Distributed Multi-Agent Decision Making Under Uncertain Communication

by

Cameron W. Pittman

M.A., Belmont University (2011)
B.A., Vanderbilt University (2009)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2023

Author ....
Department of Aeronautics and Astronautics
August 15, 2023

Certified by..
Brian C. Williams
Professor of Aeronautics and Astronautics, MIT
Thesis Supervisor

Accepted by ....
Jonathan How
R.C Maclaurin Professor of Aeronautics and Astronautics, MIT
Chair, Graduate Program Committee

Distributed Multi-Agent Decision Making Under Uncertain Communication

by

Cameron W. Pittman

Submitted to the Department of Aeronautics and Astronautics
on August 15, 2023, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This is an abstract.

Thesis Supervisor: Brian C. Williams
Title: Professor of Aeronautics and Astronautics, MIT

Acknowledgements

My journey to this SM thesis began eight years ago when I was a software engineer living in San Francisco. Two years prior, I had just wrapped four years of teaching high school science and didn't know how to code. I taught myself Python using MIT OpenCourseware on a whim because it was something I had always wanted to learn. Coding took over my life the first time I wrote a function. My first projects were physics demos and websites, but soon I learned I loved helping systems make decisions. And there is absolutely no system cooler than human spaceflight. So I sat there at my desk in SF and wondered, "I want to be a part of the next space race! What if I studied autonomy for human spaceflight? Where would I even go to learn autonomy?" (I admit I didn't know the field was called "autonomy" at the time.) Of course, MIT came up when I started searching. It took five more years of studying computer science, finding people to learn from, and integrating in the human spaceflight community, but eventually I found my way into AeroAstro and the MERS lab. It's been the journey of a lifetime and there's no way I would be here without the love and support from so many people.

First, I have to thank my amazing wife, Mo, who is the hardest working, most driven person I've ever met. She's been my biggest cheerleader and sounding board for my ideas (even when she has no idea what I'm talking about). Our newborn daughter, Catalina, is my inspiration. I think about her living in a world where there are thousands of people living and working in space. At the time of writing this, she's a month old and already making me work harder than I ever have. I never would have made it this far without a supportive family. My parents, Mark and Suzanne, and my brother, Max, have always been there for me.

My advisor, Brian Williams, has consistently surprised me with how much support he's freely given. Brian has been one of the most helpful people I've ever met, starting with the first time I introduced myself and blurted out, "hi, I'm taking your class and I think it's amazing and I want to use everything at work and can I please join your lab?" From academic mentoring, to brainstorming executive architectures, to explaining basic concepts in temporal reasoning, to envisioning the next generation of online education, he's been open, honest, and eager to share ideas. Brian, you have had a profound impact on my life and career, and for that I'm forever grateful.

Next, I have to thank all my labmates. I can't really quantify the level of impact any single person has had, so I'll be doing these chronological order of when we met.

The first people I met were my 16.413 TAs, Simon Fang and Sungkweon Hong. I would go to office hours even when I didn't have questions just because I wanted to hang out and talk autonomy with people. Simon, thank you for being a supportive TA (and introducing me to Aunty Donna). Sungkweon,

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of source codes

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

This is the introduction.

1.1 Section Header

This is a section of text. As said in... [1] “Hey”. So said [2] too that things are cool.

```
(format t "hello, world!")
```

Listing 1: This is a caption.

This is something I want to cite [3]

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Problem Statement

There is a real-world need for coordinating multiple agents that are collaborating while facing uncertain inter-agent communication in uncertain environments. This thesis will focus on two motivating scenarios drawn from collaborative space operations, though this section will include a third example from the domain of military operations to further motivate the need for modeling uncertain delay in observations. The first scenario is based on an Artemis-like extravehicular activity (EVA), where an astronaut and a rover on the lunar surface are required to coordinate to share limited downlink bandwidth. The second scenario comes from the domain of satellite swarms, where many homogeneous agents coordinate operations with tight temporal constraints. The third scenario (which will not be modeled in experiments) is one where cooperative military agents are moving into and out of regions where immediate communication is purposefully halted in order to avoid detection by an enemy.

Delay temporal networks are essential to modeling space operations, where uncertainty is systemic to all aspects of planning, scheduling, and execution. As will be described in more detail for each motivating scenario below, uncertain observation delay is unavoidable due to imperfect communication infrastructure, uncertain timing with respect to knowledge transfer between agents, and finicky scientific equipment.

2.1 Extravehicular Activities as a Motivating Scenario

NASA is preparing to send astronauts back to the lunar surface as part of the Artemis program[1]. The Artemis astronauts will continue a longstanding tradition in the U.S. space program of performing EVAs [4]. Like the Apollo astronauts of the 1960s and 1970s, Artemis astronauts will embark on EVAs wherein crews don spacesuits, egress landers, and conduct scientific expeditions on the lunar surface. There, they will survey surface features, collect samples, and in general perform field geology

[1]`https://www.nasa.gov/specials/artemis/`

[5][7]. A small number of astronauts are Ph.D. geologists by trade, and NASA is training others in the principles of field geology up to a notional masters level of understanding [8]. NASA will support the lunar activities of these astronauts through a vast infrastructure of personnel on the ground, including teams of domain-relevant scientists. Together with flight controllers and engineers from various disciplines, a science team on the ground will provide real-time feedback to ensure that Artemis astronauts maximize the scientific return of their EVAs.

The relevant actors in an EVA include extravehicular (EV) crew members, who conduct all field activities outside the vehicles and habitats, and a ground-based Mission Control Center (MCC). Typically there are two EV crew members who often, but not always, work together to complete tasks. Life support imposes a temporal bound on the overall length of excursions. As an EVA progresses, EV crews consume four non-renewable resources, comprised of oxygen, battery power, water, and CO_2 scrubbers [9]. The duration of EVAs is limited by the consumable that is on track to be depleted first across the life support systems of both EV crew members, referred to as the limiting consumable.

Mission Control Center (MCC) is comprised of hundreds of flight controllers and engineers who monitor all aspects of EVAs. Its strict hierarchical structure ensures that all space-to-ground decisions pass through the Flight Director [10]. For the purposes of this example, we treat MCC as a single actor.

When astronauts perform field science, another actor comes into play, called the ground-based Science Backroom Team (SBT). The SBT is comprised of multidisciplinary scientists who help astronauts prioritize and select scientific sample targets online [5], [11]. The science team reports their priorities to MCC, who then passes them along to the crew. The SBT behaves as a separate actor with limited communication, in that their messages may only pass directly to MCC, not the crew.

For EVAs, delay is manifested across three distinct categories: signal transmission, human operational delays, and instrument processing. Each category presents a source of delay that varies with uncertainty. For signal transmission, delay is sourced from the speed of light between planetary bodies and infrastructural deficiencies. Communication infrastructure outside of low-Earth orbit, including satellites and planetary surface signal repeaters [12], is not robust, and as such unpredictable delays and signal dropouts will be common [13]. For human operational delays, note that the primary goal of Mission Control is keeping the crew safe [14]. As such, communications from the science team to the crew may be delayed or dropped because Mission Control needs to prioritize communications and actions related to crew health and safety at the expense of science [15], [16]. Lastly, for instrument processing, there is uncertainty in the temporal relationships between the activation of complex scientific instruments and the return of useful information NO_ITEM_DATA:Sehlke2019. Scientific packages may generate high and low bandwidth data products, the uplink of which will be bottlenecked by limited bandwidth between space and ground, creating additional uncertainty between when a data product is ready and when it is available to the science team.

2.2 Motivating Scenario

A human and a rover are working together to perform scientific exploration on the lunar surface. The human is performing exploration of targets of opportunity that were identified during descent. The rover has a robotic arm, which it is using to perform sampling tasks collecting rocks in a predetermined location.

Both are in communication with mission control and scientists on Earth. Importantly, they share bandwidth on the relays and satellites used to transmit data between the Moon and Earth. CONOPS says only one agent may be using the bandwidth at a time (including uploads from Earth)

Both the human and the robot have their own schedules of events to perform and are aware of the other agent's events.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Approach

For addressing our problem statement, we envision that a high-level executive should take responsibility for managing task planning and execution with respect to temporal constraints. For this thesis, we chose to extend an existing high-level task and motion planner, *Kirk* [18]. Kirk is a complete, end-to-end executive in that it can take human-friendly problem specifications as input and send commands to hardware as output.

To clarify terminology in this thesis, the term *executive* refers to Kirk and its subsystems, while *agent* refers to the combination of an executive and the system it controls that interacts with the outside world, e.g. robotic hardware.

At a high-level, Kirk works by first taking a description of the problem domain as written by domain experts, which should include the constraints, agent dynamics, environment, and starting and goal states of the problem at hand. Kirk then generates and checks plans using an optimal satisfiability (OpSAT) solver [19], elaborates plans to sub-executives when it encounters constraints and goals it cannot plan against directly, and eventually dispatches event schedules and motion plans to hardware. For the purpose of this thesis, we primarily focus on Kirk's capability to dispatch events, though fully accounting for uncertain communication in a real agent requires that all of Kirk's aforementioned capabilities are addressed.

Some aspects of Kirk were already well-suited for coordinating multiple agents under observation delay, others were not. Specifically, our approach required research contributions in three key areas, which were then implemented in Kirk:

1. *Modeling and Controllability*: prior to execution, we must be able to model communication delay separate from temporal constraints, as well as guarantee that all temporal constraints can be satisfied
2. *Scheduling*: during execution, executives must be able to dynamically schedule and dispatch events respecting temporal constraints in spite of observation delay

3. *Coordination*: during execution, peer executives must be able to share event assignments and observations

We include an auxiliary fourth area for contributions as an engineering requirement.

4. *Robustness*: Kirk's algorithms must be based in realistic computing constraints, and Kirk must be easy to run, debug, integrate with existing autonomous systems

Our approach to each research focus will be described below.

3.1 Modeling and Controllability

We take a model-based approach to deploying autonomous systems, that is, prior to a mission, we envision that engineers and domain experts work together to model the system at hand, then during the mission (though not necessarily online), the autonomous system then takes the models as input and decides how to act as output. There are three core challenges with modeling - the first being that we need formalisms that can be ingested by our algorithms and be used to guarantee the safe execution. In other words, we need a data type to represent the phenomenon over which we want the algorithms comprising our system to reason. Next, the chosen formalism must allow us to guarantee the satisfiability of the system in that the autonomous system must be able to act in a safe manner respecting all constraints to go from the starting state to the goal state. Finally, the third challenge is that we need a human-friendly form of said formalisms such that human domain experts, who are unlikely to also be experts in autonomy, can still model their domains accurately enough such that the desired safe behavior is output by the autonomous system. We address both challenges in our approach to modeling.

States and constraints can take on arbitrary forms, and how they are modeled depends entirely on the problem domain. Classical planning problems use boolean predicates and actions to model the world (e.g. STRIPS planning problems [20]). Scheduling problems involving time constraints will have continuous temporal bounds between discrete timepoints (e.g. in the form of temporal constraint graphs [21]). Other scenarios where motion planning is the focus will likely be modeled with vectors of continuous values in $\mathbb{R}$ (e.g. often representing convex regions as in the case of the *Magellan* planner [22]). Hybrid domains combine states and constraints with mixed continuous and discrete values (e.g. using mixed-integer linear programs as demonstrated by Chen et al. [23]).

Given this thesis' emphasis on temporal scheduling, we choose to focus entirely on formalisms where states and constraints are temporal in nature. The starting state of the system is, by definition, one where time is set to 0 seconds, $t = 0$, and no events have been executed (i.e. no event assignments have been made). We then define controlled and uncontrolled set-bounded constraints between events. The goal state is one where times have been assigned to each controllable event such that all constraints are satisfied. To do so, we build our formalisms representing temporal constraints with set-bounded observation delay on top of simple temporal networks with uncertainty (STNUs) [24]. A brief explanation of our modeling strategy for temporal constraints with observation delay follows in Section 3.1.1, though we will elaborate on temporal reasoning and our chosen formalisms for it in much more detail in Chapter 4.

With a modeling formalism in hand, the second key challenge is to use the formalism to guarantee a property known as *controllability*, or that all controllable temporal constraints can be satisfied given the existing uncertainty in the STNU. There already exist a number of strategies for checking the controllability of STNUs. Examples of different strategies include the canonical work by Morris, Muscettola, and Vidal in checking for semi-reducible negative cycles (SRNCs) [25][28], as well as more exotic approaches like reframing controllability as a Satisfiable Modulo Theory (SMT) problem [29]. In our approach to controllability under observation uncertainty, we build on top of checks for SRNCs as will be shown in 4.3.

For the third challenge, we choose to extend the Reactive Model-Based Programming Language (RMPL) [30], which provides to domain experts a means for describing the constraints and goal states of their domain without requiring additional expert knowledge in autonomy. With RMPL, a human planner is capable of building control programs describing the constraints, agents, and states of the problem domain in a way that is human-readable yet highly programmable, and is independent of the underlying algorithms used by the autonomous system. As will be explained in Section 3.1.2 below, our approach was to add the ability for planners to model observation delay alongside temporal constraints in RMPL.

3.1.1 Modeling Uncertain Observation Delay in STNUs

In the case of observation delay, our model dictates that we reason over two time intervals. The first time interval represents the true length of time between two events, while the second interval represents the length of time between when an event occurs and when an executive observes the event. For ensuring that an executive takes safe actions in an uncertain environment, we assume worst-case scenario with respect to information gain. Our approach to modeling uncertain observation delay in STNUs is as follows.

1. The duration of time between two events is represented as a set-bounded interval
2. The duration of time between an event and its observation (observation delay) is represented as a set-bounded interval
3. Timestamps in event observations are ignored
4. The true duration of observation delay is not guaranteed to be learned

The first point comes directly from the STNU formalism (see Section 4.1). The second point allows for uncertainty in the amount of observation delay, e.g. in an uncertain environment, we could model observation delay for a given event as, say, $[1, \infty]$, meaning an observation of an event could arrive one second after it occurs, or never arrive, or arrive at some arbitrary time, t, $1 < t \leq \infty$ later. The third point comes from assuming worst-case scenario and prevents us from "cheating" in our scheduling algorithm. For instance, imagine two agents coordinating. If agents passed timestamp information along with events to one another, they must also be able to synchronize their clocks, potentially to an arbitrary degree of precision. The challenge of synchronizing clocks between agents is outside the scope of this thesis and may not always be possible. As such, executives only trust their own clocks. Rather than backfill potentially erroneous times for event assignments as reported by exogenous sources, the executive we envision in this thesis records times that are internally consistent with its own clock. Doing so guarantees that the actions the executive takes as a result of temporal reasoning are consistent with its model.

The fourth point, that we are not guaranteed to learn event assignments, is a result of the first three. It stands to reason that an event observation is a function of the true assignment of an event and its observation delay. If there is uncertainty in both the event assignment and delay, then we have one equation with two unknowns. Thus, the term "uncertain" in uncertain observation delay means that we are forced to reason with deciding when to act even when we are not guaranteed to learn the true times assigned to events.

We call STNUs with variable observation delay *variable-delay STNUs*, which Bhargava first proposed as the underlying data structure for checking Variable-Delay Controllability (VDC) [1], [31]. We (Pittman) co-authored a journal article with Bhargava that was submitted to the Journal of AI Research presenting VDC and its chance constrained variant. We include VDC as a contribution of this thesis, given that we (Pittman) wrote or rewrote a significant portion of the VDC article, notably including a rewrite of key proofs with novel explanations. The new proofs will will be presented in Section 4.3. Additionally, we rewrote the comparison of VDC to Partially Observable STNUs (POSTNUs) [32], including identifying and correcting a mistake in the same comparison as originally put forth by Bhargava in [31]. See Appendix A for an in-depth comparison to POSTNUs. We designed and ran the quantitative evaluation of VDC in the article. The same experiments will be included at the end of Chapter .

We formalize event observations and observation delay in Section 4.3.

3.1.2 Modeling Observation Delay in RMPL

RMPL is a key component of Kirk. This section steps through example RMPL control programs to describe their features and our modeling choices. The purpose of this section is three-fold:

1. A short walkthrough of the language is required in order to explain this thesis' contributions because an updated RMPL description in any form (e.g. manual, publication, or tutorial) has

not been publicly released since 2003 [18]

2. We must describe the modeling choices of RMPL in sufficient detail to make concrete our approach to modeling temporal constraints in human-readble form
3. The above is used to demonstrate that modeling uncertain communication delay can be naturally modeled in RMPL

This section is not meant to be a complete documentation of RMPL, rather our goal is to motivate the strength of RMPL as a modeling language for human planners describing autonomous systems with observation uncertainty.

RMPL has undergone a number of rewrites since its inception, and is currently being developed as a superset of the Common Lisp language using the Metaobject Protocol [33]. The goal is that a human should have a comfortable means for accurately modeling sufficient detail about the problem domain such that an executive can perform model-based reasoning to decide how to act.

RMPL and Kirk can be used to achieve a number of different goals. These include but are not limited to temporal scheduling, classical planning, hybrid planning. For this thesis, we focus on temporal scheduling and the ability for a human to write *control programs*, or composable constraints and goals.

For this thesis, we take the assumption that each Kirk executive is responsible for a single agent. We also ignore vehicle dynamics given this thesis' focus on contributions to temporal scheduling. However, RMPL is more flexible and allows multi-agent planning and motion planning using vehicle dynamics, which will be briefly described in Section 3.1.3.

An example of an RMPL control program for a single-agent without agent dynamics follows in Listing 2.

```lisp
;; NOTE: we omitted Lisp package definitions here for simplicity's sake

(define-control-program eat-breakfast ()
  (declare (primitive)
           (duration (simple :lower-bound 15 :upper-bound 20))))

(define-control-program bike-to-lecture ()
  (declare (primitive)
           (duration (simple :lower-bound 15 :upper-bound 20))))

(define-control-program main ()
  (with-temporal-constraint (simple-temporal :upper-bound 40)
    (sequence (:slack nil)
              (eat-breakfast)
              (bike-to-lecture))))
```

Listing 2: A sample control program composed of three constraints. `eat-breakfast` and `bike-to-lecture` designate controllable constraints, while the `main` control program enforces that the constraints are satisfied in series.

Looking past the parentheses, we can see different options for defining temporal constraints. For example, the `(duration (simple ...))` form is used to define a set-bounded temporal constraint between a `:lower-bound` and an `:upper-bound`. The `main` control program uses a different form, `(with-temporal-constraint ...)` to place an `:upper-bound` on the overall deadline for scheduling all events in the control program.

The example control programs in Listing 2 are defined without agents in that there is an assumption that the Kirk instance that executes this control program must know what the semantics of `eat-breakfast` and `bike-to-lecture` mean and how to execute them.

It could also be the case that Kirk is simply being used to produce a schedule of events offline that will be handed to an agent that knows how to execute them. As an example, perhaps a student wants some help planning their morning, so they write an RMPL control program with constraints representing everything they need to do between waking up and going to lecture, as seen in the more complex control program in Listing 3. The student could ask Kirk to produce a schedule of events that satisfies all the temporal constraints in this RMPL control program, which they would then use to plan their morning routine. See the resulting schedule produced by Kirk in Table 3.1. (Note that while normally times in RMPL are represented in seconds, we use minutes in Listing 3 and Table 3.1 for simplicity's sake.)

Table 3.1: The schedule produced by Kirk's scheduler for the student's routine before lecture as modeled in Listing 3. Note: Kirk's output has been cleaned for readability purposes.

Event	**Time (min)**
`START`	0
Start `shower`	1
End `shower`	6
Start `review-scheduling-notes`	6
Start `eat-breakfast`	6
End `review-scheduling-notes`	16
Start `review-planning-notes`	16
End `eat-breakfast`	21
End `review-planning-notes`	26
Start `pack-bag`	26
End `pack-bag`	31
Start `bike-to-lecture`	32
End `bike-to-lecture`	46
`END`	46

Listing 3 introduces the notion of control programs that are allowed to be executed simultaneously, as modeled with the `(parallel ...)` form found in the `main` control program on line 48.

Kirk is able to simulate the RMPL script in Listing 3 and produce a schedule because there were no uncontrollable constraints, that is, all control programs are under the agent's control. Say we replaced `bike-to-lecture` with `drive-to-lecture`. Due to traffic conditions, driving presents in

```
;; This file lives in the thesis code repo at:
;;      kirk-v2/examples/morning-lecture/script.rmpl
;;
;; To execute this RMPL control program as-is and generate a schedule, go to the root
;; of the thesis code repo and run the following command:
;;
;; kirk run kirk-v2/examples/morning-lecture/script.rmpl \
;;      -P morning-lecture \
;;      --simulate

(rmpl/lang:defpackage #:morning-lecture)

(in-package #:morning-lecture)

(define-control-program shower ()
  (declare (primitive)
           (duration (simple :lower-bound 5 :upper-bound 10))))

(define-control-program eat-breakfast ()
  (declare (primitive)
           (duration (simple :lower-bound 15 :upper-bound 20))))

(define-control-program review-scheduling-notes ()
  (declare (primitive)
           (duration (simple :lower-bound 10 :upper-bound 15))))

(define-control-program review-planning-notes ()
  (declare (primitive)
           (duration (simple :lower-bound 10 :upper-bound 15))))

(define-control-program pack-bag ()
  (declare (primitive)
           (duration (simple :lower-bound 5 :upper-bound 6))))

(define-control-program bike-to-lecture ()
  (declare (primitive)
           (duration (simple :lower-bound 15 :upper-bound 20))))

(define-control-program review-notes ()
  (sequence (:slack t)
    (review-scheduling-notes)
    (review-planning-notes)))

(define-control-program main ()
  (with-temporal-constraint (simple-temporal :upper-bound 60)
    (sequence (:slack t)
      (shower)
      (parallel (:slack t)
        (eat-breakfast)
        (review-notes))
      (pack-bag)
      (bike-to-lecture))))
```

Listing 3: A student's morning routine preparing for lecture as modeled in RMPL. This is a complete RMPL program that includes the required Lisp package definitions to run in Kirk.

an uncontrollable constraint. RMPL allows us to model uncontrollable constraints as in Listing 4.

```
(define-control-program drive-to-lecture ()
  (declare (primitive)
           (duration (simple :lower-bound 15 :upper-bound 20)
                     :contingent t)))
```

Listing 4: An uncontrollable, or contingent, temporal constraint in a control program.

