Interactive Planning-based Hypothesis Generation with LTS++

Shirin Sohrabi and **Octavian Udrea** and **Anton V. Riabov** and **Oktie Hassanzadeh**
IBM T.J. Watson Research Center
1101 Kitchawan Rd, Yorktown Heights, NY 10598, USA
{ssohrab, udrea, riabov, hassanzadeh}@us.ibm.com

Abstract

We present LTS++, an interactive development environment for planning-based hypothesis generation in applications with unreliable observations.

1 Introduction and Motivation

The set of planning-based tools, collectively called LTS++, presented in this paper address the hypothesis generation problem that arises in applications that require multiple hypotheses to be generated in order to reason about possibly incomplete or inconsistent sequences of observations received from external sources. For example, when analyzing observations derived from sensor data in intensive care, the goal can be to generate plausible hypotheses about the condition of the patient. The resulting hypotheses can then be further refined and analyzed to create a recovery plan for the patient. In another application, decisions aimed to prevent malware spread in computer networks can be based on hypotheses about change in behavior of individual hosts generated by reasoning about observations of network traffic over time.

The core idea of the approach to planning-based hypothesis generation we implement in LTS++ is the following. Modeling the hypothesis generation problem as one of inferring a sequence of state transitions from a sequence of observations, and transforming the sequence of observations together with the state transition model into a planning task, we employ a planner to produce a set of multiple near-optimal plans, to which we then apply an inverse transform, obtaining a set of hypotheses where each hypothesis is a sequence of states annotated with explained, missed or ignored observations.

Knowledge engineering requirements come to the forefront in designing a system like LTS++, where domain knowledge is encoded and maintained directly by the domain experts, such as clinicians or network security engineers. To address these requirements, we developed the LTS++ language that allows the domain experts to easily describe the state transition models and observations specific to their domain, without requiring the experts to learn about the underlying planning technologies or PDDL. The LTS++ browser-based integrated development environment (IDE) includes an editor with syntax highlighting and static error checking, as well as integrated tools for interactive model testing and debugging, generating and visualizing multiple hypotheses for user-provided observations. Models created in the IDE can then be deployed to LTS++ servers to generate hypotheses automatically as observations are received, generating alerts based on hypotheses for further analysis by experts or other systems.

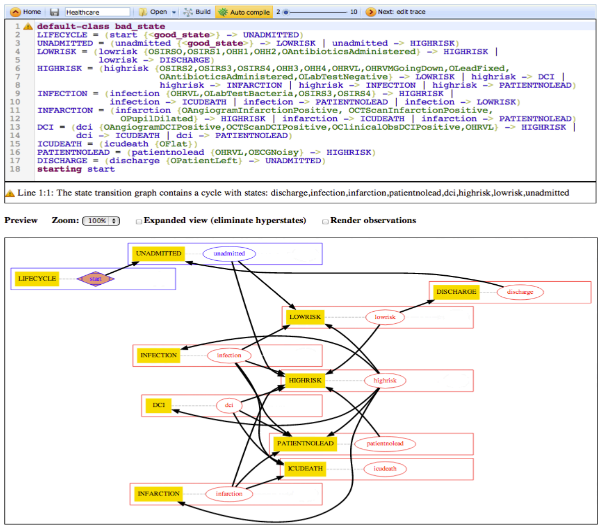
We build upon a significant body of prior research. While expert judgment is the primary method used for generating hypotheses and evaluating their plausibility, automated methods have been proposed, to assist the expert, and help improve accuracy and scalability. However, most of the existing literature makes an assumption that the observations are reliable and should all be explainable according to the model. But that is not true in general; as a further complication, we cannot assume the system model is complete. The hypothesis generation approach we propose handles the unreliable observations and incomplete models by offering multiple alternative hypotheses explaining each given observation sequence, and the LTS++ tools we developed automate the generation and evaluation of hypotheses in addition to addressing the knowledge engineering challenges of encoding and maintaining models.

