especially if they are not documented.

4 EVALUATION OF ACTIVITY RECOGNITION MODELS

For the evaluation of our activity recognition tool, we conducted several experiments in our smart environment and built various behaviour models. During development, we identified a number of modelling errors and inconsistencies in the models. Each section briefly presents the models and properties, and discusses the modelling errors found. Note that this is an empirical review of actual modelling errors, none of these errors has been artificially added to the model. An overview of the number of actions and states is shown in Table 1, giving an intuition of each model's complexity.

4.1 The Meeting Scenario

Three persons hold a presentation and discuss in a meeting room. The model's actions comprise walking to different locations, sitting on a chair, as well as actually presenting and discussing. A presentation can only start when the other persons are seated. To restrict the model, it was assumed uncommon to change seats and walk to different locations, so a person was only allowed to walk once before and between presentations.

We checked two domain-specific invariants and one unrepeatable action property: (1) at most one person is presenting, (2) when presenting, all others are seated, and (3) a person never presents twice. All properties were satisfied, no livelock was present, but one deadlock could be found. The deadlock was caused by the way walking (e. g. to seats) has been implemented. When the people enter the room they independently execute the action "walk to seat *x*". The deadlock happens if all persons execute the same action and go to the same seat, which is possible as the seat is still empty. When the persons arrive, only one can sit down, and the others could not move because walking was allowed only once.

Two options exist to remove the deadlock: reduce behaviour that leads to the deadlock or add behaviour to escape it. Reducing behaviour would require the persons to negotiate where to seat, which seems rather unnatural and complicates modelling even more. Allowing additional walking fixes the deadlock, but increases the state space to 49,765 states, making it rather costly.

4.2 The Office Domain

A room contains a printer and a coffee machine that people want to use concurrently. The model contains

Table 1: Comparison of model complexity in terms of actions, boolean state variables, and state space size.

Model	PDDL actions	ground actions	state variables	states
meeting (3 persons)	10	150	104	31,372
office (3 persons)	9	124	43	1,957,158
kitchen (1 person)	44	99	60	146,552,922

actions for entering and exiting the room and walking to different locations. Paper and ground coffee are resources that can be taken (only one at a time if the hands are free) from different locations and used to refill the printer or coffee machine. A paper jam may occur, which must be repaired before printing is possible.

We checked the following properties with respect to a single agent: (1) no hands are free if and only if some object is held, (2) when holding something it is possible to get the hands free, and (3) it is always possible to leave the room. These properties are example of an invariant (covering different state variables), an intuitive assumption, and of ensuring repeatability of an action.

The second property was violated, no additional deadlocks or livelocks have been found. Here, the agent could get some ground coffee for the coffee machine, refill the coffee machine, and get some additional ground coffee. After getting additional ground coffee, the coffee machine still has resources and the agent is unable to release the ground coffee from his hand, allowing him only to walk between different locations without making any additional progress. A simple fix is to add an additional “drop” action, allowing to put resources back.

4.3 The Kitchen Task

A single person prepares and eats a carrots soup. This model comprises detailed actions in the process of preparing and eating a carrots soup as well as cleaning afterwards. The actions respect several causal relations, such as an object can only be taken if at least one hand is free, and cutting the carrot is only possible when a knife is in a hand and the carrot is on a cutting board.

The properties we checked included an invariant and two long-term causations: (1) there is exactly one of the propositions true that zero, one, or two hands are free, (2) when a person is not hungry he has cooked, and (3) when eating, the person is seated and at the table. While the first two properties were true, the last indeed turned out to be false. The model allowed to eat without seating, it was sufficient to just stand at the table.

No deadlocks were present in the model, but a livelock could be found. As a precondition for the cook action, the spoon had to be in the hand. But in case the spoon was in the pot before cooking, it was impossible to remove the spoon, as the respective actions were too restrictive. In this case two fixes are possible. Either the actions could be relaxed to allow the spoon to be removed from the pot. Or the action could be restricted even more, preventing to put the spoon in the pot before finishing cooking. While the first option does not change the size of the state space, the second slightly decreases its size to 145 million states.