The addition of `:contingent t` to the `(duration ...)` form tells Kirk that it does not have control over when the end of `drive-to-lecture` is scheduled, rather, Nature (i.e. traffic conditions) chooses a time. Despite the lack of control over `drive-to-lecture`, we do know the drive should take between 15 and 20 minutes, hence our model includes `:lower-bound 15` and `:upper-bound 20`.

With uncontrollable constraints in a control program, we are no longer guaranteed to be able to produce a schedule offline as we show in Table 3.1. Instead, as time passes, we may only choose to schedule controllable events based on the *partial history* of contingent event assignments so far, or, in other words, perform *dynamic scheduling*. Thus, we can no longer simulate a schedule with Kirk. We must connect Kirk to a source for receiving contingent event assignments in order to make valid controllable event assignments. Our approach to dynamic scheduling is the focus of Section 3.2.

As a contribution of this thesis, our existing approach to specifying durations in RMPL was expanded to model observation delay. An example follows in Listing 5 modeling a sample collection control program with observation delay.

```
(define-control-program collect-science-sample ()
  (declare (primitive)
           (duration (simple :lower-bound 15 :upper-bound 30
                             :min-observation-delay 5
                             :max-observation-delay 15)
                     :contingent t)))
```

Listing 5: An RMPL control program describing a science data collection task with observation delay.

We can see in Listing 5 that representing set-bounded observation delay is a simple as adding `:min-` and `:max-observation-delay` to the `(duration (simple ...)  :contingent t)` form. In full, this control program represents an uncontrollable constraint with a contingent event that Nature will schedule $[15, 30]$ time units after sample collection begins. The executive will then wait an additional $[5, 15]$ time units before learning that `collect-science-sample` has been scheduled. As will be described in much greater detail in Section 4.3, the executive will only learn *that* the contingent event occurred - is not guaranteed to learn where in $[15, 30]$ the contingent event was assigned, nor will it know how much observation delay was incurred.

3.1.3 Explicitly Modeling Agents in RMPL

This section is included to expand on the features of RMPL, though note that none of these features are required for controlling distributed agents, and were not a part of the experiments for this research.

If we wanted to specify agents in a multi-agent control program, or if we wanted to take vehicle dynamics into account, RMPL gives us a means for using the Common Lisp Object System (CLOS) for defining agents, agent dynamics, and the control programs agents may execute.

An example RMPL control program with an agent is provided in Listing 6 for completeness sake from the domain of underwater robotics.

In Listing 6, `glider` refers to a low-powered autonomous underwater vehicle that prefers to traverse by following ocean currents using a buoyancy engine.[1] We see that we model a `glider` agent and its properties using standard CLOS. The `move` control program then takes a `glider` and a `location` as arguments. The `(requires ...)` form is equivalent to the preconditions of a durative action in a PDDL 2.1 [34] domain. Likewise, the `(effect ...)` form is equivalent to PDDL effects. Finally, as we saw before, the durative action also includes a temporal constraint in its `(duration ...)` form.

Kirk is able to take RMPL as input to perform classical planning, though further discussion of it falls outside the scope of this thesis.

3.2 Scheduling Temporal Events

The bulk of the technical chapters of this thesis, namely Chapters 4 and 5, describe the algorithmic insights behind the *delay scheduler*. The delay scheduler dispatches controllable events online for dynamically controllable STNUs while reasoning over observation delay in the uncontrollable events it receives. There were two key contributions that enabled the delay scheduler.

Reasoning over the controllability of STNUs with variable-observation delay had been demonstrated to be possible in prior work [35], though an explicit, online execution strategy, let alone a valid execution strategy, was never defined for variable-delay STNUs. For our first contribution, we define an execution strategy for variable-delay controllable STNUs and prove its validity.

Likewise, dynamic schedulers have been established for dispatching events from STNUs, e.g. FAST-EX [36]. For our second contribution, we defined a novel delay scheduler built on FAST-EX capable of applying the execution strategy defined in our first contribution.

We elaborate further on our approach to each contribution below.

[1] The Slocum Glider is an example: https://www.whoi.edu/what-we-do/explore/underwater-vehicles/auvs/slocum-glider/.

```
;; This code is a snippet from a file in the thesis code repo found at:
;;      kirk-v2/examples/glider/script.rmpl

(defclass glider ()
  ((id
    :initarg :id
    :finalp t
    :type integer
    :reader id
    :documentation
    "The ID of this glider.")
   (deployed-p
    :initform nil
    :type boolean
    :accessor deployed-p
    :documentaiton
    "A boolean stating if the glider is deployed at any point in time.")
   (destination
    :initform nil
    :type (member nil "start" "end" "science-1" "science-2")
    :accessor destination
    :documentation
    "The location to which the glider is currently heading, or NIL if it is not
    in transit.")
   (location
    :initarg :location
    :initform "start"
    :type (member nil "start" "end" "science-1" "science-2")
    :accessor location
    :documentation
    "The location where the glider is currently located, or NIL if it is not at
    a location (in transit).")))

(define-control-program move (glider to)
  (declare (primitive)
           (requires (and
                      (over :all (= (destination glider) to))))
           (effect (and
                    (at :start (= (destination glider) to))
                    (at :start (= (location glider) nil))
                    (at :end (= (destination glider) nil))
                    (at :end (= (location glider) to))))
           (duration (simple :lower-bound 10 :upper-bound 20))))
```

Listing 6: A snippet of an RMPL script that defines an agent and classical planning predicates and effects of a control program.

3.2.1 Defining a Valid Execution Strategy for STNUs with Variable Observation Delay

We cannot execute an STNU without first demonstrating that it is controllable. Our approach to checking the controllability of STNUs with observation delay is to apply Bhargava's Variable-Delay Controllability checker (VDC) [1]. VDC is a procedure that takes place in two stages and is $O(N^5)$ in the number of events. In the first stage, we transform the STNU with variable observation delay to one with fixed observation delay in $O(N^2)$. In the second stage, we check the controllability of the fixed-delay STNU using Bhargava's fixed-delay controllability checker (FDC) [31], [35], which is modified from Morris' $O(N^3)$ dynamic controllability check [28] such that it accounts for fixed observation delay in contingent links.

In short, the first stage process is built around the idea of modeling a worst-case scenario with respect to receiving observations. The resulting fixed-delay STNU reflects a situation where the executive learns as little as possible about the contingent events. If the fixed-delay STNU with minimal information is controllable, then so too must any situation be controllable when we learn more information.

We contribute the definition for an execution strategy for variable-delay STNUs, wherein we dispatch events according to the *dispatchable form* of the *fixed-delay* STNU, while respecting the constraints modeled in the *variable-delay* STNU. Existing controllability checks, like FDC, and execution strategies, like FAST-EX, depend on a dispatchable form, i.e. a *distance graph* representation of the STNU. The key challenge in defining an execution strategy for a variable-delay STNU is that unlike vanilla STNUs and fixed-delay STNUs, there is no dispatchable form for variable-delay STNUs. Hence why the VDC check first transforms the variable-delay STNU to a fixed-delay form. In Chapter 5, we formally define the execution strategy for variable-delay STNUs and prove its validity.

3.2.2 Online Dispatching for STNUs with Variable Observation Delay

We chose to build the delay scheduler as a modified variant of Hunsberger's FAST-EX [36] because, to the best of our knowledge, FAST-EX is the fastest dynamic scheduler published to date.

FAST-EX maps partial histories, or schedules of events up to the current time, to Real-Time Execution Decisions (RTEDs). RTEDs contain a list of events to be executed and a time (that could be from now to point in the future) to execute them. When contingent events are observed or controllable events are scheduled, it updates the distance graph to capture the information gained. To improve the online performance of dynamic scheduling, Hunsberger's insight was to reduce the space of the dispatchable form by removing edges as events are executed. It can do so by first iteratively updating the distances to and from the remaining events by performing Dijkstra's Single

Sink and Single Source Shortest Paths algorithms to and from the zero point (start event) of the distance graph.

The delay scheduler differs from FAST-EX in the way it (1) records partial histories and (2) how it generates RTEDs. For both changes, we must address special cases related to a change in the *execution space* - the time ranges of possible event assignments - that result from the variable-delay to fixed-delay STNU transformation. We make two changes for (1). First, we remove the assumption that contingent events are instantaneously observed. Essentially, we use the known fixed observation delay to decide where in the past an observed contingent event was assigned. Second, to account for one special case due to the transformation, we use observations to optimistically rewrite the variable-delay STNU in an attempt to shorten the overall makespan (see Section 1). Key to (2) is that we are allowed to *imagine* that contingent events were assigned despite never observing them. Imagining contingent events is a result of the other special case from the variable-delay to fixed-delay transformation (see Section 5.3).

3.3 Coordination

To the best of our knowledge, this thesis contributes the first framework for, and demonstration of, online coordination between dynamic schedulers with inter-agent temporal constraints.

Our challenge is to allow multiple Kirk instances to dynamically schedule simultaneously while sharing events. At a high level, our approach is that inter-agent communications take the form of event observations. Each agent's ego controllable events are sent to peers, who receive them as exogenous, uncontrollable event observations. We allow (and expect) that communications have uncertain delay, thus we apply the modeling formalisms of variable-delay STNUs to inter-agent temporal constraints.

Our approach to online coordination is as follows:

1. Each instance of Kirk receives a unique, manually written control program
2. All control programs begin execution at the same time
3. Kirk executives broadcast scheduled events to a known set of peers
4. In their own schedules, Kirk executives record event observations from their peers as they are received

The challenge of manually writing control programs that enable MA execution is non-trivial. A modeler must consider both intra-agent and inter-agent constraints that, compounded by uncertain communication, frequently contain difficult to spot conflicts. (It is no surprise that temporal decoupling is incomplete!) Furthermore, we found that translating events between executives is challenging. When writing MA control programs, it is possible that the same event has different identifiers in different STNUs. Care must be taken to ensure different executives understand the event observations they receive from their peers. In our experiments, our strategy was to carefully write MA control programs to guarantee events shared names between executives. MA control programs under uncertain communication will be discussed in detail in Section 6.2.

The second point ensures that control programs share a temporal frame of reference. However, uncertain communication was able to partially mitigate executives with clocks that did not agree. In effect, communication delay can be used to mitigate the differences in executive clock times.

The third and fourth points encapsulate our contribution to the challenge of MA communication with respect to inter-agent temporal constraints. We imagined inter-agent communications as a simple directional graph between executives. In this structure, all event nodes are publishers. Outgoing edges represent subscribers that receive all scheduled events, including both controllable events and uncontrollable event observations that the publishing agent itself receives. Event observations are then naturally propagated through the graph. We assume that communication delay in the modeled system incorporates the time events spend propagating through the graph. Event propagation will be formally defined in Section 6.3.

3.4 Robustness

Autonomy research tends to focus on ideal, generic executives that behave perfectly. For instance, temporal reasoning research assumes that controllable events are executed instantaneously at the exact correct time without fail. Reality cannot conform to ideal conditions. At minimum, CPU cycles will tick by before a scheduled event is dispatched, causing the hands of precise clocks to move when our algorithms expect them to remain static. To run on hardware, executives and agents must communicate, which adds additional time that is unaccounted for in scheduling algorithms. And finally, we need to explicitly decide how to translate temporal events to messages that hardware can execute. Given our need to deploy Kirk on real hardware, we contribute a seemingly disparate set of algorithms removing expectations of idealized performance, that, when taken together, enable deployment of temporal reasoning algorithms in real agents.

We include five contributions to dynamic scheduling and dispatching for enabling robust executives.

1. A well defined architecture for event execution with distinct scheduler, dispatcher, and driver responsibilities
2. Tolerance in event scheduling
3. Controllable event preemption
4. The separation of real and `noop` controllable events in execution decisions
5. A clock-synchronized approach for managing repeated tasks during online execution

TODO given the hardware experiments of this thesis...

This thesis identifies addresses three core issues...

We improve the delay scheduler by differentiating real and `noop` controllable events...

We remove the assumption that controllable events are instantaneously executed...

We identify drawbacks in naïve approaches to building executives using parallel and concurrent processes. We propose a clock-synchronized architecture that addresses challenges in simulating executives and better matches our expectations of order of operations behavior as programmers.

Chapter 4

Consistency Checking of Temporal Constraint Networks with Uncertain Observation Delay

Our overarching aim in this Chapter is to architect the offline portions of a single-agent pipeline that takes a temporal network with uncertain observation delay of exogenous events as input in order to execute events in the real world within time windows that guarantee temporal consistency. Given everything we know and do not know about the temporal relationship between events within and outside of our control, this chapter will lay the groundwork that there exists an execution strategy that guarantees all constraints are satisfied despite the uncertainty. The next chapter, Chapter 5, will provide accompanying online procedures for acting on said execution strategy and dispatching events with real hardware (or by generally telling an agent what to do).

The three aims of this chapter are to

1. model temporal constraints with uncertain observation delay,
2. define a consistency (controllability) checking procedure for temporal networks with uncertain observation delay, and
3. prove that the execution strategy assumed by the controllability checking procedure is safe.

In Section 4.1, we address aim (1) by outlining the necessary definitions for modeling temporal networks. Sections 4.2 and 4.3 describe our chosen model for temporal constraints with uncertain observation delay, and address aim (2) by presenting a procedure for checking the consistency thereof. Sections 4.2 and 4.3 are largely based on the work originally put forth by Bhargava et. al., [1], [31] with additional contributions by us as highlighted below. Notably, Section 4.3 addresses aim (3) by contributing novel proofs that the execution strategy assumed to exist by Bhargava et. al. is safe.

4.1 Temporal Networks

Temporal networks form the backbone of our architecture for temporal reasoning under observation delay. Simple Temporal Networks (STNs) offer the basic building blocks for most expressive temporal network formalisms [21]. An STN is composed of a set of variables and a set of binary constraints, each of which limits the difference between a pair of these variables; for example, $B - A \in [10, 20]$. Each variable denotes a distinguished point in time, called an *event*. Constraints over events are binary *temporal constraints* that limit their temporal difference; for example, the aforementioned constraint specifies that event A must happen between 10 and 20 minutes before event B.

Definition 1. STN [21]

An *STN* is a pair $\langle X, R\rangle$, where:

- X is a set of variables, called events, each with a domain of the reals $\mathbb{R}$, and
- R is a set of simple temporal constraints. Each constraint $\langle x_r, y_r, l_r, u_r\rangle$ has scope $\{x_r, y_r\} \subseteq X$ and relation $x_r - y_r \in [l_r, u_r]$.

Definition 2. Schedule [25]

A *schedule*, ξ, is a mapping of events to times, $\xi : X \rightarrow \mathbb{R}$.

An STN is used to frame scheduling problems. A schedule is feasible if it satisfies each constraint in R. We use the notation $\xi(x)$ to represent a mapping from an event, x, to a time, $x \rightarrow \mathbb{R}$, in the schedule. A schedule is *complete* if all $x \in X$ are assigned times in ξ. An STN is *consistent* if it has at least one feasible schedule that assigns all events in X.

An STN is consistent if and only if there is no negative cycle in its equivalent distance graph [21]. Let n be the number of events in a temporal network and m to be the number of constraints. Then consistency of an STN can be checked in $O(mn)$ time using the Bellman-Ford Algorithm to check for negative cycles.

While an STN is useful for modeling problems in which an agent can control the exact time of all events, it does not let us model actions whose durations are uncertain. A Simple Temporal Network with Uncertainty (STNU) is an extension to an STN that allows us to model these types of uncertain actions [24].

Definition 3. STNU [24]

An *STNU* S is a quadruple $\langle X_e, X_c, R_r, R_c\rangle$, where:

- X_e is the set of executable events with domain $\mathbb{R}$,
- X_c is the set of contingent events with domain $\mathbb{R}$,
- R_r is the set of requirement constraints of the form $l_r \leq x_r - y_r \leq u_r$, where $x_r, y_r \in X_c \cup X_e$ and $l_r, u_r \in \mathbb{R}$, and

- R_c is the set of contingent constraints of the form $0 \leq l_r \leq c_r - e_r \leq u_r$, where $c_r \in X_c$, $e_r \in X_e$ and $l_r, u_r \in \mathbb{R}$.

An STNU divides its events into executable and contingent events and divides its constraints into requirement and contingent constraints. The times of executable events are under the control of an agent, and assigned by its scheduler. STNU executable events are equivalent to events in an STN. Contingent events are controlled by nature. Contingent constraints model the temporal consequences of uncertain actions and are enforced by nature. They relate a starting executable event and an ending contingent event. To ensure causality, the lower-bound of a contingent constraint is required to be non-negative; hence, the end event of the constraint follows its start event. Contingent constraints are not allowed to be immediately followed by additional contingent constraints. Requirement constraints specify constraints that the scheduler needs to satisfy and may relate any pair of events. An STNU requirement constraint is equivalent to an STN constraint.

To clarify terminology, we sometimes refer to contingent constraints as contingent links and requirement constraints as requirement links. When we discuss contingent constraint or contingent link duration, we refer to the amount of time that actually elapses between a contingent link's starting executable event and its ending contingent event. We sometimes refer to STNUs as defined in Defintion 3 as *vanilla* STNUs (in contrast to the many "flavors" of STNUs, namely the variants with fixed and variable observation delay functions as will be defined below).

With STNs, our goal is to construct a consistent schedule for all events such that all constraints are satisfied. In STNUs, however, contingent events cannot be scheduled directly. Instead, we are interested in determining whether there is a *controllable* policy that guarantees that a schedule can be constructed such that all constraints are satisfied despite how uncertainty is resolved.

Definition 4. Situations [24]

For an STNU S with k contingent constraints $\langle e_1, c_1, l_1, u_1 \rangle, \cdots, \langle e_k, c_k, l_k, u_k \rangle$, each *situation*, ω, represents a possible set of values for all links in S, $\omega = (\omega_1, \cdots, \omega_k) \in \Omega$. The *space of situations* for S, Ω, is $\Omega = [e_1, c_1] \times \cdots \times [e_k, c_k]$.

Each *situation* in the *space of situations*, $\omega \in \Omega$, represents a different assignment of contingent links in the schedule [24]. We may represent the situation for a specific constraint as ω_i for the i-th constraint in S, or $\omega(E)$ for contingent event E. Situations are sets of intervals. To examine spaces of situations, we can make the following comparisons.

Definition 5. Comparisons of Spaces of Situations

Given two spaces of situations, Ω_1 and Ω_2, with contingent link j, $1 \leq j \leq k$,

- $\Omega_1 = \Omega_2$ if and only if $\omega_1 = \omega_2 \forall \omega_1 \in \Omega_1 \forall \omega_2 \in \Omega_2$
- $\Omega_1 \subset \Omega_2$ if situation j in Ω_1 is a subset of situation j in Ω_2, $\omega_{1j} \subset \omega_{2j}$, and all other situations are equivalent

- $\Omega_1 \subset \Omega_2$ if Ω_1 omits contingent link j, e.g. $\Omega_1 = \prod_{\substack{i=1 \\ i \neq j}}^{k} [e_{1i}, c_{1i}]$, and all other situations are equal.

Situations may be applied to STNUs.

Definition 6. Projection [24], [25]

A *Projection* is an application of a situation, ω, on an STNU S, which collapses the durations of contingent links to specific durations resulting in an STN.

A *projection* is an STN that is the result of applying a situation to an STNU, and thus the contingent links have reduced from uncertain ranges to specific durations [24], [25].

Definition 7. Execution Strategy

An *execution strategy*, $\mathcal{S}$, is a mapping of situations to schedules, $\mathcal{S} : \Omega \to \Xi$.

An *execution strategy* then naturally maps a specific resolution of the uncertainty of the contingent constraints to a set of assignments for the events of an STNU. For an STNU, time monotonically increases and we only observe *activated* contingent events, or those contingent events at the tail of a contingent link whose free event predecessor has been executed. As such, we modify our definition of ξ.

Definition 8. Partial Schedule

A *partial schedule*, ξ, is a mapping from a proper subset of events in X, X', to times, $\xi : X' \to \mathbb{R}$.

As a proper subset, ξ represents an assignment of events *so far* during the execution of an STNU. From here on, ξ refers to a partial schedule.

In the world of STNU literature, there are many forms of controllability that represent the ability of a scheduler to construct execution strategies that satisfy constraints under different conditions [24]. Three forms of controllability, *strong*, *weak*, and *dynamic* are studied most often, though in practice we omit weak controllability from our analysis. As we will see below, variable-delay controllability will unify strong and dynamic controllability into a single theory. A temporal network is *strongly controllable* (or exhibits strong controllability), if there exists a complete schedule that will satisfy all constraints for all projections of the STNU. A temporal network exhibits dynamic controllability if an execution strategy, $\mathcal{S}$, exists for a given partial schedule, ξ.

4.2 Fixed-Delay Controllability

Under fixed-delay controllability [35], we consider the problem of scheduling execution decisions when the assignment of values to contingent events is learned (if at all) after some time has passed from the initial assignment. We prefer this model because it is flexible enough to model most forms of event observation we would expect to see in a real-time execution context. Fixed-delay controllability uses a *fixed-delay function* to encode the delay between when an event occurs and when it is observed by a scheduling agent. We sometimes refer to an STNU with an associated fixed-delay function as a *fixed-delay STNU*.

