Journal of Artificial Intelligence Research 63 (2018) 281-359 Submitted 04/18; published 10/18

Watching and Acting Together: Concurrent Plan Recognition and Adaptation for Human-Robot Teams

Steven J. Levine SJLEVINE@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar St. Room 32-226
Cambridge, MA 02139 USA

Brian C. Williams WILLIAMS@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar St. Room 32-227
Cambridge, MA 02139 USA

Abstract

There is huge demand for robots to work alongside humans in heterogeneous teams. To achieve a high degree of fluidity, robots must be able to (1) recognize their human co-worker's intent, and (2) adapt to this intent accordingly, providing useful aid as a teammate. The literature to date has made great progress in these two areas – recognition and adaptation – but largely as separate research activities. In this work, we present a unified approach to these two problems, in which recognition and adaptation occur concurrently and holistically within the same framework. We introduce Pike, an executive for human-robot teams, that allows the robot to continuously and concurrently reason about what a human is doing as execution proceeds, as well as adapt appropriately. The result is a mixed-initiative execution where humans and robots interact fluidly to complete task goals.

Key to our approach is our task model: a contingent, temporally-flexible team-plan with explicit choices for both the human and robot. This allows a single set of algorithms to find implicit constraints between sets of choices for the human and robot (as determined via causal link analysis and temporal reasoning), narrowing the possible decisions a rational human would take (hence achieving intent recognition) as well as the possible actions a robot could consistently take (hence achieving adaptation). Pike makes choices based on the preconditions of actions in the plan, temporal constraints, unanticipated disturbances, and choices made previously (by either agent).

Innovations of this work include (1) a framework for concurrent intent recognition and adaptation for contingent, temporally-flexible plans, (2) the generalization of causal links for contingent, temporally-flexible plans along with related extraction algorithms, and (3) extensions to a state-of-the-art dynamic execution system to utilize these causal links for decision making.

1. Introduction

There is a huge demand for humans and robots to work alongside each other to collaboratively achieve tasks. This is apparent in a number of domains, including aerospace manufacturing, household robotics for assisting in daily chores, medical robotics for performing clinical procedures, and countless more. No matter what the domain, it is necessary for the robots to both (1) infer the intent of their human teammates, and (2) adapt to their intent appropriately. Without such fluidity, accomplishing a task in a collaborative manner with humans would be challenging and humans would likely find working with such robots to be laborious, thus limiting their adoption.

The literature to date has made great progress on these two areas – intent recognition and adaptation – but largely as separate research activities, and rarely with overlap. Numerous approaches to intent and plan recognition have been proposed (Sukthankar, Geib, Bui, Pynadath, & Goldman, 2014; Carberry, 2001). However, these approaches generally focus solely on the recognition task and not on selecting suitable adaptations for the robot once the recognition is achieved. In parallel, numerous approaches to robotic adaptations have been proposed over the years, but few focus explicitly on adapting to inferred intent.

This work takes the viewpoint that intent recognition and robot adaptation can and should be viewed as two sides of the same coin, and that any practicable cognitive robot must integrate both approaches seamlessly. Towards that view, this work presents a single set of algorithms that simultaneously performs intent recognition and selects robot adaptations within one framework and with one model. Both problems are framed in our approach as inference and consistency-based reasoning. By performing intent recognition and adaptation concurrently as opposed to separately, our system is able to achieve a greater degree of robustness and fail less often.

To achieve this duality, a key aspect of our approach is our shared plan representation. We operate on *contingent, temporally-flexible team plans*. Some actions in the plan are targeted at the human, while others are for the robot(s) to perform. Additionally, these plans contain explicit choices which are either controllable by the robot (allowing it to choose to perform or not perform certain actions), or uncontrollable by the robot (up to the human or nature to perform certain actions). The choices in these plans afford flexibility to react to different situations that may arise during execution by encoding different contingencies. Within this framework, an *intent* is a set of uncontrollable choices made by the human (or the environment, or any other uncontrollable agent), and an *adaptation* is a set of controllable choices made by the robot. An example of such a plan is shown in Figure 1. Our approach exploits the often implicit interconnections between these two sets of choices, controllable and uncontrollable. Namely, only certain combinations of uncontrollable choice outcomes made by the human and controllable choice outcomes made by the autonomous agent would be allowable together in successful team plans. Our approach reasons over these sets of choices to determine which outcomes would be possible, and uses that to predict which choices a rational human agent could consistently take (thus inferring intent), and simultaneously infer which choices the robot should take (thus adapting appropriately). By explicitly considering the coupling between human and robot decisions, our reasoning algorithms are thus able to consider intentions and adaptations from a unified viewpoint.

The basis for the interconnections between choices come from several different sources. The first is state: certain actions have preconditions that rely upon other previously-executed actions in order to succeed. As such, choosing to execute one action may require choices to execute other actions in the plan. The second is temporal requirement: certain combinations of choices may result in missing deadlines or other timing constraints imposed in the plan. A third is unanticipated disturbances: certain choices become infeasible given new, online observations that the world has changed in an unexpected way. Our technique addresses all three of these. We handle the first by reasoning about causal links, the second through temporal conflict extraction, and the third through online causal link execution monitoring. At a high level, these three techniques can be summarized as *keeping your eye on the goal*. Pike makes choices and monitors the plan always with respect to ensuring that the constraints of the human-robot team's plan can be met.

Our approach is divided into two phases: (1) an offline compilation phase, and (2) online execution. During offline compilation, we first compute implicit temporal relationships in the plan, which allows us to reason about which actions must precede which others. We then use this information to extract *labeled causal links* from the plan, which capture the dependence of certain actions on others in the plan. Labeled causal links generalize the notion of causal links from non-contingent plans to contingent temporally-flexible plans. Their labels capture the choices required for them to hold, which is necessary in our contingent plans where actions may or may not occur (depending on the choices made), and where producers and threats may be partially ordered. After extracting these labeled causal links, they are then translated into propositional state logic constraints where each solution to the constraints represents a successful execution of the team plan. Finally, these propositional constraints are then compiled into a form suitable for fast online use. The output of this compilation is a data structure that represents the space of all successful executions of the plan, and can be quickly queried online. A similar problem has been solved in the past by the Assumption-based Truth Maintenance System (ATMS) (de Kleer, 1986a); our compilation process is inspired by it, and extends the ATMS label propagation algorithms to generate a sound and complete set of prime implicants that are used online. The compilation allows human robot interaction to be performed quickly without the requirement of replanning when new observations or certain unexpected disturbances occur.

Our online execution algorithm uses the compiled constraints to quickly check if the robot is able to make certain choices online, and still respect temporal consistency and ensure that the preconditions of activities will be satisfied. The online algorithm also takes in observations in the form of outcomes to uncontrollable choices decided by the human, as well as a full set of observed state measurements which are the basis for causal-link based execution monitoring. This allows our executive to make incremental changes online to its knowledge base and adapt according to these changes. A grounded example is provided later in this section.

As Pike is a plan executive, it is useful here to briefly say a few remarks about the complexity of executing plans. It is well known that planning is a hard problem – STRIPS planning for example is PSPACE-complete (Bylander, 1994). It is less well known, however, that execution can also be computationally challenging, depending on the features that we desire from our executive. On one end of the spectrum, the easiest form of plan execution – namely dispatching the activities sequentially to agents – is computationally very simple. Each action can be dispatched in near constant time, as no effort is required from the executive to process ordering constraints (let alone metric temporal ones), monitor for disturbances, and more. At the other end of the spectrum, plan execution is much harder for the case where we have more complex plans with metric temporal constraints (some of which have uncontrollable durations), desire rich execution monitoring to discover potential faults before they are problematic, and where the executive is left with discrete choices (i.e., choices for the robot) that must be made in a least-commitment manner online. Such approaches are generally NP-hard mainly because the introduction of discrete choices requires combinatorial reasoning. In the middle of the spectrum, there are a num-

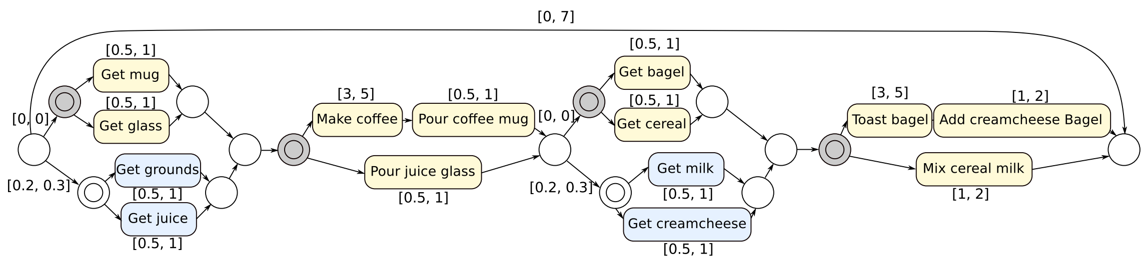


Figure 1: Contingent, temporally-flexible plan (i.e., a TPNU) for making breakfast. Circles denote events and edges denote temporal constraints. Shaded double circles denote uncontrollable choices (made by Alice) and are followed activities for Alice to perform. Unshaded double circles represent controllable choices (made by the robot), and are followed by activities targeted at the robot. Note that each activity is represented with a start and end event, but illustrated here as a box for compactness. All unlabeled temporal constraints are $[\epsilon, \infty]$ ordering constraints.

ber of approaches that provide a middle ground both in terms of polynomial computational complexity and provided features, such as executing metric temporal plans with execution monitoring but with no choice (Levine, 2012). Our work falls into the more feature-rich, yet computationally challenging end of the spectrum (see Theorem 3.1, which states that Pike's execution problem is NP-complete). We argue that an executive capable of responding to disturbances, adapting to human intent, monitoring the execution for problems, and addressing metric temporal constraints – all in a flexible, least-commitment way – would provide valuable robustness for autonomous systems and justify its greater computational complexity. Many of the techniques in this paper aim to provide reasonable performance on real-world problems despite the poor worst-case complexity guarantees.

While the focus of this paper is on human-robot collaboration, we note that we make few assumptions about the human in this work. Therefore, our approach is equally applicable to other multi-agent settings in which there are uncontrollable agents (possibly other robots).

As noted earlier, Pike is designed to take as input a contingent, temporally-flexible team plan for a human-robot team. This input plan can come from a variety of sources, such as a planner capable of outputting contingent, temporally-flexible plans (to the authors knowledge however, no such planner meets this requirements - but the closest are RAO* (Santana, Thiébaux, & Williams, 2016), tBurton (Wang & Williams, 2015), and FOND (Muise, McIlraith, & Beck, 2012)). Additionally, a control program with appropriate choice structure could be created by a human expert in RMPL and compiled into a TPNU (Kim, Williams, & Abramson, 2001; Williams, Ingham, Chung, & Elliott, 2003).

1.1 Approach in a Nutshell

We illustrate a grounded example of concurrent intent recognition and adaptation. Consider the grounded example shown in Figure 1, in which a person named Alice is making breakfast for herself with the help of her trusty robot. The left half of the plan depicts the team either making coffee (for which Alice uses a mug) or getting some juice (for which Alice uses a

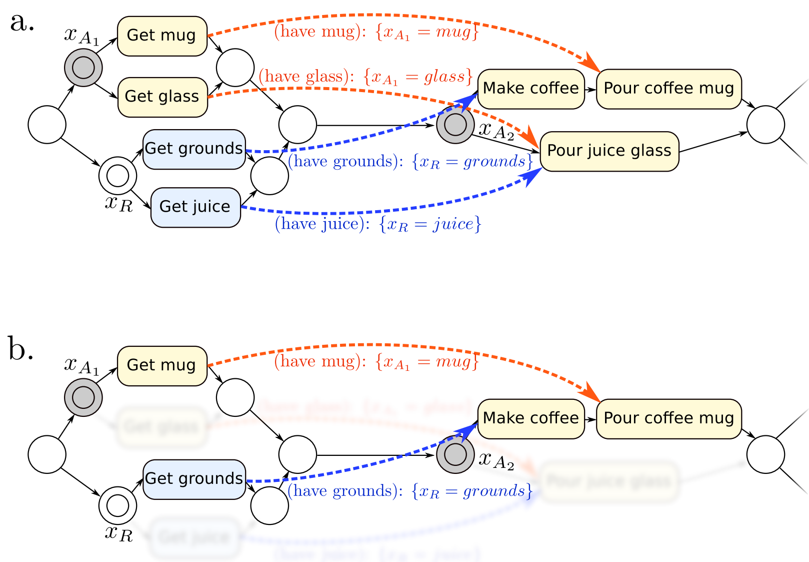


Figure 2: Kitchen example. Part (a) shows the plan annotated with labeled causal links. Part (b) shows the resulting execution if Alice chooses $x_{A_1} = mug$.

glass), while the right half depicts either making a bagel with cream cheese or getting some cereal and milk. Alice is running late for work, so she imposes an overall temporal constraint that both her food and drink must be ready within 7 minutes.

Consider just the first half of the plan, in which the team prepares a beverage — either coffee or juice. There are three decisions — at first, Alice chooses to either get a mug or a glass for her beverage. Shortly thereafter and in parallel, the robot chooses to either fetch the coffee grounds or juice from the refrigerator. Finally once all the necessary supplies have been retrieved, Alice will either make coffee and pour it into her mug, or pour the juice into her glass, depending on her preferred beverage this morning.

Key to our approach is observing that these three choices — two uncontrollable made by Alice and one controllable made by the robot — are not independent. Rather, they are tightly coupled through state constraints in this example, and more generally through temporal constraints as well. For example, if the robot chooses to get the juice out of the refrigerator, Alice will not be able to make coffee as her second choice since she will not have the coffee grounds. Such interrelationships are illustrated in Figure 2a, where we have defined three variables, x_{A_1}, x_R, and x_{A_2}, to represent the different choices. We have also annotated the plan with *labeled causal links*, denoted by dotted arcs and labeled with the environment under which they hold. These labeled causal links capture state requirements in the form of actions preconditions, and their environments imply constraints over feasible choice assignments.

Consider the execution of the example plan depicted in Figure 2b, in which Alice chooses to get the mug instead of the glass for her first choice (i.e., $x_{A_1} = mug$). Alice will therefore have a mug (necessary for pouring coffee into the mug in her second choice), but will not have a glass (necessary for pouring juice into the glass). The top-most labeled causal link, (have mug) contingent upon $x_{A_1} = mug$, will hold, yet the labeled causal link (have glass) contingent upon $x_{A_1} = glass$ will not hold. Thus, Alice cannot choose $x_{A_2} = juice$, since the pour juice action would have an unsatisfied precondition. In this way, the robot infers that Alice – a rational agent – must have the intent of making coffee (i.e., $x_{A_2} =$ *coffee*) since she picked up a mug and not a glass. Our approach therefore makes the assumption that human agents are rational, and will not knowingly make inconsistent choices.

Similar reasoning allows the robot to adapt to Alice's intent. Given that Alice is making coffee, she will require the coffee grounds — illustrated by the causal link (have grounds) contingent upon $x_R = grounds$. Since this is the only labeled causal link supporting said precondition, the robot infers that it must choose $x_R = grounds$. Thus, the robot adapts by getting the grounds instead of the juice, resulting in the final execution depicted in Figure 2b.

If we step back and consider the larger plan shown in Figure 1, we note that Alice cannot both make coffee and make a bagel because she would run out of time. The minimum time required to make coffee and toast a bagel is more than 7 minutes. Thus, if Alice chooses to grab a mug at the beginning, the robot infers that her intent must be not only to make coffee, but also to choose the less time-consuming cereal option so she will arrive at work on time. By similar causal link analysis, the robot will adapt by getting milk for her cereal instead of cream cheese for a bagel.

Consider one final case, in which Alice at first chooses $x_{A_1} = glass$. She will have enough time later for either a bagel or cereal. However, suppose while pouring her juice, an unexpected disturbance occurs. Alice's cat (slinking along the kitchen countertop) accidentally bumps the toaster oven, causing it to fall on the floor and break. The robot's sensors observe this disturbance, and the robot infers that a labeled causal link justifying the precondition of Alice toasting a bagel has been dynamically violated at run time (a working toaster would of course be required). Alice's only option, therefore, is to make cereal. The robot detects this unanticipated change in world state, instantaneously infers Alice's refined intent, and adapts accordingly.

We have just illustrated how a single algorithm, based on constraint satisfaction, concurrently achieves plan recognition and adaptation given state and temporal constraints as well as disturbances. We note that in this specific example, Alice's intent and the robot's adaptations were completely determined after she picked up the mug. This is not generally the case however — often, there may still be multiple consistent options for future choices after constraint propagation (though fewer than before). In these cases, further decisions, either by human or robot, are necessary to hone in on a single intent and adaptation.

1.2 Related Work

There is a rich literature on techniques for both intent recognition and robot adaptation, but largely as separate research activities. To the author's knowledge, there are only several other approaches that simultaneously perform explicit intent recognition and robotic adaptation from a single, core model – a key aspect of our work. The first is described by Freedman and Zilberstein (2017). This work builds upon work in compiling the probabilistic plan recognition task into a classical planning problem (Ramírez & Geffner, 2010), allowing greater foresight during recognition by using a dynamic prior. A probability distribution over different possible PDDL goals is computed, and subsequently merged into a single estimated goal state that is used to plan a response for the robot with an off-the-shelf generative classical planner. This process repeats online, resulting in a closed-loop collaborative execution that excels in its flexibility to respond to many different scenarios. Like our approach, the work of Freedman and Zilberstein uses a single model for both recognition and adaptation – in their case, a PDDL model. In many ways this is a complementary approach to ours, with a focus on achieving great flexibility by employing classical planning techniques. Comparatively, Pike focuses on executing contingent team plans with temporal constraints, achieving fast online reactivity, and monitoring the execution to predict future problems.

A second approach integrating intent recognition explicitly with robot adaptation is SAM (Pecora, Cirillo, Dell'Osa, Ullberg, & Saffiotti, 2012). Like our approach, SAM reasons about temporally flexible constraints and action preconditions (through a generalization known as synchronizations). SAM has been very successfully applied to a smart home scenario in which the autonomous system watches an elderly individual perform daily routines, and then automatically aids him or her or provides suggestions. While similar in the spirit of performing simultaneous intent recognition and adaptation, our approach is fundamentally different (and in many ways complementary). To adapt to recognized intentions, SAM employs a form of online, generative temporal planning. We instead focus on having a contingent, shared task available beforehand. Generative planning approaches are well suited to household robotics domains and can enable great flexibility, whereas having a shared contingent temporal plan is particularly well-suited to other scenarios where considering risk, deadends, low online latency, and predictability are crucial.

A third approach integrating plan recognition with adaptation is described by Geib, Weerasinghe, Matskevich, Kantharaju, Craenen, and Petrick (2016). The system developed uses plan recognition to recognize the goals of a human. The system subsequently engages in a dialog to confirm those goals, and agree upon an independent subtask for the autonomous agent to accomplish. A plan for this subtask is automatically generated and executed. A shared model is used for both plan recognition and the planning tasks. While able to achieve significant flexibility due to its generative planning, this approach is restricted to work in domains in which the robotic agent's subtasks are independent of the human. Comparatively, Pike allows tightly-coupled human robot interaction.

To the author's knowledge, most other approaches dealing with either intent recognition or robot adaptation do so as separate processes, without a single core model guiding both simultaneously. We will now discuss some of these approaches, beginning with techniques for robot adaptation.

Causal links have been used in a number of AI systems, ranging from partial order planners to execution monitoring systems (Mcallester & Rosenblitt, 1991; Penberthy & Weld, 1992; Veloso, Pollack, & Cox, 1998; Lemai & Ingrand, 2004; Levine, 2012). A causal link from activity A_P to activity A_c with predicate p denotes that activity A_P (the *producer*) achieves p as an effect, which is required as a precondition of a later-occurring activity A_c (the *consumer*) (Russell & Norvig, 1995). A_P must precede A_c in the plan, and there may be no other activity A_T (a *threat*) that negates p between A_P and A_c. Historically, causal links were traditionally used as open conditions during the planning process (Mcallester & Rosenblitt, 1991; Penberthy & Weld, 1992). When used for execution monitoring, causal links are a form of goal monitoring in which only the relevant conditions with respect to execution are monitored (Veloso et al., 1998; Levine, 2012). This monitoring allows an autonomous system to detect upcoming plan failure before it is imminent, thereby enabling proactive adaptation through replanning. In this work, we extend causal links to apply to contingent temporally-flexible plans, resulting in *labeled causal links*. Because they are crucial to reasoning about plan causal completeness, a key part of Pike's offline compilation stage is to extract these labeled causal links from the plan. They are later used to reason over possible choice outcomes online.

A number of adaptation techniques recover based on violated state constraints. Many such systems have focused on integrated planning and execution, such as IxTeT-eXeC (Lemai & Ingrand, 2004), ROGUE (Haigh & Veloso, 1998), IPEM (Ambros-Ingerson & Steel, 1988), and HOTRiDE (Ayan, Kuter, Yaman, & Goldman, 2007). Some focus on planning at reactive time scales or continuously online (Finzi, Ingrand, & Muscettola, 2004; Chien, Knight, Stechert, Sherwood, & Rabideau, 2000). The Human-Aware Task Planner (HATP), combined with SHARY, executes human-robot tasks and replans as needed (Alili, Warnier, Ali, & Alami, 2009; Clodic, Cao, Alili, Montreuil, Alami, & Chatila, 2008). The TpopExec system generalizes plans and at every time point will try to execute the first action of a subplan consistent with the current state, thus minimizing the need for replanning (Muise, Beck, & McIlraith, 2013). Both the works of Ingrand and Ghallab (2017) and Meneguzzi and de Silva (2015) offer valuable surveys that discuss the integration of planning with execution in support of robotic adaptation. Like our approach, these works all focus on techniques allowing a robot to adapt to disturbances or situations in its environment. However, these systems generally take a different strategy by focusing on planning, execution with plan repair, and/or responding to disturbances that are not explicitly modeled as human intent. We differ in that we reason explicitly over the possible choices a human is likely to make, and adapt based on this single model.

Beginning with temporally-flexible plans such as Simple Temporal Networks (STNs), efficient dispatchers have been developed that perform fast, online, least-commitment scheduling (Dechter, Meiri, & Pearl, 1991; Muscettola, Morris, & Tsamardinos, 1998; Tsamardinos, Muscettola, & Morris, 1998). Later approaches introduced uncertainty and uncontrollability into these models, resulting in executives that could adapt to many different types of temporal disturbances (Effinger, Williams, Kelly, & Sheehy, 2009; Vidal, 1999). These systems focus on adapting to temporal rather than state constraints, and do not recognize human plans. We do, however, build on the frameworks of some of these approaches. The underlying structure of our temporal plans contains set-bounded temporal constraints like these approaches, and our dispatching algorithms trace their heritage to STN dispatchers (Muscettola et al., 1998).

The Kirk executive (Kim et al., 2001) extends the notion of temporal plans to also incorporate choice, in the temporal planning network (TPN) formalism. By incorporating discrete choices, a plan now represents a set of possible subplans. The TPN is identical to the TPNU formalism used in this work, except that no distinction is made between controllable versus uncontrollable choices. Kirk is designed to assign choices optimally offline, ensuring that the resulting subplan is temporally consistent. A temporal plan dispatcher is then used for online execution. TPNs can be specified in the RMPL language (Williams et al., 2003), allowing a human operator to easily encode flexible plans with choice. The TPNU formalism used in this work derives directly from the TPNs introduced with Kirk. We experimentally compare our work against Kirk in Section 6.

Drake is an executive capable of executing temporally flexible plans with choice (such as TPNUs), scheduling and making choices online with low latency and using minimal space overhead (Conrad, Shah, & Williams, 2009; Conrad & Williams, 2011). It responds dynamically to temporal disturbances, both in terms of scheduling and also in terms of making discrete choices that are guaranteed to be feasible given those temporal disturbances. This is accomplished by fusing ideas from temporal plan execution and from the ATMS (Forbus, 1993) in order to compactly maintain the set of feasible subplans online. In this way, Drake can be seen as solving a similar problem to Kirk, but with a different approach. Instead of selecting a single optimal candidate subplan offline, Drake instead compactly maintains the set of all feasible candidate subplans and makes choices dynamically online. Pike is heavily inspired by Drake, and in fact we borrow many of its techniques and algorithms in this work. Specifically, our online execution algorithm is very similar to that of Drake, as are the temporal reasoning algorithms that we use (albeit with some minor enhancements). Pike differs from Drake as follows: (1) Pike additionally performs causal link analysis to reason about the preconditions of activities in the plan, (2) we transform these causal links into propositional constraints that are encoded alongside temporal conflicts into a knowledge base for fast online decision making, (3) we use a different constraint compilation technique that builds upon the ATMS, and (4) we view all of these technologies from the perspective of human-robot teamwork.

The Chaski executive performs fast online task reallocation for human-robot teams with temporal constraints (Shah, Conrad, & Williams, 2009). Chaski is capable of inferring if an agent – human or otherwise – is stuck, and will re-allocate tasks for the other agents appropriately to maintain plan consistency. Chaski focuses on the dynamic task assignment problem, while we take a different approach and focus instead on plan recognition and adaptation by reasoning over explicit choices in the plan. Chaski introduces two very useful concepts characterizing human-robot team work: that of "Equal Partners" and "Leader and Assistant." Pike follows the "Equal Partners" model.

Many approaches to plan, intent, and goal recognition have been proposed (Sukthankar et al., 2014; Carberry, 2001; Kautz & Allen, 1986; Bui, 2003; Avrahami-Zilberbrand, Kaminka, & Zarosim, 2005; Goldman, Geib, & Miller, 1999; Ramírez & Geffner, 2010; Pattison & Long, 2010). Many of these approaches, however, perform intent recognition without directly interacting with the human. That is, they do not attempt to interact with the human whose intent is being recognized. Our approach differs in that we perform execution concurrently and interleaved with intent recognition. An important approach that does interact with the human is the problem of sequential plan recognition (SPR), in which a set of hypotheses (corresponding to different possible recognized plans a human agent may be following) is incrementally refined by asking the human intelligent questions, such as ones designed to optimize information gain (Mirsky, Stern, Gal, & Kalech, 2016). SPR is complementary in many ways to Pike, and the two approaches could even work well together as follows: SPR could select targeted questions to ask the human, allowing the human's intent to be further refined in support of selecting the robot's adaptations. Another interesting problem for correcting differences between the world models of humans and a robot is that of generating conformant or conditional explanations for model reconciliation (Sreedharan, Chakraborti, & Kambhampati, 2018). In this task, a robot chooses explanations to explain itself and try to correct the human's mental model, even in the face of uncertainty about this model. One final interesting related problem that is complementary to this work is that of goal recognition design (Keren, Gal, & Karpas, 2014). In this problem, the task is to modify the environment or model in which agents operate, so these agents reveal their intentions earlier rather than later. We believe that goal recognition design could be very successfully applied to Pike's setting. Such techniques could be used to improve the recognition task for environments in which the Pike operates, in support of improving execution robustness.

1.3 Contributions

The main contributions of this work are:

1. Novel framework for concurrent intent recognition and robot adaptation for contingent temporally-flexible plans,

2. Generalization of causal links for contingent temporally-flexible plans, and techniques to extract them from said plans and compile them into propositional constraints,

3. An online, state-of-the art dynamic execution system that employs these causal link constraints to make online decisions.

This work is an extension of the conference paper (Levine & Williams, 2014), and makes the following additions: (1) algorithmic improvements and an updated constraint encoding, allowing Pike to handle potential threats and unordered producers to causal links (thus improving robustness), (2) greatly expanded discussion, theory, and proofs, and (3) much broader experimental validation.

1.4 Organization

This paper is organized as follows. Section 2 introduces some preliminaries and defines Pike's problem statement. We then dive into the online execution algorithm in Section 3. Next, we discuss the offline compilation that makes this online execution possible; Section 4 discusses labeled causal link extraction and a constraint transformation needed for execution. Section 5 describes a knowledge compilation approach that enables fast online querying of the constraints. Empirical evaluations are presented in Section 6. Finally in Section 7, we discuss related work and concluding remarks. An appendix presents additional algorithms and proofs of various theorems in this work.

2. Problem Statement & Solution Architecture

In this section, we define Pike's problem statement — both offline and online. We begin with some preliminaries, present the problem statement, and conclude with an outline of our solution architecture.

2.1 Preliminaries

Pike takes as input a set of possible team plans to be performed, which are represented by a Temporal Plan Network under Uncertainty (TPNU). Importantly, these plans involve activities performed both by the robot and by the human, activities that are performed concurrently, and constraints on the timing of these activities. The TPNU has its roots in the Temporal Plan Network (TPN) (Kim et al., 2001), a representation for concurrent threads of execution in a timed, decision-theoretic programming language. In the literature, a TPNU contains two types of uncontrollability: uncontrollable temporal durations (which may be either set-bounded or probabilistic in nature) (Yu, Fang, & Williams, 2014), or uncontrollability in the outcome of discrete choices (Effinger et al., 2009; Santana & Williams, 2014). We focus our attention solely on uncontrollable choices in this work, though work exists extending it to handle uncertain temporal durations via strong controllability (Karpas, Levine, Yu, & Williams, 2015). The underlying temporal structure of these representations takes inspiration from the Simple Temporal Network (Dechter et al., 1991).

An example TPNU is depicted in Figure 1. In this picture, circles denote *events*, each of which represents an instantaneous point in time. Examples of events include "the time at which the robot starts picking up the coffee" or "the time at which the human completed making coffee." Edges in the diagram represent temporal constraints. Colored boxes represent activities.

Definition 2.1 (TPNU)**.** A Temporal Plan Network under Uncertainty (TPNU) $\mathcal{T}$ is a tuple $\langle \mathcal{V}, \mathcal{E}, \mathcal{C}, \mathcal{A} \rangle$ where:

- $\mathcal{V}$ is a set of choice variables, which is partitioned into two groups: $\mathcal{V} = \mathcal{V}_C \cup \mathcal{V}_U$. Each $v \in \mathcal{V}$ is a discrete variable with a finite domain $\textsc{Domain}(v)$. $\mathcal{V}_C$ are *controllable* choices that are decidable by the executive at run time. $\mathcal{V}_U$ are *uncontrollable* choice variables that are not decidable by the executive, but rather by the human or nature.