5 CONCLUSION

Model-based approaches to activity recognition require sound models. Model checking helps finding modelling errors and is therefore supposed to improve recognition rates. Improper models can not only have negative impact on recognition results, they can possibly lead to wrong, possibly harmful, decisions. We presented a short case study of modelling errors found in activity recognition models. As a guide for practitioners and model developers, we identified and discussed important classes of sensible properties to identify modelling inconsistencies and errors, and how they may impact activity recognition.

Future work includes a detailed survey of important property classes, making it easier to formulate sensible properties. Currently, we employed an external model checking tool. An integrated, continuous tool support has to developed, which is presumed to ease model development.

ACKNOWLEDGEMENTS

This research is supported by the German Research Foundation (DFG) within the research training group GRK 1424 MuSAMA (Multimodal Smart Appliance Ensembles for Mobile Applications).

REFERENCES

Aggarwal, J. K. and Ryoo, M. S. (2011). Human activity analysis: A review. *ACM Computing Surveys*, 43(3):16:1–16:43.

Chen, L., Hoey, J., Nugent, C. D., Cook, D. J., and Yu, Z. (2012). Sensor-based activity recognition. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(6):790–808.

Cimatti, A., Pistore, M., Roveri, M., and Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84.

Clarke, Jr., E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. MIT Press.

Fox, M. and Long, D. (2003). PDDL2.1: Expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124.

Gromyko, A., Pistore, M., and Traverso, P. (2006). A tool for controller synthesis via symbolic model checking. In *Discrete Event Systems, 2006 8th International Workshop on*, pages 475–476.

Hiatt, L. M., Harrison, A. M., and Trafton, J. G. (2011). Accommodating human variability in human-robot teams through theory of mind. In *Proc. of the Int. Joint Conference on Artificial Intelligence*, pages 2066–2071.

Hoey, J., Plötz, T., Jackson, D., Monk, A., Pham, C., and Olivier, P. (2011). Rapid specification and automated generation of prompting systems to assist people with dementia. *Pervasive and Mobile Computing*, 7(3):299–318.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models*. MIT Press.

Krüger, F., Nyolt, M., Yordanova, K., Hein, A., and Kirste, T. (2014). Computational state space models for activity and intention recognition. A feasibility study. *PLoS One*.

Krüger, F., Steiniger, A., Bader, S., and Kirste, T. (2012). Evaluating the robustness of activity recognition using computational causal behavior models. In *Proc. ACM Conference on Ubiquitous Computing*, pages 1066–1074.

Krüger, F., Yordanova, K., Hein, A., and Kirste, T. (2013). Plan synthesis for probabilistic activity recognition. In *Proc. 5th ICAART*, pages 283 – 288.

Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd Int. Conference on Computer Aided Verification*, volume 6806, pages 585–591.

Lara, Ó. D. and Labrador, M. A. (2013). A survey on human activity recognition using wearable sensors. *Communications Surveys Tutorials, IEEE*, 15(3):1192–1209.

Magherini, T., Fantechi, A., Nugent, C. D., and Vicario, E. (2013). Using temporal logic and model checking in automated recognition of human activities for ambient-assisted living. *Human-Machine Systems, IEEE Transactions on*, 43(6):509–521.

Meyer, B. (1992). Applying “design by contract”. *Computer*, 25(10):40–51.

Ramírez, M. and Geffner, H. (2010). Probabilistic plan recognition using off-the-shelf classical planners. In *Proc. AAAI-10, 24th Conf. Artificial Intelligence*.

Yordanova, K. and Kirste, T. (2014). Towards systematic development of symbolic models for activity recognition in intelligent environments. In *Proc. 3rd Workshop on AI Problems and Approaches for Intelligent Environments, held at ECAI 2014*.

Yordanova, K., Nyolt, M., and Kirste, T. (2014). Strategies for reducing the complexity of symbolic models for activity recognition. In *16th International Conference on Artificial Intelligence: Methodology, Systems, Applications*.