Definition 9. Fixed-Delay Function [35]

A *fixed-delay function*, $\gamma : X_c \to \mathbb{R}^+ \cup \{\infty\}$, maps a contingent event to the amount of time that passes between when the event occurs and when its value is observed.

As a matter of convention, we use $A \xrightarrow{[l,u]} B$ to represent requirement links between events A and B and use $A \xRightarrow{[l,u]} E$ to represent contingent links between A and E. When we refer to the fixed-delay function associated with a contingent event E of some contingent constraint $A \xRightarrow{[l,u]} E$, we use the notation $\gamma(E)$, or equivalently, γ_E. Without instantaneous observation of contingent events, we must clarify the relationship between when an event occurs and when it is *observed*.

Definition 10. Contingent Event Observation

Observations, `obs`, are a mapping from contingent events to times when the agent receives events, $\texttt{obs} : E \to \mathbb{R}$, based on the relationship, $\texttt{obs}(E) = \xi(E) + \gamma(E)$.

Note that $\xi(E)$ is now indirectly learned through the relationship,

$$\xi(E) = \texttt{obs}(E) - \gamma(E)$$

We also present a revised definition of situations, Ω_f, to reflect the impact of the delay function on event observations.

Definition 11. Fixed-Delay Situations

For an STNU S with k contingent constraints $\langle e_1, c_1, l_1, u_1\rangle, \cdots, \langle e_k, c_k, l_k, u_k\rangle$ and fixed-delay function γ, each *fixed-delay situation*, ω_f, represents a possible set of *observed* values for all links in S, $\omega_f = (\omega_{f1}, \cdots, \omega_{fk})$. The *{*space of situations*}* for S, Ω_f, is $\Omega_f = [e_1, c_1] + [\gamma_1, \gamma_1] \times \cdots \times [e_k, c_k] + [\gamma_k, \gamma_k]$.

To emphasize that the *observed* value for an event is not the same as $\xi(E)$, we also use the term *observation space* as a synonym for the space of situations.

With the semantics of delayed observations in hand, we can define what it means for a fixed-delay STNU to be controllable.

Definition 12. Fixed-Delay Controllability [35]

An STNU S is *fixed-delay controllable* with respect to a delay function, γ, if and only if for the space of situations, Ω_f, there exists an $\mathcal{S}$ that will construct a satisfying schedule for requirement constraints during execution, ξ.

Importantly, fixed-delay controllability (FDC) generalizes the two concepts of controllability that are central to STNUs, strong and dynamic controllability. In particular, by using a fixed-delay function where we observe all events instantaneously, checking fixed-delay controllability reduces to checking *dynamic controllability*. Similarly, a fixed-delay function that specifies we never observe any contingent event corresponds to checking *strong controllability* [24].

To determine whether an STNU is fixed-delay controllable, we determine whether there exists a *valid* execution strategy for it.

Definition 13. Valid Execution Strategy

A *valid* $\mathcal{S}$ is one that enforces that, for any $\omega_f \in \Omega_f$, while receiving information about the durations after a fixed delay, the outputted decision respects all existing temporal constraints and ensures the existence of a subsequent valid execution strategy following that action.

As is the case for a vanilla STNU, evaluating whether a valid $\mathcal{S}$ exists for a fixed-delay STNU reduces to checking for the presence of a *semi-reducible negative cycle* in a *labeled distance graph* derived from the fixed-delay STNU [27]. The key insight for checking fixed-delay controllability is the inclusion of γ in the constraint generation rules for building the labeled distance graph [**?**].

The labeled distance graph corresponds to the constraints of the STNU with each unlabeled edge from A to B with weight w (denoted $A \xrightarrow{w} B$) representing the inequality $x_B - x_A \leq w$. Labeled edges represent conditional constraints that apply depending on the realized value of contingent links in the graph. For example, a lower-case labeled edge from A to B with weight w and lower-case label c (denoted $A \xrightarrow{c:w} B$) indicates that $x_B - x_A \leq w$ whenever the contingent link ending at C takes on its lowest possible value. An upper-case labeled edge from A to B with weight w and upper-case label C (denoted $A \xrightarrow{C:w} B$) indicates that $x_B - x_A \leq w$ whenever the contingent link ending at C takes on its highest possible value. Given a labeled distance graph, there are several valid derivations we can apply to generate additional edges (see Table 4.1). If it is possible to derive a negative cycle that is free of lower-case edges, then the STNU has a *semi-reducible negative cycle* and the STNU is not controllable.

Note that with fixed-delay controllability, the lower-case and cross-case rules are modified from the Morris and Muscettola [26], accounting for γ. More specifically, we address the case where observation delay makes it impossible to receive information about a contingent event before its immediate successor. More detail can be found in [37].

4.3 Variable-Delay Controllability

While fixed-delay controllability is quite expressive, its fundamental limitation is that it assumes that contingent event assignments, even those made after a fixed delay, are always known. If uncertainty

Edge Generation Rules			
	Input edges	Conditions	Output edge
No-Case Rule	$A \xrightarrow{u} B$, $B \xrightarrow{v} C$	N/A	$A \xrightarrow{u+v} C$
Upper-Case Rule	$A \xrightarrow{u} D$, $D \xrightarrow{C:v} B$	N/A	$A \xrightarrow{C:u+v} B$
Lower-Case Rule	$A \xrightarrow{c:x} C$, $C \xrightarrow{w} D$	$w < \gamma(C)$, $C \neq D$	$A \xrightarrow{x+w} D$
Cross-Case Rule	$A \xrightarrow{c:x} C$, $C \xrightarrow{B:w} D$	$w < \gamma(C)$, $B \neq C \neq D$	$A \xrightarrow{B:x+w} D$
Label Removal Rule	$B \xrightarrow{C:u} A$, $A \xRightarrow{[x,y]} C$	$u > -x$	$B \xrightarrow{u} A$

Table 4.1: Edge generation rules for a labeled distance graph from [cite:@Bhargava2018b].

in observation delay, and thus uncertainty in contingent event assignment, is added to the model, then we are forced to decide when to act despite imperfect knowledge of the partial history.

We now introduce this model in terms of definitions for a *variable-delay function* and *variable-delay controllability* (VDC) checking as applied to *variable-delay STNUs*. Since variable-delay semantics generalizes the notion of fixed-delay, as a matter of convenience, we also use the simplified term *delay STNUs* to refer to STNUs with variable observation delay. VDC was originally presented by Nikhil Bhargava [1]. However, we contributed significant improvements of the lemmas and proofs herein, including the addition of novel visual depictions of VDC, in our role as a coauthor with Bhargava on a journal article on the topic of VDC that was submitted to the Journal of AI Research.

Definition 14. Variable-Delay Function

A *variable-delay function*, $\bar{\gamma} : x_c \rightarrow (\mathbb{R}^+ \cup \{\infty\}) \times (\mathbb{R}^+ \cup \{\infty\})$, maps a contingent event, x_c, to an interval $[a, b]$, where $a \leq b$. The interval bounds the time that passes after $\xi(x_c)$ before that value is observed to be assigned. No prior knowledge is assumed about the distribution associated with this interval.

Importantly, this model does not assume that an executing agent infers *when* a contingent event was executed, but instead infers *that* the event was executed. Like contingent constraints, the resolved value of $\bar{\gamma}(x_c)$ will be selected by Nature during execution. Thus, the timing of when an agent receives an observation is a function of the independent resolutions of the contingent link and $\bar{\gamma}(x_c)$.

By convention, we use $\bar{\gamma}^-(x_c)$ and $\bar{\gamma}^+(x_c)$ to represent the lower-bound and upper-bound, respectively, of the range representing the possible delay in observation, i.e. $\bar{\gamma}(x_c) \in [\bar{\gamma}^-(x_c), \bar{\gamma}^+(x_c)]$.

Definition 15. Observation Projection

The *observation projection* Γ is a mapping from a contingent event to a fixed observation delay, $\Gamma \ : \ X_c \rightarrow \mathbb{R} \in [\bar{\gamma}^-(X_c), \bar{\gamma}^+(X_c)]$.

During execution, the *observation projection*, Γ, represents the resolution of observation delay. Much like how a projection collapses a vanilla STNU to an STN, the observed projection collapses a contingent link with variable-observation delay to one with fixed-observation delay. However, unlike the projection of an STNU, the observation projection is not guaranteed to be learned. We update our definitions of `obs`, ξ, and Ω accordingly.

Definition 16. Contingent Event Observation

Contingent event observations, `obs`, are a mapping from contingent events to times when the agent receives events, $\texttt{obs} : E \rightarrow \mathbb{R}$, based on the relationship, $\texttt{obs}(E) = \xi(E) + \Gamma(E)$.

Determining the precise schedule of a contingent event, ξ, is complicated by an interval bounded Γ. We must use interval-bounded contingent event assignments instead.

Definition 17. Schedule

A *schedule*, ξ, when applied to contingent events, is a mapping of events to interval-bounded times, $\xi : X_c \rightarrow (\mathbb{R}^+ \cup \{\infty\}) \times (\mathbb{R}^{++} \cup \{\infty\})$, where, for any contingent event x_c, $\xi(x_c) \in [e_{x_c} + \bar{\gamma}^-(x_c), c_{x_c} + \bar{\gamma}^+(x_c)]$.

We sometimes use interval bounded schedules for requirement events as well, where $\xi(x_r) = t = [t, t]$ for some requirement event x_r assigned to time t.

We once again revise our definition of situations, Ω_v, to reflect the impact of the variable-delay function on the space of observations.

Definition 18. Variable-Delay Situations

For an STNU S with k contingent constraints $\langle e_1, c_1, l_1, u_1\rangle, \cdots, \langle e_k, c_k, l_k, u_k\rangle$ and variable-delay function $\bar{\gamma}$, each *variable-delay situation*, ω_v, represents a possible set of *observed* values for all links in S, $\omega = (\omega_{v1}, \cdots, \omega_{vk})$. The {space of situations} for S, Ω_v, is $\Omega_v = [e_1, c_1] + [\bar{\gamma}_1^-, \bar{\gamma}_1^+] \times \cdots \times [e_k, c_k] + [\bar{\gamma}_k^-, \bar{\gamma}_k^+]$.

We see that the space of observations has likewise grown in the transition to variable observation delay. If $\bar{\gamma}^- < \bar{\gamma}^+$, Ω_v for variable observation delay is strictly larger than Ω_f for fixed-observation delay and Ω for vanilla STNUs.

Like the fixed-delay function for fixed-delay controllability, the variable-delay function relates an observation delay to a contingent event, independent of other events. We take a similar approach to defining variable-delay controllability, relative to fixed-delay controllability.

Definition 19. Variable-Delay Controllability

An STNU S is *variable-delay controllable* with respect to a variable-delay function, $\bar{\gamma}$, if and only if for the space of situations, Ω_v, there is an $\mathcal{S}$ that produces a satisfying schedule for requirement events during execution, ξ.

Determining whether a given variable-delay STNU, S, is variable-delay controllable has two components [1]. The first is to derive a fixed-delay STNU, S', with fixed-observation delay, γ, that is equivalent with respect to controllability. The second is to show that S' is fixed-delay controllable. Below, we reiterate the claims of [1], demonstrating how to derive S' from S that is equivalent with respect to controllability. We first demonstrate how to transform the contingent links from S to S', and demonstrate their correctness with respect to observation spaces, before following up with transformations to the requirement links to maintain the same scheduling semantics in S'.

For the following lemmas, let x_c be a contingent link in S, where $x_c \in [l, u]$ and variable-delay function $\bar{\gamma}(x_c)$. Let x'_c be the transformed contingent link in S' with fixed-delay function, $\gamma(x'_c)$.

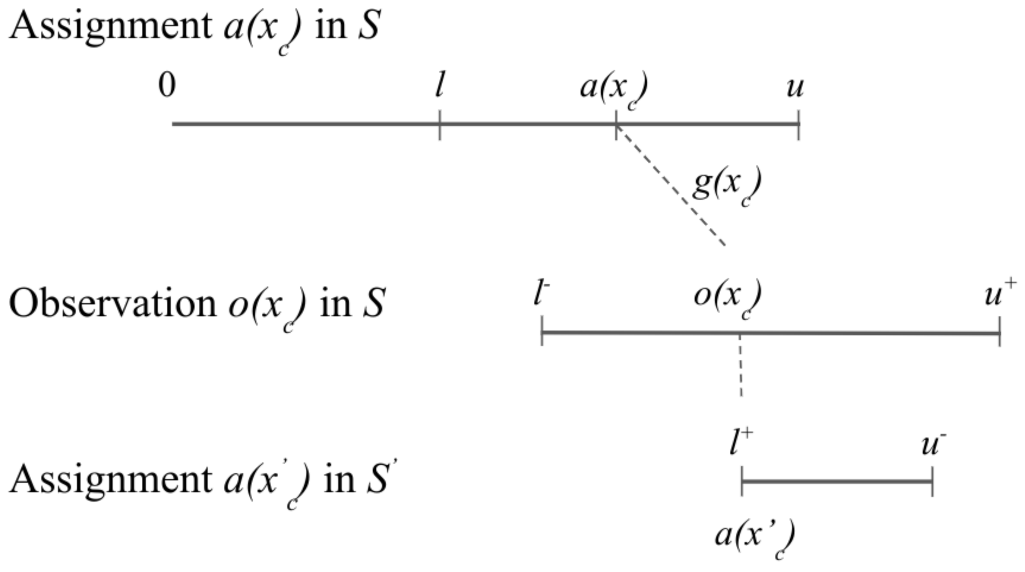

Assignment $a(x_c)$ in S
0
l
$a(x_c)$
u
$g(x_c)$
Observation $o(x_c)$ in S
l^-
$o(x_c)$
u^+
Assignment $a(x'_c)$ in S'
l^+
u^-
$a(x'_c)$

Figure 4-1: We visualize the relationship between realized assignments across S and S'. In this example, each horizontal line is a timeline monotonically increasing from left to right. Dashed lines represent observation delays. We see how an assignment in S, ξ(x$_c$)\$, realized observation delay, $g(x_c)$, and an observation in S, $\texttt{obs}(x_c)$, lead to an assignment in S', ξ(x'$_c$)\$.

Note that we receive $\texttt{obs}(x_c)$ from Nature, but make the assignment $\xi(x'_c)$ in the dispatchable form of S'. To be clear, while $\xi(x_c)$ is an interval, $(\mathbb{R} \cup \infty) \times (\mathbb{R} \cup \infty)$, $\xi(x'_c)$ is in $\mathbb{R}$. For a fixed interval, e.g. $\texttt{obs}(x_c) \in [t, t]$, we sometimes employ an equivalent representation, $\xi(x_c) = t$.

Additionally, we sometimes apply $-$ and $+$ superscripts to l and u to denote the earliest and latest times respectively that an assignment at those bounds could be observed. For instance, the relationship in Definition 16 simplifies to,

$$\texttt{obs}(x_c) = [l + \bar{\gamma}^-(x_c), u + \bar{\gamma}^+(x_c)] \tag{4.1}$$

$$\texttt{obs}(x_c) = [l^-(x_c), u^+(x_c)] \tag{4.2}$$

Lastly, we need a means to compare observation spaces if we are to transform variable-delay to fixed-delay STNUs.

Definition 20. Observation Space Mapping

Let μ be a mapping from an assignment to a situation, $\mu : \xi \to \omega$. To say that $\mu(x'_c) \subseteq \omega_v(x_c)$ means that, for any assignment of x'_c in S', there is an equivalent situation in S for x_c.

For the transitions below, it is a *valid observation space mapping*, if we can show that $\mu(x'_c) \subseteq \omega_v(x_c)$. If so, it is guaranteed that any assignment in the observation space of x'_c also has a valid assignment in the observation space of x_c.

We now have the necessary vocabulary and notation to step through the transformations from S to S'. These lemmas were first presented in [1].

Definition 21. Variable-Delay to Fixed-Delay Transformations

The *variable-delay to fixed-delay transformations* define a set of observation space mappings, where there are valid observation space mappings for all the contingent constraints in S' to S.

Thus, if there is a satisfying $\mathcal{S}$ for the fixed-delay observation space of S', it is guaranteed to simultaneously satisfy any situation in the variable-delay observation space, Ω_v, of S.

Lemma 1. *For any contingent event $x_c \in X_c$ in S, if $\bar{\gamma}^-(x_c) = \bar{\gamma}^+(x_c)$, we emulate $\bar{\gamma}(x_c)$ in S' using $\gamma(x'_c) = \bar{\gamma}^+(x_c)$.*

Proof. We translate an already fixed-bounded observation delay in the form of $\bar{\gamma}(x_c)$ to the equivalent fixed-delay function, $\gamma(x'_c)$, thus $\omega_f(x'_c) = \omega_v(x_c)$. □

Lemma 2. *For any contingent event $x_c \in X_c$, $\bar{\gamma}^+(x_c) = \infty$, we emulate $\bar{\gamma}(x_c)$ in S' as $\gamma(x'_c) = \infty$.*

Proof. There are projections where we would not receive information about x_c, therefore we have to act as if we *never* receive an observation of x_c. Any $\mathcal{S}$ that works when we do not receive information about x_c would also work when do receive an observation if we choose to ignore the observation.

None of our decisions depend on $\xi(x'_c)$, thus no observation space mapping to S is necessary. □

Lemma 3. *If $u - l \leq \bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)$, we emulate $\bar{\gamma}(x_c)$ in S' using $\gamma(x'_c) = \infty$.*

Proof. We can ignore observations of x_c because they are not guaranteed to narrow where $\xi(x_c)$ was assigned in the range $[l, u]$.

Let α be the range of $\texttt{obs}(x_c)$ when $\xi(x_c) \in [l, l]$. Let β be the range of $\texttt{obs}(x_c)$ when $\xi(x_c) \in [u, u]$. By Equation 4.1,

$$\alpha = [l^-(x_c), l^+(x_c)]$$
$$\beta = [u^-(x_c), u^+(x_c)]$$

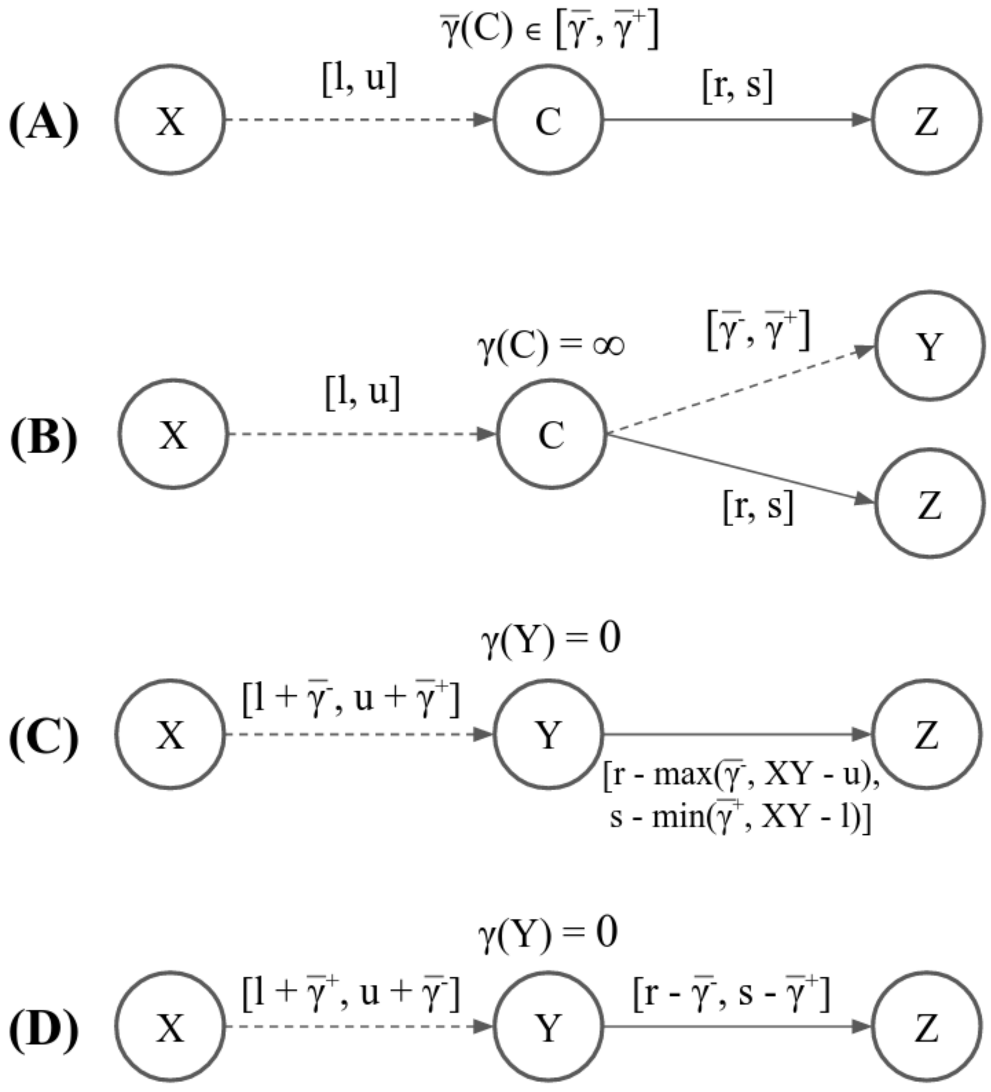

$\overline{\gamma}(C) \in [\overline{\gamma}^-, \overline{\gamma}^+]$
[l, u]
[r, s]
(A)
X
C
Z
$\gamma(C) = \infty$
$[\overline{\gamma}^-, \overline{\gamma}^+]$
[l, u]
[r, s]
(B)
X
C
Y
Z
$\gamma(Y) = 0$
$[l + \overline{\gamma}^-, u + \overline{\gamma}^+]$
$[r - \max(\overline{\gamma}^-, XY - u),$
$s - \min(\overline{\gamma}^+, XY - l)]$
(C)
X
Y
Z
$\gamma(Y) = 0$
$[l + \overline{\gamma}^+, u + \overline{\gamma}^-]$
$[r - \overline{\gamma}^-, s - \overline{\gamma}^+]$
(D)
X
Y
Z

Figure 4-2: A visualization of the lemmas used to transform contingent links with variable observation delay and subsequent requirement links.