- $\mathcal{E}$ is a set of events representing notable points in time. Each event $e \in \mathcal{E}$ is associated with a guard (denoted φ_e or $guard(e)$), which is a conjunction of choice variable assignments. An event e is *activated* if φ_e holds. Event e will be *executed* (i.e., a time scheduled for it) by the executive iff it is activated. Additionally, certain events are associated with a choice variable, denoted $v_i = \textit{variable-at-event}(e)$, denoting that v_i must be assigned by the time e is executed.

- $\mathcal{C}$ is a set of temporal constraints. Each $c \in \mathcal{C}$ is a tuple $\langle e_s, e_f, l, u, \varphi_c \rangle$ where e_s is the *from* or *start* event, e_f is the *to* or *finish* event, φ_c is a conjunction of choice variable assignments, and $l, u \in \mathbb{R}$ represent a temporal bound with the meaning that $\varphi_c \Rightarrow (l \leq e_f - e_s \leq u)$. In other words, the temporal constraint must hold if the guard holds (it is *activated*). We require that $\varphi_c \models \varphi_{e_s} \wedge \varphi_{e_f}$, so that whenever a temporal constraint is activated its *from* and *to* events must be executed. We denote a temporal constraint $c \in \mathcal{C}$ sometimes as $e_s \xrightarrow{[l,u]} e_f : \varphi_c$.

- The set $\mathcal{A}$ represents the set of *activities*. An activity $a \in \mathcal{A}$ is a tuple $\langle c, \alpha \rangle$, where $c \in \mathcal{C}$ is a temporal constraint, and α is an action that will be executed online. With $c = \langle e_s, e_f, l, u, \varphi_c \rangle$, action α starts when e_s is scheduled, and terminates when e_f is scheduled. We require that $l > 0$.

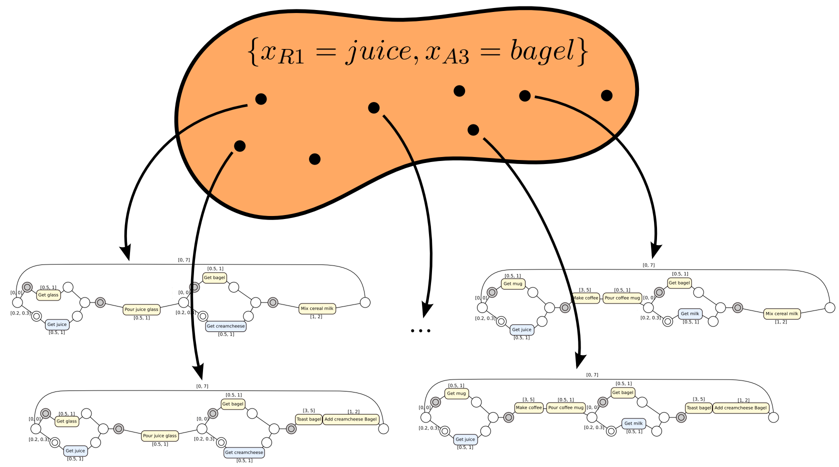


Figure 3: A TPNU is a compact encoding of many different candidate subplans, and an environment represents a subset of those candidate subplans. Shown are various candidate subplans of the TPNU from Figure 1 (note the different choices visible from their structure).

The human-robot plans involve actions targeted at both the human and the robot that follow an action model. These actions are denoted by the α field of activities in the TPNU. An *action* α may be anything that an agent can perform. Each action has predicates called *conditions* that are required to all hold true (either at its start or end of execution), and other predicates called effects that are asserted by the action and represent the changes to the world as a result of executing the action (again, either at its start or end of execution). In this way, we employ similar semantics to durative actions in PDDL 2.1 (Fox & Long, 2003), though actions could also be represented in other forms such as STRIPS or RMPL operators (Fikes & Nilsson, 1971; Williams et al., 2003). For the remainder of this paper, we treat the start event e_s and end event e_f of an activity as snap actions, each with their own preconditions and effects, denoted Preconditions(e) and Effects(e), respectively.

Central to the notion of Pike are *team scenarios / candidate subplans* and *environments*.

Definition 2.2 (Team Scenario)**.** A *team scenario* φ_S is a full assignment $x_i = v_i \wedge x_j = v_j \wedge \ldots$ to all choice variables in $\mathcal{V}$.

Definition 2.3 (Environment)**.** An *environment* φ is a partial assignment $x_i = v_i \wedge x_j = v_j \wedge \ldots$ to choice variables in $\mathcal{V}$. That is, some choice variables in $\mathcal{V}$ may not be assigned.

Throughout this paper, we sometimes denote team scenarios and environments using set notation for convenience, such as $\{x_i = v_i,\ x_j = v_j,\ \ldots\}$. The empty environment, $\{\}$, is logically equivalent to True.

Intuitively, a team scenario represents a specific *candidate subplan* of the TPNU. Given an assignment to all choice variables, we can evaluate which guard conditions hold, and hence which temporal constraints and events in TPNU are activated. All other inactivated events and temporal constraints can be discarded. A team scenario hence defines a specific *candidate subplan*, which encompasses a set of events, activities, and their associated flexible temporal constraints. For this reason, in this paper we often denote a specific candidate subplan simply by φ_S, its team scenario full assignment to choice variables. Using the terms interchangeably in this manner is convenient so that we may refer to a candidate subplan φ_S satisfying a set of constraints (we mean that φ_S is a solution to those constraints). The underlying network structure of a candidate subplan is that of a Simple Temporal Network (STN) (Dechter et al., 1991), though it is distinct from an STN by virtue of having activities, each with preconditions and effects.

An environment that is not a team scenario (i.e., a partial assignment to variables), represents a set of candidate subplans. Environment φ represents all candidate subplans φ_S for which $\varphi_S \models \varphi$. Thus, we use environments to compactly reason over large sets of candidate subplans, as illustrated in Figure 3.

Definition 2.4 (Environment Representing Set of Team Scenarios)**.** An environment φ *represents* the set of all team scenarios $\mathcal{S}(\varphi)$, where $\varphi_S \in \mathcal{S}(\varphi)$ iff $\varphi_S \models \varphi$.

For example, suppose we have three discrete variables x, y, and z, each with domain $\{1, 2\}$. The empty environment $\{\}$ is the most general, representing all eight possible scenarios. The environment $\{x = 1\}$ represents the four scenarios that assign $x = 1$. The environment $\{x = 1, y = 2, z = 1\}$, which is itself a scenario, is the most specific (representing just itself).

The conjunction of two environments $\varphi_a \wedge \varphi_b$ represents the intersection of these two sets $\mathcal{S}(\varphi_a) \cap \mathcal{S}(\varphi_b)$, and the disjunction $\varphi_a \vee \varphi_b$ represents their union $\mathcal{S}(\varphi_a) \cup \mathcal{S}(\varphi_b)$. We say that an environment $\varphi_a \wedge \varphi_b$ is *self-consistent* if it does not contain conflicting assignments to the same variable (e.g., $\{x = 1, x = 2\}$ is not self-consistent). Generally, as the number of assignments in an environment increases, it gets more specific and represents fewer team scenarios.

Given a candidate subplan φ_S, which contains a set of activated events and temporal constraints, we can consider the scheduling problem of trying to find a time assignment to every event that satisfies those temporal constraints.

Definition 2.5 (Schedule, Temporal Consistency)**.** A *schedule* T_{φ_S} for a candidate subplan φ_S assigns a time value to each event in $\mathcal{E}$ activated by φ_S. We denote the assigned time for a specific event e_i by $T_{\varphi_S}(e_i)$. A schedule is *temporally consistent* iff it satisfies all of the temporal constraints of the candidate subplan (i.e., those $c \in \mathcal{C}$ activated by φ_S). A candidate subplan is *temporally consistent* iff it has at least one temporally consistent schedule.

It is also useful to think in terms of *executions*, which represent not only the schedule but also the choices made. We can then consider not only temporal consistency, but also another very important concept for plan execution known as *causal completeness.* Intuitively, causal completeness says that the preconditions of all activities in the plan are expected to be satisfied by the time those activities are executed.

Definition 2.6 (Execution)**.** An *execution* is a tuple $\langle \varphi_S, T_{\varphi_S} \rangle$ where φ_S is a team scenario (i.e., representing a candidate subplan) and T_{φ_S} is a schedule for that candidate subplan. An execution is *temporally consistent* iff T_{φ_S} satisfies all of the temporal constraints of φ_S. An execution is *causally complete* iff the precondition of every event activated by φ_S is satisfied at the time of its execution in T_{φ_S}, assuming no disturbances. An execution is *correct* iff it is both temporally consistent and causally complete.

The related concept of a *partial execution* represents the state in the midst of execution – some choices have been made so far, and some events have been executed – but not necessarily the entire plan.

Definition 2.7 (Partial Execution)**.** A *partial execution* is a tuple $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ where φ_{ex} is a partial assignment to choice variables in $\mathcal{V}$ (i.e., an environment), and $\tilde{T}_{\varphi_{ex}}$ is a partial schedule that assigns time values to a subset of the events in $\mathcal{E}$. A partial execution $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ can be *extended* to execution $\langle \varphi_S, T_{\varphi_S} \rangle$ iff $\varphi_S \models \varphi_{ex}$ and $T_{\varphi_S}(e_i) = \tilde{T}_{\varphi_{ex}}(e_i)$ for all e_i assigned by $\tilde{T}_{\varphi_{ex}}$.

We can also describe a partial execution as correct, if additional choices and scheduling decisions can be made that would extend it to a correct execution:

Definition 2.8 (Correct Partial Execution)**.** A partial execution $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ is *temporally consistent* iff there exists an extending execution $\langle \varphi_S, T_{\varphi_S} \rangle$ that is temporally consistent. A partial execution $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ is *causally complete* iff there exists an extending execution $\langle \varphi_S, T_{\varphi_S} \rangle$ that is causally complete. Finally, a partial execution $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ is *correct* iff there exists an extending execution $\langle \varphi_S, T_{\varphi_S} \rangle$ that is correct, i.e. both temporally consistent and causally complete.

Execution begins with the empty partial execution, where $\varphi_{ex} = \{\}$ (i.e., no choices made yet) and $\tilde{T}_{\varphi_{ex}}$ makes no time assignments. As execution proceeds, Pike will make choices that only result in correct partial executions.

So far, we have not distinguished between choices made by robotic agents versus choices made by human or environmental agents. We therefore further define two related concepts: the *intent* and the *adaptation*. An intent is a set of assignments to the uncontrollable variables made by the human; we think of this as the true intentions of what the human plans to do. An adaptation is similarly an assignment to the controllable variables, and represents a way that the robot may react to the human's intentions.

Definition 2.9 (Intent / Adaptation)**.** An *intent* is an environment consisting solely of assignments to uncontrollable variables in $\mathcal{V}_U$. An *adaptation* is an environment consisting solely of assignments to controllable variables in $\mathcal{V}_C$. Furthermore, an *intent scenario* is an intent that assigns a value to every variable in $\mathcal{V}_U$, and an *adaptation scenario* is an adaptation that assigns a value to every controllable variable $\mathcal{V}_C$.

Based on these definitions, we see that an intent scenario and an adaptation scenario jointly define all choices for the team.

Observation. *Suppose φ_I is an intent scenario, and φ_A is an adaptation scenario. Then $\varphi_I \wedge \varphi_A$ assigns all variables in $\mathcal{V}$, and is hence a team scenario.*

2.1.1 Labeled Value Sets

The *labeled value set* (LVS) is a data introduced in Drake's labeled temporal reasoning algorithms (Conrad & Williams, 2011) for compactly recording the tightest possible constraint over many team scenarios.

Suppose we have a variable t, and we have deduced that $t < 2$ universally holds. Later on, suppose we deduce the additional constraint that when the finite-domain variable $x = 1$, the constraint $t < 1$ holds. This is a tighter constraint, so if we have some information about the world (namely that $x = 1$), we can use the tighter constraint $t < 1$ in our subsequent reasoning. Writing this as a table, we would have two entries; the first would be that $t < 2$ holds in all circumstances, and the second would be that $t < 1$ holds in the specific circumstance where $x = 1$. Suppose we also deduce that $t < 5$ whenever $x = 2$. This information is redundant, as we already have deduced that the tighter constraint $t < 2$ always holds. It is *dominated* by the other constraints we already have discovered.

Such is the intuition behind the labeled value set, which encodes the tightest value for a constraint as a function of choice. The LVS stays compact using two strategies: (1) not keeping track of unnecessary relations that are dominated by others already deduced, and (2) using environments to compactly represent values over many different scenarios. We use the LVS for several different purposes within Pike.

Labeled value sets operate with respect to a *relation* $<_R$, which provides a total ordering over elements. We use the standard numeric $<$ relation to compare numbers in its temporal reasoning algorithms. However, we additionally use a different relation (defined later) to compare TPNU events chronologically, given temporal flexibility. Therefore, we keep the following discussion general with the denoted relation $<_R$. Along with this relation, we also have related quantities ∞_R, $-\infty_R$ (representing the maximal and minimal possible elements of the relation), and operations $\leq_R$, $\max_R$, and $\min_R$.

Definition 2.10 (Labeled Value Pair)**.** A *labeled value pair* is a tuple (a_l, φ_l), where a_l is some value and φ_l is an environment.

A labeled value pair represents that a constraint value of a_l holds whenever φ_l holds. For example, with relation $<$ over a variable t, the labeled value (a_l, φ_l) means that $\varphi_l \Rightarrow t < a_l$.

It is unnecessary to store labeled value pairs corresponding to any constraints implied by other constraints. This is the notion of *dominance*.

Definition 2.11 (Dominance)**.** A labeled value (a_d, φ_d) *dominates* a weaker labeled value (a_w, φ_w), iff 1.) $a_d <_R a_w$, and 2.) $\varphi_w \models \varphi_d$.

Again taking $<$ as our relation, $(1, \{x = 1\})$ dominates $(2, \{x = 1, y = 2\})$. Whenever $\{x = 1, y = 2\}$ holds and the constraint $t < 2$ is active, then $x = 1$ must also hold and the tighter constraint $t < 1$ is also active. Thus, since the dominating pair both applies more generally and is tighter, the weaker, dominated pair may be discarded.

Definition 2.12 (Labeled Value Set)**.** A *labeled value set* $L = \{(a_1, \varphi_1), (a_2, \varphi_2), \ldots\}$ is a set of labeled value pairs, such that no pair in the set dominates any other pair.

Since no labeled value dominates any other, the LVS remains attempts to remain compact by omitting superfluous information. In some circumstances however, an LVS may grow quite large – there could be at worst a labeled value pair for every possible environment, resulting in an LVS with exponential size. In practice, we find that LVSs typically stay much more compact than this worst-case guarantee and that the non-dominating property is effective at significantly reducing the number of labeled value pairs.

The AddLVS(p, L) method is responsible for adding a new labeled value pair p minimally to LVS L and ensuring the non-dominance invariant. It operates as follows. If p

is dominated by any other pair $p_i \in L$, then p is not added and AddLVS(p, L) returns False. Otherwise, p is added to L, any other $p_i \in L$ dominated by p is removed, and the method returns True. We also define a related method, MergeLVS(L, L_{add}) that adds every labeled value pair of L_{add} minimally to L using AddLVS.

Perhaps the most useful operation on an LVS L is *querying* it, which computes the tightest possible constraint value that holds over all scenarios in a given environment. It answers the question: given L, what is the tightest bound that holds over all $\varphi_s \in \mathcal{S}(\varphi)$? In other words, what is the smallest value a that can be guaranteed for the constraint $t <_R a$? We denote the query operation over L under an environment φ is as $Q_L(\varphi)$. To compute $Q_L(\varphi)$, we assemble a set P containing all labeled values $(a_i, \varphi_i) \in L$ such that $\varphi \models \varphi_i$. If P is not empty, then we return $\min_R\{a_i \mid (a_i, \varphi_i) \in P\}$. Otherwise, we return ∞_R.

We can additionally perform certain binary operations on LVSs, such as addition and subtraction. This is quite useful as a "labeled generalization" of normal addition and subtraction, allowing for an elegant formulation of various algorithms such as Floyd Warshall (Conrad & Williams, 2011). To perform a binary operation on two LVSs, every possible combination of their labeled values is considered and the binary operation (ex. addition) is applied to the values. The conjunction of their environments is also taken, forming a new candidate labeled value for the result. Only the dominating labeled values of this result are kept (Conrad & Williams, 2011). For example, suppose we wish to add $L_1 = \{(2, \{x = 1\}), (3, \{y = 2\}), (4, \{\})\}$ to $L_2 = \{(1, \{x = 2\}), (3, \{\})\}$. The resulting LVS contains the sum, taking into account each possible combination of environments but pruning out dominated labeled values: $\{(4, \{x = 2, y = 2\}), (5, \{x = 1\}), (5, \{x = 2\}), (6, \{y = 2\}), (7, \{\})\}$.

2.2 Inputs and Outputs

In order to prepare for execution, Pike takes in the following inputs offline:

- A TPNU $\mathcal{T}$ representing a contingent, temporally-flexible human-robot plan.
- An action model specifying conditions and effects of actions in the plan.
- The initial state of the world. Specified as a set of state predicates.
- The desired goal state of the world. Specified as a set of state predicates.

Online during execution, Pike additionally takes in the following inputs:

- A stream of alerts $\mathcal{A}(t)$ for when activities terminate (assumed to be within durations).
- A stream of uncontrollable choice outcomes $\mathcal{U}(t)$ as they occur.
- A stream of predicates $\mathcal{P}(t)$ describing current state estimates.

Pike outputs the following during online execution:

- A stream of dispatched actions, targeted at both the robot and human, at temporally consistent times.
- A stream of controllable choice assignments that maintain a correct partial execution.

- A flag detecting future plan failure, as soon as it is detected.

We adopt similar temporal controllability semantics as Drake; namely, we treat each activity as begin controllable (Vidal, 2000), meaning that our executive controls the start times of activities but may not arbitrarily and instantaneously choose their end times. Rather, these times are determined by nature or by the human, and the executive will be told online when each activity ends immediately afterward. We do not address the issue of temporal controllability further in this paper (i.e., dynamic controllability), but note that this is an important field and that notable work has been done in characterizing it for conditional plans (Hunsberger, Posenato, & Combi, 2012).

When execution begins, the partial execution is empty – no choices have been made and no events have been scheduled. As execution proceeds, choices and scheduling decisions are made. Given the current partial execution $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$, if Pike chooses to execute event e_i at time t, we would have the resulting partial execution with more choice variables assigned and events scheduled: $\langle \varphi_{ex} \wedge \varphi_{e_i}, \tilde{T}_{\varphi_{ex}} \cup \{e_i = t\} \rangle$. Pike is permitted to execute e_i at time t (and hence make associated controllable choices) if and only if this resulting partial execution would be correct. This execution property is key to Pike's robustness, and will be proved later in this work by Theorem 4.7.

We can also view the above in terms of intents and adaptations. Pike may choose any controllable choice (i.e., an adaptation) such that there remains at least one consistent adaptation scenario and one intent scenario for the human consistent with this adaptation scenario and satisfying the plan's preconditions and temporal constraints. We can express this in terms of partial executions as follows. Consider some controllable choice assignments (i.e., an adaptation) represented by environment φ_{A_i}, and an associated event e_i. Pike is permitted to make the choices φ_{A_i} and simultaneously execute event e_i at time t if and only if there exists an adaptation scenario φ_A, an intent scenario φ_I, and a schedule $T_{\varphi_A \wedge \varphi_I}$ where the execution $\langle \varphi_A \wedge \varphi_I, T_{\varphi_A \wedge \varphi_I} \rangle$ is correct and extends the partial execution $\langle \varphi_{ex} \wedge \varphi_{A_i}, \tilde{T}_{\varphi_{ex}} \cup \{e_i = t\} \rangle$. Please note that such semantics allow Pike to restrict the future possibilities for the human, but the end result is a mixed-initiative execution.

By not distinguishing intents from adaptations, we can employ a single framework to check causal completeness and temporal consistency for both kinds of choices – hence concurrently performing intent recognition and adaptation via one mechanism. As long as the partial execution remains correct, there exists some human and robot choices that yield a correct execution. Therefore, for the remainder of this paper, we distinguish between intents and adaptations seldomly and only when required, treating both more generally as assignments to choice variables within the context of ensuring a correct execution.

2.3 Solution Architecture

Given this problem statement, we revisit our solution approach. An algorithmic architecture of Pike is shown in Figure 4, and the rest of this paper describes its various components.

First, offline compilation begins with the Labeled All-pairs Shortest Path (Labeled APSP) algorithm (the upper-left component in Figure 4). This is the core temporal reasoning algorithm in Pike. Given a TPNU as input, it outputs a matrix $D_{i,j}$, where an entry for row i, column j in the matrix is a labeled value set containing important temporal information relating events e_i and e_j of the TPNU. Intuitively, this labeled value set contains

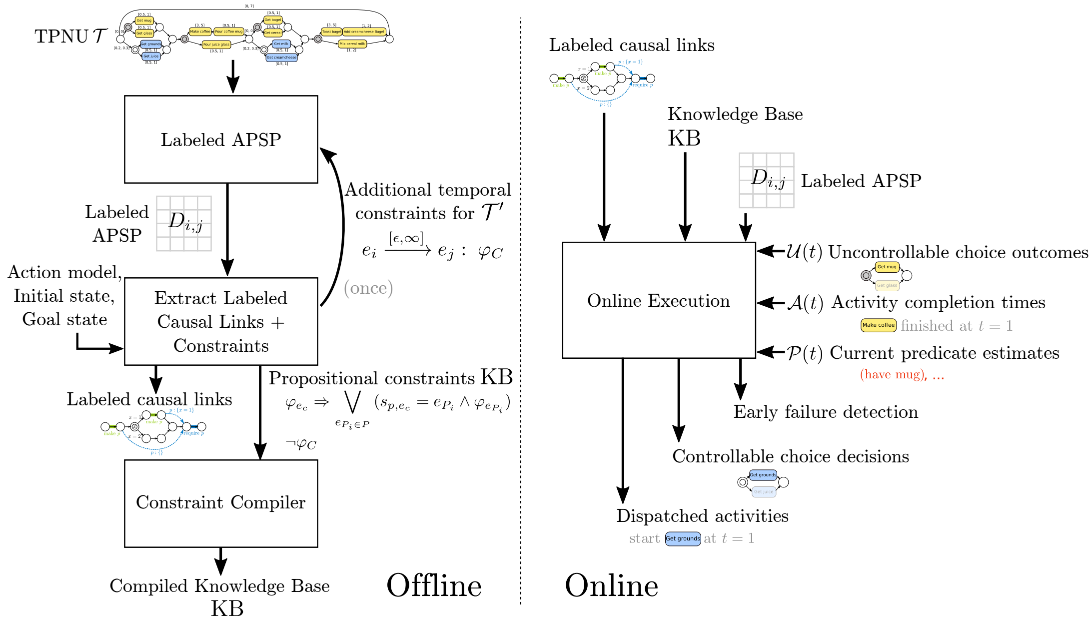


Figure 4: Algorithmic architecture of PIKE. Computation begins with the upper component of the Offline component, and flows downward. After offline compilation, online execution occurs.

the *tightest* possible inferred temporal relationships between events i and j, as a function of the TPNUs choices. PIKE uses this information for three purposes: (1) to schedule events and activities online, making sure that they respect the temporal constraints, (2) to infer ordering relationships between events for the next step (e.g., "e_i always occurs before e_j when $x = 1$ holds"), and (3) to infer that certain combinations of choices will always be temporally infeasible (e.g., "if we choose $x = 1$ and $y = 1$, we are guaranteed to miss a deadline"). The labeled APSP algorithm is described in Section 4.1.

Next, after temporal relationships between events have been computed via the labeled APSP algorithm, causal link reasoning is performed (middle-left component in Figure 4). This process extracts a set of labeled causal links for every precondition of every event in the plan. It makes use of the ordering relations present in $D_{i,j}$ as part of this process. The main output of this process is a set of labeled causal links, as well as a set of propositional constraints KB used for ensuring causal completeness later during online execution. The set of solutions to KB captures the space of all candidate subplans that admit correct executions. Depending on the structure of the TPNU, additional temporal constraints may be added to the TPNU – effectively creating a new *augmented TPNU*. This requires re-computing the labeled APSP a second time due to the newly-added temporal constraints. An updated $D_{i,j}$ is obtained[1]. All of this causal link reasoning is described in Sections 4.2, 4.3, and 4.4.

Finally, we generate a compiled representation of our constraints KB that is suitable for efficient, low-latency use during online execution (lower-left component in Figure 4).

1. We will only need to compute the labeled APSP at most twice.

Specifically, the goal is to generate a representation of KB that allows us to quickly make certain key queries needed for online execution:

- CorrectTeamPlanExists?(KB). Returns True if and only if KB is satisfiable; i.e., if there exists a candidate subplan that admits at least one correct execution.
- CouldCommitToAdditionalConstraints?(KB, $\{F_1, ..., F_n\}$). Returns True if and only if there would exist a candidate subplan admitting a correct execution after adding the additional set of constraints; that is, if KB $\wedge$ $(F_1 \wedge ... \wedge F_n)$ is satisfiable.
- CommitToConstraints(KB, $\{F_1, ..., F_n\}$). Adds the constraints to KB conjunctively; i.e., KB $\leftarrow$ KB $\wedge$ $(F_1 \wedge ... \wedge F_n)$.

Our compilation scheme is implemented by computing the prime implicants of the propositional constraints of constraints KB via a label propagation algorithm we call the πTMS. This compilation is described in Section 5. This completes the offline compilation stage.

During online execution (rightmost component in Figure 4), the TPNU is executed, activities are dispatched to the robot, and interaction with the human and the environment occurs. As its input, the online execution algorithm takes in the outputs generated from offline compilation: (1) $D_{i,j}$ to help decide when to schedule events and dispatch activities, (2), the compiled knowledge base KB to decide what choices in the TPNU can be made to ensure a correct execution, and (3) the set of labeled causal links, to monitor for future problems during execution. Additionally, online execution takes as input certain "live" data as it becomes observed online: (1) the outcomes to uncontrollable choices made by the human $\mathcal{U}(t)$, (2) notifications of when dispatched activities complete $\mathcal{A}(t)$, and (3) an estimate of the world state $\mathcal{P}(t)$ used for execution monitoring. The output of online execution is a live, streaming dispatch of (1) activities targeting the robot, (2) a series of controllable choice decisions for the robot, and (3) early signaling of failure, if it is detected via execution monitoring. The online execution algorithm is described in Section 3.

This completes the architecture and interface description of our approach. As Pike is a plan executive whose primary goal is to execute plans, we dive right into the online execution algorithm next in Section 3. Our presentation defers the details of offline compilation to Sections 4 and 5, where their context and usefulness may be more clear after seeing the details of their use during online execution in Section 3.

3. Online Execution Strategy

The online execution algorithm is responsible for (1) scheduling events and dispatching associated activities at the proper times, (2) assigning values to controllable choices, and (3) monitoring the execution for potential problems. It does so by analyzing choices made by the human, respecting the plan's temporal constraints, and observing the state of the world to detect upcoming failures. Our approach takes a least-commitment approach to execution, leaving as much flexibility as possible to the online executive. This flexibility – in terms of both controllable choices and scheduling decisions – affords a greater degree of robustness than executives operating with grounded plans where such decisions have been already made offline before execution begins. Least-commitment executives such as Pike are able to exploit new information that becomes available online and adapt to it without full replanning.

While rich, least-commitment execution approaches can be very beneficial, they do come at a cost in terms of worst case theoretical complexity:

Theorem 3.1 (Checking for a Correct Execution is NP-Complete)**.** *The problem of checking if there exists a correct execution* $\langle \varphi_S, T_{\varphi_S} \rangle$ *for a TPNU* $\mathcal{T}$ *is NP-complete.*

Proof. See Appendix B. □

Checking for a correct execution is a key operation of Pike that occurs during online execution. Despite this worst case theoretical bound, we argue (and show experimentally in Section 6.2) that least-commitment approaches such as Pike are well worth the cost due to their improved robustness online. Additionally, we employ a number of techniques in this work that evade this worst case bound and endow good performance in many practical, real-world domains.

Our online execution approach takes as input the output of the offline compilation process: (1) a matrix $D_{i,j}$ containing temporal information computed by the labeled APSP, (2) the set of extracted labeled causal links, and (3) a knowledge base KB representing a compiled version of constraints derived from the causal links. Additionally, the online execution algorithm takes as input certain observations that only become available online, including (1) a stream of uncontrollable choice outcomes $\mathcal{U}(t)$ representing the choices made by the human and/or environment, (2) a stream of activity finish notifications $\mathcal{A}(t)$ denoting when dispatched activities finish, and (3) a stream of estimated state predicates $\mathcal{P}(t)$ used for execution monitoring.

Our online execution strategy is heavily inspired by Drake (Conrad, 2010; Conrad & Williams, 2011), and follows the same overall structure. Drake's execution strategy can be seen as a labeled generalization of the STN dispatching algorithm (Muscettola et al., 1998) for plans with choice. It maintains *labeled* generalizations of many datastructures, and uses them to find new propositional constraints that would be required to hold in order to execute events at the current time. Drake uses a compiled version of these constraints to "greedily" schedule events and dispatch associated activities, and to make choices whenever possible such that doing so would not cause temporal inconsistency. Pike follows this same strategy, and borrows many of Drake's temporal reasoning algorithms and labeled datastructures as we shall discuss shortly. Pike augments Drake's execution strategy by additionally considering causal completeness and performing causal-link execution monitoring, two attributes which we argue are very important for human robot collaboration. Pike may choose to schedule any event at any time, such that there would remain at least one correct execution of the TPNU. Despite the apparent greediness of the approach, choices are never made that entail execution failure. As execution proceeds and choices are made by both the human and robot, the space of correct executions becomes successively smaller. The result is a mixed-initiative execution, in which the human and robot make choices together, and possibly constrain each other's future choices (such that at least one correct execution remains).