2 Key Ideas Implemented in LTS++

Planning Formulation We transform the hypothesis generation problem into a planning problem. In particular, we extend the work of Sohrabi et al. 2011 to address unreliable observations and generate multiple near-optimal lowest-cost plans, mapping the generated plans to hypotheses. This mapping ensures that lower cost plans are mapped to more plausible hypotheses, hence finding a number of lowest-cost plans results in the same number of most plausible hypotheses. See [Sohrabi *et al.*, 2013; Riabov *et al.*, 2015] for more details.

Top-k Planning Our LTS++ implementation uses an efficient planner that finds top-k plans, i.e., k plans such that no valid plans with lower cost exist. We have evaluated several algorithms for this purpose, and currently use the k-shortest path algorithm K* [Aljazzar and Leue, 2011].

Knowledge Engineering We have designed the LTS++ language, derived from LTS (Labeled Transition System), for defining a model. The LTS++ language allows specification of the observations and the states of the entity (e.g., state of the patient could be Delayed Cerebral Ischemia (DCI) or Infection). It also allows domain experts to describe the transitions between states, as well as the association between states and observations. The language is supported by the IDE.



```
default-class bad_state
LIFECYCLE = (start {<good_state>} -> UNADMITTED)
UNADMITTED = (unadmitted {<good_state>} -> LOWRISK | unadmitted -> HIGHRISK)
LOWRISK = (lowrisk {OSIRSO,OSIRS1,OHH1,OHH2,OAntibioticsAdministered} -> HIGHRISK |
           lowrisk -> DISCHARGE)
HIGHRISK = (highrisk {OSIRS2,OSIRS3,OSIRS4,OHH3,OHH4,OHRVL,OHRVMGoingDown,OLeadFixed,
                      OAntibioticsAdministered,OLabTestNegative} -> LOWRISK | highrisk -> DCI |
                      highrisk -> INFARCTION | highrisk -> INFECTION | highrisk -> PATIENTNOLEAD)
INFECTION = (infection {OHRVL,OLabTestBacteria,OSIRS3,OSIRS4} -> HIGHRISK |
             infection -> ICUDEATH | infection -> PATIENTNOLEAD | infection -> LOWRISK)
INFARCTION = (infarction {OAngiogramInfarctionPositive, OCTScanInfarctionPositive,
              OPupilDilated} -> HIGHRISK | infarction -> ICUDEATH | infarction -> PATIENTNOLEAD)
DCI = (dci {OAngiogramDCIPositive,OCTScanDCIPositive,OClinicalObsDCIPositive,OHRVL} -> HIGHRISK |
       dci -> ICUDEATH | dci -> PATIENTNOLEAD)
ICUDEATH = (icudeath {OFlat})
PATIENTNOLEAD = (patientnolead {OHRVL,OECGNoisy} -> HIGHRISK)
DISCHARGE = (discharge {OPatientLeft} -> UNADMITTED)
starting start
```

Line 1:1: The state transition graph contains a cycle with states: discharge,infection,infarction,patientnolead,dci,highrisk,lowrisk,unadmitted

Preview Zoom: 100% Expanded view (eliminate hyperstates) Render observations

Figure 1: LTS++ IDE

3 LTS++ Development Environment

Model Editor The top part of the model editor screen (Figure 1) is the LTS++ language editor with syntax highlighting and the bottom part is the automatically generated transition graph. In the editor, the states appear in blue. The observations are specified within the curly brackets and appear in green. You can specify multiple observations by using space or comma between observations. The transitions between states are specified using arrows. Multiple transitions between states can be specified using a vertical bar. The LTS++ model editor automatically detects errors in LTS++ language and shows them below the text editor.