We can show that $u^-(x_c) \leq l^+(x_c)$.

$$\begin{aligned} u - l &\leq \bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c) \\ u + \bar{\gamma}^-(x_c) &\leq l + \bar{\gamma}^+(x_c) \\ u^-(x_c) &\leq l^+(x_c) \end{aligned}$$

The lower bound of β is less than the upper bound of α, thus $\alpha \cap \beta$. An observation $\texttt{obs}(x_c) \in [u^-(x_c), l^+(x_c)]$ could be the result of $\xi(x_c) = [l, l]$, $\xi(x_c) = [u, u]$, or any value $\xi(x_c) \in [l, u]$. Observations provide no information about the underlying contingent constraint, therefore we ignore $\texttt{obs}(x_c)$.

None of our decisions depend on $\xi(x'_c)$, thus no observation space mapping to S is necessary. □

Lemma 4. *If $u - l > \bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)$, we can emulate $\bar{\gamma}(x_c)$ under minimal information by replacing the bounds of x_c with $x'_c \in [l^+(x_c), u^-(x_c)]$ and letting $\gamma(x'_c) = 0$.*

Proof. Under Lemma 4, observations $\texttt{obs}(x_c)$ are guaranteed to narrow the range of $\xi(x_c)$.

We have the same ranges for α and β as in Lemma 3, however we can show that $u^-(x_c) \geq l^+(x_c)$ instead.

$$\begin{aligned} u - l &\geq \bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c) \\ u + \bar{\gamma}^-(x_c) &\geq l + \bar{\gamma}^+(x_c) \\ u^-(x_c) &\geq l^+(x_c) \end{aligned}$$

Thus, receiving an observation is guaranteed to narrow the derived range of $\xi(x_c)$. The transformation tightens the range of x'_c to one where there is maximum ambiguity of the assignment of x_c while guaranteeing an execution strategy for any assignment of $x_c \in [l, u]$.

Based on the derivations above, it is clear that $\mu(x'_c)$ maps to the observation space where there is ambiguity as to the projection of $\xi(x_c) \in [l, u]$. We must also show that $\mu(x'_c)$ has mappings to the extrema of $\xi(x_c)$. We start with the earliest $\xi(x'_c)$.

$$\xi(x'_c) = l^+(x_c) = l + \bar{\gamma}^+(x_c)$$

We show that that this assignment of $\xi(x'_c)$ can be modeled as the following observation in S.

$$\texttt{obs}(x_c) \in [l + \bar{\gamma}^-(x_c), l + \bar{\gamma}^+(x_c)]$$
$$\texttt{obs}(x_c) \in [l, l] + \Gamma(x_c)$$

It is possible that $\xi(x_c) = [l, l]$. As such, all observations in $\texttt{obs}(x_c)$ may share the same execution strategy because the underlying temporal constraints depend on $\xi(x_c)$, not $\texttt{obs}(x'_c)$ or $\Gamma(x_c)$. We may expand the range of the observation space when we map to S with $\mu(x'_c)$.

$$\mu : l^+(x_c) \to \omega_v(x_c)$$
$$\omega_v(x_c) = [l + \bar{\gamma}^-(x_c), l + \bar{\gamma}^+(x_c)]$$

We see that μ has a valid observation space mapping to the minimum of the range of $\omega_v(x_c)$. We use the same argument for the maximum.

$$\xi(x'_c) = u + \bar{\gamma}^-(x_c)$$

Observations anywhere in $[u + \bar{\gamma}^-(x_c), u + \bar{\gamma}^+(x_c)]$ may share execution strategies because, it is possible that in all cases, $\xi(x_c) = [u, u]$. We may then expand the range of the observation space when we map to S.

$$\mu : u^-(x_c) \to \omega_v(x_c)$$
$$\omega_v(x_c) = [u + \bar{\gamma}^-(x_c), u + \bar{\gamma}^+(x_c)]$$

Thus, $\mu(x'_c)$ maps to the maximum of the range of $\omega_v(x_c)$. The transition creates assignments in S' that map to the entire $\omega_v(x_c)$ in S.

$\square$

This concludes the modifications required to transform a contingent event $x_c \in X_c$ in S to its equivalent $x'_c \in X_c$ in S'. What remains is to address the transformation of requirement links, $x_r \in X_r$, in S such that their transformed equivalents, $x'_r \in X_r$ in S', express the same execution semantics in S' as they did in S. We will demonstrate the correctness of the transformations after Lemma 6.

Lemma 5. *If we have contingent link $X \Rightarrow C$ with duration $[l, u]$, outgoing requirement link $C \to Z$ with duration $[u, v]$ with an unobservable C, and contingent link $C \Rightarrow Y$ with range $[\bar{\gamma}^-(x_c), \bar{\gamma}^+(x_c)]$,* *we can emulate the role of the original requirement link during execution with a new link* $Y \rightarrow Z$ *with bounds* $[u - max(\bar{\gamma}^-(x_c), XY - u), v - min(\bar{\gamma}^+(x_c), XY - l)]$*, where* XY *is the true duration of* $X \Rightarrow Y$*.*

Proof. See Figure 4-2c for reference. From an execution perspective, X and Y are the only events that can give us any information that we can use to reason about when to execute Z (since C is wholly unobservable).

If we execute Z based on what we learn from Y, then we use our information from Y to make inferences about the true durations of $X \Rightarrow C$ and $C \Rightarrow Y$ based on $X \Rightarrow Y$. We know that the lower-bound of $C \Rightarrow Y$ is at least $XY - b$ and that its upper-bound is at most $XY - a$. But we also have the a priori bounds on the contingent link that limit its range to $[\bar{\gamma}^-, \bar{\gamma}^+]$. Taken together, during execution we can infer that the true bounds of $C \Rightarrow Y$ are $[max(\bar{\gamma}^-, XY - b), min(\bar{\gamma}^+, XY - a)]$. Since we have bounds only on Z's execution in relation to C, we can then infer a requirement link $Y \rightarrow Z$ with bounds $[u - max(\bar{\gamma}^-, XY - b), v - min(\bar{\gamma}^-, XY - a)]$.

If we try to execute Z based on information we have about X, we must be robust to any possible value assigned to $X \Rightarrow C$. This means that we would be forced to draw a requirement link $X \rightarrow Z$ with bounds $[u + b, v + a]$. But we know that $u - max(\bar{\gamma}^-, XY - b) \leq u + b - XY$ and $v - min(\bar{\gamma}^-, XY - a) \geq v + a - XY$, which means that the bounds we derived from Y are at least as expressive as the bounds that we would derive from X. □

Since we have a local execution strategy that depends on the real value of XY, we can try to apply this strategy to the contingent link that we restricted in Lemma 4, in order to repair the remaining requirement links.

Lemma 6. *If we have an outgoing requirement link* $C \rightarrow Z$ *with duration* $[u, v]$*, where* C *is a contingent event, we can emulate the role of the original requirement link by replacing its bounds with* $[u - \bar{\gamma}^-(x_c), v - \bar{\gamma}^+(x_c)]$*.*

Proof. See Figure 4-2d for reference. If we directly apply the transformation from Lemma 5 and Figure 4-2c to our original STNU, we introduce complexity through the need to reason over *min* and *max* operations in our link bounds. However, from Lemma 4, we know that in a controllability evaluation context, it is acceptable for us to simplify the $X \Rightarrow Y$ link to a stricter range of $[a + \bar{\gamma}^+, b + \bar{\gamma}^-]$, instead of $[a + \bar{\gamma}^-, b + \bar{\gamma}^+]$. This means that for the purpose of evaluating controllability, we can assume $a + \bar{\gamma}^+ \leq XY \leq b + \bar{\gamma}^-$. When we evaluate the requirement link $Y \rightarrow Z$, we see $max(\bar{\gamma}^-, XY - b) = \bar{\gamma}^-$ and $min(\bar{\gamma}^+, XY - a) = \bar{\gamma}^+$. This gives us bounds of $[u - \bar{\gamma}^-, v - \bar{\gamma}^+]$ for the $Y \rightarrow Z$ requirement link as seen in Figure 4-2d. □

Lemma 6 handles outgoing requirement edges connected to contingent events. In addition, we must handle incoming edges.

Corollary 6.1. *If we have an incoming requirement link $Z \to C$ with duration $[u, v]$, where C is a contingent event, we can replace the bounds of the original requirement link with $[u + \bar{\gamma}^+(x_c), v + \bar{\gamma}^-(x_c)]$.*

Proof. A requirement link $Z \to C$ with bounds $[u, v]$ can be immediately rewritten as its reverse $C \to Z$ with bounds $[-v, -u]$. After reversing the edge, we can apply Lemma 6 to get $Y \to Z$ with bounds $[-v-\bar{\gamma}^-, -u-\bar{\gamma}^+]$, which we can reverse again to get $Z \to Y$ with bounds $[u+\bar{\gamma}^+, v+\bar{\gamma}^-]$. $\square$

We can examine a concrete example of Lemmas 4, 5, and 6 to show equivalence in the transformation from Figure 4-2a to 4-2d. We start by building an example of 4-2a. Let $X \overset{[2,5]}{\Longrightarrow} C$ with $\bar{\gamma}(C) \in [1, 2]$ and $C \xrightarrow{[11,20]} Z$. If we learn of event C at time 4, then one possibility is that the realized duration of C could have been 2 with an observation delay of 2. In this case, event Z must be executed in $[13, 22]$. However, if the realized duration of C were 3 with an observation delay of 1, then Z would fall in $[14, 23]$. Given we cannot distinguish between the possibilities, we take the intersection of the intervals, yielding $Z \in [14, 22]$. Likewise, if we learn of C at time 6, then C could have been realized at time 5 with an observation delay of 1 or it could have been realized at time 4 with an observation delay of 2. In the first case, Z must then fall in $[16, 25]$, while in the second, Z would fall in $[15, 24]$. The intersection yields $[16, 24]$.

By the semantics represented in Figure 4-2d, we can build an equivalent network with $\gamma(Y) = 0$ by setting $X \overset{[4,6]}{\Longrightarrow} Y$ and $Y \xrightarrow{[10,18]} Z$. If Y is observed at time 4, Z must be executed in $[14, 22]$. If Y is observed at time 6, Z then must be executed in $[16, 24]$. The execution semantics for both cases match the equivalent networks from 4-2a described above.

After applying Lemma 4, despite the limited expected range of assignments in x'_c in S' compared to x_c in S, we can show that Lemma 6 guarantees a satisfying schedule for any $\texttt{obs}(x_c) \in [l^-(x_c), u^+(x_c)]$ using an $\mathcal{S}$ that employs *buffering* and *imagining* contingent events.

Definition 22. Buffering

Buffering a contingent event x_c is an execution strategy where, if x_c is observed earlier than the lower bound of the observation space $\texttt{obs}(x_c) < \omega_f^-(x'_c)$, we assign $\xi(x'_c)$ to the lower bound of the observation space, $\xi(x'_c) = \omega_f^-(x'_c)$.

Definition 23. Imagining

Imagining a contingent event x_c is an execution strategy where, if x_c is observed later than the upper bound of the observation space, $\texttt{obs}(x_c) > \omega_f^+(x'_c)$, we assign $\xi(x'_c)$ to the upper bound of the observation space, $\xi(x'_c) = \omega_f^+(x'_c)$.

Lemma 7. *If S' is fixed-delay controllable after applying Lemmas 4, 5, and 6 to contingent event Y with following requirement event Z, there is a valid $\mathcal{S}$ for any observation in the observation space of S, $\omega_v(Y) = [a^-(Y), b^+(Y)]$.*

Proof. We first note the observation space of S' is a subinterval of the original observation space of S, $\omega_f(Y') \subset \omega_v(Y)$, and there are two distinct ranges of observations that are not in $\omega_f(Y')$.

$$\begin{aligned}
&\omega_f(Y') = [a + \bar{\gamma}^+(Y), b + \bar{\gamma}^-(Y)];\ \omega_v(Y) = [a + \bar{\gamma}^-(Y), b + \bar{\gamma}^+(Y)] \\
&\omega_f(Y') \not\supset [a + \bar{\gamma}^-(Y), a + \bar{\gamma}^+(Y))\ \ (\textit{"Early" observations}) \\
&\omega_f(Y') \not\supset (b + \bar{\gamma}^+(Y), b + \bar{\gamma}^+(Y)]\ \ (\textit{"Late" observations})
\end{aligned}$$

We address the early observations first. The range of early assignments of $\xi(Y)$ in S that we care about are the ones that could produce an observation $\texttt{obs}(Y) \leq a + \bar{\gamma}^+(Y)$, which is $\xi(Y) = [a, a+(\bar{\gamma}^+(Y)-\bar{\gamma}^-(Y))]$. We rewrite the range of early assignments as $\xi(Y) = a+(\bar{\gamma}^+(Y)-\bar{\gamma}^-(Y))-\epsilon$, where $0 \leq \epsilon \leq (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y))$. By the semantics of S, the range of assignments of $\xi(Z)$ is then,

$$\begin{aligned}
\xi(Z) &= [a + (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)) - \epsilon, a + (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)) - \epsilon] + [u, v] \\
\xi(Z) &= [a + u + (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)) - \epsilon, a + v + (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)) - \epsilon]
\end{aligned}$$

The earliest assignment of Y' in S' is $\xi(Y') = a + \bar{\gamma}^+(Y)$. By the semantics of S', the range of assignments of $\xi(Z')$ is then,

$$\begin{aligned}
\xi(Z') &= [a + \bar{\gamma}^+(Y), a + \bar{\gamma}^+(Y)] + [u - \bar{\gamma}^-(Y), v - \bar{\gamma}^+(Y)] \\
\xi(Z') &= [a + u + (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)), a + v]
\end{aligned}$$

We see that $\xi(Z') \subseteq \xi(Z)$ for any ϵ, meaning the execution strategy when $\xi(Y') = a + \bar{\gamma}^+(Y)$ results in a valid assignment of $\xi(Z)$ for all early observations of $\xi(Y)$. We are safe to buffer early observations to $\xi(Y') = a + \bar{\gamma}^+(Y)$.

We use the same argument for imagining late observations. The range of late assignments of $\xi(Y)$ in S that we care about are the ones that could produce an observation $\texttt{obs}(Y) \geq b + \bar{\gamma}^-(Y)$, which is $\xi(Y) = b - (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)) + \epsilon$. By the semantics of S, the range of assignments of $\xi(Z)$ is then,

$$\begin{aligned}
\xi(Z) &= [b - (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)) + \epsilon, b - (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)) + \epsilon] + [u, v] \\
\xi(Z) &= [b + u - (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)) + \epsilon, b + v - (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y)) + \epsilon]
\end{aligned}$$

The last assignment of Y' in S' is $\xi(Y') = b + \bar{\gamma}^-(Y)$. By the semantics of S', the range of assignments of $\xi(Z')$ is then,

$$\xi(Z') = [b + \bar{\gamma}^-(Y), b + \bar{\gamma}^+(Y)] + [u - \bar{\gamma}^-(Y), v - \bar{\gamma}^+(Y)]$$
$$\xi(Z') = [b + u, b + v - (\bar{\gamma}^+(Y) - \bar{\gamma}^-(Y))]$$

We see that $\xi(Z') \subseteq \xi(Z)$ for any ϵ, meaning the execution strategy when $\xi(Y') = b + \bar{\gamma}^-(Y)$ results in a valid assignment of $\xi(Z)$ for all late observations of $\xi(Y)$. In practice, there is no reason to wait until after $\texttt{obs}(Y) = b + \bar{\gamma}^-(Y)$ to receive a late observation. As soon as we see the clock has reached $b + \bar{\gamma}^-(Y)$, we are safe to imagine that $\texttt{obs}(Y)$ has been received. $\square$

4.4 Experimental Analysis

VDC experiments from VDC paper

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Dynamic Scheduling with Delayed Event Monitoring

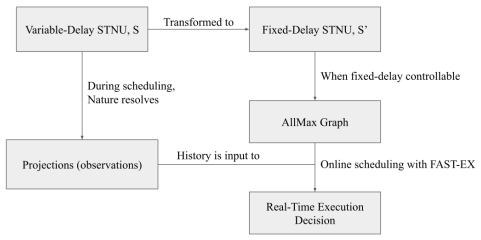

Variable-Delay STNU, S
Transformed to
Fixed-Delay STNU, S'
When fixed-delay controllable
During scheduling, Nature resolves
AllMax Graph
Projections (observations)
History is input to
Online scheduling with FAST-EX
Real-Time Execution Decision

Figure 5-1: From a variable-delay STNU to scheduling decision

Now that we have shown there exist

This section represents the start of contributions from this paper. Our aim is to describe *delay scheduling*, an RTED-based strategy for dispatching free events in the case where there is set-bounded observation delay for contingent events. As will be shown, delay scheduling is an extension to dynamic scheduling using the execution strategy as outlined in the variable-delay controllability checking procedure.

We also separate scheduling from dispatching. In our view, the responsibilities of a scheduler end with the generation of an execution decision. The dispatcher then uses execution decisions to send commands to hardware at a time that is consistent with execution decision.

At the conclusion of this Section, we extend our approach to scheduling variable-delay STNUs by introducing an optional procedure that addresses a shortcoming in the semantics of scheduling a variable-delay STNU. The shortcoming takes the form of potentially unnecessary wait times that are added after receiving contingent event assignments, extending the makespan of procedures. We present a generate-and-test algorithm to partially mitigate said shortcomings.

Bhargava et al. [1] addressed this ambiguity in contingent event assignment by first transforming the VDC STNU into a controllability-equivalent fixed-delay STNU. With fixed observation delay, we *do* have the guarantee that we learn the exact assignment of contingent events (so long as the observation delay is not infinite). Thus, scheduling a fixed-delay STNU only differs from scheduling a vanilla STNU in that we must subtract a fixed observation delay when we make contingent event assignments. Otherwise, the dispatchable form is the same as in the case of a vanilla STNU, and we can choose any STNU scheduling algorithm to generate execution decisions.

The flow from variable-delay STNU to fixed-delay STNU to dispatchable form may appear sufficient to enable scheduling of variable-delay STNUs, but we must contend with a novel issue: the execution spaces of the original variable-delay STNU and its transformed fixed-delay equivalent are mismatched. Nature is obliged to respect the uncertainties of the original variable-delay STNU. As will be shown later, the fixed-delay equivalent reduces the execution space to make the controllability check tractable. As such, we may receive observations outside the range of the contingent links in the fixed-delay STNU, which we must reconcile with the dispatchable form. See Figure 5-1 for an overview of the information flow in scheduling a variable-delay STNU.

5.1 Scheduling through Real-Time Execution Decisions

An STNU, S, that exhibits dynamic controllability can be *scheduled* dynamically (or *online*). At a high-level, dynamic scheduling is the process of mapping the history of event assignments to the execution time of future free events. We follow the scheduling work by Hunsberger [36], [38], which describes an $O(N^3)$ procedure, FAST-EX, for dynamic scheduling of STNUs. At its core is the notion of *Real-Time Execution Decisions* (RTEDs), which map a timepoint to a set of requirement events to be executed and are generated based on *partial schedules* of STNUs being executed. `WAIT` decisions may also be produced, reflecting the need to wait for the assignment of a contingent event before continuing. RTED-based scheduling applies a dynamic programming paradigm by first creating a dispatchable form of temporal constraints offline, updating the dispatchable form as the partial schedule is updated online, and querying the dispatchable form online to quickly find the next free event to schedule [39].

Definition 24. Real-Time Execution Decisions [39]

A *Real-Time Execution Decision* is a two-tuple $\langle t, \chi \rangle$, where:

- t is a time with domain $\mathbb{R}$,
- χ is a set of $x_r \in X_r$ to be executed at time t

Definition 25. Partial Schedule

A *Partial Schedule* for an STNU is a mapping $\xi : t \rightarrow \mathbb{R}$, where t is a proper subset of timepoints in X with domain $\mathbb{R}$.

A partial schedule is the set of assignments *so far* during execution. They can be represented by $\xi_t = \{(A, \xi(A)\}$ notation, meaning that by some time t, event A has been scheduled at time $\xi(A)$. By definition, all STNU executions begin with a partial schedule $\xi_0 = \emptyset$, meaning no timepoints have been assigned at time 0. As timepoints are executed, the partial schedule grows. For example, if event A is assigned to 1 and B to 2, then at time \$2, $\xi_2 = \{(A, 1), (B, 2)\}$.

The dispatchable form employed by FAST-EX is the *AllMax* distance graph, which is produced by the Morris $O(N^4)$ DC-checking procedure [27].

Definition 26. AllMax Distance Graph [26]

The *AllMax* distance graph is a distance graph exclusively consisting of unlabeled and upper-case edges.