We describe Pike's execution algorithm here, and will illustrate it shortly with an example. Each event e_i is associated with an *execution window*, which captures the earliest and latest allowable absolute times during which e_i can be executed without violating any temporal constraints. Each execution window is defined by its *lower_bound*(e_i) and *upper_bound*(e_i). In traditional dispatchers for STNs, these lowerbounds and upperbounds for each event are numbers representing absolute times (Muscettola et al., 1998). Whenever an event is executed, its scheduled time is propagated locally to all neighbors, causing a tightening of those events' execution windows. An event may only be executed if the current time is within its execution window. Pike uses a generalization of these execution windows, called labeled execution windows (Conrad, 2010). These encode the dependence of each event's execution time on the choice variables. As different choices are made, different temporal constraints become active or inactive, thus affecting each event's execution window. *lower_bound*(e_i) and *upper_bound*(e_i) are each labeled value sets instead of numbers in this formalism. Specifically, a labeled value (l, φ_l) in *lower_bound*(e_i) means that in all scenarios where φ_l holds, event e_i must be executed at l or later; that is, $\varphi_l \Rightarrow e_i \geq l$. Similarly, a labeled value in (u, φ_u) in *upper_bound*(e_i) means that in all scenarios where φ_u holds, e_i must be executed by u or earlier; that is, $\varphi_u \Rightarrow e_i \leq u$. Note that if φ is the empty environment, we are equivalent to the unlabeled approach for STNs. The LVS relation $<_R$ for *upper_bound*(e_i) is $<$, and for *lower_bound*(e_i) it is $>$, capturing the notion that earlier upperbounds and later lowerbounds are tightest. Before execution begins, we set up an execution window for each event of the plan, initializing each to the broadest range possible. For each e_i, *lower_bound*(e_i) is initialized to the LVS $\{(-\infty, \{\})\}$, and *upper_bound*(e_i) is similarly initialized to $\{(\infty, \{\})\}$. If an event e_i is executed at some time t, then we propagate the execution window of other events e_j as follows using $D_{i,j}$ (computed in Section 4.1) (Conrad, 2010):

$$\begin{aligned}&\textsc{MergeLVS}(\textit{upper\_bound}(e_i), \{(t, \{\})\} + D_{e_i,e_j}) \\ &\textsc{MergeLVS}(\textit{lower\_bound}(e_i), \{(t, \{\})\} - D_{e_j,e_i})\end{aligned} \tag{1}$$

During execution, Pike must check if an event e_i can be executed at the current time t. To do so, we assemble a set of constraints $F_1, ..., F_n$ that would all need to hold. If there exists a team scenario φ_S that satisfies the conjunction of KB with these constraints $\{F_1, ..., F_n\}$, then e_i can safely be executed now (Conrad, 2010). We refer to the procedure that obtains these constraints as GetConstraintsRequiredToExecute(e_i, t). It produces the constraints $F_1, ..., F_n$ using $D_{i,j}$ as follows:

1. The guard of e_i, φ_{e_i}.
2. True if e_i is directly executable, otherwise False. Event e_i is not directly executable if (1) it is the end event of an activity (which is scheduled externally and reported to Pike via $\mathcal{A}(t)$), (2) if φ_{e_i} mentions an assignment to a currently unassigned uncontrollable variable (which Pike is not allowed to assign), or (3) if the variable *variable-at-event*(e_i) is uncontrollable and unassigned. Note that if we add False conjunctively, we effectively prevent e_i from being executed now.
3. $\neg\varphi_l$ for any labeled value (l, φ_l) in *lower_bound*(e_i) if $t < l$.
4. $\neg\varphi_u$ for any labeled value (u, φ_u) in *upper_bound*(e_i) if $t > u$.
5. $\neg\varphi$ for any (w, φ) in D_{e_i,e_j} where e_j is a yet un-executed event, and $w < 0$.

6. $\neg\varphi$ for any (w, φ) in D_{e_i,e_j} where e_j is a yet un-executed event and is also not directly executable, and $w \leq 0$.

We provide some brief intuition about these constraints, but refer the reader to Drake for full details (Conrad, 2010). Constraint 1 ensures that the guard of the event holds, a requirement to execute e_i. Constraint 2 prevents Pike from executing non-directly executable events. These are events that are scheduled externally by the environment (for instance, the end event of activities), events whose guards depend on a yet-unassigned uncontrollable choice variable, or events that are "branching points" where uncontrollable choices must be made. This prevents Pike from effectively choosing the outcome of uncontrollable variables. Constraint 3 handles the case of violated lowerbounds: *lower_bound*(e_i) dictates that e_i must be executed *after* time l whenever φ_l holds. If we instead execute it *beforehand* at $t < l$, then φ_l cannot hold. We therefore add $\neg\varphi_l$ to the set of constraints. Similar reasoning holds for the upperbounds in Constraint 4. Constraint 5 handles the case of violated ordering constraints. If the labeled APSP dictates that event e_j must be executed *before* event e_i whenever φ holds, but we are executing e_i now before the un-executed e_j, then φ cannot hold. Finally, Constraint 6 addresses a similar case where e_j is unexecuted and is not directly executable (e.g., the end event of an activity), and must precede or occur at the same time as e_i.

Theorem 3.2 (GetConstraintsRequiredToExecute is correct)**.** *Let $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ be a temporally consistent partial execution.* GetConstraintsRequiredToExecute(e_i, t) *returns a set of constraints $F_1, ..., F_n$ such that a team scenario φ_S satisfies $F_1, ..., F_n$ if and only if the partial execution that would result if e_i were executed at time t – namely $\langle \varphi_{ex} \wedge \varphi_{e_i}, \tilde{T}_{\varphi_{ex}} \cup \{e_i = t\} \rangle$ – can be extended to a temporally consistent execution $\langle \varphi_S, T_{\varphi_S} \rangle$.*

Proof. Not proven here – see prior work on Drake for details (Conrad & Williams, 2011). □

High-level pseudo code for the overall online execution algorithm is shown in Algorithm 1. The process begins by initializing the execution. A queue $Q_{remaining}$ of events remaining to be executed is created. It initially contains every event. Next, the initialization routine sets the initial time to $t = 0$, and associates an execution window with each event. The online algorithm then proceeds to enter a loop on Line 2, only terminating once there are no more remaining events in $Q_{remaining}$, or failure has been detected. An appropriately tiny pause is introduced at the end of each loop cycle (Line 19) so that time advances. Within each loop iteration, a number of checks and updates are performed that will be described shortly. Then, a `repeat` loop (Lines 10 - 18) begins which does the main work of executing the plan by executing individual events. Inside this `repeat` loop is an inner `for` loop (Lines 11 - 17) that iterates each event in $Q_{remaining}$, checks if the event can be executed now via CanExecuteEventNow? (Line 12), and then executes it if so (Line 14). The CanExecuteEventNow?(e_i, t) procedure computes the set of constraints $F_1, ..., F_n$ that must hold if e_i is executed at time t, and then checks if these constraints can be satisfied along with KB (we prove later in Theorem 4.7 that, due to the constraints in KB, CanExecuteEventNow? is correct). The `repeat` loop keeps repeating, only exiting when no more events can be executed by the inner `for` loop at the present time t.

Note that the above algorithm may execute any event as long as the associated set of constraints can hold. Sometimes during execution, it may be the case that multiple events

Algorithm 1: OnlineExecution()

Input: A matrix $D_{i,j}$, a knowledge base KB compiled with causal link and temporal conflict constraints, streams of uncontrollable choice assignments $\mathcal{U}(t)$, state estimates $\mathcal{P}(t)$, and activity completions $\mathcal{A}(t)$

Output: Stream of controllable choice assignments, dispatched planned activities, and immediate failure if problem is detected.

InitializeExecution()
while $|Q_{remaining}| > 0$ **do**
    CheckActivatedCausalLinks()
    Process any just-finished activities in $\mathcal{A}(t)$
    Call CommitToConstraints(KB, $\{U\}$) for any observed uncontrollable choice assignments $U \in \mathcal{U}(t)$
    Process missed execution windows
    if *not* CorrectTeamPlanExists?(KB) **then**
        return Failed
    end
    repeat
        for *event* $e \in Q_{remaining}$ **do**
            executable? , $\{F_1, ..., F_n\} \leftarrow$ CanExecuteEventNow?(e, t)
            if *executable?* **then**
                ExecuteEvent($e, \{F_1, ..., F_n\}$)
                break
            end
        end
    until *no event is executed*
    Sleep for a small time to allow time t to advance
end

Algorithm 2: CanExecuteEventNow?()

Input: An event e, current time t

Output: Returns two values: (1) a boolean for whether e can be executed at time t, and (2) a set of constraints that must hold if so.

$\{F_1, ..., F_n\} \leftarrow$ GetConstraintsRequiredToExecuteEvent(e, t)
if CouldCommitToAdditionalConstraints?($KB, \{F_1, ..., F_n\}$) **then**
    return True, $\{F_1, ..., F_n\}$
else
    return False, {False}
end

are eligible for execution. Currently, Pike picks one arbitrarily. In the future, it may be possible to employ other approaches to rank the events, such as considering which event would leave the greatest flexibility for the remainder of the execution.

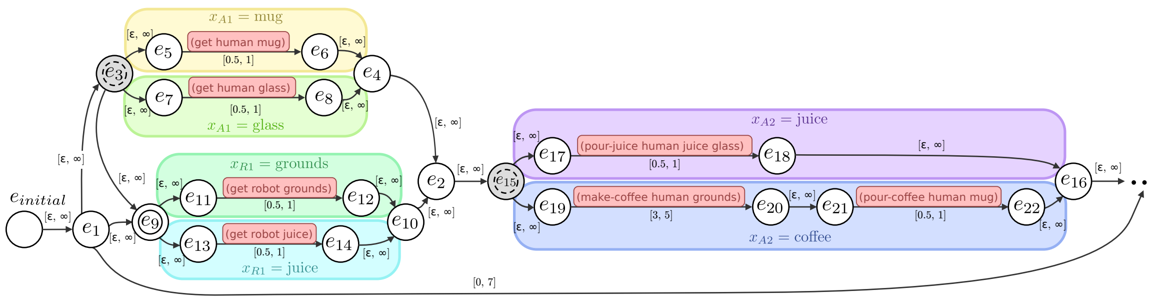


Figure 5: Plan for making breakfast, with events and temporal constraints labeled. ϵ is a small number (0.001 here) to ensure strict ordering constraints. Shaded backgrounds indicate guard conditions over the indicated events and temporal constraints. The second half of the plan is omitted for brevity.

We illustrate these algorithms by describing an example execution. Figure 5 shows a slightly modified version of the TPNU discussed earlier in Figure 1, now annotated more precisely with event names (ex., e_1), temporal constraints (ex., ordering constraints such as $[\epsilon, \infty]$ for a small value ϵ), and guard conditions (shaded backgrounds). The second half of the plan (toasting a bagel or having cereal) has been omitted for brevity but is still part of the TPNU. Assume offline compilation is finished, and therefore we have computed (1) the labeled APSP matrix $D_{i,j}$, (2) a set of labeled causal links (namely those illustrated in Figure 2), and (3) a knowledge base KB that can be queried.

For this example, a partial list of propositional constraints represented by KB is shown below. Full details about how these constraints are derived are discussed in Section 4.3 and Section 4.4, but we omit such details here to focus on the online execution algorithm. They capture relationships amongst choice variables that are required to ensure causal completeness as well as avoiding temporal conflicts. The variable assignments of the form $s_{p,e_c} = e_P$ where p is a predicate intuitively mean e_P has been chosen to support e_c: there is a labeled causal link where producer event e_P has an effect p that is a precondition of the consumer event e_c. The propositional constraints for this example are:

$$
\begin{aligned}
x_{A2} = \text{coffee} &\Rightarrow (s_{(\text{has mug}),e_{21}} = e_6 \wedge x_{A1} = \text{mug}) \\
x_{A2} = \text{coffee} &\Rightarrow (s_{(\text{has grounds}),e_{19}} = e_{12} \wedge x_{R1} = \text{grounds}) \\
x_{A2} = \text{juice} &\Rightarrow (s_{(\text{has glass}),e_{17}} = e_8 \wedge x_{A1} = \text{glass}) \\
x_{A2} = \text{juice} &\Rightarrow (s_{(\text{has juice}),e_{17}} = e_{14} \wedge x_{R1} = \text{juice}) \\
&\ldots \\
\neg(x_{A4} = \text{bagel} &\wedge x_{A2} = \text{coffee})
\end{aligned}
\tag{2}
$$

Execution now begins. Suppose the current time is $t = 0$. The online execution algorithm will iterate over each event, checking to see if it is executable or not. Let us consider and see if the event $e_{initial}$ is executable. Following the constraints above, PIKE derives that the following set $\{F_1, ..., F_n\}$ of additional constraints would need to hold in order to execute $e_{initial}$ now at $t = 0$:

Algorithm 3: ExecuteEvent($e, \{F_1, ..., F_n\}$)

Input: An event e, a set of constraints $\{F_1, ..., F_n\}$

Mark e as executed at time t
Remove e from $Q_{remaining}$
CommitToConstraints(KB, $\{F_1, ..., F_n\}$)
Propagate execution time t time to other events
Prune any other events with now-inconsistent guards from $Q_{remaining}$
[Optional: prune execution windows with now-inconsistent labels]
Dispatch any activities starting at e
If *variable-at-event*(e) is a controllable variable, commit to a consistent choice
Activate / Deactivate any labeled causal links starting / ending at e

- **Constraint 1:** True ($e_{initial}$ has guard True so it will always be executed)
- **Constraint 2:** True ($e_{initial}$ is directly executable - not the end event of an activity nor does its guard contain an assignment to an unassigned uncontrollable variable)
- **Constraint 3:** None. $t \not< -\infty$ so no constraint is added.
- **Constraint 4:** None. $t \not> \infty$ so no constraint is added.
- **Constraint 5:** None, as $e_{initial}$ has no predecessors.
- **Constraint 6:** None (same reasoning as above).

We have just the single constraint True. As a result, the output of the function CouldCommitToAdditionalConstraints?(KB, {True}) will be True since adding the constraint True conjunctively will not remove any correct team scenarios. Pike will therefore execute event $e_{initial}$ now at $t = 0$ by calling ExecuteEvent($e_{initial}$, {True}) on Line 14.

Algorithm 3 shows the pseudo code for the ExecuteEvent($e, \{F_1, ..., F_n\}$) procedure, which is called to actually execute some event e. The first steps in executing event e are to mark it as executed at time t, remove it from $Q_{remaining}$, and add the required constraints $\{F_1, ..., F_n\}$ to KB. Then, we propagate this time t to all e's neighbors so that their execution windows are appropriately updated. For example, consider event e_6 with execution window $lower\_bound(e_6) = \{(-\infty, \{\})\}$, $upper\_bound(e_6) = \{(\infty, \{\})\}$. Given the previously computed APSP distances $D_{e_6, e_{initial}} = \{(-2.021, \{x_{A1} = \text{mug}, x_{A2} = \text{coffee}, x_{A4} = \text{bagel}\}), (-0.503, \{x_{A1} = \text{mug}\}), (-0.001, \{\})\}$ and $D_{e_{initial}, e_6} = \{(\infty, \{\})\}$, we can update via Equation 1 to

$$
\begin{aligned}
lower\_bound(e_6) = \{&(0.001, \{\}), \\
&(0.503, \{x_{A1} = \text{mug}\}), \\
&(2.021, \{x_{A1} = \text{mug}, x_{A2} = \text{coffee}, x_{A4} = \text{bagel}\})\} \\
upper\_bound(e_6) = \{&(\infty, \{\})\}
\end{aligned}
$$

Similar updates are computed for the execution windows of all other events remaining to be scheduled.[2]

After propagating execution times to other events, we prune events whose guard conditions are now inconsistent with KB by removing them from $Q_{remaining}$. This is illustrated later in this example when a choice is observed. We may also optionally prune invalid labeled values from execution windows, as they will not affect algorithm correctness but could impact system latency. We found experimentally, however, that this could at times actually increase the worst-case execution latency time (as removal could be expensive). Hence this step is optional.

Next, after any pruning has occurred, Pike will check to see if e is the start event of any activity in the TPNU. If so, then the activity is dispatched at the current time. This is not the case for $e_{initial}$.

The final step in ExecuteEvent(...) is to *activate* or *deactivate* any labeled causal links starting or ending at event e, respectively. An activated labeled causal link is one that is continually monitored during execution by comparison with the current estimated world state $\mathcal{P}(t)$, which we illustrate later.

After executing $e_{initial}$ and propagating to its neighbors, the execution loop (Line 10 of OnlineExecution) continues trying to find other events that are executable. It turns out that for this example, none are – the plan's temporal constraints force all other events to be executed strictly after $e_{initial}$, but the time has not yet advanced past $t = 0$. For any other event, there must therefore be some labeled value $(l, \{\})$ for $l > 0$ in its lower bound execution window (per the propagation step above). By Constraint 3, $\{F_1, ..., F_n\}$ would contain the constraint $\neg$True = False, which cannot be conjunctively added to KB while still maintaining a team scenario. Therefore no other event is executable at $t = 0$, so Pike must wait for time to advance, and the `repeat` loop exits (Line 18).

Suppose it is now later, $t = 1$. We may now execute event e_1, as $\{F_1, ..., F_n\}$ would contain the single constraint True. The time $t = 1$ is propagated to other events in the plan. This pattern continues, and guarantees that events will be scheduled at their proper times and that the partial execution remains correct.

Suppose later that we observe a message in $\mathcal{U}(t)$ (Line 5) that the human is making the choice to pick up the mug, namely $x_{A1} = \text{mug}$. Such inferences are made external to Pike in a separate activity recognition module such as LCARS (Lane, 2016), and reported to Pike via $\mathcal{U}(t)$. This triggers a call to CommitToConstraints(KB, $\{x_{A1} = \text{mug}\}$), adding this assignment to the knowledge base KB. Additionally, since x_{A1} is an uncontrollable variable, other events in the plan may become executable now that it has been observed. For example, event e_3 can now be executed since we have observed the outcome of its requisite choice variable, as can e_9 and e_5. As e_5 is the start event of the `(get human mug)` activity, this action is dispatched (though in practice the human will have already begun doing it as the activity recognition system reported the outcome of x_{A1}).

To illustrate the mechanics by which human choices influence robot adaptations, we point out that the robot may *not* choose to execute event e_{13} (the start of the `(get robot juice)` activity). Doing so would result in $\{F_1, ..., F_n\}$ containing $x_{R1} = \text{juice}$ (which is

2. In the works of Conrad (2010) and Muscettola et al. (1998), a minimal dispatchable form is computed that only requires propagating event updates to local neighbors, not to all other events. Such an approach could be adopted here but is omitted for simplicity.

$\varphi_{e_{13}}$). This would cause

$$\text{CouldCommitToAdditionalConstraints?}(\text{KB}, \{x_{R1} = \text{juice}\})$$

to return False, which can be seen by logically analyzing the set of constraints in KB. We show that KB would become inconsistent. KB currently consists of Equation 2, plus the assignment $x_{A1} = \text{mug}$. If the executive were to additionally add the assignment $x_{R1} = \text{juice}$ as would be required to execute e_{13}, then the human could not later consistently get coffee $x_{A2} = \text{coffee}$ (doing so would require $x_{R1} = \text{grounds}$ instead by the second constraint). Thus we infer that the human must later pour juice $x_{A2} = \text{juice}$. By the third constraint, this requires that the human get the glass $x_{A1} = \text{glass}$, which contradicts the earlier added assignment of $x_{A1} = \text{mug}$. We have shown that adding the constraint $x_{R1} = \text{juice}$ leads to an unsatisfiable KB; therefore CouldCommitToAdditionalConstraints? returns False and the executive cannot execute e_{13} now.

On the other hand, event e_{11} (the start of the `(get robot grounds)` activity) can be executed now, as its guard condition $x_{R1} = \text{grounds}$ is consistent with KB. Once executed, the `(get robot grounds)` activity is dispatched to the robot. This will in turn trigger lower-level motion planning and control algorithms to physically implement this action. We also see that the robot is properly adapting to the human's intent.

Once a choice has been made or observed, certain events are no longer executable because their guards are inconsistent with KB. For example, after observing the choice $x_{A1} = \text{mug}$, events e_7 and e_8, which each have a guard $x_{A1} = \text{glass}$, can never be executed. These events are therefore pruned from execution by removing them from $Q_{remaining}$ and never considered again as candidates to be executed.

Shortly later, the executive receives messages in $\mathcal{A}(t)$ (Line 4) informing that the `(get human mug)` and `(get robot grounds)` activities have finished successfully. Pike hence calls ExecuteEvent(...) for the end events of those two activities. This makes events e_6 and e_{12} now directly executable, so Pike executes them now. This in turn allows other events to become executable, and the execution process continues.

It may be the case that, during execution, an activity takes too long and violates a temporal constraint. This circumstance is detected within the OnlineExecution(...) method on Line 6 by detecting missed execution windows. This routine follows logic similar to Constraint 4 in constructing $\{F_1, ..., F_n\}$. If the current time t is greater than the upperbound u for some $(u, \varphi) \in \textit{upper\_bound}(e_i)$, then we add the constraint $\neg\varphi$ to KB. This is because we cannot go back in time, and there is no way that we will meet the given upperbound required under environment φ. For example, if the upper bound part of the execution window for some event e_i is the labeled value set $\{(3, \{x = 1\}), (6, \{\})\}$ and the current time is $t = 4$, then the executive can infer that it has missed the upper bound of 3 required by the choice $x = 1$. Therefore, the constraint $\neg(x = 1)$ is added to KB.

Finally, the last aspect of execution that we wish to illustrate is execution monitoring. Our executive employs causal link-based execution monitoring to detect failures early, hopefully before they become critical. Recall that there is a labeled causal link where consumer e_{21} has a precondition of `(have mug)` that is achieved by producer event e_6 (the end of the `(get human mug)` activity). If the predicate `(have mug)` ceases to be true after e_6 is executed – for instance, if the mug somehow gets knocked off of the table by accident – then this labeled causal link is violated. Unlike earlier work in execution monitoring however, a violated causal link in our setting does not necessarily imply plan failure (Levine, 2012). For example, if the plan contains sufficient flexibility to avoid executing e_{21} (i.e., by choosing x_{A2} = juice), then execution can still succeed. Some TPNUs contain explicit contingency options to address likely causes of failure such as unreliable robotic hardware, where the robot has multiple choices of whether or not to pick something up (this is a form of k-fault tolerance). However, at this point during execution in our breakfast example, the robot cannot "change its mind" and commit to x_{A2} = juice. Hence this causal link violation implies plan failure. We do not need to wait until the consumer activity `(pour-coffee human mug)` is actually executed to report the failure; our executive will detect it as early as possible during execution (such as, for example, immediately after the `(get human mug)` activity finishes). Such early failure detection is extremely beneficial in practice in many autonomous systems, as it provides additional time to replan (Levine, 2012).

A procedure called CheckActivatedCausalLinks (Line 3) is responsible for detecting causal link violations that occur online. It iterates over all activated labeled causal links from e_P to e_c with predicate p. If $p \notin \mathcal{P}(t)$, then CommitToConstraints(KB, $\{\neg(s_{p,e_c} = e_P)\}$) is called. For example, suppose that after executing e_6 but before e_{12} the executive discovers from the current state estimates $\mathcal{P}(t)$ that `(has mug)` $\notin \mathcal{P}(t)$. Then, the executive commits to the additional constraint $\neg(s_{(\text{has mug}),e_{21}} = e_6)$. As can be seen from Equation 2, this invalidates the right-hand side of the first implication constraint, therefore precluding the choice x_{A2} = coffee. Since this choice must be made however, KB becomes inconsistent. If there had been sufficient flexibility in the plan to handle this causal link violation, then KB would still have remained consistent and execution would have succeeded (barring more violations), albeit possibly by forcing some later choice assignments. In this way, Pike is able to react to certain unmodeled disturbances in the world, and predict future plan failure before it occurs.

This concludes our explanation of Pike's online execution algorithm. In summary, we employ a strategy of least commitment (both in terms of timing and controllable choices). The resulting execution obeys the plan's temporal constraints by employing techniques from Drake. Pike additionally makes controllable choices using a compiled knowledge base KB describing the dependencies between controllable and uncontrollable choices, allowing it to ensure that the partial execution remains correct. The result is a mixed-initiative execution where the robot adapts to the human's intentions, the plan's timing constraints, and unmodeled disturbances.

4. Using Causal Links for Decision Making

Previously in Section 3, we discussed the online execution component of Pike and showed the manner in which it used the results of offline compilation. In this Section, we now step back and discuss the details of offline compilation that make low-latency, correct online execution possible.

Specifically, we discuss how we extract labeled causal links from the plan and how they apply to online decision making. These labeled causal links capture important relationships between choices in the plan by making explicit its causal structure, and they are used by our executive to make intelligent choices online via KB. From the perspective of human-robot

Algorithm 4: OfflineCompilation()

Input: TPNU $\mathcal{T}$, action model, initial and goal states
Output: Labeled APSP matrix $D_{i,j}$, labeled causal links, compiled propositional constraints KB

$D_{i,j}$ ←Compute labeled APSP of $\mathcal{T}$
Extract labeled causal links from $\mathcal{T}$ using $D_{i,j}$
Generate augmented TPNU $\mathcal{T}'$ and KB using $\mathcal{T}$ and labeled causal links
if *any temporal constraints were added to* $\mathcal{T}'$ **then**
  $D_{i,j}$ ← Compute labeled APSP of $\mathcal{T}'$
end
Add temporal conflict constraints to KB
Compile propositional constraints KB

interaction, this causal structure allows Pike to choose robot adaptations that are logically consistent with the human's intentions.

We first introduce some preliminaries, including algorithms and data structures for performing labeled temporal reasoning. These are necessary for causal link extraction, as producers must precede consumers in time in our conditional plans. We then introduce labeled causal links, a generalization of causal links for contingent, temporally-flexible plans. We also introduce a transformation from our input TPNU to an *augmented* TPNU along with a set of constraints based on these causal links, which together allow Pike to maintain a correct partial execution online.

High-level pseudo code for the entire offline compilation procedure is illustrated in Algorithm 4. First, a labeled all-pairs shortest path matrix $D_{i,j}$ is computed, which provides temporal reasoning that is necessary both for online execution and for causal link extraction. Next, labeled causal links and threats are extracted, and encoded as a propositional theory KB. An augmented TPNU is then constructed based on these causal links. This process sometimes results in new temporal constraints being added to ensure causal completeness; if so, the labeled APSP algorithm is re-run to reflect these new temporal constraints. Finally, the propositional constraints KB are compiled into a form suitable for efficient use during online execution.

4.1 Labeled APSP

In this section, we describe the labeled all-pairs shortest path algorithm (APSP), which enables Pike to reason about the temporal constraints in the TPNU as a function of the choices made. The resulting data structure, a matrix of shortest path LVSs, is used for extracting labeled causal links and also for online dispatching (as it is a "dispatchable form", meaning that it makes explicit all implicit temporal constraints) (Dechter et al., 1991).

The labeled APSP algorithm is a strict generalization of the Floyd Warshall all-pairs shortest path algorithm (Conrad, 2010). The Floyd Warshall algorithm takes as input a weighted directed graph (often called a "distance graph" in the scheduling literature) with N vertices, and outputs an $N \times N$ matrix $d_{i,j}$, where each entry in the matrix contains the shortest path length (a number) from i to j. These shortest path distances represent required timing constraints (Dechter et al., 1991).

In contrast, the labeled APSP instead generates a matrix $D_{i,j}$ where each entry is an LVS representing the shortest path from i to j as a function of choice. Its input is the *labeled distance graph*, which similarly generalizes of the distance graph from the STN literature. Instead of each edge weight between two vertices being a single number, it is now a labeled value set, representing the edge weight as a function of what choices are made (Conrad & Williams, 2011). Just as an STN can be readily transformed into a distance graph (Dechter et al., 1991), a TPNU (as well as many other temporal representations, such as the DTP) can be readily transformed into a labeled distance graph through a straightforward process. Each vertex in the labeled distance graph is an event from the TPNU, and each labeled simple temporal constraint is converted into two labeled weights going in opposite directions between the associated events, corresponding to the upper and lower bound constraints (Conrad, 2010).

For this work, we use the labeled APSP algorithms as introduced in Drake, with some additional modifications. These modifications compute tighter bounds for each LVS by considering the finite domains of variables and adding additionally, logically implied labeled values when appropriate. Details about the labeled APSP algorithm with supplemental modifications can be found in Appendix C.

Given the resulting output $D_{i,j}$ from the labeled all-pairs shortest path, we introduce some necessary terminology. First, we discuss temporal conflicts.

Definition 4.1 (Temporal Conflict)**.** An environment φ is called a *temporal conflict* iff $Q_{D_{e_i,e_i}}(\varphi) < 0$ for any event e_i

Theorem 4.1 (Temporally Inconsistent Candidate Subplan)**.** *A candidate subplan φ_S is temporally inconsistent iff there exists a temporal conflict environment φ where $\varphi_S \models \varphi$.*

Proof. Not proven here – see prior work on Drake for details (Conrad & Williams, 2011). □

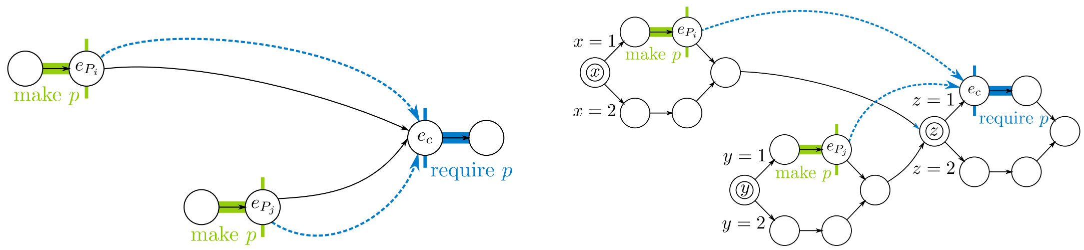
Temporal conflicts are important, because they allow us to detect temporally inconsistent candidate subplans so Pike can avoid them online. The definition above is a direct analogy to the negative cycles of distance graphs for STNs (Dechter et al., 1991).