Model Testing To test the model, a sequence of observations can be entered by clicking on "Next: edit trace" from the LTS++ IDE main page. The tool automatically generates planning problems from the LTS++ specification and entered trace. The generated hypotheses are the result of running a planner and finding the most plausible hypotheses ranked by plausibility from highest to lowest. Figure 2 shows an example of hypotheses generated for the critical care model; the result is automatically generated by our tool. Each hypothesis is shown as a sequence of states matched to an observed event sequence. The observations that are explained by a state are shown in green ovals, and unexplained observations are shown in purple. The arrows between the observations show the sequence of observations in the trace. Each hypothesis is associated with a cost. The lower the cost value, the more plausible is the hypothesis.

Model Discovery and Update Our tool uses a simple bootstrapping technique to discover an initial model given a set of historical observations. Several candidate models will be presented to the domain expert who can choose one as an initial LTS++ model and further improve it. We also implement automated model updates to produce better quality hypotheses as we do not assume the model will be accurate in perpetuity. To do this, we use an aggregate measure of the plausibility of top-N hypotheses as our optimization criteria. Using a genetic algorithm, we attempt small atomic changes to the model (e.g., addition and deletion of states, observations and transitions) and measure the increase in aggregate hypothesis plausibility as a result. In subsequent generations, we combine the promising atomic changes and repeat until we can no longer increase hypothesis plausibility.

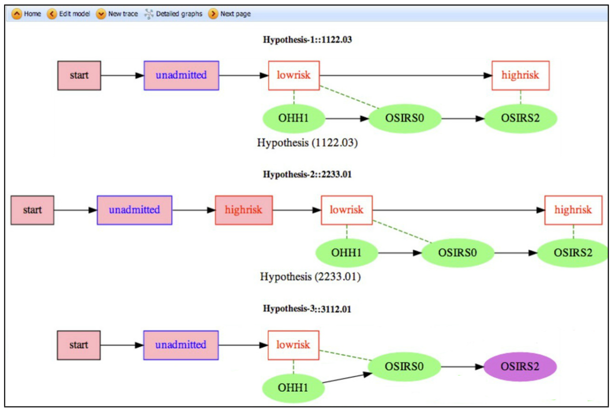


Figure 2: Sample Healthcare Example

Model Composition A single LTS++ model describes a state transition system for a single type of entity, such as a patient. Given multiple entities, each with their own associated model, our tool also allows automated composition of multiple models. It does so by considering a cross product of all possible joint states while paying special attention to the association between observations and the combined states.

Hypothesis Clustering Many of the generated hypotheses are only slightly different from each other. That is, they do seem to be duplicates of each other, except for one or more states or actions that are different. To consolidate similar plans produced by the planner, we apply a clustering algorithm to cluster similar plans and present clusters of plans, where each cluster can be replaced by its representative plan. To find similar plans, we leverage existing research in diverse planning (e.g., [Nguyen *et al.*, 2012]).

References

[Aljazzar and Leue, 2011] Husain Aljazzar and Stefan Leue. K*: A heuristic search algorithm for finding the k shortest paths. *AIJ*, 175(18):2129–2154, 2011.

[Nguyen *et al.*, 2012] Tuan Anh Nguyen, Minh Binh Do, Alfonso Gerevini, Ivan Serina, Biplav Srivastava, and Subbarao Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *AIJ*, 190:1–31, 2012.

[Riabov *et al.*, 2015] Anton V. Riabov, Shirin Sohrabi, Daby M. Sow, Deepak S. Turaga, Octavian Udrea, and Long H. Vu. Planning-based reasoning for automated large-scale data analysis. In *ICAPS*, pages 282–290, 2015.

[Sohrabi *et al.*, 2011] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. Preferred explanations: Theory and generation via planning. In *AAAI*, pages 261–267, 2011.

[Sohrabi *et al.*, 2013] Shirin Sohrabi, Octavian Udrea, and Anton Riabov. Hypothesis exploration for malware detection using planning. In *AAAI*, pages 883–889, 2013.