The key idea of FAST-EX is maintaining accurate distances from a zero point, Z, of the graph to all events. At the outset of execution, all events from S are present as nodes in *AllMax*. As contingent events are observed, *AllMax* performs update steps using Dijkstra Single Source/Sink Shortest Path (SSSP) to maintain distances to unexecuted events, while also removing executed events. We include pseudo-code of the update step in Figure 1.

```
Input: Time $t$; Set of newly executed events Exec $\subseteq X_e \cup X_r$; AllMax Graph $G$; Distance
       matrix $D$, where $D(A, B)$ is the distance from $A$ to $B$
Output: Updated $D$
  FAST-EX Update:
1     for each continent event $C \in$ Exec do
2         Remove each upper-case edge, $Y \xrightarrow{C:-w} A$, labled by $C$;
3         Replace each edge from $Y$ to $Z$ with the strongest replacement edge;
4     end
5     for each event $E \in$ Exec do
6         Add lower-bound edge $E \xrightarrow{-t} Z$;
7     end
8     For each event $X$, update $D(X, Z)$ using Dijkstra Single-Sink Shortest Paths;
9     for each event $E \in$ Exec do
10        Add upper-bound edge $Z \xrightarrow{t} E$;
11    end
12    For each event $X$, update $D(Z, X)$ using Dijkstra Single-Source Shortest Paths;
```

Algorithm 1: Algorithm for updating distances for all events in relation to Z upon the execution of an event. Adapated from [3], Fig. 19.

Given an accurate distance matrix, we find the next RTED as follows. Let U_x be the set of unexecuted free timepoints. If U_x is empty, then the RTED is to `WAIT`. Otherwise, we find the lower bound of the earliest executable time point and the set of executable events associated with it.

$$t = \min\{-D(X, Z) \mid X \in U_x\} \tag{5.1}$$

$$\chi = \{X \in U_x \mid -D(X, Z) = t\} \tag{5.2}$$

We cannot execute events in the past. Let `now` be the current time, i.e. the last timepoint captured in ξ. It is possible that $t \leq \texttt{now}$, in which case we must reassign t to guarantee that $t > \texttt{now}$. To do so, we update t as follows, where t_U is earliest *upper* bound of the executable timepoints,

$$t_U = \min\{D(Z, X) \mid X \in U_x\} \tag{5.3}$$

$$t = \frac{\texttt{now} + t_U}{2} \tag{5.4}$$

So long as $t_U > \texttt{now}$, we know that the reassignment of t ensures $t > \texttt{now}$.

5.2 Scheduling with Variable-Observation Delay

To solidify the process of scheduling a variable-delay STNU, consider the following analogy.

> Alex wants to go hiking in the woods. The area is unfamiliar to them, so they ask their friend, Sam, who hiked these trails a long time ago, to give them directions to traverse from the trailhead to a particularly spectacular overlook. Sam has a working idea of the trail map, but their memory is imperfect. Regardless, they guarantee Alex that their directions will lead Alex to the overlook even if the woods have changed over the years. Sam writes down directions like "turn left after 500 meters at the giant oak tree" and "turn right after 100 meters when you see the brook." Alex knows that Nature will not necessarily obey Sam's directions. They may observe a giant oak tree earlier than expected, so they must then wait to take the next trail going left. Or the brook may have dried up, so they imagine they saw one near where Sam thought it would be and take the next right. While hiking, Alex is charged with reconciling Sam's directions with their own observations. Even though they may identify the landmarks in Sam's directions earlier or later than expected, their actions will need to follow Sam's instructions to maintain the guarantee of reaching the overlook.

In our analogy, S models the current state of the hiking trails and the full range of projections, while S' is Sam's working memory of them. Sam's directions are the execution strategy described by the AllMax graph we get by checking the fixed-delay controllability of S'. Observations of Nature obey S. Alex is charged with reconciling their observations from S with Sam's hiking directions from S'. The analogy ends here, though, as the math and logic of temporal reasoning do not neatly translate into hiking. Luckily, we have more information than Alex. Unlike human memory, which is untrustworthy and irrational, the fixed-delay STNU, S', is created by a set of Lemmas with deterministic outcomes. As such, we have the means to interpret how observations in S *would appear* in S', which will be critical in adapting our fixed-delay execution strategy in response to variable observation delay.

Our key challenge for scheduling an STNU with variable observation delay is reconciling observations from S with the dispatchable form from S'.

5.3 Recording Contingent Event Assignments

During execution, we observe the outcome of contingent events $\texttt{obs}(x_c)$ in S, but we make assignments in the dispatchable form of $\xi(x'_c)$ in S'. Despite being equivalent with respect to controllability, the bounds of contingent links x_c in S and x'_c in S' are not equivalent.

We need a modified procedure for contingent event assignments that wraps FAST-EX. No modifications to FAST-EX are necessary to schedule fixed-delay STNUs because checking FDC includes the procedure of creating the same AllMax graph that FAST-EX requires.

We now present our strategy for recording observations during execution as derived from the transformations outlined in Section 4.3.

Lemma 8. *For any contingent event, $x_c \in S$ or $x'_c \in S'$, observing x_c at time $t \in [l^-(x_c), u^+(x_c)]$ fixes the observation to $\texttt{obs}(x_c) = [t, t] = t$.*

Proof. Prior to execution, observations are defined as set-bounded intervals from the earliest possible observation at $l^-(x_c)$ to the last possible observation at $u^+(x_c)$. Receiving an observation $\texttt{obs}(x_c) = t$ during execution eliminates all members of the pre-execution interval except t. □

Lemma 9. *For any contingent event $x'_c \in X_c$ in fixed-delay controllable S', if $\gamma(x'_c) = \infty$, we mark the event executed but do not assign ξ(x'$_c$) in the dispatchable form of S'.*

Proof. If we are scheduling a fixed-delay STNU, S', that is already known to be fixed-delay controllable, an execution strategy must exist that is independent of the assignment of $\xi(x'_c)$ when $\gamma(x'_c) = 0$. We are not required to record $\xi(x'_c)$ when $\gamma(x'_c) = \infty$ to guarantee controllability and may safely ignore it.

We mark the event executed to prevent it from appearing in future RTEDs.

□

Lemma 9 may be applicable to any contingent events, $x'_c \in X_c$ in S' that were transformed from the variable-delay form S using Lemmas 1, 2, or 3.

Lemma 10. *For any contingent event $x'_c \in X_c$ in fixed-delay controllable S', if $\gamma(x'_c) \in \mathbb{R}$, we assign $\xi(x'_c) = \texttt{obs}(x_c) - \gamma(x'_c)$ in the dispatchable form of S'.*

Proof. The central challenge of checking fixed-delay controllability is determining that an execution strategy exists that allows an agent to wait an additional $\gamma(x'_c)$ time units after a contingent event has been assigned to learn its outcome. Importantly, the γ function is not used to modify the edges of the labeled distance graph, which are derived from the constraints $r \in R_e \cup R_c$ in S'.

As $\gamma(x'_c)$ resolves to a known and finite value, we can derive the true value of ξ(x'$_c$)\$ to be assigned in the labeled distance graph. Contingent event assignments, ξ(x'$_c$)\$, are recorded in the labeled distance graph as follows, where $\texttt{obs}(x_c)$ is the resolved observation,

$$\xi(x'_c) = \texttt{obs}(x_c) - \gamma(x'_c) \tag{5.5}$$

□

Next, in comparing the bounds of x_c and x'_c when $u - l \geq \bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)$, $x'_c \in [l^+(x_c), u^-(x_c)]$ (Lemma 4) there are three regimes of observations of $\texttt{obs}(x_c)$ we must consider:

1. $\texttt{obs}(x_c) \in [l^-(x_c), l^+(x_c))$, ie. strictly earlier than the range of $\xi(x'_c)$,
2. $\texttt{obs}(x_c) \in [l^+(x_c), u^-(x_c)]$, ie. the range equivalent to x'_c, and
3. $\texttt{obs}(x_c) \in (u^-(x_c), u^+(x_c)]$, ie. strictly later than the range of $\xi(x'_c)$.

Nature decides in which regime we receive $\texttt{obs}(x_c)$. We are faced with the unique challenge of deciding how to act when Nature selects an $\texttt{obs}(x_c)$ that fails to follow the constraints of S', eg. $\texttt{obs}(x_c) < l^+(x_c) \vee \texttt{obs}(x_c) > u^-(x_c)$, which would lead to an assignment, $\xi(x'_c)$, in the first or third regimes above. In plainer words, the contingent links of S and S' do not have the same constraints. We make assignments in S', but we receive observations from S. We need to decide how to act when we observe a contingent event earlier or later than we expect according to S', because if we blindly assigned $\xi(x'_c)$ outside its constraints from S', we lose the guarantee of controllability. Our only choice is to find a strategy to assign x'_c that respects the constraints of S', despite observing x_c earlier or later than expected. We do so by reasoning over the possible *range* of assignments, $\xi(x_c)$, that could have led to a particular observation, $\texttt{obs}(x_c)$. What we find is that, due to the uncertainty in observation delay, we are allowed to *modify* our assignment of $\xi(x'_c)$ to ensure it respects S'. We present two modification strategies for addressing the first and third cases, which we call *buffering* and *imagining* respectively.

We first address the case where $\texttt{obs}(x_c) < l^+(x_c)$.

Lemma 11. *If a contingent event, $x_c \in X_c$, is observed earlier than the bounds of x'_c in S' for a fixed-delay controllable S', $\texttt{obs}(x_c) < l^+(x_c)$, we perform a buffering operation by letting $\xi(x'_c) = l^+(x_c)$ in S'.*

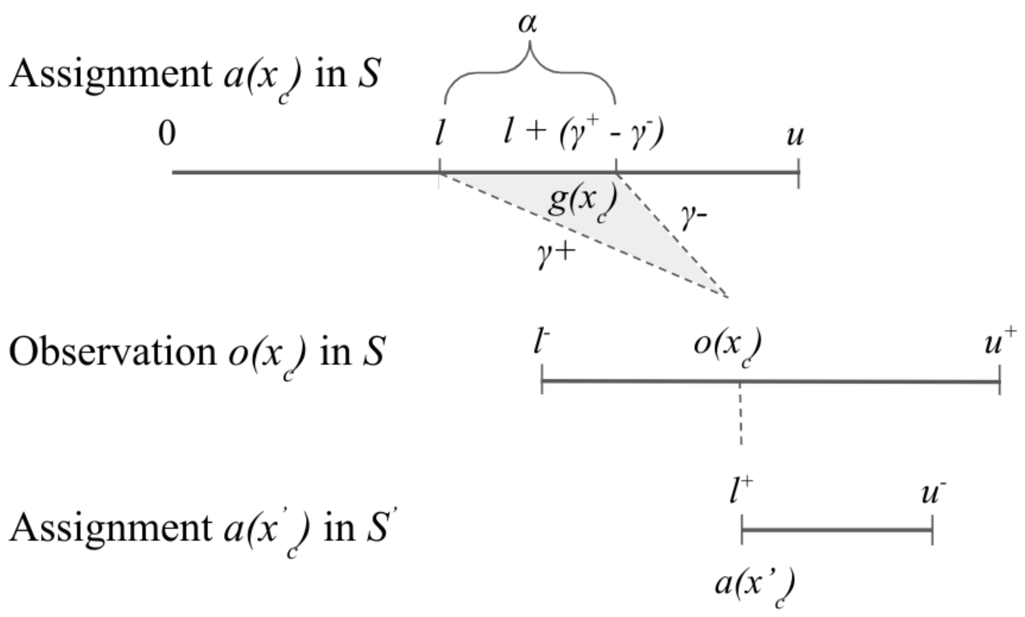

α
Assignment $a(x_c)$ in S
0
l
$l + (\gamma^+ - \gamma^-)$
u
$g(x_c)$
$\gamma-$
$\gamma+$
Observation $o(x_c)$ in S
l^-
$o(x_c)$
u^+
l^+
u^-
Assignment $a(x'_c)$ in S'
$a(x'_c)$

Figure 5-2: Here, we show how the combination of $\xi(x_c)$ and $\bar{\gamma}(x_c)$ lead to an assignment of $\xi(x'_c)$ in S'. We see the range $\alpha \in [l, l + \bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)$ representing the earliest and latest assignments of ξ(x_c)$ that could result in $\texttt{obs}(x_c) \in \xi(x'_c) \in [l^+(x_c)$, 1ˆ+(x_c)]$. The grey region represents the range of possible observation delays, $\bar{\gamma}(x_c)$, supporting $\xi(x'_c) \in [l^+(x_c), l^+(x_c)]$.

Proof. To demonstrate why buffering is sound, we compare the bounds of x_c in S and x'_c in S' to show that our execution strategy for $\xi(x'_c)$ is applicable to any $\xi(x_c) \in [l, l^+(x_c)]$.

We know that S' is fixed-delay controllable when $\xi(x'_c) \in [l^+(x_c), u^-(x_c)]$. Consider an observation at the lower bound of $ξ(x'_c), $\texttt{obs}(x_c) = l^+(x_c)$. We can discern the range of possible assignments of x_c in S (Using Lemma 8 to rewrite $o(x_c) = l^+(x_c)$ as $o(x_c) = [l^+(x_c), l^+(x_c)]$).

$$\begin{aligned}
\texttt{obs}(x_c) &= \xi(x_c) + \bar{\gamma}(x_c) \\
\xi(x_c) &= \texttt{obs}(x_c) - \bar{\gamma}(x_c) \\
\xi(x_c) &= [l^+(x_c), l^+(x_c)] - [\bar{\gamma}^-(x_c), \bar{\gamma}^+(x_c)] \\
\xi(x_c) &= [l, l + (\bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c))]
\end{aligned}$$

Let $\alpha = [l, l + (\bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c))]$ for this Lemma.

Given S' is fixed-delay controllable, there must exist an execution strategy when $\xi(x'_c) = l^+(x_c)$, which entails the same execution strategy applies for any assignment of $\xi(x_c) \in \alpha$. Thus, during execution, if we can show that $\xi(x_c) \subseteq \alpha$, we can safely act as if $\xi(x'_c) = l^+(x_c)$.

Now, let $\texttt{obs}(x_c) = l^+(x_c) - \epsilon$ for some small, positive ϵ. Accordingly, it is the case that $\xi(x_c)$ must fall in the range,

$$\xi(x_c) = [(l^+(x_c) - \epsilon) - [\bar{\gamma}^-(x_c), \bar{\gamma}^+(x_c)]$$
$$\xi(x_c) = [l^+(x_c) - \epsilon, l^+(x_c) - \epsilon] - [\bar{\gamma}^-(x_c), \bar{\gamma}^+(x_c)]$$
$$\xi(x_c) = [l - \epsilon, l + (\bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)) - \epsilon]$$

Of course, $\xi(x_c)$ must respect the original bounds of x_c, $x_c \in [l, u]$.

$$\xi(x_c) = [l - \epsilon, l + \bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c) - \epsilon] \cap [l, u]\xi(x_c) \qquad = [l, l + (\bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)) - \epsilon]$$

Let $\beta = [l, l + (\bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)) - \epsilon]$ for this Lemma. See Figure 5-2 for a visual representation of how an observation $\texttt{obs}(x_c)$ is interpreted as an assignment ξ(x'c)$ during scheduling.

We see that $\beta \subset \alpha$. Thus, if we receive an observation $\texttt{obs}(x_c)$ earlier than $l^+(x_c)$, we may safely buffer by applying the execution strategy from an assignment of $\texttt{obs}(x_c) = \xi(x'_c) = l^+(x_c)$. □

Next,we address the case where $\texttt{obs}(x_c) > u^-(x_c)$.

Lemma 12. *If a contingent event, $x_c \in X_c$, will be observed after the bounds of x'_c, $\texttt{obs}(x_c) > u^-(x_c)$, we imagine we have received it by assigning $\xi(x'_c) = u^-(x_c)$ in S'.*

Proof. We apply the same argument to *imagining* late events. We now consider an observation at the upper bounds of x'_c, $\texttt{obs}(x_c) = \xi(x'_c) = u^-(x_c)$. We then have a new α representing the range of the earliest and latest assignments to $\xi(x_c)$,

$$\alpha = u^-(x_c) - g(x_c)$$
$$= [u^-(x_c), u^-(x_c)] - [\bar{\gamma}^-(x_c), \bar{\gamma}^+(x_c)]$$
$$\alpha = [u - (\bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)), u]$$

Once again, if S' is fixed-delay controllable, there must exist an execution strategy for $\xi(x'_c) = u^-(x_c)$. It follows that we can apply this execution strategy when $\xi(x_c) \in \alpha$.

If we receive a late observation, $\texttt{obs}(x_c) = u^-(x_c) + \epsilon$, we find that $\xi(x_c)$ must fall in the range of a new β, where

$$\begin{aligned}\beta &= \big[(u^-(x_c)+\epsilon) - g(x_c)\big] \cap [l,u] \\ &= \big[[u^-(x_c)+\epsilon, u^-(x_c)+\epsilon] - [\bar{\gamma}^-(x_c), \bar{\gamma}^+(x_c)]\big] \cap [l,u] \\ &= [u - (\bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)) + \epsilon, u+\epsilon] \cap [l,u] \\ \beta &= [u - (\bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)) + \epsilon, u]\end{aligned}$$

We find that $\beta \subset \alpha$ again and can safely imagine that we received $\texttt{obs}(x_c) = u^-(x_c)$. Of course, we need not wait to receive a late observation of x_c only to assign it to a time in the past. During execution, if we have not received $\texttt{obs}(x_c)$ by $u^-(x_c)$, we imagine an observation arrived at $\texttt{obs}(x_c) = u^-(x_c)$ and thus assign $\xi(x'_c) = u^-(x_c)$. We then ignore the real observation of x_c that we receive later. □

We have addressed the key issue of reconciling observations from S with the dispatchable form from S'. We now present a dispatcher and wrapper algorithms on top of FAST-EX that combine to add robustness for variable observation delay.

5.4 Modified FAST-EX for Variable Observation Delay

We present an overview of the scheduling algorithm below with explanations following.

While we made a careful distinction between x_c and x'_c in our discussion of scheduling, in our implementation it was important to be able to easily replace one with another when looking up values in hash-tables and lists. For instance, to implement Equation 5.5, we receive x_c but key the fixed-delay function on x'_c. Rather than adding an additional translation layer, we give each temporal event in S a unique name, all of which get copied to their equivalent events in S'. Hash-tables are keyed on event names, vastly simplifying lookups in the AllMax graph, delay function, and elsewhere.

Let x be a temporal event, $x \forall x \in X_c \cup X_e$.

```
Input: AllMax Graph $G$; fixed-delay function $\gamma(x'_c)$; Observation $\mathtt{obs}(x_c)$
Output: Updated AllMax Graph $G$
Initialization:
1  $\xi(x'_c) \leftarrow \mathtt{obs}(x_c) - \gamma(x'_c)$;
VDC-FAST-EX-Update:
2  for $l \in S'.contingentLinks()$ do
3      $x_c \leftarrow l.endpoint()$;
4      $a, b \leftarrow l.bounds()$;
5      if $\bar{\gamma}^+(x_c) == \infty$ or $\bar{\gamma}^+(x_c) == \bar{\gamma}^-(x_c)$ then
6          $\gamma'(x_c) \leftarrow \bar{\gamma}^+(x_c)$;
7      endif
8      else if $b - a < \bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)$ then
9          $\gamma'(x_c) \leftarrow \infty$;
10     endif
11     else
12         $l.setBounds(a + \bar{\gamma}^+(x_c), b + \bar{\gamma}^-(x_c))$;
13         $\gamma'(x_c) \leftarrow 0$;
14         for $l' \in x_c.outgoingReqLinks()$ do
15             $u, v \leftarrow l'.bounds()$;
16             $l'.setBounds(u - \bar{\gamma}^-(x_c), v - \bar{\gamma}^+(x_c))$;
17         end
18         for $l' \in x_c.incomingReqLinks()$ do
19             $u, v \leftarrow l'.bounds()$;
20             $l'.setBounds(u + \bar{\gamma}^+(x_c), v + \bar{\gamma}^-(x_c))$;
21         end
22     endif
23 end
24 return $S', \gamma'$
```

Algorithm 2: Algorithm for updating the AllMax graph when an observation arrives

5.4.1 Real vs No-op Events

The introduction of buffering and imagining events creates a new distinction between temporal events: there are events that need to be executed by the agent and there are those events that do not. We call these *real* and *no-op* ("no operation") events. Both contingent *and* requirement events may fall into either category. Below, we present our rationale for the distinction between real and no-op events, and how we modify real-time execution decisions accordingly.

To start, both buffered and imagined contingent events are no-ops. Both cases represent timepoints that we use to update our dispatchable form to maintain consistency with S'.

Consider the process of normalization of an STNU [27]. While building the labeled distance graph during a dynamic controllabillity check, we rewrite contingent links such that their lower bounds are always 0. For instance, for a contingent event C and free event E, $C - E \in [l, u]$, during normalization we create a new requirement event, C', fixed at the lower bound of the contingent link, and then shift the bounds of the contingent link to start at 0 while maintaining the original range, $u - l$. This results in two constraints: $E - C' \in [l, l]$ and $C - C' \in [0, u - l]$ that still reflect the original contingent link's semantics.

To a scheduler, there is no distinction between the semantics of a real event, as modeled by a human planner writing an STNU for an agent to execute, and C', an artifact of checking controllability. Both are modeled in the AllMax distance graph forming the basis of RTED generation. However, an agent does not need to execute any task in the outside world to satisfy $E - C'$. We take a view that the only information our agent has about the timepoints it should execute comes from the input STNU. Thus, we need RTEDs to reflect the distinction between requirement events that are *real*, meaning the agent is responsible for taking some action to execute them, and those that are *no-ops*, or algorithmic by-products that require no operation. This distinction naturally leads to the following addendum to the definition of RTEDs.

Definition 27. Event-No-op Pair

An *Event-No-op Pair*, *event-noop*, is a two-tuple, $\langle x, noop \rangle$, where:

- x is an event in $X_e \cup X_c$,
- *noop* is a boolean, where if true, the event does not correspond to an action an agent should take, else real.