We can also define the very useful concept of precedence, which allows us to determine if some events must be scheduled before other events in all temporally consistent executions.

Definition 4.2 (Precedence)**.** Suppose we have two events, e_i and e_j, with guards φ_{e_i} and φ_{e_j} respectively. Suppose that we also have a third, *context* environment φ_C. We say that e_i precedes e_j in context φ_C, denoted $e_i \prec e_j \Big|_{\varphi_C}$, iff $Q_{D_{e_j,e_i}}(\varphi_{e_i} \wedge \varphi_{e_j} \wedge \varphi_C) < 0$.

When the context $\varphi_C = \{\}$, we denote $e_i \prec e_j \Big|_{\{\}}$ as just $e_i \prec e_j$ for brevity. We define a similar *succession* operator $\succ$ analogous to $\prec$: $e_i \succ e_j$ iff $Q_{D_{e_i,e_j}}(\varphi_{e_i} \wedge \varphi_{e_j}) < 0$.

If $e_i \prec e_j$, then e_i will be executed before e_j in all temporally consistent executions in which both e_i and e_j are activated. If an additional context environment φ_C is provided, it has the effect of conditioning on those sets of choices; e_i is executed before e_j whenever φ_C holds.



(a) A choice-less plan with two producers and one consumer. Black lines are $[0, \infty]$ temporal constraints, and lighter dotted lines represent causal links. Either could be the supporting causal link during execution.

(b) A similar plan to the above, but now with choices surrounding each activity. Labeled causal links are required for this plan, and will allow us to reason about consistent sets of choices amongst x, y, and z.

Figure 6: Labeled causal links enable an executive to ensure that preconditions are satisfied in contingent plans.

A few properties about precedence that hold when $\varphi_{e_i} \wedge \varphi_{e_j} \wedge \varphi_C$ is not a temporal conflict:

- It is possible for $e_i \prec e_j\big|_{\varphi_C}$ to hold, or for $e_j \prec e_i\big|_{\varphi_C}$ to hold, but not both.
- It is possible that neither $e_i \prec e_j\big|_{\varphi_C}$ nor $e_j \prec e_i\big|_{\varphi_C}$ hold. In that case, a precise a priori ordering between the two events cannot be determined before the plan is executed. There are temporally consistent executions in which e_i comes before e_j, and other(s) in which e_j comes before e_i.
- There are thus three possibilities: (1) $e_i \prec e_j\big|_{\varphi_C}$, (2) $e_j \prec e_i\big|_{\varphi_C}$, or (3) neither.

Definition 4.3 (Incomparability)**.** If neither $e_i \prec e_j\big|_{\varphi_C}$ nor $e_j \prec e_i\big|_{\varphi_C}$, we say that e_i and e_j are *incomparable* in context φ_C, denoted by $e_i \parallel e_j\big|_{\varphi_C}$.

4.2 Labeled Causal Links

In this section, we introduce labeled causal links and motivate their use in executing contingent plans.

Given a correct, totally ordered plan and a STRIPS-like action model, the process of extracting causal links from said plan (for use during execution monitoring) is straightforward. For each precondition p of a *consumer* action A_c in the plan, we may find its causal link by regressing backwards from A_c in the plan until we find a *producer* action A_P that produces p as an effect. In this case, we associate a single, unique causal link with every precondition p of every action A_c in the plan.

Things become more complicated when extracting causal links from partially-ordered plans, including those with metric temporal constraints. Aside from inferring precedence via a transitive closure operation (or in the case of metric temporal constraints, an all-pairs shortest path algorithm), there may in general be multiple *candidate causal links* for each precondition of each action in the plan (Levine, 2012). This is illustrated in Figure 6a, which shows two partially ordered producer actions that produce p as an effect for the later-occurring consumer that requires p as a precondition. The $[0, \infty]$ temporal bounds permit any ordering of the two producers; the top may come first, second, or they may overlap. So, which of the two actions will be the causal link? For the purposes execution monitoring and detecting violated causal links, only the latest producer matters. Hence, there is flexibility during execution since the two producers are unordered. If during execution, the top-most producer finishes first, then we designate the bottom-most action as providing the final support for the consumer – and vice versa. Due to such cases, there may in general be multiple possible candidate causal links for each precondition of each action in the plan. The plan is only at risk of being causally incomplete if the latest-most occurring causal link is violated (Levine, 2012).

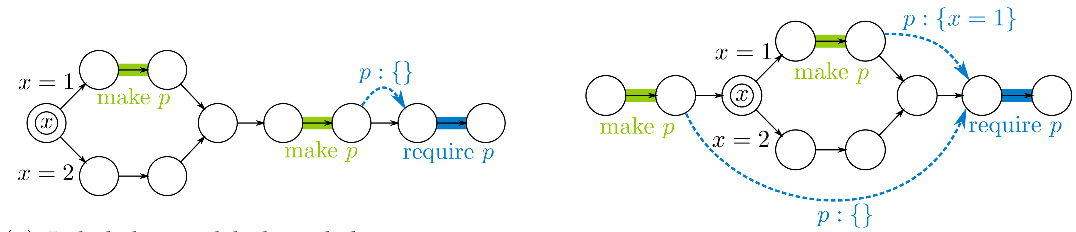
Things are complicated further in the case of metric temporal plans with choice, such as the TPNU representation embraced by this work. With the addition of choices, certain producer actions may not even execute depending on what choices are made. As a result, for each precondition p in the plan, we now have a set of *labeled causal links*, at least one of which must hold in order to guarantee that the precondition is met in the plan. The label captures the dependence of the causal link on choices made in the plan.

For example, suppose we generalize the earlier example Figure 6a to Figure 6b, where each of the activities are now conditioned on a choice. In this case, each of the activities may or may not be activated depending on the choice assignments, and may not be executed. The causal links are therefore now contingent upon the choices made. For example, if the executive chooses $x = 2$, then the top-most causal link will vanish since the producer is not executed. Similarly, the bottom-most causal link vanishes if $y = 2$ is chosen. If $z = 1$ is chosen (and hence the consumer will be executed), then either $x = 1$ or $y = 1$ must hold for the consumer's precondition to be supported (and for the plan to be causally complete). But if $z = 2$ is chosen, then there are no constraints on either x or y. This is the core intuition behind labeled causal links: the causal links now depend on the choices made, and for a plan to be causally complete, there must be at least one activated causal link supporting every precondition of every activated event. We therefore *label* each causal links with its requisite choice environments. In the example, the top-most causal link is labeled with $\{x = 1\}$, and the bottom-most one is labeled with $\{y = 1\}$.

The notion of a threat can also be generalized for contingent plans. A threat that asserts $\neg p$ as an effect may or may not be activated depending on the choices made. Note that if a threat is indeed activated, and if the consumer and producer that it threatens are also activated, the executive may still possibly be able to ensure a causally complete plan execution by making certain choices to activate other, later-occurring labeled causal links. This will be described more later.

Given this intuition, we are now prepared to define a labeled causal link.

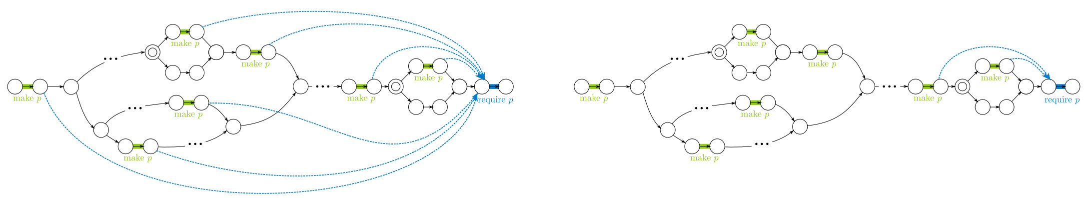
Definition 4.4 (Labeled Causal Link)**.** A *labeled causal link* is a tuple $\langle e_P, e_c, p, \varphi\rangle$ where e_P is the *producer* event with $p \in \text{Effects}(e_P)$, e_c is a consumer event occurring after e_P with $p \in \text{Preconditions}(e_c)$, p is a predicate, and the label φ is an environment. The label φ will be the execution environment of e_P, φ_{e_P}.



(a) Labeled causal link with later-occurring producer dominates earlier one (not shown).

(b) No causal link dominance here.

Figure 7: Causal link dominance examples. Unlabeled temporal constraints are $[0, \infty]$.



(a) All labeled causal links.

(b) Only non-dominated labeled causal links.

Figure 8: Comparison of labeled causal links before and after dominated ones have been removed. With fewer links, the problem is simplified.

We define one final useful property related to labeled causal links: causal link *dominance*. While not strictly necessary for correctness, the notion of dominance allows us to prune out many would-be causal links that are superseded by others and need not be considered. A labeled causal link for some consumer event e_c with producer e_{P_i} dominates another labeled causal link with the same consumer but different producer e_{P_j} iff (1) whenever e_{P_j} is activated then e_{P_i} is also activated, (2) when all three events are activated then e_{P_i} must occur after e_{P_j}, and (3) when all three events are activated then e_{P_i} must precede e_c.

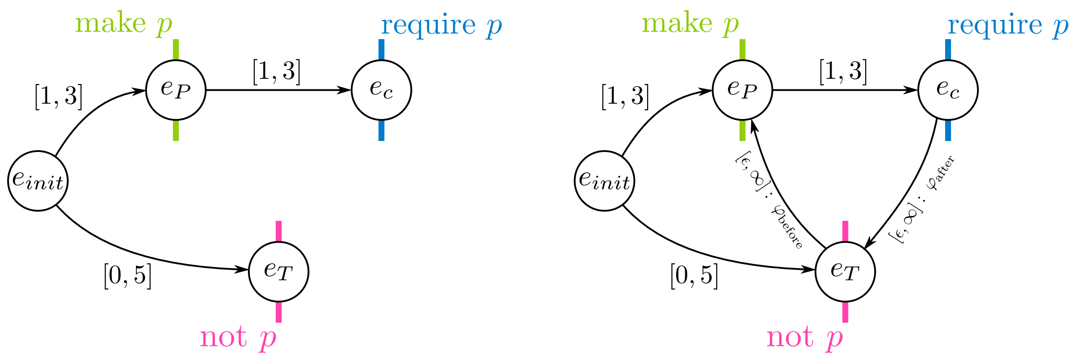
Definition 4.5 (Labeled Causal Link Dominance)**.** A labeled causal link $\langle e_{P_i}, e_c, p, \varphi_{e_{P_i}} \rangle$ *dominates* another labeled causal link $\langle e_{P_j}, e_c, p, \varphi_{e_{P_j}} \rangle$ iff all of the following conditions hold:

1. $\varphi_{e_{P_j}} \models \varphi_{e_{P_i}}$

2. $e_{P_j} \prec e_{P_i} \Big|_{\varphi_{e_c}}$

3. $e_{P_i} \prec e_c \Big|_{\varphi_{e_{P_j}}}$

Some examples of causal link dominance are shown in Figure 7. Consider Figure 7a, in which we have a plan with a single choice of a producer, followed by a second producer. The labeled causal link with the later-occurring producer dominates the other one. This is because whenever the earlier producer is activated, so is the later-occurring one. The situation is however reversed in Figure 7b, where the order of the choice is reversed. In this case, the later-occurring producer is not necessarily activated whenever the earlier one is,



(a) Original TPNU $\mathcal{T}$. (b) The augmented TPNU $\mathcal{T}'$.

Figure 9: A TPNU example, original and augmented.

so there is no domination. This structure in Figure 7b is useful for modeling contingency or recovery actions in a plan; should there be an unexpected disturbance that negates p sometime before the choice, the executive may recover by choosing $x = 1$ in the later choice, thus ensuring that the precondition will be supported.

It is safe to ignore dominated causal links from consideration during the compilation process, where the augmented TPNU is constructed.

Theorem 4.2 (Dominated Causal Links are Irrelevant)**.** *Dominated labeled causal links do not influence the correctness of an execution.*

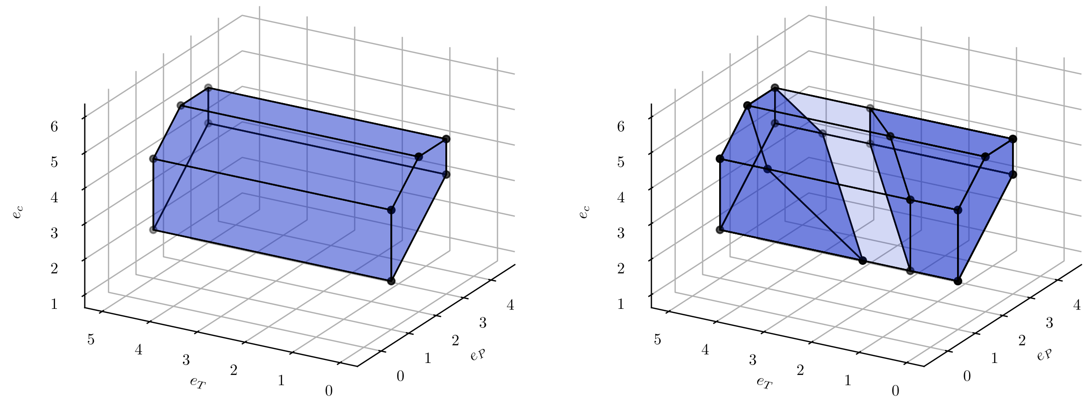
Proof. See Appendix A. □

Pruning dominated causal links can have a positive impact on the executive's performance by significantly decreasing the size of the compiled solutions found. This is illustrated in Figure 8b, where a number of labeled causal links can be pruned to obtain much simpler constraint structure.

4.3 Augmented TPNUs

Now that we have introduced labeled causal links, we discuss how they can be used online to guide controllable choices and scheduling decisions. To guarantee correct executions that are causally complete, we transform the original TPNU $\mathcal{T}$ into a new TPNU $\mathcal{T}'$ which we call the *augmented* TPNU, and also generate a knowledge base of propositional constraints we call KB. Together, this augmented TPNU $\mathcal{T}'$ and set of constraints KB describes the space of all possible correct executions of the original TPNU $\mathcal{T}$. As we shall illustrate shortly, $\mathcal{T}'$ along with KB represents every correct execution of $\mathcal{T}$ – but it also prunes out incorrect executions from $\mathcal{T}$ that are temporally consistent yet causally incomplete. Online during execution, Pike uses $\mathcal{T}'$ and KB to make controllable choices and scheduling decisions in such a way as to maintain a correct partial execution.

Consider the example TPNU shown in Figure 9a. This TPNU has four events, each of which must be assigned a time to form a schedule. We assume here that event e_{init} is scheduled at $t = 0$, leaving three remaining events to be scheduled. This TPNU has no



(a) Space of all temporally consistent executions of $\mathcal{T}$.

(b) Space of all correct (temporally consistent and causally complete) executions of $\mathcal{T}$.

Figure 10: Various spaces of executions.

choice variables, so there is just one candidate subplan φ_S. What is the space of possible temporally consistent schedules for $\mathcal{T}$? We visualize it in Figure 10a. The x, y, and z axes represent the scheduled times of events e_P, e_T, and e_c, respectively. Each point in this 3D space represents a schedule for $\mathcal{T}$, and the shaded region represents the set of all temporally consistent schedules. This space is convex, as the scheduling problem can be equivalently reformulated as a linear program.

While all schedules visualized in Figure 10a are temporally consistent, they are not all correct. This highlights one of the key differences between the scheduling problem for temporal networks and the plan execution problem: causal completeness must be enforced. Event e_c has precondition p, event e_P asserts p as an effect (and is therefore a producer for a labeled causal link), and event e_T asserts $\neg p$ as an effect and is a potential threat. Any execution $\langle \varphi_S, T_{\varphi_S} \rangle$ in which e_T is scheduled between e_P and e_c – namely $T_{\varphi_S}(e_P) \leq T_{\varphi_S}(e_T) \leq T_{\varphi_S}(e_c)$ – is causally incomplete. We can also see that $T_{\varphi_S}(e_P) < T_{\varphi_S}(e_c)$ is required in any causally complete execution (any temporally consistent execution satisfies this however, due to the $e_P \xrightarrow{[1,3]} e_c$ temporal constraint). The space of all correct executions is visualized in Figure 10b. We see that a large slice has been removed relative to Figure 10a: these pruned executions were temporally consistent yet causally incomplete, and represent executions where e_T occurred between e_P and e_c.

It is visually apparent From Figure 10b that there are two separate subregions. Our approach in Pike is to represent each of these subregions by a different candidate subplan φ'_S of the augmented TPNU $\mathcal{T}'$. By modifying the original TPNU $\mathcal{T}$ appropriately, the augmented TPNU $\mathcal{T}'$ contains more candidate subplans than the original, and each represents a set of correct executions. Pike covers the entire space of all correct executions in this way. In our visualized example, $\mathcal{T}$ has a single candidate subplan and $\mathcal{T}'$ has two, one for each of the subregions.

Creating the augmented TPNU $\mathcal{T}' = \langle \mathcal{V}', \mathcal{E}, \mathcal{C}', \mathcal{A} \rangle$ and KB is accomplished by copying $\mathcal{T}$, and subsequently adding new controllable choice variables to $\mathcal{V}'$, new temporal constraints to $\mathcal{C}'$, and propositional constraints to KB. In this example, one of the new choice variables

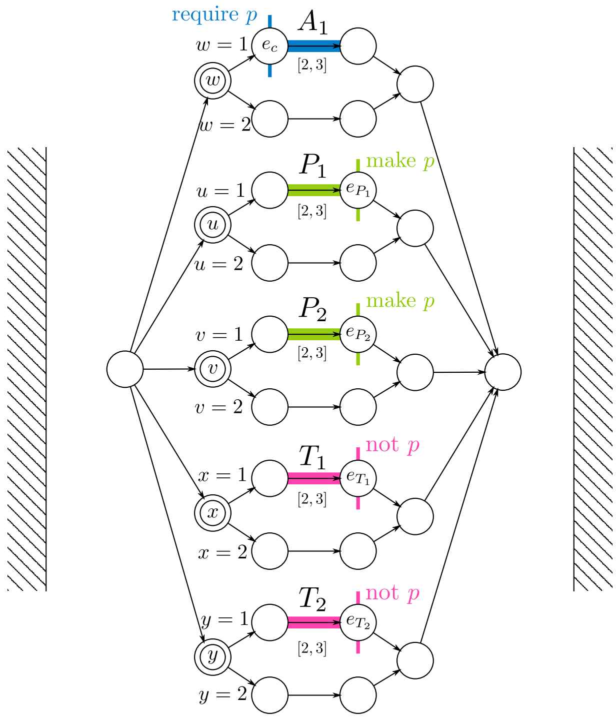


Figure 11: A larger example TPNU. Unlabeled temporal constraints are $[0, \infty]$.

added to $\mathcal{V}'$ will choose whether e_T will come before or after the labeled causal link from e_P to e_c. We will introduce the full encoding shortly, but for now we simply call this variable $o \in \{-1, +1\}$ where $o = -1$ indicates that e_T must come before e_P and $o = +1$ indicates that e_T must come after e_c. We also add two additional temporal constraints: $e_T \xrightarrow{[\epsilon,\infty]} e_P : \varphi_{\text{before}}$ and $e_c \xrightarrow{[\epsilon,\infty]} e_T : \varphi_{\text{after}}$, resulting in the augmented TPNU shown in Figure 9b. These new temporal constraints will act as ordering constraints that are activated based on o: the guards φ_{before} and φ_{after} are defined such that φ_{before} holds if $o = -1$, and φ_{after} holds if $o = +1$. There are thus two candidate subplans of $\mathcal{T}'$, one corresponding to each possible assignment, differing only in which newly added temporal constraint is activated. Each of these candidate subplans has an associated set of temporally consistent executions, shown as the left and right convex subregions of Figure 10b. By Theorem 4.5 (discussed and proved later), all of these executions are guaranteed to be causally complete and correct. In this way, Pike represents the complex space of correct executions of $\mathcal{T}$ using a factored approach, where $\mathcal{T}$ is transformed into an augmented TPNU $\mathcal{T}'$ with an associated KB. Every candidate subplan φ'_S of $\mathcal{T}'$ that satisfies KB represents a simpler, convex set of correct executions (see Theorem 4.6). Taking all of these candidate subplans of $\mathcal{T}'$ together, Pike maintains the full space of possible executions.

Plans may be additionally complicated with the addition of choices, multiple candidate causal links, and multiple threats. A larger example with these features is illustrated in Figure 11. There are five activities, which are conditioned on choice variables w, u, v, x and y. Activity A_1 has a precondition of p. Two activities P_1 and P_2 both produce p as an effect. We also have two potential threats T_1 and T_2 that negate p as an effect. All unlabeled temporal constraints are $[0, \infty]$, so any activity may come before or after (or overlap) other activities. This example involves both *potential producers* and *potential threats*.

Which producers support the precondition p of event e_c? The only events that produce p are e_{P_1} and e_{P_2}. However, neither is guaranteed to precede e_c due to the loose temporal constraints of the problem. These are hence referred to as *potential producers*. We address potential producers by adding additional labeled temporal constraints to $\mathcal{C}'$ of the augmented TPNU, as well as a new controllable choice variable to $\mathcal{V}'$. This new variable chooses which of the different possible producer events — e_{P_1} or e_{P_2} in this case — will be chosen to enforce support for the precondition p of event e_c (i.e., which event will be the producer in the supporting causal link). We define this new choice variable, s_{p,e_c}, as[3]

$$s_{p,e_c} = \begin{cases} e_{P_1} & \text{if } e_{P_1} \text{ will be the producer} \\ e_{P_2} & \text{if } e_{P_2} \text{ will be the producer} \\ \bot & \text{if neither will be the producer} \end{cases}$$

We also add the propositional constraint below to KB, which asserts that if e_c is activated (as is the case when $w = 1$), then one of the producers must also be activated and chosen as the support:

$$(w = 1) \Rightarrow (s_{p,e_c} = e_{P_1} \wedge u = 1) \vee (s_{p,e_c} = e_{P_2} \wedge v = 1)$$

In this way, we translate a labeled causal link into a propositional state logic constraint.

We must also enforce that if $s_{p,e_c} = e_{P_1}$ is chosen, event e_{P_1} must precede e_c. This resolves the potential producer and forms the basis for a labeled causal link. We accomplish this by adding the following temporal constraint to $\mathcal{C}'$, labeled appropriately for the environment where this choice is made and where the consumer and producer activities are both activated:

$$e_{P_1} \xrightarrow{[\epsilon,\infty]} e_c : \quad \{s_{p,e_c} = e_{P_1}, u = 1, w = 1\}$$

Note that we only need to add such additional temporal constraints if the temporal flexibility of the plan could allow for the producer event to happen after the consumer. If $e_{P_1} \prec e_c$, no additional temporal constraints are added to $\mathcal{C}'$.

We add a similar constraint for e_{P_2}:

$$e_{P_2} \xrightarrow{[\epsilon,\infty]} e_c : \quad \{s_{p,e_c} = e_{P_2}, v = 1, w = 1\}$$

Next, we address the two potential threats T_1 and T_2. For the following arguments, suppose that P_1 is chosen to be the supporting producer. Hence, the labeled causal link from event e_{P_1} to event e_c over predicate p will be the supporting causal link, and $s_{p,e_c} = e_{P_1}$. To avoid threats to this labeled causal link, no other action may be scheduled during the interval between e_{P_1} and e_c that negates p. Both e_{T_1} and e_{T_2} are *potential threats*, because the loose flexibility in the temporal constraints admits some executions that are causally

3. We include $\bot$ in the domain of s_{p,e_c} to represent that s_{p,e_c} may acceptably be left unassigned if e_c is not activated. This is useful for there to be a solution if causal link violations cause $\neg(s_{p,e_c} = e_{P_1})$ and $\neg(s_{p,e_c} = e_{P_1})$ constraints to be added to KB: namely deactivating e_c by choosing $w = 2$.

Constraint	Reason
Propositional for KB:	
$\varphi_{e_c} \Rightarrow \bigvee_{e_{P_i} \in P} (s_{p,e_c} = e_{P_i} \wedge \varphi_{e_{P_i}})$	At least one activated producer
$\neg\varphi_C$	Avoid definite threat
$\neg\varphi_C$	Avoid temporal conflict
Temporal for $\mathcal{T}'$:	
$e_{P_i} \xrightarrow{[\epsilon,\infty]} e_c : \ \{s_{p,e_c} = e_{P_i}\} \wedge \varphi_{e_{P_i}} \wedge \varphi_{e_c}$	Producers precede consumers
$e_c \xrightarrow{[\epsilon,\infty]} e_{T_j} : \ \varphi_C$	Force threat after consumer
$e_{T_j} \xrightarrow{[\epsilon,\infty]} e_{P_i} : \ \varphi_C$	Force threat before producer
$e_{T_j} \xrightarrow{[\epsilon,\infty]} e_{P_i} : \ \{o_{p,e_c,e_{P_i},e_{T_j}} = -1\} \wedge \varphi_C$	Force threat before producer
$e_c \xrightarrow{[\epsilon,\infty]} e_{T_j} : \ \{o_{p,e_c,e_{P_i},e_{T_j}} = +1\} \wedge \varphi_C$	Force threat after consumer

Table 1: Summary of constraints added for $\mathcal{T}'$ and KB.

complete and others that are not. They must each be scheduled either before e_{P_1} or after e_c, as was the case in the earlier example. We resolve these potential threats by adding additional temporal constraints to $\mathcal{C}'$ to enforce this, and by adding a new controllable choice variable to $\mathcal{V}'$ choosing whether it will come before or after:

$$o_{p,e_c,e_{P_1},e_{T_1}} = \begin{cases} -1 & \text{if } e_{T_1} \text{ will precede } e_{P_1} \\ +1 & \text{if } e_{T_1} \text{ will succede } e_c \end{cases}$$

A similar choice variable $o_{e_c,p,e_{P_1},e_{T_2}}$ is also added, representing the ordering of e_{T_2}. We also add the following temporal constraints to $\mathcal{C}'$:

$$e_{T_1} \xrightarrow{[\epsilon,\infty]} e_{P_1} : \ \{s_{p,e_c} = e_{P_1},\ o_{p,e_c,e_{P_1},e_{T_1}} = -1,\ u = 1,\ x = 1,\ w = 1\}$$
$$e_c \xrightarrow{[\epsilon,\infty]} e_{T_1} : \ \{s_{p,e_c} = e_{P_1},\ o_{p,e_c,e_{P_1},e_{T_1}} = +1,\ u = 1,\ x = 1,\ w = 1\}$$

These temporal constraints guarantee that if T_1 is activated and would threaten the labeled causal link from e_{P_1} to e_c, then it will be scheduled either before or after the labeled causal link. Two similar labeled temporal constraints are added for e_{T_2}, completing the threat resolutions for the case where e_{P_1} is chosen as the support. We also have similar constraints to the above corresponding to the case where e_{P_2} is chosen as the support. These constraints effectively implement similar threat resolution rules as POCL (Penberthy & Weld, 1992), though generalized for contingent plans.

4.4 Causal Link Extraction & Constructing the Augmented TPNU

In the previous sections, we have illustrated some examples of labeled causal link extraction, along with examples of constructing the augmented TPNU $\mathcal{T}'$ and constraints KB. In this section, we codify these examples by introducing appropriate algorithms. Pseudo code is shown in Algorithm 5, and a summary listing of all additional constraints is shown in Table 1.

Without loss of generality, we assume that there are two auxiliary events in the plan: (1) $e_{initial}$ which precedes all other events and has $\text{Effects}(e_{initial})$ set to the initial conditions, and (2) e_{goal} which succeeds all other events and has $\text{Preconditions}(e_{goal})$ set to the goal conditions. This encoding technique allows labeled causal links to be extracted from the initial conditions and goals without treating them separately from other events. Similar techniques have been used in other approaches (Muise, Beck, & McIlraith, 2016).

Our algorithm takes the following steps for each precondition p of each event e_c. First, it defines a new partial order relation over pairs of events for determining causal link dominance $<^{dom}_{e_c}$. It is defined as follows:

$$e_i <^{dom}_{e_c} e_j = \begin{cases} \text{True} & \text{if } e_j \prec e_i\Big|_{\varphi_c} \text{ and } e_i \prec e_c\Big|_{\varphi_j} \\ \text{False} & \text{otherwise} \end{cases}$$

This new relation will be helpful for determining labeled causal link dominance, through the use of an LVS, ξ (Line 4). This LVS ξ will use $<^{dom}_{e_c}$ as its relation $<_R$, and values will be events (as opposed to relation $<$ and values being numbers, as used earlier for temporal reasoning). If $e_i <^{dom}_{e_c} e_j$ and $\varphi_{e_j} \models \varphi_{e_i}$, it can be shown from the definition of labeled causal link dominance that a labeled causal link with producer e_i dominates a labeled causal link with producer e_j (both having consumer e_c). Therefore, a labeled value (e_i, φ_{e_i}) will dominate a different labeled value (e_j, φ_{e_j}) in ξ iff the labeled causal link with producer e_i dominates the labeled causal link with producer e_j. The environment entailment checks of the LVS take care of the first criterion required for labeled causal link dominance in Definition 4.5, and the $<^{dom}_{e_c}$ operator takes care of the second and third criteria.

During labeled causal link extraction, we add all potential producers, as well as all potential threats, to ξ. We therefore ensure that we find only the relevant, latest-occurring, non-dominated causal link producers and threats. Specifically, for each event e that produces either p or $\neg p$ in the plan, we add (e, φ_e) to ξ as long as $e_c \prec e$ does not hold (hence either $e \prec e_c$ or $e \parallel e_c$) (Line 7). The resulting ξ contains the minimal set of non-dominating possible producers and threats. For the example in Figure 11, $\xi = \{(e_{P_1}, \{u = 1\}), (e_{P_2}, \{v = 1\}), (e_{T_1}, \{x = 1\}), (e_{T_2}, \{y = 1\})\}$.