Definition 28. RTED with Operational Distinction

A *Real-Time Execution Decision with Operational Distinction* is a two-tuple $\langle t, \textit{event-noops} \rangle$, where:

- t is a time with domain $\mathbb{R}$,
- *event-noops* is a set of *event-noop* pairs to be executed at time t.

For convenience and simplicity, and given the similarities between RTED and RTED with Operational Distinction, future references to RTEDs will always mean RTEDs with Operational Distinctions.

5.5 Dispatching

This thesis contributes a dynamic dispatching algorithm for which the process of generating RTEDs is a subroutine. In our view, RTEDs are not commands to the agent. Rather, they inform the agent of the time windows where actions ensure consistency. As such, a dedicated dispatcher layer is required to translate RTEDs to real actions at the right time. The dispatcher will request RTEDs and then wait until the time window of the execution to trigger their execution.

A *dynamic dispatcher* (or just "dispatcher") is an interface layer situated between the scheduler and a *driver* that communicates with hardware. The dispatcher has a two-fold responsibility: it triggers the execution of RTEDs in the outside world by communicating with the driver (Section 5.5.1), and it relays observations from the outside world about the execution of events to the scheduler (Section 5.5.2). An explicit dispatching layer allows us to centralize the logic for interacting with the outside world therein, keeping the scheduler simple. In the implementation of Kirk used in this thesis, the scheduler wholly consists of the algorithms described above, nothing more. We go so far as to enforce that the scheduler itself has no notion of a clock. Instead, the dispatcher has a clock. When the dispatcher wants the scheduler to update itself, it is required to send both an event and a elapsed time to the scheduler.

Consequently, the dispatching algorithm is separate from the scheduler. As such, there is no hard requirement on the FAST-EX-based scheduler described above. Any scheduling algorithm that produces RTEDs adhering to Definition 28 would be compatible with the dispatcher described below.

5.5.1 Dynamic Event Dispatching

The dynamic dispatcher runs the main loop of the executive's temporal reasoning routine. The inner loop, Algorithm 4, is responsible for retrieving the latest RTEDs and firing driver commands when the clock indicates that the agent is inside RTED time windows. The outer loop, Algorithm, 3, runs continuously until the scheduler reports that there are no free events remaining to schedule. The dispatcher requests RTEDs with blocking synchronous calls, while the dispatcher and driver communicate asynchronously. The dispatcher spawns a thread to make non-blocking calls to the driver's interface to execute events. The dispatcher and driver also share a FIFO queue that the driver can append messages to indicating the successful execution of events.

We now provide a walkthrough of the dynamic dispatching algorithm. For simplicity's sake, the term *schedule* here is shorthand for whatever data structures the scheduler uses to generate RTED. *Updating the schedule* may be used to refer to making an event assignment in the scheduler, triggering any necessary changes to the schedule.

The interaction between the inner and outer loop is limited. The inner loop returns a Boolean indicating whether there are executable events remaining. The outer loop is a simple `while` that repeats until it receives `false` from the inner loop. Otherwise, the only communication between the inner and outer loops is a variable containing the last RTED that was generated but not executed. The outer loop creates the variable and passes it by reference to the inner loop. The inner loop is free to use or modify the variable as it sees fit.

We break the inner loop of algorithm into three distinct phases.

1. Receive execution confirmation from the driver.
2. Collect an RTED and confirm the clock time is within the execution window.
3. If there is an RTED:
   (a) send executable events to the driver, else
   (b) immediately assign all `noop` events to the current time.

Our goal in the inner loop is to dispatch events to the driver only after updating the schedule, collecting an up-to-date RTED, and confirming we are within the time window of the RTED. The loop will exit before reaching the dispatch step if any conditions are not met.

For the first step, we ask the scheduler if there are any remaining executable events. If there are none, we return `false` to signal the loop's termination, otherwise we continue.

Next, we check the FIFO queue for any event execution messages returned from the driver. The presence of a message would indicate that the driver has successfully executed a free event. We iteratively pop messages off the queue and update the schedule with the events and execution time contained in each message. Note that the scheduler update is a blocking operation because we need an up-to-date schedule to guarantee future RTEDs are consistent. We then invalidate the last RTED generated.

The second step starts once we have popped all messages from the driver off the queue. If we do not have a valid RTED from the last iteration of the inner loop, we ask the scheduler for one and save it to the referenced variable from the outer loop. Given that we interact with the driver asynchronously, it is possible that the current RTED is one that has already been sent to the driver but we have yet to receive a message confirming its execution. If so, there is nothing to do so we return `true`.

Lastly, we compare the suggested time in the RTED against the clock's elapsed time. Given the relationship between the scheduler, inner loop, and driver, we do not assume that dispatched events are executed instantaneously by the driver. We know that execution contends against delays such as the computational time in simply calling a function, to network latency, to robotic hardware that takes a moment to interpolate a motion plan from waypoints. In some contexts, it may make sense to preempt execution by dispatching events some small amount of time *before* the clock time reaches the RTED execution window. We call this preemption time ϵ, where $\epsilon \in \mathbb{R}^{\geq 0}$. Thus, we dispatch events, `dispatch-p`, when `dispatch-p` $= (t_{\text{RTED}} - t_{\text{clock}} \leq \epsilon)$. If $\epsilon = 0$, the dispatcher is not allowed to preemptively dispatch events before the RTED time. We allow the human operator to choose an ϵ that is consistent with the operational context for the driver.

If `dispatch-p` is `false`, we are too early to execute the RTED and so the loop returns `true`. Otherwise we continue.

Once we reach the third stage, we are guaranteed to be able to safely dispatch events because (1) we have confirmed that the RTED we have in hand has unexecuted events that have never been dispatched, and (2) that we are in a time window that the scheduler has told us is consistent with the STNU's constraints. Going forward, we take advantage of the operational distinction we added to Hunsberger's RTEDs in Definition 28. Using the *noop* property of each *event-noop* pair in the RTED, we filter the *event-noop* pairs into a set of `noop` events and a set of real events. The real events are asynchronously sent to the driver. We then loop through the `noop` events and schedule them in turn.

Finally, because events were dispatched, the inner loop returns `true`.

```
Initialization: $RTED_{last} \leftarrow \varnothing$
  Dynamic Dispatching Outer Loop:
1   while Calling inner loop with $RTED_{last}$ returns true do
2   |   continue
3   end
```
Algorithm 3: The outer loop of the dynamic dispatching algorithm.

The biggest contributor to the performance of the inner loop, Algorithm 4, is updating the schedule. Assuming the *Scheduler* is the Delay Scheduler described in Section 5.4, then performing an assignment of an event will trigger the FAST-EX update that runs in $O(N^3)$ [36, p. 144] with the number of events in the STNU. In the worst case, all events in the STNU arrive at the same time, whether as messages from the driver in the FIFO queue, or RTED `noop` events. Thus, the dynamic dispatcher's inner loop runs in $O(N^4)$.

5.5.2 Observing Contingent Events

The dispatcher relays contingent event observations to the scheduler. In the base case, when a contingent event is observed, the dispatcher updates the schedule with the event and current clock time. If this were the only responsibility of the dispatcher when receiving a contingent event, we would end the section here. However, this interface is also where we implement an *Optimistic Rescheduling* technique to address a problem inherent to the buffering performed by the Delay Scheduler.

We describe Optimistic Rescheduling below and present the full contingent event observation algorithm.

1. Optimistic Rescheduling

Input: *Scheduler*; *Driver*; FIFO queue, *Queue*; $RTED_{last}$; ϵ;
Output: Boolean whether the outer loop should continue
Initialization: $events_{real} \leftarrow \{\}$; $events_{\mathbf{noop}} \leftarrow \{\}$;

```
Dynamic Dispatching Inner Loop:
1  if Scheduler has no more unexecuted events then
2      return false;
3  endif
4  for message in Queue do
5      Pop message;
6      for $event, t_{execution}$ in message do
7          Set $\xi(event) = t_{execution}$ in Scheduler;
8      end
9      $RTED_{last} \leftarrow \varnothing$;
10 end
11 $RTED \leftarrow$ a new RTED from Scheduler; //Equations 5.2 and 5.4
12 if $RTED = RTED_{last}$ then
13     return true;
14 endif
15 $RTED_{last} \leftarrow RTED =$;
16 if $t_{RTED} - t_{current} > \epsilon$ then
17     return true;
18 endif
19 for event-noop pair in $RTED_{event\text{-}noops}$ do
20     if event-noop[noop] is true then
21         Add event-noop[x] to $events_{\mathbf{noop}}$;
22     else
23         Add event-noop[x] to $events_{real}$;
24     endif
25 end
26 Asynchronously send all $events_{real}$ to the Driver;
27 for event in $events_{\mathbf{noop}}$ do
28     Set $\xi(event) = t_{RTED}$ in Scheduler;
29 end
30 return true;
```

Algorithm 4: The inner loop of the dynamic dispatching algorithm.

We return to problem of potentially unnecessary wait time created by the buffering execution strategy described in Lemma 11. First, we use an example to demonstrate how buffering early contingent events results in a reduction of the execution space. Then we contribute a technique for managing event observations that circumvents the loss of execution space.

Consider the following variable-delay controllable STNU, which we will refer to as *Bufferable*.

$$A \overset{[1,7]}{\Longrightarrow} \overset{\bar{\gamma}\in[1,3]}{B} \xrightarrow{[5,9]} C$$

Following the semantics of the delay scheduler, we would first transform *Bufferable* to its fixed-delay equivalent, *Bufferable*$'$ by applying Lemma 4.

$$A' \overset{[4,8]}{\Longrightarrow} \overset{\gamma=0}{B'} \xrightarrow{[4,6]} C'$$

If we assume A is executed at $t = 0$, the only question is when to schedule C (or its fixed-delay equivalent, C'). According to the semantics of *Buffering*, if B is observed at $t = 2$, we know that B was assigned at $t = 1$. Thus, we only need to wait until $t = 6$ to schedule C. However, the delay scheduler would schedule according the constraints found in *Buffering*$'$, wherein $\xi(B') = 2$ falls earlier than the lower bound of $A' \overset{[4,8]}{\Longrightarrow} B'$, triggering Lemma 11. As a result, we act as if $\xi(B') = 4$ and then wait for the lower bound of $B' \xrightarrow{[4,6]} C'$. The end result is that C' is assigned to a later time of $t = 8$.

From a human mission manager perspective, this wait appears to be a waste. Time is money. And in the case of planetary exploration, time is safety. If a NASA flight controller were to ask why your software is telling astronauts on Moon to just stand there doing nothing, responding that your algorithm *does not know* if it is safe to act, would be unacceptable. Therefore, we contribute a generate-and-test approach that looks for opportunities to avoid buffering when contingent events arrive before their expected windows in the fixed-delay STNU. The goal of this method is to dispatch future events earlier if possible.

At its core, Optimistic Rescheduling consists of copying the original variable-delay STNU then rewriting it to reflect the resolution of uncertainty so far. Key to rewriting the variable-delay STNU is narrowing the constraint and observation delay to match what was observed. We then re-perform controllability checks. If controllable, we have a new schedule that removes the need to buffer this contingent event. If not controllable, we do nothing, buffer the contingent event as planned, and continue dispatching against the original schedule.

We now step through the Event Observations with Optimistic Rescheduling algorithm (Algorithm 5) in detail.

We cannot know if an event is buffered if we do not attempt to schedule it. Our first step is to schedule an event like normal. If scheduling is possible without buffering, we simply return

Input: Original VDC STNU S; Equivalent fixed-delay function γ;
Partial history ξ; Executed events map $Ex(S, x)$; Observed contingent event x; Normalized lower bound $\hat{x}$; Current time t;
Output: Boolean whether x was successfully scheduled, VDC STNU

```
Event Observations with Optimistic Rescheduling:
1  $successp, bufferedp \leftarrow$ updateSchedule(S, x, t);
2  if $\neg bufferedp$ then
3      return $successp, S$;
4  endif
5  $S^* \leftarrow$ rewriteSTNU(S, x, t);
6  if $S^*$ is not variable-delay controllable then
7      return $successp, S$;
8  endif
9  for $a$ in $\xi$ //$a$ is an assignment
10  do
11     if $\gamma(a[event]) \neq \infty$ then
12         updateSchedule($S^*$, $a[event]$, $a[time] + \gamma(a[event])$);
13     endif
14 end
15 for $event$ in $Ex(S)$ do
16     $Ex(S^*, x) \leftarrow Ex(S, x)$
17 end
18 updateSchedule($S^*$, $\hat{x}$, t);
19
20 return true, $S^*$;
```

Algorithm 5: An Algorithm for observing contingent events with Optimistic Rescheduling.

whether scheduling was successful.

If the event was buffered, then we begin to optimistically reschedule. We do so by tightening the bounds of the original VDC STNU, $S_{original}$, based on the observation we received, which is the responsibility of Algorithm 6, implementing Lemma 13.

If the rewritten STNU, S^*, is found to be VDC, we prepare to schedule it. First we iterate through all the assignments in the partial schedule and make the same assignments against the new STNU. When assignments are made, we subtract out the fixed observation delay. In this loop, we add the observation delay back, lest it be subtracted from the original observation twice.

If any contingent events with infinite delay were observed, they would have been marked executed but not assigned. We iterate through the executed events of S and mark the same events executed in S^*.

The distance graph, partial schedule, and executed events of S^* now match that of S before x_c was received. We are almost safe to record a new observation. Lastly, we must address the executable event representing the normalized lower bound of x_c, $\hat{x}_c$. During scheduling, we would have received an RTED consisting of $\langle l + \bar{\gamma}^+(x_c), \hat{x}_c\rangle$. Given that x_c arrived before $l + \bar{\gamma}^+(x_c)$, we never would have assigned $\hat{x}_c$, so we assign $\xi(\hat{x}_c) = t$ now. We finally update the schedule with the contingent event that arrived.

Lemma 13. *If a contingent event, $x_c \in X_c$, where $u - l > \bar{\gamma}^+(x_c) - \bar{\gamma}^-(x_c)$, is observed at time t and when $t < l + \bar{\gamma}^+(x_c)$, we may replace x_c and $\bar{\gamma}(x_c)$ with a constraint, x_c^*, and variable-delay function, $\bar{\gamma}(x_c^*)$, with narrower bounds as follows.*

$$\begin{aligned} x_c^* &= [l^*, u^*] \\ x_c^* &= [\max(l, t - \bar{\gamma}^+(x_c)), \min(u, t - \bar{\gamma}^-(x_c))] \\ \bar{\gamma}(x_c^*) &= [\max(\bar{\gamma}^-(x_c), t - u), \min(\bar{\gamma}^+(x_c), t - l)] \end{aligned}$$

Proof. Buffering is only possible if the conditions of Lemmas 4 and 11 are triggered. By Lemma 4, we are guaranteed to be able to narrow where in the range $[l, u]$ x_c was scheduled. By Lemma 11, we know that rewritten bounds will lead to an assignment of x_c that is no later than $l + \bar{\gamma}^+(x_c)$. Our tool for narrowing the bounds is Equation 5.5, which allows us to use the observation to reason over the assignment and observation delay. Our strategy is to look at the extreme cases leading to an observation.

We start by reasoning over the earliest and latest assignments respectively. In order for x_c to be assigned as early as possible, l^*, we assume the delay has taken on its maximum value, $\bar{\gamma}^+(x_c)$.

$$\xi(x_c) = \texttt{obs}(x_c) - \gamma(x_c) \tag{5.6}$$

$$l^* = t - \bar{\gamma}^+(x_c) \tag{5.7}$$

Likewise, to find the last possible assignment leading to an observation, we subtract the smallest observation delay, $\bar{\gamma}^-(x_c)$.

$$u^* = t - \bar{\gamma}^-(x_c) \tag{5.8}$$

Given that Nature will adhere to the constraints originally put forth in S, the bounds of x_c^* must remain within the bounds of x_c. Hence, we guarantee the lower bound is at least l while the upper bound is at most u.

$$l^* = \max(l, t - \bar{\gamma}^+(x_c))$$
$$u^* = \min(u, t - \bar{\gamma}^-(x_c))$$

We use the same logic for narrowing the observation delay. If x_c was assigned as late as possible, u, then the observation delay would be minimized, $\bar{\gamma}^-(x_c^*)$. Likewise, if x_c was assigned as early as possible, l, the observation delay would be maximized, $\bar{\gamma}^+(x_c^*)$. The narrowed lower and upper bounds of $\bar{\gamma}(x_c)^*$ are as follows.

$$\gamma = \texttt{obs}(x_c) - \xi(x_c)$$
$$\bar{\gamma}^-(x_c^*) = t - u$$
$$\bar{\gamma}^+(x_c^*) = t - l$$

As before, the bounds of $\bar{\gamma}(x_c^*)$ must stay within the original bounds of $\bar{\gamma}(x_c)$, leaving us with the following narrowed observation delay.

$$\bar{\gamma}^-(x_c^*) = \max(\bar{\gamma}^-(x_c), t - u) \tag{5.9}$$
$$\bar{\gamma}^+(x_c^*) = \min(\bar{\gamma}^+(x_c), t - l) \tag{5.10}$$

□

We revisit the example from the beginning of this section to see Lemma 13 in action. As we saw before, any $\texttt{obs}(B)$ before $t = 4$ will result in buffered assignments.

$$A \xRightarrow{[1,7]} \overset{\bar{\gamma}\in[1,3]}{B} \xrightarrow{[5,9]} C$$

Let $t = 3$. We will step through the reasoning for narrowing the bounds of x_c accordingly.

$$
\begin{aligned}
x_c^* &= [\max(l, t - \bar{\gamma}^+(x_c)), \min(u, t - \bar{\gamma}^-(x_c))] \\
x_c^* &= [\max(1, 3-3), \min(7, 3-1)] \\
x_c^* &= [1, 2]
\end{aligned}
$$

$$
\begin{aligned}
\bar{\gamma}(x_c^*) &= [\max(\bar{\gamma}^-(x_c), t-u), \min(\bar{\gamma}^+(x_c), t-l)] \\
\bar{\gamma}(x_c^*) &= [\max(1, 3-7), \min(3, 3-1)] \\
\bar{\gamma}(x_c^*) &= [1, 2]
\end{aligned}
$$

We find that $\xi(x_c)$ must have fallen somewhere in the range of $[1, 2]$, while $\bar{\gamma}(x_c)$ was resolved somewhere in $[1, 2]$. Looking at the extremes, it is clear that there are multiple combinations of the assignment and observation delay that could lead to an observation at $t = 3$. While the narrowed range allows for observations other than $t = 3$, for instance, if $\xi(x_c) = 2$ and $\texttt{obs}(x_c) = 2$ yielding an observation at $t = 4$, there are no other ranges of assignments or observation delay outside of $\xi(x_c) \in [1, 2]$ and $\bar{\gamma}(x_c) \in [1, 2]$ that would allow an observation at $t = 3$.

```
Input: VDC STNU $S_{original}$; Variable-delay function $\bar{\gamma}$;
Observed contingent event $x$; Observation time $t$;
Output: VDC STNU
Initialization: $S_{new} \leftarrow \texttt{copy}(S_{original})$
  Rewrite STNU:
1     for constraint in $S_{new}$ do
2         if constraint ends in $x$ then
3             $constraint[lower] \leftarrow \max(constraint[lower], t - \bar{\gamma}^+(x))$;
4             $constraint[upper] \leftarrow \min(constraint[upper], t - \bar{\gamma}^-(x))$;
5             $\bar{\gamma}^-(x) \leftarrow \max(\bar{\gamma}^-(x), t - constraint[upper])$;
6             $\bar{\gamma}^+(x) \leftarrow \max(\bar{\gamma}^+(x), t - constraint[lower])$;
7         endif
8     end
9     return $S_{new}$;
```

Algorithm 6: An Algorithm for rewriting an STNU given the resolution of uncertainty of a contingent link.

The complexity of Algorithm 5 is dominated by the loop over `updateSchedule`. Each call to `updateSchedule` is $O(N^3)$ in the number of events. In the worst case scenario, every event up to the last contingent event has been scheduled, giving us a complexity of $O(N^4)$. We discuss potential means for improving Optimistic Rescheduling in Section 9.1.

5.6 Experimental Analysis

Scheduling always follows constraints

Optimistic vs normal VDC scheduling

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Coordinating Multiple Agents under Uncertain Communication

In this chapter, we present a novel MA framework for dynamic event scheduling with inter-agent temporal constraints. Our framework adheres to the variable observation delay modeling framework presented in Chapter 4, making it robust to uncertain communication.

Online MA coordination of event dispatching allows executives to dynamically decide when to act given the resolution of inter- and intra-agent temporal constraints. In our formulation, each executive has its own STNU with contingent events it expects to observe and free events it is responsible for monitoring. We do not distinguish between contingent events that are the free events scheduled by peer agents and contingent events from any other source in Nature. There are no restrictions on inter-agent constraints, though they must avoid chained contingencies the same way that vanilla, single-agent STNUs do [25].

We set forth the following requirements for the framework we contribute in this thesis.

- Executives are *not* required to have perfect knowledge of the complete state of the world, nor are they required to even *agree* on the state of the world. Rather, their knowledge should be consistent with the temporal constraints and observation delay modeled in their individual STNUs.
- Executives are allowed to ignore observations.
- *All* inter-agent communications must be explicitly modeled.

To our knowledge, no such online scheduler for MA coordination has been proposed. In this chapter, we first present a grounded Artemis-like scenario to motivate coordination. Next, we describe a modeling framework for MA control programs that is necessary for establishing coordination between agents. Next, we define an event propagation algorithm used to guarantee that event observations match individual agent STNUs. We finish by presenting experimental analysis of our event propagation algorithms.