We then partition each event e in $(e, \varphi_e) \in \xi$ into those that produce p as an effect into set P (producers), and those that produce $\neg p$ into set T (threats) on Line 10.

Next, we make a new controllable choice variable s_{p,e_c} with domain $\{e_{P_1}, ..., \bot\}$ for each e_{P_i} in P. On Line 12, we add the following propositional constraint to KB:

$$\varphi_{e_c} \Rightarrow \bigvee_{e_{P_i} \in P} (s_{p,e_c} = e_{P_i} \wedge \varphi_{e_{P_i}})$$

Additionally, for each e_{P_i}, we check whether $e_{P_i} \prec e_c$. If so, then we continue (no additional temporal constraint is needed to order this producer before the consumer). Otherwise, it

Algorithm 5: ConstructAugmentedTPNUwithKB()

Input: TPNU $\mathcal{T} = \langle \mathcal{V}, \mathcal{E}, \mathcal{C}, \mathcal{A} \rangle$

Output: Augmented TPNU $\mathcal{T}' = \langle \mathcal{V}', \mathcal{E}, \mathcal{C}', \mathcal{A} \rangle$ with associated constraints KB

$\mathcal{T}' = \langle \mathcal{V}', \mathcal{E}, \mathcal{C}', \mathcal{A} \rangle \leftarrow$ copy of $\mathcal{T}$, KB $\leftarrow$ True
foreach $e_c \in \mathcal{E}$ **do**
    foreach $p \in$ Preconditions(e_c) **do**
        $\xi \leftarrow$ new LVS with relation $<^{dom}_{e_c}$
        foreach $e \in \mathcal{E} \setminus \{e_c\}$ *with* p *or* $\neg p$ *in* Effects(e) **do**
            if *not(*$e_c \prec e$*) and* $\varphi_e \wedge \varphi_{e_c}$ *is self consistent* **then**
                AddLVS($(e, \varphi_e), \xi$)
            end
        end
        Partition events from ξ into P (producers) and T (threats)
        Add new choice variable s_{p,e_c} to $\mathcal{V}'$ with domain $\{e_{P_1}, e_{P_2}, ..., \bot\}$ for $e_{P_i} \in P$
        KB $\leftarrow$ KB $\wedge \left(\varphi_{e_c} \Rightarrow \bigvee_{e_{P_i} \in P} (s_{p,e_c} = e_{P_i} \wedge \varphi_{e_{P_i}})\right)$
        foreach $e_{P_i} \in P$ *where not(*$e_{P_i} \prec e_c$*)* **do**
            $\mathcal{C}' \leftarrow \mathcal{C}' \cup \left(e_{P_i} \xrightarrow{[\epsilon,\infty]} e_c : \{s_{p,e_c} = e_{P_i}\} \wedge \varphi_{e_{P_i}} \wedge \varphi_{e_c}\right)$
        end
        foreach $e_{P_i} \in P$ **do**
            foreach $e_{T_j} \in T$ **do**
                $\varphi_C \leftarrow \{s_{p,e_c} = e_{P_i}\} \wedge \varphi_{e_{P_i}} \wedge \varphi_{e_{T_j}} \wedge \varphi_{e_c}$
                continue to next if φ_C is self inconsistent or temporally infeasible
                if $e_{P_i} \prec e_{T_j}\big|_{\varphi_C}$ *and* $e_{T_j} \prec e_c\big|_{\varphi_C}$ **then**
                    KB $\leftarrow$ KB $\wedge \left(\neg\varphi_C\right)$ ▷ Avoid definite threat
                else if $e_{P_i} \prec e_{T_j}\big|_{\varphi_C}$ *and* $e_{T_j} \parallel e_c\big|_{\varphi_C}$ **then**
                    $\mathcal{C}' \leftarrow \mathcal{C}' \cup \left(e_c \xrightarrow{[\epsilon,\infty]} e_{T_j} : \varphi_C\right)$ ▷ Force threat after consumer
                else if $e_{P_i} \parallel e_{T_j}\big|_{\varphi_C}$ *and* $e_{T_j} \prec e_c\big|_{\varphi_C}$ **then**
                    $\mathcal{C}' \leftarrow \mathcal{C}' \cup \left(e_{T_j} \xrightarrow{[\epsilon,\infty]} e_{P_i} : \varphi_C\right)$ ▷ Force threat before producer
                else if $e_{P_i} \parallel e_{T_j}\big|_{\varphi_C}$ *and* $e_c \parallel e_{T_j}\big|_{\varphi_C}$ **then**
                    ▷ Force threat either before or after
                    Add new choice variable $o_{p,e_c,e_{P_i},e_{T_j}}$ to $\mathcal{V}'$ with domain $\{-1, +1\}$
                    $\mathcal{C}' \leftarrow \mathcal{C}' \cup \left(e_{T_j} \xrightarrow{[\epsilon,\infty]} e_{P_i} : \{o_{p,e_c,e_{P_i},e_{T_j}} = -1\} \wedge \varphi_C\right)$
                    $\mathcal{C}' \leftarrow \mathcal{C}' \cup \left(e_c \xrightarrow{[\epsilon,\infty]} e_{T_j} : \{o_{p,e_c,e_{P_i},e_{T_j}} = +1\} \wedge \varphi_C\right)$
                end
            end
        end
    end
end

	$e_{T_j} \prec e_{P_i}\big\|_{\varphi_C}$	$e_{T_j} \parallel e_{P_i}\big\|_{\varphi_C}$	$e_{P_i} \prec e_{T_j}\big\|_{\varphi_C}$
$e_{T_j} \prec e_c\big\|_{\varphi_C}$	C1: Nothing required.	C2: Force e_{T_j} before e_{P_i} to resolve threat.	C3: Definite threat: avoid through conflict.
$e_c \parallel e_{T_j}\big\|_{\varphi_C}$	C4: Impossible.	C5: Force e_{T_j} either before e_{P_i} OR after e_c via choice variable.	C6: Force e_{T_j} after e_c to resolve threat.
$e_c \prec e_{T_j}\big\|_{\varphi_C}$	C7: Impossible.	C8: Impossible.	C9: Nothing required.

Figure 12: Labeled causal link threat resolution cases. Moving across horizontally, we have the three possible precedence relations between e_{P_i} and e_{T_j}. Vertically, we have the three possible relations between e_{T_j} and e_c. The environment $\varphi_C = \{s_{p,e_c} = e_{P_i}\} \wedge \varphi_{e_{P_i}} \wedge \varphi_{e_c} \wedge \varphi_{e_{T_j}}$.

is possible for e_{P_i} to occur after e_c; we prevent this by adding to $\mathcal{C}'$ the labeled temporal constraint $e_{P_i} \longrightarrow e_c, [\epsilon, \infty] : \{s_{p,e_c} = e_{P_i} \wedge \varphi_{e_c} \wedge \varphi_{e_{P_i}}\}$ on Line 14.

We must also resolve any potential threats of this labeled causal link. Based on the different possible precedence relations amongst the consumer e_c, producer e_{P_i}, and threat e_{T_j}, different constraints may be added to $\mathcal{T}'$ and KB. A case-based analysis is depicted in Figure 12. Depending on the case, either a conflict environment is added, or additional labeled temporal constraints are added to resolve the threat. We iterate over all pairs of potential producer events e_{P_i} and all potential threat events e_{T_j} (Line 16). The context environment is set such that all of e_c, e_{P_i}, and e_{T_j} are activated, and e_{P_i} is selected as the supporting labeled causal link. Hence, $\varphi_C = \{s_{p,e_c} = e_{P_i}\} \wedge \varphi_{e_{P_i}} \wedge \varphi_{e_c} \wedge \varphi_{e_{T_j}}$. We evaluate all precedence relations with respect to this context environment.

In case C1 above, e_{T_j} is not a threat. Since $e_{T_j} \prec e_{P_i}\big|_{\varphi_C}$ and $e_{P_i} \prec e_c\big|_{\varphi_C}$, the threat will not interfere with the causal link. The predicate in question is negated before the producer event asserts it. Similarly in C9, the threat in question will occur after the consumer event. Therefore, we can be sure that the threat event will not interfere.

Cases C4, C7, and C8 represent impossible situations if φ_C is a temporally consistent environment (which is checked on Line 19). Since we add appropriately labeled ordering constraints ensuring that producers precede consumers, we will have that $e_{P_i} \prec e_c\big|_{\varphi_C}$. In cases C4 and C7, we also have that $e_{T_j} \prec e_{P_i}\big|_{\varphi_C}$, so we can derive that $e_{T_j} \prec e_c\big|_{\varphi_C}$. This however contradicts the premises in cases C4 and C7. Similar reasoning holds for case C9.

In C3, the threat is guaranteed to occur temporally between the causal link producer and the consumer. This means that the potential threat is in fact a definite threat, and there is nothing that can be done about it except to avoid it. Therefore, we add a conflict disallowing φ_C to KB (Line 21).

In C2, we know that the threat precedes the consumer, but it is incomparable to the producer (which also precedes the consumer). Therefore, we can resolve this threat by adding a labeled temporal ordering constraint forcing the threat to occur *before* the producer event (Line 25). If this results in a temporally feasible plan after re-running the APSP, we have resolved the threat. If not, we will extract a temporal conflict similar to C3 above.

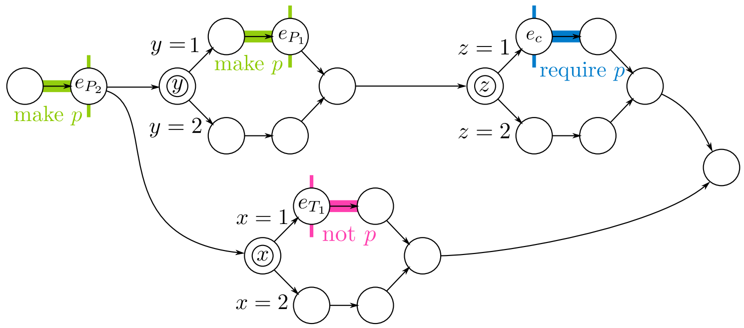


Figure 13: An example TPNU.

Similar reasoning also holds for C6, in which we know that the threat must occur after the producer, but could potentially occur before the consumer e_c. We therefore resolve this threat by forcing the threat to occur *after* the consumer event via an additional labeled temporal constraint (Line 23).

Case C5 is the most complicated, and involves adding an additional controllable choice variable. In C5, the threat event can occur anywhere temporally; it could occur before the producer, between the producer and the consumer, or after the consumer. We wish to move the threat so that it occurs either before or after. We hence add a new controllable choice variable $o_{p,e_c,e_{P_i},e_{T_j}}$ with domain $\{-1,+1\}$ to $\mathcal{V}'$. If $o_{p,e_c,e_{P_i},e_{T_j}} = -1$, we choose for the threat to come before e_{P_i}; otherwise, we choose for it to come after e_c. We add two new temporal constraints to enforce this (Line 27):

- $e_{T_j} \xrightarrow{[\epsilon,\infty]} e_{P_i} : \{o_{p,e_c,e_{P_i},e_{T_j}} = -1\} \wedge \varphi_C$
- $e_c \xrightarrow{[\epsilon,\infty]} e_{T_j} : \{o_{p,e_c,e_{P_i},e_{T_j}} = +1\} \wedge \varphi_C$

If φ_C holds, exactly one of the above two constraints will be activated. This forces the executive to schedule the threat either before or after the labeled causal link interval, thereby resolving it.

Note that we could always do case C5, i.e., for cases C2, C3, and C6. We avoid this however for efficiency, so that we do not need to add unnecessary choice variables and temporal constraints to the problem. This is the key motivation for such case-based reasoning.

This concludes our description of the algorithms that extract dominating labeled causal links and use them generate an augmented TPNU $\mathcal{T}'$ along with the propositional constraints KB. We provide one more grounded example to illustrate. In the TPNU shown in Figure 13, there is one consumer event and two preceding producer events. There is additionally a potential threat that is ordered after the first producer, but may come either before or after the second producer.

The result of running Algorithm 5 on this example will be the following additional constraints for $\mathcal{T}'$ and KB:

- Propositional: $z = 1 \Rightarrow (s_{p,e_c} = e_{P_2}) \vee (s_{p,e_c} = e_{P_1} \wedge y = 1)$
- Temporal: $e_c \xrightarrow{[\epsilon,\infty]} e_{T_1} : \{z = 1, x = 1, s_{p,e_c} = e_{P_2}\}$

- Temporal: $e_{T_1} \xrightarrow{[\epsilon,\infty]} e_{P_1} : \{z = 1, y = 1, x = 1, s_{p,e_c} = e_{P_1}, o_{p,e_c,e_{P_1},e_{T_1}} = -1\}$
- Temporal: $e_c \xrightarrow{[\epsilon,\infty]} e_{T_1} : \{z = 1, y = 1, x = 1, s_{p,e_c} = e_{P_1}, o_{p,e_c,e_{P_1},e_{T_1}} = +1\}$

4.5 Ensuring Correct Partial Executions

We show in this section that, when Pike uses the augmented TPNU $\mathcal{T}'$ and KB to guide execution, the resulting partial execution will remain correct. This is a key property of Pike, and a central goal of using the augmented TPNU. In the context of human-robot interaction, this means that Pike will make controllable choices and scheduling decisions such that there is always some way for the plan to succeed (i.e., some way for the human and robot team to make choices that achieve the plan goals). We introduce a series of theorems (proved in Appendix A), culminating in Theorems 4.7 and 4.4, showing that, when Pike executes $\mathcal{T}'$ with KB, it will make online decisions properly in this manner.

We begin by formalizing the relationship between $\mathcal{T}$ and $\mathcal{T}'$ in terms of correct executions and partial executions.

Theorem 4.3 (Executions from $\mathcal{T}$ and $\mathcal{T}'$ correspond)**.** *An execution $\langle \varphi_S, T_{\varphi_S} \rangle$ of the original TPNU $\mathcal{T}$ is correct if and only if there exists a corresponding correct execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of the augmented TPNU $\mathcal{T}'$ where $\varphi'_S \models \varphi_S$.*

Proof. See Appendix A. □

Theorem 4.4 (Correct Executions of Original $\mathcal{T}$)**.** *Let $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ be a partial execution at some point during execution of the augmented TPNU $\mathcal{T}'$. This partial execution is correct with respect to the augmented TPNU $\mathcal{T}'$ if and only if it is also correct with respect to the original TPNU $\mathcal{T}$.*

Proof. See Appendix A. □

Theorem 4.3 maps correct executions of $\mathcal{T}$ to at least one correct execution in $\mathcal{T}'$ with an identical schedule. The relationship between these two sets is a surjection; every correct execution of $\mathcal{T}$ maps to at least one (possibly multiple) executions of $\mathcal{T}'$, corresponding to different allowable assignments to the s_{p,e_c} or o_{p,e_c,e_P,e_T} variables. Relatedly, in Theorem 4.4, we make a similar connection in terms of partial executions.

The following two theorems highlight key properties of $\mathcal{T}'$, and are visualized by Figure 10b. Theorem 4.5 asserts that every temporally consistent execution of $\mathcal{T}'$ is also correct. Crucially, this property does not hold for $\mathcal{T}$, which could admit temporally consistent yet causally incomplete executions. By constructing $\mathcal{T}'$ and KB, we ensure that Pike can achieve any temporally consistent execution and rest assured that it will also be causally complete.

Theorem 4.5 (Temporally consistent executions of $\mathcal{T}'$ with KB are also causally complete, and correct)**.** *Let $\langle \varphi'_S, T_{\varphi_S} \rangle$ be a temporally consistent execution of $\mathcal{T}'$ where φ'_S satisfies KB. Then this execution is also causally complete (and hence correct).*

Proof. See Appendix A. □

Theorem 4.6 describes the space of candidate subplans whose variable assignments satisfy KB. We can therefore think of the solutions of KB as representing the space of all candidate subplans that admit a correct execution.

Theorem 4.6 (φ'_S satisfies KB iff correct)**.** *φ'_S satisfies KB if and only if there exists a correct execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}$'.*

Proof. See Appendix A. □

Based on these theorems, we can prove that the CanExecuteEventNow? procedure, which is crucial for online execution correctness and is shown in Algorithm 2, is correct. Specifically, it will allow an event to be executed and controllable choices to be made by Pike if and only if the resulting partial execution would remain correct.

Theorem 4.7 (CanExecuteEventNow? is correct)**.** *Let $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ be the current, correct partial execution of $\mathcal{T}'$. Then* CanExecuteEventNow?(e_i, t) *returns* True *at time t if and only if the partial execution of $\mathcal{T}'$ that would result if e_i is executed at time t – namely $\langle \varphi_{ex} \wedge \varphi_{e_i},\ \tilde{T}_{\varphi_{ex}} \cup \{e_i = t\} \rangle$ – is correct.*

Proof. See Appendix A. □

By Theorem 4.7, Pike's controllable choices and scheduling decisions keep the partial execution correct for $\mathcal{T}'$, the TPNU being executed. By Theorem 4.4, correct partial executions of $\mathcal{T}'$ correspond to correct partial executions of $\mathcal{T}$. Therefore, we have proven that Pike will maintain a correct partial execution of $\mathcal{T}$ – a central goal of Pike in ensuring that its choices can match the human's possible intentions.

This concludes our discussion of labeled causal link extraction, and the generation of the augmented TPNU $\mathcal{T}'$ and corresponding constraints. Next, we discuss how to compile the constraints for efficient use during online execution.

5. Constraint Compilation

In this section, we describe a method to compile these constraints into a knowledge base KB for efficient online execution. The goal of this knowledge compilation is to compactly represent all possible candidate subplans φ'_S that admit a correct execution for the human-robot team, and to allow certain queries about these candidate subplans to be computed for efficient online execution. Per Theorem 4.6, this means that we must compactly represent the space of all solutions of KB. Towards this goal, we employ a well-known technique from the knowledge compilation community: prime implicants. We briefly introduce the πTMS, a label propagation mechanism building upon the Assumption-based Truth Maintenance System (ATMS) that generates a sound and complete set of prime implicants for our theory incrementally. The resulting prime implicants can be used to efficiently implement the query operations on KB needed for online execution, as discussed shortly. Note that while the size of KB may in the worst case grow exponentially with the size of the TPNU, the use of prime implicants offers significant gains in compactness (as shown later by Figure 21).

We begin with background about the ATMS and prime implicants. We then illustrate the key algorithms we use to generate prime implicants.

5.1 Background

In this section, we introduce background information about the ATMS and prime implicants.

5.1.1 ATMS

The Assumption-based Truth Maintenance System, or ATMS, is a knowledge base that allows a problem solver to efficiently reason about facts without prematurely committing to them (de Kleer, 1986a). The ATMS introduces assumptions – facts whose certainty is not known and may be changed by the problem solver with little overhead. This fast context switching is taken advantage of during online execution, when querying which choices can consistently be made online.

We take a propositional logic viewpoint of the ATMS and introduce some terminology. A *node* N is logically equivalent to a proposition. An *assumption* is a special type of node whose truthfulness may be unknown beforehand. A node that represents False is called a *contradiction node*. A *justification* is logically equivalent to the implication $N_{A_1} \wedge ... \wedge N_{A_k} \Rightarrow N_C$ where each node N_{A_i} is an *antecedent* and N_C is the *consequent*. A *nogood* or *conflict* is a conjunction that entails an inconsistency; its resolution is logically equivalent to $\neg(N_{C_1} \wedge ... \wedge N_{C_k})$. An environment is a conjunction of assumption propositions (though it is often denoted as a set for convenience).

An environment φ_e *manifests* a conflict φ_C iff $\varphi_e \models \varphi_C$. An environment φ_e *resolves* a conflict φ_C iff $\varphi_e \models \neg\varphi_C$ (Williams & Ragno, 2007). An environment may manifest a conflict, resolve it, or neither. If φ_e manifests a conflict, then any environment φ whose assignments are a superset of φ_e (i.e., $\varphi \models \varphi_e$) also manifests that conflict. Similar logic holds for resolving a conflict. However, if an environment φ_e neither resolves nor manifests a conflict, then there exists some superset environment that manifest the conflict, and some other superset that resolves it.

Each node in an ATMS is associated with a *label*, which is a set (disjunctive) of environments. A node *holds* in environment φ iff it can be propositionally derived from φ and all justifications in the ATMS, and False cannot be propositionally derived via a conflict (Forbus, 1993). An environment φ_e in node N's label means that N will hold in any superset environment $\varphi \models \varphi_e$ – except those manifesting a conflict. A label is *minimal* if no environment in it is a subset of any other environment (i.e., no environment entails any other). The ATMS enforces that no environment in an ATMS label may manifest a conflict, but it is possible for supersets of the environment to manifest conflicts (i.e., environments in a label need not resolve all conflicts). The ATMS employs label propagation algorithms to ensure that a sound, complete, and minimal set of environments is incrementally derived for each node's label, given the justifications and conflicts encoded so far (Forbus, 1993).

5.1.2 Prime Implicants

An *implicant* φ of a propositional theory $\mathcal{C}$ is a conjunction of variables (i.e., an environment) that entails the theory, $\varphi \models \mathcal{C}$. A *prime implicant* φ_P is an implicant that is minimal in size; no subset of assignments of φ_P themselves form an implicant. The intuition for this is that if φ is an implicant, then any superset must also be an implicant. We therefore need to only maintain the smallest ones, namely the prime implicants. There is also the related concept of an implicate and prime implicate. An *implicate* θ of a theory $\mathcal{C}$ is a disjunction which logically follows from the theory, namely $\mathcal{C} \models \theta$.

While there are many different knowledge compilation techniques that could be successfully applied to this task (Darwiche & Marquis, 2002), we choose to pursue prime implicants in this work. We are motivated to do so for two several reasons, including: (1) the potential to be more compact than full model enumeration, (2) the ability to compute the relevant consistency queries online in polynomial time in the number of prime implicants, and (3) the ability to update a prime implicant database efficiently (via bounded conjunctions in polynomial time) (Darwiche & Marquis, 2002), a necessary operation for Pike's online execution when we add new constraints to KB. Note that while it is possible for the number of prime implicants to exceed the number of models for a theory (Quine, 1959), our aim here is to improve the compactness of our solution representation so as to improve the efficiency of computing Pike's online queries. Experimental results verify that prime implicants are indeed generally more compact for our problem structure (see Figure 21).

5.2 The πTMS

We introduce the πTMS (short for Prime Implicant TMS) – a technique for incrementally and efficiently computing the prime implicants of a theory over finite-domain variables. Like the ATMS, the πTMS uses label propagation to store a set of environments in a label associated with each node. The πTMS, however, uses a form of consensus over finite-domain variables within each label to generate prime implicants and more compactly represent the solution space.

We wish for labels in the πTMS to contain prime implicants of associated theories. To achieve this, we modify the semantics of labels. In an ATMS, no environment in a label may *manifest* any conflicts. The πTMS strengthens this, by requiring that each environment in a label *resolve* all known conflicts. The difference between these semantics lies in the supersets of an environment: the ATMS permits supersets of environments to manifest conflicts, but the πTMS does not. If some environment φ_e in an ATMS label has a superset φ'_e that manifests a conflict of the theory, then φ_e cannot be an implicant of the theory. In the πTMS, we therefore modify the label semantics to guarantee that all environments in a node's label must resolve all conflicts, thereby guaranteeing that all supersets also resolve all conflicts.

5.3 Encoding Constraints in the πTMS

In this section, we discuss how we take a theory $\mathcal{C}$ of constraints over finite-domain variables and hierarchically encode it into the πTMS. With our encoding, each constraint in $\mathcal{C}$ is represented by a node in the πTMS, such that the node is propositionally derivable iff the constraint holds. In the earlier example shown in Figure 13, recall that causal link extraction process derived the constraint $z = 1 \Rightarrow (s_{p,e_c} = e_{P_2}) \vee (s_{p,e_c} = e_{P_1} \wedge y = 1)$.

To encode this constraint in the πTMS, it is broken down hierarchically into a series of nodes (each representing a subformulae) where each holds if the corresponding constraint holds. This is similar in spirit to the well-known Tseitin transformation (Tseitin, 1968), in which additional variables represent the decomposed portions of constraints. πTMS nodes are propositionally derivable iff the associated decomposed constraints hold. We

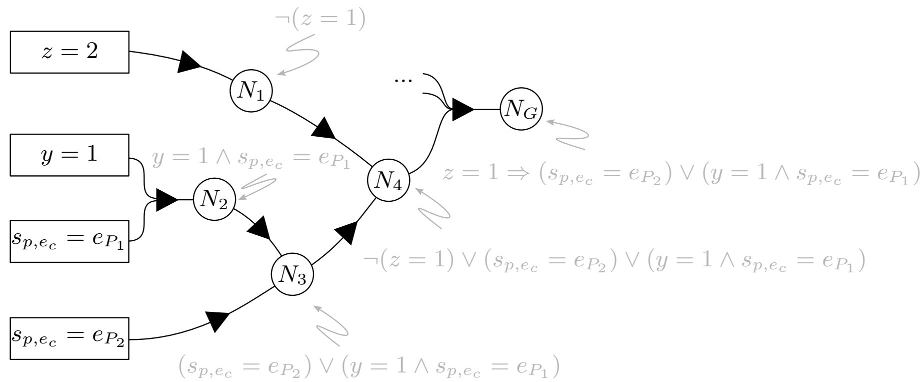


Figure 14: Hierarchical encoding of constraint in πTMS. Following the notation introduced by Forbus (1993), boxes represent assumptions, circles represent nodes, and arrows represent justifications with the multiple tails representing a conjunction. Gray annotations represent the constraints represented by the associated nodes.

recursively traverse the constraints starting from its top-level conjunction, then following the logical order of operations and applying De Morgan's law as appropriate, to create more nodes hierarchically. The example constraint presented earlier would be encoded into πTMS via the justifications illustrated in Figure 14. The node N_G is the overarching "goal" representing our entire theory, in this case the single constraint we have extracted (in general, more constraints would be conjoined for N_G). The example implication is first converted to the equivalent disjunction $\Big(\neg(z = 1)\Big) \vee \Big((s_{p,e_c} = e_{P_2}) \vee (s_{p,e_c} = e_{P_1} \wedge y = 1)\Big)$, which can be represented by a node N_4 and the two disjuncts inside by nodes N_1 (representing $\neg(z = 1)$) and N_3 (representing $(s_{p,e_c} = e_{P_2}) \vee (s_{p,e_c} = e_{P_1} \wedge y = 1)$). We can then encode the disjunction via the two implications $N_1 \Rightarrow N_4$ and $N_3 \Rightarrow N_4$ as the only justifications in the πTMS for N_4. This guarantees that N_4 will be propositionally derivable iff either N_1 or N_3 holds, thus capturing the disjunction. We can use a similar technique to encode a conjunction. By justifying a node N_2 with the implication $y = 1 \wedge s_{p,e_c} = e_{P_1} \Rightarrow N_2$ as the only justification for N_2, we guarantee that N_2 will be propositionally derivable iff both $y = 1$ and $s_{p,e_c} = e_{P_1}$ hold – a conjunction. A negation of an assignment can be encoded by creating a disjunction of all other variable domain values. For instance, the N_1 node (which represents $\neg(z = 1)$) can here be justified with the single implication $z = 2 \Rightarrow N_1$, guaranteeing that N_1 will be derivable only if $\neg(z = 1)$ holds.

The one exception to this hierarchical decomposition rule applies to conflicts. When encoding a conflict, i.e. a negation of a conjunction of assignments, we instead make use of contradiction nodes (which represent False) originating from the ATMS. Specifically, we create a new contradiction node, and add a single justification with this contradiction as the consequent. The antecedents contain the assignments present in the conflict.

5.4 Incrementally Computing Prime Implicants via Consensuses

Here, we describe our approach to compactly computing prime implicants: consensuses.

Each node's label is expressed in disjunctive normal form (DNF) – i.e., it is a disjunction of environments, which are themselves conjunctions. The following Lemma can be used repeatedly to compute implicants of the DNF. The resulting implicants are smaller than the environments used to generate it, and will eventually result in computing prime implicants when used repeatedly.

Lemma 5.1 (Consensus for Finite Domain Variables)**.** *Suppose theory $\mathcal{C}$ can be expressed in DNF in the form* $\left[\bigvee_{i=1...N} (\varphi_i \wedge x = v_i)\right] \vee \ldots$ *for some finite-domain variable x with domain $v_1, ..., v_N$. Then* $\bigwedge_{i=1...N} \varphi_i$*, if self-consistent, is an implicant of $\mathcal{C}$; i.e.:*

$$\bigwedge_{i=1...N} \varphi_i \models \bigvee_{i=1...N} (\varphi_i \wedge x = v_i)$$

Proof. $\bigwedge_{i=1...N} \varphi_i \models \varphi_j$ for all $j = 1 \ldots N$. Hence, each of the φ_i in the disjunction are be entailed. Since x must take some value from its domain, exactly one of the $(\varphi_i \wedge v_i)$ terms must hold, causing the disjunction to be satisfied. □

This result can be seen as the finite-domain generalization of the consensus theorem (Kean & Tsiknis, 1990) for boolean variables:

$$\varphi_1 \wedge \varphi_2 \models (\varphi_1 \wedge x) \vee (\varphi_2 \wedge \neg x)$$

Suppose we have a node N with label $L : \{\{x = 1, y = 1, z = 1\}, \{x = 2, y = 1, z = 1\}, \{x = 2, y = 3\}, \{x = 3\}\}$, where each variable x, y, and z has a domain of $\{1, 2, 3\}$. We can express L logically in DNF as

$$(x = 1 \wedge y = 1 \wedge z = 1) \ \vee (x = 2 \wedge y = 1 \wedge z = 1) \vee$$
$$(x = 2 \wedge y = 3) \ \vee (x = 3)$$

Lemma 5.1 applies to this DNF for the variable x and its three domain values $1, 2, 3$, where $\varphi_1 : y = 1 \wedge z = 1$, $\varphi_2 : y = 1 \wedge z = 1$ (note that $\varphi_2 : y = 3$ could equivalently be chosen), and φ_3 : True. Therefore, $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$, namely $y = 1 \wedge z = 1$, is an implicant of the theory. This implicant may be added disjunctively to the label L as a new environment, as it will not change its semantics (i.e., the set of represented team scenarios will not change). By "minimally," we refer to the ATMS operation in which an environment φ_i is added to a list L of environments: any $\varphi_j \in L$ that entails φ_i is removed from L, and φ_i is added to L only if it is does not entail any other $\varphi_j \in L$. In our example, the first two terms in the DNF are entailed by the new implicant, so it is added minimally and those terms removed. The resulting label L may be represented more compactly with just three environments as $\{\{y = 1, z = 1\}, \{x = 2, y = 3\}, \{x = 3\}\}$.

It is important to note that not all consensuses yield a useful implicant. For example, if we had chosen $\varphi_3 : y = 3$, the resulting consensus would have been $y = 1 \wedge y = 3 \wedge z = 1$. This is not self-consistent due to the conflicting variable assignment for y. While False is indeed an implicant of any theory, it does not provide any useful information in our case, so we do not consider it.

5.5 πTMS Label Propagation Algorithms

Now that we have illustrated both the hierarchical node structure of the πTMS as well as the intuition behind computing prime implicants from an existing label, we put the two together to describe the πTMS label propagation algorithms.

We begin by describing the semantics of each node's label. Intuitively, the label of a node contains a set of prime implicants for the theory containing that node's constraint, as described by Invariant 5.2 below:

Invariant 5.2 (Labels of πTMS Contain Prime Implicants)**.** *Let node N represent constraint C in theory $\mathcal{C}$. Then, the label of N contains a sound and complete set of prime implicants for a new theory $\mathcal{C}'$, which consists of the constraint C as well as all conflict constraints in $\mathcal{C}$.*

We illustrate this by continuing with the same example as before. Since there are no conflicts in the theory, the label of node N_4 will contain the complete set of prime implicants for the theory containing just the single constraint: $\{z = 2\}$, $\{s_{p,e_c} = e_{P_2}\}$, and $\{y = 1, s_{p,e_c} = e_{P_1}\}$. Any assignment of variables containing the assignment $z = 2$ will entail the causal link constraint being satisfied. Consider the node N_3, which represents the constraint $(s_{p,e_c} = e_{P_2}) \vee (y = 1 \wedge s_{p,e_c} = e_{P_1})$. By Invariant 5.2, the label for N_3 contains the prime implicants of a theory containing just this constraint. The computed label for N_3 is $\{s_{p,e_c} = e_{P_2}\}$, $\{y = 1, s_{p,e_c} = e_{P_1}\}$.

As another example, suppose that during online execution, Pike adds the conflict $\neg(s_{p,e_c} = e_{P_2})$. This could happen, for example, if the causal link is detected to be violated online. In that case, the theory we wish to encode is

$$z = 1 \Rightarrow (s_{p,e_c} = e_{P_2}) \vee (s_{p,e_c} = e_{P_1} \wedge y = 1)$$
$$\neg(s_{p,e_c} = e_{P_2})$$

Note in this example that we have a conflict. All prime implicants must resolve this conflict. The computed label for N_G contains prime implicants $\{z = 2, s_{p,e_c} = e_{P_1}\}$, $\{z = 2, s_{p,e_c} = \perp\}$, and $\{y = 1, s_{p,e_c} = e_{P_1}\}$.

In the first example, it is no coincidence that the label of N_3 is a subset of the label of N_4. The ATMS label propagation algorithms – from which the πTMS derives – allow environments to flow through the constraint structure to compute the labels for other nodes hierarchically. The label of any assumption node of a πTMS contains a single environment containing the associated variable assignment. This is the "base case," as that variable assignment is an implicant of the theory containing just that assignment. We may then "build up" hierarchically. For a conjunction (shown visually in Figure 14 as a justification with two tails), we may take all possible combinations of implicants of the subtheories (from each label) and conjoin them to form an implicant of the new theory – this is accomplished by the ATMS Weave operation (de Kleer, 1986a). For a disjunction, we may simply merge the implicants of the subtheories.

The πTMS label propagation algorithms differ from the ATMS procedures in that they compute prime implicants at every label propagation update – thus aiming to keep propagation to a minimum and find prime implicants early on during propagation. Upon adding a new environment to a label, all possible consensuses (which are implicants of the theory) are computed and added to the label per the above illustration. Note that this does not change the team scenarios represented by that label's prime implicants, but it can make the label more compact. Sometimes, new implicants can be computed based on the implicants that were just added – so this process repeats iteratively until no new implicants can be added to a label. We keep track of the new implicants that were ultimately added and only propagate those implicants forward. This can have the effect of reducing unnecessary propagations. The resulting label is generally more compact than that of the standard ATMS, and can oftentimes be computed faster as well. For domains in which the number of models which would be computed by the ATMS is prohibitively large, the πTMS is often able to tractably complete the computation and takes orders of magnitude less space and time (see experimental results and Figure 21).

The illustrated approach is based on a generalized form of consensus. Yet since consensus is the dual of resolution, we can also view the above implicant generation in a different light. Hyper-resolution in boolean logic (Robinson, 1965) is a generalization of resolution for performing multiple resolutions in CNF in a single step, re-stated below (in its negative form):

$$\frac{\begin{array}{c} C_1 \vee \neg x_1 \\ C_2 \vee \neg x_2 \\ \ldots \\ C_n \vee \neg x_n \\ x_1 \vee x_2 \vee \ldots \vee x_n \vee D \end{array}}{\bigvee_i^n C_i \vee D}$$

Each x_i is positive literal, and each C_i is a disjunctive clause of literals. The derived conclusion $\bigvee_i^n C_i \vee D$ is the *hyper-resolvent* and is an implicate (as opposed to an implicant in consensus), the disjunction $x_1 \vee x_2 \vee \ldots \vee x_n \vee D$ is the *nucleus*, and each clause $C_i \vee \neg x_i$ is a *satellite*.

To view consensus as the dual of hyper-resolution, we exploit the fact that computing *implicants* (conjunctive) of a theory $\mathcal{C}$ can be accomplished by negating *implicates* (disjunctive) of the theory's negation $\neg\mathcal{C}$ (Elliott, 2004). Continuing with the same example, we can view the implicant $y = 1 \wedge z = 1$ through the lens of hyper-resolution by computing its associated implicate of $\neg\mathcal{C}$. As $\mathcal{C}$ is expressed in DNF, we can easily express $\neg\mathcal{C}$ in CNF by negating each literal and swapping $\wedge$ and $\vee$:

$$\begin{array}{r} (\neg(x = 1) \vee \neg(y = 1) \vee \neg(z = 1)) \wedge \\ (\neg(x = 2) \vee \neg(y = 1) \vee \neg(z = 1)) \wedge \\ (\neg(x = 2) \vee \neg(y = 3)) \wedge \\ (\neg(x = 3)) \end{array}$$

Since we operate over finite-domain variables, each of which must take a domain value, we can add in the extra clause on the last line below and perform hyper-resolution:

$$\begin{array}{r} (\neg(x=1) \vee \neg(y=1) \vee \neg(z=1)) \wedge \\ (\neg(x=2) \vee \neg(y=1) \vee \neg(z=1)) \wedge \\ (\neg(x=2) \vee \neg(y=3)) \wedge \\ (\neg(x=3)) \wedge \\ (x=1 \vee x=2 \vee x=3) \\ \hline \\ \neg(y=1) \vee \neg(z=1) \end{array}$$

The first, second, and fourth clauses are the satellites, the nucleus is the final disjunction $x = 1 \vee x = 2 \vee x = 3$, and the hyper-resolvent is $\neg(y = 1) \vee \neg(z = 1)$, which is an implicate of $\neg\mathcal{C}$. Therefore, its negation $y = 1 \wedge z = 1$ is an implicant of $\mathcal{C}$. Effectively, we are taking advantage of the fact that each variable must take a value from its domain, allowing us to perform hyper-resolution over this disjunction.

Previous work has examined the problem of incorporating hyper-resolution into the ATMS, and argued the complete case intractable (de Kleer, 1986b). However, we argue that our approach in this work, which is incremental and seeks to minimize the number of resolutions performed through efficient propagation, is computationally feasible in many situations. Experimental results show that the πTMS is tractable for many of Pike's tested problem instances.

5.6 πTMS Algorithms

Here, we introduce the πTMS label propagation algorithms that modify those of the ATMS.

Given a label L and an assignment, $x = v_i$, we define the *candidate satellites* $L_{sat}^{x=v_i}$ as a set consisting of all environments in L that assign $x = v_i$, except with that single assignment stripped off the environment. These correspond to the satellites in hyperresolution, removing the nucleus. Taking the same example as earlier where $L = \{\{x = 1, y = 1, z = 1\}, \{x = 2, y = 1, z = 1\}, \{x = 2, y = 3\}, \{x = 3\}\}$, the result of computing $L_{sat}^{x=1}$ would be $\{\{y = 1, z = 1\}\}$, corresponding to φ_1 described earlier. Computing $L_{sat}^{x=2}$ yields $\{\{y = 1, z = 1\}, \{y = 3\}\}$, which are both of the possible φ_2 values described earlier.

The actual work of computing the implicants occurs by computing $L_{sat}^{x=v_1} \times L_{sat}^{x=v_2} \times ... \times L_{sat}^{x=v_k}$ for each $v_i \in$ Domain(x). One environment from each set of candidates satellites must be chosen, and the result joined conjunctively together if consistent. This is accomplished via the cross product operation $\times$. In a nutshell, the $\times$ operation finds the cross product of all possible environments incrementally, prunes inconsistent combinations early before they are fully constructed, and attempts to maintain compactness by only keeping track of minimal, non-entailing environments. The algorithm is similar to the ATMS Weave operation (Forbus, 1993). Please note that this can, in the worst case, result in a combinatorial explosion of consensuses. Crucially, each of the results of the cross product is an implicant of the DNF formula represented by L. Some (but not all) could be prime implicants, and others could entail other terms in L.

Now that we have described how consensuses can be computed for a single variable, we describe how these algorithms are integrated into the overall label propagation algorithm,

Algorithm 6: πTMSUpdate(a, L)

Input: A node a with associated label L_a, and a list of new environments L to be added to a's label

```
if a is a contradiction node then
    Add each e ∈ L minimally to nogoods
    foreach φ_N actually added to nogoods above do
        SplitAllLabelsOnConflict(φ_N)
    end
else
    L_added = AddAndGeneratePrimeImplicants(L_a, L)
    foreach justification J where a is an antecedent do
        Propagate(J, a, L_added)
        [Optional: Split L_added on any newly-added nogoods]
    end
end
```

forming the πTMS. The πTMS borrows the ATMS Propagate and Weave algorithms unchanged, and changes the Update method to the new πTMSUpdate, which incorporates prime implicant generation (Forbus, 1993). Intuitively, Update is called when a new environment is added to a node's label. This in turn triggers propagation to other ATMS nodes via justifications, and the Weave method is called to combine environments conjunctively for this propagation.

Instead of solely adding the new environments to a label minimally as the original Update method does, πTMSUpdate computes prime implicants via the above consensus procedures given the new environments, possibly resulting in new, smaller environments being produced that are entailed by the ones being originally added. Only these entailed environments are added and propagated forward, not the entailing ones – thus reducing unnecessary computation.

Pseudo code for πTMSUpdate(a, L) is shown in Algorithm 6. There are two cases: (1) if node a is a contradiction node, and (2) if it is not. If a is not a contradiction node, then we add new environments to a's label, and also compute new prime implicants for the theory rooted at a by calling AddAndGeneratePrimeImplicants. These new prime implicants can also be added to a, but may entail other environments originally in L. The procedure returns a list L_{added} of environments that were ultimately added to L_a (including new implicants derived) to be propagated forwarded in the similar spirit as the ATMS. Otherwise in case (2) if a is a contradiction node, then all of the environments in L are conflicts. In this case, we split all labels in the πTMS, by ensuring that each one resolves the conflicts. This can be accomplished in a manner similar to that of Conflict-Directed A* (Williams & Ragno, 2007). To split an environment in a label on a conflict, we do nothing if that environment resolves the conflict. If the environment manifests the conflict, we remove it from the label. Otherwise, we create a new series of environments, each containing additional assignments that conflict with conflict. Each of these new environments resolves the conflict. Note that this can at times result in a significant increase in the number of environments in a label.

Algorithm 7: AddAndGeneratePrimeImplicants(L, L_+)

Input: A label L, and a label L_+ to be added to L.
Output: Adds the environments from L_+ to L, along with any new consensuses. Returns a label containing all environments actually added to L.

$Q \leftarrow \{\}$
$L_{added} \leftarrow \{\}$
Add each $\varphi_i \in L_+$ minimally to L
foreach φ_i *actually added to* L *above* **do**
  Add φ_i to L_{added} minimally
  Add any variables mentioned in φ_i to Q if not already present
end
$L_{consensuses} =$ ConsensusUntilFixedPoint(L, Q)
Add each $\varphi_i \in L_{consensuses}$ minimally to L_{added}
return L_{added}

The AddAndGeneratePrimeImplicants method is shown in Algorithm 7. It is responsible for adding the environments in L_+ to the label L, generating any new prime implicants of L, adding those to L, and returning a minimal set of environments L_{added} that have been added to L. Note that L_{added} may contain both environments from L_+ and newly-generated prime implicants.

The AddAndGeneratePrimeImplicants maintains a queue Q of variables, over which to find consensuses. Whenever a new environment φ_i is added to L, all of the variables referenced in φ_i are pushed onto Q (if they're not on there already). This is because it may now be possible to compute new consensuses over those variables. After each $\varphi_i \in L_+$ is added to L minimally and Q is populated, then the routine ConsensusUntilFixedPoint is called with Q, which continually computes new consensuses until no new ones can be generated and returns the resulting additional environments.

The ConsensusUntilFixedPoint takes as input a label L as well as a queue Q of variables. For each variable in Q, it computes consensuses / implicants by taking the cross product of satellite candidate sets. Each of the resulting consensuses is a new implicant that can be added to L. If some newly-computed implicant φ_i is added minimally to L, then its variables are pushed onto Q if not present already. The process is repeated until the queue is empty, meaning that no more consensuses can be computed. This "fixed point" means that L is now a set of prime implicants, as no new, smaller implicants can be derived. This completes the algorithmic description of the πTMS.

5.7 Using the πTMS for Online Queries

Here, we describe how the prime implicants computed by the πTMS are used to enable Pike's reactive online execution. Specifically, we describe how the queries over KB outlined in Section 2.3 are implemented, given the prime implicants of the goal node N_G.

- CorrectTeamPlanExists?(KB): Returns True iff the label of N_G contains at least one environment; else False if it is empty.

- COULDCOMMITTOADDITIONALCONSTRAINTS?(KB, F): F is a set of constraints. We take advantage of the fact that F contains only assignment and/or conflicts (and not, for example, arbitrary propositional sentences) – this is guaranteed via the constraints noted in Section 3. Without loss of generality, let φ_a denote an environment representing the conjunction of all assignments in F, and C denote the set of conflicts in F. To implement the procedure, we search for some prime implicant φ_i in node N_G's label where $\varphi_i \wedge \varphi_a$ does not manifest any of the new conflicts C. If there is such a φ_i, then there exists some scenario extending φ_i, consistent with φ_a, that satisfies all of KB's constraints in addition to F.
- COMMITTOCONSTRAINTS(KB, F). Like the above, F will only contain conflict constraints or assignments. For each conflict, we encode the constraint by creating a new contradiction node, and adding a single justification to it corresponding to the conflict. During label propagation, all labels will be split on this conflict, guaranteeing that they resolve the conflict. For assignments $x = v_i$ in F, we encode a conflict $\neg(x = v_j)$ for each domain value $v_j \neq v_i$.

By following the above procedures, we can use the set of prime implicants in the label of N_G to answer the online queries required by PIKE's online execution system. This enables a robust user interaction that can adapt to the human's intent as well as other disturbances.

6. Evaluation

In this section, we focus on evaluating two key claims of this work: (1) *concurrent* intent recognition and adaptation is advantageous over other approaches that do so *separately*, and (2) PIKE supports fluid human-robot teamwork through its goal-directed execution.

We validate (1) in simulation, comparing PIKE against a competing approach based on Kirk (Kim et al., 2001) that recognizes intent and adapts via separate processes. We evaluate (2) with two hardware demonstrations of human-robot teamwork.

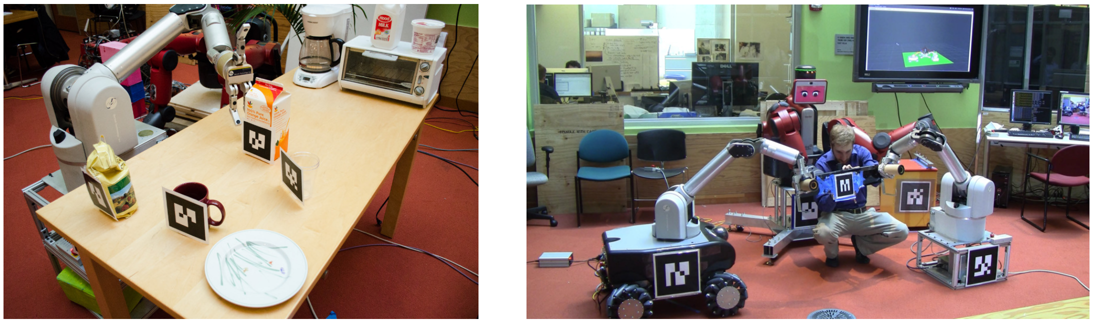
We then provide additional performance measurements from simulation on randomly-structured problems to quantify PIKE's scalability. Finally, we conclude this section with a discussion about the appropriate amount of flexibility for TPNUs executed with PIKE.

6.1 Evaluation in Robotic Testbed

We describe two proof-of-concept hardware demonstrations where PIKE has successfully been applied to tasks requiring human-robot collaboration. These demonstrations show promise that PIKE could successfully be applied to other similar real-world tasks.

Figure 15a shows the familiar household robotics breakfast domain described earlier in this work. The TPNU for this domain was manually generated. A robotic arm is capable of picking up either coffee grounds or orange juice. The human may pick up a coffee mug or a glass for juice. This demo behaves exactly as described earlier: if the human operator picks up the coffee mug, the robot will infer the human's intent to make coffee and react by fetching the nearby coffee grounds. Similarly, had the human instead picked up the glass, the robot would have adapted by handing the person orange juice.

A computer-vision based sensing system was implemented to act as the activity recognizer in this example. All objects were annotated with AR tags, allowing their 3D location



(a) Kitchen example in hardware. (b) Airplane manufacturing demo.

Figure 15: Various human-robot hardware demos of Pike. At left, the breakfast TPNU as described earlier in this document. At right, an airplane manufacturing scenario.

to be measured. To recognize the human's low-level actions, motion was detected on either the mug or the glass. If for example the sensing system detected the mug being raised into the air by the human, then the choice x_{A1} = mug was published to Pike.

Figure 15b shows a more complex example, in which a human and heterogeneous team of robots collaboratively construct part of an airplane wing in a manufacturing setting. In this demo, two robots (right and left) lift up some internal framing of an airplane wing for the human and the red Baxter robot to collaboratively assemble together. A second piece, the "rib" must be attached to the frame using a temporary fastener called a cleco. Depending on the action taken next by the human, the Baxter robot adapts accordingly. If the human picks up pliers, Baxter will fetch a cleco fastener. If the human instead picks up a cleco, Baxter will pick up the pliers. If the human picks up both the plier and the cleco, the robot detects this and waits idly. After the first cleco is fastened, a similar process occurs for a second cleco. In this way, the robot is able to adapt to the human's inferred intent in a manufacturing setting. The TPNU for this domain was manually generated.

In both of these scenarios – the robotic breakfast manufacturing and the airplane manufacturing – the robots successfully and proactively assisted the human by selecting adaptations consistent with the team's goals. As a result, the team was able to complete their tasks. Qualitatively, the robots acted responsively and chose adaptations at reactive timescales with little noticeable latency after the activity recognizer published uncontrollable choices to Pike.

6.2 Simulations

In addition to the hardware demonstrations above, we also validate Pike in simulation. First, we show empirically that Pike's approach – namely concurrently recognizing human intent and adapting online – is advantageous over other approaches where intent recognition and adaptation occur as separate processes. Second, we evaluate the scalability of Pike.

6.2.1 Advantage of Concurrent Intent Recognition & Adaptation

Pike performs intent recognition and adaptation *concurrently*: the two occur in an interleaved manner online during execution. This stands in contrast to other approaches where intent recognition occurs *separately* from adaptation, such as systems in which intent is recognized offline prior to execution, followed subsequently by a suitable adaptation being chosen and acted upon during the separate execution phase.

In this section, we show the benefit of concurrent intent recognition and adaptation over an approach that performs separate intent recognition and adaptation. Specifically, we compare Pike against a variant of Kirk (Kim et al., 2001). Kirk takes as input a TPNU and makes all choices optimally offline based on a cost function such that the resulting candidate subplan is temporally consistent. This choice-less subplan can then be dispatched online with execution monitoring (Levine, 2012). We use a Kirk-inspired strategy as a competing approach that performs intent recognition and adaptation separately as follows. Before execution begins, an intent is "inferred" by guessing any intent scenario φ_I such that there exists a corresponding adaptation scenario φ_A where the candidate subplan $\varphi_I \wedge \varphi_A$ admits a correct execution. In other words, a single intent that is consistent with at least one adaptation is chosen, instead of maintaining multiple hypotheses for possible intents as Pike does. After this offline intent inference, adaptation occurs subsequently in the separate execution phase in which the robot acts by following φ_A. In this way, Kirk has made all of the controllable and uncontrollable choices in the plan, resulting in a single candidate subplan to be executed.

If there are multiple possible feasible intents, this separated approach may of course incorrectly guess the true intent. If this occurs during execution and the Kirk executive determines that it made an uncontrollable choice decision incorrectly, replanning is triggered. For the purpose of these tests, we compute the suffix of the existing TPNU, backtracking the last incorrect uncontrollable choice and making it correctly based on the now-observed uncontrollable choice. This approach does not work in every domain, however, so full generative planning is required in the general case. Furthermore, in many domains it is necessary to execute a recovery activity if an incorrect choice is made during execution.

Example 6.1 (Recovery Activity)**.** Suppose a robot that is helping to prepare breakfast incorrectly infers the human's intent and hence adapts incorrectly: the human's true intent is to prepare juice, but the robot incorrectly assumes the person wants coffee and hence picks up the coffee grounds. Once the robot becomes aware of its mistake, it cannot simply pick up the juice immediately – it must first execute a recovery activity to repair the world state caused by the incorrect adaptation. In this example, the recovery activity would be to first put down the coffee grounds so that the robot has an empty gripper again – a necessary precondition for picking up the juice in the new plan.

These recovery activities take time, and delay the completion of the overall task. For these reasons, it is crucial for an online executive to minimize the number of execution failures that trigger replanning. Having many such failures / replans will result in a poor quality of interaction with the human, and could substantially delay the human robot team.

We choose a Kirk-inspired approach as our incumbent due to its similarities with Pike. Both have a similar problem statement, models, and inputs and outputs. Our implementation also shares as much code as possible: Pike and our Kirk variant share the same temporal reasoning, dispatching, and execution monitoring components. The key difference between the two is, again, that Pike performs intent recognition and adaptation

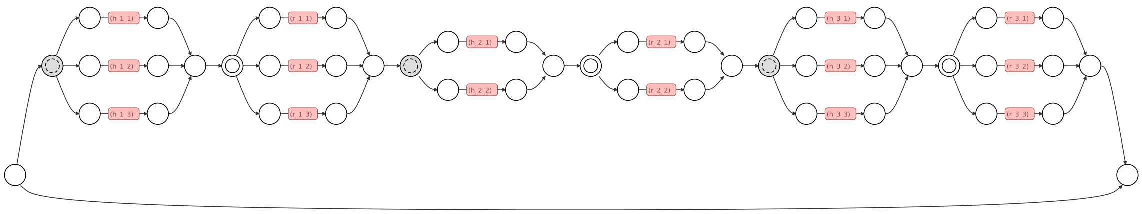


Figure 16: An example k-intents plan for $k = 18$ and structure $[3, 2, 3]$.

concurrently while maintaining multiple hypotheses, but Kirk performs intent recognition separately from the adapting execution phase and maintains a single candidate hypothesis.

To compare this variant of Kirk with Pike in a controlled manner, we introduce an artificial domain called the k-intents domain, generate thousands of example problems from this domain with varying structure, and dispatch each using the incumbent version of Kirk while counting the number of execution failures that trigger replanning during each dispatch. A problem from the k-intents domain consists of a TPNU as well as an action model that imposes causal links. An example is shown in Figure 16. Each TPNU has a structure denoted by a set of positive integers $[N_1, N_2, ..., N_m]$, representing a sequence of choices between different activities. For each N_i, the TPNU contains a pair of choices: (1) a human-made uncontrollable choice $y_i \in \{1, 2, ..., N_i\}$ followed sequentially by (2) a robot-made controllable choice $x_i \in \{1, 2, ..., N_i\}$. Each outcome $y_i = j$ activates a single activity denoted h_{ij}, and similarly each outcome $x_i = j$ activates a single activity denoted r_{ij}. Our action model is such that h_{ij} has a single at-end add effect of predicate p_{ij}, and r_{ij} has a single at-start precondition of p_{ij}, thus forming a causal link from h_{ij} to r_{ij}. Therefore, in any causally complete execution, $y_i = x_i$ for all pairs of choices $i = 1, \dots, m$. We call this a k-intents plan because it has k possible intent scenarios, where $k = \prod_{i=1}^{m} N_i$. In terms of temporal durations, all activities of each y_i and x_i choice have identical randomized temporal duration of the form $[1 + w, 2 + w]$ where w is sampled uniformly in the range $[0, \frac{1}{2}]$. Finally, the k-intents domain requires that if the executive makes an incorrect choice and begins dispatching the wrong activity, then a recovery activity must be executed before re-planning (thereby backtracking from the mistake) with duration half that of the incorrect activity's lower bound (i.e., $\frac{1+w}{2}$).

A k-intents problem has the property that for each intent scenario φ_I, there exists a single adaptation scenario φ_A that together form a correctly executable candidate subplan. This enables the following observation:

Observation (Pike requires no re-plans for k-intents)**.** *Given a k-intents problem,* Pike *will execute it and never trigger replanning.*

Proof. Pike compactly encodes the set of all correct team subplans in KB. Whenever an uncontrollable choice $y_i = j$ is observed, Pike is immediately able to infer from KB that the corresponding choice $x_i = j$ must be made. This results in a successful execution where Pike waits for uncontrollable choices to be observed, then makes the corresponding controllable choice correctly – never failing and triggering a replan. □

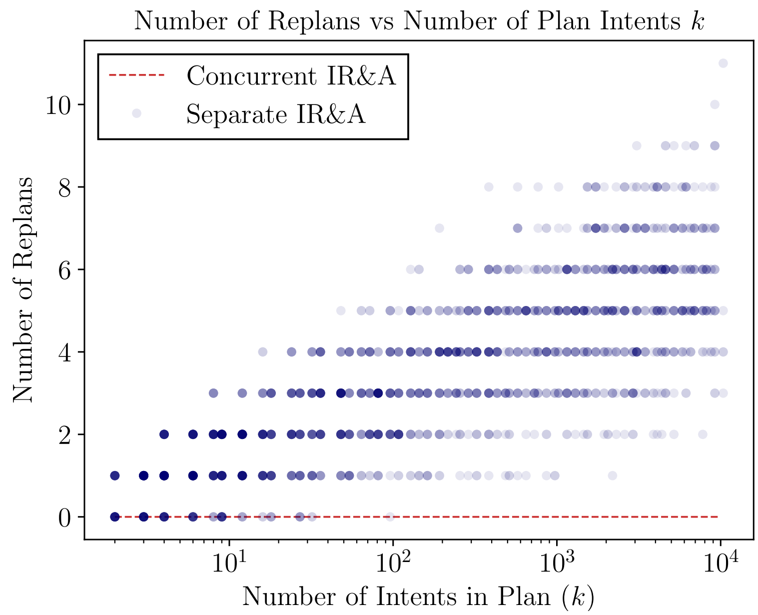


Figure 17: Comparison of replans for concurrent intent recognition and adaptation (dotted line) versus separate intent recognition and adaptation (dots).

This nice property of Pike does not hold for Kirk, however:

Observation (Kirk may need to replan for k-intents)**.** *Given a k-intents problem, Kirk may need to replan, possibly multiple times, during execution.*

Proof. In Kirk, intent recognition happens first, in which an intent φ_I is chosen. Then separately, an adaptation φ_A is chosen based on φ_I. However, during execution it may very well be the case that the φ_I guessed at start does not match the actual intent chosen by the human, which will be revealed online one assignment at a time through a sequence of uncontrollable choice observations $\mathcal{U}(t)$. If this happens, for example if $\varphi_I \models (y_i = j)$ but Kirk observed that actually the human chose $y_i = j'$ for some $j \neq j'$, then execution will fail and replanning will occur. □

We experimentally validate these observations, measuring the number of replans required by the Kirk strategy. Figure 17 compares Kirk's approach of separate intent recognition & adaptation to Pike's concurrent approach that maintains multiple hypotheses. We generate 2000 instances from the k-intents domain and execute each with our Kirk variant and with Pike, counting the number of replans. Each point represents a k-intents problem instance, where k is shown on the x-axis and the number of replans needed by the Kirk-inspired approach is shown on the y-axis. $N_i \in \{2, 3\}$ for all i in these tests. To generate a single problem, k is sampled logarithmically (so that the x-axis is uniform), and the closest factorization by 2's and 3's is computed. This factorization is shuffled, yielding a randomized structure for the k-intents problem. Pike and our Kirk variant then each execute this problem instance, and the number of replans is recorded.