6.1 Motivating Scenario from EVAs

We envision a scenario with an astronaut and a robot coordinating on the lunar surface. The astronaut is performing scientific exploration while the robot performs remote construction tasks. The concept of operations allows for the astronaut to use a rover to traverse away from the robot in search of promising scientific samples. Due to the position of surface relays and general uncertainty in lunar topology, there is an uncertain time delay between agents.

Bandwidth between Mission Control on Earth and the Moon is limited. There are low and high bandwidth communications available to both agents. Low bandwidth is responsible for transmitting critical data (e.g. suit telemetry), while high bandwidth communications are reserved for purposes such as video calls and large dumps of scientific data. It is not possible for both the astronaut and the robot to use high bandwidth communications simultaneously. Thus, there is a need for the agents to coordinate such that they make effective use of high bandwidth communications without stepping on each others toes, so to speak.

We hone in on a point in an EVA where there is substantial time delay between the astronaut and robot. The astronaut has set out far from the robot in search of scientifically interesting rock samples. Meanwhile, the robot is preparing to perform a drilling operation. The astronaut's sample collection work involves spectroscopy and video imagery, which is being sent to Mission Control using the high bandwidth connection. It will take between 15 and 30 minutes to downlink all the data. As soon as sample collection is over, the robot can use the high bandwidth connection to perform a drilling operation.

We say that the astronaut "owns," or is responsible for sharing observations of, the start and end of the experiment, while the robot similarly owns the drilling operation.

6.2 Multi-Agent Control Programs

This thesis introduces challenges in writing control programs for multiple agents who need to coordinate. We do not claim to solve all aspects of coordination, rather we present a framework for simple scenarios with the key feature being that agents need to agree on the *order* of a subset of events. We start by presenting an example of inter-agent temporal constraints, followed by defining a modeling framework for guaranteeing that agents agree about the order of events in their respective partial histories. To the best of our knowledge, we are unaware of any other MA framework for coordinating the order of event histories.

Consider two agents, `agent1` and `agent2`, that are scheduling STNUs S_1 and S_2 respectively. S_1 and S_2 share a subset of semantically similar episodes, e_1 and e_2. `agent1` "owns" e_1, meaning it is responsible for scheduling the free event e_1-start and observing the contingent event e_1-end, while `agent2` owns e_2.

A simplified MA view of the constraints is as follows.

$$e_1\text{-start} \xRightarrow{[15,30]} e_1\text{-end} \xrightarrow{[0,\infty]} e_2\text{-start} \xRightarrow{[22,26]} e_2\text{-end}$$

From `agent1`'s perspective, S_1 models the following constraints. We add a `noop` start event, Z, to simplify coordination.

$$Z \xrightarrow{[0,0]} e_1\text{-start} \xRightarrow{[15,30]} e_1\text{-end} \xRightarrow{[0,\infty]} e_2\text{-start} \xRightarrow{[22,26]} e_2\text{-end}$$

S_2 is then modeled as follows.

$$Z \xRightarrow{[0,0]} e_1\text{-start} \xRightarrow{[15,30]} e_1\text{-end} \xrightarrow{[0,\infty]} e_2\text{-start} \xRightarrow{[22,26]} e_2\text{-end}$$

For the sake of controllability of S_1 and S_2, we would simply add $[0,0]$ free constraints between consecutive contingent constraints.

We will walk through scheduling this scenario from the perspective of both agents. First, we describe their actions in the case that there is no communication delay, then we introduce the need for communication, and finally we add delay to communications. This scenario will motivate our analysis of the challenges that arise in MA control programs.

If both agents have perfect knowledge of the world (instantaneous knowledge of events), scheduling is trivial. `agent1` and `agent2` execute Z simultaneously. `agent1` schedules e_1-start and `agent2` instantaneously receives an observation of e_1-start. e_1-end arrives in $[15, 30]$ later, which again, both agents observe simultaneously. Now `agent2` is free to act. It schedules e_2-start, which `agent1` observes instantaneously. e_2-end arrives $[22, 26]$ later and is observed simultaneously by both agents.

Now, we enforce that `agent1` "owns" e_1 and is the only agent that can observe it directly. Likewise, `agent2` owns e_2. In order for an agent to learn about an episode they do not own, they must receive a communication from the agent who does. After `agent1` schedules e_1-start, it must send a message to `agent2`. `agent2` receives said message, which it interprets as an observation of e_1-start. If communications are instantaneous, the partial histories of both agents agree on the assignment of e_1-start. Later e_1-end is observed by `agent1`, who is then responsible for relaying a communication to `agent2` indicating that it is safe to assign e_1-end. `agent2` is now free to schedule e_2-start, and follows the same pattern of sending messages that events have been scheduled to `agent1`. After all events have been scheduled, the histories of `agent1` and `agent2` still agree on the times assigned to each event.

We now show that adding delay to the communications between agents forces us to add *synthetic* episodes to S_1 and S_2 to maintain event ownership. We now say that for S_2, $\bar{\gamma}(e_2\text{-end}) = [5, 15]$. In other words, `agent2` learns that `agent1` has finished e_2 some time in $[5, 15]$ after `agent1` has made the same assignment.

Once again, `agent1` schedules e_1-start and sends a message to `agent2`. Unlike before, their partial histories no longer match because `agent2` will assign e_1-start to some time in $[5, 15]$ after `agent1`.

Adding communication introduces a coordination challenge, even when said communication is instantaneous. Once again, we assume both agents execute Z simultaneously. When `agent1` schedules e_1-start, it must take the additional

We define coordination as sharing an understanding of when their peers have scheduled a subset of events. To maintain consistency, we need to maintain the order of events. i.e.

6.3 Event Propagation

At a high level, scheduled events propagate through a simple directed graph of connected executives. We put checks in place to ensure that cycles do not cause infinitely recursed event observations.

Definition 29. Communication Graph

A *communication graph* C is a tuple $\langle V, E \rangle$, where:

- V is a set of vertices representing peer executives,
- E is a set of directed edges between $v \in V$ representing the path of event observation propagation,
- Each edge $e_i \in E$ is a pair (o, t), where $o, t \in V$ represent the origin and termination of the edge respectively.

Loops, or self-edges, are not allowed, i.e. for any vertex $v_i \in V$, no single edge $e_i \in E$ may both originate and terminate at v_i.

For some executive $v_i \in V$ with outgoing edges in E, (v_i, v_j), $\cdots$, (v_i, v_k), any scheduled events that v_i assigns, whether free or contingent, are propagated to all peer executives v_j, $\cdots$, v_k. Likewise, all contingent events received from Nature are propagated to peers. Finally, any events v_i receives from other agents are also relayed to peers.

Definition 30. Event Propagation Messages

An *event propagation message* m is a tuple $\langle x, P \rangle$, where:

- x is a set of one or more events scheduled simultaneously,
- $P \subseteq V$ is a set of executives who have already received the message.

Recognize that Definition 30 is vague in defining x. Event propagation messages are passed between agents, and each agent has its own STNU. In some cases, x will be free events, in others x will be contingent events. The type of event makes no difference to the algorithm so we do not distinguish between them here.

Events that are received in m, $m[x]$, are handled the same as observations of contingent events during scheduling. Lemmas 8, 9, and 10 are applied as appropriate when the observation of $m[x]$ arrives.

For an edge $(v_i, v_j) \in E$, it is possible that v_j receives events that are not present in its STNU.

Because we have not defined a temporal decoupling-like algorithm wherein an STNU for multiple-agents is programmatically separated into individual STNUs (see the discussion of multi-agent STNUs [40] in Section 9.2), we are reliant on human planners to write STNUs for each agent by hand. As a result, there is no guarantee that x is meaningful to a given agent.

To be more specific, there is no guarantee that any event $x_i \in x$ in the event propagation message has an equivalent event in X_c of the STNU being executed by any receiving agent $v_j \in V$. If agent v_j cannot find x_i in their X_c, then x_i can be ignored. As will be discussed in Algorithm 7, we represent x using a type that can be compared for equivalence with the events in an agent's STNUs, e.g. a list of strings.

We use P to avoid cycles in event propagation. As will be shown in Algorithm 7, agent v_i will avoid propagating x to any agents in P. Agent v_i will also grow P when it relays m to other agents by appending to P itself and all outgoing agents $v_j, \cdots, v_k$.

Timing information, e.g. timestamps, is explicitly excluded from m. Dynamic scheduling and the variable-delay STNU and event observation, `obs`, formalisms do not account for timestamps. Instead, we expect that passing messages for event propagation between executives takes an amount of time in the domain $\mathbb{R}^+$. Thus, when v_j expects to receives an event, $x_i \in x$, from v_i, the time delay can be naturally modeled in the variable-delay function, $\bar{\gamma}(x_i)$, in the STNU that v_j will execute.

If event propagation messages were to include accurate timestamps, we would need to modify the way events are recorded during scheduling, impacting scheduling Lemmas 8, 9, and 10. Scheduling events in the past could also impact controllability. For these reasons, we avoid the inclusion of timestamps in event propagation messages.

By Definition 30, events received from other agents are no different than events received from Nature, and no special considerations are required for scheduling.

We now walk through the process of passing messages between agents as shown in Algorithm 7. We use the same *Event Propagation* algorithm in three cases:

1. When an agent v_i schedules free events x,
2. When v_i receives an observation from Nature of contingent events x,

3. When v_i receives an incoming message m_i with contingent events $m_i[x]$ from another agent in V.

Let `peers` be a mutable set initialized to the terminal vertices for all $e \in E$ originating at v_i.

In the first case, agent v_i fulfills its responsibilities as defined in C by broadcasting x to its `peers`, who will receive x as exogenous contingent events. The outgoing message m_o that will be passed to `peers` will include enough information such that no agent should receive a given x more than once. To do so, we let P be a set of all agents that will have observed x when m_o is received by `peers`, $P = \{v_i, p \ \forall \ p \in \texttt{peers}\}$. We finalize $m_o = \langle x, P \rangle$, which we simultaneously transmit to each p in `peers`. Transmission is a "fire and forget" operation, where v_i does not wait for acknowledgment from any p that m_o was received.

The second case plays out the same as the first, the only difference being that x is itself observed from Nature. Once again, we let P be a list of v_i and all `peers`, and then transmit m_o simultaneously to all `peers`.

The third case is a relay operation. Agent v_i is responsible for propagating events $m_i[x]$ that it has just observed, but we want to avoid sending the events to `peers` who have already observed them. We remove those agents from `peers` accordingly with a set difference operation: $\texttt{peers} = \texttt{peers} - m_i[P]$. Likewise, we grow the list of agents who have received x, which is now $P = P \cup \texttt{peers}$. Agent v_i composes a new $m_o = \langle m_i[x], P \rangle$ and transmits it to `peers`.

Ideally, the Event Propagation algorithm should run on a separate thread from the main scheduling loop, else we run the risk of incurring unnecessary delays in observing and dispatching events.

```
Input: Incoming message $m_i$; Scheduled events $x$; Self $v_i \in V$; Set of outgoing peers $\subset V$
  Event Propagation:
1     peers $\leftarrow$ peers $- m_i[P]$;
2     $P \leftarrow \{m_i[P]\} \cup \{v_i\} \cup$ peers;
3     $x \leftarrow x$ or $m_i[x]$;
4     $m_o \leftarrow \langle x, P \rangle$;
5     for each $p$ in peers do
6         Perform a non-blocking transmission of $m_o$ to $p$;
7     end
```

Algorithm 7: An event propagation algorithm that avoids recursive message passing.

The complexity of Algorithm 7 is trivially $O(N)$, where N is the number of executives in $V - 1$. The limiting factor to the performance of Event Propagation will be the time it takes to transmit messages between agents, which, to reiterate, should be modeled in the delay functions for any inter-agent temporal constraints.

6.4 Experimental Analysis

We performed two analyses of the Event Propagation algorithm. The first was a hardware demonstration performed on a Barrett WAM manipulator in the MERS lab. The second is a massively multi-agent simulation. Both will be described below.

6.4.1 Hardware Demonstration

We built a demonstration of the motivating scenario of this thesis in our lab using a Barrett WAM manipulator representing the robot, and Valve Steam Deck representing the astronaut.

Each agent has their own RMPL control program, which we include in Listings 7 and 8. Note that each control program is nearly identical. The control programs related to the high bandwidth handoff, `human-downlink-science`, `sync`, and `robot-drilling`, are nearly identical, differing in observation delay and whether the `sync` event is controllable. Adding observation delay reflects uncertain communication between the agents, while the `sync` activity serves to keep the STNUs of the agents aligned.

6.4.2 Massively Multi-Agent Simulation

```
;;;; -*- Mode: common-lisp; -*-

(defpackage #:scenario1)

(in-package #:scenario1)

(define-control-program human-downlink-science ()
  (declare (primitive)
           (duration (simple :lower-bound 15 :upper-bound 30)
                     :contingent t)))

(define-control-program sync ()
  (declare (primitive)
           (duration (simple :lower-bound 5 :upper-bound 15
                             :min-observation-delay 0
                             :max-observation-delay 1)
                     :contingent t)))

(define-control-program robot-drilling ()
  (declare (primitive)
           (duration (simple :lower-bound 22 :upper-bound 26
                             :min-observation-delay 0
                             :max-observation-delay 2)
                     :contingent t)))

(define-control-program human-closeout ()
  (declare (primitive)
           (duration (simple :lower-bound 10 :upper-bound 30))))

(define-control-program main ()
  (with-temporal-constraint (simple-temporal :upper-bound 480)
    (sequence (:slack nil)
      (human-downlink-science)
      (sync)
      (robot-drilling)
      (human-closeout))))
```

Listing 7: The control program the astronaut uses while collecting and downlinking scientific data.

```lisp
;;;; -*- Mode: common-lisp; -*-

(defpackage #:scenario1)

(in-package #:scenario1)

(define-control-program human-downlink-science ()
  (declare (primitive)
           (duration (simple :lower-bound 15 :upper-bound 30
                             :min-observation-delay 5
                             :max-observation-delay 15)
                     :contingent t)))

(define-control-program sync ()
  (declare (primitive)
           (duration (simple :lower-bound 5 :upper-bound 15))))

(define-control-program robot-drilling ()
  (declare (primitive)
           (duration (simple :lower-bound 22 :upper-bound 26
                             :min-observation-delay 0
                             :max-observation-delay 1)
                     :contingent t)))

(define-control-program robot-poweroff ()
  (declare (primitive)
           (duration (simple :lower-bound 10 :upper-bound 30))))


(define-control-program main ()
  (with-temporal-constraint (simple-temporal :upper-bound 480)
    (sequence (:slack nil)
      (human-downlink-science)
      (sync)
      (robot-drilling)
      (robot-poweroff))))
```

Listing 8: The control program the robot uses to decide when to act with respect to learning the astronaut has finished collecting scientific data.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

An Architecture for Robust Scheduling and Dispatching

7.1 Clock-Synchronized Dispatching and Monitoring

In this section, we propose an architecture for organizing and executing long-running tasks in an autonomous executive.

As described in Section 5.5.1, the dispatching algorithm should loop uninterrupted until all executable events have been scheduled. There are, however, other tasks that an executive needs to perform during execution, ranging from mission critical monitoring of contingent events, to more mundane logging of debug and info-level messages to a human operator. Naturally, modern CPUs and programming languages provide multi-threaded or multi-process applications to handle concurrent, long-running tasks. While parallel computing is a perfectly viable option for building an executive, we found that a clock-synchronized approach for long-running tasks more naturally aligns with our autonomy reasoning abilities, and gives us more flexibility in the capabilities of Kirk.

7.1.1 Challenges of Parallel Computing for Long-Running Tasks

Consider the high-level responsibilities of Kirk, or any goal-based task and motion planner, during execution. At a minimum, one must,

1. monitor progress towards its goals,
2. decide when to act,
3. send commands to hardware, and
4. communicate with a human operator.

If we were to structure an executive based on these responsibilities, we may naturally start allocating each responsibility to its own thread. Monitoring progress might be a loop that constantly checks sensors or sensor-fused data. Deciding when to act would be the online dynamic dispatching algorithm described in Section 5.5.1. We would not want hardware commands to block dispatching, so we once again create a new thread. Meanwhile, the last thread would collect information about the running executive, e.g. clock times, execution progress, and explanations of the actions taken, in order to pass it along to a human operator.

While missions may take hours or days, simulations of said missions should take seconds. There are many reasons to simulate. Before deploying an executive on expensive hardware in an extreme environment, an operator may rightfully want to observe the behavior of the executive under a wide range of potential mission conditions. Or we may want to try a new long-running task or motion planning algorithm, but not want to wait hours to see how it performs. Or we may want to compare and benchmark different options for task and motion planning. For these reasons and more, we need the ability to reliably run an executive at faster than real-time speeds with confidence that its behavior in simulation is reflective of its performance in the real-world. Thus, during simulation, we add another responsibility:

5. update the clock at faster than real-time speeds.

Parallel computing starts to break down when we want to simulate a mission due to synchronization challenges. For instance, the event monitoring thread may be waiting to simulate a contingent event observation at time t. We would need to ensure that the operation in the monitoring thread that checks the clock time runs at least once while the clock is at t. If the clock is running too quickly, t could be missed and the simulated contingent event is never observed. We could make the check more robust by either slowing the clock down or adding tolerance to the time check, e.g. checking that the clock is within a range near t instead of exactly t, but the underlying problem remains.

There may also be temporal dependencies between threads. Algorithm 4 assumes that the time does not change while it is running. During real-time operations, it is effectively the case that time is not passing, but we lose the guarantee in simulation. It could be that the dispatcher believes it to be time t when it dispatches an event, but by the time the event is scheduled or the driver receives a command, the clock is now at time $t' \gg t$, impacting the controllability of the STNU.

None of the problems introduced by parallel computing are insurmountable, but they add code and complexity to an already complex system. This creates two fundamental concerns that caused us to rewrite the online architecture of Kirk. First is that adding code *post-hoc* to decision-making algorithms to address special cases, like faster than real-time clocks, means that the code we simulate and the code we run during missions fundamentally differ. This differentiation reduces our confidence that testing and verifying the executive in simulation is indicative of its performance in real-time.

Second is that additional complexity increases the surface area for failures and anomalous behavior. Instead, we propose a clock-based synchronization approach that is no less efficient during real-time operations while requiring no change to our decision-making algorithms to enable faster than real-time simulation.

7.1.2 Clock-Based Synchronization

Clock-based synchronization hands control of long-running tasks to a shared clock. Rather than allowing each thread to run independently, the clock takes responsibility for executing tasks at an appropriate interval. We call each interval a *tick*.

At each tick, every *task* (Definition 31) is run. Tasks are lambda functions that communicate to the clock by their return values. If a task returns `true`, it is interpreted as a signal that the clock may continue ticking. Returning `false` tells the clock that ticking should stop. So long as all tasks want the clock to continue ticking, it should. If any task returns `false`, ticking should stop.

Definition 31. Task

A *task* is a lambda function that takes nothing as input and returns a Boolean. The return value indicates whether the task wants the clock to continue ticking.

Before the clock starts to run, the executive adds tasks to a queue. The tasks will be run consecutively in the order they appear.

Implemented in a real-time clock, the ticking algorithm should run as fast as possible. We do so by implementing Algorithm 8, which recursively calls itself until it receives a signal from a task that it should stop.

```
Input: Boolean $tickp$; Task queue $queue$;
  clockTick:
1   if $tickp \wedge (queue[length] > 0)$ then
2       for $task$ $in$ $queue$ do
3           $tickp \leftarrow (\texttt{task}() \wedge tickp)$;
4       end
5       clockTick($tickp, queue$)
6   endif
```

Algorithm 8: A recursive algorithm for clock-synchronized tasks in real-time.

We simulate a faster than real-time clock with Algorithm 9. The key difference is the additional clock advancement operation added before recursing. Unlike a synchronized thread approach, we are guaranteed an order of operations between tasks and the clock. We know tasks will run in the order they appear in *queue*, the clock will advance, then the tasks will run again.

If Algorithms 8 and 9 are implemented as class methods, *tickp* naturally lends itself to be a class property. As such, other interfaces can be built for controlling *tickp*, ultimately leading to a robust clock that can be arbitrarily started and stopped as required. In the implementation of Kirk for this

```
Input: Boolean $tickp$; Task queue $queue$; Sleep duration $d$;
  clockTick:
1   if $tickp \wedge (queue[length] > 0)$ then
2       for $task$ $in$ $queue$ do
3           $tickp \leftarrow (\texttt{task}() \wedge tickp)$;
4       end
5       Advance clock by $d$;
6       clockTick($tickp, queue, d$)
7   endif
```

Algorithm 9: A recursive algorithm for clock-synchronized tasks in faster than real-time.

thesis, this paradigm enables us to perform time consuming offline planning upon its initialization, including tasks like running the pipeline to go from RMPL to a distance graph, before starting the clock when we are ready to start scheduling.

7.2 Single-Responsibility Principle

Not an exact law but something we followed with scheduler > dispatcher > driver layers

Chapter 8

Evaluation

This is a chapter on evaluating all this stuff.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 9

Discussion and Future Work

9.1 Optimistic Rescheduling

What if we looked for conflicts? Could we possibly search for a time to dispatch early in the buffered execution space?

Smarter rewriting STNU. Could we update the existing d-graph directly and check it for SRNCs? maybe?

9.2 Coordination

We were focused on addressing the multi-agent (MA) online scheduling problem. Before scheduling, we must contend with planning, e.g. building variable-delay STNUs for each agent. We considered extending the two existing planning approaches described below to model variable observation delay between agents. We ultimately decided neither were fit for the motivating scenarios of this thesis. Instead, we used a manual planning approach more akin to the ISS EVA planning process.