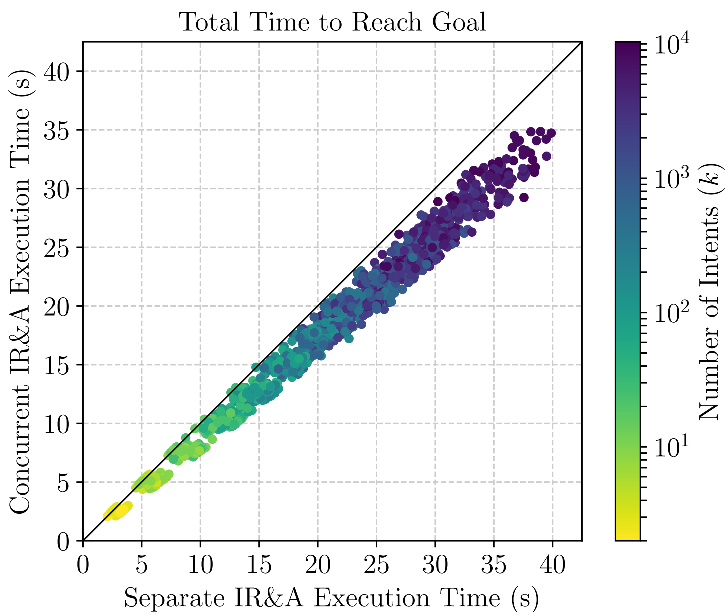


Figure 18: Comparison of total time to complete the task for concurrent intent recognition and adaptation versus separate intent recognition and adaptation.

These results in Figure 17 show that as the number of intents in the plan grows, the average number of replans required by Kirk also grows. Since the number of intents is often exponential in the number of uncontrollable choices, this indicates that the number of replans tends to grow roughly linearly with the number of uncontrollable choices in the plan. Pike did not replan once for any of these problems.

In addition to counting the number of replans, we now analyze the same data from a different perspective: the total time required to complete the task by executing the entire plan correctly. This includes a number of factors:

1. **Total online dispatch time.** The total time required for online dispatching of the temporal plans (including new plans from replanning). Problem instances with larger k tend to have more choices and activities, and hence take more time.

2. **Any online replanning time and recovery actions.** If execution fails due to an incorrect intent inference and replanning is triggered, we measure the time required to replan and come up with a new TPNU suffix that completes execution, and perform temporal and causal link reasoning on this TPNU suffix. Note that we do not count the initial offline compilation time for either Pike or the Kirk approach (we assume that the initial compilation can be computed beforehand). We additionally measure the time delay required for the recovery activity taking $\frac{1}{2}$ of the lower bound duration of the incorrectly-dispatched activity.

3. **Extra overhead of maintaining multiple hypotheses.** Pike has a slightly higher overhead than Kirk due to its online reasoning about many possible intents and adaptations (as opposed to Kirk, which considers only a single hypothesis).

For our Kirk variant, our implementation of online replanning and generating the TPNU suffix is specific to the k-intents domain and hence very fast. In addition, the temporal reasoning and causal link reasoning – while employing the same algorithms as Pike– in practice require less overhead when used with Kirk, because after it chooses a single candidate subplan, the inactivated temporal constraints and activities are eliminated from consideration.

Figure 18 shows the results of executing the 2000 k-intents problems with both Pike and the Kirk variant. Each problem instance is represented by a point, where the x-axis shows the total time for the Kirk variant and the y-axis shows the total time for Pike. Points are shaded by k. We see in general a clear trend in which Pike consistently completes the task faster than Kirk, especially for larger k. For this data, the average performance improvement of Pike over Kirk is 13.4%. We note that this improvement is largely dominated by the time delay incurred by recovery activities. If we were to (unrealistically) ignore such time delays and assume recovery activities were instantaneous, the performance improvement of Pike over the Kirk variant would be a much more modest 2.2%. Such a measurement only considers differences in terms of online replanning (which Pike never performs for k-intents) and the extra overhead of Pike maintaining multiple hypotheses. While this 2.2% figure is modest for k-intents, it does demonstrate that the costs incurred by Pike's maintaining multiple hypotheses is generally more than offset by the cost of replanning multiple times during execution, even when that replanning is fast.

In domains where replanning is costly and recovery activities must be taken, the results of Figure 17 and Figure 18 show a notable advantage for approaches such as Pike that concurrently perform intent recognition & adaptation by simultaneously and compactly maintaining many hypotheses.

6.2.2 Scalability Evaluation

In addition to the above, we evaluate Pike's performance in terms of compilation speed, execution latency, and KB compactness. For this, we generate different random, structured problem instances. Each problem instance consists of a randomly-generated TPNU, as well as a randomly-generated action model to impose causal link structure. While the problem instances we test on are randomly-generated, we attempt to add structure to the problems to improve realism. However, as with many artificially, randomized-generated domains, this setting is likely significantly more challenging to Pike than more engineered, real-world problems such as those demonstrated on robotic hardware.

We generate each TPNU as follows. We randomly sample a sequence of elements, where each element is either an activity, a choice, or parallel substructure. For choices and parallel substructure, we further sample more sequences of elements to build up the TPNU recursively. When sampling a sequence, the length is chosen from a binomial distribution whose parameters vary based on the nested depth of the sequence. For example, a top-level sequence of a TPNU may have a maximum of 10 elements, whereas a sequence being generated for a parallel or choice substructure nested one level deep may have a maximum length of 3 elements in our tests. Once the number of elements has been determined, we

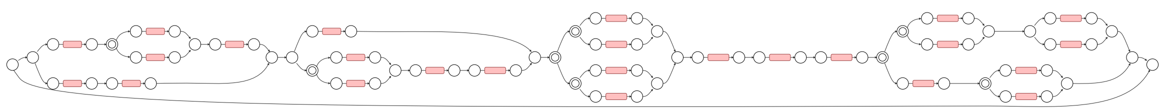
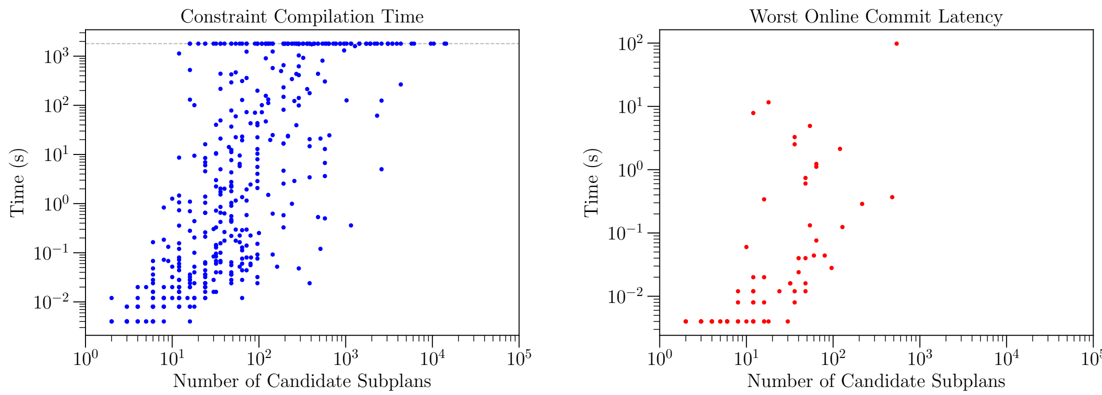


Figure 19: An example of a randomly-generated, structured TPNU with sequence, parallel, and choice structure. Not shown for brevity are the activity durations, nor the causal link structure imposed by the action model. This TPNU has 25 activities and 64 candidate subplans.

randomly choose which structure type the element will be: an action, a choice, or a parallel substructure. The probability mass function for this sampling also varies based on nested depth; our tests enforce that no more parallel substructure will be sampled at depths of 3 or more, thus limiting the recursion depth and size of the generated TPNUs. For action substructure, a random activity name is chosen, as are random integer-valued lower and upper bounds. For both parallel and choice substructure, we further sample more sequences of elements at the increased nesting level, corresponding to each "branch." Finally, after all of the substructure has been recursively constructed, the final step in TPNU generation imposes an overall temporal constraint that may "squeeze" the plan. This upper bound of this temporal constraint is currently chosen to be between 0 and 10% shorter than the minimum time required to execute the TPNU, thereby creating temporal conflicts and making certain candidate subplans of the TPNU temporally inconsistent. An example of a randomly-generated TPNU is shown in Figure 19.

Once the TPNU has been constructed, we generate random causal link structure via an action model. We use PDDL 2.1 (Fox & Long, 2003) to specify the *at start* conditions and *at end* effects of activities in the TPNU to enforce causal link relations. Our approach works as follows. For each activity A_c in the TPNU, we randomly generate several *at start* conditions p_i for the activity (in our tests, $i \in [1, 4]$ and is chosen via a binomial distribution). For each of these new conditions added, we create (possibly multiple) supporting producer actions $A_{P_{ij}}$ that assert those conditions as effects (thereby creating labeled causal links). To do this, we compute the set of preceding actions by searching the TPNU structure for all actions that are guaranteed to come before our consumer action (we also include a special activity $A_{initial}$ representing the initial conditions). As a result, our tests will never require Pike to add temporal constraints to order a potential producer before its consumer. The producer candidates $A_{P_{ij}}$ will be chosen randomly from this set, and p_i will be added as an *at end* effect to each. If $A_{initial}$ is chosen, then p_i will be added as an initial condition to the PDDL problem. The number of labeled causal links is chosen from a binomial distribution (in our tests, $j \in [1, 4]$). Next, we randomly generate threat activities $A_{T_{ik}}$. Threats are randomly chosen from the set of actions that precede at least one of the producers $A_{P_{ij}}$. The number of threats is chosen from a binomial distribution, whose parameters limit the number to the number of causals links or preceding actions. For each threat $A_{T_{ik}}$, $\neg p_i$ is added as an *at end* effect. Finally, the resulting action model, as initial state, and goal state are written to PDDL domain and problem files.

We generated 500 such random TPNUs with associated action models, and ran Pike on each accordingly. Many of these problems admit no correct execution but still serve as



(a) Constraint compilation time

(b) Maximum online commit latency.

Figure 20: Constraint compilation time for the πTMS, and the maximum latency measured during online execution. The x-axis is a rough measure of the complexity of the plan; it is the number of candidate subplans of the TPNU. The y-axis is time in seconds.

a useful point of reference. Our benchmarking setup ran Pike with an upper time limit of 30 minutes compilation time on each problem. Figures 20a and 20b show the results of running these simulated examples on Pike, using the πTMS to compile the constraints. On the left, we see the time required for Pike to compile the constraints and compute the set of prime implicants (this plot does not include the time to compute the labeled APSP). For these graphs, each dot represents a problem instance. The y-axis measures time, and the x-axis measures the number of candidate subplans of the TPNU. This number of candidate subplans can be computed via a recursive formula, traversing the elements that were used in constructing the TPNU: sequential structure yields the product of the count of each sub element, parallel structure operates similarly, choice structure yields the sum of the count of sub elements, and activities yield the value 1. Applying this to the example shown in Figure 19 yields 64 candidate subplans. Note that in general, TPNUs that appear relatively small actually have exponentially many candidate subplans.

As we can see, there is a large variance in compilation times, with the 30 minute timeout band occurring at the top of the plot (124 samples, or 24.8% of problems). As expected, compilation time increases with increasing number of candidate subplans.

We note that for many domains of interest, including manufacturing and high-risk settings, spending a significant amount of time beforehand in a compilation process is an acceptable trade-off to improve online performance. In these settings, there is typically plenty of available time for the robotic system to plan and prepare before execution begins. It is therefore not problematic in these domains to spend a significant amount of time before execution begins on the offline reasoning phase in order to achieve a low online latency.

In Figure 20b, we see the worst case execution latency. Of the 500 problems generated, 94 TPNs admitted correct executions, and 75 of those 94 did not timeout with the πTMS and were executed. Across each simulated execution, this is the worst-case time of the event execution loop in the OnlineExecution procedure shown in Algorithm 1 over the entire course of execution. We note that most calls to this function are actually significantly much

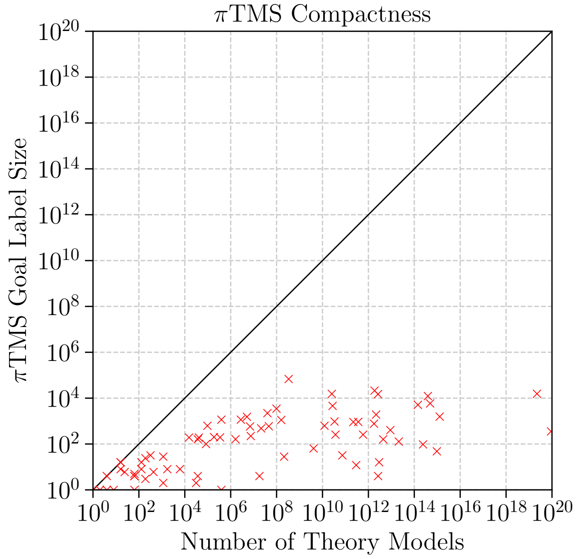


Figure 21: Compactness of the πTMS. The x-axis measures the number of models in the encoded theory as measured by the DSHARP model counter; the y-axis are the number of prime implicants computed by the πTMS. This ratio is generally much less than 1, indicating significant space savings.

faster, due to work in caching solution results and the fact that queries for certain events tend to be much more demanding than for others. The results presented therefore represent a conservative, worst-case estimate for these challenging, randomly-generated problems. We see that latency times are generally reactive except for several examples, with 85.3% of executions having a worst-case latency of 0.5 seconds or less. There are, however, certain cases for which significantly delays have been recorded; we attribute these to the difficulty afforded by the random causal link structure.

A key goal of the πTMS is to encode the solution space efficiency. Figure 21 quantifies this, by plotting (only for problem instances that admit correct executions) the ratio of the πTMS goal label size to the number of solution models, as measured by the DSHARP model counter (Muise, McIlraith, Beck, & Hsu, 2010). As can be seen, the number of prime implicants never exceeds the number of models (despite this being theoretically possible), but rather in most cases is orders of magnitude less. This indicates that our attempts to compactly represent the entire solution space of KB are fruitful.

6.3 How Much Flexibility is Best?

In this section, we provide a recommendation on how much flexibility should be afforded to an executive such as Pike, which accepts a very expressive input representation (i.e., the TPNU) that can vary wildly in flexibility.

On one end of the spectrum, Pike can be given an extremely rigid TPNU: one with no choices, and where every activity appears sequentially one after the other. This TPNU could further have tight or even completely rigid temporal durations (for instance, $[2, 2.01]$ or $[2, 2]$, respectively) limiting the possible temporally consistent schedules. Such a TPNU could be generated by a classical planner or by a temporal planner that generates a fixed schedule (Benton, Coles, & Coles, 2012). When given a rigid TPNU, Pike is very limited in the ways it can adapt to achieve online robustness. As a result, such rigid plans may be brittle in very dynamic domains in which disturbances regularly occur. The performance of online execution would likely be poor due to failures caused by disturbances, and without replanning, Pike would be powerless to do better.

On the other end of the spectrum, Pike can be given an extraordinarily flexible TPNU: one that it is so flexible, in fact, that it could encode every possible plan up to a given maximum length. Imagine a generalization of the TPNU shown in Figure 11, in which we have a parallel choice structure for every possible grounded action, repeated k times. This would allow any plan (up to length k) to be encoded. Essentially, the entire planning process would be deferred to Pike, which would choose the plan dynamically online. Virtually no work would be done beforehand by an offline planner. Pike could in theory handle and successfully execute such a TPNU, though in practice it would be intractable even for moderate values of k. Pike's offline compilation algorithm would be required to reason concurrently about every possible plan up to length k, and – even if this were tractable – there would be substantial online overhead in maintaining and reasoning about such a huge set, impacting online latency. So, given this other extreme in which there is a tremendous degree of flexibility afforded to the executive, Pike would also perform poorly.

Despite these two extremes, evidence exists (e.g., via the total time to reach the goal in Figure 18) suggesting that affording *some* degree of flexibility to the online executive can be very beneficial to improving overall performance. This raises the question: how much flexibility should there be in the TPNUs given to a least-commitment executive such as Pike?

We recommend taking a middle ground approach, in which the TPNU encodes as much flexibility as possible, such that tractability is maintained. One way to achieve this is to construct TPNUs for Pike that have contingencies to handle only the most common or most likely outcomes that may arise during execution. This may mean, for example, modeling the most likely actions that a human is likely to take, or modeling contingencies for a robot to retry certain tasks that are likely to fail (e.g., if the robot has an unreliable gripper, it may need to try several times before successfully picking up an object). However, contingencies for exceedingly unlikely situations probably do not need to be reasoned about or encoded in the TPNU (e.g., the situation in which an asteroid falls from the sky and hits the robot). By constructing appropriately flexible TPNUs, Pike will be able to avoid intractability while achieving a robust online execution.

7. Conclusion & Future Work

This work introduces Pike, an executive for human-robot teamwork that views intent recognition and robot adaptation as two fundamentally interwoven problems. We argue that any autonomous system must address both of these problems to successfully work alongside humans. Pike uses a single set of algorithms and a single model to concurrently perform intent recognition and find suitable robot adaptations for contingent, temporally-flexible team plans. The result is a mixed-initiative execution in which the human and robot work together and influence one another.

We achieve this through the use of temporal reasoning and causal link analysis, which allows us to find relationships between possible choices in the team plan. This allows Pike to make controllable choices online that will be consistent with choices made by the human, thus adapting to intent. For this purpose, we introduce labeled causal links as well as related extraction algorithms, and show how these labeled causal links can be used to guide correct decisions online by generating additional temporal and propositional constraints. This allows the robot to quickly and effectively make choices online based on the outcomes of the human's decisions, the preconditions and effects of activities in the plan, temporal flexibility, and unanticipated disturbances.

There are a number of avenues for future work. One is to examine the effectiveness of other promising knowledge compilation techniques other than prime implicants, such as d-DNNF for example, to represent the set of candidate subplans admitting a correct execution (Muise et al., 2010). Second, we believe it would be valuable to validate Pike's ability to enhance fluid human-robot teamwork through a user study in which team fluency metrics are measured (Nikolaidis & Shah, 2013). Finally, a third avenue that we are currently pursuing takes a probabilistic approach that models the distribution over likely human choices, so that the robot may adapt to likely human intents in a chance-constrained manner and better predict the probability of plan success.

Acknowledgments

We are very grateful to the members of the MERS group in MIT CSAIL for their invaluable feedback and discussions on this work. We are particularly indebted to Christian Muise for a plethora of helpful technical discussions and insights relating to this research and earlier drafts of this paper, as well as to Andreas Hofmann for his valuable guidance and advice throughout the research process. The anonymous journal reviewers provided thought-provoking and detailed feedback that was extremely valuable in strengthening the final version of this paper. We would also like to thank our collaborators at the Mitsubishi Electric Company, Yuki Sato and Akihiko Honda, for their generous support and wise insights from the industrial perspective. This work was funded in part by Boeing grant MIT-BA-GTA-1 and by Mitsubishi grant 024909-00001.

Appendix A. Proofs

Here we provide proofs of key theorems throughout this work. Note that proofs are presented in a different order than in the paper, for logical progression and clarity here.

Theorem 4.3 (Executions from $\mathcal{T}$ and $\mathcal{T}'$ correspond)**.** *An execution $\langle \varphi_S, T_{\varphi_S} \rangle$ of the original TPNU $\mathcal{T}$ is correct if and only if there exists a corresponding correct execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of the augmented TPNU $\mathcal{T}'$ where $\varphi'_S \models \varphi_S$.*

Proof. $\Rightarrow$. We may assume there exists a correct execution $\langle \varphi_S, T_{\varphi_S} \rangle$ of $\mathcal{T}$. It is possible to choose assignments to the additional s_{p,e_c} and o_{p,e_c,e_P,e_T} variables in $\mathcal{V}'$ not present in $\mathcal{V}$, appending those assignments to φ_S conjunctively to create a φ'_S where $\varphi'_S \models \varphi_S$. We use this to construct an execution with an identical schedule, $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}'$. Since $\langle \varphi_S, T_{\varphi_S} \rangle$ is causally complete, every precondition of every event must be satisfied by the time of the event's execution. We also know that there can be no consumers that come before their supporting producers in T_{φ_S}, and no threats may occur between producers or consumers. We can therefore trace this support through properly selected assignments to the s_{p,e_c} and o_{p,e_c,e_P,e_T} variables. The additional guarded temporal constraints present in $\mathcal{T}'$ but not in $\mathcal{T}$ must be satisfied by T_{φ_S}, as the added temporal constraints only serve to preclude such causally incomplete executions. Therefore, the execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}'$ must be temporally consistent. We now argue that it is also causally complete. Since $\mathcal{T}'$ and $\mathcal{T}$ share the same events $\mathcal{E}$ (each with preconditions and effects from the same activities $\mathcal{A}$), and the events occur at the same time in both executions by virtue of sharing the same schedule T_{φ_S}, the preconditions of all events would be satisfied in $\mathcal{T}'$ at their proper times – just the same as in $\mathcal{T}$. Therefore, the execution $\langle \varphi_S, T_{\varphi_S} \rangle$ of $\mathcal{T}'$ is causally complete, and correct.

$\Leftarrow$. We may assume there exists a correct execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}'$. Let φ_S be the projection of φ'_S (which assigns all the variables $\mathcal{V}'$ of $\mathcal{T}'$) onto the variables $\mathcal{V}$ of $\mathcal{T}$ – i.e., removing the additional s_{p,e_c} and o_{p,e_c,e_P,e_T} variable assignments. Similar to the above case, we know that both executions are causally complete since they share the same schedule, events, and activities with preconditions and effects. Furthermore, since $\mathcal{T}$ contains a subset of the temporal constraints from $\mathcal{T}'$, and since T_{φ_S} satisfies all of those constraints from $\mathcal{T}'$, it must also satisfy all of the temporal constraints of $\mathcal{T}$. The execution $\langle \varphi_S, T_{\varphi_S} \rangle$ is therefore temporally consistent. We have shown it is both temporally consistent and causally complete, so it is correct with respect to $\mathcal{T}$. □

Theorem 4.5 (Temporally consistent executions of $\mathcal{T}'$ with KB are also causally complete, and correct)**.** *Let $\langle \varphi'_S, T_{\varphi_S} \rangle$ be a temporally consistent execution of $\mathcal{T}'$ where φ'_S satisfies KB. Then this execution is also causally complete (and hence correct).*

Proof. We show that the additional propositional and temporal constraints added during Pike's compilation guarantee causal completeness. For φ'_S to be a solution of KB, it must satisfy all of KB's constraints and assign all variables, including the original variables, the s_{p,e_c} variables, and the o_{p,e_c,e_P,e_T} variables.

Let event e_c be any event activated by φ'_S with precondition p; for causal completeness, we show that p is expected to hold at the time e_c is scheduled by T_{φ_S}. Suppose without loss of generality that $s_{p,e_c} = e_P$[4]. By the additional temporal constraints added to the problem guarded in part by this assignment, we know that (1) e_P must also be activated by φ'_S, and (2) e_P must precede e_c in T_{φ_S} (else T_{φ_S} would be temporally inconsistent). Furthermore, any threat e_T cannot occur between e_P and e_c in T_{φ_S}, as required by the other temporal constraints added and possibly the o_{p,e_c,e_P,e_T} variables. Therefore, assuming no unmodeled disturbances, p will hold by the time e_c is scheduled in T_{φ_S}.

Thus since the execution is temporally consistent and causally complete, it is also correct. □

4. The reader may wonder why we exclude the possibility of $s_{p,e_c} = \perp$. Any such assignment however is guaranteed to not satisfy KB since e_c is activated, and hence φ_S would not even be represented by KB.

Theorem 4.6 (φ'_S satisfies KB iff correct)**.** *φ'_S satisfies KB if and only if there exists a correct execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}$'.*

Proof. $\Rightarrow$. We show that if φ'_S satisfies KB, then there exists a correct execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}$'. Since all temporal conflicts are encoded in KB, by Theorem 4.1, there exists a temporally consistent execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}'$. By Theorem 4.5, this temporally consistent execution must also be correct.

$\Leftarrow$. We show that if there exists a correct execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}'$, then φ'_S satisfies all of the constraints in KB. This execution must be both temporally feasible and causally complete. Since it is temporally feasible, T_{φ_S} satisfies all of the temporal constraints activated by φ'_S. So, φ'_S must satisfy all of the temporal conflict constraints derived by the labeled APSP present in KB. Next, we consider the causal completeness constraints in KB. Since the execution is causally complete, we know that there must be some supporting producer e_P for each precondition p of every consumer event e_c, and any threats must be resolved. We may therefore assume that φ_S assigns $s_{p,e_c} = e_P$ and any associated o_{p,e_c,e_P,e_T} variables appropriately to satisfy the additional propositional constraints in KB. Since all constraints in KB are satisfied, then φ'_S satisfies KB. □

Theorem 4.7 (CanExecuteEventNow? is correct)**.** *Let $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ be the current, correct partial execution of $\mathcal{T}'$. Then* CanExecuteEventNow?(e_i, t) *returns* True *at time t if and only if the partial execution of $\mathcal{T}'$ that would result if e_i is executed at time t – namely $\langle \varphi_{ex} \wedge \varphi_{e_i},\ \tilde{T}_{\varphi_{ex}} \cup \{e_i = t\} \rangle$ – is correct.*

Proof. The CanExecuteEventNow? procedure begins by obtaining the set of additional constraints $F_1, ..., F_n$ required to execute event e_i at time t (one of these constraints is φ_{e_i}). It then returns whether KB $\wedge \Big(F_1 \wedge ... \wedge F_n \Big)$ is satisfiable. We prove that this conjunction is satisfiable if and only if $\langle \varphi_{ex} \wedge \varphi_{e_i},\ \tilde{T}_{\varphi_{ex}} \cup \{e_i = t\} \rangle$ is correct.

$\Rightarrow$. We may assume that there exists some φ'_S satisfying KB $\wedge \Big(F_1 \wedge ... \wedge F_n \Big)$. φ'_S must therefore satisfy both KB and $F_1 \wedge ... \wedge F_n$. Since $F_1 \wedge ... \wedge F_n$ is satisfied by φ'_S, then by Theorem 3.2, the partial execution $\langle \varphi_{ex} \wedge \varphi_{e_i},\ \tilde{T}_{\varphi_{ex}} \cup \{e_i = t\} \rangle$ can be extended to a temporally consistent execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}'$. Considering that KB is satisfied, then by Theorem 4.5, this temporally consistent execution must also be causally complete and hence correct. Therefore, the partial execution is correct.

$\Leftarrow$. We may assume that $\langle \varphi_{ex} \wedge \varphi_{e_i},\ \tilde{T}_{\varphi_{ex}} \cup \{e_i = t\} \rangle$ is correct, or namely that there exists an extending execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}'$ that is correct. By Theorem 4.6, φ'_S satisfies KB. Since this execution is temporally consistent, by Theorem 3.2, φ'_S satisfies $F_1 \wedge ... \wedge F_n$. Therefore, the assignment φ'_S satisfies KB $\wedge \Big(F_1 \wedge ... \wedge F_n \Big)$. □

Theorem 4.4 (Correct Executions of Original $\mathcal{T}$)**.** *Let $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ be a partial execution at some point during execution of the augmented TPNU $\mathcal{T}'$. This partial execution is correct with respect to the augmented TPNU $\mathcal{T}'$ if and only if it is also correct with respect to the original TPNU $\mathcal{T}$.*

Proof. The partial execution $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ is correct with respect to $\mathcal{T}'$ iff it can be extended to a correct execution $\langle \varphi'_S, T_{\varphi_S} \rangle$ of $\mathcal{T}'$. By Theorem 4.3, this correct execution of $\mathcal{T}'$ exists iff $\langle \varphi_S, T_{\varphi_S} \rangle$ is a correct execution of $\mathcal{T}'$, where $\varphi'_S \models \varphi_S$. This correct execution of $\mathcal{T}$ must be an extension of $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$. Since $\langle \varphi_S, T_{\varphi_S} \rangle$ is an extended and correct execution of $\mathcal{T}$ iff $\langle \varphi_{ex}, \tilde{T}_{\varphi_{ex}} \rangle$ is correct with respect to $\mathcal{T}$, we have proven the claim. □

Theorem 4.2 (Dominated Causal Links are Irrelevant)**.** *Dominated labeled causal links do not influence the correctness of an execution.*

Proof. Suppose we have a TPNU $\mathcal{T}$ containing consumer event e_c with precondition p. Further suppose there is a dominating labeled causal link from producer $e_{P_{dom}}$, and a dominated labeled causal link from producer e_P. As a thought experiment, we introduce a modified version of $\mathcal{T}$, which we call $\mathcal{T}_{rem}$, that is exactly the same as $\mathcal{T}$ except that p has been removed from the effects of e_P (and thus the dominated labeled causal link has disappeared). We will prove that if some execution $\langle \varphi_S, T_{\varphi_S} \rangle$ is correct with respect to $\mathcal{T}$, then it is also correct with respect to $\mathcal{T}_{rem}$ – thus demonstrating the irrelevance of the dominated labeled link with respect to correctness, and highlighting that it can be safely removed from consideration.

Proof by contradiction. We assume that execution $\langle \varphi_S, T_{\varphi_S} \rangle$, even though correct with respect to $\mathcal{T}$, is incorrect with respect to $\mathcal{T}_{rem}$. There can be only one way for this to happen: if the precondition p of e_c is no longer satisfied in the execution of $\mathcal{T}_{rem}$ due to p having been removed as an effect from e_P. If this is the case, then e_P must be the supporting producer of e_c in the execution of $\mathcal{T}$ – i.e., e_c must be activated, e_P must be activated, e_P must be the latest-occurring producer before e_c, and no threat event e_T is executed during the time between e_P and e_c. If e_P were not the supporting producer in the execution of $\mathcal{T}$, then removing p from the effects of e_P would have no bearing on the execution correctness since there would be some other supporting producer for e_c or e_c would not be activated.

As e_P is the supporting producer for e_c in the execution of $\mathcal{T}$, e_P must be activated. By the definition of labeled causal link dominance, $\varphi_{e_P} \models \varphi_{e_{P_{dom}}}$, so $e_{P_{dom}}$ must also be activated. Additionally, by the definition of labeled causal link dominance, $e_P \prec e_{P_{dom}} |_{\varphi_{e_c}}$ and $e_{P_{dom}} \prec e_c|_{\varphi_{e_P}}$. Therefore $T_{\varphi_S}(e_P) < T_{\varphi_S}(e_{P_{dom}}) < T_{\varphi_S}(e_c)$. No activated threats may be scheduled in the range between $e_{P_{dom}}$ and e_c, since they are guaranteed to already be scheduled outside the strictly larger range from e_P to e_c. We have therefore shown that the producer $e_{P_{dom}}$ is activated, is a later-occurring producer than e_P, and is not threatened. This contradicts our earlier conclusion that e_P must have been the supporting producer. Therefore, our initial assumption must be incorrect, and so the execution $\langle \varphi_S, T_{\varphi_S} \rangle$ must be correct with respect to $\mathcal{T}_{rem}$. □

Appendix B. Online Execution is NP-Complete

In this section, we prove that Pike's online execution strategy is NP-complete. Specifically, we show that the problem of determining whether a correct execution exists for a TPNU is NP-complete. Note that this is a key operation performed by Pike both offline, and also online during execution as execution proceeds.

Theorem 3.1 (Checking for a Correct Execution is NP-Complete)**.** *The problem of checking if there exists a correct execution $\langle \varphi_S, T_{\varphi_S} \rangle$ for a TPNU $\mathcal{T}$ is NP-complete.*

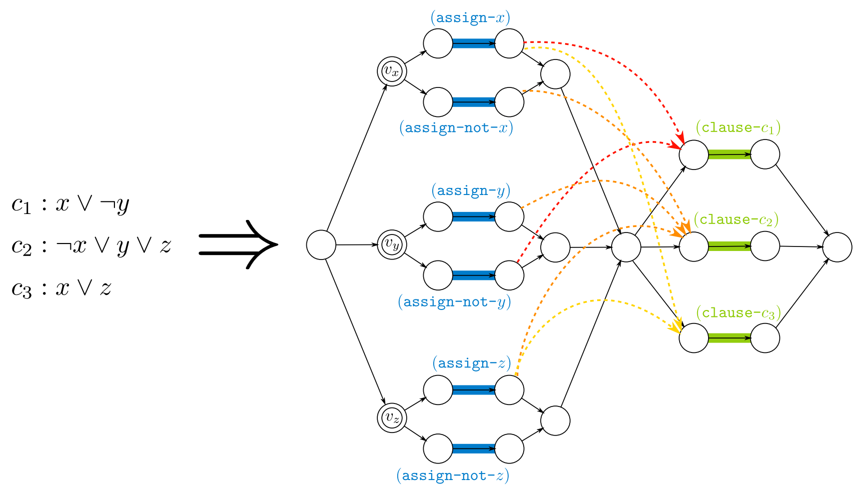


Figure 22: Example transformation of a SAT theory to an equivalent TPNU. The SAT theory is satisfiable if and only if the TPNU is causally complete. The leftmost activities represent assigning SAT variables to true or false. Rightmost activities represent clauses. Dotted arrows represent causal link structure over the (`clause-holds` ...) predicates. For each causal link of a given shade, at least one of the producer activities must be activated for the TPNU admit a correct execution.

Proof. To prove NP-completeness, we show that (1) this problem is in NP, and (2) that it is NP-hard.

To show that this problem is in NP, it suffices to show that a polynomial time algorithm exists that can verify if an execution $\langle \varphi_S, T_{\varphi_S} \rangle$ is correct with respect to $\mathcal{T}$. This can be done easily by (1) checking whether T_{φ_S} satisfies the temporal constraints of $\mathcal{T}$, and subsequently (2) computing the predicted state at each event execution time in T_{φ_S}, and checking if if the preconditions of each event are satisfied by those states. If both of these checks pass, the execution is both temporally consistent and causally complete, and hence correct.

To show that this problem is NP-hard, we describe a polynomial time reduction from the NP-complete SAT problem. Specifically, we show how an arbitrary SAT problem can be translated into a TPNU with a suitable action model such that this TPNU admits a correct execution iff the SAT theory is satisfiable.

In our translation, models of the SAT theory map one-to-one with candidate subplans that admit a correct execution. An example of our encoding is shown in Figure 22. To construct such a TPNU, we first create a small choice structure with two activities for each boolean variable x_i in the SAT problem. We include a finite-domain choice variable v_{x_i} in our TPNU corresponding to x_i. Its domain is $v_{x_i} \in \{\text{True}, \text{False}\}$, corresponding to whether x_i or $\neg x_i$ holds in a model of the SAT theory, respectively. We create an activity for each case: (`assign-`x_i) and (`assign-not-`x_i). Both of these activities have no preconditions, and we will specify the effects of these activities shortly. Our choice structure is such that activity (`assign-`x_i) will be activated and dispatched iff $v_{x_i} = \text{True}$ and (hence x_i holds in the corresponding model of the SAT theory), otherwise (`assign-not-`x_i) will be activated iff $v_{x_i} = \text{False}$ (and $\neg x_i$ holds). Our TPNU contains such a choice structure for each variable in our SAT theory. These structures are placed in parallel in the TPNU.

Temporally after all of these choice structures, we add additional activities to the TPNU corresponding to each disjunctive clause c_i of the SAT theory (without loss of generality, we assume the theory is expressed in CNF). For each clause, we create an activity (`clause-`c_i) with preconditions that will be specified shortly, and with no effects.

All temporal constraints in the TPNU are ordering constraints $[\epsilon, \infty]$.

Finally, we now tie the clause activities to the variable activities, in such a way as to guarantee that the TPNU admits a correct execution iff the SAT theory is satisfiable. We do this by carefully choosing the preconditions of the (`clause-`c_i) activities and effects of the (`assign-`...) activities so as to encode causal link structure. Each clause c_i in the theory contains a set of literals, each denoted l_{ij}. We note that l_{ij} will be either positive (such as x_j) or negative (such as $\neg x_j$). For each clause c_i, we add a precondition to the activity (`clause-`c_i) that is denoted (`clause-holds` c_i). For each literal l_{ij} in clause c_i, we add an effect to the corresponding activity (`assign-`l_{ij}) of (`clause-holds` c_i).

This completes our translation from SAT to a TPNU, which can be performed in polynomial time. We now show its correctness; namely that the SAT theory is satisfiable iff the TPNU admits a correct execution.

Let φ_S be some team scenario of the TPNU, and let A be the corresponding full assignment of the SAT theory. Additionally, let T_{φ_S} be any schedule for φ_S satisfying the temporal constraints (one is guaranteed to exist due to our problem's temporal constraints). The execution $\langle \varphi_S, T_{\varphi_S} \rangle$ is temporally consistent. We show that it is causally complete (and hence correct) iff A is a model of the SAT theory. To do this, we show that the preconditions of (`clause-`c_i) will be satisfied in execution $\langle \varphi_S, T_{\varphi_S} \rangle$ iff the corresponding clause c_i in the SAT theory is satisfied by A. This is because each (`clause-`c_i) has the single precondition (`clause-holds` c_i), which will hold iff at least one of the activities (`assign-`l_{i0}), (`assign-`l_{i1}), ... is activated by φ_S. This is analogous to clause c_i holding iff at least one of the literals $l_{i0}, l_{i1}, ...$ is true. Therefore, all activities in the TPNU will have their preconditions met (and hence the execution is correct) iff all of the clauses in the SAT theory are satisfied (and hence the theory is satisfiable).

□

Appendix C. Extensions to the Labeled APSP

In this appendix, we describe a modified version of the labeled all-pairs shortest path algorithm introduced in Drake (Conrad & Williams, 2011). This modification is designed to improve the quality of querying the individual matrix entries, and relies on computing new labeled values logically implied by others. We first introduce those new labeled values below, and then proceed to describe our modified labeled APSP algorithm.

C.1 Resolutions for LVSs

We now present a novel extension to the original LVS that improves compactness and the tightness of the query operator for finite-domain variables through a form of resolution.

Consider an example LVS in which we have a single discrete-domain variable $x \in \{1, 2\}$, and the constraint on t with the following LVS L:

$$t < \{(3, \{x = 1\}), (4, \{x = 2\})\}$$

What is the value $Q_L(\{\})$, the tightest constraint on t over all environments? Since $\{\}$ entails neither $\{x = 1\}$ nor $\{x = 2\}$, neither labeled value in the LVS apply, and hence the LVS cannot guarantee any bound. $Q_L(\{\}) = \infty$, so we are left with the loosest possible constraint $t < \infty$.

We can improve the tightness of the returned constraint, however, if we consider the discrete domain of the variable x. We add *completions* to the LVS, which intuitively are new labeled value sets logically implied by others in the LVS. These completions allow the query operation to return tighter values.

Returning to the above example, the LVS actually can guarantee the bound $t < 4$, which is of course much tighter than $t < \infty$. This is because x will be either 1 or 2 in any scenario, so one of the two labeled values must apply. We can thus be sure that the "loosest" possible value (here, 4) will hold for the constraint. So, given the domain of x, this LVS could equivalently be represented as $\{(3, \{x = 1\}), (4, \{\})\}$.

Here is another example. Suppose we have

$$t < \{(-1, \{x = 1, y = 2\}), (-2, \{x = 2, y = 2\}), (\infty, \{\})\}$$

where $x, y \in \{1, 2\}$. We are interested in the query $Q_L(\{y = 2\})$. The query operation on this LVS would result in $t < \infty$ as the tightest constraint, but again, we can do better. Noting that if $y = 2$ then either one of $\{x = 1, y = 2\}$ or $\{x = 2, y = 2\}$ must hold, we can guarantee the looser of these labeled values, namely $t < -1$.

In general, when computing $Q_L(\varphi)$ and L contains a set of labeled values whose environments partition $\mathcal{S}(\varphi)$, then at least one of these labeled values must hold true. We may therefore take the loosest constraint value among these labeled values.

Theorem C.1. *Suppose we have a discrete variable $x \in \{v_1, v_2, \ldots, v_k\}$, and an LVS over t with relation $<_R$ of the form*

$$t <_R \{(a_1, \varphi_1 \wedge \{x = v_1\}), (a_2, \varphi_2 \wedge \{x = v_2\}), \ldots, (a_k, \varphi_k \wedge \{x = v_k\}), \ldots\}$$

where $\varphi_1, \ldots, \varphi_k$ are arbitrary environments such that

$$\varphi_M = \bigwedge_{i=1}^{k} \varphi_i \neq \perp$$

Then, we can guarantee that $\varphi_M \Rightarrow t \leq_R \max_R \Big\{a_1, a_2, \ldots, a_k\Big\}$.

Proof. Since $\varphi_M = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$, we know that $\varphi_M \models \varphi_i$ for all $i = 1 \ldots k$. Thus, if φ_M holds, then each of the φ_i must hold. We also know that in any scenario, precisely one of the $x = v_j$ constraints must hold. Therefore, exactly one of the $\varphi_i \wedge \{x = v_i\}$ environments must hold, if φ_M holds. In other words, $\varphi_M \models \varphi_i \wedge \{x = v_i\}$ for exactly one i. We do not know which i however, and hence which labeled value applies, but we know that exactly one does. As such, when φ_M holds, we conservatively take the loosest possible constraint among the i labeled values, namely $t \leq_R \max_R \Big\{a_1, a_2, \ldots, a_k\Big\}$. ∎

Building upon this theorem, we devise the following inference rule to augment an LVS with additional implied labeled values:

$$\frac{\begin{array}{c}(a_1, \varphi_1 \cup \{x = v_1\}) \\ (a_2, \varphi_2 \cup \{x = v_2\}) \\ \cdots \\ (a_k, \varphi_k \cup \{x = v_k\})\end{array}}{\left(\max_i {}_R \{a_i\},\ \ \bigwedge_i \varphi_i\right)}$$

where $x \in \{v_1, ..., v_k\}$ and $\bigwedge_i \varphi_i \neq \perp$.

This inference rule for LVSs is analogous to hyper-resolution in boolean logic. Since the new labeled value is logically implied by the other labeled values (i.e., an implicant), it can safely be added to L without changing correctness while improving the tightness of the query operator.

Algorithm 8: FindLVSCompletions(L, x)

Input: An LVS L, and a variable x over which to find completions
Output: An LVS C of completions, suggesting pairs to add.

```
1  W ← {(−∞_R, {})}
2  foreach assignment x = v_i in x's domain do
3      Y = {}
4      foreach (a_L, φ_L) ∈ L do
5          if φ_L assigns x = v_i then
6              φ_R = φ_L \ {x = v_i}
7              AddLVS((a_L, φ_R), Y)
8          end
9      end
10     W ← LVSBinaryOp(max_R(·,·), Y, W)
11 end
12 return W
```

The pseudo code for an algorithm to find all such completions, FindLVSCompletions, is outlined in Algorithm 8. This algorithm finds completions over the variable x for LVS L. It maintains an LVS W, which is incrementally grown to contain all completions that could be added to L. W starts containing only the weakest possible completion, $(\infty_R, \{\})$. W is then updated to contain completions for each possible variable assignment $x = v_i$ in x's domain. For each assignment $x = v_i$, a temporary LVS Y is produced that contains all pairs consistent with that assignment but stripping off $x = v_i$ from the label of the value. This happens in Lines 3 – 9. Line 10 incrementally computes multiple possible φ_M, along with the corresponding maxima by employing a binary operation between W and Y to update W.

Algorithm 9: LABELEDTIGHTFLOYDWARSHALL

Input: A TPNU $\langle \mathcal{V}, \mathcal{E}, \mathcal{C}, \mathcal{A} \rangle$

Output: $D_{i,j}$, a matrix of shortest path distances between pairs of events in $\mathcal{E}$ (each entry an LVS)

1 **foreach** $i, j \in \mathcal{E}$ **do**
2     $D_{ij} \leftarrow \{(\infty, \{\})\}$
3 **end**
4 **foreach** $i \in \mathcal{E}$ **do**
5     $D_{ii} \leftarrow \{(0, \{\})\}$
6 **end**
7 **foreach** $\langle i, j, l, u, \varphi \rangle \in \mathcal{C}$ **do**
8     $D_{ij} \leftarrow \{(u, \varphi)\}$
9     $D_{ji} \leftarrow \{(-l, \varphi)\}$
10 **end**
11 **foreach** $k \in \mathcal{E}$ **do**
12   **foreach** $i \in \mathcal{E}$ **do**
13   **foreach** $j \in \mathcal{E}$ **do**
14             $C_{ij} = \text{LVSBINARYOP}(+, D_{ik}, D_{kj})$
15             $D_{ij} \leftarrow \text{MERGEWITHCOMPLETIONS}(D_{ij}, C_{ij})$
16   **end**
17   **end**
18 **end**
19 **return** D

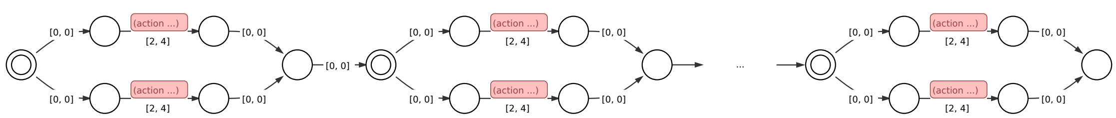


Figure 23: A TPNU with N choices and completely symmetric temporal constraints. Without completions, the LVS in $D_{i,j}$ from the last event to the first event would in general contain 2^N labeled values. With completions, it would contain just one.

C.2 Labeled APSP

We present a modified version of the original labeled APSP algorithm extended to generate LVS completions as described earlier. For the original version, please see Drake (Conrad, 2010). Pseudo code for the labeled APSP is shown in Algorithm 9. It takes in a TPNU with events $\mathcal{E}$ and temporal constraints $\mathcal{C}$, performs the above transformation to a labeled distance graph implicitly, and then runs a generalized version of Floyd Warshall. The algorithm begins in Lines 1 – 10 by initializing a matrix $D_{i,j}$ with a new LVS for each pair of events. The shortest distances from every event to itself is $\{(0, \{\})\}$. Other off-diagonal entry weights are added corresponding to each episode, and others are all set to $\{(\infty, \{\})\}$.

Lines 11 – 18 provide the signature "triple for loops" of the Floyd Warshall algorithm. The key difference between our labeled APSP algorithm and the original presented in Drake

Algorithm 10: MERGEWITHCOMPLETIONS(A, L)

Input: An LVS A, and another LVS L that should be merged into A
Output: Returns updated A

1 $Q \leftarrow \{\}$
2 **foreach** *pair* (a_l, φ_l) *in* L **do**
3   added? = ADDLVS$(A, (a_l, \varphi_l))$
4   **if** *added?* **then**
5   Push each variable listed in φ_l to Q if not already present
6   **end**
7 **end**
8 **while** Q *is not empty* **do**
9     $x \leftarrow$ pop variable from Q
10     $C \leftarrow$ FINDLVSCOMPLETIONS(A, x)
11   **foreach** *pair* (a_c, φ_c) *in* C **do**
12   added? = ADDLVS$(A, (a_c, \varphi_c))$
13   **if** *added?* **then**
14   Push each variable listed in φ_c to Q if not already present
15   **end**
16   **end**
17 **end**
18 **return** A

is shown in Line 15, which calls a new method MERGEWITHCOMPLETIONS(D_{ij}, C_{ij}), instead of the original MERGE method. This method computes completions for the LVS, ensuring that queries performed upon it will return the tightest possible values.

Pseudo code for MERGEWITHCOMPLETIONS is shown in Algorithm 10. This method takes as input an LVS A and a second LVS L whose pairs will be added to A. The algorithm first adds each labeled value in L to A, maintaining dominance via ADDLVS. The algorithm also maintains a queue Q over variables for which completions may exist. This queue contains a list of all the variables referenced by any labeled value pair added to A. The second phase of the algorithm, beginning on Line 8, repeatedly pops variables off of Q and calls the FINDLVSCOMPLETIONS algorithm to find new implied labeled values for L given the current variables that were added. Each newly generated completion is added to A, and more variables are possibly pushed onto Q if more resolutions could be possible. Finally, the modified A is returned.

An example where our modified labeled APSP is beneficial is shown in Figure 23. In this example, a TPNU is shown with N sequential and identical sets of choices of two actions, each with temporal bounds of $[2, 4]$. Without the MERGEWITHCOMPLETIONS modification above, the LVS shortest path from the first event to the last event of the plan would contain 2^N pairs; one for each possible team scenario. However, with our addition, this LVS will contain just a single pair.

While operating with completions can in some cases greatly improve the performance of the labeled APSP algorithm, such as in the example in Figure 23, this is not always the case.

We have found that the performance in general is greatly dependent on the numerics of the temporal constraints, and on any encoded "symmetry" across the temporal constraints. For example, if the $[2, 4]$ temporal constraints in Figure 23 were randomized by adding Gaussian noise to them, the performance would drastically decrease, due to the fact that the shortest possible path between different events would now be a much more complex function of what choices are made.

References

Alili, S., Warnier, M., Ali, M., & Alami, R. (2009). Planning and plan-execution for human-robot cooperative task achievement. In *19th International Conference on Automated Planning and Scheduling*.

Ambros-Ingerson, J. A., & Steel, S. (1988). Integrating planning, execution and monitoring. In Shrobe, H. E., Mitchell, T. M., & Smith, R. G. (Eds.), *Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, MN, USA, August 21-26, 1988.*, pp. 83–88. AAAI Press / The MIT Press.

Avrahami-Zilberbrand, D., Kaminka, G., & Zarosim, H. (2005). Fast and complete symbolic plan recognition: Allowing for duration, interleaved execution, and lossy observations. In *Proc. of the AAAI Workshop on Modeling Others from Observations, MOO*.

Ayan, N. F., Kuter, U., Yaman, F., & Goldman, R. P. (2007). Hotride: Hierarchical ordered task replanning in dynamic environments. In *Proceedings of the 3rd Workshop on Planning and Plan Execution for Real-World Systems (held in conjunction with ICAPS 2007)*, Vol. 2.

Benton, J., Coles, A. J., & Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs. In McCluskey, L., Williams, B. C., Silva, J. R., & Bonet, B. (Eds.), *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI.

Bui, H. H. (2003). A general model for online probabilistic plan recognition. In Gottlob, G., & Walsh, T. (Eds.), *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pp. 1309–1318. Morgan Kaufmann.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artif. Intell.*, *69*(1-2), 165–204.

Carberry, S. (2001). Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, *11*(1-2), 31–48.

Chien, S. A., Knight, R., Stechert, A., Sherwood, R., & Rabideau, G. (2000). Using iterative repair to improve the responsiveness of planning and scheduling.. In *AIPS*, pp. 300–307.

Clodic, A., Cao, H., Alili, S., Montreuil, V., Alami, R., & Chatila, R. (2008). SHARY: A supervision system adapted to human-robot interaction. In *ISER*, Vol. 54 of *Springer Tracts in Advanced Robotics*, pp. 229–238. Springer.

Conrad, P. R., Shah, J. A., & Williams, B. C. (2009). Flexible execution of plans with choice. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Conrad, P. R., & Williams, B. C. (2011). Drake: An efficient executive for temporal plans with choice. *Journal of Artificial Intelligence Research*, *42*, 607–659.

Conrad, P. R. (2010). Flexible execution of plans with choice and uncertainty. Master's thesis, Massachusetts Institute of Technology.

Darwiche, A., & Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, *17*(1), 229–264.

de Kleer, J. (1986a). An assumption-based TMS. *Artif. Intell.*, *28*(2), 127–162.

de Kleer, J. (1986b). Extending the ATMS. *Artif. Intell.*, *28*(2), 163–196.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial intelligence*, *49*(1), 61–95.

Effinger, R. T., Williams, B. C., Kelly, G., & Sheehy, M. (2009). Dynamic controllability of temporally-flexible reactive programs.. In *ICAPS*.

Elliott, P. H. (2004). An efficient projected minimal conflict generator for projected prime implicate and implicant generation. Master's thesis, Massachusetts Institute of Technology.

Fikes, R., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, *2*(3/4), 189–208.

Finzi, A., Ingrand, F., & Muscettola, N. (2004). Model-based executive control through reactive planning for autonomous rovers. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, Vol. 1, pp. 879–884. IEEE.

Forbus, K. D. (1993). *Building Problems Solvers*, Vol. 1. MIT press.

Fox, M., & Long, D. (2003). PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, *20*, 61–124.

Freedman, R. G., & Zilberstein, S. (2017). Integration of planning with recognition for responsive interaction using classical planners. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pp. 4581–4588.

Geib, C. W., Weerasinghe, J., Matskevich, S., Kantharaju, P., Craenen, B. G. W., & Petrick, R. P. A. (2016). Building helpful virtual agents using plan recognition and planning. In Sturtevant, N., & Magerko, B. (Eds.), *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2016, October 8-12, 2016, Burlingame, California, USA.*, pp. 162–168. AAAI Press.

Goldman, R. P., Geib, C. W., & Miller, C. A. (1999). A new model of plan recognition. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pp. 245–254. Morgan Kaufmann Publishers Inc.

Haigh, K. Z., & Veloso, M. M. (1998). Interleaving planning and robot execution for asynchronous user requests. In *Autonomous agents*, pp. 79–95. Springer.

Hunsberger, L., Posenato, R., & Combi, C. (2012). The dynamic controllability of conditional STNs with uncertainty. *CoRR*, *abs/1212.2005*.

Ingrand, F., & Ghallab, M. (2017). Deliberation for autonomous robots: A survey. *Artif. Intell.*, *247*, 10–44.

Karpas, E., Levine, S. J., Yu, P., & Williams, B. C. (2015). Robust execution of plans for human-robot teams.. In *International Conference on Automated Planning and Scheduling (ICAPS) Robotics Track*.

Kautz, H. A., & Allen, J. F. (1986). Generalized plan recognition.. In *AAAI*, Vol. 86, pp. 32–37.

Kean, A., & Tsiknis, G. K. (1990). An incremental method for generating prime implicants/impicates. *J. Symb. Comput.*, *9*(2), 185–206.

Keren, S., Gal, A., & Karpas, E. (2014). Goal recognition design. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Kim, P., Williams, B. C., & Abramson, M. (2001). Executing reactive, model-based programs through graph-based temporal planning. In *IJCAI*, pp. 487–493.

Lane, S. D. (2016). Propositional and activity monitoring using qualitative spatial relations. Master's thesis, Massachusetts Institute of Technology.

Lemai, S., & Ingrand, F. (2004). Interleaving temporal planning and execution in robotics domains. In *AAAI*, Vol. 4, pp. 617–622.

Levine, S. J. (2012). Monitoring the execution of temporal plans for robotic systems. Master's thesis, Massachusetts Institute of Technology.

Levine, S. J., & Williams, B. C. (2014). Concurrent plan recognition and execution for human-robot teams. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Mcallester, D., & Rosenblitt, D. (1991). Systematic nonlinear planning. In *In Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 634–639.

Meneguzzi, F., & de Silva, L. (2015). Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *Knowledge Eng. Review*, *30*(1), 1–44.

Mirsky, R., Stern, R., Gal, Y. K., & Kalech, M. (2016). Sequential plan recognition. In Kambhampati, S. (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 401–407. IJCAI/AAAI Press.

Muise, C., Beck, J. C., & McIlraith, S. A. (2013). Flexible execution of partial order plans with temporal constraints. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pp. 2328–2335. AAAI Press.

Muise, C., McIlraith, S., Beck, J. C., & Hsu, E. (2010). Fast d-DNNF compilation with sharpSAT. In *Proceedings of the 8th AAAI Conference on Abstraction, Reformulation, and Approximation*, pp. 54–60. AAAI Press.

Muise, C. J., Beck, J. C., & McIlraith, S. A. (2016). Optimal partial-order plan relaxation via MaxSAT. *Journal of Artificial Intelligence Research*, *57*, 113–149.

Muise, C. J., McIlraith, S. A., & Beck, J. C. (2012). Improved non-deterministic planning by exploiting state relevance. In McCluskey, L., Williams, B. C., Silva, J. R., & Bonet, B. (Eds.), *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI.

Muscettola, N., Morris, P. H., & Tsamardinos, I. (1998). Reformulating temporal plans for efficient execution. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998.*, pp. 444–452.

Nikolaidis, S., & Shah, J. A. (2013). Human-robot cross-training: computational formulation, modeling and evaluation of a human team training strategy. In Kuzuoka, H., Evers, V., Imai, M., & Forlizzi, J. (Eds.), *ACM/IEEE International Conference on Human-Robot Interaction, HRI 2013, Tokyo, Japan, March 3-6, 2013*, pp. 33–40. IEEE/ACM.

Pattison, D., & Long, D. (2010). Domain independent goal recognition. In Ågotnes, T. (Ed.), *STAIRS 2010 - Proceedings of the Fifth Starting AI Researchers' Symposium, Lisbon, Portugal, 16-20 August, 2010*, Vol. 222 of *Frontiers in Artificial Intelligence and Applications*, pp. 238–250. IOS Press.

Pecora, F., Cirillo, M., Dell'Osa, F., Ullberg, J., & Saffiotti, A. (2012). A constraint-based approach for proactive, context-aware human support. *Journal of Ambient Intelligence and Smart Environments*, *4*(4), 347–367.

Penberthy, J. S., & Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In Nebel, B., Rich, C., & Swartout, W. R. (Eds.), *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, USA, October 25-29, 1992.*, pp. 103–114. Morgan Kaufmann.

Quine, W. V. (1959). On cores and prime implicants of truth functions. *The American Mathematical Monthly*, *66*(9), 755–760.

Ramírez, M., & Geffner, H. (2010). Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

Robinson, J. A. (1965). Automatic deduction with hyper-resolution. *International journal of computer mathematics*, *1*(3), 227–234.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: a modern approach*. Prentice hall.

Santana, P. H., Thiébaux, S., & Williams, B. (2016). RAO*: An algorithm for chance-constrained POMDP's. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Santana, P. H., & Williams, B. (2014). Chance-constrained consistency for probabilistic temporal plan networks. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Shah, J. A., Conrad, P. R., & Williams, B. C. (2009). Fast distributed multi-agent plan execution with dynamic task assignment and scheduling.. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Sreedharan, S., Chakraborti, T., & Kambhampati, S. (2018). Handling model uncertainty and multiplicity in explanations via model reconciliation. In de Weerdt, M., Koenig, S., Röger, G., & Spaan, M. T. J. (Eds.), *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018.*, pp. 518–526. AAAI Press.

Sukthankar, G., Geib, C., Bui, H. H., Pynadath, D., & Goldman, R. P. (2014). *Plan, Activity, and Intent Recognition: Theory and Practice* (1st edition). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Tsamardinos, I., Muscettola, N., & Morris, P. H. (1998). Fast transformation of temporal plans for efficient execution. In Mostow, J., & Rich, C. (Eds.), *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pp. 254–261. AAAI Press / The MIT Press.

Tseitin, G. S. (1968). On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, *2*, 115–125.

Veloso, M. M., Pollack, M. E., & Cox, M. T. (1998). Rationale-based monitoring for planning in dynamic environments.. In *AIPS*, pp. 171–180.

Vidal, T. (1999). Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, *11*(1), 23–45.

Vidal, T. (2000). A unified dynamic approach for dealing with temporal uncertainty and conditional planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pp. 395–402.

Wang, D., & Williams, B. (2015). tburton: A divide and conquer temporal planner. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Williams, B. C., Ingham, M. D., Chung, S. H., & Elliott, P. H. (2003). Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE*, *91*(1), 212–237.

Williams, B. C., & Ragno, R. J. (2007). Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, *155*(12), 1562–1595.

Yu, P., Fang, C., & Williams, B. C. (2014). Resolving uncontrollable conditional temporal problems using continuous relaxations.. In *International Conference on Automated Planning and Scheduling (ICAPS)*.