The first planning approach we considered was to model the system as a Multi-Agent STNU (MASTNU) [40]. MASTNUs allow modelers to describe temporal constraints between multiple agents, then check the overall dynamic controllability of the system. To check the controllability of a MASTNU, the first step is to perform temporal decoupling with the goal of producing individual dynamically controllable STNUs for each agent that can be dynamically scheduled per usual. While superficially promising, there is a considerable drawback to this approach, namely that temporal decoupling is sound but not complete, i.e. temporal decoupling may report failure even when the MASTNU is dynamically controllable. This limits the utility of MASTNUs as a planning tool.

The other approach to this problem we are aware of is Stedl's Hierarchical Reformulation (HR) algorithm [41]. HR begins with a MA temporal plan network (TPN), which is similar to a MASTNU (though HR pre-dates MASTNUs). Stedl's key insight is to avoid inter-agent communication altogether by reformulating constraints between groups of agents such that they are strongly controllable. As such, no communication between agents is required. A centralized dispatcher is then responsible for then handing events to agents. We also assume that there is no central authority, making HR a poor fit for our problem domain.

Both MASTNUs and HR assume communications between agents are either instantaneous or impossible, i.e. with an infinite delay. As we will see in Section 4.3, our formalism for variable observation delay allows a *spectrum* of communication delay. While we felt it was possible to shoehorn uncertain observation delay into MASTNUs or HR, we felt both were a poor choice because of their pre-existing expectations with respect to communication. In combination with our focus on online scheduling, we decided to forgo extending either formalism to account for observation delay. Instead our planning process simply consists of manually writing variable-delay STNUs with intra-agent and inter-agent temporal constraints by hand.

We believe it may be possible for MASTNUs or HR to be expanded to include variable observation delay, though we leave that problem for future research.

We considered framing our approach to inter-agent communication as a distributed consensus problem because we believed we needed a means for disparate agents to agree on the state of the world. Existing distributed consensus algorithms like Paxos [42] or Raft [43] would then be integrated into the communication layer of Kirk and take responsibility for ensuring that agents agree on which events have been scheduled.

Ultimately the drawbacks of a distributed consensus approach outweighed the benefits. Chiefly, both Paxos and Raft assume that communications are either instantaneous and freely available or that agents have gone dark (i.e. can no longer communicate). This communication model is incongruous with the explicitly modeled communications of the VDC formalism. Furthermore, the VDC formalism allows us to model that agents never receive communications, negating the requirement for distributed consensus.

Appendix A

Comparison of Variable-Delay STNUs to Partially Observable STNUs

The delay scheduler is flexible in that so long as it receives a variable-delay STNU, it is capable of scheduling. Human modelers have flexibility in how they represent temporal constraints in that there are many flavors of STNUs, each with their own advantages and disadvantages. Earlier, we presented RMPL as a modeling language that is compiled to variable-delay STNUs. There are other choices for modeling frameworks. Here, we present a comparison of variable-delay STNUs to POSTNUs [32], a flavor that is similar in many respects. This section presents a comparison of variable-delay STNUs and POSTNUs, including transformations that allow some classes of POSTNUs to be represented as variable-delay STNUs.

One of the strengths of the variable-delay controllability model is its ability to generalize the concepts of strong and dynamic controllability. This technique was first seen in greater depth in the context of POSTNUs. In an STNU, all contingent events are either instantaneously observable under a dynamic controllability model or entirely unobservable under a strong controllability model. In POSTNUs, contingent events can be marked observable and unobservable. To say that a POSTNU is dynamically controllable equates to asserting that it is possible to construct a schedule during execution that respects all constraints if the scheduler only receives information about observable contingent events.

While, superficially, POSTNUs and delay STNUs appear to model distinct problems in temporal reasoning, all delay STNUs can be accurately represented as POSTNUs. While the converse is not true, there is a subset of POSTNUs that delay STNUs are able to model. It is advantageous to translate POSTNUs to delay STNUs when possible because we are guaranteed to finish controllability checks for delay STNUs in polynomial time, while evaluating the controllability of POSTNUs in general is harder [32]. Furthermore, the sub-class of POSTNUs that can be checked efficiently and accurately, those without chained contingent links [44], are members of the subset of POSTNUs that can be expressed directly as STNUs with fixed-delay, and likewise variable-delay, functions , [31, p. 59]. The converse is also true - we may emulate STNUs with variable-delay functions as POSTNUs without chained contingent links. Below, we elaborate on translations from delay STNUs to POSTNUs, before describing how we can express POSTNUs without chained contingent links as STNUs with fixed-delay functions. Note that this is not a comprehensive list of transformations between delay STNUs and POSTNUs - our aim is to describe the minimum set of transformations required to model POSTNUs without chained contingent links as delay STNUs. For the following discussion, let S be a delay STNU and P be an equivalent POSTNU.

We present these transformations to demonstrate that the delay scheduler is capable of scheduling POSTNUs without chained contingencies. So long as it receives a variable-delay STNU, the fact that POSTNUs can be scheduled allows modelers additional flexibility in their means of representing the problem domain.

We start with an S that consists of the links $A \Rightarrow B$ and $B \rightarrow D$ with delay function $\bar{\gamma}(B)$. See Figure A for an example translation between a fixed-delay STNU and a POSTNU.

Lemma 14. *For contingent link $A \Rightarrow B$ in S with observation delay $\bar{\gamma}(B) = [l, u]$, where $0 \leq l \leq u < \infty$, and outgoing requirement link $B \rightarrow D$, we may emulate observation delay in P by copying $A \Rightarrow B$ and $B \rightarrow D$ to P, enforcing that B is unobservable, and adding a new observable contingent event, B' with $B \xRightarrow{[l,u]} B'$.*

Proof. In both S and P, we do not observe B directly, yet we define an outgoing requirement link, $B \rightarrow D$, that depends entirely on the resolution of B. The only information available to reason about the assignment of B comes in the form of an indirect observation, B' or $\bar{\gamma}(B)$, received after a delay in $\mathbb{R}^{\geq 0}$. If P was not equivalent to S, we would be able to learn the assignment of B without waiting $\bar{\gamma}(B)$ time units after its true assignment. Thus, because we must wait $B' - B \in [l, u]$ time units to learn the assignment of B, and $B \rightarrow D$ has equivalent constraints between S and P, P must model the same semantics as S. □

Lemma 15. *For a contingent event B with variable-delay function $\bar{\gamma}(B) = [0, 0]$ in S, we may emulate the same constraints with an observable contingent event, B in P.*

Proof. The variable-delay function enforces instantaneous observation. By the definition of observable contingent events in the POSTNU model, we will observe B instantaneously. □

A POSTNU with a chained contingency is defined as follows. Consider a chain of contingent events, $A \Rightarrow B$ and $B \Rightarrow C$. If B has one or more outgoing links to free or contingent events other than C, it is a chained contingency. Lemma 14 results in a POSTNU without chained contingencies, hence variable-delay STNUs fall into the subclass of POSTNUs that can be checked efficiently [44].

In the other direction, to transform a POSTNU without chained contingencies into an STNU with variable-delay functions, we need to address three cases of contingent constraints: (1) unobservable contingent events not immediately followed by other contingent constraints, (2) unobservable contingent events immediately followed by other contingent constraints, and (3) observable contingent constraints.

Lemma 16. *For an unobservable event B in P, with no outgoing contingent links, we can emulate it in S with a contingent event B and upper bound of its variable-delay function set to $\bar{\gamma}^{+}(B) = \infty$.*

Proof. We will not observe B nor will outgoing contingent links provide information about B. As such we define $\bar{\gamma}^{+}(B) = \infty$ in S.[1] From a controllability standpoint for both S and P, we know the *a priori* bounds of B but will not learn its true assignment. □

Lemma 17. *For an unobservable contingent link $A \xRightarrow{[m,n]} B$ in P, with a single outgoing link, $B \xRightarrow{[w,z]} C$, we replace the two constraints from P in S with a concatenated constraint, $A \xRightarrow{[m+w,n+z]} C$.*

Proof. The only information we may receive is the observation of C. Given there are no other outgoing links from B, folding B into the successive contingent constraint can not affect the semantics of the network. The bounds of the new link, $A \xRightarrow{[m+w,n+z]} C$ is the result of summing the intervals of $A \xRightarrow{[m,n]} B$ and $B \xRightarrow{[w,z]} C$: $[m,n] + [w,z] = [m+w, n+z]$. □

Note that we did not specify whether C is observable in P. After applying Lemma 17, we then apply either Lemma 15 or 16 to C.

Lemma 18. *For an observable contingent event $A \xRightarrow{[m,n]} B$ in P, with a single outgoing link to an observable contingent event, C, $B \xRightarrow{[w,z]} C$, we create three constraints in S: $A \xRightarrow{[m,n]} B$, $B \xrightarrow{[0,0]} B'$, and $B' \xRightarrow{[w,z]} C$ where $\bar{\gamma}(B) = [0,0]$.*

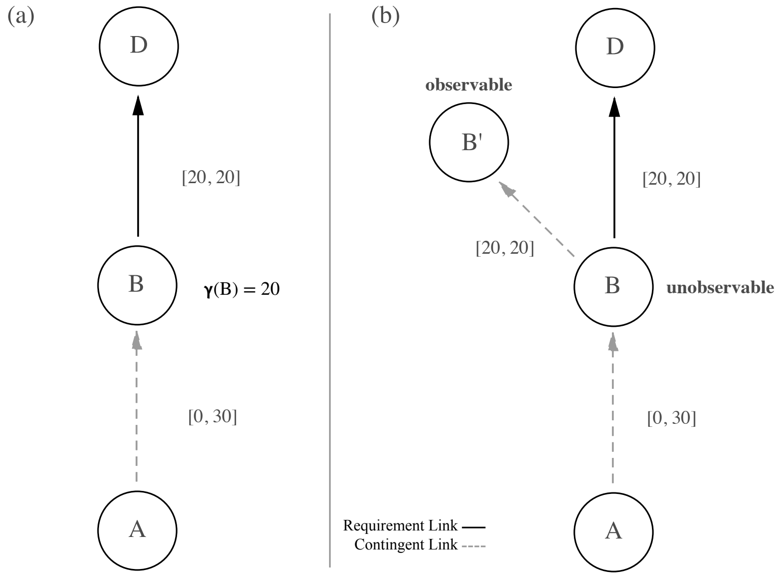
Proof. Given C is observable in P, a simulated free event, B' in S, can be scheduled simultaneously with B. Any contingent constraints following B now start at an executable event and are thus valid constraints in S. B is observable, so we need no observation delay according to Lemma 15. □

Thus, delay STNUs are sufficiently capable of expressing all POSTNUs that can efficiently be checked for controllability using today's tractable POSTNU algorithms.

It is not clear if controllability can be checked more efficiently across a greater subset of POSTNUs beyond those without chained contingencies. However, it is worth highlighting that variable-delay

[1]Note: ,p. [31, p. 60] erroneously claims that we should define $\gamma(B) = 0$ for unobservable events.

Figure A-1: (a) An STNU with a contingent constraint that has a certain delay. (b) One possible way of rewriting the STNU as an equivalent POSTNU. This particular POSTNU exhibits a chained contingency, as B is a contingent event that starts a contingent constraint and is connected to B' via a contingent constraint.



controllability can be leveraged to construct improved algorithms with respect to scheduling and controllability of POSTNUs. The model for observation delay proposed by variable-delay controllability can be expressed exactly as a POSTNU with a "single-headed" chained contingency[2] as shown in Figure Ab; the main difference is that we represent the contingent link between B and B' with our variable-delay function $\bar{\gamma}(B)$. Hence, the algorithm we present for variable-delay controllability can be used to both solve POSTNUs without chained contingencies, as described above, as well as those POSTNUs with single-headed chained contingencies. Approaches inspired by variable-delay controllability have been used to further expand POSTNU dynamic controllability checking in more expressive chained instances [45]. We hope that insights from variable-delay controllability will continue to expand the subset of POSTNUs that can be controllability checked, and as such we advocate for continued development of the theory of variable-delay controllability as a relevant framework for modelers.

[2]To borrow the term "single-headed" from [45].

Bibliography

[1] N. Bhargava, C. Muise, and B. C. Williams, Variable-delay controllability, in *IJCAI International Joint Conference on Artificial Intelligence*, 2018, vol. 2018-July, pp. 46604666. doi: 10.24963/ijcai.2018/648.

[2] M. J. Miller, Decision Support System Development For Human Extravehicular Activity, Georgia Institute of Technology, 2017.

[3] D. Wang and B. C. Williams, TBurton: A divide and conquer temporal planner, in *Proceedings of the National Conference on Artificial Intelligence*, 2015, vol. 5, pp. 34093417.

[4] J. W. McBarron, Past, present, and future: The U.S. EVA Program, *Acta astronautica*, vol. 32, no. 1, pp. 514, 1994, doi: 10.1016/0094-5765(94)90143-0.

[5] C. P. Sonnett, REPORT of the AD HOC WORKING GROUP ON APOLLO EXPERIMENTS AND TRAINING on the SCIENTIFIC ASPECTS OF THE APOLLO PROGRAM, NASA, 1963.

[6] J. M. Hurtado, K. Young, J. E. Bleacher, W. B. Garry, and J. W. Rice, Field geologic observation and sample collection strategies for planetary surface exploration: Insights from the 2010 Desert RATS geologist crewmembers, *Acta astronautica*, vol. 90, no. 2, pp. 344355, 2013, doi: 10.1016/j.actaastro.2011.10.015.

[7] K. Young and T. Graff, Planetary Science Context for EVA, in *NASA EVA Exploration Workshop*, 2020, p. 40.

[8] A. Kanelakos, Artemis EVA Flight Operations - Preparing for Lunar EVA Training & Execution, in *NASA EVA Exploration Workshop*, 2020, p. 47.

[9] C. Campbell, Advanced EMU Portable Life Support System (PLSS) and Shuttle/ISS EMU Schematics, a Comparison, *42nd international conference on environmental systems*, pp. 118, 2012, doi: 10.2514/6.2012-3411.

[10] E. S. Patterson and D. D. Woods, Shift changes, updates, and the on-call architecture in space shuttle mission control, *Computer supported cooperative work*, vol. 10, no. 3-4, p. 27, 2001, doi: 10.1023/A:1012705926828.

[11] S. J. Payler *et al.*, Developing Intra-EVA Science Support Team Practices for a Human Mission to Mars, *Astrobiology*, vol. 19, no. 3, pp. 387400, 2019, doi: 10.1089/ast.2018.1846.

[12] D. A. Coan, Exploration EVA System Concept of Operations Summary for Artemis Phase 1 Lunar Surface Mission, NASA, Houston, TX, 2020.

[13] M. A. Seibert, D. S. Lim, M. J. Miller, D. Santiago-Materese, and M. T. Downs, Developing Future Deep-Space Telecommunication Architectures: A Historical Look at the Benefits of

Analog Research on the Development of Solar System Internetworking for Future Human Spaceflight, *Astrobiology*, vol. 19, no. 3, pp. 462477, Mar. 2019, doi: 10.1089/ast.2018.1915.

[14] M. J. Miller and K. M. Feigh, *Addressing the envisioned world problem: A case study in human spaceflight operations*, vol. 5. Cambridge University Press, 2019. doi: 10.1017/dsj.2019.2.

[15] M. J. Miller, K. M. McGuire, and K. M. Feigh, Information flow model of human extravehicular activity operations, *Ieee aerospace conference proceedings*, vol. 2015-June, 2015, doi: 10.1109/AERO.2015.7118942.

[16] M. J. Miller, K. M. McGuire, and K. M. Feigh, Decision Support System Requirements Definition for Human Extravehicular Activity Based on Cognitive Work Analysis, *Journal of cognitive engineering and decision making*, vol. 11, no. 2, pp. 136165, 2017, doi: 10.1177/1555343416672112.

NO_ITEM_DATA:Sehlke2019
[18] B. C. Williams, M. D. Ingham, S. H. Chung, and P. H. Elliott, Model-based programming of intelligent embedded systems and robotic space explorers, *Proceedings of the ieee*, vol. 91, no. 1, pp. 212236, 2003, doi: 10.1109/JPROC.2002.805828.

[19] B. C. Williams and R. J. Ragno, Conflict-directed A* and its role in model-based embedded systems, *Discrete applied mathematics*, vol. 155, no. 12, pp. 15621595, 2007, doi: 10.1016/j.dam.2005.10.022.

[20] R. E. Fikes and N. J. Nilsson, Strips: A new approach to the application of theorem proving to problem solving, *Artificial intelligence*, vol. 2, no. 3-4, pp. 189208, 1971, doi: 10.1016/0004-3702(71)90010-5.

[21] R. Dechter, I. Meiri, and J. Pearl, Temporal constraint networks, *Artificial intelligence*, vol. 49, no. 1-3, pp. 6195, 1991, doi: 10.1016/0004-3702(91)90006-6.

[22] E. Fernández González, Generative Multi-Robot Task and Motion Planning Over Long Horizons, Massachusetts Institute of Technology, 2018.

[23] J. Chen, B. C. Williams, and C. Fan, Optimal mixed discrete-continuous planning for linear hybrid systems, in *HSCC 2021 - Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control (part of CPS-IoT Week)*, 2021, vol. 1. doi: 10.1145/3447928.3456654.

[24] T. Vidal and H. Fargier, Handling Contingency in Temporal Constraint Networks: From Consistency to Controllabilities, *Journal of experimental and theoretical artificial intelligence*, vol. 11, no. 1, pp. 2345, 1999, doi: 10.1080/095281399146607.

[25] P. H. Morris, N. Muscettola, and T. Vidal, Dynamic Control Of Plans With Temporal Uncertainty, 2001.

[26] P. Morris and N. Muscettola, Temporal dynamic controllability revisited, *Proceedings of the national conference on artificial intelligence*, vol. 3, pp. 11931198, 2005.

[27] P. Morris, A structural characterization of temporal dynamic controllability, *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*, vol. 4204 LNCS, pp. 375389, 2006, doi: 10.1007/11889205
$_2$8.

[28] P. Morris, Dynamic controllability and dispatchability relationships, *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*, vol. 8451 LNCS, no. Dc, pp. 464479, 2014, doi: 10.1007/978-3-319-07046-9 $_3$3.

[29] A. Cimatti, A. Micheli, and M. Roveri, Solving temporal problems with uncertainty using SMT: Strong Controllability, *Constraints*, vol. 20, no. 1, pp. 129, 2012, doi: 10.1007/s10601-014-9167-5.

[30] M. Ingham, R. Ragno, A. Wehowsky, and B. Williams, The Reactive Model-based Programming Language, MIT Space Systems and Artificial Intelligence Laboratories, 2002.

[31] N. Bhargava, Multi-Agent Coordination under Limited Communication, Massachusetts Institute of Technology, 2020.

[32] M. D. Moffitt, On the partial observability of temporal uncertainty, *Proceedings of the national conference on artificial intelligence*, vol. 2, pp. 10311037, 2007.

[33] G. Kiczales, J. Des Rivières, and D. G. Bobrow, *The art of the metaobject protocol.* Cambridge, Mass: MIT Press, 1991.

[34] M. Fox and D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, *Journal of artificial intelligence research*, vol. 20, pp. 61124, 2003, doi: 10.1613/jair.1129.

[35] N. Bhargava, C. Muise, T. Vaquero, and B. Williams, Delay Controllability : Multi-Agent Coordination under Communication Delay, Massachusetts Institute of Technology, Cambridge, MA, 2018.

[36] L. Hunsberger, Efficient execution of dynamically controllable simple temporal networks with uncertainty, *Acta informatica*, vol. 53, no. 2, pp. 89147, 2016, doi: 10.1007/s00236-015-0227-0.

[37] N. Bhargava, C. Muise, T. Vaquero, and B. Williams, Managing communication costs under temporal uncertainty, in *IJCAI International Joint Conference on Artificial Intelligence*, 2018, vol. 2018-July, pp. 8490. doi: 10.24963/ijcai.2018/12.

[38] L. Hunsberger, A faster execution algorithm for dynamically controllable STNUs, *Proceedings of the 20th international symposium on temporal representation and reasoning*, pp. 2633, 2013, doi: 10.1109/TIME.2013.13.

[39] L. Hunsberger, Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies, *Time 2009 - 16th international symposium on temporal representation and reasoning*, pp. 155162, 2009, doi: 10.1109/TIME.2009.25.

[40] G. Casanova, C. Pralet, C. Lesire, and T. Vidal, Solving dynamic controllability problem of multi-agent plans with uncertainty using mixed integer linear programming, *Frontiers in artificial intelligence and applications*, vol. 285, pp. 930938, 2016, doi: 10.3233/978-1-61499-672-9-930.

[41] J. L. Stedl, Managing Temporal Uncertainty Under Limited Communication: A Formal Model of Tight and Loose Team Coordination, Massachusetts Institute of Technology, 2004.

[42] L. Lamport *et al.*, The Part-Time Parliment, *Acm transactions on computer systems*, vol. 16, no. 2, pp. 373386, 1998, doi: 10.1145/568425.568433.

[43] D. Ongaro and J. Ousterhout, In search of an understandable consensus algorithm, *Proceedings of the 2014 usenix annual technical conference, usenix atc 2014*, pp. 305319, 2014.

[44] A. Bit-Monnot, M. Ghallab, and F. Ingrand, Which contingent events to observe for the dynamic controllability of a plan, *Ijcai international joint conference on artificial intelligence*, vol. IJCAI-16, no. July, pp. 30383044, 2016.

[45] P. H. Morris and A. Bit-monnot, Dynamic Controllability with Single and Multiple Indirect Observations, in *ICAPS 2019 - Proceedings of the 29th International Conference on Automated Planning and Scheduling*, 2019, p. 9.